

# ZMotion Basic 编程手册

Version 3.3.0

# 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](http://www.zmotion.com.cn)

# 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。

# 版权说明

本手册版权归深圳市正运动技术有限公司所有，未经正运动公司书面许可，任何人不得翻印、翻译和抄袭本手册中的任何内容。本手册中的信息资料仅供参考。由于改进设计和功能等原因，正运动技术有限公司保留对本资料的最终解释权！内容如有更改，恕不另行通知！如需新版资料，请联系我司相关人员。



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
1.1. 运动控制产品概述 .....	1
1.2. 运动控制产品优点 .....	1
1.3. 控制器主要功能描述 .....	2
1.4. 控制器应用场景 .....	2
1.5. 控制器接口 .....	3
1.6. 控制器的使用 .....	4
第二章 ZDevelop 软件编程 .....	7
2.1. 编程软件简介 .....	7
2.2. 新建工程 .....	7
2.3. 在线命令与输出 .....	11
2.4. 示波器的使用 .....	14
2.4.1. 示波器面板介绍 .....	14
2.4.2. 示波器数据导入导出 .....	15
2.4.3. 示波器采样方法 .....	16
2.4.4. 示波器使用注意事项 .....	16
2.4.5. 示波器使用例程 .....	17
2.5. 程序调试 .....	21
2.5.1. 进入程序调试 .....	21
2.5.2. 任务与监控窗口 .....	21
2.5.3. 调试工具栏的使用 .....	22
2.5.4. 断点调试 .....	22
2.6. 视图窗口 .....	23
第三章 Basic 编程基础 .....	25
3.1. 编程基础知识 .....	25
3.1.1. 程序 .....	25
3.1.2. 数据相关 .....	28
3.2. ZDevelop 的三种编程方式 .....	32
3.2.1. 混合编程 .....	32
3.2.2. PLC 与 BASIC 互相调用 .....	32
3.3. 寄存器 .....	33
3.3.1. TABLE .....	34
3.3.2. FLASH .....	35
3.3.3. VR .....	36
3.3.4. MODBUS .....	37
3.4. 多任务编程 .....	39
3.4.1. 多任务概念 .....	39
3.4.2. 多任务状态查看 .....	40
3.4.3. 多任务启动与停止 .....	41
3.4.4. 任务暂停与恢复 .....	43
3.4.5. Basic 和 PLC 任务相互调用 .....	44
3.4.6. 多任务示例 .....	45
3.5. 三种中断类型 .....	47
3.5.1. 掉电中断 .....	47
3.5.2. 外部中断 .....	48
3.5.3. 定时器中断 .....	48
3.6. 运动缓冲 .....	49
3.6.1. 运动缓冲概念 .....	49
3.6.2. 运动缓冲区 .....	49
3.6.3. 运动缓冲区堵塞 .....	51

3.6.4.	运动缓冲中输出 .....	53
第四章	通讯方式 .....	54
4.1.	串口通讯 .....	54
4.1.1.	串口类型 .....	54
4.1.2.	串口连接方法 .....	56
4.2.	网口通讯 .....	58
4.3.	CAN 总线通讯 .....	61
4.3.1.	CAN 接线 .....	61
4.4.	U 盘接口 .....	62
4.5.	EtherCAT 总线通讯 .....	64
4.5.1.	EtherCAT 总线初始化 .....	64
4.5.2.	EtherCAT 总线与驱动器通讯 .....	67
4.5.3.	EtherCAT 总线连接扩展模块 .....	68
4.6.	RTEX 总线通讯 .....	69
第五章	运动控制功能 .....	73
5.1.	常见运动模式 .....	73
5.1.1.	单轴点动 .....	73
5.1.2.	电子凸轮 .....	75
5.1.3.	电子齿轮 .....	77
5.1.4.	手轮 .....	77
5.2.	插补运动 .....	79
5.2.1.	插补概念 .....	79
5.2.2.	连续插补 .....	82
5.3.	前瞻预处理 .....	82
5.4.	原点回零 .....	84
5.5.	限位相关指令 .....	87
5.6.	位置锁存 .....	90
5.7.	硬件比较输出 .....	91
5.8.	精准输出 .....	92
5.9.	振镜控制系统 .....	92
5.9.1.	振镜说明 .....	92
5.9.2.	振镜应用流程 .....	96
5.10.	机械手 .....	101
5.10.1.	机械手相关概念 .....	101
5.10.2.	正解运动与逆解运动 .....	102
5.10.3.	机械手支持的功能 .....	103
5.10.4.	机械手应用举例 .....	103
5.11.	G 代码 .....	108
第六章	轴相关说明 .....	109
6.1.	轴的概念 .....	109
6.2.	轴号说明 .....	109
6.3.	轴状态 .....	110
6.4.	轴速度 .....	111
6.4.1.	速度曲线 .....	111
6.4.2.	SP 速度 .....	113
6.5.	轴映射 .....	115
6.6.	轴类型 .....	116
第七章	运动指令 .....	120
7.1.	单轴运动指令 .....	120
	ADDAX -- 运动叠加 .....	120
	CANCEL -- 单轴停止/轴组停止 .....	122

DATUM -- 回零 .....	124
DATUM_OFFSET -- 原点位置偏移 .....	128
VMOVE -- 持续运动 .....	129
FORWARD -- 正向运动 .....	130
REVERSE -- 负向运动 .....	131
MOVEMODIFY -- 修改运动位置 .....	131
7.2. 多轴运动指令 .....	134
RAPIDSTOP -- 全部轴停止 .....	134
MOVE -- 直线运动 .....	136
MOVEABS -- 直线运动-绝对 .....	138
MOVEMODIFY2 -- 运动新位置 .....	139
MOVECIRC -- 圆心画弧 .....	141
MOVECIRCABS -- 圆心画弧-绝对 .....	143
MOVECIRC2 -- 三点画弧 .....	144
MOVECIRC2ABS -- 三点画弧-绝对 .....	145
MHELICAL -- 圆心螺旋 .....	146
MHELICALABS -- 圆心螺旋-绝对 .....	147
MHELICAL2 -- 三点螺旋 .....	148
MHELICAL2ABS -- 三点螺旋-绝对 .....	150
MECLIPSE -- 椭圆 .....	152
MECLIPSEABS -- 椭圆-绝对 .....	154
MSPHERICAL -- 空间圆弧 .....	156
MOVESPIRAL -- 渐开线圆弧 .....	160
MOVESPLINE / MOVESPLINEABS -- 样条插补 .....	162
MOVE_TURNABS -- 旋转台插补 .....	163
MCIRC_TURNABS -- 旋转台插补-绝对 .....	165
MOVESMOOTH -- 倒圆角 .....	166
MOVEMODIFY2 -- 修改多轴运动位置 .....	167
*SP -- 运动单独速度 .....	169
MOVESCAN -- 振镜运动 .....	171
7.3. 特殊运动指令 .....	173
MOVE_PAUSE -- 运动暂停 .....	173
MOVE_RESUME -- 运动恢复 .....	174
MOVE_PT -- 单位时间距离 .....	175
MOVE_PTABS -- 单位时间距离绝对 .....	176
MOVE_OP -- 缓冲输出 .....	178
MOVE_OP2 -- 缓冲输出 2 .....	182
MOVE_TABLE -- 缓冲 Table .....	183
MOVE_PARA -- 缓冲参数 .....	184
MOVE_PWM -- 缓冲 PWM .....	187
MOVE_SYNMOVE -- 缓冲触发其他轴 .....	187
MOVE_ASYNMOVE -- 缓冲触发其他轴 2 .....	188
MOVE_TASK -- 缓冲开启任务 .....	189
MOVE_AOUT -- 缓冲模拟量输出 .....	190
MOVE_DELAY -- 缓冲延时 .....	191
MOVE_WAIT -- 缓冲等待 .....	192
MOVE_CANCEL -- 缓冲停止 .....	193
MOVELIMIT -- 速度限制 .....	193
7.4. 同步运动指令 .....	194
CONNECT -- 同步运动 .....	194
CONNPATH -- 同步运动 2 .....	196

CAM -- 凸轮表运动.....	197
CAMBOX -- 跟随凸轮表运动 .....	201
MOVELINK -- 自动凸轮 .....	203
MOVESLINK -- 自动凸轮 2 .....	209
MOVELINK_MODIFY -- 同步距离修改.....	212
MOVESYNC -- 同步运动 .....	216
FLEXLINK -- 激励运动 .....	221
7.5. 运动设置指令 .....	223
CLUTCH_RATE -- 连接速度.....	223
ENCODER_RATIO -- 编码器齿轮比 .....	225
STEP_RATIO -- 电机齿轮比.....	226
BACKLASH -- 反向间隙补偿 .....	226
PITCHSET -- 螺距补偿 .....	227
7.6. 机械手指令 .....	228
CONNFRAME -- 机械手逆解.....	228
CONNREFRAME -- 机械手正解.....	231
FRAME -- 机械手类型.....	232
FRAME_STATUS -- 机械手轴状态.....	232
FRAME_TRANS2 -- 正逆解坐标转换 .....	232
FRAME_ROTATE -- 工件坐标系变换 .....	234
FRAME_ROTATE2 -- 坐标系变换计算 .....	236
WORLD_DPOS -- 世界坐标系.....	239
MOVER_L/MOVER_LABS -- 关节轴直线插补 .....	239
MOVER_C/MOVER_CABS -- 关节轴平面圆弧.....	240
MOVER_C3/MOVER_C3ABS -- 关节轴空间圆弧.....	241
FRAME_CAL -- 参数矫正 .....	242
第八章 程序结构与流程指令 .....	243
8.1. 程序符号 .....	243
' -- 注释.....	243
_ -- 换行.....	243
: -- 标号 .....	243
8.2. 数据定义指令 .....	244
CONST -- 常量定义.....	244
DIM -- 变量定义.....	244
LOCAL -- 局部定义.....	245
GLOBAL -- 全局定义.....	245
8.3. 数组操作指令 .....	246
DMINS -- 数组链表插入.....	246
DMADD -- 数组批量增加.....	246
DMDEL -- 数组链表删除.....	247
DMCPY -- 数组拷贝.....	247
DMSET -- 数组赋值 .....	248
8.4. 自定义子函数指令 .....	248
SUB -- 自定义子函数 SUB .....	248
SUB_PARA -- SUB 传递参数 .....	249
SUB_IFPARA -- SUB 传参判断.....	249
GOSUB/CALL -- SUB 调用 .....	250
GSUB -- 自定义子函数-G 代码格式.....	251
GSUB_PARA -- GSUB 传递参数 .....	251
GSUB_IFPARA -- 判断 GSUB 是否传入参数.....	252
END SUB -- 自定义函数结束.....	252

RETURN -- 函数返回值.....	252
8.5.    结构体定义指令 .....	253
STRUCTURE -- 结构体定义 .....	253
UNION -- 共用体定义 .....	255
8.6.    跳转指令 .....	256
GOTO -- 强制跳转.....	256
ON GOSUB -- 条件跳转 .....	256
ON GOTO -- 条件跳转 2.....	257
8.7.    条件判断指令 .....	257
IF -- 条件判断结构 .....	257
THEN -- 条件判断结构 .....	258
ENDIF -- 条件判断结构.....	258
ELSEIF -- 条件判断结构.....	258
8.8.    循环指令 .....	259
FOR -- for 循环结构 .....	259
TO -- for 循环结构.....	259
STEP -- for 循环结构.....	259
NEXT -- for 循环结构 .....	260
WHILE -- while 循环结构.....	260
WEND -- while 循环结构.....	260
EXIT -- 退出循环.....	260
REPEAT -- 条件循环 .....	261
UNTIL -- 条件结构.....	261
8.9.    等待执行指令 .....	261
DELAY -- 延时.....	261
WAIT UNTIL -- 等待条件满足 .....	261
WAIT IDLE -- 等待轴停止.....	262
WAIT LOADED -- 等待轴缓冲空 .....	262
8.10.   ZINDEX 指针指令 .....	264
ZINDEX_LABEL -- 建立指针索引 .....	264
ZINDEX_CALL -- 访问 SUB 函数.....	264
ZINDEX_ARRAY -- 访问数组 .....	265
ZINDEX_VAR -- 访问变量.....	265
ZINDEX_STRUCT -- 访问结构体.....	266
第九章 任务相关指令 .....	267
9.1.    任务启停指令 .....	267
RUN -- 启动文件任务 .....	267
RUNTASK -- 启动 SUB 任务 .....	267
END -- 结束 .....	268
STOP -- 停止文件任务 .....	268
STOPTASK -- 停止 SUB 任务 .....	269
HALT -- 停止全部任务 .....	269
PAUSE -- 暂停全部任务 .....	269
PAUSETASK -- 暂停指定任务 .....	270
RESUMETASK -- 恢复指定任务 .....	270
9.2.    三次文件任务指令 .....	270
FILE3_RUN -- 执行 FILE3 任务 .....	270
FILE3_ONRUN --FILE3 回调函数.....	271
FILE3_GOTO -- FILE3 强制跳转.....	271
FILE3_LINE -- FILE3 行号.....	271
9.3.    任务参数指令 .....	272

BASE_MOVE -- 指定主轴 .....	272
PROC_STATUS -- 任务状态 .....	272
PROC -- 任务编号 .....	273
PROCNUMBER -- 当前任务编号 .....	273
PROC_LINE -- 任务行号 .....	273
PROC_PROGRESS -- 任务指令进度 .....	274
PROC_PRIORITY -- 任务优先级 .....	274
ERROR_LINE -- 任务错误行号 .....	274
RUN_ERROR -- 任务错误码 .....	275
TICKS -- 任务计数周期 .....	275
TIME_TICKUS -- 任务计数周期 .....	275
第十章 运算符及数学函数指令 .....	277
10.1. 算术运算指令 .....	277
+ -- 加法运算 .....	277
-- 减法运算 .....	277
* -- 乘法运算 .....	278
/-- 除法运算 .....	278
\-- 整除运算 .....	278
<< -- 左移位 .....	279
>> -- 右移位 .....	279
MOD -- 求余数 .....	280
ABS -- 绝对值 .....	281
10.2. 比较运算指令 .....	281
= -- 比较/赋值运算 .....	281
<> -- 不等于 .....	281
> -- 大于 .....	282
>= -- 大于等于 .....	282
< -- 小于 .....	283
<= -- 小于等于 .....	283
10.3. 逻辑运算指令 .....	283
AND -- 按位与 .....	283
OR -- 按位或 .....	284
NOT -- 按位非 .....	285
XOR -- 按位异或 .....	285
EQV -- 按位同或 .....	286
10.4. 三角函数指令 .....	286
SIN -- 三角函数正弦 .....	286
ASIN -- 三角函数反正弦 .....	286
COS -- 三角函数余弦 .....	287
ACOS -- 三角函数反余弦 .....	287
TAN -- 三角函数正切 .....	287
ATAN -- 三角函数反正切 .....	287
ATAN2 -- 三角函数反正切 2 .....	288
10.5. 指数运算指令 .....	288
EXP -- 指数 .....	288
SQR -- 平方根 .....	288
LN -- 自然对数 .....	289
LOG -- 对数底为 10 .....	289
10.6. 数据操作指令 .....	289
SET_BIT -- 按位设置 .....	289
CLEAR_BIT -- 按位置 0 .....	290

READ_BIT -- 按位读取 .....	291
READ_BIT2 -- 按位读取 2 .....	291
FRAC -- 返回小数 .....	292
INT -- 返回整数 .....	292
SGN -- 返回符号 .....	292
IEEE_IN -- 组合浮点数 .....	292
IEEE_OUT -- 提取单字节 .....	293
\$ -- 16 进制 .....	294
10.7. 字符串操作指令 .....	294
CHR -- ASCII 码打印 .....	294
HEX -- 16 进制打印 .....	294
STRLEN -- 返回字符串长度 .....	295
TOSTR -- 格式化输出 .....	295
STRCOMP -- 字符串比较 .....	295
STRFIND -- 字符串搜索 .....	296
VAL -- 字符转数值 .....	296
10.8. 常数指令 .....	297
PI -- 圆周率 .....	297
TRUE -- 真值 .....	297
FALSE -- 假值 .....	297
ON -- 开启 .....	297
OFF -- 关闭 .....	297
10.9. 高级运算指令 .....	298
CRC16 -- CRC 检验计算 .....	298
DTSMOOTH -- table 平滑 .....	298
B_SPLINE -- B 样条平滑 .....	299
TURN_POSMAKE -- 旋转坐标计算 .....	300
ZCUSTOM -- 运动参数计算 .....	300
ZMATH64 -- 64 位计算 .....	304
MODBUS_DOUBLE -- 读取 MODBUS .....	305
第十一章 轴参数与轴状态指令 .....	306
11.1. 轴选择 .....	306
BASE -- 轴选择/轴组选择 .....	306
AXIS -- 临时轴选择 .....	307
11.2. 基本参数指令 .....	307
UNITS -- 脉冲当量 .....	307
ATYPE -- 轴类型 .....	308
AXIS_ADDRESS -- 轴地址设置 .....	310
AXIS_ENABLE -- 单轴使能 .....	312
11.3. 速度参数指令 .....	313
SPEED -- 运动速度 .....	313
ACCEL -- 加速度 .....	314
DECEL -- 减速度 .....	315
CREEP -- 爬行速度 .....	316
LSPEED -- 起始速度 .....	317
FORCE_SPEED -- SP 速度 .....	318
STARTMOVE_SPEED -- SP 运动开始速度 .....	319
ENDMOVE_SPEED -- SP 运动结束速度 .....	320
FASTDEC -- 快减减速度 .....	322
MSPEED -- 实际反馈速度 .....	323
SPEED_RATIO -- 速度比例 .....	323

SRAMP -- 加减速曲线 .....	325
VP_SPEED -- 当前运动速度 .....	326
INTERP_FACTOR -- 插补速度计算 .....	327
11.4. 轴状态查看指令 .....	329
MTYPE -- 当前运动类型 .....	329
NTYPE -- 下条运动类型 .....	330
AXISSTATUS -- 轴状态 .....	330
IDLE -- 运动状态 .....	331
ADDAX_AXIS -- 叠加轴号 .....	332
AXIS_STOPREASON -- 轴停止原因 .....	332
LINK_AXIS -- 连接轴号 .....	333
11.5. 运动前瞻指令 .....	333
CORNER_MODE -- 拐角设置 .....	333
DECEL_ANGLE -- 拐角减速开始 .....	338
STOP_ANGLE -- 拐角减速结束 .....	339
FULL_SP_RADIUS -- 限速半径 .....	340
SPLIMIT_RADIUS -- 限速值 .....	341
ZSMOOTH -- 倒角半径 .....	341
MERGE -- 连续插补 .....	341
11.6. 运动缓冲指令 .....	342
LOADED -- 缓冲空 .....	342
MOVES_BUFFERED -- 当前缓冲数 .....	343
REMAIN_BUFFER -- 剩余缓冲数 .....	343
MOVE_MARK -- 运动标号 .....	343
MOVE_CURMARK -- 当前运动标号 .....	344
LIMIT_BUFFERED -- 运动缓冲限制 .....	344
11.7. 位置相关指令 .....	345
DPOS -- 轴指令位置 .....	345
MPOS -- 编码器反馈位置 .....	345
DEFPOS -- 位置偏移 .....	345
OFFPOS -- 偏移位置 .....	346
ENDMOVE -- 当前运动目标位置 .....	347
VECTOR_MOVED -- 当前运动距离 .....	348
REMAIN -- 当前运动剩余距离 .....	349
VECTOR_BUFFERED -- 缓冲剩余距离 .....	349
ENDMOVE_BUFFER -- 缓冲最终位置 .....	350
11.8. 原点回零指令 .....	351
DATUM_IN -- 映射原点输入 .....	351
HOMEWAIT -- 回零反找延时 .....	351
11.9. JOG 运动指令 .....	353
FAST_JOG -- 映射点动输入 .....	353
FWD_JOG -- 映射正向 JOG 输入 .....	354
REV_JOG -- 映射负向 JOG 输入 .....	355
JOGSPEED -- JOG 速度 .....	355
FHOLD_IN -- 映射保持输入 .....	356
FHSPEED -- 保持速度 .....	357
11.10. 编码器相关指令 .....	358
ENCODER -- 编码器原始值 .....	358
ENCODER_STATUS -- 编码器状态 .....	358
ENCODER_FILTER -- 编码器滤波 .....	358
PP_STEP -- 编码器内部比例 .....	359



11.11. 锁存相关指令 .....	359
REGIST -- 锁存 .....	359
REG_INPUTS -- 锁存输入映射 .....	363
MARK -- 锁存触发 .....	364
MARKB -- 锁存 2 触发 .....	364
MARKC -- 锁存 3 触发 .....	364
MARKD -- 锁存 4 触发 .....	365
OPEN_WIN -- 锁存开始坐标范围 .....	365
CLOSE_WIN -- 锁存结束坐标范围 .....	365
REG_POS -- 锁存位置 .....	365
REG_POSB -- 锁存 2 位置 .....	366
REG_POSC -- 锁存 3 位置 .....	366
REG_POSD -- 锁存 4 位置 .....	366
11.12. 限位参数指令 .....	367
FS_LIMIT -- 正向软限位设置 .....	367
RS_LIMIT -- 负向软限位设置 .....	367
FWD_IN -- 映射正限位输入 .....	368
REV_IN -- 映射负限位输入 .....	368
ALM_IN -- 映射报警输入 .....	369
11.13. 限幅参数指令 .....	369
REP_OPTION -- 坐标循环模式 .....	369
REP_DIST -- 坐标循环位置 .....	371
FE -- 当前随动误差 .....	371
FE_LIMIT -- 最大随动误差设置 .....	371
FE_RANGE -- 报警时随动误差 .....	371
11.14. 高级设置指令 .....	372
INVERT_STEP -- 脉冲模式设置 .....	372
MAX_SPEED -- 脉冲频率限制 .....	373
AXIS_ZSET -- 精准 op 设置 .....	373
AXIS_MODE -- connect 运动保持 .....	374
MOVEOP_DELAY -- 缓冲输出延时 .....	376
DAC -- 总线轴模拟量控制 .....	376
ERRORMASK -- 错误时操作 .....	379
ZSCAN_CORRECT -- 振镜矫正 .....	379
11.15. 预留指令 .....	380
D_GAIN -- 微分增益 .....	380
I_GAIN -- 积分增益 .....	380
OV_GAIN -- 速度增益 .....	380
P_GAIN -- 比例增益 .....	380
VFF_GAIN -- 前馈增益 .....	381
SERVO -- 闭环开关 .....	381
TRANS_DPOS .....	381
第十二章 输入输出相关指令 .....	382
12.1. 输入相关指令 .....	382
IN -- 输入口 .....	382
AIN -- 模拟量输入 .....	382
ZSIMU_IN -- 仿真 IN 输入 .....	383
ZSIMU_AIN -- 仿真模拟量输入 .....	383
ZSIMU_ENCODER -- 仿真编码器输入 .....	383
INVERT_IN -- 反转输入 .....	383
IN_SCAN -- 扫描输入变化 .....	384

IN_EVENT -- 读取输入变化 .....	384
SCAN_EVENT -- 检测变化 .....	385
IN_BUFF -- 读取输入缓冲 .....	385
INFILTER -- 输入口滤波 .....	386
12.2. 输出相关指令 .....	386
OP -- 输出口 .....	386
AOUT -- 模拟量输出 .....	387
READ_OP -- 读取输出口 .....	387
12.3. 位置比较输出指令 .....	388
PSWITCH -- 软件位置比较输出 .....	388
HW_PSWITCH -- 硬件位置比较输出 .....	390
HW_TIMER -- 硬件定时 .....	392
HW_PSWITCH2 -- 总线硬件位置比较输出 .....	394
HW_PS2AXISNUM -- 设置 PS2 轴号 .....	402
HW_PS2COUNTS -- PS2 比较的点数 .....	404
12.4. PMW 控制指令 .....	405
PWM_FREQ -- PWM 频率 .....	405
PWM_DUTY -- pwm 占空比 .....	405
第十三章 通讯相关指令 .....	407
13.1. 串口通讯指令 .....	407
SETCOM -- 串口配置 .....	407
ADDRESS -- 控制器站号 .....	409
COM_UNUSED -- 指定串口 .....	409
13.2. CAN 通讯指令 .....	410
CAN -- CAN 通讯 .....	410
CANIO_ADDRESS -- CAN 通讯设置 .....	412
CANIO_ENABLE -- CAN 使能 .....	413
CANIO_STATUS -- CAN 扩展板状态 .....	413
CANIO_INFO -- CAN 扩展板信息 .....	414
13.3. 自定义通讯指令 .....	414
GET # -- 读取字符 .....	414
OPEN # -- 打开自定义网口通讯 .....	415
PRINT # -- 输出字符串 .....	416
PUTCHAR # -- 输出字符 .....	417
PORT_TARGET -- IP 和端口号配置 .....	418
13.4. 打印输出指令 .....	419
PRINT -- 打印信息 .....	419
ERRSWITCH -- 信息输出设置 .....	420
TRACE -- 打印信息 2 .....	421
WARN -- 警告信息 .....	421
ERROR -- 错误信息 .....	421
13.5. 通道参数指令 .....	422
PORT -- 通道编号 .....	422
PORT_STATUS -- 通道状态 .....	423
FILE_PORT -- 当前通道文件号 .....	424
PROTOCOL -- 通道通讯协议 .....	424
ETH_MODE -- 网口模式设置 .....	424
13.6. MODBUS 通讯指令 .....	425
MODBUS_BIT -- 位寄存器 .....	425
MODBUS_IEEE -- 字寄存器-32 位浮点型 .....	426
MODBUS_LONG -- 字寄存器-32 位整型 .....	426

MODBUS_REG -- 字寄存器-16 位整型 .....	427
MODBUS_STRING -- 字寄存器-字节 .....	427
MODBUSM_DES -- modbus 通讯连接 .....	428
MODBUSM_DES2 -- 控制器间网口通讯 .....	429
MODBUSM_STATE -- modbus 通讯状态 .....	431
MODBUSM_REGSET -- 写对端保持寄存器 .....	431
MODBUSM_REGGET -- 读对端保持寄存器 .....	432
MODBUSM_3XGET -- 读对端输入寄存器 .....	433
MODBUSM_BITSET -- 写对端线圈 .....	433
MODBUSM_BITGET -- 读对端线圈 .....	433
MODBUSM_1XGET -- 读对端离散输入 .....	434
13.7. 控制器互联直接命令指令 .....	434
SEND_RESULT -- 读取 send 结果 .....	434
SEND_CMD -- send 命令 .....	435
SEND_CMDAXIS -- send 命令 .....	435
SEND_ASSIGN -- send 命令 .....	435
SEND_QUERY -- send 命令 .....	436
SEND_QUERYSET -- send 命令 .....	437
13.8. 控制器互联文件发送指令 .....	437
SEND_ZAR -- U 盘操作 .....	437
SEND_FLASH -- 数据拷贝 .....	437
SEND_FILE -- 拷贝 U 盘数据 .....	438
SEND_IFLASH -- 拷贝 flash 数据 .....	438
SEND_PERCENT -- 查询指令进度 .....	439
ZAR_CONTROL -- 查看控制器类型 .....	439
第十四章 系统相关指令 .....	440
14.1. 控制器加密指令 .....	440
APP_PASS -- 密码 .....	440
LOCK -- 锁定控制器 .....	440
UNLOCK -- 解锁控制器 .....	441
14.2. 系统时间指令 .....	441
DATE -- 系统日期 .....	441
DATES\$ -- 系统日期 .....	441
DAY -- 系统星期 .....	442
DAY\$ -- 系统星期 .....	442
RTC_DATE -- 系统日期 .....	442
TIME -- 系统时间 .....	443
TIMES\$ -- 系统时间 .....	443
RTC_TIME -- 系统时间 .....	444
14.3. 轴系统参数指令 .....	444
WDOG -- 轴总使能 .....	444
DISABLE_GROUP -- 轴分组 .....	444
ERROR_AXIS -- 报错轴号 .....	445
MOTION_ERROR -- 报错轴列表 .....	445
ERROR_SET -- 报错输出 .....	445
RADIUS_ERRSET -- 圆弧插补检查 .....	446
14.4. IP 参数指令 .....	447
IP_ADDRESS -- IP 地址 .....	447
IP_ADDRESS2 -- IP 地址 2 .....	447
IP_GATEWAY -- IP 网关 .....	448
IP_NETMASK -- IP 掩码 .....	448

IP_IFDHCP -- IP 自动获取.....	448
IP_IFDHCP2 -- IP 自动获取 2 .....	449
14.5. 控制器信息指令 .....	449
VERSION_DATE -- 系统固件版本 .....	449
VERSION -- 系统软件版本 .....	450
ID_HARDWARE -- 控制器硬件型号 .....	450
CONTROL -- 控制器软件型号 .....	450
SYSTEM_ZSET -- 控制器设置.....	451
LEDOUT -- 控制器指示灯 .....	452
SERIAL_NUMBER -- 控制器唯一 ID.....	452
SERVO_PERIOD -- 总线通讯周期.....	453
SYS_ZFEATURE -- 系统规格 .....	453
14.6. TABLE 数组指令 .....	455
TABLE -- 系统缺省数组 .....	455
TSIZE -- table 大小 .....	455
TABLESTRING -- 字符串格式打印 table .....	455
14.7. 示波器相关指令 .....	456
TRIGGER -- 触发示波器 .....	456
SCOPE -- 数据采样 .....	456
SCOPE_POS -- 采样点数 .....	457
14.8. VR 相关指令.....	458
CLEAR -- 清除 VR.....	458
VR -- 掉电保存 .....	458
VR_INT -- 掉电保存整型.....	458
VRSTRING -- 掉电保存字符串 .....	459
第十五章 存储相关指令 .....	460
15.1. U 盘相关指令 .....	460
FILE -- U 盘文件操作 .....	460
U_STATE -- U 盘状态 .....	463
U_READ -- 从 U 盘读取 .....	464
U_READDBL -- 从 U 盘读取-double .....	464
U_READ2 -- 从 U 盘读取 2 .....	465
U_READ2DBL -- 从 U 盘读取 2-double .....	465
U_READDSB -- DSB 文件读取 .....	466
U_WRITE -- 输出到 U 盘 .....	467
U_WRITEDBL -- 输出到 U 盘-double .....	467
STICK_READ -- U 盘读取到 table.....	468
STICK_WRITE -- table 输出到 U 盘 .....	468
STICK_READVR -- U 盘读取到 vr.....	469
STICK_WRITEVR -- vr 输出到 U 盘 .....	469
15.2. FLASH 相关指令.....	470
FLASH_WRITE -- flash 存储.....	470
FLASH_WRITEDBL -- flash 存储-double.....	471
FLASH_READ -- flash 读取 .....	471
FLASH_READDBL -- flash 读取-double .....	472
FLASH_READ2 -- flash 读取 2 .....	472
FLASH_READ2DBL -- flash 读取 2-double .....	473
FLASHVR -- 拷贝 RAM 的数据 .....	474
FLASH_SECTSIZE -- flash 变量数 .....	475
FLASH_SECTES -- flash 块数.....	475
第十六章 中断相关指令 .....	476

16.1.	三种中断指令 .....	476
	INT_ENABLE -- 中断总开关 .....	476
	ONPOWEROFF -- 掉电中断 SUB.....	477
	INT_ONn -- 外部输入中断 SUB.....	478
	INT_OFFn -- 外部输入中断 SUB.....	478
	ONTIMERn -- 定时器中断 SUB.....	479
	INT_CYCLE -- 中断周期执行 .....	479
16.2.	定时器指令 .....	480
	TIMER_IFEND -- 定时器状态.....	480
	TIMER_START -- 启动定时器 .....	481
	TIMER_STOP -- 停止定时器.....	481
第十七章	总线相关指令 .....	482
17.1.	编号释义 .....	482
	槽位号 .....	482
	设备号 .....	482
	驱动器编号 .....	482
17.2.	基础指令 .....	482
	SLOT_SCAN -- 总线扫描 .....	482
	SLOT_START -- 总线开启 .....	483
	SLOT_STOP -- 总线停止 .....	484
	?*SLOT -- 打印总线接口 .....	484
	?*ETHERCAT -- 打印 EtherCAT 总线状态.....	485
	?*RTEX -- 打印 Rtex 总线状态 .....	485
	ZTEST -- EtherCAT 总线信息查询.....	486
17.3.	SDO 操作指令 .....	487
	SDO_WRITE -- 数据字典写入 .....	487
	SDO_WRITE_AXIS -- 数据字典写入.....	488
	SDO_READ -- 数据字典读取.....	489
	SDO_READ_AXIS -- 数据字典读取.....	489
17.4.	设备相关指令 .....	490
	NODE_COUNT -- 设备个数.....	490
	NODE_STATUS -- 设备状态 .....	491
	NODE_AXIS_COUNT -- 设备电机数.....	491
	NODE_IO -- 设备 IO .....	491
	NODE_AIO -- 设备模拟量.....	492
	NODE_INFO -- 设备信息 .....	492
	NODE_PROFILE -- PDO 预设置 .....	493
	NODE_PDOBUFF -- 特殊设备 PDO 设置.....	493
	NODE_PRESET -- 设备预配置 .....	494
17.5.	驱动器相关指令 .....	495
	DRIVE_MODE -- 驱动器模式.....	495
	DRIVE_PROFILE -- 驱动器 PDO 设置 .....	495
	DRIVE_CW_MODE -- 驱动器设置 .....	499
	DRIVE_CONTROLWORD -- 驱动器控制字.....	499
	DRIVE_STATUS -- 驱动器状态 .....	501
	DRIVE_IO -- 驱动器 IO .....	502
	DRIVE_TORQUE -- 驱动器力矩 .....	502
	DRIVE_FE -- 驱动器误差.....	503
	DRIVE_FE_LIMIT -- 驱动器误差限制 .....	503
	DRIVE_CLEAR -- 清除报警 .....	504
	DRIVE_READ -- 参数读取.....	504

DRIVE_WRITE -- 参数写入 .....	506
第十八章 简易例程 .....	509
18.1. 常用操作 .....	509
IO 操作 .....	509
SP 指令连续插补 .....	509
字符串与数据相互转化 .....	510
手轮 .....	510
飞剪应用 .....	511
位置比较输出 .....	511
掉电保存 .....	511
机械手应用 .....	511
编码器读取 .....	512
自定义 G 代码 .....	513
18.2. 模块通讯 .....	516
CAN 通讯 .....	516
触摸屏通讯 .....	517
自定义网口通讯 .....	521
控制器之间通讯 .....	522
自定义串口通讯和字符串使用 .....	522
18.3. 总线初始化 .....	523
EtherCAT 初始化程序 .....	523
Rtex 初始化程序 .....	524
第十九章 错误与调试 .....	525
19.1. 常见问题列表 .....	525
问题排查 .....	525
19.2. 解决办法 .....	527
手动运动调试 .....	527
断点调试 .....	528
示波器抓取 .....	529
寄存器查看 .....	529
在线发送命令 .....	530
打印程序信息 .....	530
IO 口快速检测 .....	531
轴参数状态判断 .....	531
附录一 错误码列表 .....	533
附录二 模块扩展 .....	544
ZCAN 扩展模块 .....	544
扩展接线 .....	544
资源映射 .....	545
EtherCAT 扩展模块 .....	549
扩展接线 .....	549
资源映射 .....	550
附录三 触摸屏通讯 .....	551
控制器与触摸屏通讯简介 .....	551
控制器与触摸屏连接 .....	552
连接 ZHD 系列触摸屏 .....	552
连接第三方触摸屏 .....	553
附录四 ETHERCAT 通讯 .....	558
附录五 RTEX 总线 .....	562

# 第一章 运动控制产品简介

## 1.1. 运动控制产品概述

运动控制实现了对机械传动部件的位置、速度、加速度等的实时控制，使其按照预期的轨迹与规定的运动参数完成相应的动作。

控制系统以处理器、检测机构、执行机构为核心，实现逻辑控制、位置控制、轨迹加工控制、机器人运动控制等。其中处理器通常是可编程控制器、单片机、或运动控制器，相当与系统的大脑，主要负责对接受到的信号进行逻辑处理，并给执行机构下发命令，协调系统的正常运转。检测机构通常由各种传感器构成，相当与系统的眼睛，目的是检测系统条件的变化并反馈给控制器，执行机构通常由伺服单元、阀门构成，相当于系统的双手，主要执行控制器下发的命令。

运动控制器是运动控制系统的核心部件，负责产生运动路径的控制指令，用于设备的逻辑控制，将运动参数分配给需要运动的轴，并对被控对象的外部环境变化及时做出响应。

通用运动控制器通常都提供一系列运动规划方法，基于对冲击、加速度和速度等这些可影响动态轨迹精度的量值加以限制，提供对运动控制过程的运动参数的设置和运动相关的指令，使其按预先规定的运动参数和规定的轨迹完成相应的动作。

## 1.2. 运动控制产品优点

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，单个控制器可支持 128 轴的运动控制。

运动控制产品支持直线、圆弧、空间圆弧、椭圆、螺旋等多种插补运动，单个插补通道最多支持 16 轴联合插补。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、精准输出、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、六关节等 30 多种机械手运动控制算法，单个控制器可以控制多个机械手，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。

总线型运动控制产品支持 EtherCAT、RTEX 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 总线、RTEX 总线和脉冲轴三者混合使用。

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，同时还支持总线轴硬件位置锁存与位置比较输出。

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、U 盘、CAN 总线、RS485 串口、RS232 串口等通讯接口，通过 CAN 总线或 EtherCAT 总线可以连接正运动的扩展模块，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）。

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 VC，VB，C#，PYTHON，LABVIEW 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. 控制器主要功能描述

控制器的主要功能简介：

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、轴参数等
编程语言种类		ZDevelop 编程（BASIC, PLC, HMI），或常用上位机编程语言
在线命令		在线命令栏输入指令参数发送给控制器立即执行
通讯接口	串口	232 串口和 485 串口，支持 MODBUS_RTU 协议和自定义通讯
	网口	通讯速度快，接线方便，支持 MODBUS_TCP 协议和自定义通讯
	U 盘	插入 U 盘，数据交互
	CAN 总线	连接 ZIO 扩展模块，控制器互联
	EtherCAT 总线	连接 EtherCAT 驱动器或 EtherCAT 扩展模块
	RTEX 总线	连接 RTEX 驱动器
数据类型	自定义数组	集中相同数据类型的元素，默认浮点型
	自定义变量	默认浮点型
	自定义常量	可以是布尔型，字符串型，时间型，日期型，整型等
	寄存器	自带四类寄存器，TABLE, MODBUS, VR, FLASH
常用运动控制功能	点位运动	JOG 点动
	插补运动	直线、圆弧、空间圆弧、椭圆、螺旋等多种插补，支持连续插补
	电子齿轮	建立主从轴之间的电子齿轮连接
	电子凸轮	分凸轮表运动和自动凸轮两类
	运动叠加	不同轴的运动叠加
	轨迹前瞻	根据前瞻参数，速度自行优化
	位置锁存	根据外部信号触发的发生记录轴的位置
	位置比较输出	到达比较点输出 OP 信号，连续比较，快速响应
	精准输出	OP 快速响应

### 1.4. 控制器应用场景

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。

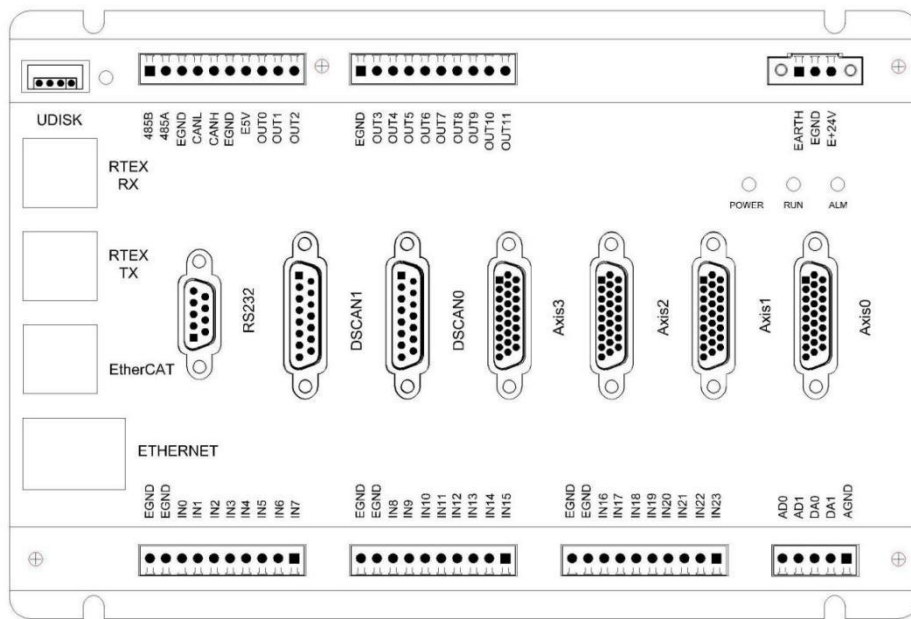
## 1.5. 控制器接口

这里用 ZMC420SCAN 总线型运动控制器为例展开说明。

ZMC420SCAN 总线型运动控制器支持 EtherCAT 总线和 RTEX 总线连接，最多支持 20 个轴的运动控制，支持不同类型轴(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，支持全功能运动控制，产品详细参数参见硬件手册。

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 总线接线两端需要并接 120 欧姆的电阻)，扩展方法参见“[模块扩展](#)”。

ZMC420SCAN 控制器外观如下图所示：



接口功能如下表：

标识	接口	个数	说明
RS232	232 串口	1 个	采用 MODBUS_RTU 协议
485	485 串口	1 个	采用 MODBUS_RTU 协议
CAN	CAN 总线	1 个	连接 CAN 扩展模块或控制器
ETHERNET	网口	1 个	采用 MODBUS_TCP 协议，通过交换机扩展接口个数
EtherCAT	EtherCAT 总线	1 个	连接 EtherCAT 驱动器或 EtherCAT 扩展模块
RTEX	RTEX 总线	1 个	连接 RTEX 驱动器
UDISK	U 盘	1 个	插入 U 盘设备
E +24V	主电源	1 个	24V 直流电源供电

IN	数字量输入	24 个	NPN 类型，内部 24V 供电
OUT	数字量输出	12 个	NPN 类型，内部 24V 供电
AD	模拟量输入	2 个	精度 12 位，0-10V
DA	模拟量输出	2 个	精度 12 位，0-10V
DSCAN	振镜轴接口	2 个	接入激光振镜，支持 XY2-100 协议
Axis	脉冲轴接口	4 个	每个接口包含脉冲输出和编码器输入

## 1.6. 控制器的使用

### 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（采用 VC，VB，C#，PYTHON，LABVIEW 等编程）。

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。

### 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

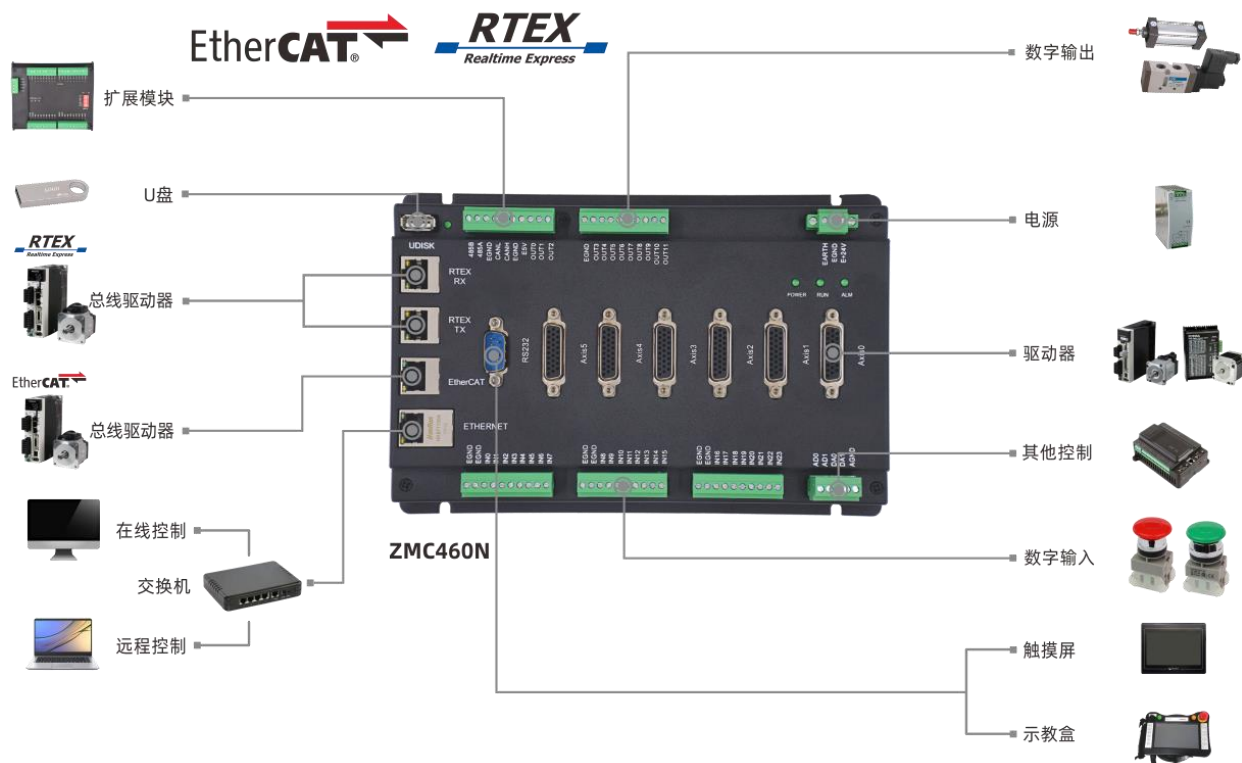
使用 ZDevelop 软件按步骤 1 的设计编写程序，软件快速使用方法参见本文“[新建工程](#)”小节，或打开 ZDevelop 软件菜单栏“帮助”-“ZDevelop 帮助”查看软件具备的各种功能用法介绍，编写任务和程序模块，进行程序模拟调试。

编程需要设置的参数：BASE 选择轴号、ATYPE 轴类型、UNITS 脉冲当量、SPEED 轴速度、ACCEL 轴加速度，DECEL 轴减速度等基础轴参数，再给轴发送运动指令。

若使用 EtherCAT 总线或 RTECH 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“[ZCAN 扩展模块](#)”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。

### 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，控制器接线参考如下图。



**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下图，使用总线连接驱动器只需用网线直接插入对应的 EtherCAT 或 RTEX 接口。

脉冲轴与编码器接口(DB26母头)

针脚号	信 号
1	EGND
2	ALM(IN24-29)
3	S/ON(OUT12-17)
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	备用
26	备用

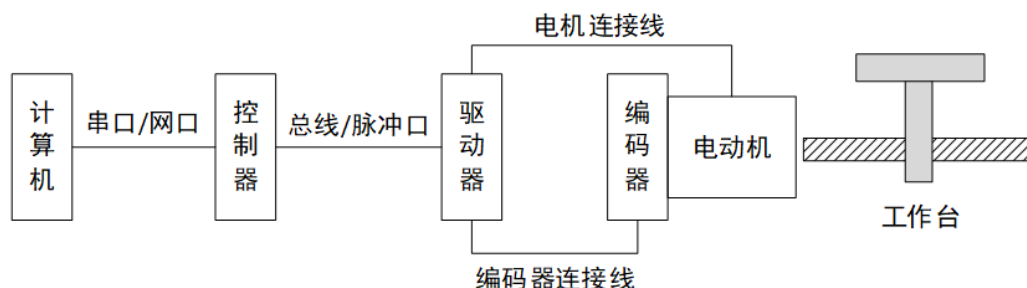
松下A6伺服驱动器

针脚号	信 号
36	ALM-
37	ALM+
29	S/ON
22	OA-
49	OB-
24	OZ-
5	SIGN
4	/PULS
41	COM-
7	COM+
21	OA+
48	OB+
23	OZ+
6	/SIGN
3	PULS
13	GND
25	GND

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，电机接 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 电源单独供电后才可使用。

扩展模块接线：支持扩展 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。

配置参考：



#### 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。

## 第二章 ZDevelop 软件编程

### 2.1. 编程软件简介

ZDevelop 是 ZMotion 系列运动控制器的 PC 端程序开发调试与诊断软件，通过它用户能够很容易的对控制器进行程序编辑与配置，快速开发应用程序、实时监控轴运行参数以及对运动控制器正在运行的程序进行实时调试，支持中英双语环境。

ZDevelop 编程软件支持 ZBasic、ZPLC 梯形图、ZHMI 组态编程，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。

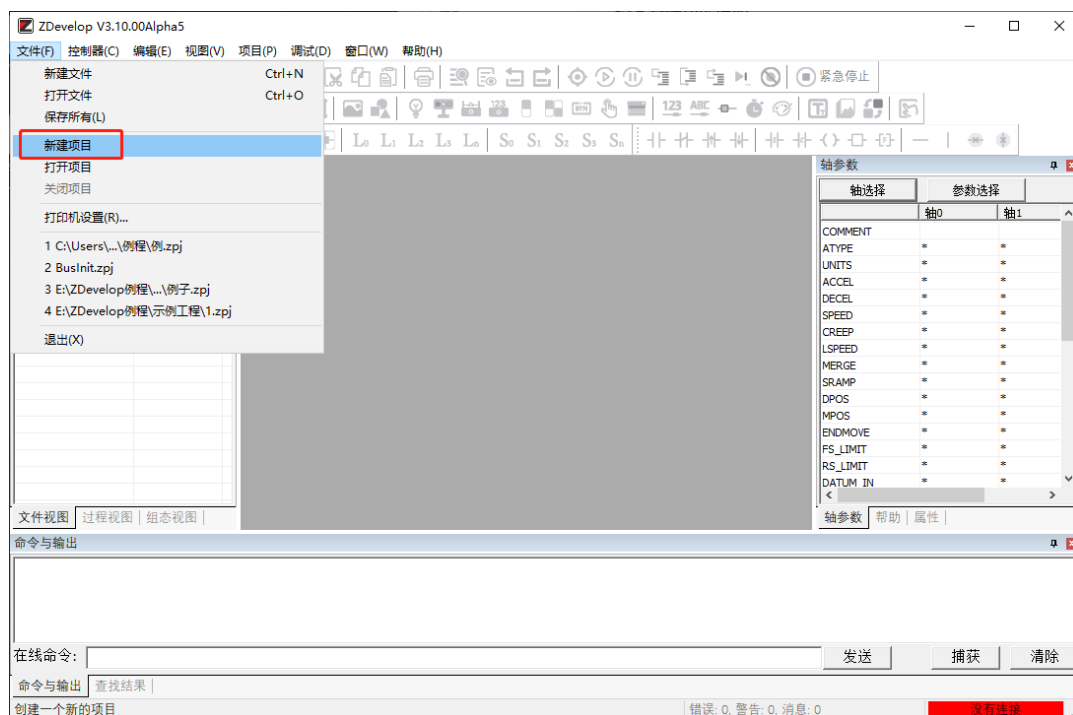
ZBasic 具有实时多任务的特性，多个 ZBasic 程序可以同时构建并多任务实时运行，使得复杂的应用变得简单易行。

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。

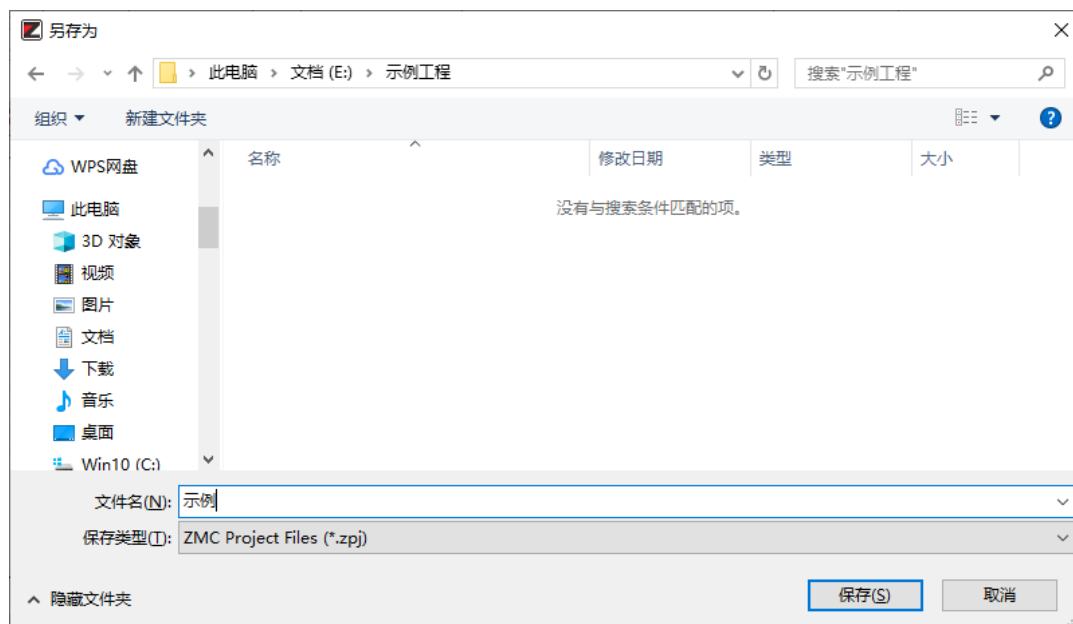
### 2.2. 新建工程

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 V3.10，更新软件版本请前往正运动官方网站下载，网址：[www.zmotion.com.cn](http://www.zmotion.com.cn)。

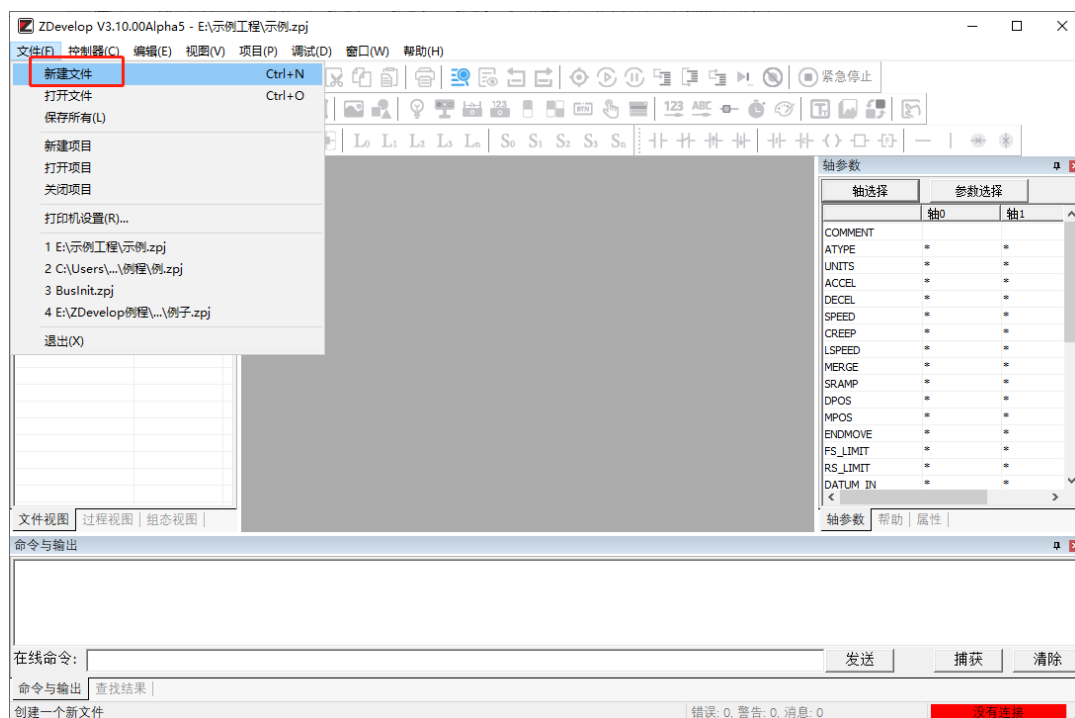
1. 新建项目：菜单栏“文件”→“新建项目”。



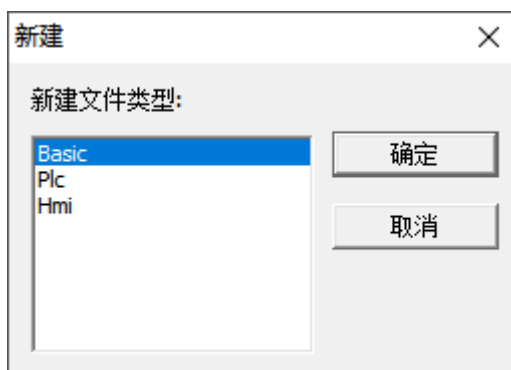
2. 点击“新建项目”后弹出“另存为”界面，选择一个文件夹打开，输入文件名后保存项目，后缀为“.zpj”。



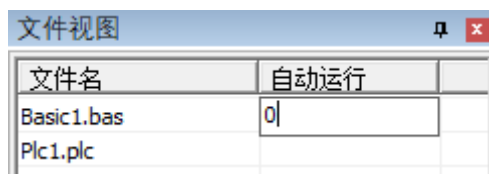
3. 新建文件：菜单栏“文件”→“新建文件”。



点击“新建文件”后，出现下图所示的弹窗，支持 Basic/PLC/Hmi 混合编程，这里选择新建的文件类型为 Basic 后确认。



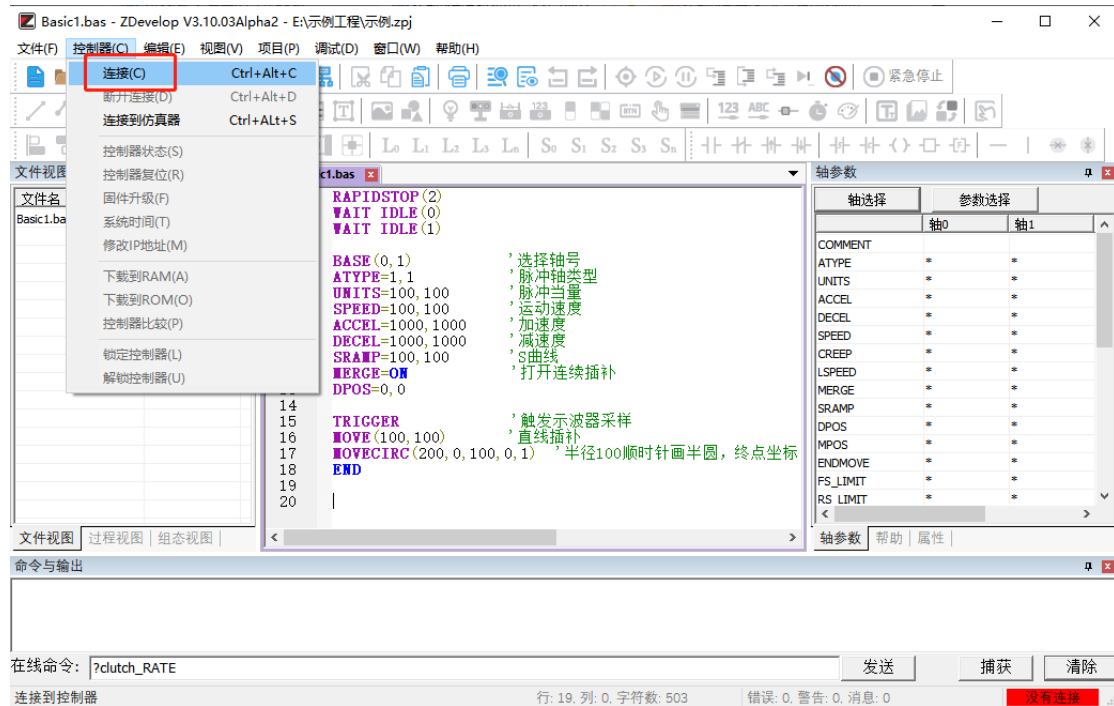
4. 设置文件自动运行：如下图，双击文件右边自动运行的位置，输入任务号“0”。



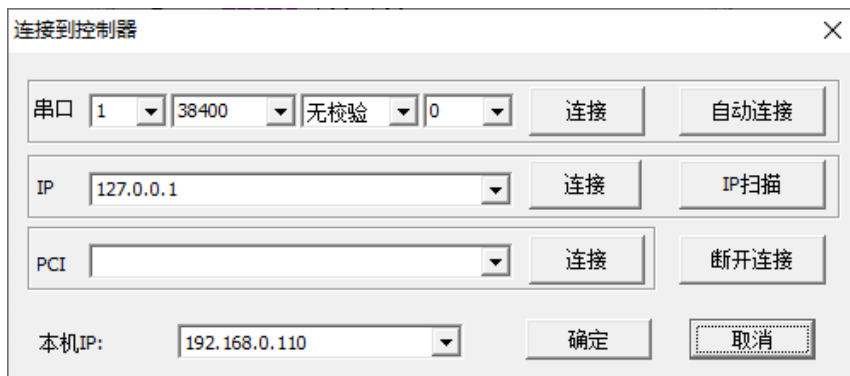
5. 编辑程序：程序编写完成，点击保存文件，新建的 Basic 文件会自动保存到项目 zpj 所在的文件夹下。

6. 连接到控制器：在程序输入窗口编辑好程序，点击“控制器”→“连接”。

没有控制器是可选择连接到仿真器仿真运行，点击“连接”→“连接到仿真器”，便可成功连接到仿真器，并弹出仿真器连接成功提示。



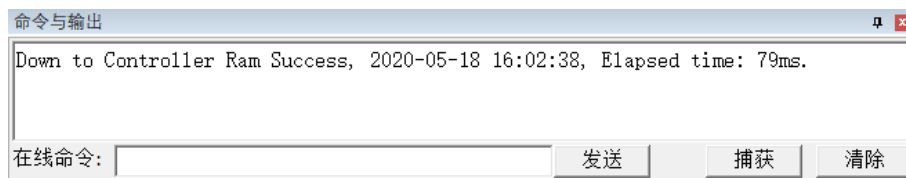
点击“连接”弹出“连接到控制器”窗口，可选择串口连接或网口连接，选择匹配的串口参数或网口 IP 地址后，点击连接即可。连接成功命令与输出窗口打印信息：Connected to Controller:ZMC432 Version:4.64-20170623.



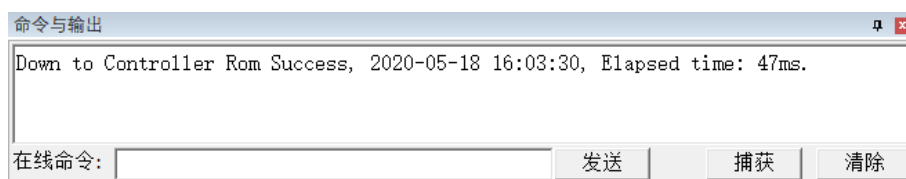
串口连接和网口连接的详细方法参见 ZDevelop 软件菜单栏“帮助”→“ZDevelop 帮助”文档。

7. 下载程序：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。

成功下载到 RAM:



成功下载到 ROM:

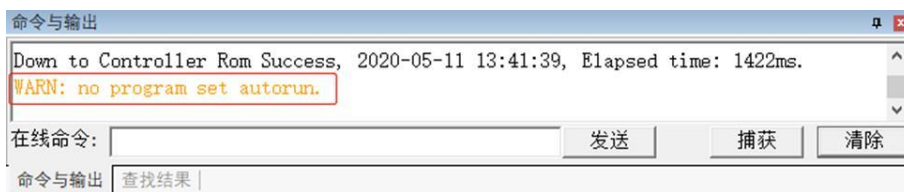




RAM 下载掉电后程序不保存，ROM 下载掉电后程序保存。下载到 ROM 的程序下次连接上控制器之后程序会自动按照任务号运行。

注意事项：

1. 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。
2. ZMC00x 系列控制器不支持下载到 RAM。
3. 不建立项目的时候，只有 Bas 文件无法下载到控制器。
4. 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。
5. 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.



## 2.3. 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：

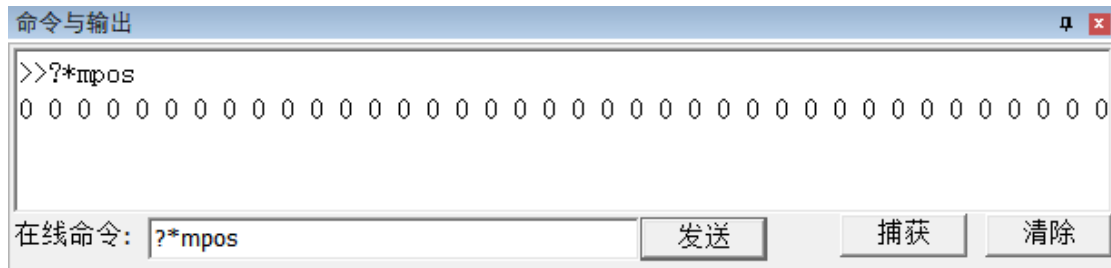
- 0: TRACE、WARN、ERROR 指令全部不输出
- 1: 只输出 ERROR 指令
- 2: 输出 WARN、ERROR 指令
- 3: TRACE、WARN、ERROR 指令全部输出

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，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

?\*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

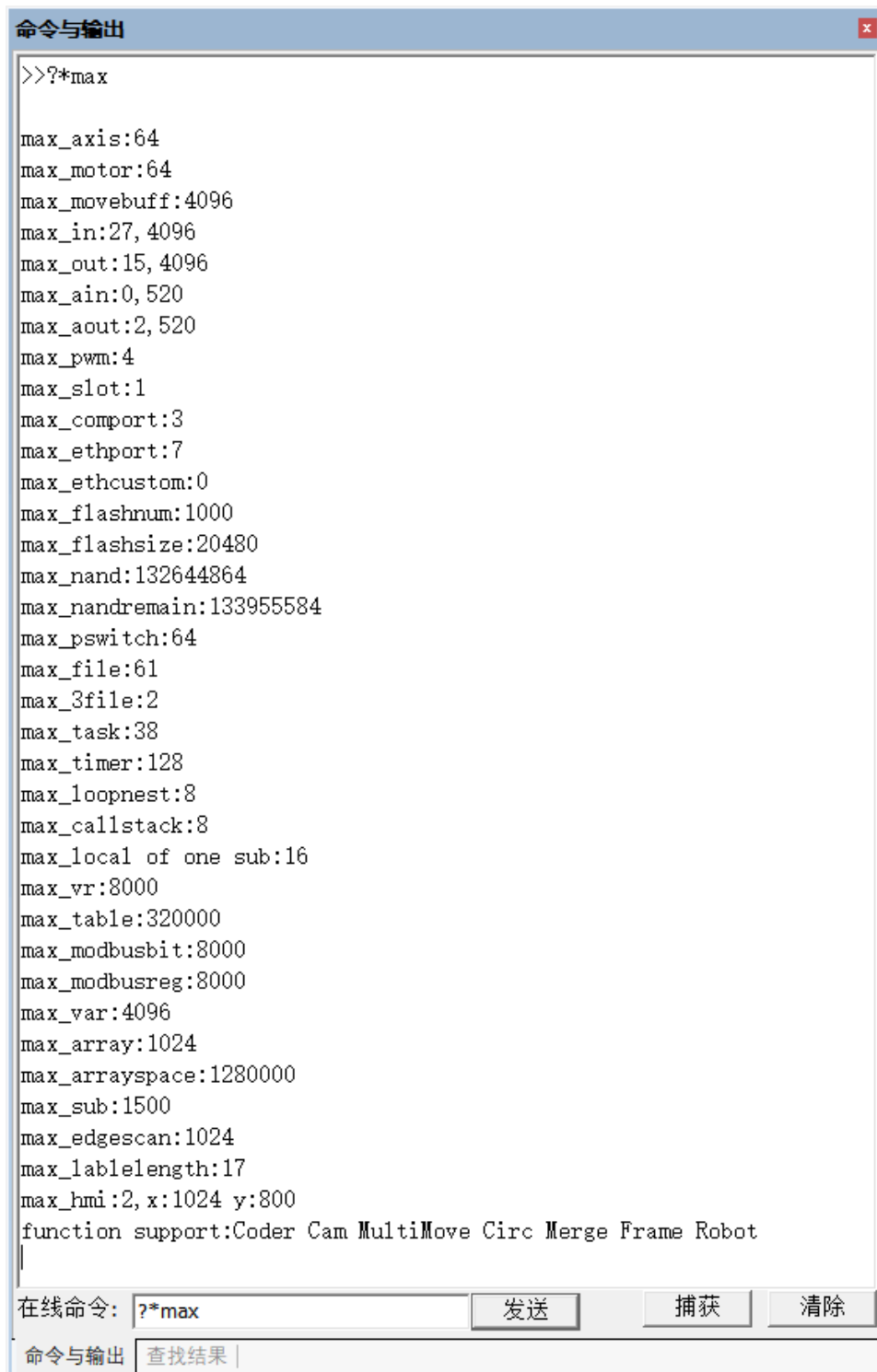
?\*RTEX: 打印 Rtex 总线连接设置状态

?\*FRAME: 打印机械手参数，需要 161022 及以上固件支持

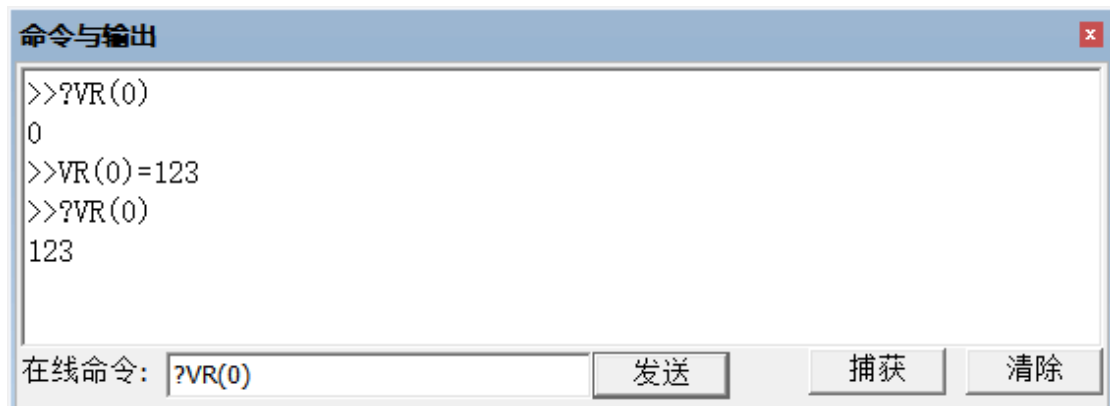
?\*SLOT: 打印出控制器槽位口信息（RTEX 口，EtherCAT 口）

?\*PORT: 打印所有 PORT 通讯口

连接控制器后，使用?\*max 打印控制器所有规格参数结果如下：




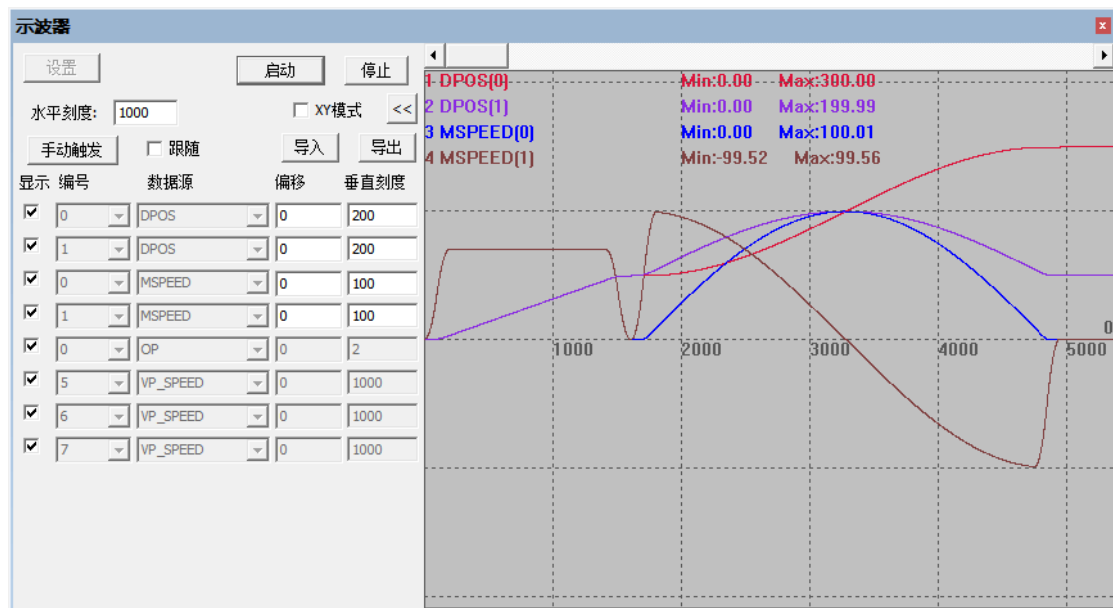
修改变量的值。通过“在线命令”可以实现 VR 变量、TABLE 变量、MODBUS 变量、全局变量、系统设置、轴参数、轴状态变量的设置与修改。下图以修改 VR 变量值为例。



## 2.4. 示波器的使用

### 2.4.1. 示波器面板介绍

示波器属于程序调试与运行中极其重要的一个部分，用于把肉眼看不到的信号转换成图形，便于研究各种信号变化过程。示波器利用控制器内部处理的数据，把数据显示成图形，利用示波器可以显示各种不同的信号，如轴参数、轴状态等。在“视图”→“示波器”中，打开示波器窗口或点击快捷按钮 。



示波器必须先启动后触发才能成功采样。打开示波器，设置好相关参数之后点击“启动”，可手动触发采样，也可在程序里加入“TRIGGER”指令自动触发示波器采样。

示波器基础设置按钮功能：

设置：打开示波器设置窗口，设置示波器相关参数。

启动：启动示波器(但不触发示波器采样)。

停止：停止示波器采样。

XY 模式：勾选时切换到 XY 平面显示两个轴的插补合成轨迹。

<<：按下隐藏通道名称和峰值，只显示通道编号。

手动触发：手动触发示波器采样按钮（自动触发使用 TRIGGER 指令）。

跟随：开启跟随后，横轴自动移动到实时采样处，跟随波形显示。

显示：选择当前通道曲线是否显示。

编号：选择需要采集的数据源编号，如：轴号、数字量 IO 编号、模拟量 IO 编号、TABLE 编号、VR 编号、MODBUS 编号等。

数据源：选择采集的数据类型，下拉菜单选择，多种类型参数可选。

偏移：波形纵轴偏移量设置。

垂直刻度：纵轴一格的刻度。

水平刻度：横轴一格的刻度。

若要设置示波器参数，如轴编号、数据源以及启动示波器设置窗口，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。



示波器设置对话框，包含以下配置项：

参数	当前值/选项	通道颜色
通道数	4	通道1: 红色
深度	30000	通道2: 紫色
间隔	1	通道3: 青色
<input checked="" type="checkbox"/> 自动使用TABLE数组末尾		通道4: 绿色
TABLE位置	200000	通道5: 黄色
背景颜色	灰色	通道6: 棕色
显示类型	点	通道7: 橙色
<input type="checkbox"/> 连续采集		通道8: 品红
<input checked="" type="checkbox"/> 导出参数		

底部按钮：确定、取消

通道数：要采样的通道总数。

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，单位时间内数据量越大。

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，也可以自定义配置，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的数据。

连续采集：不开启连续采集时，到达采样深度后便停止采样，开启了连续采集之后示波器会持续采样。

导出参数：需要导出示波器数据时勾选。

## 2.4.2. 示波器数据导入导出

导入：示波器必须在停止状态下才能导入数据，导入成功能将采样波形复现出来。

导入采样数据方法：点击“导入”，选择导入的数据文件为之前从示波器导出的文件类型后打开即可。

导出：导出参数包括示波器参数设置情况，以及各个通道的数据类型和每个采样点数据。

导出采样数据方法：先在设置里勾选“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，导出数据为文本文件。

### 2.4.3. 示波器采样方法

1. 打开工程项目，连接控制器或仿真器，再打开示波器窗口（操作示波器窗口之前需要连接到控制器或仿真器才可以操作）。

2. 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（一般来说自动使用 TABLE 数组末尾空间即可）和采样类型等，设置完成确认保存当前设置。

3. 再选择采样数据编号和数据源，点击“启动”按钮。

4. 将程序下载到控制器运行，程序里需要包含 TRIGGER 自动触发示波器采样指令，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。

若波形精度不高或显示不完整，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上述采样过程。

若需要采样的时间较长，开启“连续采集”功能。

### 2.4.4. 示波器使用注意事项

示波器采样时间计算

例如深度：10000，间隔：5

如果系统周期 SERVO\_PERIOD=1000，也就是 1ms 轨迹规划周期，间隔 5 表示每 5ms 采集一个数据点，一共采集 10000 次数据，采集时间长度为 50s。

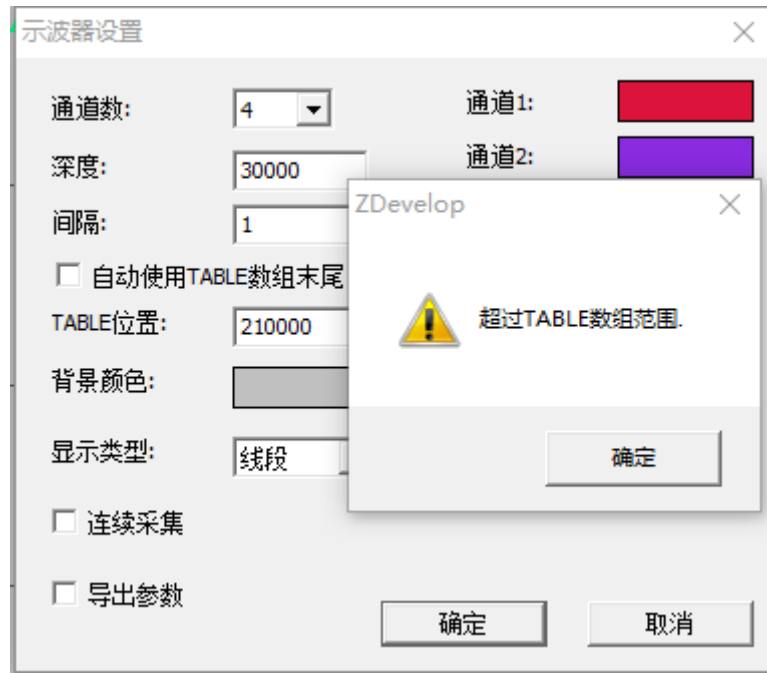
TABLE 数据末尾存储空间计算：

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，此时根据采样数据占用空间大小自动计算起始空间地址。

计算方法：采样数据占用空间大小=通道数\*深度

例：若控制器的 TABLE 空间大小为 320000，采样 4 个通道，深度为 30000，每个采样点占用一个 TABLE，所以会占用  $4 \times 30000 = 120000$  个 TABLE 位置， $320000 - 120000 = 200000$ ，此时 TABLE 的起始位置为 200000。

数据存放的位置也可以自定义配置，若按上面的通道数和深度，起始 TABLE 空间自定义时不能超过 200000，否则无法设置，如下图。



示波器采样数据占用的空间不要与程序使用的 TABLE 数据区域重合。

控制器 TABLE 空间大小可使用 TSIZE 指令读取、在“控制器状态”窗口查看或在线命令?\*max 打印查看。

连续采集功能：

不选择连续采集时，到达采样深度后示波器自动停止采样。

在示波器“设置”里勾选连续采集，再开启示波器，示波器触发采样后会持续采样，到达采样深度后仍继续采样，忽略设置的采样深度，直到按下停止才会停止采样。

连续采集的所有波形采样数据均能导出。

## 2.4.5. 示波器使用例程

例一：连续轨迹前瞻应用

RAPIDSTOP(2)

WAIT IDLE(0)

WAIT IDLE(1)

BASE(0,1)

DPOS=0,0

ATYPE=1,1

UNITS=100,100

SPEED=100,100

ACCEL=1000,1000

DECEL=1000,1000

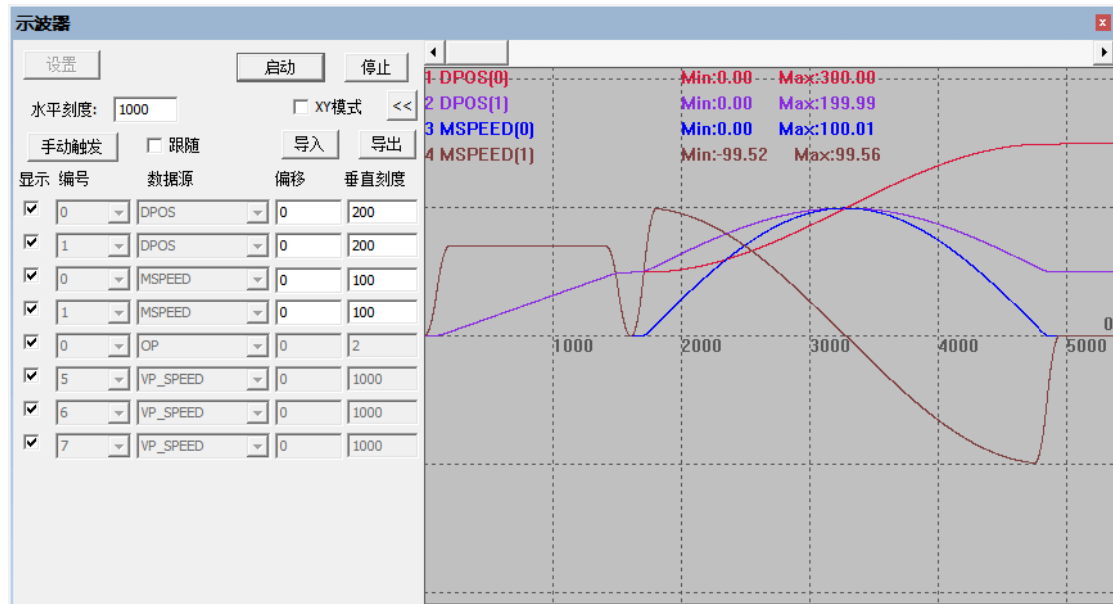
SRAMP=100,100

MERGE=ON

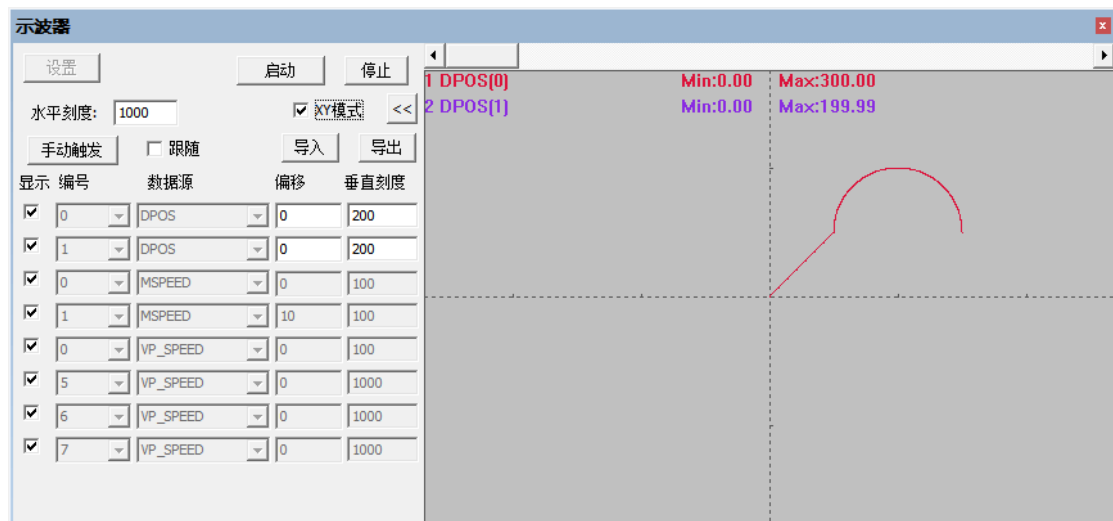
CORNER\_MODE=2

'启动拐角减速

$\text{DECEL\_ANGLE} = 15 * (\text{PI}/180)$  '设置开始减速角度  
 $\text{STOP\_ANGLE} = 45 * (\text{PI}/180)$  '设置结束减速角度  
 $\text{FORCE\_SPEED}=100$  '等比减速时起作用  
 $\text{TRIGGER}$  '自动触发示波器  
 $\text{MOVE}(100,100)$   
 $\text{MOVECIRC}(200,0,100,0,1)$  '半径 100 顺时针画半圆，终点坐标 (300,100)  
 示波器采样轴 0 和轴 1 的速度和位置曲线：



XY 模式下的二轴插补合成轨迹：



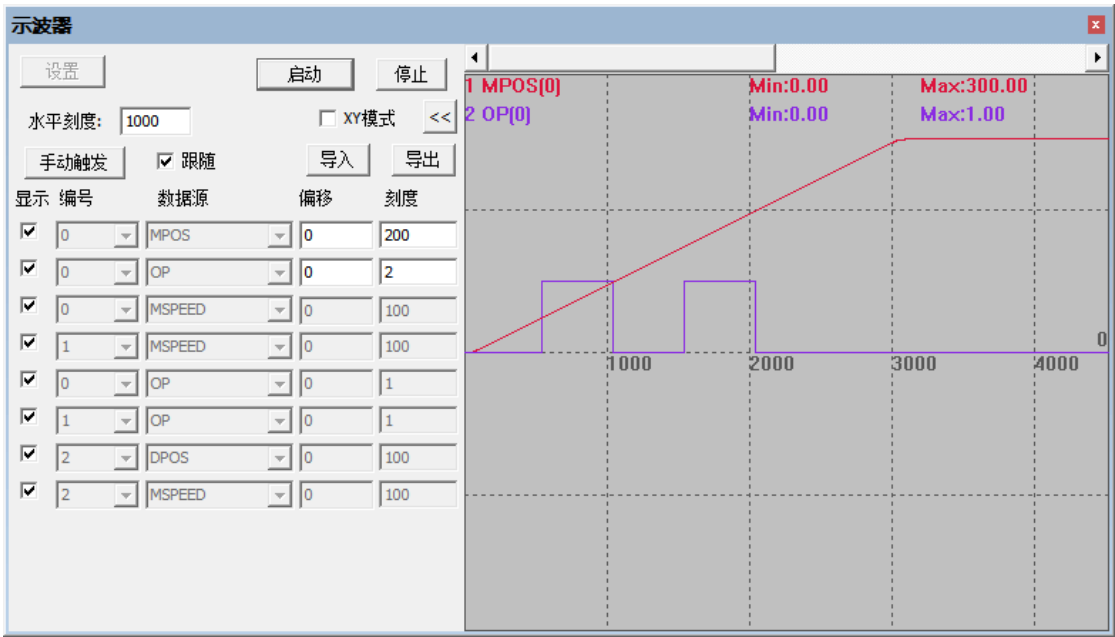
例二：PSO 位置同步输出,到达比较点输出 OP 信号。





```
RAPIDSTOP(2)
WAIT IDLE(0)

BASE(0)
DPOS=0
MPOS=0
ATYPE=1
UNITS=100
SPEED=100
ACCEL=1000
DECEL=1000
OP(0,OFF)
TABLE(0,50,100,150,200)      '比较点坐标
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                        '自动触发示波器采样
MOVE(300)
```



例三：电子凸轮的应用

RAPIDSTOP(2)

WAIT IDLE(0)

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

'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)) '计算每小段位移

TABLE(deg/stepdeg,X) '存储 TABLE

TRACE deg/stepdeg,X

NEXT deg

TRIGGER '触发示波器采样

WHILE 1 '循环运动

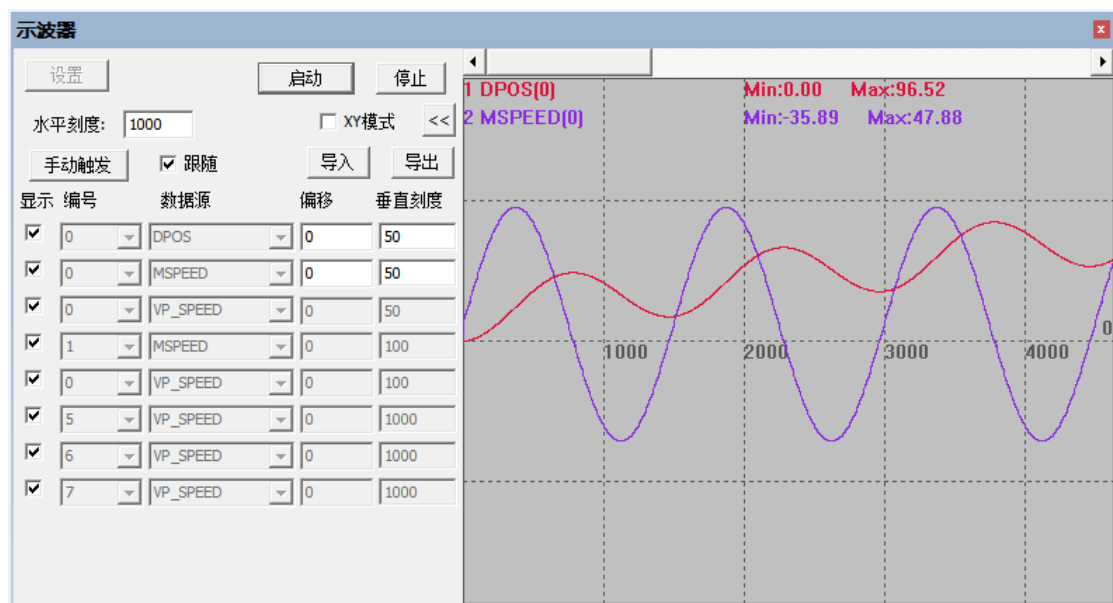
CAM(0, 360/stepdeg, 0.1, 300) '虚拟跟踪总长度 300

WAIT UNTIL IDLE '等待运动停止

WEND

END

运动轨迹：每个凸轮指令运动总时间=distance/speed=300/200=1.5s



## 2.5. 程序调试

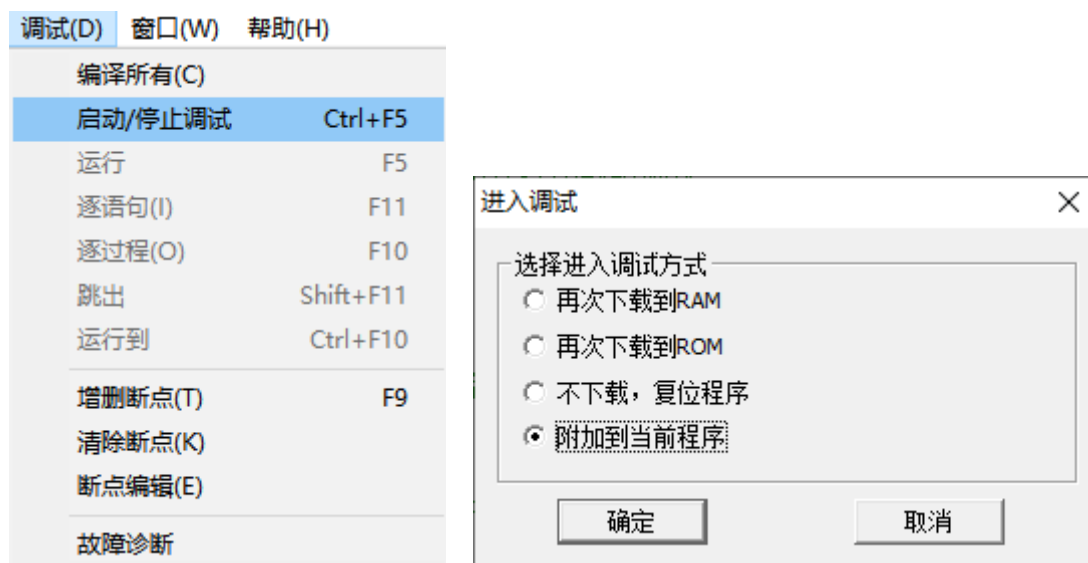
### 2.5.1. 进入程序调试



**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。**

调试功能可以快速调试程序，查看程序中各任务的运行情况。

ZDevelop 连接控制器后，从菜单栏选择“调试”→“启动/停止调试”弹出下方窗口。



进入调试有以下四种方式：

再次下载到 RAM：表示程序再次下载到 RAM 运行，RAM 掉电不保存。

再次下载到 ROM：表示程序再次下载到 ROM 运行，ROM 掉电保存。

不下载，复位程序：表示不下载程序，重新运行之前下载的程序，并打开任务窗口显示目前的运行状态。

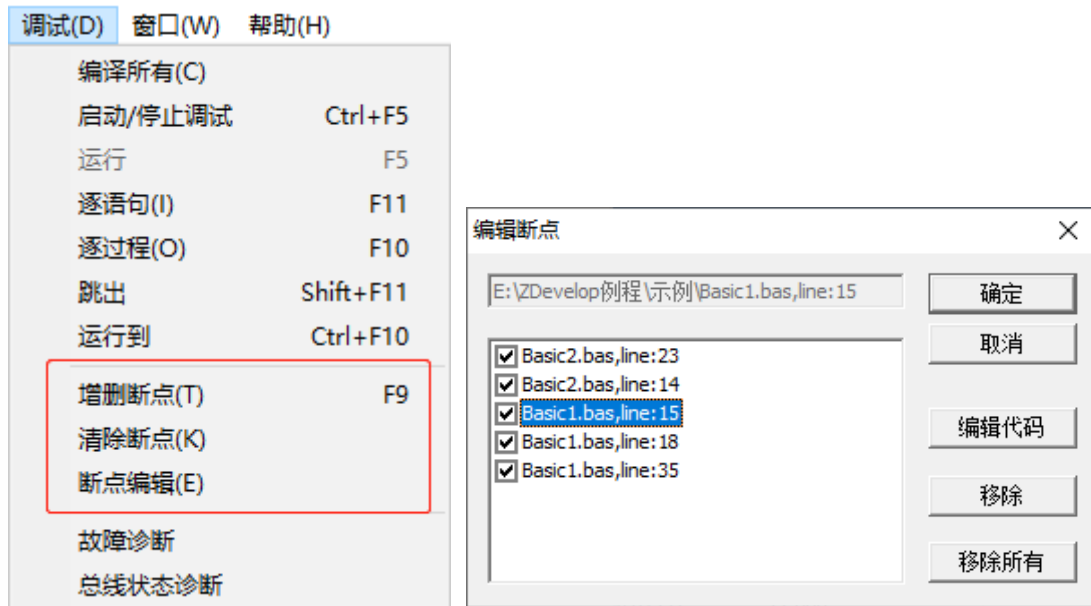
附加到当前程序：表示此时程序不下载，仅打开任务窗口显示目前的运行状态。

### 2.5.2. 任务与监控窗口

选择进入调试的方式后，即可打开任务与监视窗口。

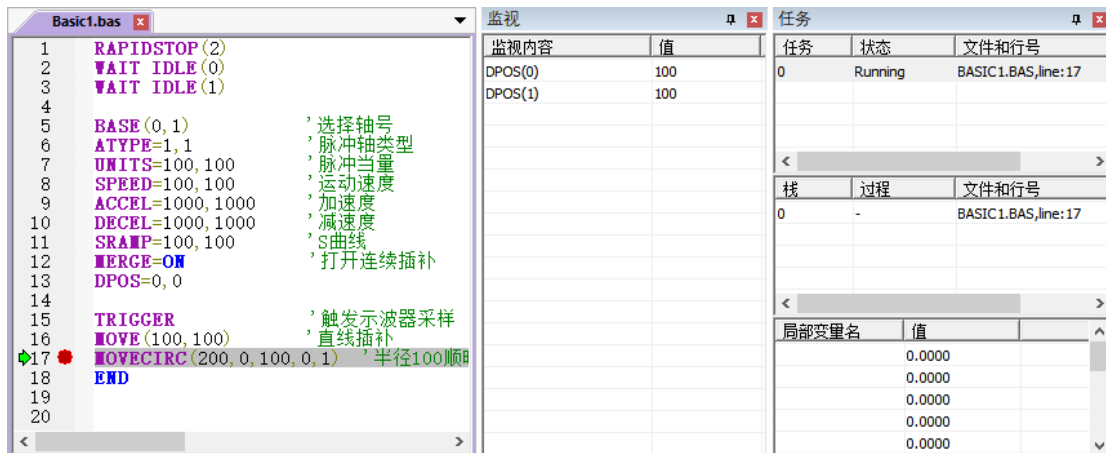


断点快捷键 F9 添加、增删断点按钮或菜单栏“调试”→“增删断点”，断点可以添加多个，菜单栏“调试”→“清除断点”用于一次性清除项目文件中的所有断点。编辑断点窗口可快速移除目标断点或定位到断点处编辑代码。



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，调试的光标停在第 17 行，此时第 17 行的语句并未执行，第 16 行的语句已执行，按下 F11 一次才能执行第 17 行。



如果断点是设置在循环中，那么下次循环运行到断点处还是会停止程序。

程序调试完成后，需要清除所有断点再下载程序到控制器运行。否则打印信息提示 Warn file:"Basic1.BAS" line:17 task:0, Paused. 断点后的程序暂不扫描。

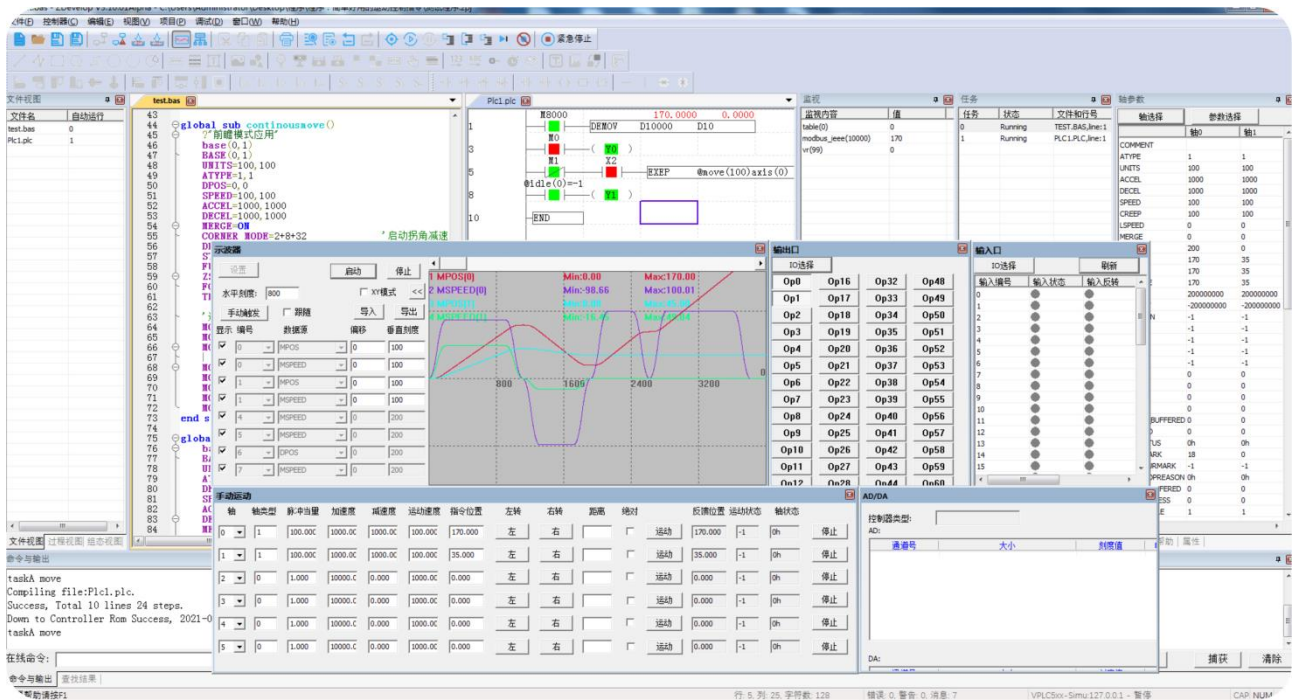
程序在运行途中出现 warn 警告，仍可以继续运行，程序下载后打印 ERROR 错误会停止运行。

## 2.6. 视图窗口

ZDevelop 软件有多种视图窗口，用户能够很容易的对控制器进行程序编辑与配置，快速开发应用程序、实时监控轴运行参数以及对运动控制器正在运行的程序进行实时调试。

例如，轴参数窗口可以监控运动控制中常见的参数，可读写的轴参数在窗口内双击后直接修改，只读参数不支持修改；输入输出窗口监控 IO 的状态，手动运动窗口快速调试轴的运行状态。

更多视图窗口及其功能描述参见 ZDevelop 软件帮助菜单，打开“ZDevelop 使用手册”查看。



## 第三章 Basic 编程基础

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

### 3.1. 编程基础知识

#### 3.1.1. 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。

#### 1. 常见程序结构

为编写算法，我们一般要用到以下几种程序结构描述方式：顺序、选择、循环、延时、等待和子程序调用，子程序调用参见下节。

##### 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
END
```

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
END
```

### 4. 延时

程序遇到 DELAY 延时语句，会停止对应时间后，再继续向下执行。

例程:

```
PRINT 1
DELAY(2000) '延时 2000ms
PRINT 2     '打印 1 延时 2000ms 后打印 2
```



END

## 5. 等待

程序遇到 WAIT 等待语句时，会停止在此行，直到 WAIT 条件满足，才继续向下执行。

例程

BASE(0,1)

MOVE(100,100)

WAIT IDLE           '等待当前插补运动结束

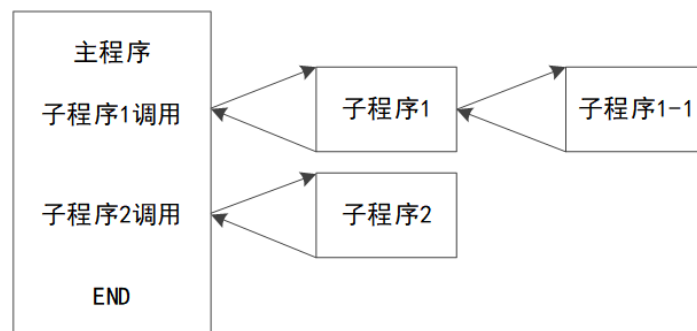
PRINT "运动完成"

程序除了在执行到 WAIT 等待语句，DELAY 延时语句时会阻塞，另外在扫描到运动指令时，如果轴的运动缓冲区满了，此时程序会停在当前运动指令行，直到当前运动完成，缓冲区空出一条，程序就会继续往下执行。缓冲区查看运动缓冲说明。

## 2. 子程序

编程过程中会经常应用到子程序（Basic 中使用 SUB 指令定义子程序），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：



SUB 子程序可作为一个子程序开启，运行完 END SUB 返回主程序，也可以使用任务指令 RUNTASK 开启，作为一个任务单独运行，作为而任务开启后与主程序无关，运行完成后子程序任务结束，不返回主程序。

**主程序调用子程序嵌套最多 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。

示例：

SUB sub1()           '定义过程 SUB1，只能在当前文件中使用

  ?1

  ...

END SUB           '自定义 SUB 过程结束

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                '自定义 SUB 过程结束

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                    '自定义 SUB 过程结束
```

### 3.1.2. 数据相关

#### 1. 数据定义

##### 1. 变量定义

变量是用户可以自定义的参数，变量用于暂时保存与外部设备的通信数据或任务内部处理需要的数据，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。

变量定义指令：分为全局变量（GLOBAL）、文件模块变量（DIM）、局部变量（LOCAL）三种。

全局变量（GLOBAL）：可以在项目内的任意文件中使用；

文件模块变量（DIM）：只能在本程序文件内部使用；

局部变量（LOCAL）：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。

示例：

```
GLOBAL g_var2          '定义全局变量 g_var2
DIM VAR1                '定义文件变量 VAR1
```

```
SUB aaa()
    LOCAL v1            '定义局部变量 V1
    v1=100
END SUB
```

##### 2. 常量定义

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，常量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。

示例：

```
CONST MAX_VALUE = 100000    '定义文件常量
GLOBAL CONST MAX_AXIS=6     '定义全局常量
```

##### 3. 数组定义

数组指定是指将相同属性的数据集中后对其进行统一定义，并对数据个数进行指定。构成数组的各数据称为“元素”。

数组定义相关指令为 GLOBAL、DIM，不支持 LOCAL 定义。

数组定义时注意数组空间大小的指定，不能使用超出定义范围的空间，否则程序报错数组空间超限。

示例：

DIM array1(15) '定义文件数组，可使用的数组空间编号为 0~14，共 15 个空间

GLOBAL array2(10) '定义全局数组，可使用的数组空间编号为 0~9，共 10 个空间

## 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。各种运动指令的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 <sup>32</sup> -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

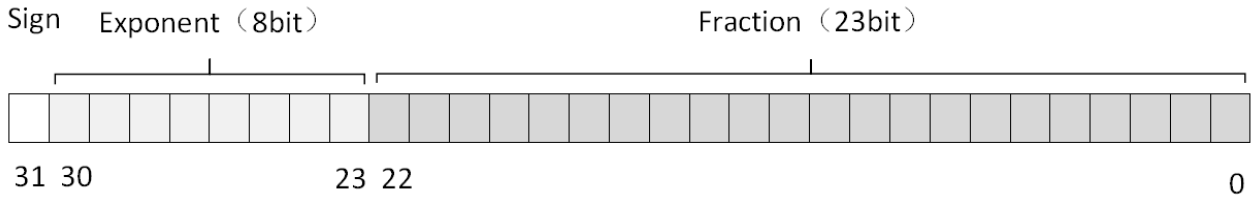
指令的输入或输出变量的数据类型由指令确定。

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。

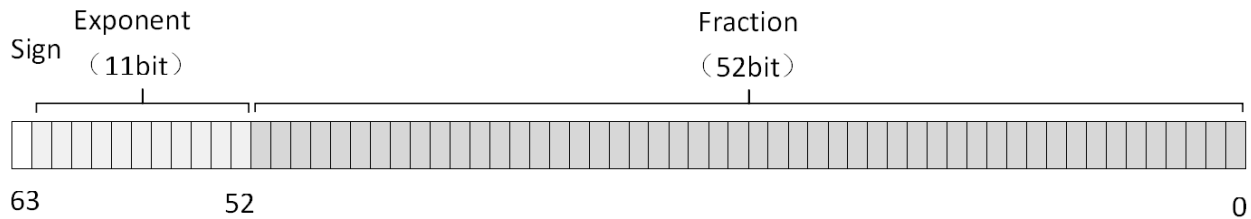
单精度浮点数 32 位，如下图。

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
<a href="#">MODBUS_BIT</a>	布尔型	0 或 1
<a href="#">MODBUS_REG</a>	16 位整型	-32768 到 32767
<a href="#">MODBUS_LONG</a>	32 位整型	-2147483648 到 2147483647
<a href="#">VR_INT</a>		
<a href="#">MODBUS_IEEE</a>	32 位浮点型	-3.4028235E+38 到 -1.401298E-45
<a href="#">VR</a>		
<a href="#">TABLE</a> ，自定义数组，变量 (ZMC3 系列及之前)		
<a href="#">TABLE</a> ，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
<a href="#">VRSTRING</a>	字符	一个字符占一个 VR
<a href="#">MODBUS_STRING</a>	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。

### 3. 数据操作

不同类型数据之间的操作要注意数据类型，类型不匹配会产生下列问题：

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
<a href="#">DIM</a>	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
<a href="#">CHR</a>	把 ASCII 转成一个字符串常量，此字符串只有一个字节
<a href="#">MODBUS_STRING</a>	标准 MODBUS 协议定义字符串，每个 16 位寄存器存储 2 个字节
<a href="#">VRSTRING</a>	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
<a href="#">VAL</a>	数字字符串转换为数值
<a href="#">TOSTR</a>	数值转换为字符串
<a href="#">STRCOMP</a>	字符串比较函数
<a href="#">DMCPY</a>	数组拷贝函数，可以用于拷贝字符串
<a href="#">HEX</a>	返回为十六进制格式，只能用于打印
<a href="#">DATE\$</a>	按 dd: mm: yyyy 格式返回 DATE 设置日期
<a href="#">DAY\$</a>	返回本日的星期英文名
<a href="#">TIME\$</a>	按 24 小时的格式 hh: mm: ss 返回当前时间

## 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（轴类型、脉冲当量、轴速度等），以及相关安全设置（正负硬件限位、正负软件限位、报警信号、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：轴参数指令、[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。

## 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 FLASH 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，因为掉电执行时间非常短，建议只存几个数据。

使用 SETCOM 指令可以把 VR 与 MODBUS\_REG 寄存器匹配起来，设置指令参数 variable，详细设置参见 [SETCOM](#) 指令。

例程：设置 variable=3，将一个 VR\_INT 映射到两个 MODBUS\_REG 地址，换算关系：VR\_INT(num)=MODBUS\_REG(num)\*2^16+MODBUS\_REG(num+1)

```
SETCOM(38400,8,1,0,0,3)      '配置掉电存储
```

```
VR_INT(0)=0
```

```
MODBUS_REG(0)=1             '低 16 位值为 1
```

```
MODBUS_REG(1)=2             '高 16 位值为 2
```

```
?VR_INT(0)                  '结果：131073
```

```
END
```

VR 是掉电非易失的，可以无限次的读写，数据保存时间为 10 年，机器设备关键参数建议存储到 FLASH，FLASH 空间更大，上电时从 FLASH 读出数据写至各变量中。

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

## 3.2. ZDevelop 的三种编程方式

### 3.2.1. 混合编程

ZDevelop 软件可以支持三种编程方式，分别为 ZBASIC、ZPLC 梯形图和 ZHMI 组态编程，支持三种语言混合编程，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	文件名	自动运行
Basic1.bas	0	Plc1.plc	0
Basic2.bas	1	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，其他公司的示教盒也能连接到控制器上，请使用该公司提供的示教盒编程软件。

### 3.2.2. PLC 与 BASIC 互相调用

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](#)。

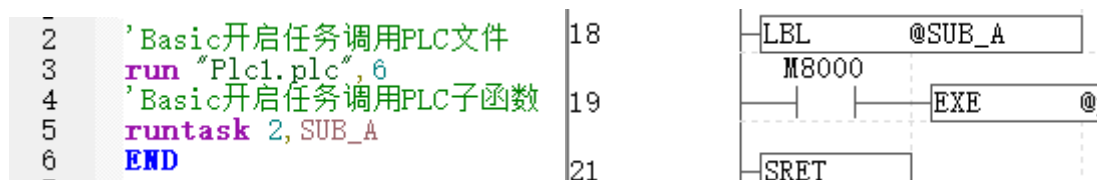
特殊继电器 D 对应 [MODBUS\\_REG](#)。

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”。



### 3.3. 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，不同的控制器存储空间大小不同。

### 3.3.1. 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。

1) TABLE 指令读写数据。

TABLE(0) = 10                      'TABLE(0)赋值 10

TABLE(10,100,200,300)        '批量赋值, TABLE(10)赋值 100, TABLE(11)赋值 200, TABLE(12)赋值 300

2) [TSIZE](#) 指令可读取 TABLE 空间大小，还可修改 TABLE 空间大小（不能超出 TABLE 最大空间）。

PRINT TSIZE                      '打印出控制器 TABLE 大小

TSIZE=10000                      '设置 TABLE 的大小，不能超过控制器 TABLE 最大 SIZE

3) [TABLESTRING](#) 指令按照字符串格式打印 TABLE 里的数据。

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:\

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

CAM(10,17,100,500)                      '运动轨迹为 TABLE(10)到 TABLE(17)

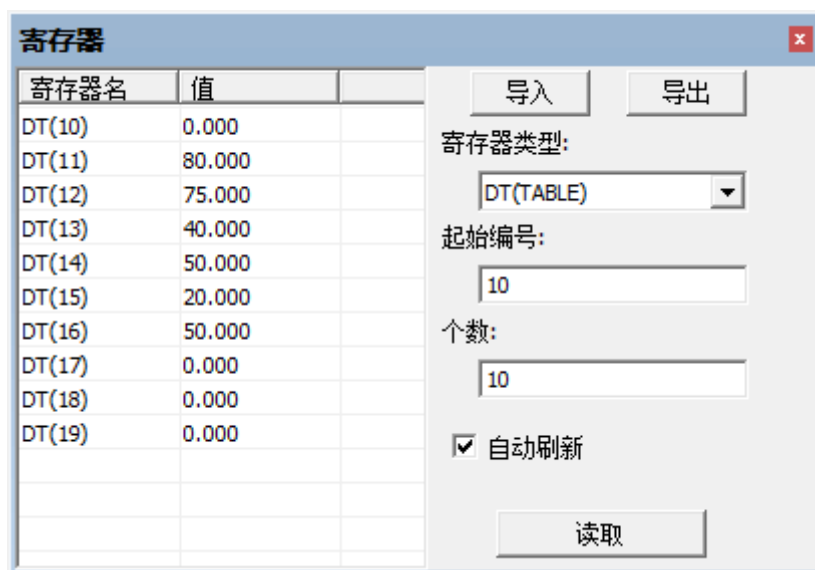
查看 TABLE 内数据的方式有 2 种：

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。



### 3.3.2. 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。

注意：FLASH 在读取之前先要写入，否则会提示警报 WARN。

FLASH 使用方法：

GLOBAL VAR '变量定义

GLOBAL ARRAY1(200) '数组定义

DIM ARRAY2(100)

'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

'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 '读取次序与写入次序一致

### 3.3.3. VR

[VR](#) 寄存器具有掉电存储功能，可无限次读写，但数据空间较小，一般只有 1024 或者更少，最新系列控制器的 VR 空间为 8000，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、VR\_INT、VRSTRING 共用一个空间，地址空间是重叠的，VR 和 VR\_INT 读写方法相同，VRSTRING 保存 ASCII 码，一个字符占用一个 VR。

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 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。

例一：VR 使用方法

```
VR(0) = 10.58           '赋值
aaa = VR(0)             '读取
```

例二：VR 寄存器数据相互转换

```
VR(100)=10.12
VR_INT(100)=VR(100)     '数据转换
?VR_INT(100)            '打印结果：10，浮点数转换成整数，丢失小数位
```

例三：VRSTRING 存储字符串

```
VRSTRING(0,4) = "abc"   '从 VR(0)开始保存字符串
PRINT VRSTRING(0,4)     '打印结果：abc
```

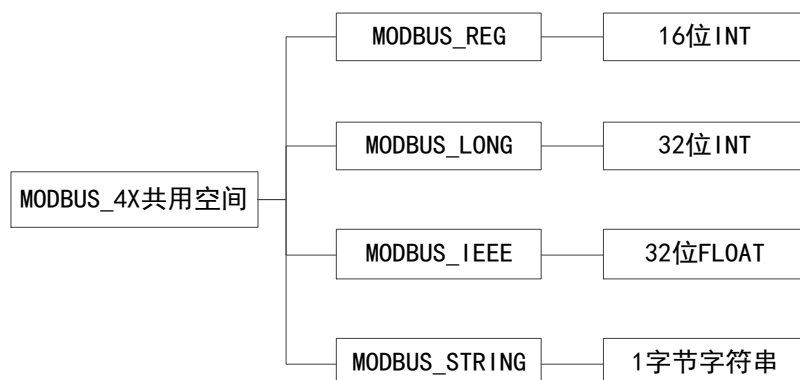


### 3.3.4. MODBUS

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。

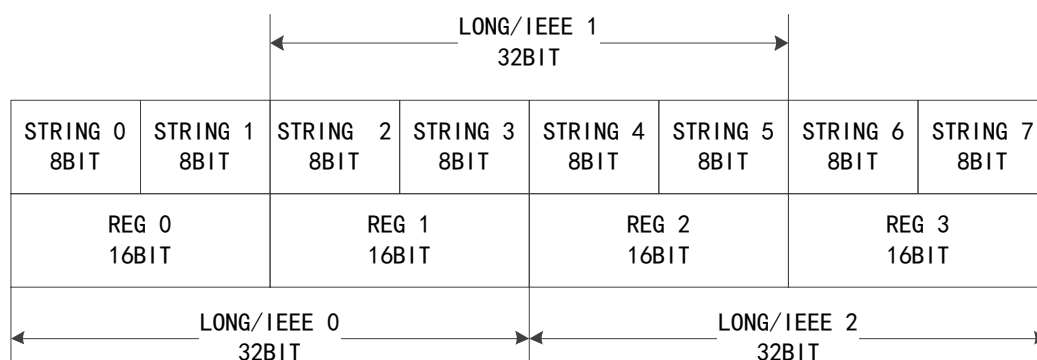
MODBUS\_LONG(1) 占用 MODBUS\_REG(1) 与 MODBUS\_REG(2) 两个 REG 地址。

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(0)=70000        'modbus\_long 赋值 70000，modbus\_reg 范围-32768~32767

?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)

<pre> 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) </pre>	<table border="1"> <thead> <tr> <th>监视内容</th><th>值</th></tr> </thead> <tbody> <tr> <td>MODBUS_LONG(0)</td><td>70000</td></tr> <tr> <td>MODBUS_REG(0)</td><td>4464</td></tr> <tr> <td>MODBUS_REG(1)</td><td>1</td></tr> </tbody> </table>	监视内容	值	MODBUS_LONG(0)	70000	MODBUS_REG(0)	4464	MODBUS_REG(1)	1
监视内容	值								
MODBUS_LONG(0)	70000								
MODBUS_REG(0)	4464								
MODBUS_REG(1)	1								

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 不会有问題, 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 字寄存器地址	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	模拟量输入 AIN，读用 MODBUS_REG

## 3.4. 多任务编程

### 3.4.1. 多任务概念

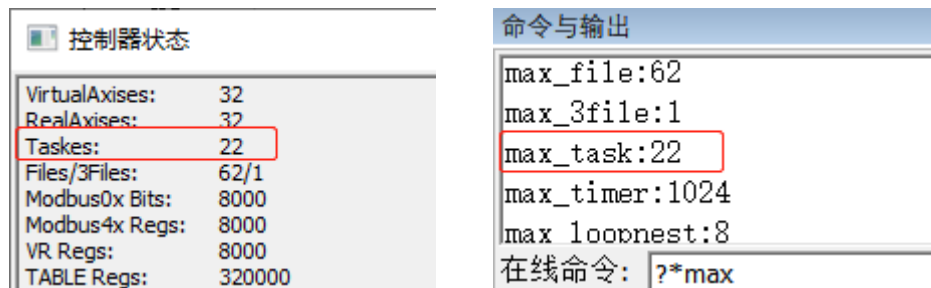
任务是执行 I/O 刷新和用户程序等一系列指令处理的功能，一个任务是指一个正在运行的程序。

如果多个程序模块能够互不干扰的同时运行，则称为多任务，多任务编程在 ZDevelop 软件上实现。

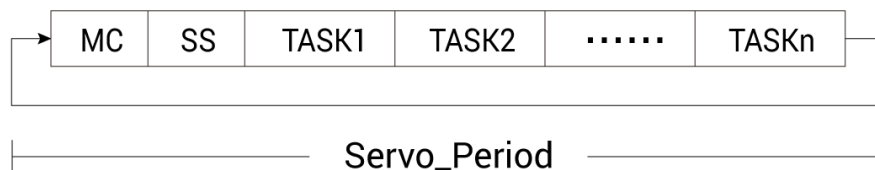
多任务可以将复杂的程序分成几个部分，分别开任务来同时执行，每个部分的任务是独立的，这样就可以使设备的复杂运动过程变得简单明了，编程更灵活，没有多任务的场合程序只能顺序执行，使得程序的执行效率十分低下。

ZMC 运动控制器支持多任务编程，每个任务都有自己唯一的编号，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。

不同型号支持的任务数有所不同，支持的具体任务数量，可连接控制器之后，在 ZDevelop 软件菜单栏“控制器状态”查看或在线命令发送?\*max 指令查看，如下图，表示该控制器支持 22 个任务，任务编号范围为 0-21。



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、EtherCAT 通讯、中断的实现。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）

**TASK1、…、TASKn:** 对应于各个任务的运行，第 1 个任务到第 n 个任务。

在一个控制周期内，不同的任务根据当前执行的指令的差异，任务占用的时间也会有差异，并不完全相同，任务在默认情形下不存在优先级，可通过 PROC\_PRIORITY 指令去设置某个任务的优先级。

Basic 的所有任务只扫描运行一次（除非程序内有死循环才会一直运行）。一个工程项目下 Basic 文件

支持同时存在多个自动运行任务。

PLC 主任务循环扫描执行，PLC 子程序任务只运行一次。一个工程项目下 PLC 文件建议只设置一个自动运行任务号。

HMI 程序需要设置自动运行任务号，初始化函数只扫描执行一次，周期函数循环扫描。一个工程项目下 HMI 文件仅支持一个，组态程序要运行只能通过给 HMI 文件设置自动运行任务号。

文件视图		任务		
文件名	自动运行	任务	状态	文件和行号
Basic1.bas	0	0	Stopped	BASIC1.BAS,line:31
Basic2.bas	1	1	Stopped	BASIC2.BAS,line:18
Basic3.bas		2	Running	PLC1.PLC,line:1
Plc1.plc	2	3	Running	HMI1.HMI,line:1
Hmi1.hmi	3			

控制器同时处理四个任务，如上图，任务 0123 之间是并行的，互不干扰，控制器下载程序之后这四个文件任务同时启动，同时还能在文件任务执行的时候，使用任务指令开启 SUB 子程序任务或标记任务，SUB 子程序任务或标记任务一旦开启，便与主程序无关，任务运行停止后可重复触发任务执行。

控制器多任务的优势：

程序模块化：用户可以将程序编写成多个较小的、特定的程序，来实现客户设备指定的功能。

并发性：每个任务可以独立运行，任务开启后，不受其他任务的影响。

简化错误处理：划分多任务运行后错误处理变得简单，只需处理出错的任务。

命令交互：程序处于运行状态时，用户也可以随时进行命令交互，如在线修改运动参数，在线命令栏发送指令等，其他程序不受影响。

### 3.4.2. 多任务状态查看

任务状态有三种，正在运行、停止和暂停，任务状态查看有如下 3 种方式。

#### 1. 任务指令查看

PROC\_STATUS：任务状态查看，只读参数。返回值：0-任务停止，1-任务正在运行，3-任务暂停中。

示例：

PRINT PROC\_STATUS(0) '打印任务 0 状态

?\*PROC\_STATUS '打印控制器支持的所有任务的状态

#### 2. 任务窗口查看

在菜单栏“调试”→“启动/停止调试”打开任务窗口，如下图。

任务窗口可以查看已经开启的任务的任务编号、运行状态、当前文件和运行行号，看不到未开启的任务。

任务	状态	文件和行号
0	Stopped	BASIC1.BAS,line:15
1	Running	BASIC2.BAS,line:107
2	Paused	BASIC1.BAS,line:26
3	Stopped	PLC1.PLC,line:13
6	Running	PLC1.PLC,line:1

Basic 的任务在程序扫描完成后，任务变为 Stopped 状态，PLC 主任务由于会循环扫描，所以一直处于 Running 状态。

### 3. 打开菜单栏“调试”→“故障诊断”窗口。

可查看控制器支持的所有任务编号的状态、所处的文件和运行的行号。

此窗口也可以显示各任务报错的故障信息。

故障诊断

控制器状态

Power:

Run:

Alm:

控制器信息

软件类型: ZMC432

软件版本: 4.640-2017062

Ip地址: 192.168.0.36

硬件版本: 432-0

控制器ID: 170501269

诊断结果:

Task:0 Running. file:"BASIC1.BAS" line:11:

Task:1 Running. file:"BASIC2.BAS" line:94:

Task:2 Stopped.

Task:3 Stopped.

Task:4 Stopped.

Task:5 Stopped.

Task:6 Running. file:"PLC1.PLC" line:4:

Task:7 Stopped.

Task:8 Stopped.

Task:9 Stopped.

确定

取消

### 3.4.3. 多任务启动与停止

#### 1. 多任务操作指令

多任务主要操作指令如下:

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

PAUSETASK: 暂停指定任务



RESUMETASK: 恢复指定任务，恢复后任务从停止处继续往下执行

Basic 和 PLC 的任务操作均是使用以上指令。

控制器状态和?\*MAX: 可以查看控制器支持的任务总数与文件总数。

控制器状态	命令与输出
VirtualAxes: 32	max_file:62
RealAxes: 32	max_3file:1
Tasks: 22	max_task:22
Files/3Files: 62/1	max_timer:1024
Modbus0x Bits: 8000	在线命令: ?*max
Modbus4x Regs: 8000	
VR Regs: 8000	
TABLE Regs: 320000	

## 2. 多任务启动

任务启动有三种方式，分别是自动运行任务号设置、RUN 指令和 RUNTASK 指令，使用指令开启任务时，程序扫描执行到该指令后再开启任务。

开启任务时注意任务编号的填写，任务不能重复开启。

1) 自动运行任务号: 在“文件视图”窗口设置自动运动任务号。控制器上电后首先执行带自动运行任务号的文件，自动运行任务号 Basic 文件可设置多个，PLC 文件和 HMI 文件仅支持一个。自动运行文件为并行运行，上电后同时开启。

2) RUN 指令将文件作为一个任务启动。

示例: RUN "TuXing\_001.bas",2     '将 TuXing\_001.bas 文件作为任务 2 启动

3) RUNTASK 指令将 SUB 子程序或带标签程序作为一个任务启动。可跨文件开启全局定义的 SUB 子程序，要开启任务的标签程序只能存在本文件内。

示例: RUNTASK 1,task\_home     '以任务 1 启动 task\_home 子程序

## 3. 多任务停止

停止任务指令有 STOPTASK, STOP, HALT 三种。

任务停止再启动就会从头执行任务。

开启任务时，一般先使用 STOPTASK 停止任务，再 RUNTASK 开启，避免任务出现重复开启报错。

1) STOPTASK 支持停止文件任务、SUB 子程序任务和带标签的任务。

示例: STOPTASK 2     '停止任务 2

2) STOP 指令支持停止 Basic 文件任务，推荐使用 STOPTASK 指令，操作更简单。

3) HALT 指令停止所有任务

示例: HALT     '停止项目内的所有任务

快速停止所有任务还可以使用软件菜单栏的“紧急停止”按钮。

示例: 项目内有两个任务，下载运行后，任务 0 和任务 1 正在运行中。



文件视图		任务		
文件名	自动运行	任务	状态	文件和行号
Basic11.bas	0	0	Running	BASIC11.BAS,line:45
		1	Running	BASIC11.BAS,line:82

发送在线命令: STOPTASK 0

停止任务 0。

文件视图		任务		
文件名	自动运行	任务	状态	文件和行号
Basic11.bas	0	0	Stopped	BASIC11.BAS,line:57
		1	Running	BASIC11.BAS,line:82

再次启动任务可以再次下载程序。

上程序不能使用 **RUN** 指令开启自动运行文件任务 0，因为任务 0 中自动开启的任务 1 仍在运行，若使用指令再次开启任务 0，会导致任务 1 重复开启。若停止的任务 1，则可以使用 **RUNTASK** 指令单独开启任务 1。

### 3.4.4. 任务暂停与恢复

暂停任务使用 **PAUSETASK** 指令，恢复任务使用 **RESUMETASK** 指令。恢复后任务从暂停处继续向下执行。暂停的任务支持停止。

1) PAUSETASK: 暂停指定任务

示例：PAUSETASK 1 '暂停任务 1'

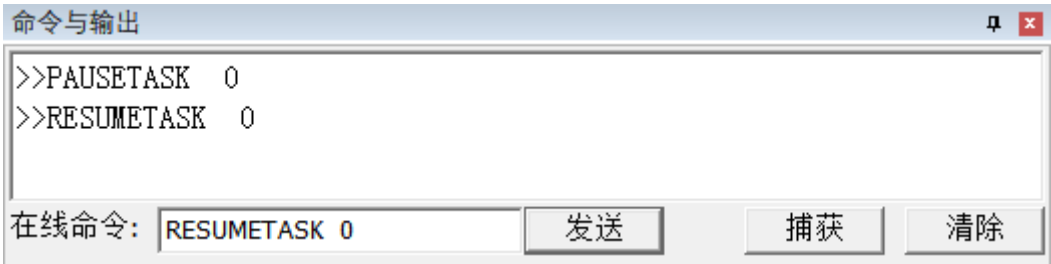
## 2) RESUMETASK: 恢复指定任务

示例: RESUMETASK 1 '继续运行任务 1'

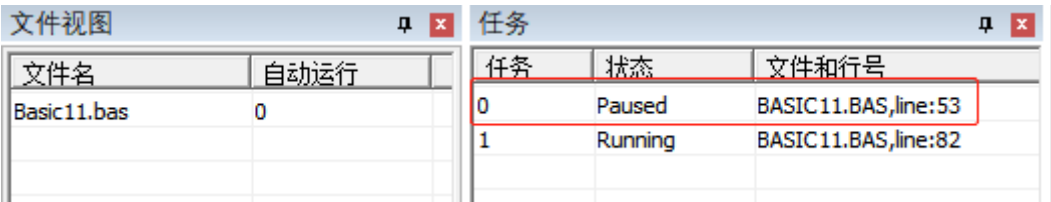
示例：项目内有两个任务，下载运行后，任务 0 和任务 1 正在运行中。

文件视图		任务		
文件名	自动运行	任务	状态	文件和行号
Basic11.bas	0	0	Running	BASIC11.BAS,line:45
		1	Running	BASIC11.BAS,line:82

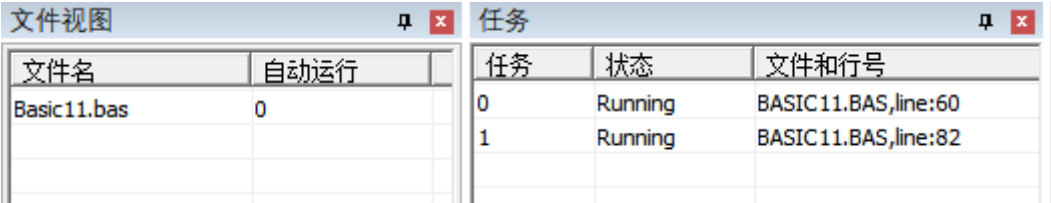
发送在线命令控制任务的暂停或恢复。



在线命令发送：PAUSETASK 0  
任务 0 被暂停。



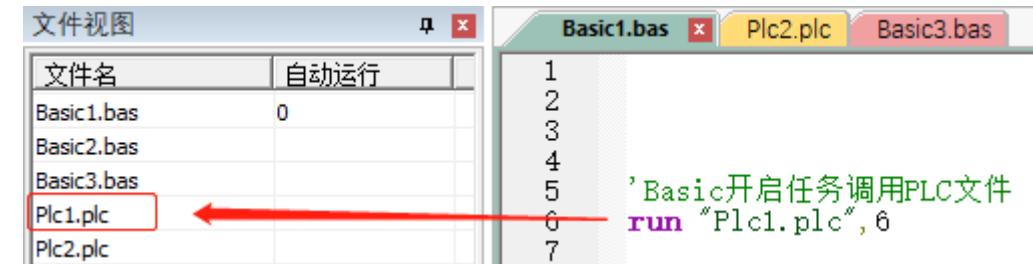
在线命令发送：RESUMETASK 0  
任务 0 恢复运行状态。



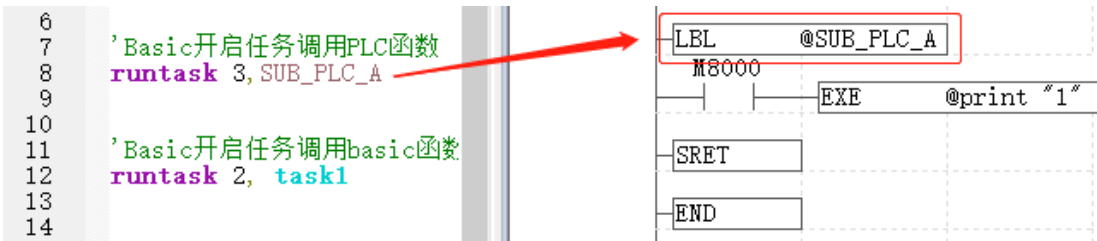
3.4.5. Basic 和 PLC 任务相互调用

1. Basic 调用 PLC 任务

1) Basic 文件内使用 RUN 指令调用 PLC 文件。



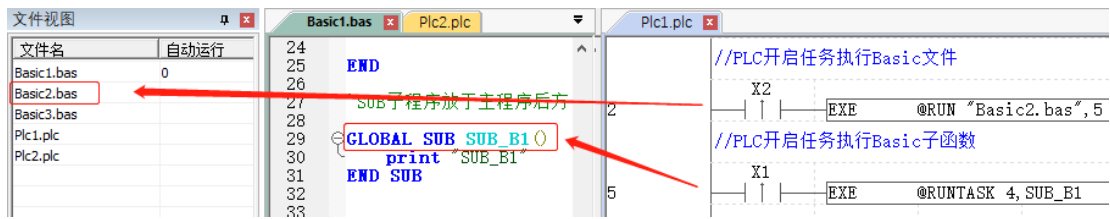
2) Basic 文件内使用 RUNTASK 指令调用 PLC 内 LBL 指令定义的子程序。



2. PLC 调用 Basic 任务

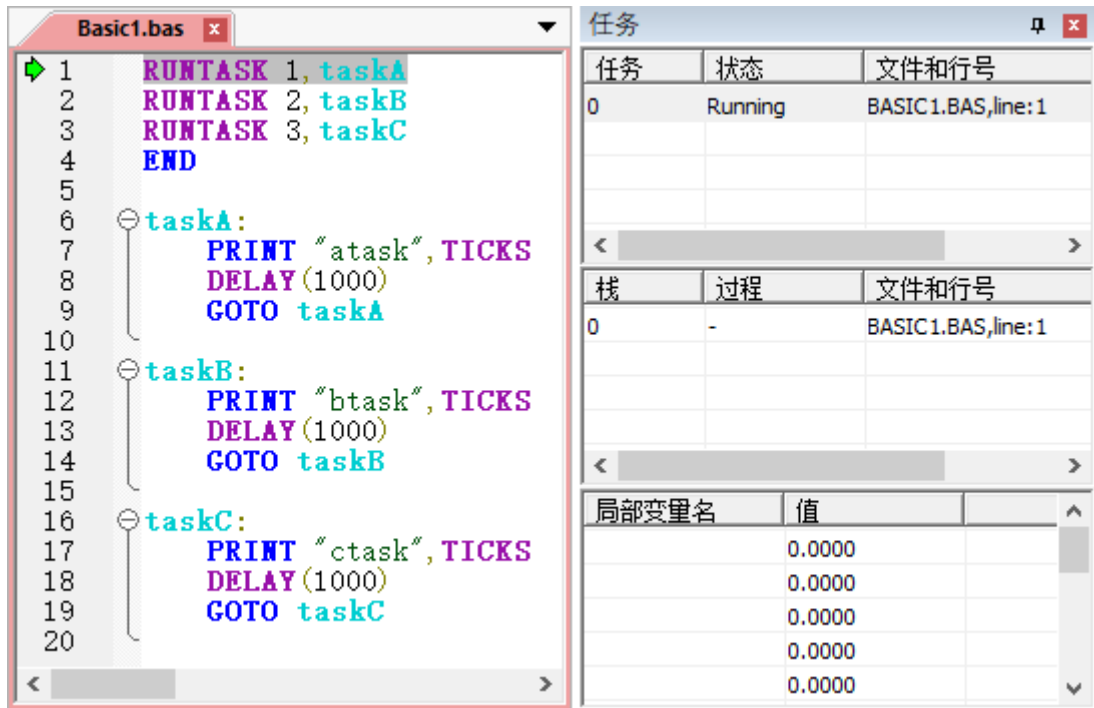
PLC 内使用 EXE 或 EXEP(脉冲执行)指令，调用 Basic 的任务指令，从而调用 Basic 文件任务或子程序

任务。



### 3.4.6. 多任务示例

如下例程，开始单步调试，观察左侧光标的指向，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，遇到 END 主任务 0 停止扫描，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

**Basic1.bas**

```

1  RUNTASK 1, taskA
2  RUNTASK 2, taskB
3  RUNTASK 3, taskC
4  END
5
6  taskA:
7      PRINT "atask", TICKS
8      DELAY(1000)
9      GOTO taskA
10
11 taskB:
12     PRINT "btask", TICKS
13     DELAY(1000)
14     GOTO taskB
15
16 taskC:
17     PRINT "ctask", TICKS
18     DELAY(1000)
19     GOTO taskC
20

```

**任务**

任务	状态	文件和行号
0	Running	BASIC1.BAS,line:2
1	Running	BASIC1.BAS,line:7

**栈**

栈	过程	文件和行号
0	-	BASIC1.BAS,line:2

**局部变量名**

局部变量名	值
	0.0000
	0.0000
	0.0000
	0.0000
	0.0000

任务 0 启动任务 1 运行

**Basic1.bas**

```

1  RUNTASK 1, taskA
2  RUNTASK 2, taskB
3  RUNTASK 3, taskC
4  END
5
6  taskA:
7      PRINT "atask", TICKS
8      DELAY(1000)
9      GOTO taskA
10
11 taskB:
12     PRINT "btask", TICKS
13     DELAY(1000)
14     GOTO taskB
15
16 taskC:
17     PRINT "ctask", TICKS
18     DELAY(1000)
19     GOTO taskC
20

```

**任务**

任务	状态	文件和行号
0	Running	BASIC1.BAS,line:3
1	Running	BASIC1.BAS,line:8
2	Running	BASIC1.BAS,line:12

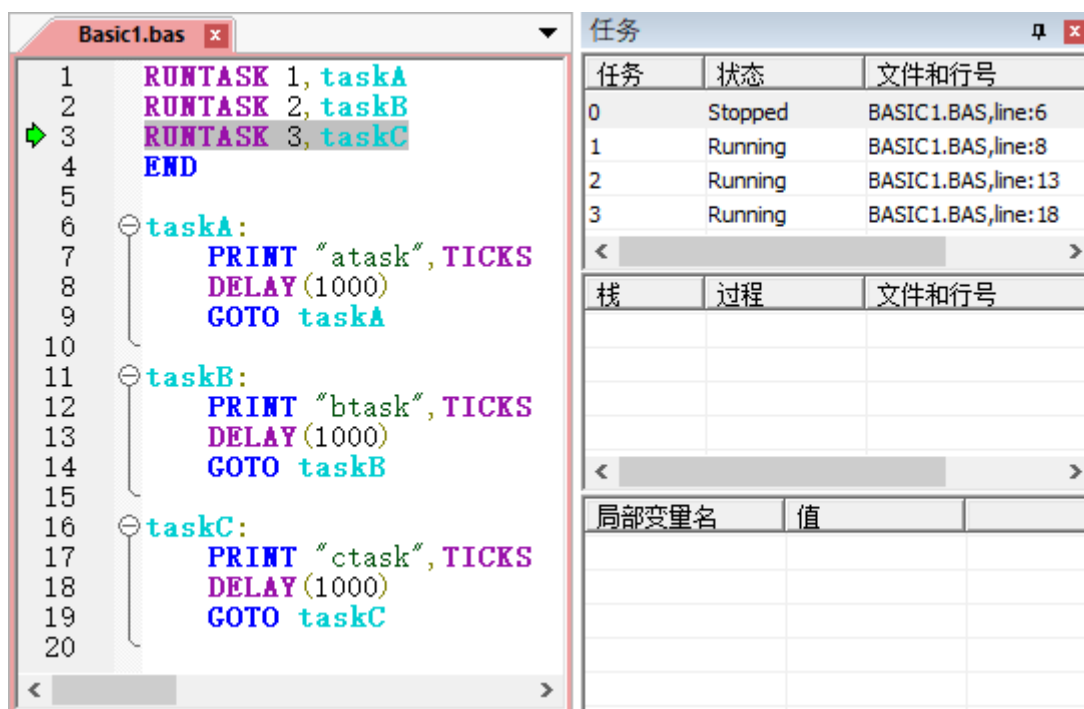
**栈**

栈	过程	文件和行号
0	-	BASIC1.BAS,line:3

**局部变量名**

局部变量名	值
	0.0000
	0.0000
	0.0000
	0.0000
	0.0000

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

## 3.5. 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。

三类中断运行时，中断程序单独占用一个任务号运行，不存在压栈的情况。

### 中断使用注意事项：

各中断之间无优先级，支持中断嵌套，多个中断可以同时执行，同一时间处理的中断函数不宜过多。

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：

尽量减少中断的数量，很多应用都可以用循环扫描来处理。

如果有一个中断处理函数特别长的话，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。

### 3.5.1. 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](#)。

示例：

```
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                    '主程序结束
```

```
GLOBAL SUB ONPOWEROFF () '掉电中断
    VR(0) = DPOS(0)      '保存坐标到 VR
    VR(1) = DPOS(1)
    VR(2) = DPOS(2)
END SUB
```

### 3.5.2. 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](#)。

示例：

```
INT_ENABLE=1          '开启中断
END                  '主程序结束
```

```
GLOBAL SUB INT_ON0 () '外部上升沿中断程序
    PRINT "输入 IN0 上升沿触发"
END SUB
```

```
GLOBAL SUB INT_OFF0 () '外部下降沿中断程序
    PRINT "输入 IN0 下降沿触发"
END SUB
```

### 3.5.3. 定时器中断

达到设定时间后执行的功能，必须是全局的 SUB 函数定时器中断支持同时开启多个，个数由定时器个数决定，定时器个数根据控制器型号，使用?\*max 打印查看。

相关函数：[ONTIMERn](#)。

示例：

```
INT_ENABLE=1          '开启中断
TIMER_START(0,100)    '定时器 0 开启，100ms 后执行一次
END                  '主程序结束
```

```
GLOBAL SUB ONTIMER0() '中断程序
    PRINT "ontimer0 enter"
    'TIMER_START(0,100) '希望周期执行中断，在 SUB 里再次打开定时器
```

END SUB

## 3.6. 运动缓冲

### 3.6.1. 运动缓冲概念

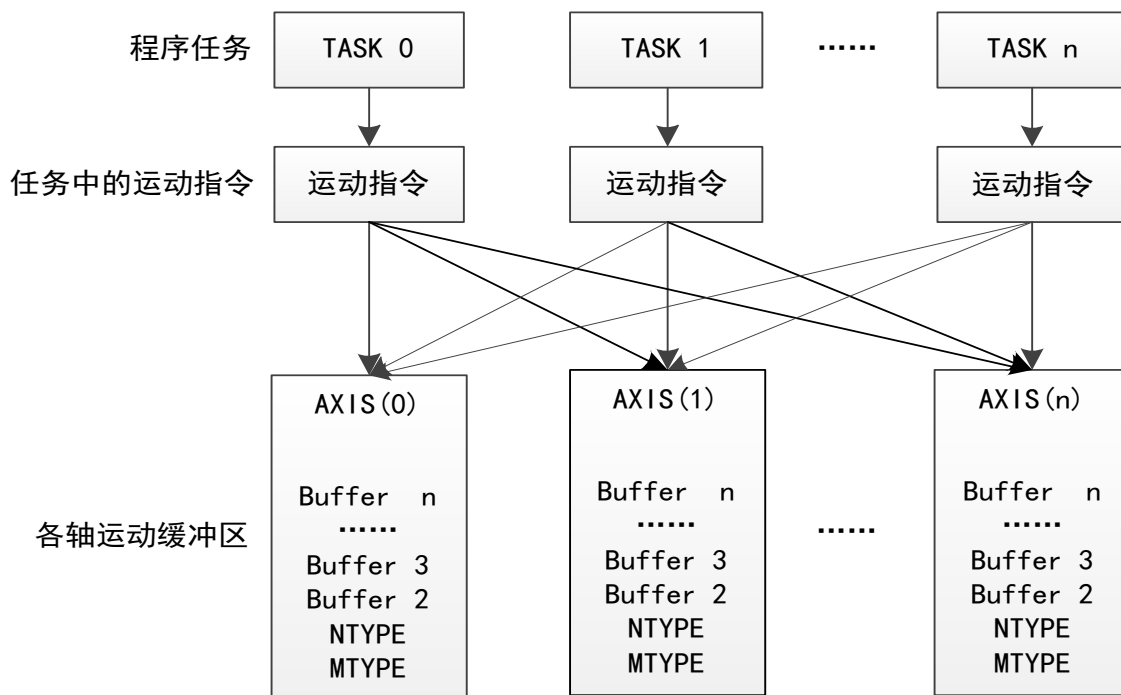
在运行运动指令时，为了防止程序堵塞，控制器提供了一个缓冲区来保存进入运动缓冲的运动缓冲队列，把这个功能叫做运动缓冲，这样程序就能正常向下扫描，不会堵塞。

ZMotion 运动控制器具有多级的运动缓冲，当运动缓冲开启的时候，程序在扫描识别到程序任务的第一条运动指令时，将运动指令分配到指定轴的运动缓冲区，电机开始运动，此时程序继续向下扫描到第二条运动指令时，再往运动缓冲区中存，在不断扫描存入运动指令的同时，从运动缓冲区中依次取出运动指令执行。

MTYPE, NTYPE 分别是当前运行的运动指令和第一个缓冲运动指令。

任意一段程序的运动指令都可以进入任意轴的运动缓冲区，由轴号指定。

每个轴的运动缓冲区都是独立的，互不干扰。



### 3.6.2. 运动缓冲区

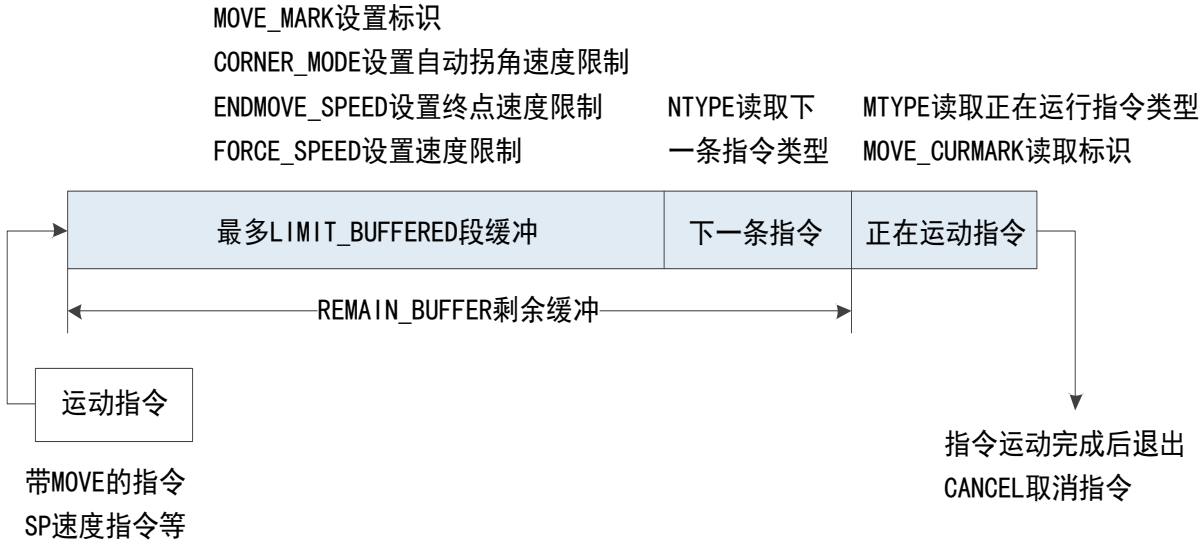
程序扫描过程中，将扫描到的运动指令存入对应轴的运动缓冲区，在从运动缓冲区中按先进先出的顺序，依次取出运动指令执行，能进入运动缓冲区的指令除了运动指令之外，还包括一系列运动缓冲中输出指令。

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

插补运动缓冲在主轴的运动缓冲区。

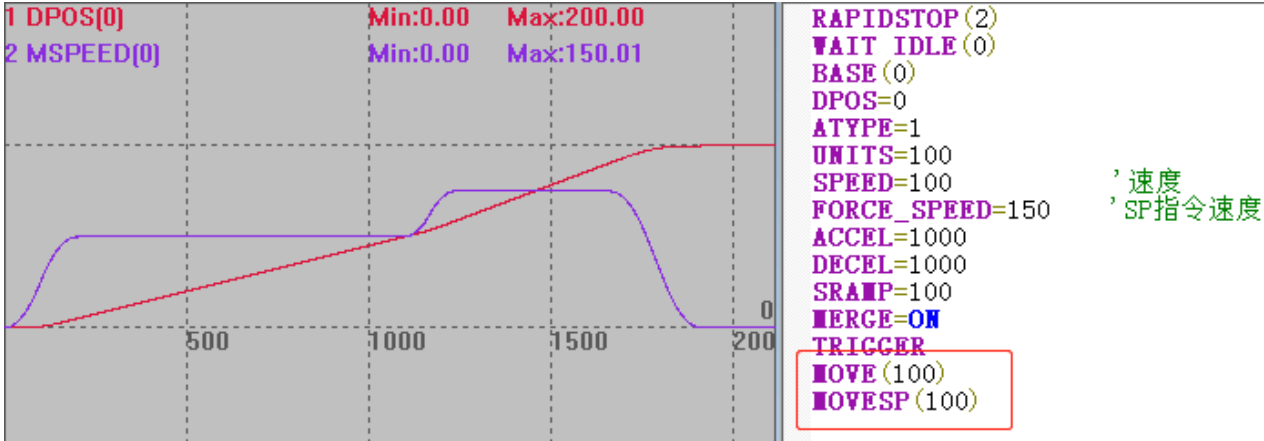
缓冲多条运动指令时，为了判断当前运动执行到哪一条，提供 MOVE\_MARK 运动标号和 MOVE\_CURMARK 当前运动标号指令。MOVE\_MARK 运动标号每扫描一条运动指令+1；MOVE\_CURMARK 指令为当前运动的标号，提示当前运动到第几条运动指令，所有运动完成后为-1。

当前运动完成后会自动执行运动缓冲区内的下一条运动。运动指令全部执行完后，运动缓冲区为空，或者使用 CANCEL/RAPIDSTOP 指令清空运动缓冲区。



SP 指令又称 SP 运动指令，使用 SP 运动指令时(MOVESP、MOVECIRCSP 等直接在运动指令后方加上 SP)，SP 指令的速度按 SP 速度参数运动，而不是 SPEED 速度，SP 速度包含 FORCE\_SPEED、ENDMOVE\_SPEED 和 STRATMOVE\_SPEED，会随 SP 运动指令写入运动缓存区。

SP 指令与非 SP 指令的运行效果如下，MOVE(100)的速度是 SPEED=100，MOVESP(100)的速度是 FFORCE\_SPEED=150。

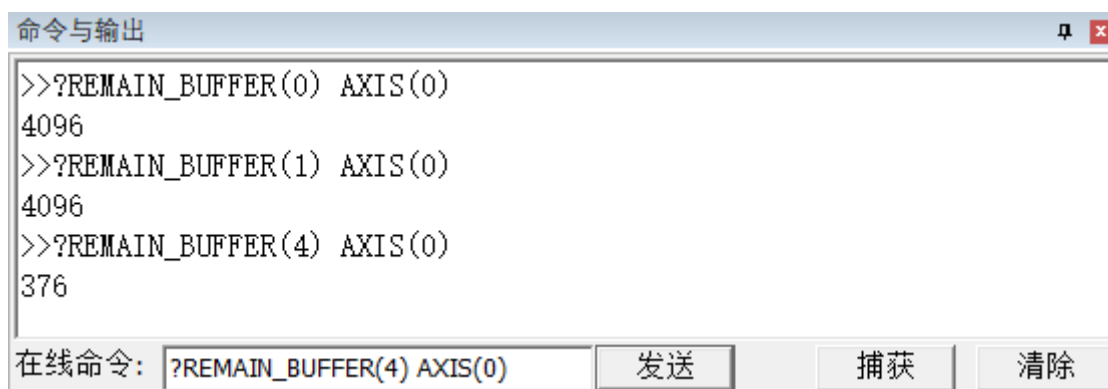


ZMC4 系列运动控制器每个轴可支持多达 4096 段运动缓冲（不同型号的控制器缓冲个数有区别，具体情况参见控制器硬件手册说明或使用 ?\*max 打印查看 max\_movebuff 参数），可以手动设置 LIMIT\_BUFFERED 运动缓冲限制。





每个轴的运动缓冲都是独立的，互不干扰，且轴的缓冲区大小相同，通过指令 `REMAIN_BUFFER(MTYPE) AXIS(n)` 查看某个轴的剩余可用缓冲区的个数。



不同的运动指令占用的缓冲空间是不同的，越复杂的运动占用的运动缓冲空间越多。例如：ZMC432 控制器，运动缓冲区大小为 4096，缓冲区一次性可缓冲的 `MOVE` 直线插补指令和 `MOVECIRC` 圆弧插补指令个数是不同的。

### 3.6.3. 运动缓冲区堵塞

由于每个轴的运动缓冲空间是有限的，当扫描太多运动指令放入运动缓冲区时，多级运动缓冲区全部被塞满，如果程序继续扫描到更多的运动指令，程序也会被堵塞，直到运动指令依次完成并退出，运动缓冲区有了空位，运动指令才会继续进入运动缓冲区。

例：以 V3.10 版本仿真器为例，默认为 4096 个运动缓冲，下图例程中显示该控制器的运动缓冲区最多能存 459 条圆弧插补指令，下载程序后后打印 `i` 的值为 458，表示当前 `FOR` 循环并未执行完，程序堵塞了。

Basic1.bas

```
1 RAPIDSTOP(2)
2 WAIT IDLE(0)
3 WAIT IDLE(1)
4
5 BASE(0,1)           ' 选择轴号
6 ATYPE=1,1           ' 脉冲轴类型
7 UNITS=100,100        ' 脉冲当量
8 SPEED=100,100        ' 运动速度
9 ACCEL=1000,1000      ' 加速度
10 DECEL=1000,1000     ' 减速度
11 SRAMP=100,100       ' S曲线
12 MERGE=ON            ' 打开连续插补
13 DPOS=0,0
14 TRIGGER              ' 触发示波器采样
15 FOR i=0 TO 1000
16   MOVECIRC(200,0,100,0,1)
17 NEXT
18 END
```

轴参数

轴选择	参数选择	轴0	轴1
LOADED		0	0
MSPEED		96.8600	24.8500
MTYPE		4	4
NTYPE		4	4
REMAIN		171.8500	0
VECTOR_BUFFERED		144050.3500	144050.3500
VP_SPEED		100	9.5100
AXISSTATUS		0h	0h
MOVE_MARK		459	6
MOVE_CURMARK		0	0
AXIS_STOPREASON		800h	0h
MOVES_BUFFERED		458	458

运动缓冲堵塞效果-圆弧插补

Basic1.bas

```
1 RAPIDSTOP(2)
2 WAIT IDLE(0)
3 WAIT IDLE(1)
4
5 BASE(0,1)           ' 选择轴号
6 ATYPE=1,1           ' 脉冲轴类型
7 UNITS=100,100        ' 脉冲当量
8 SPEED=100,100        ' 运动速度
9 ACCEL=1000,1000      ' 加速度
10 DECEL=1000,1000     ' 减速度
11 SRAMP=100,100       ' S曲线
12 MERGE=ON            ' 打开连续插补
13 DPOS=0,0
14 TRIGGER              ' 触发示波器采样
15 FOR i=0 TO 5000
16   MOVE(100,100)      ' 直线插补
17 NEXT
18 END
```

轴参数

轴选择	参数选择	轴0	轴1
LOADED		0	0
MSPEED		70.7100	70.7100
MTYPE		1	1
NTYPE		1	1
REMAIN		128.7400	0
VECTOR_BUFFERED		579099.0200	579099.0200
VP_SPEED		100	70.7100
AXISSTATUS		0h	0h
MOVE_MARK		4096	6
MOVE_CURMARK		1	1
AXIS_STOPREASON		800h	0h
MOVES_BUFFERED		4094	4094

运动缓冲堵塞效果-直线插补

下图当从运动缓冲区取出部分圆弧运动指令执行之后，缓冲区有了空间，FOR 循环继续执行，并存入运动指令到运动缓冲区。指令执行退出运动缓冲区后，只要运动缓冲区的空间够，新的运动指令一条条往运动缓冲区中存。

轴参数		
轴选择	参数选择	
	轴0	轴1
LOADED	0	0
MSPEED	56.5100	-82.5100
MTYPE	4	4
NTYPE	4	4
REMAIN	147.5000	0
VECTOR_BUFFERED	144025.2000	144025.2000
VP_SPEED	100	15.7600
AXISSTATUS	0h	0h
MOVE_MARK	499	0
MOVE_CURMARK	40	40
AXIS_STOPREASON	0h	0h
MOVES_BUFFERED	458	458

为防止运动缓冲区堵塞导致程序无法继续往下扫描的情况，我们可以在扫描运动指令的时候加入判断处理程序，确认缓冲区有空间了才扫描运动指令。

3.6.4. 运动缓冲中输出

运动缓冲中输出指令能进入运动缓冲区，可在运动缓冲中开启 OP 口、延时、输出参数，输出 PWM、开启任务等，详细指令说明参见运动指令章节。

普通输出与运动缓冲中输出的区别：  
普通输出指令程序扫描到该行指令便执行输出。

运动缓冲中输出指令在程序扫描之后，将其存入运动缓冲区，运动缓冲区按先进先出的顺序依此取出指令执行，直到取出该输出指令时才会执行输出。

示例：对比 OP 和 MOVE\_OP 的输出效果

RAPIDSTOP(2)  
WAIT IDLE(0)

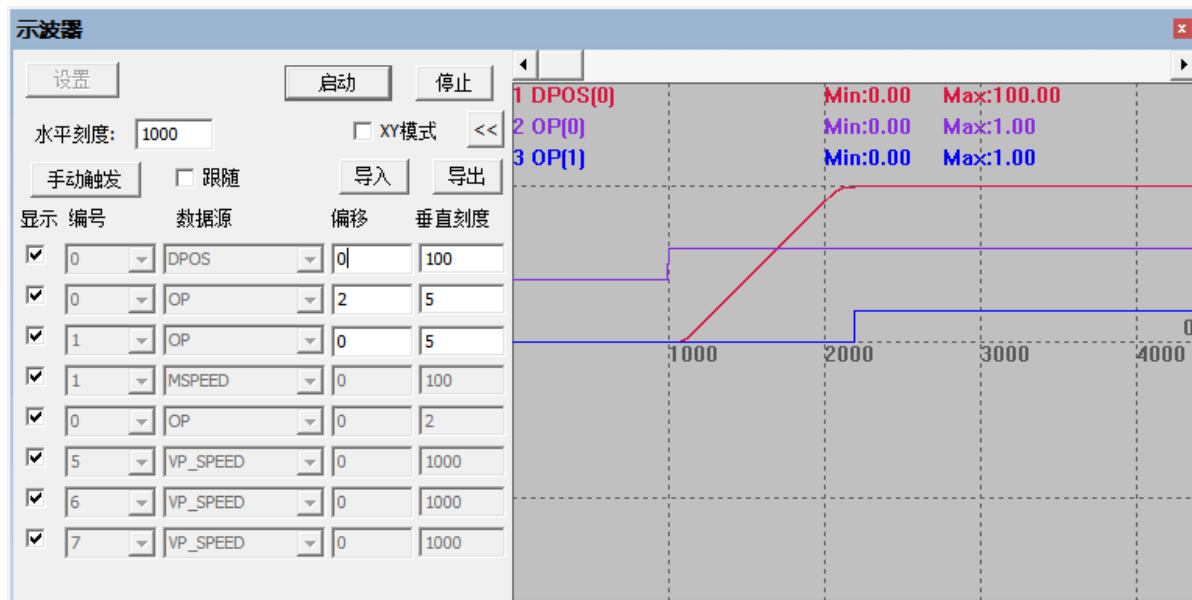
BASE(0)	'选择轴 0
DPOS=0	
UNITS=100	'脉冲当量
SPEED=100	'速度
ACCEL=1000	'加速度
DECEL=1000	'减速度
SRAMP=100	'S 曲线
TRIGGER	'触发示波器采样
OP(0,3,\$0)	'关闭输出口 0-3
DELAY(1000)	'延时
MOVE(100)	
MOVE_OP(1,ON)	'运动缓冲中输出
OP(0,ON)	'普通输出

END

例子运行效果：

延时 1s 之后，程序扫描到 OP 指令，输出口 0 立即执行输出。

MOVE\_OP 把 IO 操作指令填入运动缓冲区，所以在运行完 MOVE(100)之后，输出口 1 才输出。



## 第四章 通讯方式

### 4.1. 串口通讯

#### 4.1.1. 串口类型

控制器包含三类串口，RS232、RS485 和 RS422，其中 RS232 串口所有控制器都包含，绝大部分控制器都包含 RS485 串口，只有少数型号控制器有 RS422 串口。

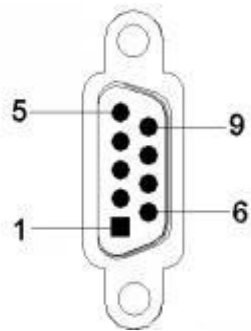
控制器的串口协议均为 MODBUS\_RTU，默认做从端，RS232 和 RS485 可通过 SETCOM 指令配置成主端，通讯速率等参数同样也是通过 SETCOM 指令配置，。

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，掉电不保存。

##### 1. RS232 串口

控制器的 RS232 接口可以做 MODBUS 主站或从站，支持 1 个主站发送数据，1 个从站接收数据。做主站时，可连接驱动器、变频器、温控仪等，进行数据读出与写入的控制。做从站时，可连接人机界面，用来监控运行状态，常用于连接 PC 或人机界面。

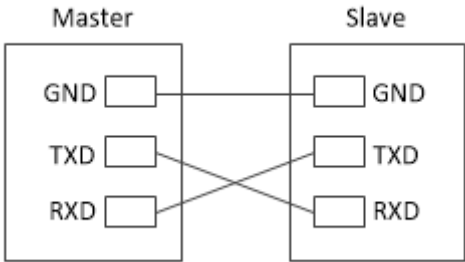
RS232 控制器采用 DB-9 接口，针脚信号说明如下：



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	电源地
9	E5V	外部电源 5V 输出

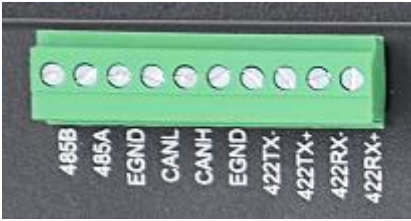
RS232 的标准接线只需要三根线即可，2 根数据信号 TXD 和 RXD，1 根地线 GND，数据信号 TXD 与 RXD 交叉连接，再将 GND 连到一起。

接线参考如下：



2. RS485 串口

主要提供主/从站的多台通讯设备联机，理论上支持 128 个节点，一主多从。做主站时，可连接驱动器、变频器、温控仪等，进行数据读出与写入的控制；做从站时，能与 PLC 通讯，可连接人机界面，用来监控运行状态。

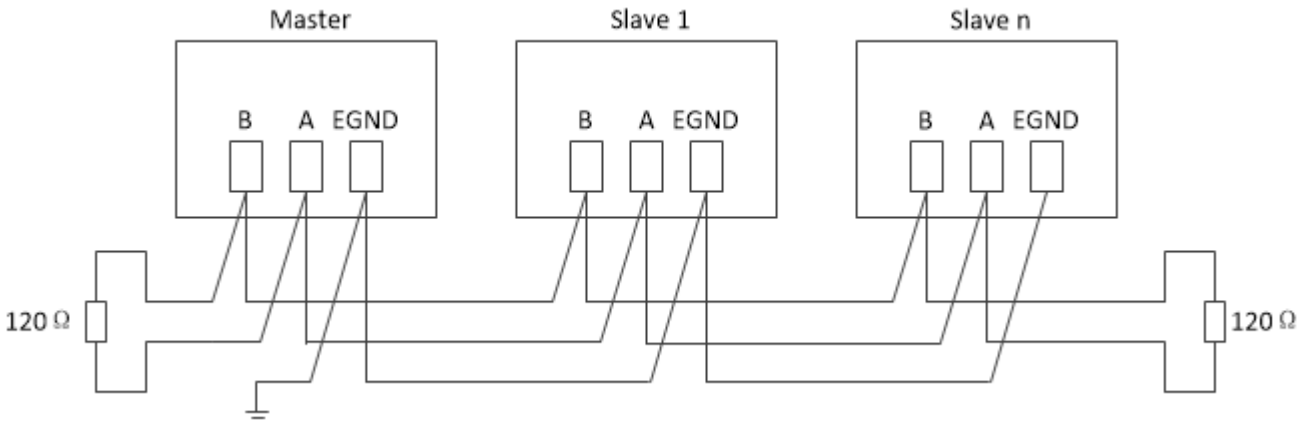


RS485 接口采用差分传输方式，通过判断 A 与 B 之间的电压差来确定是高电平或低电平。

针脚名称	说明
485B	485-
485A	485+
EGND	电源地

控制器的 RS485 接口采用了简易接线方式，如下图所示，控制器的 485A、485B、GND 地线，分别接

第一个从站的 A、B、地线，然后再接第二个从站的 A、B、地线(A 接 A，B 接 B，信号共地)，并且控制器和最后一个从站的 485A 和 485B 要并联 120 欧电阻防止信号反射，线缆需要使用屏蔽双绞线，避免信号干扰，每个节点支线的距离要小于 3m。



### 3. RS422 串口

仅 ZMC3 系列的部分控制器型号带 RS422 串口。

RS422 数据传输特性与 485 相同。RS422 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。

针脚名称	说明
422TX-	发送数据-
422TX+	发送数据+
422RX-	接受数据-
422RX+	接受数据+
EGND	电源地

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。

## 4. 1. 2. 串口连接方法

串口支持 MODBUS 通讯协议 RTU 模式，常用于连接电脑或触摸屏，通讯时注意串口的参数要匹配，不管哪种串口，除了端口号和接线方法有所不同，默认参数与操作指令都是相同的。

PC 使用串口连接控制的方法如下。

先接好线，在 ZDevelop 菜单栏点击“控制器”→“连接”，打开如下连接到控制器窗口，会自动列出本计算机上可用的串口号，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。

连接到控制器

串口

38400

无校验

0

连接

自动连接

IP

127.0.0.1

500

连接

IP扫描

PCI

连接

断开连接

本机IP:

192.168.0.163

确定

取消

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，若串口连接失败检查串口号是否正确，修改电脑的通讯端口 COM 的配置，使其与控制器的默认参数一致。

串口参数设置均使用 SETCOM 指令，串口参数是掉电不保存的，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。

串口默认为 MODBUS 从端，可修改 SETCOM 指令的 MODE=14 设置为主端，或 MODE=0 开启串口自定义通讯，即无协议模式，串口自定义通讯模式下使用 GET #指令从自定义串口通道里读取数据，PRITNT #指令从自定义串口通道里输出字符串，PUTCHAR #发指令从自定义串口通道里输出字符（ASCII 码）。

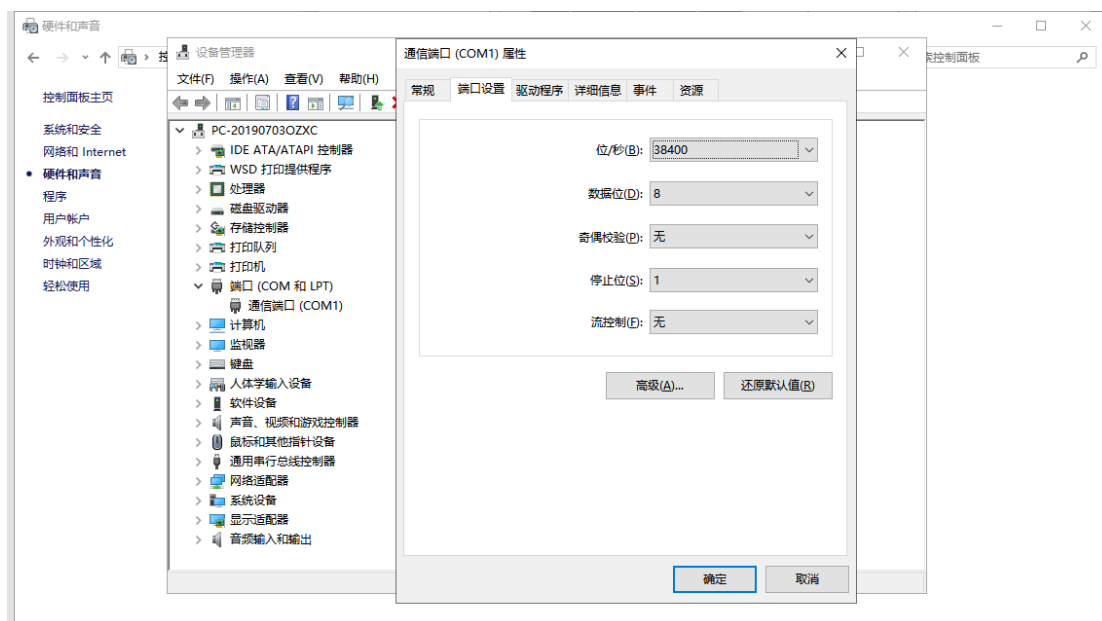
SETCOM 指令 mode 参数配置协议：

Mode 值	描述
0	RAW 数据模式，无协议，此时可以使用 GET #读取；PRITNT #、PUTCHAR #发送
4（缺省）	MODBUS 从端（16 位整数）
14	MODBUS 主端（16 位整数）
15	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)

串口的 MODBUS 通讯方法与串口的自定义通讯方法参见微信公众号“正运动小助手”相关教程。

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。

## 4.2. 网口通讯

控制器网口为 EtherNET 接口，支持 MODBUS\_TCP 通讯协议，常用于连接电脑或触摸屏，控制器一般有一个 EtherNET 接口，但是底层至少有两个网口通道，当需要使用网口连接多个设备的时候，可借助交换机，支持网口连接多少个设备使用？\*PORT 打印查看网口通道个数。

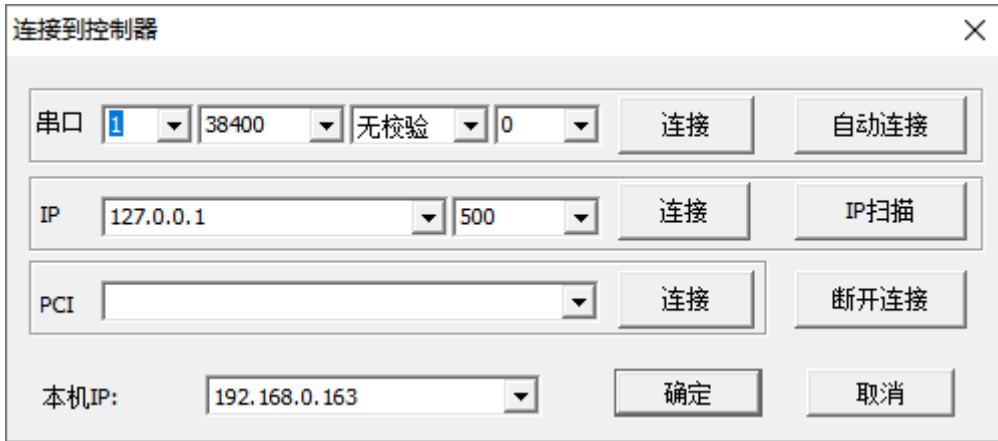




网线建议使用带屏蔽层的双绞线，保证通讯质量，在 ZDevelop 菜单栏点击“控制器”→“连接”，打开如下连接到控制窗口，IP 地址在下拉列表中选择，会自动查找当前局域网可用的控制器 IP 地址，选择正确的 IP 之后点击连接。

网口连接是需要保证控制器 IP 地址与电脑的 IP 地址处于同一网段，即四段 IP 地址的前三段相同，最后一段不同，否则会连接失败，连接失败时修改控制器 IP 或者电脑 IP 其中之一即可。

控制器出厂的 IP 是 192.168.0.11，IP 一经修改永久保存。

该窗口标题为“连接到控制器”，包含三个配置区域。第一区域为串口配置，显示“串口 1”，波特率“38400”，校验“无校验”，位“0”，右侧有“连接”和“自动连接”按钮。第二区域为网口配置，显示 IP “127.0.0.1”，端口“500”，右侧有“连接”和“IP扫描”按钮。第三区域为 PCI 配置，显示“PCI”，右侧有“连接”和“断开连接”按钮。窗口底部显示“本机IP: 192.168.0.163”，并有“确定”和“取消”按钮。

控制器网口也支持自定义通讯，使用 OPEN #指令打开自定义网口通讯，OPEN #指令支持配置网口通讯主从端，GET #指令从自定义网口通道里读取数据，PRITNT #指令从自定义网口通道里输出字符串，PUTCHAR #发指令从自定义网口通道里输出字符（ASCII 码）。

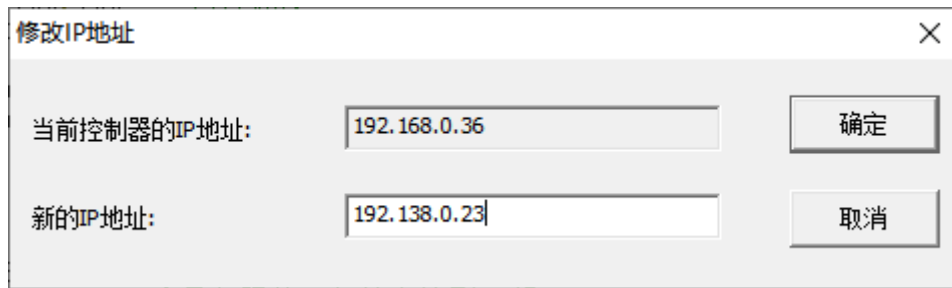
#### 控制器连接失败可能故障原因及解决办法：

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文 控制器网口通道全部被占用，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

#### 修改控制器 IP 地址

先使用串口连接控制器，获取控制器 IP 地址，再修改控制器 IP 地址。

方法一：可以通过菜单栏“控制器”→“修改 IP 地址”窗口直接修改控制器 IP 地址。



修改IP地址

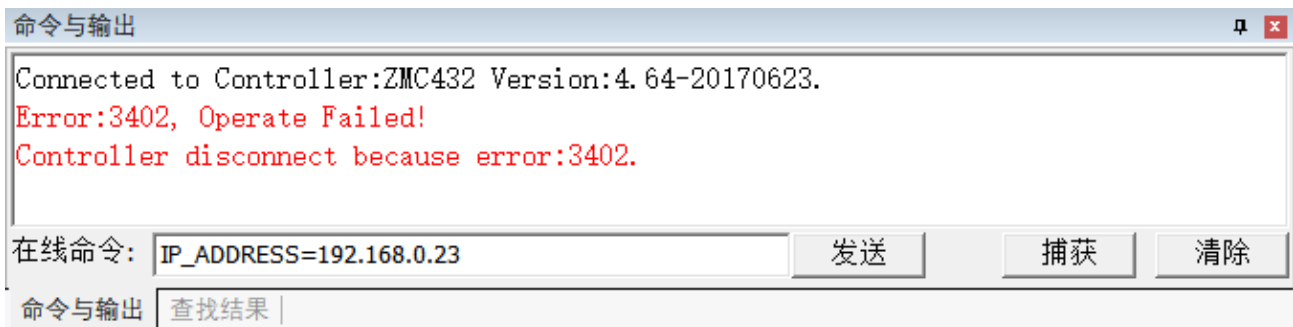
当前控制器的IP地址: 192.168.0.36

新的IP地址: 192.138.0.23

确定 取消

方法二：通过 IP\_ADDRESS 指令发送在线命令修改。

指令发送修改成功之后自动断开连接，在线命令打印控制器连接错误信息，通过网口连接选择新 IP 地址 192.168.0.23 再次连接控制器，IP 地址修改成功后永久有效。



命令与输出

Connected to Controller:ZMC432 Version:4.64-20170623.  
Error:3402, Operate Failed!  
Controller disconnect because error:3402.

在线命令: IP\_ADDRESS=192.168.0.23

发送 捕获 清除

命令与输出 查找结果 |

### 修改电脑 IP 地址

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。

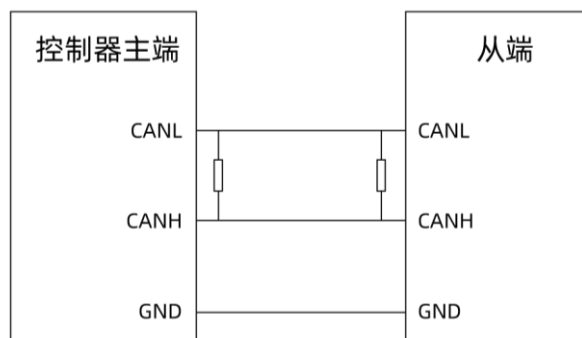
设置好之后，再次打开“连接到控制器”窗口尝试连接到控制器。



## 4.3. CAN 总线通讯

### 4.3.1. CAN 接线

控制器的 CAN 总线接口用于接 ZCAN 扩展模块，或连接控制器。CAN 总线连接控制器接线方法与连接 ZCAN 扩展模块相同，不同之处在于板块上没有集成 120 电阻，需要在 CANL 和 CANH 的首尾两端各接一个 120 欧姆电阻，CAN 总线连接扩展模块接线方法参见“[ZCAN 扩展模块](#)”章节。



控制器使用 CAN 总线连接时，此时控制器之间可以使用 CAN 指令通讯，数据通过 TABLE 传递。

控制器缺省做 CAN 通讯的主端，控制器之间的 CAN 通讯需要把一个控制器配置为从端，使用 CANIO\_ADDRESS 指令配置主从端，CANIO\_ADDRESS=32 配置为主端，CANIO\_ADDRESS=其他值配置

为从端。

指令语法: 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 (参见下表, 模式 6/7 适用标准帧, 模式 16/17 适用扩展帧)

tablenum: 数据存储的 TABLE 位置

值	描述
6	接收, 没有数据时, identifier<0
7	发送
16 (需要升级固件)	带扩展支持接收, 没有数据时, identifier<0
17 (需要升级固件)	发送扩展数据, 普通数据使用 7 发送

例子:

'发送端: 第一个控制器

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) '发送数据

'接收端: 第二个控制器

CANIO\_ADDRESS=1 '设置为 CAN 从端, 此参数设置一次即可

CAN(0,6,0) '接收数据

?TABLE(0)

## 4.4. U 盘接口

正运动大部分运动控制器都带有一个标准 U 盘接口。

没有控制器的场合, 可以在 ZDevelop 根目录新建 udisk 文件夹模拟 U 盘使用, 连接到仿真器, 调试 U 盘指令。

U 盘接口主要有三方面的用途:

### 1. 程序升级

通过 U 盘口, 下载打包好的 zar 程序包, 方便客户更新系统程序。

程序升级之前事先将 zar 程序包下载到 U 盘里面。使用 FILE 指令加载 U 盘文件成功后, zar 程序自动开始运行。

示例:

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 '扫描文件成功判断

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

### 2. 加载三次文件

使用 FILE 指令加载 U 盘里保存的三次文件执行。

示例：

```
IF U_STATE=TRUE THEN    '判断 U 盘是否插入
    FILE "FIND_FIRST",".Z3P",800    '查找 Z3P 文件
    ?"文件名: "VRSTRING(800,20),"等待下载"
    FILE "COPY_TO",VRSTRING(800,20),VRSTRING(800,20)    '下载 Z3P 文件
    ?"下载 Z3P 文件完成"
ENDIF
```

### 3. U 盘与寄存器数据交互

U 盘支持读写变量和数组。

FLASH 数据拷贝：多个控制器中 FLASH 存储的数据可以通过 U 盘来相互传递。

VR 寄存器、TABLE 寄存器与 U 盘里的数据互相传递。

读写文件类型为 SD(filename).BIN 或 SD(filename).CSV，不同的指令可操作的文件类型有所区别。

不同型号的控制器的 U 盘接口的使用方法都是相同的，将 U 盘插在控制器上的 UDISK 端口即可，控制器上电后有 U 盘插入时，U 盘指示灯亮。

在对 U 盘进行操作之前，先使用 U\_STATE 指令判断 U 盘的状态，确保 U 盘能成功通讯，再使用 U 盘相关指令操作。

示例：

```
DIM a,array1(2)    '变量、数组定义
a=123
array1(0)=10
array1(1)=20

IF U_STATE = TRUE THEN    '判断 U 盘是否插入
    U_WRITE 0,a,array1    '将变量、数组写入 U 盘的 SD0 文件
    a=456
    array1(0)=11
    U_READ 0,a,array1    '读取 U 盘文件 SD0 数据
    PRINT a,array1(0)    '结果：123, 10
    IF a <> 123 THEN    '判断 U 盘读写是否成功
        PRINT "U 盘读取错误"
    ELSE
        PRINT "U 盘成功读取"
    ENDIF
ELSE
    PRINT "U 盘未插入"
ENDIF
END
```

U 盘更多操作参见“[U 盘相关指令](#)”章节。

## 4.5. EtherCAT 总线通讯

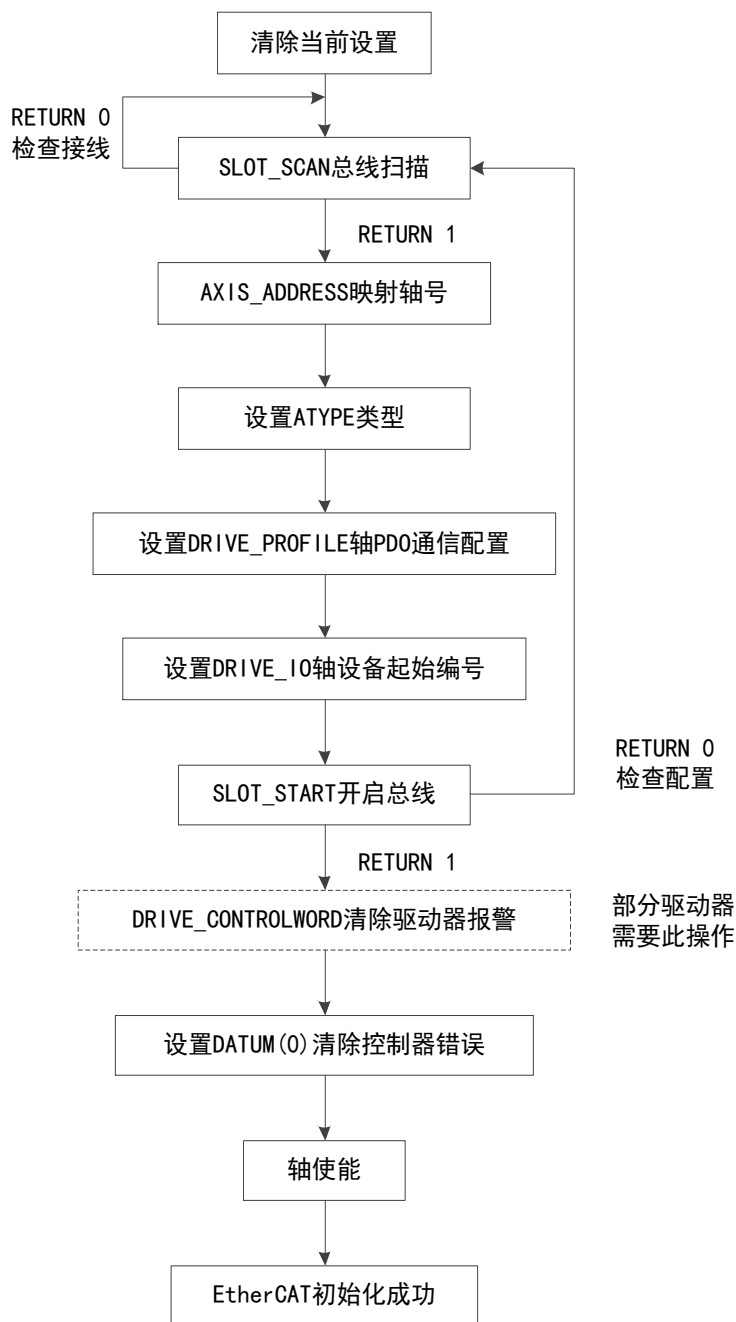
### 4.5.1. EtherCAT 总线初始化

EtherCAT 总线接口可用于连接 EtherCAT 伺服驱动器和 EtherCAT 扩展模块，无论连接什么模块，EtherCAT 总线都需要编写一段 EtherCAT 总线初始化程序来进行电机和 EtherCAT 扩展模块的使能。使能之后的应用与脉冲电机一致，运动指令都是相同的。

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。

EtherCAT 初始化程序一般过程：初始化程序参见[例程](#)



#### 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编号 0 依次递增编号，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，详细请参见总线指令章节说明。

EtherCAT 相关指令：

指令	含义
<a href="#">SLOT_SCAN</a>	扫描设备
<a href="#">SLOT_START</a>	总线启动
<a href="#">SLOT_STOP</a>	总线停止
<a href="#">ATYPE</a>	轴类型，例如 65 为 EtherCAT 周期位置控制
<a href="#">AXIS_ADDRESS</a>	轴地址配置
<a href="#">WDOG</a>	轴统一使能控制
<a href="#">SERVO_PERIOD</a>	刷新率
<a href="#">AXIS_ENABLE</a>	轴使能
<a href="#">SDO_WRITE</a>	SDO 操作
<a href="#">SDO_READ</a>	SDO 操作
<a href="#">SDO_WRITE_AXIS</a>	驱动器 SDO 操作
<a href="#">SDO_READ_AXIS</a>	驱动器 SDO 操作
<a href="#">?*ETHERCAT</a>	总线信息输出
<a href="#">NODE_COUNT</a>	设备个数
<a href="#">NODE_STATUS</a>	设备状态
<a href="#">NODE_AXIS_COUNT</a>	设备带驱动个数
<a href="#">NODE_IO</a>	设备 IO 编号
<a href="#">NODE_AIO</a>	设备 AIO 编号
<a href="#">NODE_INFO</a>	设备信息读取
<a href="#">NODE_PROFILE</a>	设备 PROFILE 设置，选择 PDO 报文内容
<a href="#">DRIVE_FE</a>	驱动误差
<a href="#">DRIVE_FE_LIMIT</a>	驱动误差设置
<a href="#">DRIVE_STATUS</a>	驱动状态字
<a href="#">DRIVE_TORQUE</a>	驱动器力矩反馈
<a href="#">DRIVE_MODE</a>	驱动器运动模式



<a href="#">DRIVE_PROFILE</a>	驱动器 PROFILE 设置，选择 PDO 报文内容
<a href="#">DRIVE_CONTROLWORD</a>	驱动器控制字
<a href="#">DRIVE_CW_MODE</a>	驱动器控制字操作模式
<a href="#">DRIVE_IO</a>	远程 IO 编号设置
<a href="#">AXISSTATUS</a>	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。

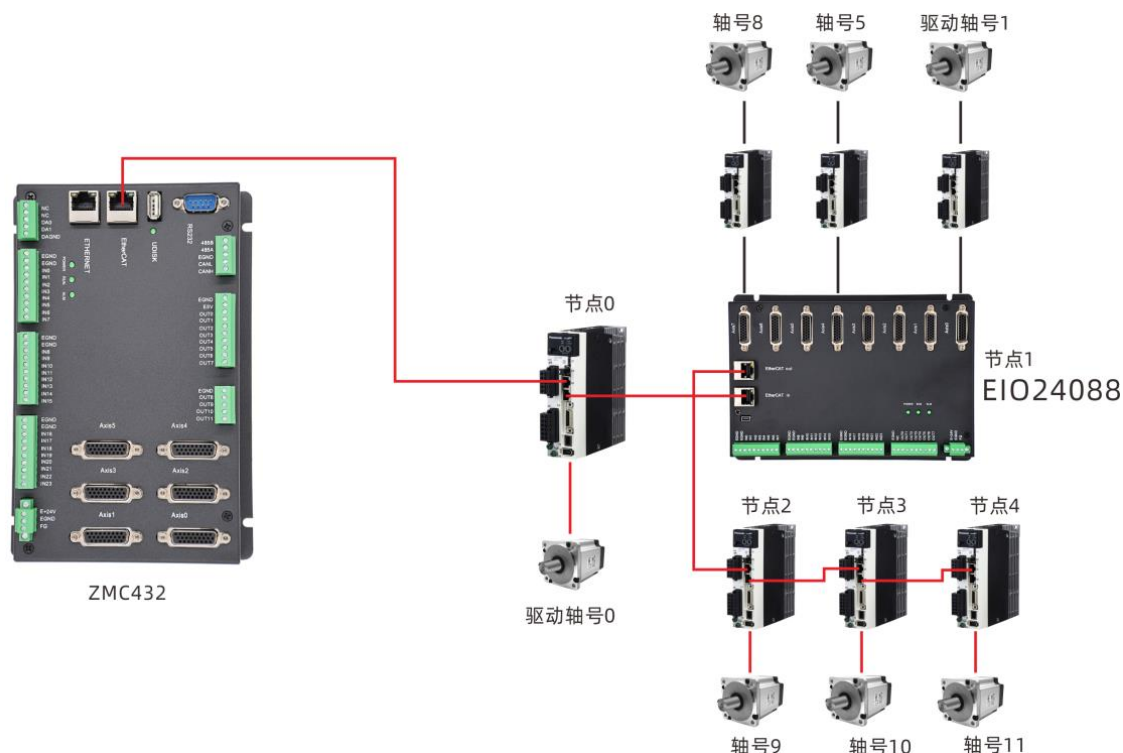
## 4.5.2. EtherCAT 总线与驱动器通讯

伺服驱动器或与 EtherCAT 扩展模块控制器接线遵循的规则相同，使用一根网线将控制器的 EtherCAT 总线端口与其他设备的 EtherCAT 口相连。

注意伺服驱动器的 EtherCAT 接口有两个，有些驱动器这两个口可以随意接，有些分为 EtherCAT IN 和 EtherCAT OUT，IN 口接上一级设备，OUT 口接下一级设备，二者不能混用，要注意连接顺序。

多轴控制时伺服驱动器的 EtherCAT OUT 口再连接下一级驱动设备的 EtherCAT IN 口，依此类推。

EtherCAT 总线的接线配置如下图：



EtherCAT 总线槽位号默认为 0。

设备号(node)，又叫节点，是指一个槽位上连接的所有设备的编号，从 0 开始，按设备在总线上的连接顺序自动编号。

控制器会自动识别出槽位上的驱动器，编号从 0 开始，按驱动器在总线上的连接顺序自动编号。驱动器编号与设备号不同，只给槽位上的驱动器设备编号，其他设备忽略。

EtherCAT 总线和 RTECH 总线的编号规则相同。

EtherCAT 总线上连接的电机需要编写一段 EtherCAT 总线初始化程序来进行使能。使能之后若 ATYPE=65 位置模式，用法与脉冲电机一致，运动指令都是相同的；若 ATYPE=66 速度模式或 ATYPE=67 力矩模式或，此时不能使用运动指令，只能使用 DAC 指令控制轴持续运动，停止将 DAC=0。

### 4.5.3. EtherCAT 总线连接扩展模块

EtherCAT 扩展模块可扩展 IO 和脉冲轴，接线规则与 EtherCAT 驱动器相同，接线图参考上节，接线时注意扩展模块上的 EtherCAT IN 和 EtherCAT OUT 不能混用。

按要求接线完成，先对 EtherCAT 扩展模块进行初始化，初始化过程中需要对扩展的 IO 和扩展的脉冲轴资源进行映射才能使用，扩展资源映射在总线扫描之后总线开启之前按下述方法操作映射。

EtherCAT 总线上的 IO 映射采用 NODE\_IO 指令(数字量)、NODE\_AIO 指令(模拟量)设置，轴映射采用 AXIS\_ADDRESS 指令映射轴号。

#### 扩展资源映射方法：

##### 1. IO 映射

Slot 槽位号和 node 设备号按照与控制器的连接顺序，从 0 开始自行编号。

NODE\_IO 指令设置设备的数字量 IO 起始编号，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设置，NODE\_AIO 指令使用与 NODE\_IO 指令基本相同。

语法：

NODE\_IO(slot, node)=iobase

slot: 槽位号, 0-缺省

node: 设备编号, 编号从 0 开始

ioBASE: 映射 IO 起始编号, 设置结果只会是 8 的倍数

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。0-缺省, 同时设置 AIN、AOUT, 读取时只读 AIN; 3-AIN; 4-AOUT

IO 映射示例:

```
SLOT_SCAN(0)           '扫描总线
IF NODE_COUNT(0)>0 THEN '判断槽位 0 上是否有设备
    NODE_IO(0,0)=32      '设置槽位 0 接口设备 0 的 IO 起始编号为 32
    NODE_AIO(0,1,3)=8    '设置槽位 0 接口设备 1 的 AIN 起始编号为 8
ENDIF
```

## 2. 轴映射

总线轴需要进行轴映射操作, 采用 AXIS\_ADDRESS 指令映射, 操作方法如下:

AXIS\_ADDRESS(轴号)=(槽位号<<16)+驱动器编号+1

轴映射写在总线初始化程序中, 扫描总线之后, 开启总线之前。

示例:

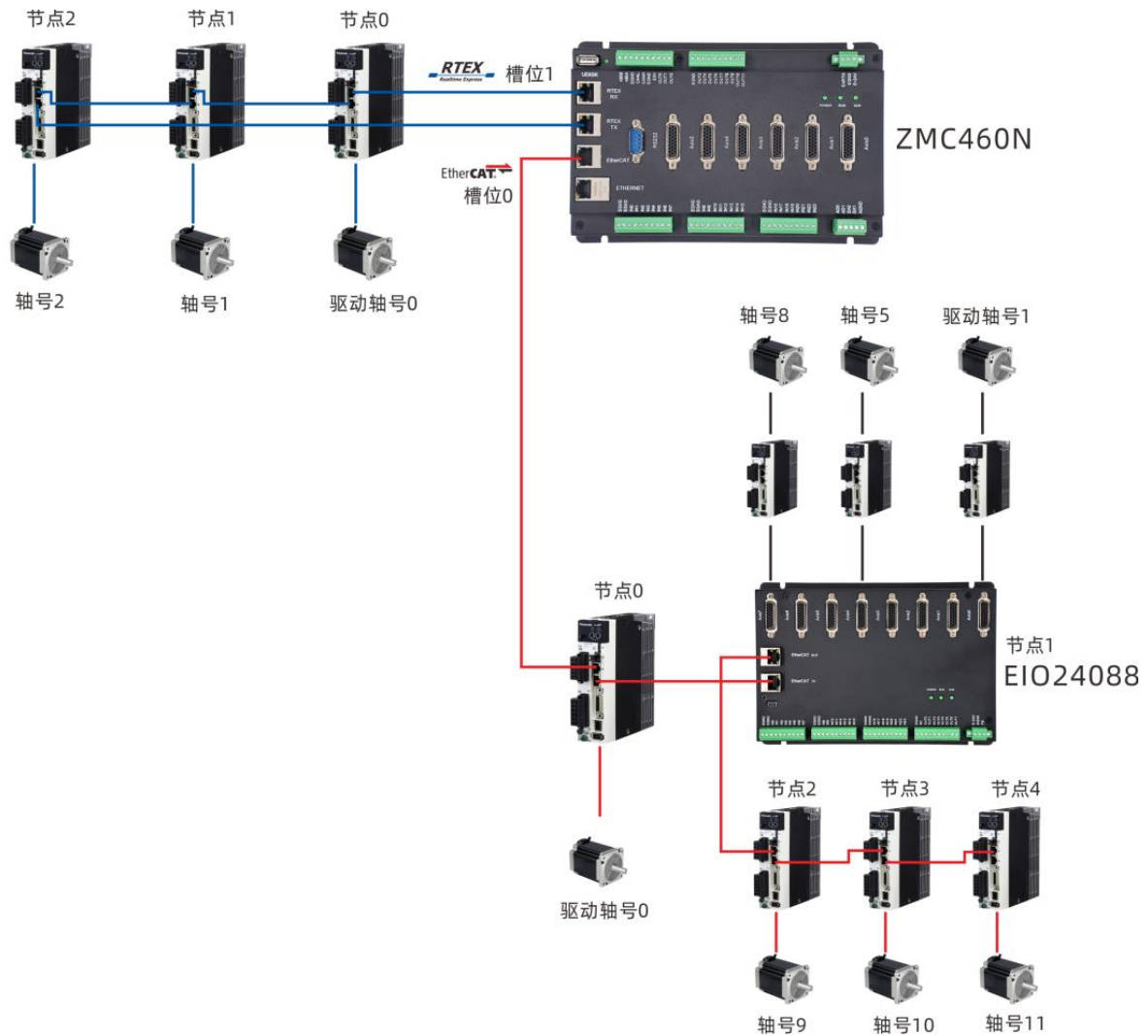
```
AXIS_ADDRESS (0)=(0<<16)+0+1 '第一个 ECAT 驱动器, 驱动器编号 0, 绑定为轴 0
AXIS_ADDRESS (2)=(0<<16)+1+1 '第二个 ECAT 驱动器, 驱动器编号 1, 绑定为轴 2
AXIS_ADDRESS (1)=(0<<16)+2+1 '第三个 ECAT 驱动器, 驱动器编号 2, 绑定为轴 1
ATYPE(0)=65                    '设置为 ECAT 轴类型, 65-位置 66-速度 67-转矩
ATYPE(1)=65
ATYPE(2)=65
```

## 4.6. RTEX 总线通讯

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, 使用前先需要编写一段总线初始化程序使能。

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。

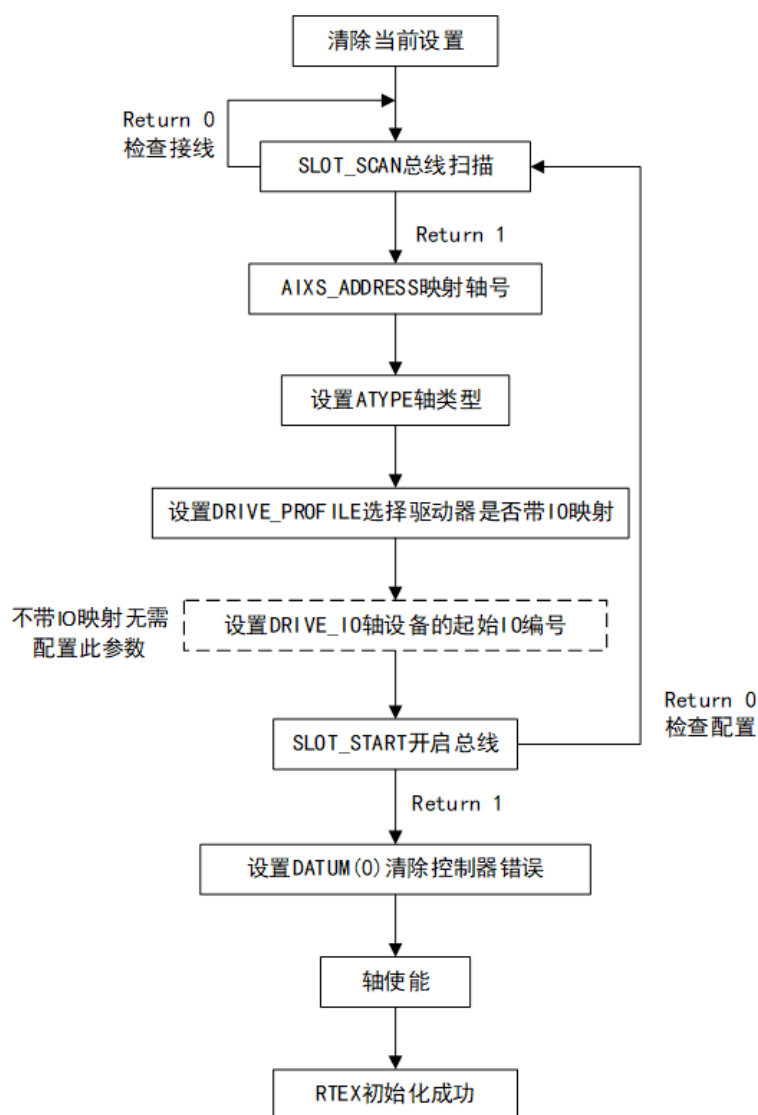
RTEX 的接线方法参见下图:



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：初始化程序参见[例程](#)



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，详细请参见总线指令章节说明。

RTEX 相关指令：

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

## 第五章 运动控制功能

### 5.1. 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，分为 JOG 点动、MOVE 寸动和 VMOVE 持续运动三类。
2. 连续轨迹运动：该控制又称为插补，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，有轨迹前瞻预处理功能，插补运动说明参见下节。
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。

#### 5.1.1. 单轴点动

相关指令：

指令	说明	用法
<a href="#">VMOVE</a>	持续运动，可选正负方向	VMOVE(运动方向)
<a href="#">CANCEL</a>	单轴停止，四种停止模式	CANCEL(模式)
<a href="#">JOGSPEED</a>	JOG 时的运动速度	JOGSPEED=速度值
<a href="#">FWD_JOG</a>	正向 JOG 输入对应的输入口编号	FWD_JOG=输入编号
<a href="#">REV_JOG</a>	负向 JOG 输入对应的输入口编号	REV_JOG=输入编号
<a href="#">FHSPEED</a>	在 FHOLD_IN 被按下时的保持速度	FHSPEED=速度值
<a href="#">FHOLD_IN</a>	保持输入对应的输入点编号	FHOLD_IN=输入编号
<a href="#">FAST_JOG</a>	快速点动输入编号	FAST_JOG=输入编号

VMOVE 为单轴持续运动指令，当 VMOVE 执行后，除非使用 CANCEL 或 RAPIDSTOP 清除运动缓存，否则会一直运转。

当前面的 VMOVE 运动没有停止时，后方的 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。

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。

MOVE 指令控制轴寸动，设定寸动的距离，给目标轴发送有限个脉冲，可用在单轴或多轴的运动场合，可一次性给多个轴发送脉冲，在点位运动的控制器上只能控制各个轴单独运动，无法联合插补。

单轴点动例一：VMOVE 持续运动

RAPIDSTOP(2)

WAIT IDLE(0)

```

BASE(0)           '选择轴号
ATYPE=1           '轴类型设置
UNITS=100         '脉冲当量设置
SPEED=100         '速度设置
ACCEL=1000        '加速度设置
DECEL=1000        '减速度设置
SRAMP=100         'S 曲线
DPOS=0            '当前位置清 0
TRIGGER

```

```

WHILE 1                                '循环运动
    IF SCAN_EVENT(IN(0))>0 AND IDLE=-1 THEN      'IN(0)按下往左运动
        VMOVE(-1)
    ELSEIF SCAN_EVENT(IN(1))>0 AND IDLE=-1 THEN  'IN(1)有按下往右运动
        VMOVE(1)
    ELSEIF SCAN_EVENT(IN(0))<0 OR SCAN_EVENT(IN(1))<0 THEN
        CANCEL                                '取消输入时停止运动
    ENDIF
WEND
END

```

单轴点动例二：JOG 点动

注意映射 JOG 开关后，一定要 INVERT\_IN 反转电平，因为 ZMC 系列控制器是 OFF 信号有效，若不反转，则导致信号接入时为 OFF，控制器判断有输入，立即控制轴运动。

```

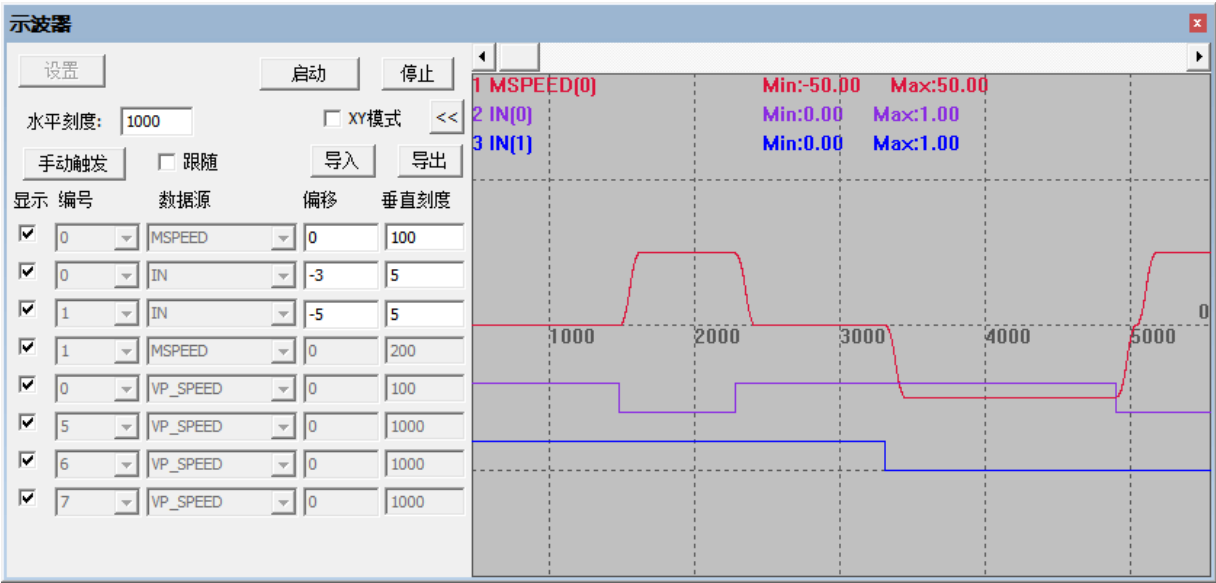
BASE(0)           '选择轴 0
ATYPE=1           '脉冲轴类型
DPOS=0            '坐标清 0
UNITS=100         '脉冲当量
SPEED =100        '主轴速度
ACCEL=1000        '加速度
DECEL=1000        '减速度
SRAMP=100         'S 曲线
TRIGGER           '自动触发示波器
JOGSPEED=50       'JOG 速度 50
FWD_JOG=0         'IN0 作为正向 JOG 开关

```



REV\_JOG=1            'IN1 作为负向 JOG 开关  
INVERT\_IN(0,ON)       '输入 0 信号反转  
INVERT\_IN(1,ON)       '输入 1 信号反转

运行效果：  
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。  
输入 0、1 同时有信号输入时，轴 0 正向运行。

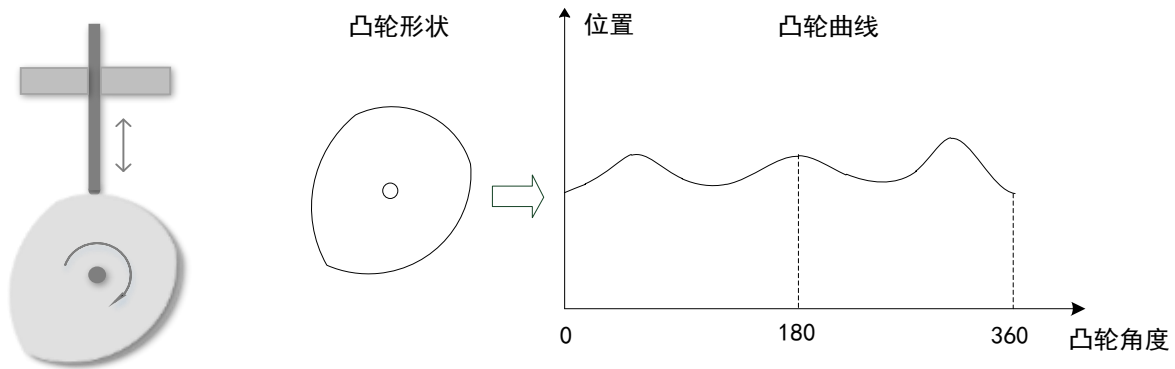


5.1.2. 电子凸轮

相关指令：

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

机械凸轮主要由主动件和从动件构成，将旋转运动转换为线性运动，基本结构如下左图。主动件为在金属盘上加工轮廓曲线，一般为匀速旋转运动；从动件一般与凸轮轮廓接触，主动件在运动时，从动件往复运动，运动轨迹由凸轮轮廓决定。



由于机械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。因此在很多应用上，使用电子凸轮替代机械凸轮。

电子凸轮指的是用户通过构造凸轮曲线，如上右图，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，以到达主动件与从动件之间的运动关系，不需要额外的机械结构辅助便可完成凸轮运动，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。

正运动控制器为用户提供了多种电子凸轮运动形式：

CAM：凸轮表运动；

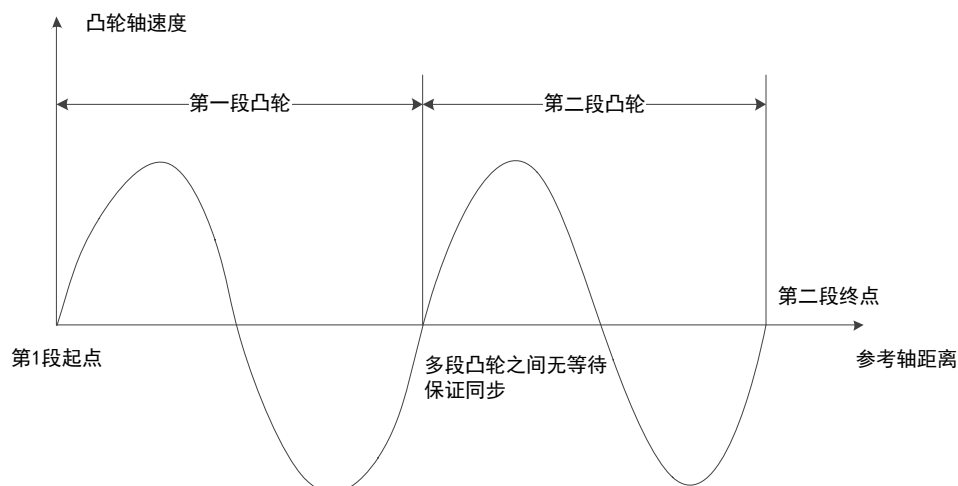
CAMBOX：跟随凸轮表运动；

MOVELINK、MOVESLINK：特定轨迹凸轮运动，又称追剪运动，一般应用于追剪应用；

FLEXLINK：特定轨迹凸轮运动，又称飞剪运动，一般应用于飞剪应用。

凸轮运动过程中不支持运动暂停，凸轮指令执行完成便停止，使用 CANCEL 或 RAPIDSTOP 指令强制取消凸轮运动。

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。

### 5.1.3. 电子齿轮

相关指令：

指令	说明	用法
<a href="#">CONNECT</a>	连接轴运动	CONNECT (比率, 连接轴轴号)
<a href="#">CONNPATH</a>	连接轴运动	CONNPATH (比率, 连接轴轴号)
<a href="#">CANCEL</a>	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，电子齿轮功能用于两个轴的连接，将主轴与从轴按照一个常数齿轮比建立连接，不需要物理齿轮，使用指令直接设置电子齿轮的比值，由于是使用软件实现的，故电子齿轮比率可以随时更改，一个主轴能够驱动多个从轴。

电子齿轮功能通过指令 **CONNECT**、**CONNPATH** 实现，将一个轴按照一定比例连接到另一个轴上做跟随运动，一条运动指令就能驱动两个电机的运行，通过对这两个电机轴移动量的检测，将位移偏差反馈到控制器并获得同步补偿，这样能使两个轴之间的位移偏差量控制在精度允许范围内。

电子齿轮连接的是脉冲个数，例如主从轴连接比例为 1:5，给主轴发送 1 个脉冲，此时对应给从轴发送 5 个脉冲。

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。

**CONNPATH** 与 **CONNECT** 的相同点：二者的使用语法相同，连接的都是脉冲个数，**CONNPATH** 连接到单个轴的运动的的效果与 **CONNECT** 相同。

**CONNPATH** 与 **CONNECT** 的区别：**CONNECT** 连接的是单个轴的目标位置。**CONNPATH** 是连接的是插补轴的矢量长度，此时需要连接在插补运动的主轴上，连接到插补从轴上无法跟随插补运动。**CONNPATH** 会跟踪 XY 轴插补的的矢量长度变化，而不是跟踪单独的 X 轴或者 Y 轴。

### 5.1.4. 手轮

相关指令：

指令	说明	用法
<a href="#">CONNECT</a>	连接手轮	CONNECT (比率, 连接轴轴号)
<a href="#">CANCEL</a>	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，属于编码器的一种，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，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，使用 **CONNECT** 指令建立手轮轴与跟随轴的连接，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  $n \times \text{ratio}$  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

## 5.2. 插补运动

### 5.2.1. 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。

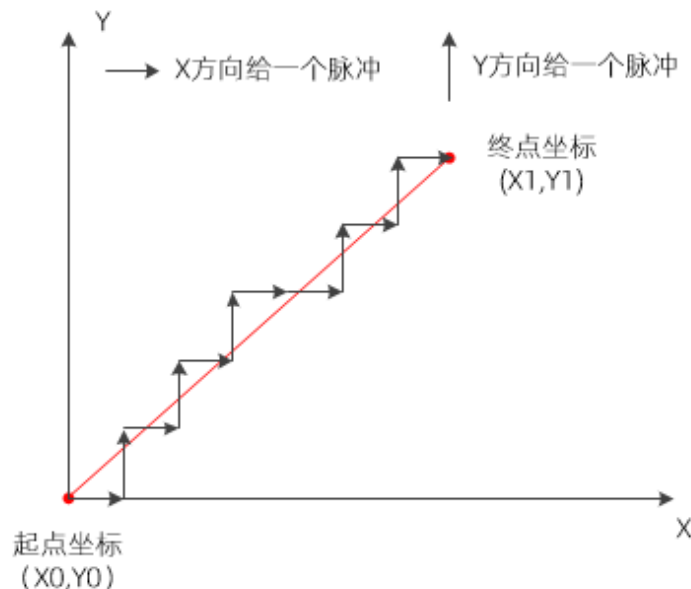
插补运动指令会存入运动缓冲区，再依次从运动缓冲区中取出指令执行，直到插补运动全部执行完。

#### 1. 直线插补原理

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。

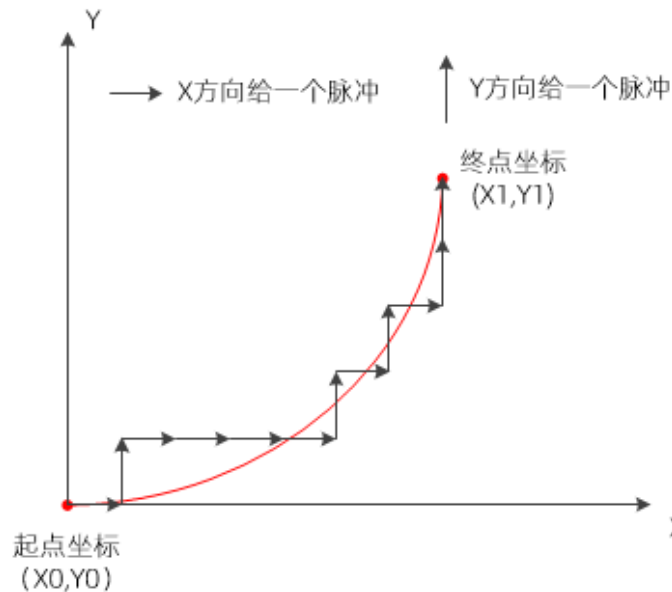
假设轴需要在在  $XY$  平面上从点 $(X_0, Y_0)$ 运动到点 $(X_1, Y_1)$ ，其直线插补的加工过程如下图所示。



#### 2. 圆弧插补原理

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。

### 3. 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，单通道最多支持 16 轴联合插补；
- 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。

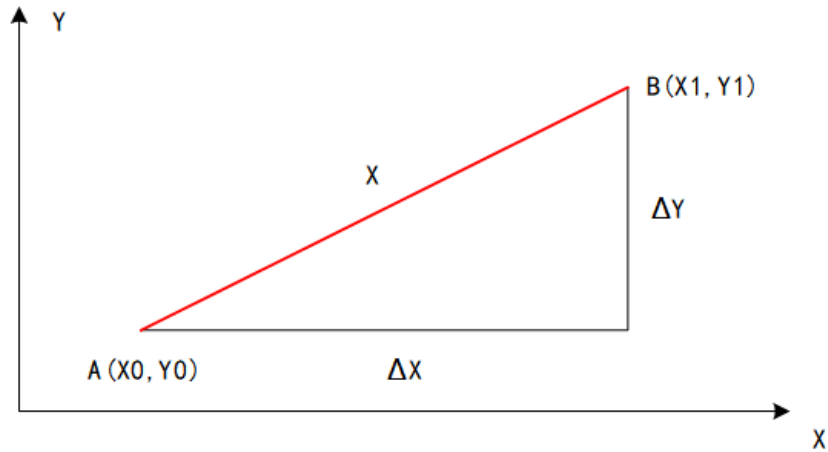
### 4. 二轴直线插补

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  $\Delta X$ ，轴 1 的运动距离为  $\Delta Y$ ，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： $X=[(\Delta X)^2+(\Delta Y)^2]^{\frac{1}{2}}$

轴 0 实际速度： $S_0=S \times \Delta X/X$

轴 1 实际速度： $S_1=S \times \Delta Y/X$



### 5. 三轴直线插补

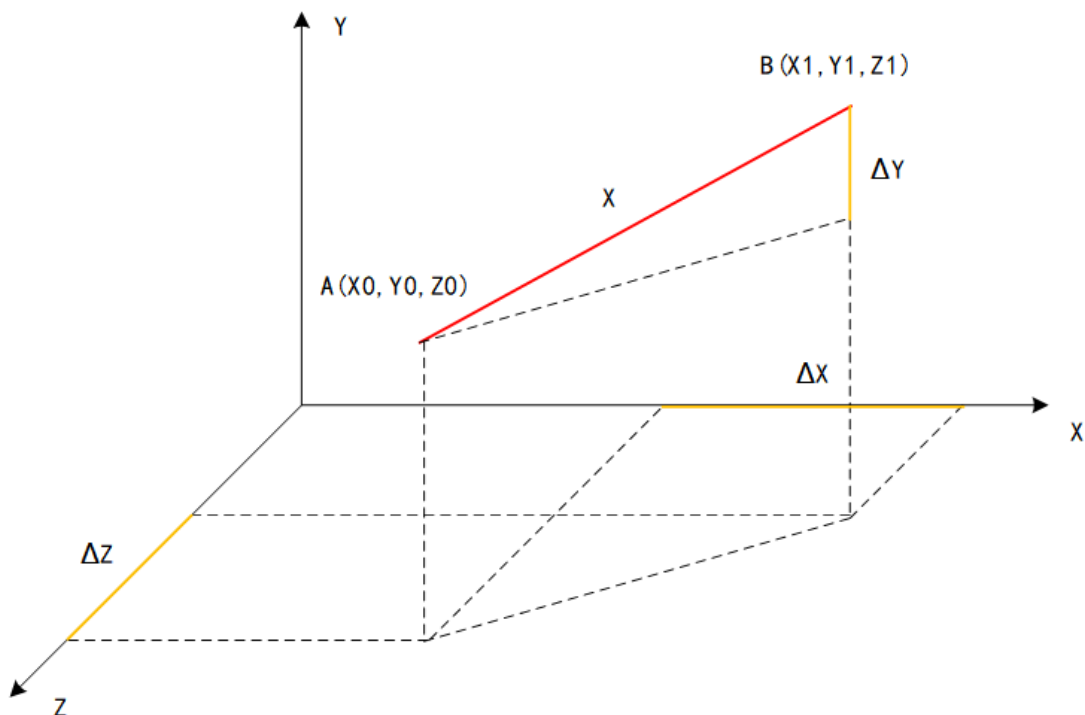
轴 0、轴 1 和轴 2 三轴参与直线插补运动，如下图，3 轴直线插补运动从 A 点运动到 B 点，XYZ 轴同时启动，并同时到达终点，设置轴 0 的运动距离为  $\Delta X$ ，轴 1 的运动距离为  $\Delta Y$ ，轴 2 的运动距离为  $\Delta Z$ ，插补主轴轴 0 的运动速度为  $S$ ，各个轴的实际速度为主轴的分速度，不等于  $S$ ，此时：

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：  $S_0=S \times \Delta X/X$

轴 1 实际速度：  $S_1=S \times \Delta Y/X$

轴 2 实际速度：  $S_2=S \times \Delta Z/X$



### 6. 多轴直线插补

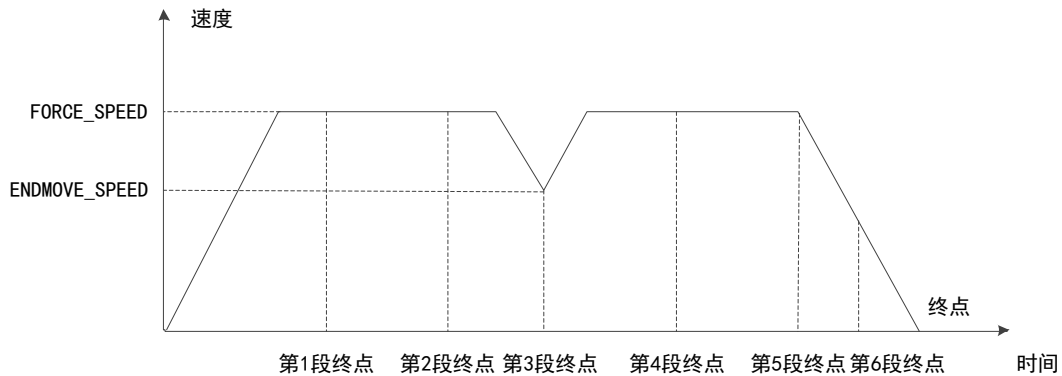
多轴直线插补可以理解为轴的多个自由度，是在三维空间里的直线插补。以四轴插补为例，一般是三个轴在 XYZ 平面走直线，另一个轴为旋转轴，按照一定的比例关系做跟随运动。



### 5.2.2. 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。



### 5.3. 前瞻预处理

前瞻运动相关指令：

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

在实际加工过程中，为追求加工效率会开启连续插补，运动轨迹的拐角处若不减速，当拐角较大时，会对机台造成较大冲击，影响加工精度。若关闭连续插补，使拐角处减速为 0，虽然保护了机台，但是加工效率受到了较大影响，所以提供了前瞻指令，使在拐角处自动判断是否将拐角速度降到一个合理的值，既不会影响加工精度又能提高加工的速度，这就是轨迹前瞻功能的作用。

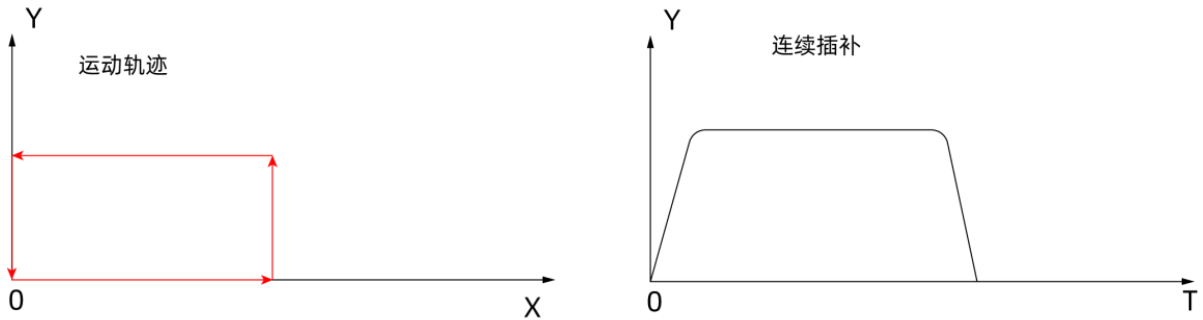
运动控制器的轨迹前瞻可以根据用户的运动路径自动计算出平滑的速度规划，减少机台的冲击，从而提高加工精度。自动分析在运动缓冲区的指令轨迹将会出现的拐点，并依据用户设置的拐角条件，自动计算拐角处的运动速度，也会依据用户设定的最大加速度值计算速度规划，使任何加减速过程中的加减速都不超过 ACCEL 和 DECEL 的值，防止对机械部分产生破坏冲击力。

使用轨迹前瞻和不使用轨迹前瞻的速度规划情况：

假设运动轨迹如下左图，走一个长方形轨迹，分为四段直线插补运动。

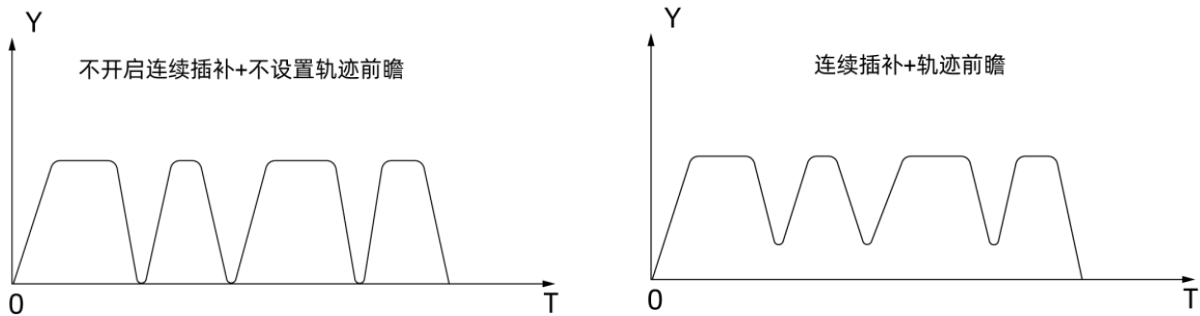


模式一：开启了连续插补后，得出的主轴速度随时间变化的曲线如下右图，主轴的速度是连续的，轨迹拐角处仍不减速，高速运行时拐角处冲击较大。



模式二：模式一条件下，关闭连续插补，得出的主轴速度随时间变化的曲线如下左图，每走完一段直线后便减速到 0 再开始第二段直线运动，加工效率不高。

模式三：模式一条件下，开启连续插补，并设置了轨迹前瞻参数，得出的主轴速度随时间变化的曲线如下右图，拐角处按照一定的比例减速，加工效率比模式二高。



以上模式若希望速度曲线更柔和，只需通过 **SRAMP** 指令设置速度为 **S** 曲线。

轨迹前瞻的主要指令 **CORNER\_MODE**，用于拐角处的速度规划，常用的三种模式，根据加工的轨迹的实际要求选择不同的模式。

此参数是在插补运动指令调用前生效，一般在参数初始化中设置拐角模式。因为前瞻运动参数会随着运动指令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，根据轨迹段的要求设置合适的前瞻模式，在执行运动指令的时候自动优化轨迹。

注意：**CORNER\_MODE** 模式一旦设置参数会存入控制器内，取消需要设置 **CORNER\_MODE=0**，否则下次运行时，之前设置的 **CORNER\_MODE** 仍会生效。

**CORNER\_MODE** 指令参数说明：

位	值	描述
0	1	预留
1	2	自动拐角减速。 按 <b>ACCEL</b> ， <b>DECEL</b> 加减速度。 此参数是在 <b>MOVE</b> 函数调用前生效。 减速角度根据 <b>DECEL_ANGLE</b> 和 <b>STOP_ANGLE</b> 指令设定。 减速拐角参考速度以 <b>FORCE_SPEED</b> 速度为参考，一定要设置合理的 <b>FORCE_SPEED</b> 。
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。

CORNER\_MODE=2 拐角减速应用场合：不改变运动轨迹，仅在拐角处自动判断是否减速，一般用于改善机台抖动的问题，对轨迹精度有要求，对速度要求没那么快的场合。

CORNER\_MODE=8 小圆限速应用场合：不改变运动轨迹，一般应用在圆弧加工，根据圆弧半径计算当前圆弧的限制速度。

CORNER\_MODE=32 自动倒角应用场合：改变运动轨迹，不会降低速度，针对轨迹拐角较大的场合，倒角处运动轨迹做自动平滑处理，所以一般应用在对速度要求快，对轨迹精度要求不高的场合。

轨迹前瞻的应用例程参见 [CONNER MODE](#) 指令。

## 5.4. 原点回零

相关指令：

指令	说明	用法
<a href="#">DATUM</a>	原点回零模式选择	DATUM（回零模式值）
<a href="#">DATUM_IN</a>	映射原点开关	DATUM_IN = 输入编号
<a href="#">FWD_IN</a>	映射正向限位开关	FWD_IN = 输入编号
<a href="#">REV_IN</a>	映射负向限位开关	REV_IN = 输入编号
<a href="#">SPEED</a>	快速找原点速度	设置速度值
<a href="#">CREEP</a>	找原点反向爬行速度	设置速度值
<a href="#">INVERT_IN</a>	输入信号反转	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，各轴按照设置回零的方式自动回零。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。

回零时机台需要接入原点开关（指示原点的位置）和正负限位开关（均为传感器，传感器检测到信号后，表示有输入信号，传给控制器处理）。

单轴找原点时，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。控制器正/负限位信号生效后，会立即停止轴，停止减速度为 FASTDEC。

ZMC 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。

DATUM 指令支持的回零模式如下表：

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

Z 信号回零必须配置为带 Z 信号 ATYPE。

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。

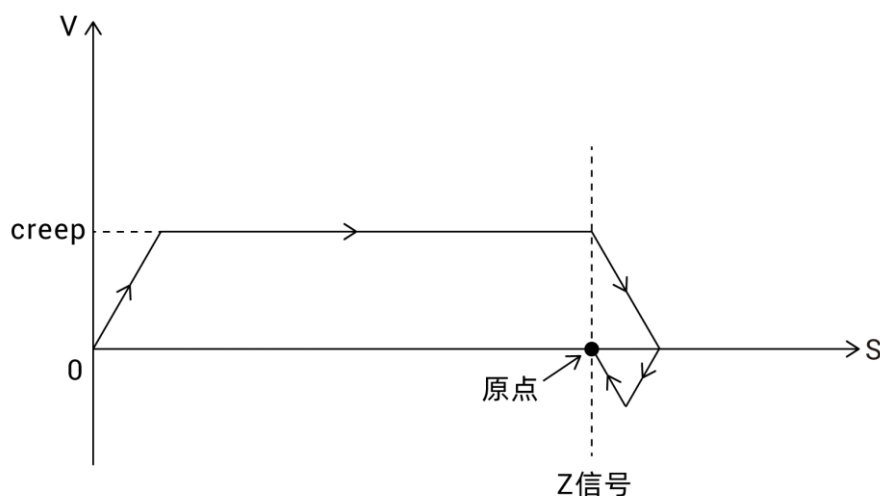
总线控制器使用以上控制器找原点模式完成后，需要手动清零 MPOS。回零模式加 100（模式 100+n 和 110+n 分别对应 n 和 10+n），表示接入编码器后可以自动清零 MPOS(仅限 4 系列，ATYPE=4)

常见回零方式详解：

### 1. 回零模式 1

如下图，DATUM(1)轴以 CREEP 速度正向运行,直到 Z 信号出现后开始减速，减速到 0 之后再反向回到 Z 信号处，此时将 DPOS 值重置为 0，停止后所处位置为原点，回零途中若碰到限位开关会直接停止。

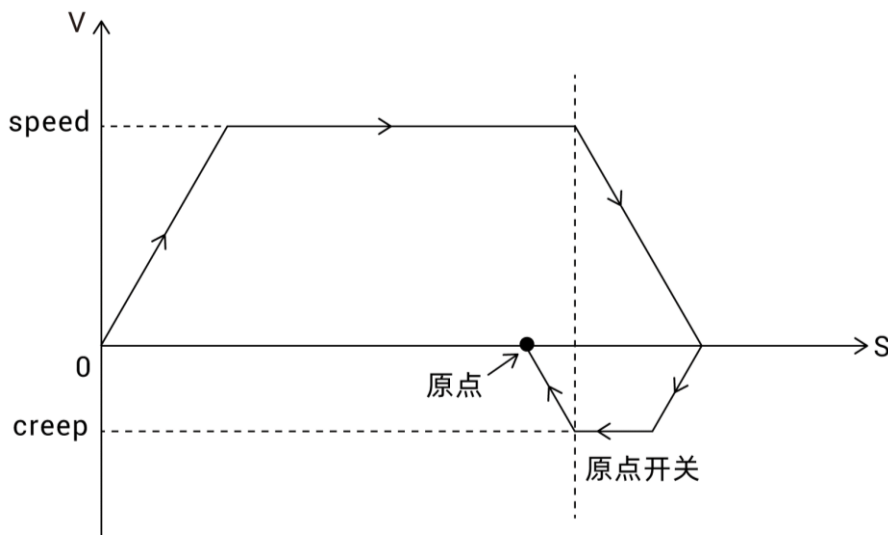
回零模式 2 与模式 1 找原点运动方向相反。



## 2. 回零模式 3

如下图，DATUM(3)轴以 SLEEP 速度快速正向运行,直到碰到原点开关后开始减速，减速到 0 之后再反向以 CREEP 速度找原点，再次碰到原点之后减速停止，轴停止之后将 DPOS 值重置为 0，当前所处位置为原点，回零途中若碰到限位开关会直接停止。

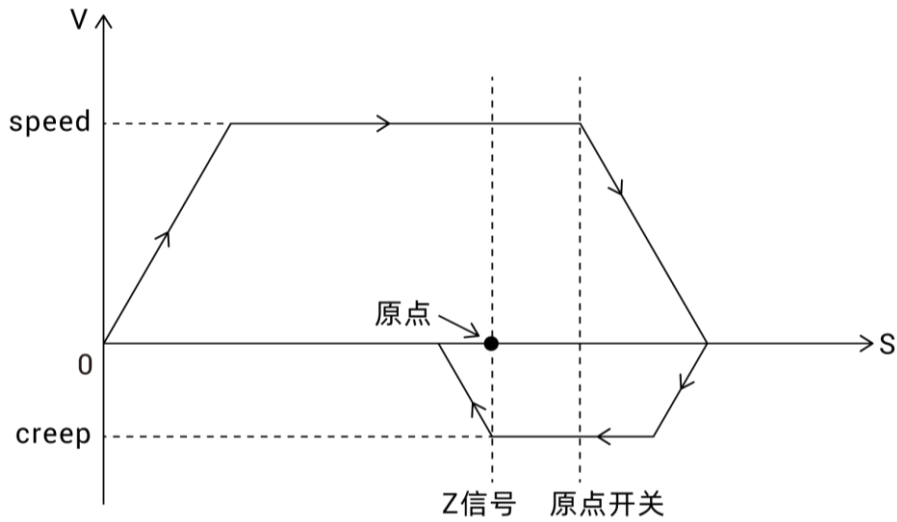
回零模式 4 与模式 3 找原点运动方向相反。



## 3. 回零模式 5

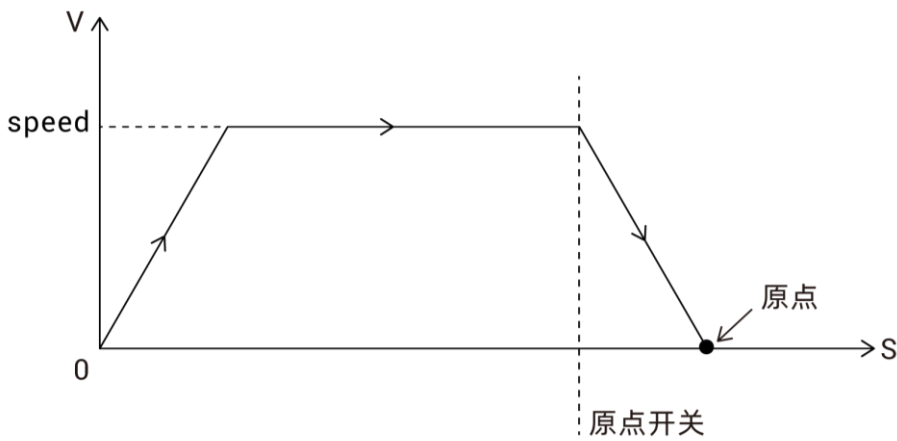
如下图，DATUM(5)轴以 SLEEP 速度快速正向运行,直到碰到原点开关后开始减速，减速到 0 之后再反向以 CREEP 速度运动，直到 Z 信号出现之后减速停止，遇到 Z 信号便将 DPOS 值重置为 0，Z 信号出现的位置为原点，回零途中若碰到限位开关会直接停止。

回零模式 6 与模式 5 找原点运动方向相反。



4. 回零模式 8

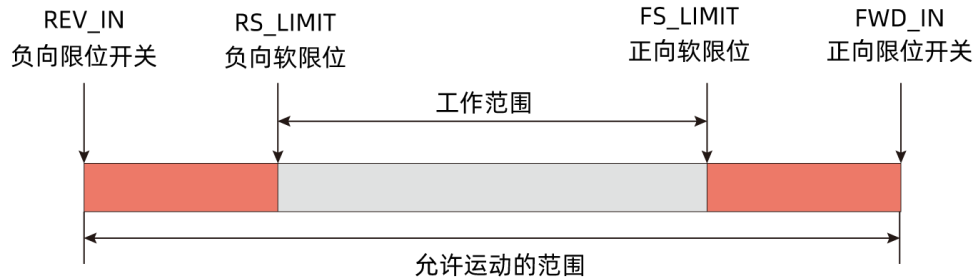
如下图，DATUM(8)轴以 SPEED 速度快速正向运行,直到碰到原点开关后开始减速，减速到 0 之后便将 DPOS 值重置为 0，停止后所处位置为原点，回零途中若碰到限位开关会直接停止。  
回零模式 9 与模式 8 找原点运动方向相反。



5.5. 限位相关指令

指令	说明	用法
<a href="#">FS_LIMIT</a>	正向软限位设置	FS_LIMIT=正限位位置
<a href="#">RS_LIMIT</a>	负向软限位设置	RS_LIMIT=正限位位置
<a href="#">FWD_IN</a>	映射正限位输入	FWD_IN=输入编号
<a href="#">REV_IN</a>	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采用减速度 **FASTDEC** 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。

轴在碰到限位的时候才会产生停止信号，此时由于减速需要一定的时间，实际轴的位置会越过限位一定距离，假设停止时 **SPEED** 速度是  $v_0$ ，快减速 **FASTDEC** 为  $a$ ，计算公式： $v_t^2 - v_0^2 = 2as$ ，带入下方数据： $0 - 100^2 = 2 * (-1000) * s$ ，得出过冲的距离  $s = 5$ ，由此可得，增大 **FASTDEC** 和减小 **SPEED** 都能达到减小过冲的目的。

示例：

```

BASE(0)           '选择轴 0
ATYPE=1           '轴类型设置
UNITS=100          '脉冲当量 100
SPEED=100          '速度 100units/s
ACCEL=500          '加速度
DECEL=500          '减速度
FASTDEC=1000       '快减速 100units/s/s
DPOS=0             '坐标清 0
FS_LIMIT=200       '设置正向软限位 200units
MOVE(300)          '运动 300units
WAITIDLE(0)        '等待轴停止
?DPOS(0)           '打印结果：205units
  
```

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 "SOFTLIMIT 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 "SOFTLIMIT RS", *DPOS
RS_LIMIT = -200000000 '关闭软限位
FS_LIMIT = 200000000
END

```

打印结果：

```

Axis:0 AXISSTATUS:200h,FSOFT
SOFTLIMIT FS    101
Axis:0 AXISSTATUS:400h,RSOFT
SOFTLIMIT 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。





## 5.6. 位置锁存

相关指令：

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

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。

脉冲轴类型一般采用 R0, R1, Z 脉冲这三种锁存；总线轴类型一般采用 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 REG\_INPUT，需要输入口支持锁存功能；
- 3) 设置锁存模式 REGIST，等待锁存触发 MARK；
- 4) 锁存完成打印锁存位置信息 REG\_POS；
- 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。

控制器锁存方式参见 [REGIST](#) 指令描述。



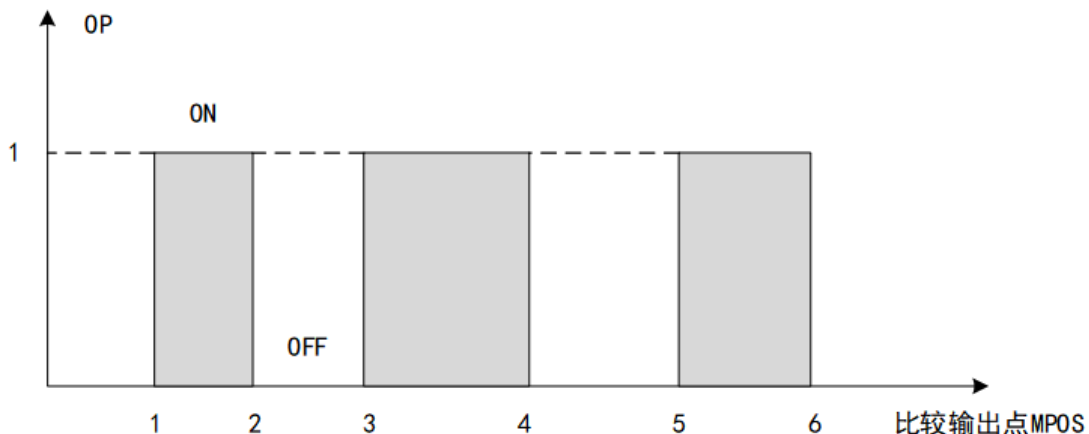
## 5.7. 硬件比较输出

相关指令：

指令	说明	用法
<a href="#">HW_PSWITCH</a>	硬件位置比较输出	设置比较点
<a href="#">HW_PSWITCH2</a>	总线硬件位置比较输出	设置比较点和比较输出口
<a href="#">HW_TIMER</a>	硬件定时输出	周期输出

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，指定输出口电平翻转，到达位置 2 电平再次翻转，到达位置 3 电平再翻转，直到比较完所有的点后，电平维持最后一次翻转后的状态。



硬件比较输出需要 3 系列部分型号和 4 系列及以上控制器支持，需要操作支持此功能的输出口，控制器支持软件比较输出 [PSWITCH](#) 指令，硬件比较输出 [HW\\_PSWITCH](#) 指令（仅支持脉冲轴），[HW\\_PSWITCH2](#) 指令（脉冲轴、总线轴均支持）。

对于脉冲轴，[HW\\_PSWITCH](#) 和 [HW\\_PSWITCH2](#) 的区别是，[HW\\_PSWITCH](#) 的轴和输出有一一对应的关系，不需要指定输出轴号；[HW\\_PSWITCH2](#) 可以在支持该功能的输出口中指定。[HW\\_PSWITCH](#) 指令同一时间可操作多个输出口同时输出。[HW\\_PSWITCH2](#) 指令支持的控制器型号更多。

比较编码器的反馈 [MPOS](#) 位置精度更高，接入了编码器（脉冲轴轴类型 [ATYPE](#) 为 4 或总线轴类型）便比较编码器反馈位置 [MPOS](#)，没有接入编码器时（脉冲轴轴类型 [ATYPE](#) 为 1 或 7）比较目标位置 [DPOS](#)。

若比较位置为大量连续的等间距输出，可使用 [HW\\_TIMER](#) 硬件定时输出，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，使用 [HW\\_PSWITCH](#) 和 [HW\\_PSWITCH2](#) 指令，[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。

## 5.8. 精准输出

相关指令：

指令	说明	用法
<a href="#">MOVE_OP</a>	缓冲中输出	MOVE_OP (编号,状态)
<a href="#">AXIS_ZSET</a>	开启精准输出	按位设置功能
<a href="#">MOVEOP_DELAY</a>	缓冲输出延时	可提前或延时输出

MOVE\_OP 指令默认为普通输出，普通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，能大大提高工艺的精度，同时可以使用 MOVEOP\_DELAY 指令调整响应时间（提前或延迟）。

精准输出脉冲输出方式的最小误差 1 个脉冲，总线控制方式的最小误差 1us 以内。

支持硬件比较输出功能的控制器才可使用精准输出功能，二者使用同样的硬件资源。

使用 AXIS\_ZSET 指令设置是否开启精准输出，开启后再使用 MOVE\_OP 指令精准输出生效，注意输出通道要选择支持精准输出的通道，也就是支持硬件比较输出的通道，不同型号个数有所区别，一般特殊功能从 IO 编号 0 开始。

精准输出触发方式有两种，目标位置 DPOS 触发或编码器反馈 MPOS 触发。

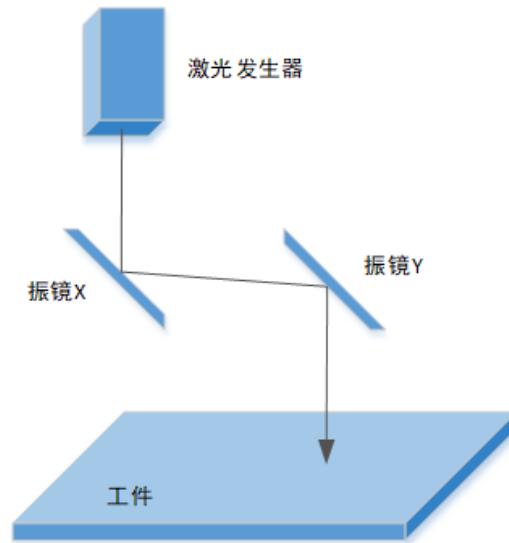
当不带编码器反馈时，精准输出功能自动使用命令位置 DPOS 来比较触发，电机总是有一定跟随误差的（跟随误差=DPOS-MPOS），带编码器反馈时使用编码器反馈 MPOS 触发会更准确，是否启动编码器位置同样也是通过 AXIS\_ZSET 指令配置。根据不同的驱动器差异性效果，也可以使用 MOVEOP\_DELAY 参数来调整 IO 输出的准确时刻。

## 5.9. 振镜控制系统

### 5.9.1. 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。

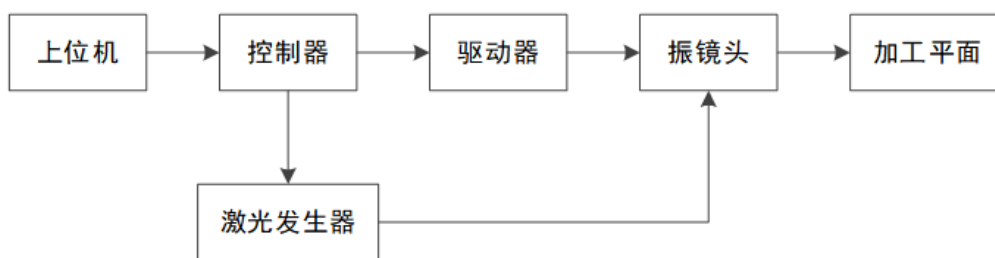


激光振镜运动两种基本的运动：一种为跳转运动，一种为打标运动。

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，其转子上通过机械组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。

振镜运动采用缓冲区运动方式，即用户需要向轴运动缓冲区传递运动及工艺数据，然后启动缓冲区运动，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。

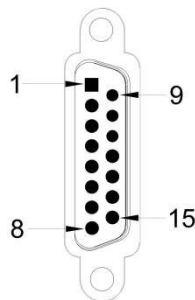
工作方便，防伪能力强。

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+

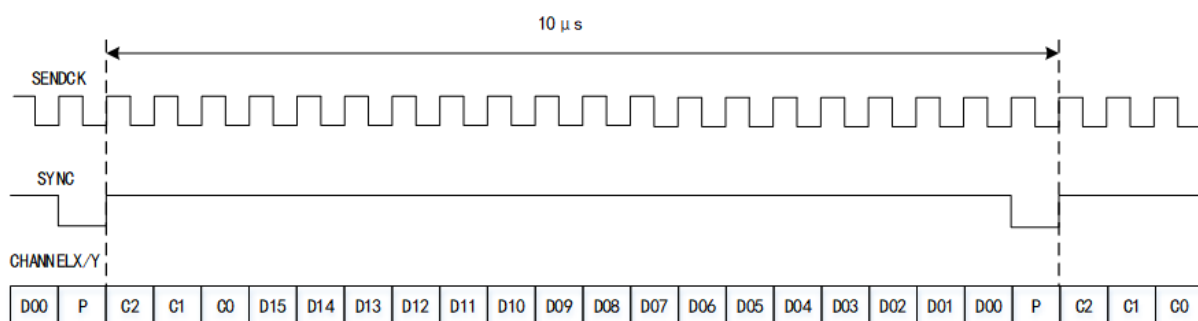
11	X channel+	振镜 X 通道信号+
12	Y channel+	振镜 Y 通道信号+
13	NC	保留
14	STATUS+	振镜状态信号+
15	GND	数字地

## 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。

## 5.9.2. 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以下控制方式：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：

针脚	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)

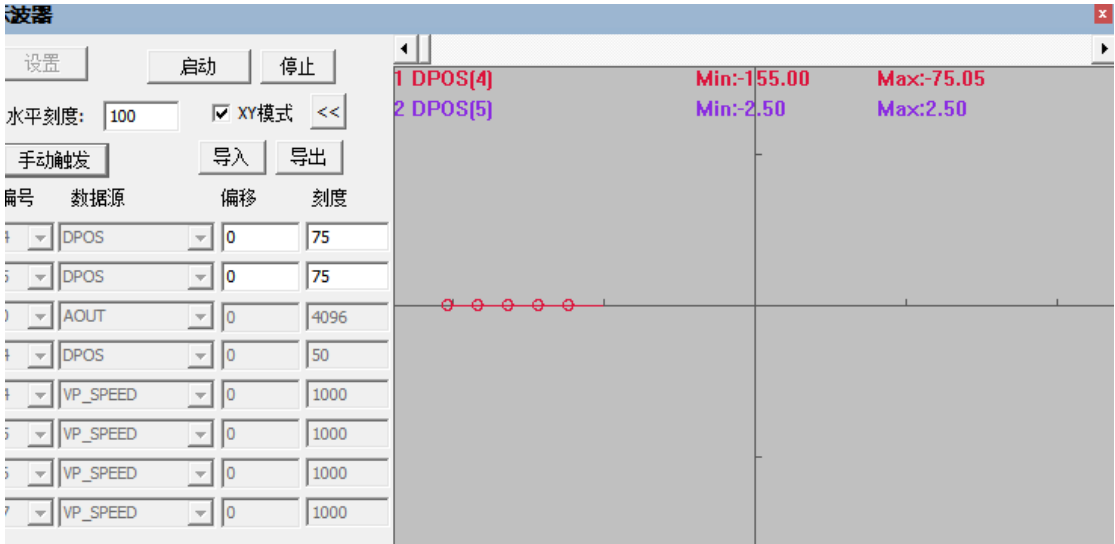


```
ATYPE=21,21
'设置基本参数
UNITS=300,300
SPEED=5000,5000
ACCEL=SPEED*20,SPEED*20
DECEL=SPEED*20,SPEED*20
MOVEABS(0,0)
FORCE_SPEED=5000
'打开连续插补
MERGE=ON
AXIS_ZSET(4)=3    '开启 MOVE_OP 的精准输出功能
'设置频率
PWM_FREQ(2)=2000
WHILE 1
    IF MODBUS_BIT(0)=ON THEN
        MODBUS_BIT(0)=OFF
        OP(0,OFF)
        BASE(4,5)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

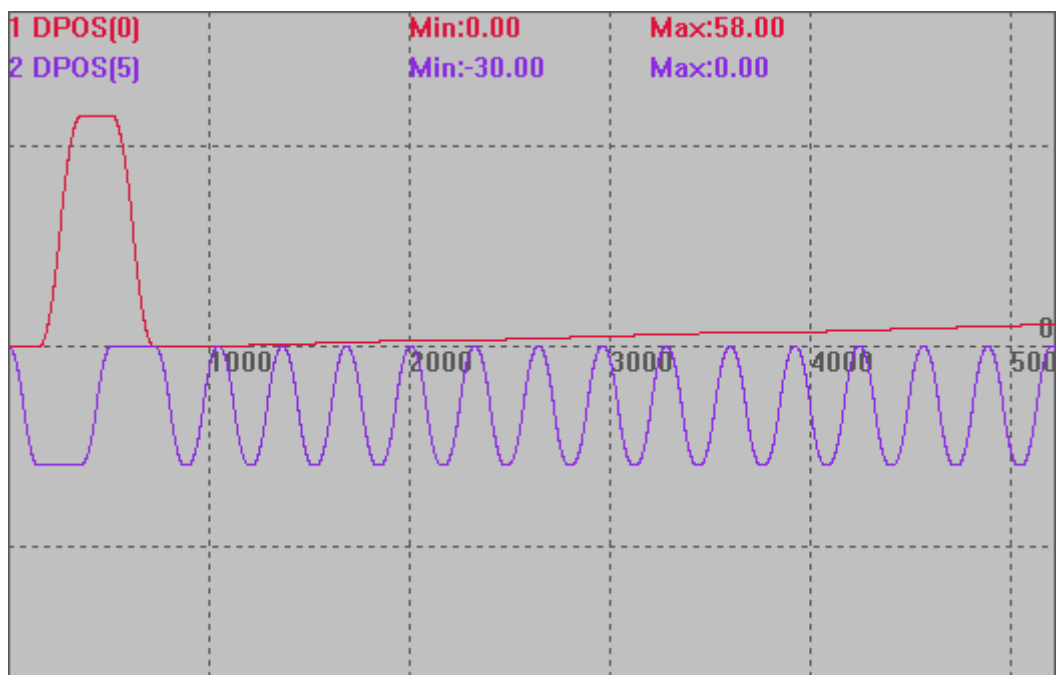
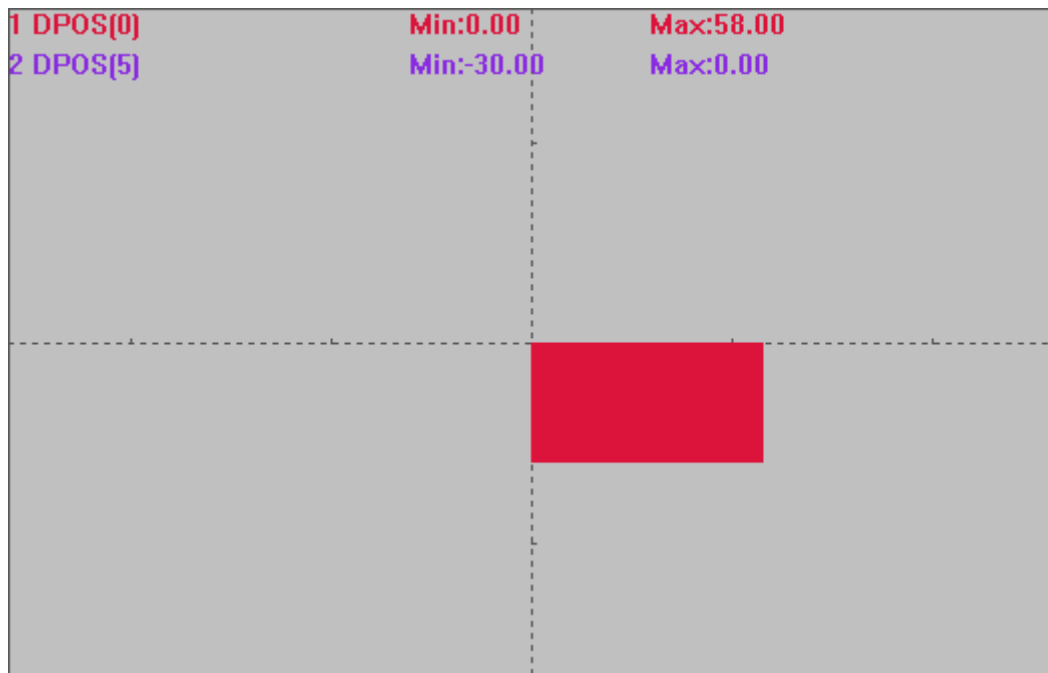
        MOVE_OP(0,ON)
        TRIGGER

        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:



## 5.10. 机械手

正运动控制器支持三十多种机械手算法，根据机械手的类型 **frame** 建立机械手连接后使用，能平滑、精准的控制机械手运动，详细使用说明参见“正运动机械手指令手册”。

### 5.10.1. 机械手相关概念

#### 1. 关节轴与虚拟轴

##### 1) 关节轴

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。

##### 2) 虚拟轴

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绕轴的三个旋转轴，用来确定机械手末端工作点的加工轨迹与坐标。

#### 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

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。

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。

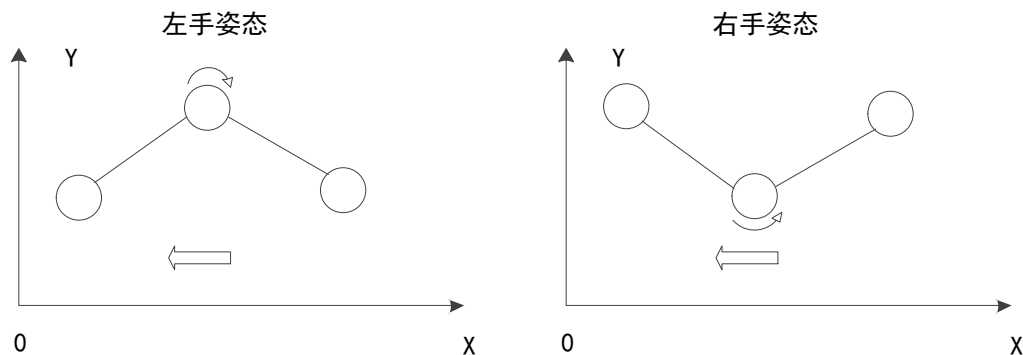
坐标系转换是指在描述同一个空间时，由原来的坐标系转换为另一个坐标系。机械手使用中，常用于确定工件坐标系。

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若干工件坐标系，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。

#### 3. 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。



#### 4. 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。

### 5.10.2. 正解运动与逆解运动

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。

#### 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。

## 5.10.3. 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。

## 5.10.4. 机械手应用举例

一般来说，生产加工时选择逆解模式，通过给虚拟轴发送坐标位置，控制机械手运动，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。

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. 建立机械手正逆解连接。

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

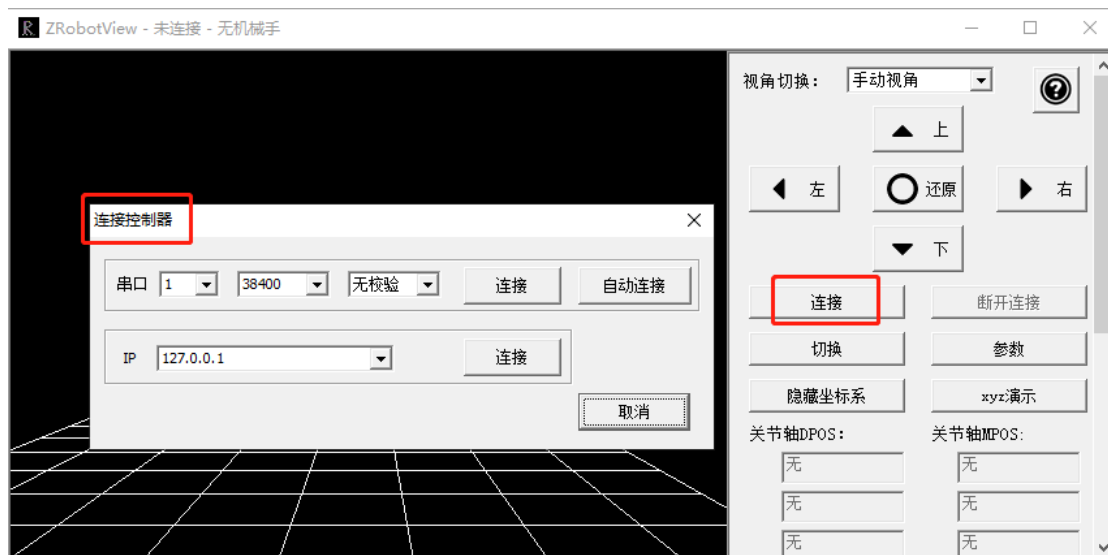
```

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"

```

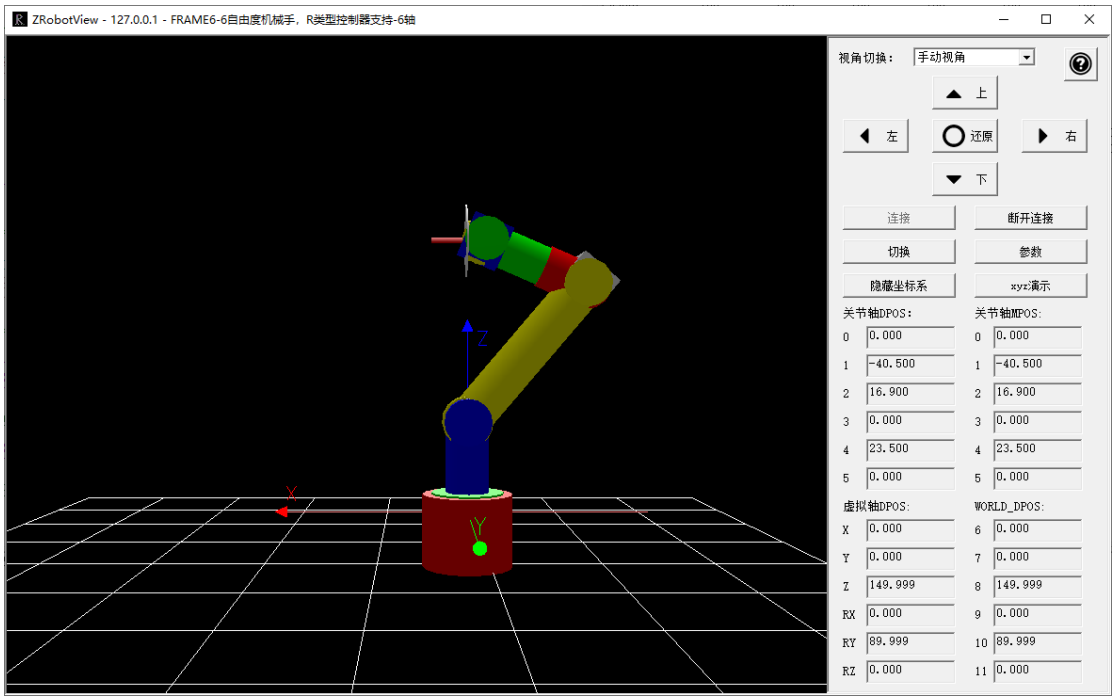
```
ELSEIF SCAN_EVENT(IN(0))<0 THEN '输入 0 下降沿触发  
  BASE(6,7,8,9,10,11) '选择虚拟轴号  
  CONNREFRAME(6,0,0,1,2,3,4,5) '启动正解连接。  
  WAIT LOADED '等待运动加载。  
  ?"正解模式"  
ENDIF  
WEND  
END
```

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，先建立正解或逆解连接（不建立无法在 ZRobotView 软件上显示机械手），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：





## 5.11. 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，简称 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情况。

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](#)”。

## 第六章 轴相关说明

### 6.1. 轴的概念

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	使用控制器的脉冲轴接口、EtherCAT 总线或 RTEX 总线接口和驱动器连接，给设备分配轴号，将 1 台电机作为 1 个轴使用。
虚拟轴	运动控制器内的虚拟轴，不使用实际驱动器，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。

电机轴：主动运行，电机根据控制器发出的脉冲来运动，发送的脉冲数根据运动参数变化量\*UNITS 确定，目标需求位置由 DPOS 参数体现。

编码器轴：被动运行，编码器跟随电机转动，产生脉冲，反馈控制器，控制器接收的脉冲数查看 ENCODER 指令确定，编码器反馈位置由 MPOS 参数体现。

### 6.2. 轴号说明

#### 1. 脉冲轴号

脉冲电机轴：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。发送的脉冲数根据运动参数变化量\*UNITS 确定，目标需求位置由 DPOS 参数体现。

编码器轴：被动运行，跟随电机转动，产生脉冲，反馈控制器，控制器接收的脉冲数查看 ENCODER 指令确定，编码器反馈位置由 MPOS 参数体现。

使用脉冲轴的时候，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轴号）



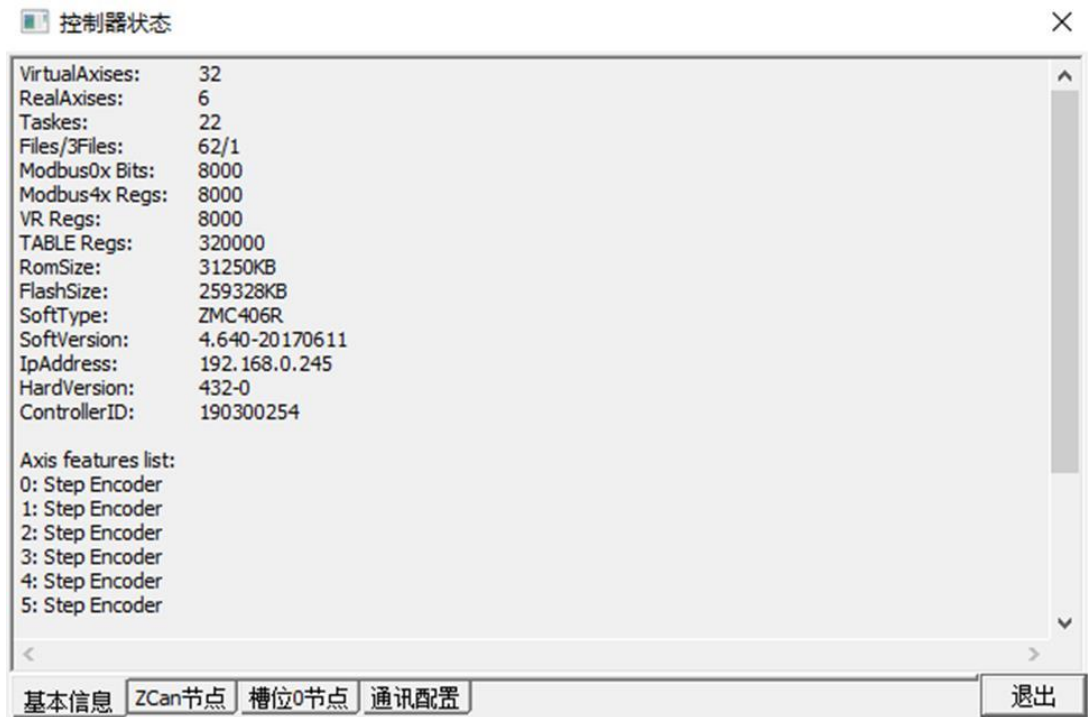
以下方控制器状态为例，每个脉冲轴接口支持接什么类型的轴参见“控制器状态”窗口的 Axis features list 描述，Step 为脉冲输出，Encoder 为编码器反馈。

轴号后备注为“Step Encoder”的就可以配置为既有脉冲输出 Step 又有编码器反馈输入 Encoder，当 ATYPE=4 时满足脉冲输出和编码器反馈在同一个轴号上，此时 DPOS 和 MPOS 都是真实的。当 ATYPE=1 或 7 时，此时只有脉冲输出，接入的编码器反馈在其他轴号上，规则参见下文，DPOS 为真，MPOS 复制 DPOS 的值。

轴号后面只有“Encoder”的就是反馈轴所占的轴号，例如轴 6，反馈轴默认的 ATYPE 都是 3（ATYPE 为 3 为时对应正交编码器，可以改为 6，对应脉冲方向型编码器）。

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. 总线轴号

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，电机轴和编码器共用一个轴号。

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

6.3. 轴状态

`AXISSTATUS` 指令查看轴的各种状态。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 MAX_SPEED	4096	1000h
14	机械手指令坐标错误	16384	4000h
18	电源异常	262144	40000h
19	精准输出缓冲溢出	524288	80000h
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

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。

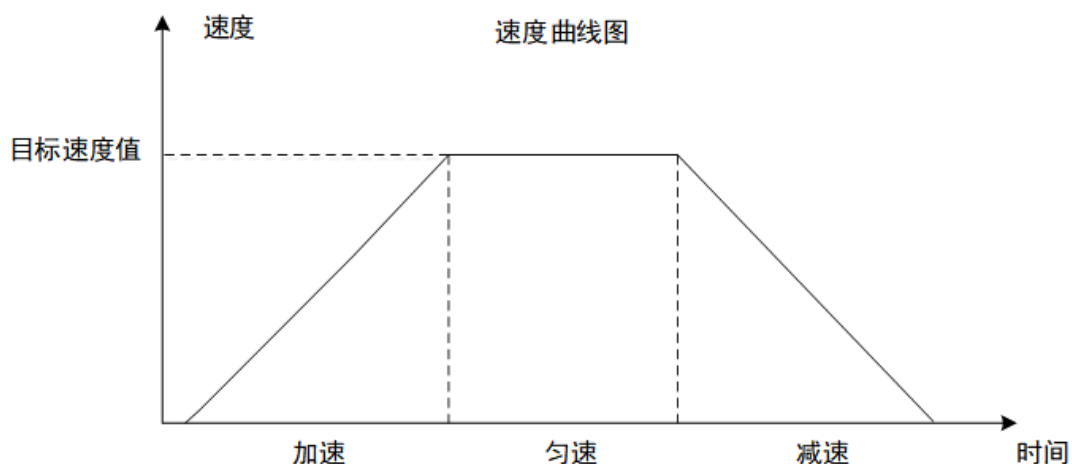
IDLE 指令用于判断加在轴上的运动指令是否完成，运动中返回 0，运动结束返回-1，程序中一般使用 WAIT IDLE(轴号)语句判断轴状态。

MTYPE 指令用于判断轴当前的运动类型，例如 MTYPE 返回值为 1，表示正在进行 MOVE 运动。

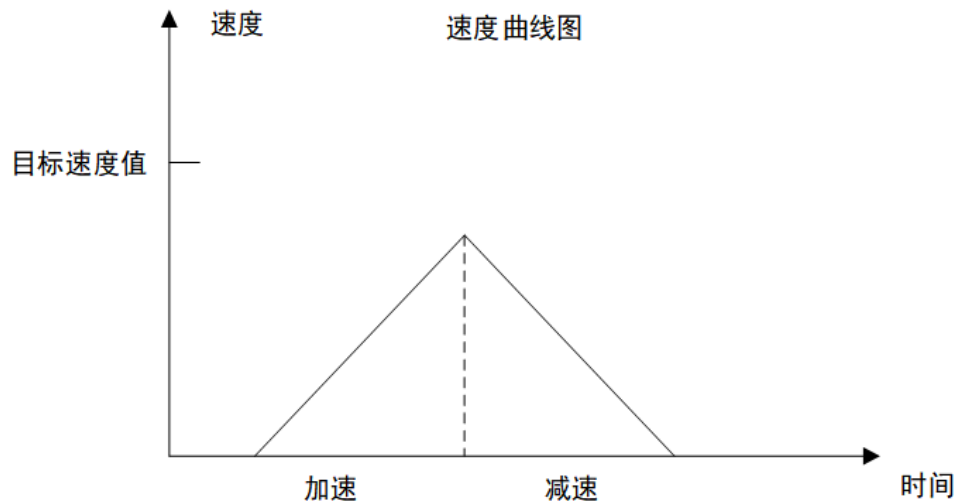
## 6.4. 轴速度

### 6.4.1. 速度曲线

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。



常用的速度指令有 SPEED 运动速度，ACCEL 加速度，DECEL 减速度，FASTDEC 快减速等，在轴参数初始化时设置完成，作为运动指令的速度参考。

### 1. 梯形图曲线

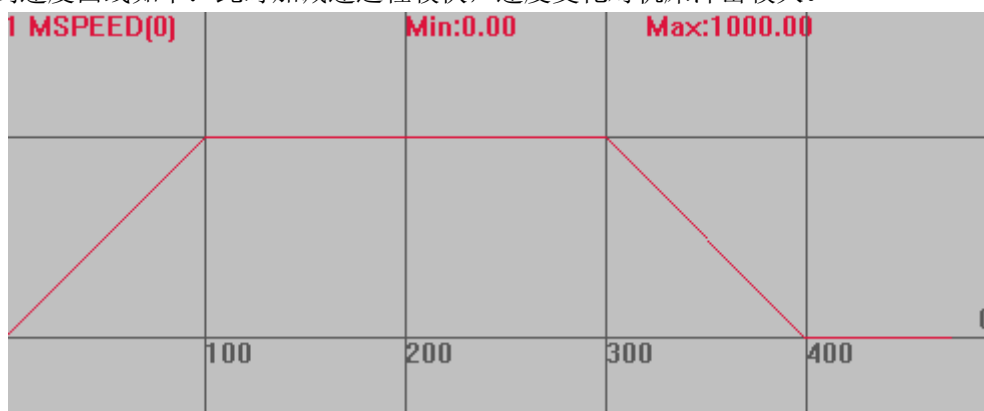
不设置 SRAMP（令 SRAMP 等于 0），速度曲线为梯形曲线，在此速度规划方式下，速度曲线按梯形曲线变化。保持速度、加减速等参数值不变。

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(0)
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

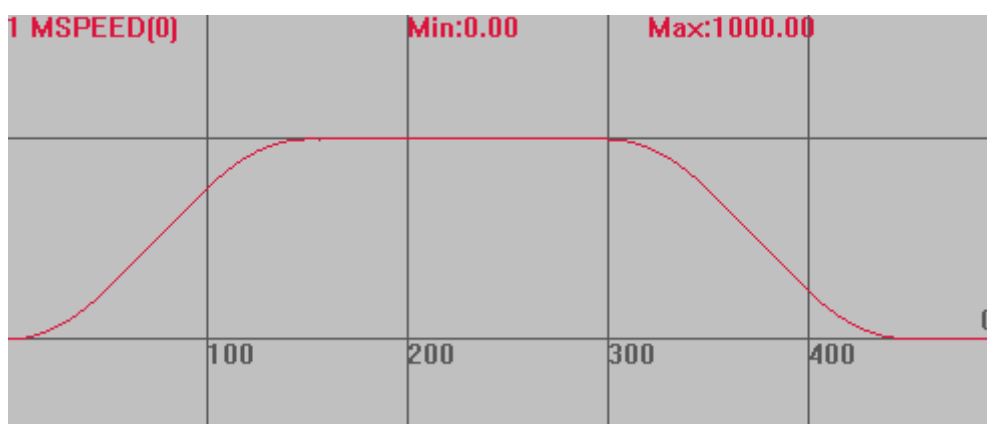
通过设置 SRAMP 的值来设定合适的加减速变化率，使得速度曲线平滑，在机械启停或加减速时减少抖

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。

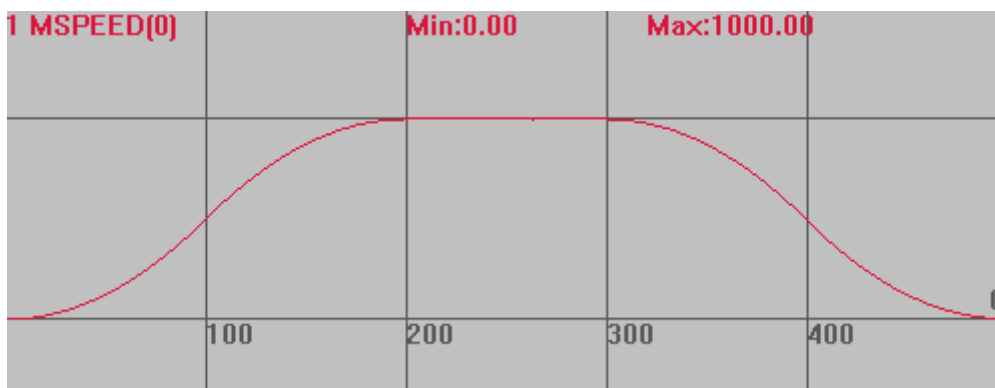
例程：

```
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：加减速时更柔和。



当 SRAMP=100 时，得到 s 型曲线如下：加减速过程变长。



## 6.4.2. SP 速度

SP 速度应用于插补运动指令带 SP 后缀之后的指令（例如 MOVESP、MOVECICRSP），此时运动速度采用 FORCE\_SPEED 参数，而不是 SPEED 参数。

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。

RAPIDSTOP(2)

WAIT IDLE(0)

WAIT IDLE(1)

BASE(0,1) '选择 XY 轴

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

SRAMP=100,100 'S 曲线

MERGE = ON '启动连续插补

TRIGGER

'第一段

FORCE\_SPEED = 50 '第一段速度 50

STARMOVE\_SPEED = 20 '第一段起始的速度 20

ENDMOVE\_SPEED = 10 '第一段结束的速度 10

MOVESP(40,40)

'第二段

FORCE\_SPEED = 60 '第二段速度 60

STARMOVE\_SPEED = 30 '第二段的起始速度 30

ENDMOVE\_SPEED = 40

MOVESP(50,50)

'第三段

FORCE\_SPEED = 80 '第三段速度 80

STARMOVE\_SPEED = 30 '第三段的起始速度 30

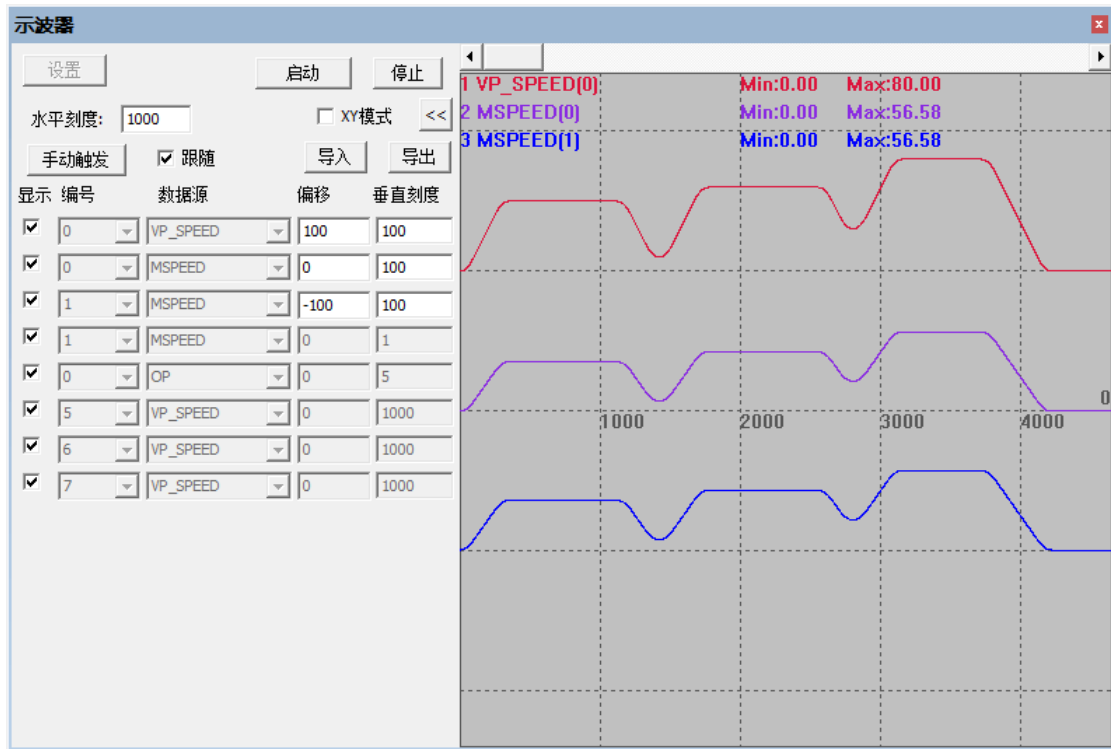
ENDMOVE\_SPEED = 20

MOVESP(60,60)

END

速度变化曲线：从速度为 0 开始运动，第一段的 STARMOVE\_SPEED = 20 不起作用，第一段结束速度 ENDMOVE\_SPEED = 10 表示速度降为 10 后第一段运动完成；第二段运动实际从速度 10 开始运动，到 ENDMOVE\_SPEED = 40 结束；第三段的起始速度 STARMOVE\_SPEED = 30，小于第二段结束速度 40，第二段完成后速度会降到 30，第三段完成后后面没有运动指令了，所以速度降为 0，ENDMOVE\_SPEED 不起作用。





## 6.5. 轴映射

使用控制器的本地脉冲轴时，是不需要轴映射的，采用默认的轴编号即可，参见[轴号说明](#)章节。

使用总线轴和扩展的脉冲轴时，均需要先映射绑定轴号之后才能使用，脉冲轴若要更改默认轴号，支持重新映射配置轴号，映射轴号均采用 [AXIS ADDRESS](#) 轴映射指令，不同类型的轴映射的语法不同。

轴号可以随意映射，但要在控制器支持的轴数范围内，且映射的轴编号不能重复，一般按顺序依次映射，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。

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。

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；

配置之前要将 AXIS 0-3 设成虚拟轴 ATYPE=0，再使用 [AXIS\\_ADDRES](#) 指令映射驱动器的轴号，配置完成之后根据轴的特性配置 ATYPE，后续便可对轴 0-3 发送命令。

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。

注意，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。

## 6.6. 轴类型

轴类型使用 `ATYPE` 指令根据当前轴的特性去配置，在用户程序初始化时应第一时间完成轴类型的配置，类型不匹配会报错。

所有未分配的轴默认为虚拟轴，`ATYPE` 的值为 0。

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
20	振镜类型，带振镜状态反馈 振镜连接不上 <code>AXISSTATUS</code> 的 bit2 会置位，ENCODER 返回原始的发送位置，脉冲单位 ZMC408SCAN 支持
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
22	振镜轴类型，带振镜位置反馈 振镜连接不上 <code>AXISSTATUS</code> 的 bit2 会置位，振镜报警 <code>AXISSTATUS</code> 的 bit3 会置位 MPOS 返回反馈位置，做了反矫正处理，ENCODER 返回原始的反馈位置脉冲单位 ZMC408SCAN 支持
24	远程编码器轴类型 ZHD500X 上手轮使用，需要控制器 5 系列 20180404 以上固件版本支持
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 <code>DRIVE_PROFILE</code> 要设置为 20 或以上
67	EtherCAT 周期力矩模式，需支持 EtherCAT <code>DRIVE_PROFILE</code> 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

### 1. ATYPE=0 虚拟轴

多轴同步运动时可作为机器的主轴，从轴都跟随此虚拟主轴。

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 ADDAX 指令（轴叠加）进行设置，然后将各个虚拟轴的运动叠加到实轴。

### 2. ATYPE=1 或 7 脉冲轴

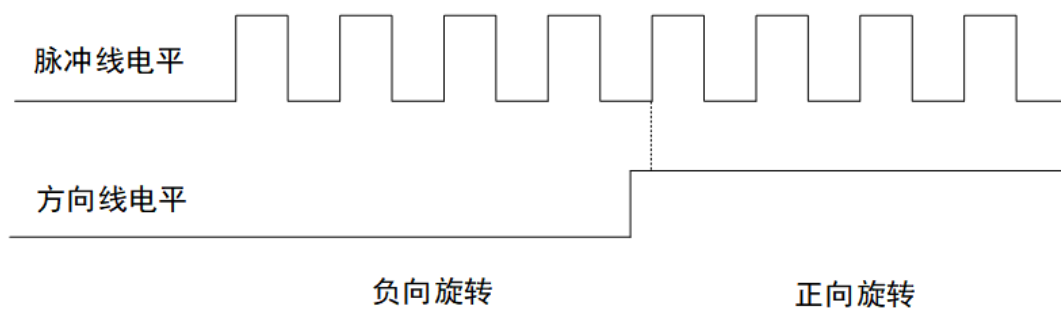
轴的运动由控制器发脉冲控制，脉冲的方向决定电机的转向，根据发送脉冲的频率控制轴运动的快慢。

控制器脉冲输出的三种模式：脉冲+方向、双脉冲、正交脉冲，采用 INVERT\_STEP 指令配置，默认是脉冲+方向模式。

#### 1) 脉冲方向模式：

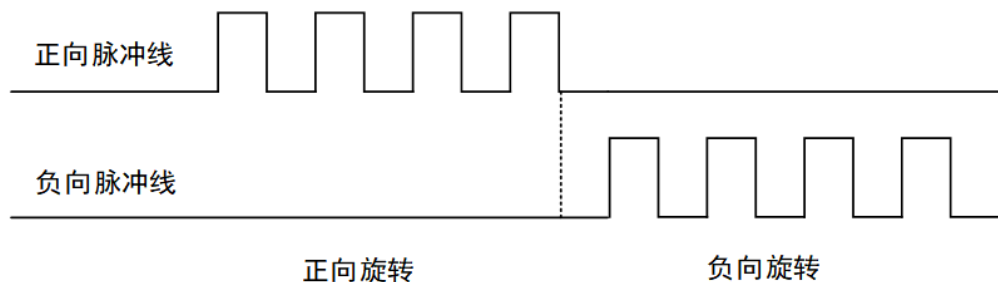
PUL+、PUL-输出指令脉冲串，脉冲数对应电机运行距离，而脉冲频率对应电机运行速度。

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

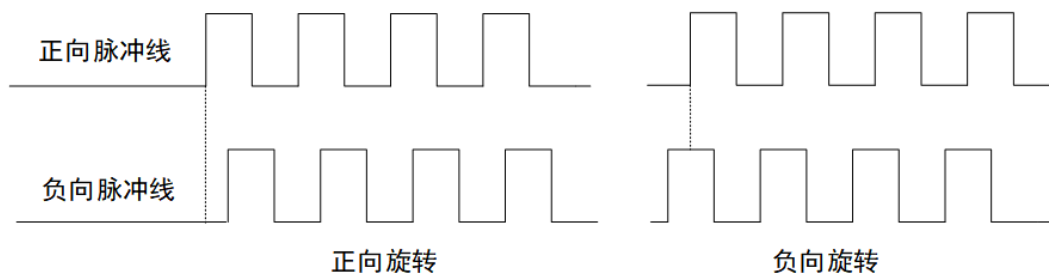
两根线都输出脉冲信号，CW 为正方向脉冲信号，CCW 为反方向脉冲信号，通常都是差分方式输出，两信号相位相差角度，根据相位超前或滞后来决定。



#### 3) AB 相：正交脉冲工作方式

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。

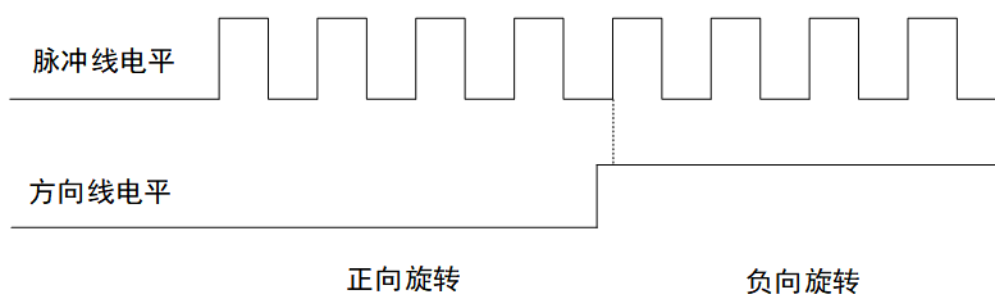
通过两个脉冲之间的相位差来达到计数或编码的作用。



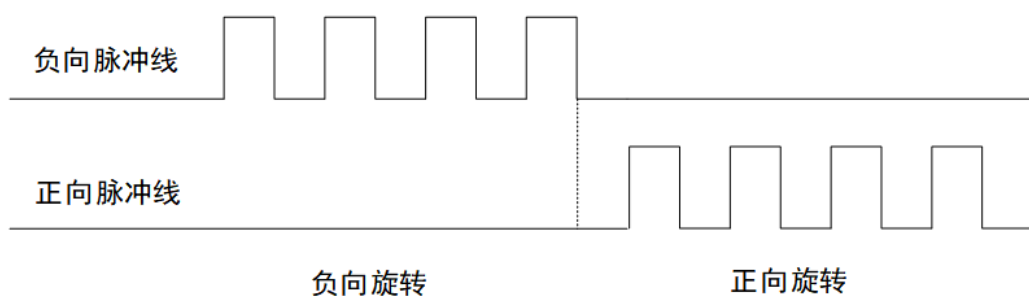
### 极性对调

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：



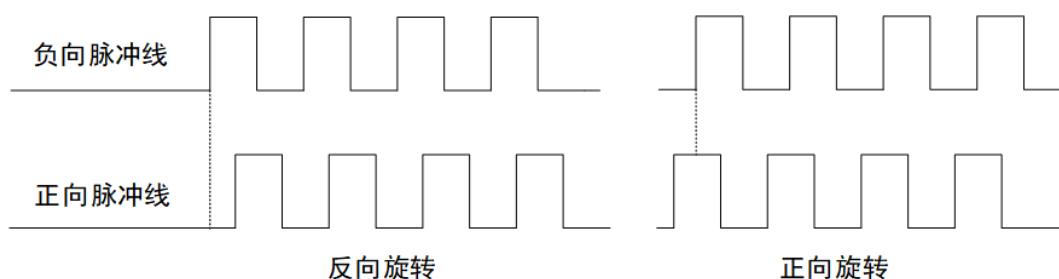
#### 2) CW/CCW：双脉冲工作方式



#### 3) BA 相：正交脉冲工作方式

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（4 系列及以上系列支持）。

### 3. ATYPE=3 或 6 编码器轴

编码器单独占用一个轴号时，根据编码器类型，轴类型可选 3 或 6。

### 4. ATYPE=4 脉冲轴和编码器轴共用轴号

当前的脉冲轴带编码器反馈时，轴类型设为 4，脉冲输出和编码器输入信号在同一个轴上。

### 5. ATYPE=8 CAN 扩展轴

使用 CAN 总线扩展轴时，扩展的脉冲轴的轴类型设成 8，扩展轴上接入编码器轴的轴类型设为 9。

### 6. ATYPE=21 振镜轴号

接入激光振镜设备时，需要将振镜轴类型设为 21，激光振镜轴部分型号支持。

### 7. ATYPE=50,51,52 RTEX 总线轴号

使用 RTEX 总线驱动器时，轴类型只能在以上三个中选择，其中 ATYPE=50 是位置模式，使用运动指令控制电机运行；ATYPE=51 速度模式，速度模式下使用 DAC 指令设置电机的运行速度，并持续运行；ATYPE=52 力矩模式，力矩模式下使用 DAC 指令设置电机的力矩，并持续运行，速度和力矩模式下不能使用运动指令，故无需设置轴参数，将 DAC=0 停止运行。

速度和力矩模式下若要切换模式，为防止事故，先将 DAC 置 0 后再使用 ATYPE 指令切换。

注意：修改 ATYPE 切换到力矩模式之前，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。再通过参数 Pr3.17 设置速度限制。当 Pr3.17(速度限制选择)的设定值是 0 时，通过 Pr3.21 设置速度限制，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采用 Pr3.21 还是 Pr3.22。

### 8. ATYPE=65,66,67 EtherCAT 总线轴号

使用 EtherCAT 总线驱动器时，轴类型只能在以上三个中选择，其中 ATYPE=65 是位置模式，使用运动指令控制电机运行；ATYPE=66 速度模式，速度模式下使用 DAC 指令设置电机的运行速度，并持续运行，速度单位有两个，脉冲数/S 和 R/MIN，由驱动器确定；ATYPE=52 力矩模式，力矩模式下使用 DAC 指令设置电机的力矩，并持续运行，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%，速度和力矩模式下不能使用运动指令，故无需设置轴参数，将 DAC=0 停止运行。

速度和力矩模式下若要切换模式，为防止事故，先将 DAC 置 0 后再使用 ATYPE 指令切换。

## 第七章 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。

### 7.1. 单轴运动指令

#### ADDAX -- 运动叠加

类型	单轴运动指令
描述	<p>运动叠加，把一个轴的运动叠加到另一个轴。</p> <p><b>ADDAX 指令叠加的是脉冲个数，而不是设置的 units 单位。</b></p> <p>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。</p> <p>假设轴 A 的 UNITS 是 100，轴 B 的 UNITS 是 50，叠加轴运动 100</p> <p>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 <math>100 \times 100 / 50 = 200</math>。</p> <p>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 <math>100 \times 50 / 100 = 50</math>。</p> <p>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。</p> <p>支持串联叠加，A 运动叠加到 B，B 在叠加到 C。</p> <p>支持并联叠加，A 运动同时叠加到 B、C。</p> <p>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。</p> <p><b>ADDAX 在轴 MTYPE 为 FRAME 或 REFRAME 的时候不起作用。</b></p>
语法	ADDAX(axis)
适用控制器	通用

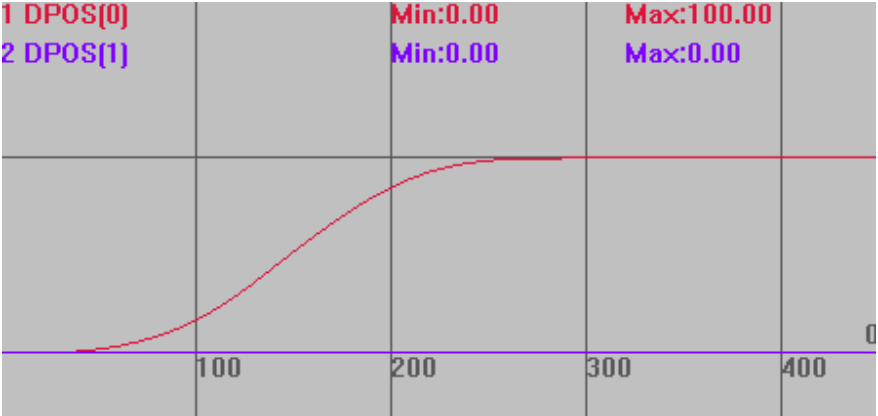
例子

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  $100 \times 100 / 200 = 50$   
                          '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

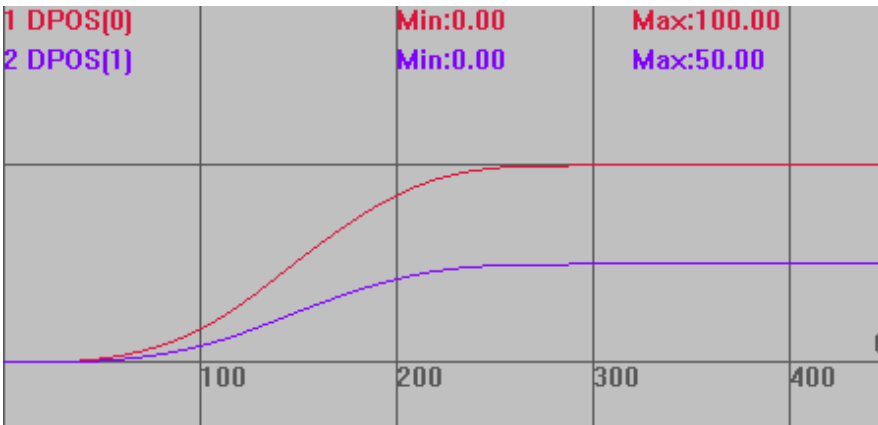
DPOS(1)垂直刻度 100



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

类型	单轴运动指令								
描述	<p><b>BASE 轴减速停止，如果轴参与插补，也停止插补运动。</b></p> <p>如果指定轴在 <b>BASE</b> 轴列表中，无论 <b>CANCEL</b> 主轴或者 <b>BASE</b> 轴列表中的轴，都停止轴组的插补运动</p> <p><b>MODE0~2</b> 减速度按 <b>FASTDEC</b> 和 <b>DECEL</b> 中最大的值。</p> <p><b>CANCEL</b> 后要调用绝对位置运动，必须先 <b>WAIT IDLE</b> 等待停止完成。</p>								
语法	<p>CANCEL(mode)</p> <p>mode: 模式选择</p> <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>	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
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	<p>例一</p> <p>BASE(0)</p> <p>DPOS=0</p> <p>SRAMP=0</p> <p>ATYPE=1</p> <p>UNITS=100</p> <p>SPEED=1000</p> <p>ACCEL=1000</p> <p>DECEL=1000                    '减速度设为 1000</p> <p>FASTDEC=10000                '快减减速度设为 10000</p> <p>TRIGGER                        '自动触发示波器</p> <p>MOVE(1000)                    '当前运动</p> <p>MOVE(-1000)                  '缓冲运动</p> <p>CANCEL(1)                      '此时轴只执行完 MOVE(1000)</p> <p>运动轨迹</p> <p>MSPEED(0)垂直刻度 1000</p> 								



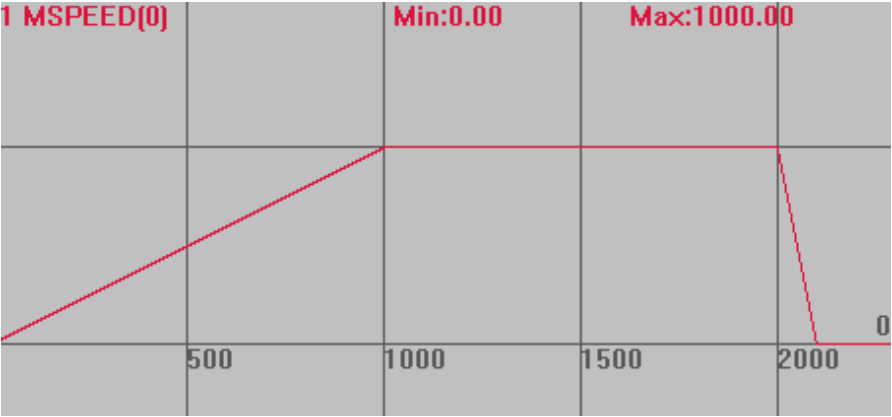
因为取消的是缓冲，当前运动还是按减速度正常停止。

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

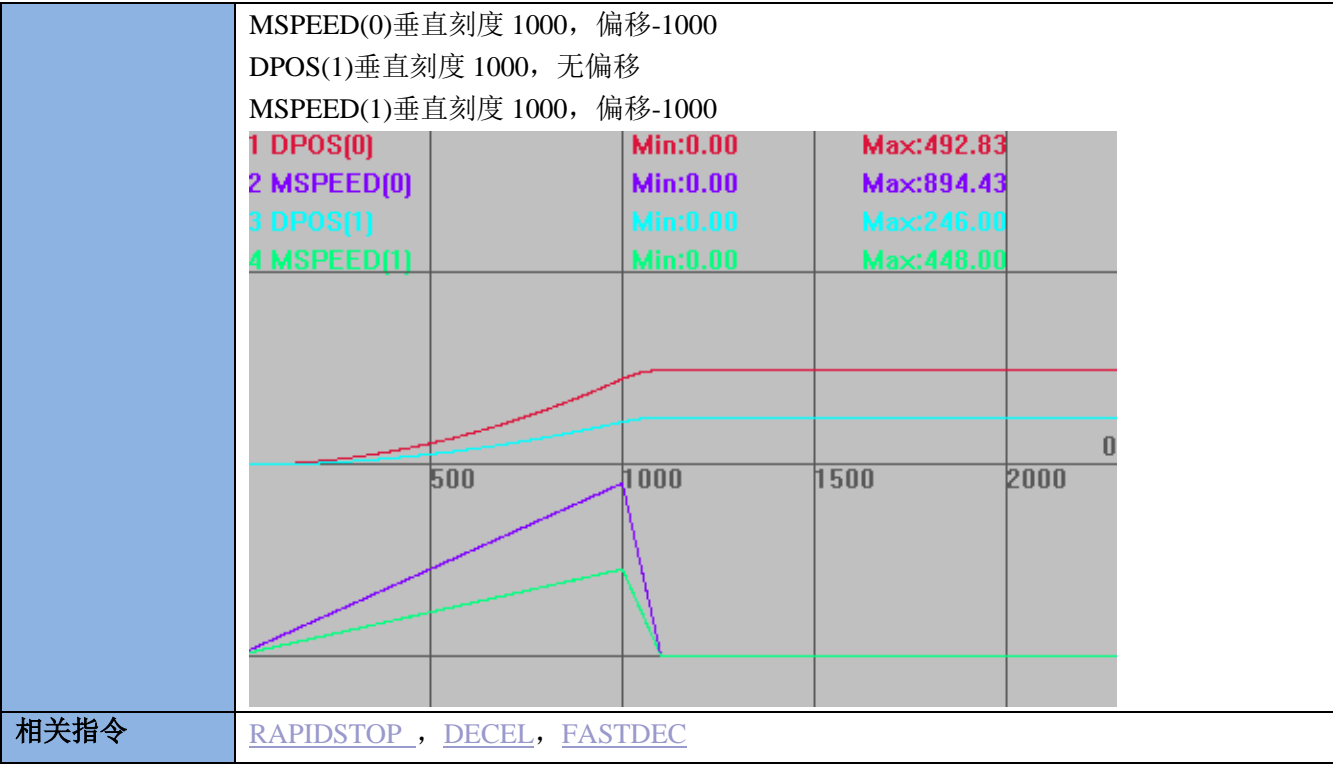


例三

```
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



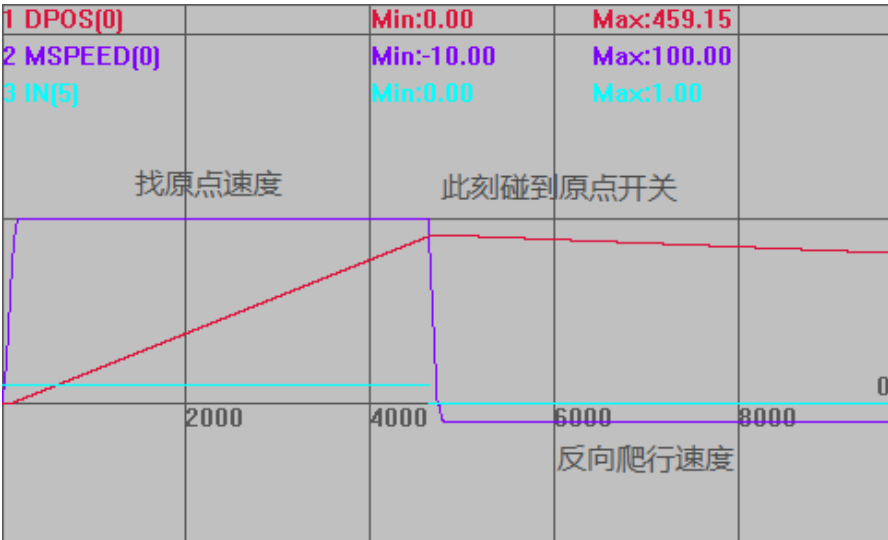
DATUM -- 回零

类型	单轴运动指令						
描述	<p>单轴找原点运动。</p> <p>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。</p> <p>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。</p> <p>Z 信号回零必须配置为带 Z 信号 ATYPE（ATYPE=4 或者 7）。</p> <p>多轴回零时，需要每个轴都使用 datum 指令。</p> <p>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。</p>						
语法	<p>DATUM(mode)，DATUM(21,mode2)</p> <p>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。</p> <p>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)</p> <table><tr><td>值</td><td>描述</td></tr><tr><td>0</td><td>清除所有轴的错误状态。</td></tr><tr><td>1</td><td>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。</td></tr></table>	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。
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 after 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



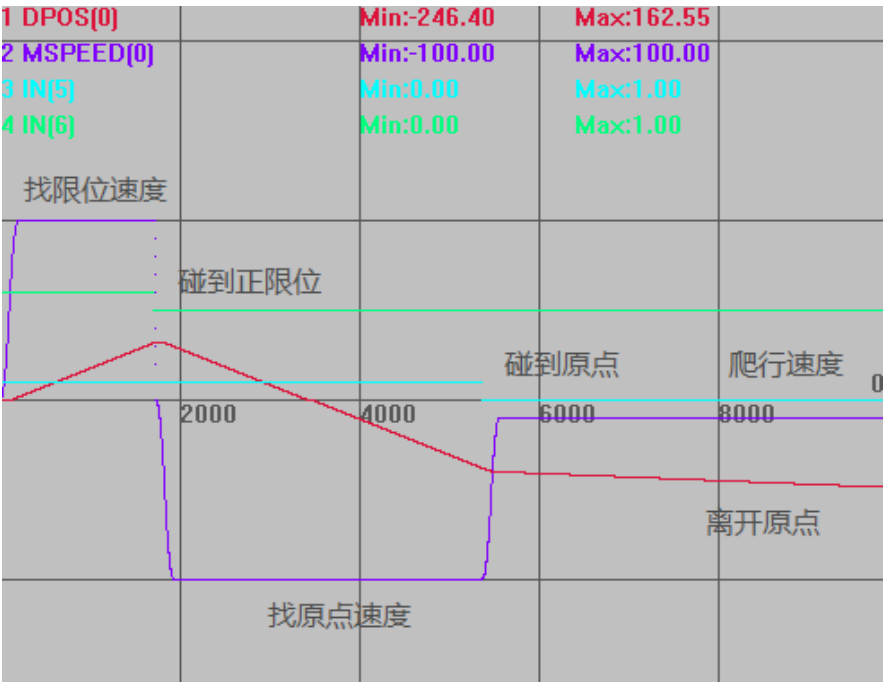
碰到原点开关之后反向爬行直到离开原点开关，此时 DPOS 清 0，回零运动完成。为了直观体现，所以爬行过程比较长，实际应用中，爬行过程很短。

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)   '反转电平信号，一般常闭的信号才进行反转  
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 after 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。

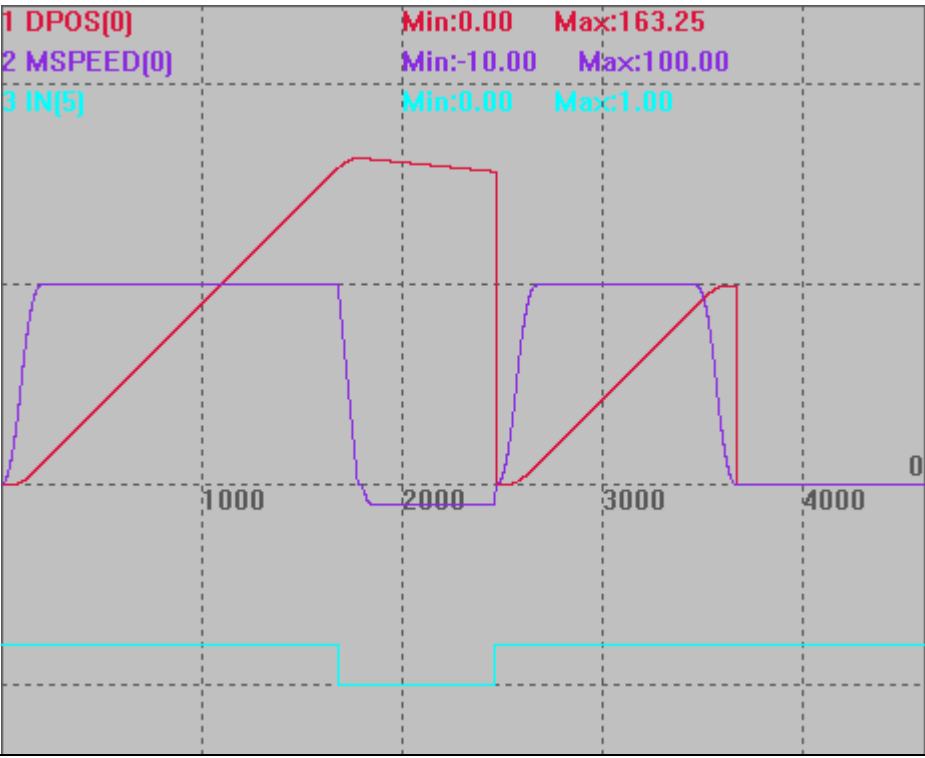
bit13, bit12, bit10 的值组合含义如下表:

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																											
<p>例四 Rtex 总线回零(松下 A6N 伺服)</p> <p>先根据 <a href="#">Rtex 初始化例程</a>正常使能电机</p> <p>SPEED=100<span style="float:right">'找零速度</span></p> <p>ACCEL=1000<span style="float:right">'加减速</span></p> <p>DECEL=1000</p> <p><b>DATUM</b>(21,\$11)<span style="float:right">'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</span></p> <p>根据驱动器手册确定回零模式</p> <table><tr><td rowspan="14">初始化模式</td><td>11h</td><td>Z 相</td></tr><tr><td>12h</td><td>HOME ↑ *2</td></tr><tr><td>13h</td><td>HOME ↓ *3</td></tr><tr><td>14h</td><td>POT ↑ *2</td></tr><tr><td>15h</td><td>POT ↓ *3</td></tr><tr><td>16h</td><td>NOT ↑ *2</td></tr><tr><td>17h</td><td>NOT ↓ *3</td></tr><tr><td>18h</td><td>EXT1 ↑ *2</td></tr><tr><td>19h</td><td>EXT1 ↓ *3</td></tr><tr><td>1Ah</td><td>EXT2 ↑ *2</td></tr><tr><td>1Bh</td><td>EXT2 ↓ *3</td></tr><tr><td>1Ch</td><td>EXT3 ↑ *2</td></tr><tr><td>1Dh</td><td>EXT3 ↓ *3</td></tr></table>					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
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																													
	<p>相关指令</p> <p><a href="#">DATUM IN</a>，<a href="#">DATUM OFFSET</a>，<a href="#">INVERT IN</a></p>																														

## DATUM\_OFFSET -- 原点位置偏移

类型	单轴运动指令
描述	设置原点的位置偏移。 回原点成功后，轴移动到偏移位置。
语法	DATUM_OFFSET(axis)=distance distance: 偏移距离
适用控制器	通用
例子	<p>BASE(0)</p> <p>DPOS=0</p> <p>ATYPE=1</p> <p>SPEED = 100                '找原点速度</p> <p>CREEP = 10                '反向爬行速度</p>

	<p>DATUM_IN=5       '输入 IN5 作为原点开关</p> <p>INVERT_IN(5,ON) '反转 IN5 电平信号，常开信号进行反转(ZMC 控制器)</p> <p>TRIGGER           '自动触发示波器</p> <p>DATUM_OFFSET(0)=100       '轴 0 回零后再偏移</p> <p>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</p> <p>运动轨迹与速度曲线：轴最终停止 DATUM_OFFSET 设置的位置。</p> <p>DPOS(0)垂直刻度 100</p> <p>MSPEED(0)垂直刻度 100</p> <p>IN(5)垂直刻度 5，偏移-5</p> 
相关指令	<a href="#">DATUM</a>

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

	<div>ATYPE=1</div> <div>SPEED=100</div> <div>ACCEL=1000</div> <div>DECEL=1000<div>'减速度设为 1000'</div></div> <div>SRAMP=100</div> <div>VMOVE(-1)<div>'持续负向运动'</div></div> <div>WAIT UNTIL IN(0)=ON<div>'等待输入 1 有效'</div></div> <div>VMOVE(1)<div>'持续正向运动'</div></div>
--	---

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

1 DPOS[0]

2 MSPEED[0]

3 IN[0]

Min:-230.10

Min:-100.00

Min:0.00

Max:509.90

Max:100.00

Max:1.00

FORWARD -- 正向运动

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



	<p>例二</p> <p><b>FORWARD</b> AXIS(1)      '轴 1 正向运动</p> <p>WAIT UNTIL IN(1)=ON      '等待输入 1 有效</p> <p>CANCEL(2) AXIS(1)</p>
相关指令	<a href="#">REVERSE</a> , <a href="#">VMOVE</a>

## REVERSE -- 负向运动

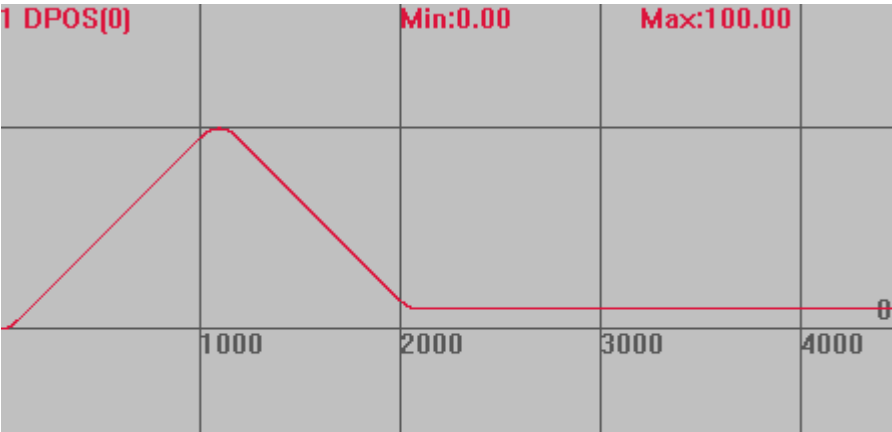
类型	单轴运动指令
描述	<p><b>BASE</b> 选择轴负向运动。</p> <p>必须 <b>CANCEL</b> 后才能切换 <b>FORWARD</b>。</p>
语法	REVERSE [axis(轴号)]
适用控制器	通用
例子	<p>例一</p> <p>BASE(0)</p> <p><b>REVERSE</b>      '轴 0 持续负向运动</p> <p>WAIT UNTIL IN(1)=ON      '等待输入 1 有效</p> <p>CANCEL(2)</p> <p>例二</p> <p><b>REVERSE</b> AXIS(1)      '轴 1 负向运动</p> 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

## MOVEMODIFY -- 修改运动位置

类型	单轴运动指令
描述	<p>修改上一个运动的目标位置。</p> <p>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</p> <p>需要 WAIT 指令才能正确使用。见例二</p> <p>连续插补时使用 <b>MOVEMODIFY</b> 会破坏速度连续性。</p> <p><b>MOVEMODIFY</b> 同时对多轴运动时不一定为直线插补。</p>
语法	<p>MOVEMODIFY (distance )</p> <p>distance: 单个轴的运动距离</p> <p>目前只支持单轴修改。</p>
适用控制器	通用
例子	<p>例一</p> <p>BASE(0)</p>

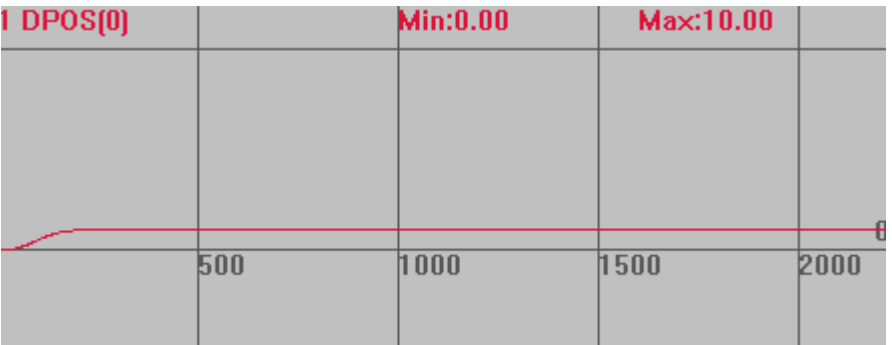
UNITS=100                   '脉冲当量设置  
DPOS=0  
SPEED=100                  '速度设置  
ACCEL=1000                '加速度设置  
DECEL=1000  
TRIGGER                    '自动触发示波器  
MOVEABS(100)  
MOVEABS(10)                '此时轴会先运动到 100，在往回运动到 10

运动轨迹  
DPOS(0)垂直刻度 100

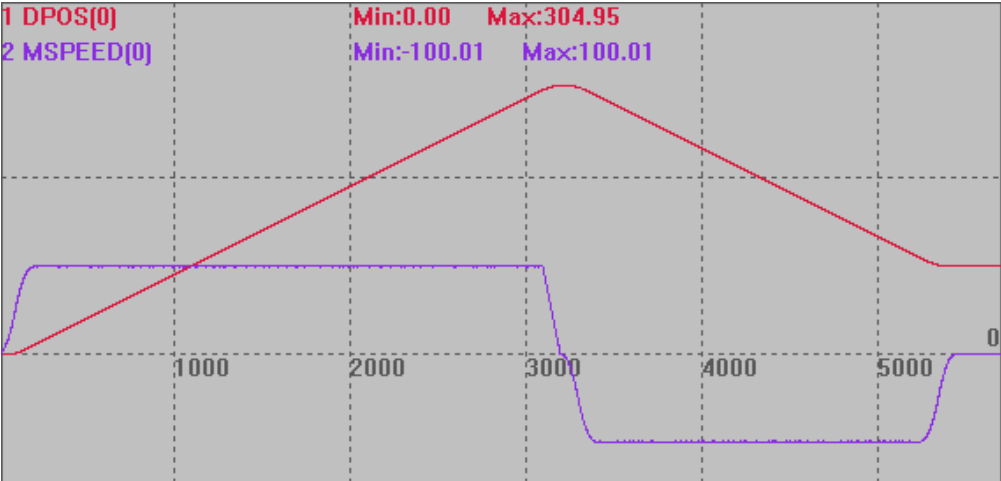
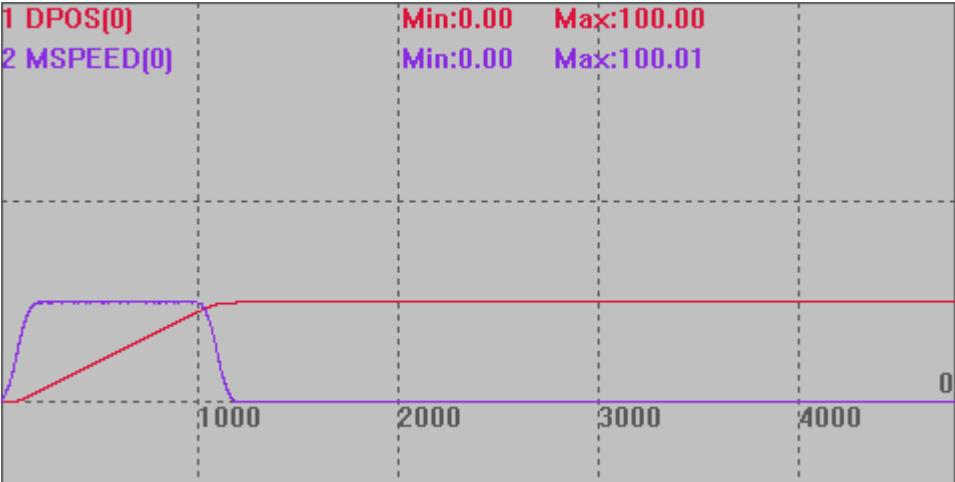


将 MOVEABS 指令变为如下指令：  
**MOVEMODIFY(100)**  
**MOVEMODIFY(10)** '此时轴会直接运动到 10 位置，MOVEMODIFY 不进入运动缓冲

运动轨迹  
DPOS(0)垂直刻度 100



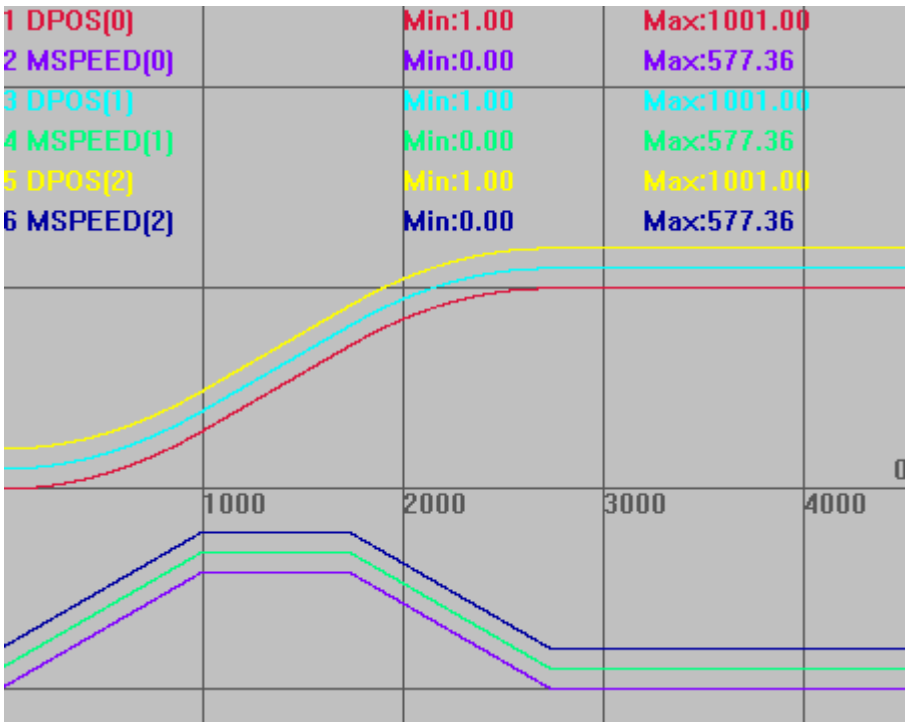
例二  
BASE (0)  
UNITS=100                   '脉冲当量设置  
DEFPOS(0)  
SPEED=100                  '速度设置  
ACCEL=1000                '加速度设置

	<div>DECEL=1000</div> <div>SRAMP=100</div> <div>TRIGGER'自动触发示波器</div> <div>MOVEABS(500)</div> <div>WAIT UNTIL DPOS &gt;=300'等待运动到 300 时，才修改终点位置</div> <div>MOVEMODIFY (100)'修改目标位置为 100，此时减速停止后再反向运动</div> <div>使用 WAIT，运动轨迹</div> <div>DPOS(0)垂直刻度 200</div> <div>MSPEED(0)垂直刻度 200</div> <div></div> <div>不使用 WAIT，直接运动到 100，运动轨迹</div> <div>DPOS(0)垂直刻度 300</div> <div></div>
相关指令	<a href="#">MOVEMODIFY2</a>

## 7.2. 多轴运动指令

### RAPIDSTOP -- 全部轴停止

类型	多轴运动指令								
描述	<p>所有轴立即停止，如果轴参与插补，也停止插补运动。</p> <p>Mode0~2 减速度按 FASTDEC 和 DECEL 中最大的值。</p> <p>RAPIDSTOP 后要调用绝对位置运动，必须先 WAIT IDLE 等待停止完成。</p>								
语法	<p>RAPIDSTOP (mode)</p> <p>mode: 模式选择</p> <table border="1"> <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>	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
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	<p>例一</p> <p>BASE(0,1,2)</p> <p>DPOS=1,1,1</p> <p>ATYPE=1,1,1</p> <p>UNITS=100,100,100</p> <p>SPEED=1000                      '插补合成运动速度 100</p> <p>ACCEL=1000</p> <p>DECEL=1000                      '减速度设为 1000</p> <p>FASTDEC=10000                  '快减减速度设为 10000</p> <p>TRIGGER</p> <p>MOVE(1000,1000,1000)          '当前运动</p> <p>MOVE(-1000,-1000,-1000)      '缓冲运动</p> <p><b>RAPIDSTOP(1)</b>                  '此时轴只执行完当前运动</p> <p>运动轨迹和速度曲线</p> <p>DPOS(0)垂直刻度 1000，无偏移</p> <p>MSPEED(0)垂直刻度 1000，偏移-1000</p> <p>DPOS(1)垂直刻度 1000，偏移 100</p> <p>MSPEED(1)垂直刻度 1000，偏移-900</p> <p>DPOS(2)垂直刻度 1000，偏移 200</p> <p>MSPEED(2)垂直刻度 1000，偏移-800</p>								



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

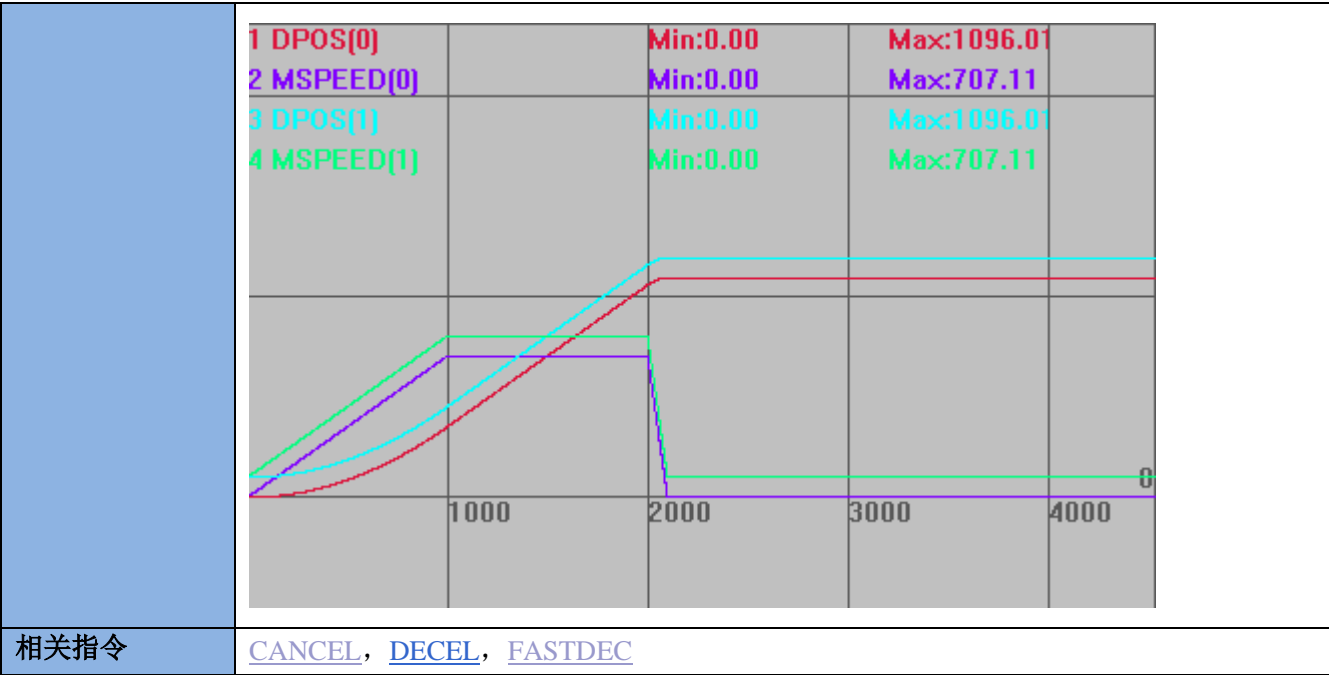
运动轨迹和速度曲线

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



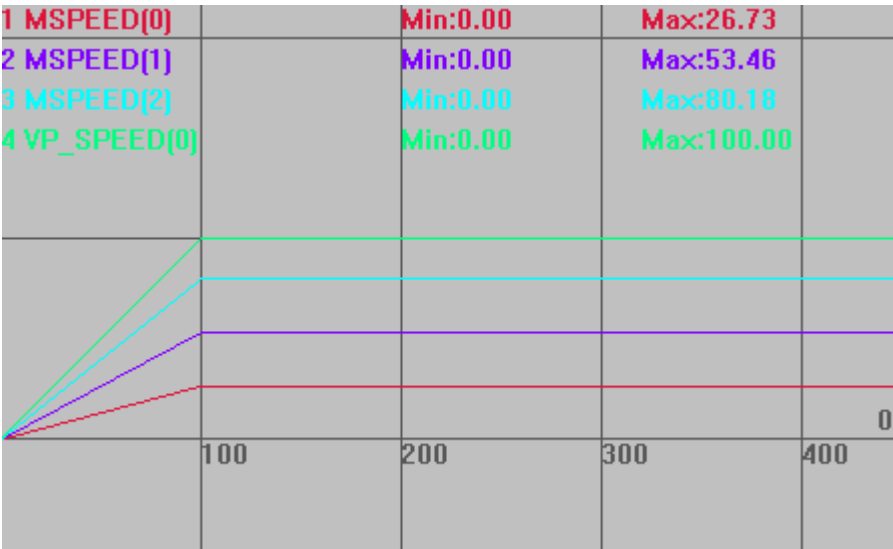
MOVE -- 直线运动

类型	多轴运动指令
描述	<p>直线插补运动，相对运动一段距离。</p> <p>插补运动时只有主轴速度参数有效，主轴是 <b>BASE</b> 的第一个轴，运动参照这个轴的参数。</p> <p>自定义速度的连续插补运动可以使用 <b>SP</b> 后缀的指令，见*<b>SP</b> 描述。</p> <p>插补运动距离 <math>X = \sqrt{X_0^2 + X_1^2 + X_2^2 + \dots + X_n^2}</math></p> <p>运动时间 <math>T = X / \text{主轴 SPEED}</math></p>
语法	<p>MOVE(distance1 [,distance2 [,distance3 [,distance4...]]])</p> <p>distance1: 第一个轴运动距离</p> <p>distance2: 下一个轴运动距离</p>
适用控制器	通用
例子	<p>例一</p> <p>BASE(0,1,2) '主轴为轴 0</p> <p>ATYPE=1,1,1 '设为脉冲轴类型</p> <p>UNITS=100,100,100 '脉冲当量设置</p> <p>SPEED=100,10,1000 '只有主轴速度 100 起作用，作为合成运动的速度</p> <p>ACCEL=1000,1000,1000</p> <p>DECEL=1000,1000,1000</p> <p>DPOS = 0,0,0</p> <p>TRIGGER '自动触发示波器</p> <p><b>MOVE</b>(500,1000,1500) '轴 0，1，2 直线插补相对距离</p>

WAIT IDLE                   '等待运动停止  
PRINT \*DPOS                '打印结果，500,1000,1500

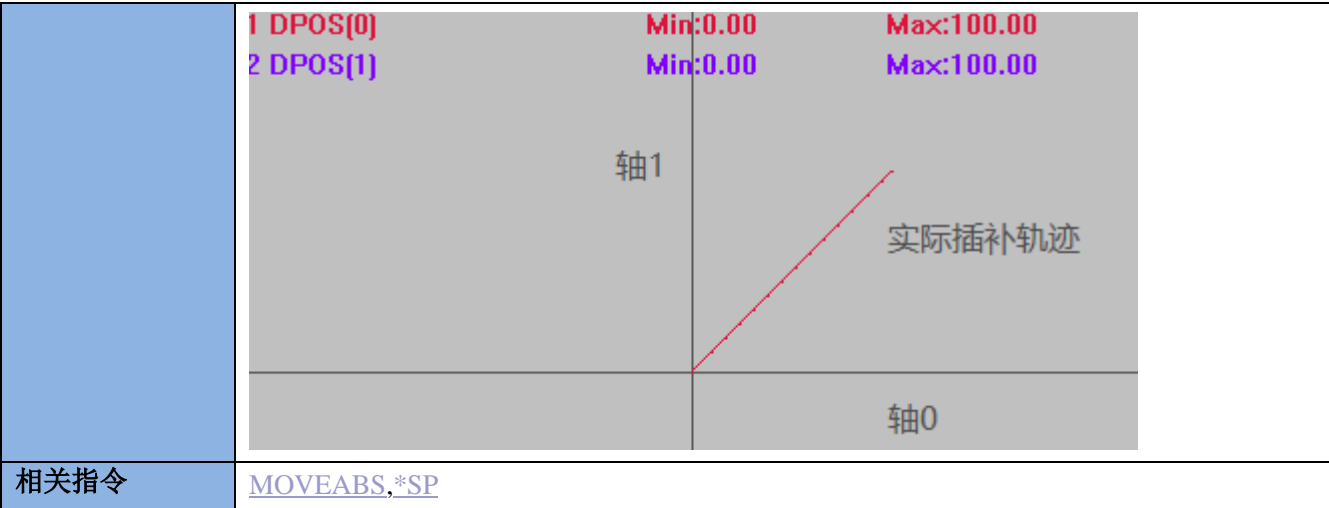
插补运动各轴实际速度为主轴速度的分速度

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)

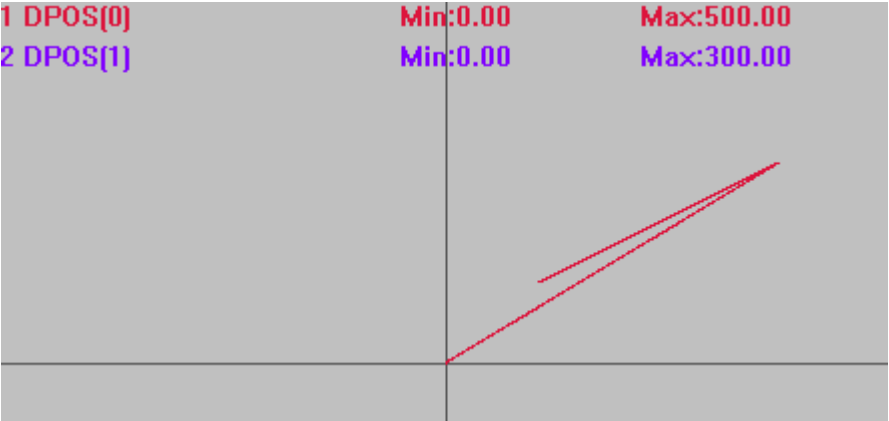
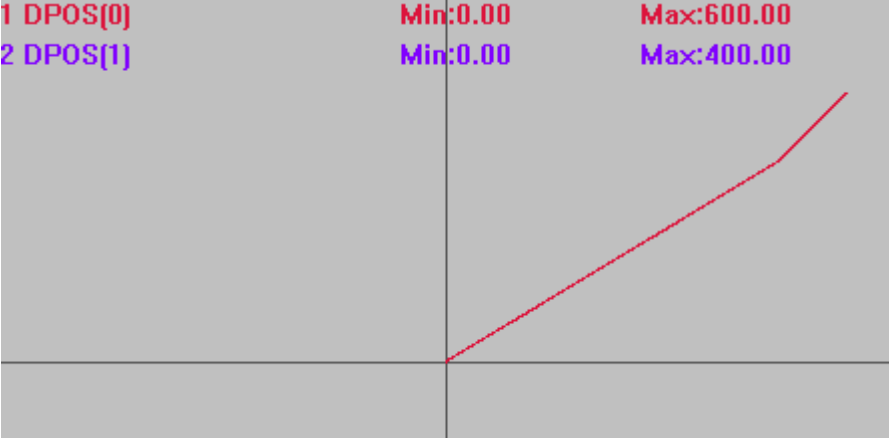
插补轨迹  
DPOS(0)水平刻度 100  
DPOS(1)垂直刻度 100



MOVEABS -- 直线运动-绝对

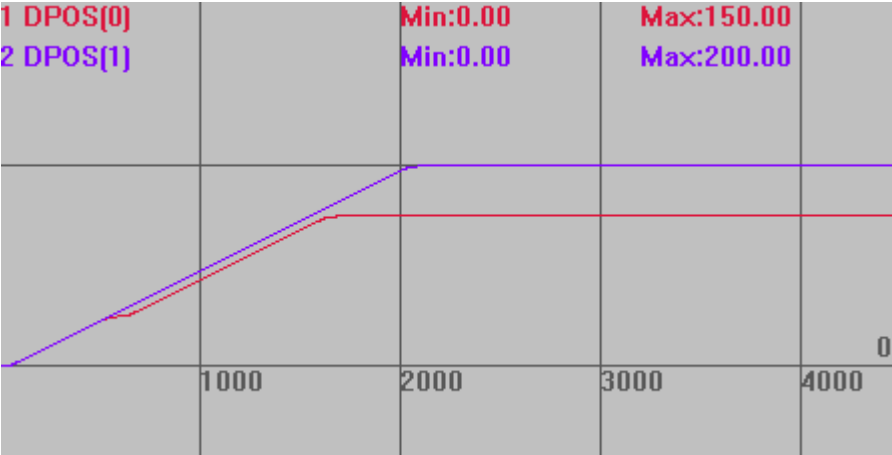
类型	多轴运动指令
描述	直线插补运动，绝对运动到指定坐标。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。
语法	MOVEABS(position1[, position2[, position3[, position4...]]]) position1: 第一个轴运动坐标 position2: 下一个轴运动坐标
适用控制器	通用
例子	BASE(0,1) ATYPE=1,1 UNITS=100,100 DPOS=0,0 MPOS=0,0 SPEED=100,100 ACCEL=1000,1000 DECEL=1000,1000 TRIGGER '自动触发示波器' MOVEABS(500,300) '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



	<div><div><div>1 DPOS[0]</div><div>2 DPOS[1]</div></div><div><div>Min:0.00</div><div>Min:0.00</div></div><div><div>Max:500.00</div><div>Max:300.00</div></div></div>  <p>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</p> <div><div><div>1 DPOS[0]</div><div>2 DPOS[1]</div></div><div><div>Min:0.00</div><div>Min:0.00</div></div><div><div>Max:600.00</div><div>Max:400.00</div></div></div> 
相关指令	<a href="#">MOVE</a> , <a href="#">*SP</a>

MOVEMODIFY2 -- 运动新位置

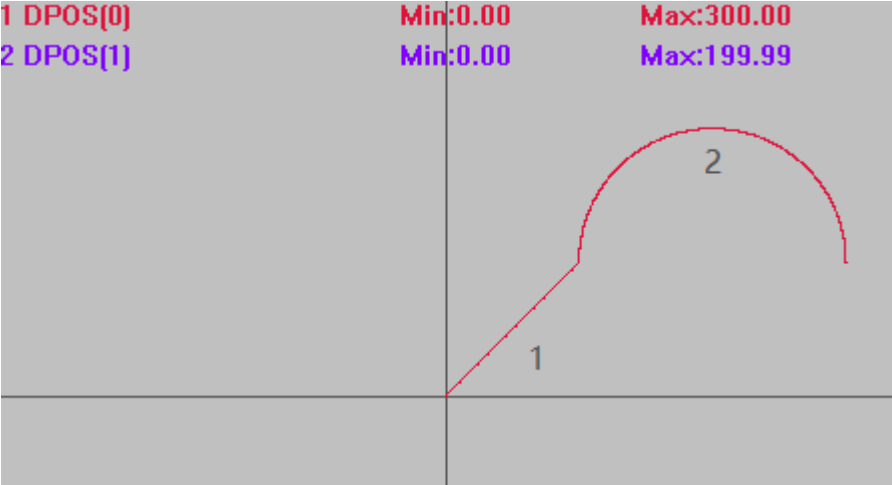
类型	运动指令
描述	<p>强制停止原来的运动，接着按原来的速度与加速度定位到新目标位置。</p> <p>前面没有运动时与 MOVEABS 效果一样，但不会进入运动缓冲。见例一。</p> <p>需要 WAIT 指令才能正确使用。见例二。</p> <p>连续插补时使用 MOVEMODIFY2 会破坏速度连续性。</p> <p>MOVEMODIFY2 同时对多轴运动时不一定为直线插补。</p>
语法	<p>MOVEMODIFY2 (abspos1, abspos2,[...])</p> <p>abspos1: BASE 第一个轴的新目标位置</p> <p>abspos2: BASE 第二个轴的新目标位置</p> <p>3X 系列 20161209 以上固件支持.</p> <p>4 系列 20170509 以上固件支持。</p>

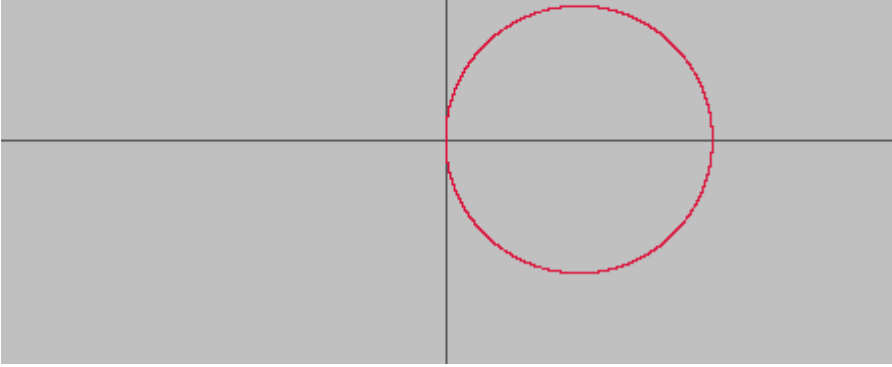
适用控制器	特殊固件
例子	<div><div>例一</div><div>BASE(0,1) ATYPE=1,1                   '设为脉冲轴类型 DPOS=0,0 SPEED = 100,100 ACCEL=1000                   '加速度设置 DECEL=1000 TRIGGER MOVE(200) <b>MOVEMODIFY2(50,200)</b>   '取消 MOVE（200），强制定位新的位置 50,200 MOVE(100)</div><div>运动轨迹 DPOS(0)垂直刻度 200 DPOS(1)垂直刻度 200</div><div></div><div>例二</div><div>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)&gt;=100   '等待轴 0 运行过 100 <b>MOVEMODIFY2(50,200)</b> MOVE(100)</div><div>运动轨迹 竖直刻度同上</div></div>

	<div><div>1 DPOS[0]</div><div>2 DPOS[1]</div></div> <div><div>Min:0.00</div><div>Min:0.00</div><div>Max:150.00</div><div>Max:200.00</div></div> <div></div> <div>0</div>
相关指令	<a href="#">MOVEMODIFY</a>

## MOVECIRC -- 圆心画弧

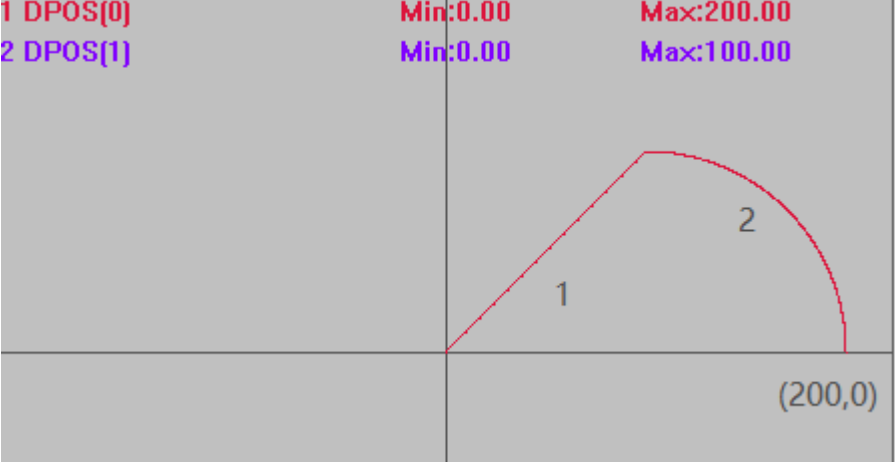
类型	多轴运动指令
描述	<p>两轴圆弧插补，圆心画弧，相对运动。</p> <p>BASE 第一轴和第二轴进行圆弧插补，相对移动方式，当终点距离为 0 时为整圆。</p> <p>自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。</p> <p>使用时需获得圆心和圆弧终点相对于起始点坐标。</p> <p>坐标位置请确保正确，否则实际运动轨迹会错误。</p> <div></div> <p>假设起始点 A 坐标为 (100,100)，圆心 C 坐标为 (400,100)，终点 B 坐标为 (400,400)。</p> <p>圆心 C 相对于起始点 A 的坐标为 (300,0)，终点 B 相对于起始点 A 的坐标为 (300,300)。</p>
语法	<p>MOVECIRC(end1, end2, centre1, centre2, direction)</p> <p>end1: 终点第一个轴运动坐标，相对于起始点</p> <p>end2: 终点第二个轴运动坐标，相对于起始点</p> <p>centre1: 圆心第一个轴运动坐标，相对于起始点</p> <p>centre2: 圆心第二个轴运动坐标，相对于起始点</p>

	direction: 0-逆时针, 1-顺时针
适用控制器	通用
例子	<div>BASE(0,1)</div> <div>ATYPE=1,1           '设为脉冲轴类型</div> <div>UNITS=100,100</div> <div>DPOS=0,0</div> <div>SPEED=100,100</div> <div>ACCEL=1000,1000</div> <div>DECEL=1000,1000</div> <div>TRIGGER               '自动触发示波器</div> <div>MOVE(100,100)         '先运动 100,100 位置</div> <div><b>MOVECIRC(200,0,100,0,1)</b>   '半径 100 顺时针画半圆，终点坐标（300,100）</div> <div> 插补轨迹</div> <div>DPOS(0)垂直刻度 150</div> <div>DPOS(1)垂直刻度 150</div> <div><div><div>1 DPOS[0]</div><div>2 DPOS[1]</div></div><div><div>Min:0.00</div><div>Min:0.00</div></div><div><div>Max:300.00</div><div>Max:199.99</div></div></div> <div></div> <div> 其他条件不变，将运动指令修改。</div> <div><b>MOVECIRC(0,0,100,0,0)</b>   '半径 100，圆心（100，0）逆时针画圆</div> <div> 插补轨迹</div> <div>竖直刻度同上</div>

	<div><div>1 DPOS[0]Min:0.00Max:199.99</div><div>2 DPOS[1]Min:-100.00Max:100.00</div></div>
相关指令	<a href="#">MOVECIRCABS</a> , <a href="#">MOVECIRC2</a> , <a href="#">*SP</a>

MOVECIRCABS -- 圆心画弧-绝对

类型	多轴运动指令
描述	<p>两轴圆弧插补，圆心画弧，绝对运动。</p> <p>BASE 第一轴和第二轴进行圆弧插补，绝对移动方式。</p> <p>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。</p> <p><b>MOVECIRCABS 不支持走整圆，整圆使用 MOVECIRC。</b></p>
语法	<p>MOVECIRCABS(end1, end2, centre1, centre2, direction)</p> <p>end1: 终点第一个轴运动坐标，绝对位置</p> <p>end2: 终点第二个轴运动坐标，绝对位置</p> <p>centre1: 圆心第一个轴运动坐标，绝对位置</p> <p>centre2: 圆心第二个轴运动坐标，绝对位置</p> <p>direction: 0-逆时针，1-顺时针</p> <p>坐标位置请确保正确，否则实际运动轨迹会错误。</p>
适用控制器	通用
例子	<p>BASE(0,1)</p> <p>ATYPE=1,1           '设为脉冲轴类型</p> <p>UNITS=100,100</p> <p>DPOS=0,0</p> <p>SPEED=100,100</p> <p>ACCEL=1000,1000</p> <p>DECEL=1000,1000</p> <p>TRIGGER               '自动触发示波器</p> <p>MOVE(100,100)        '先运动 100,100 位置</p> <p><b>MOVECIRCABS(200,0,100,0,1)</b> '半径 100 顺时针画 1/4 圆，终点坐标</p>

	<div>' (200,0)</div> <div>插补轨迹</div> <div>DPOS(0)垂直刻度 100</div> <div>DPOS(1)垂直刻度 100</div> <div><div><div>1 DPOS[0]</div><div>2 DPOS[1]</div></div><div><div>Min:0.00</div><div>Min:0.00</div></div><div><div>Max:200.00</div><div>Max:100.00</div></div></div> <div></div>
相关指令	<a href="#">MOVECIRC</a> , <a href="#">MOVECIRC2ABS</a> , <a href="#">*SP</a>

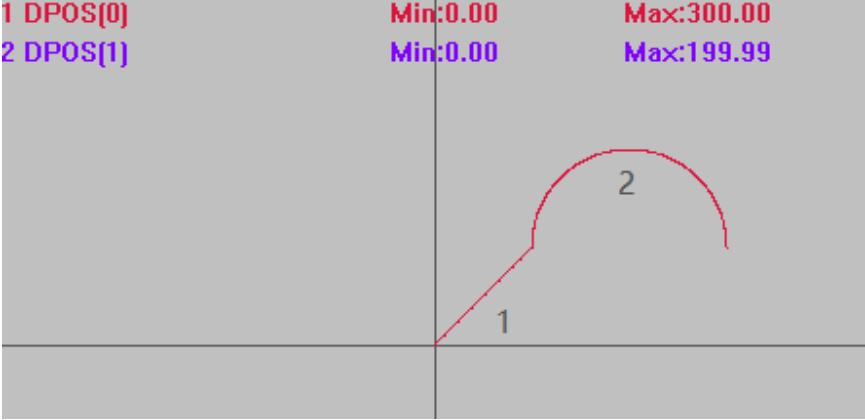
## MOVECIRC2 -- 三点画弧

类型	多轴运动指令
描述	<p>两轴圆弧插补，三点画弧，相对运动。</p> <p>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。</p> <p>自定义速度的连续插补运动可以使用 <b>SP</b> 后缀的指令，见<b>*SP</b> 描述。</p> <p><b>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。</b></p>
语法	<p>MOVECIRC2(mid1, mid2, end1, end2)</p> <p>mid1: 中间点第一个轴坐标，相对起始点距离</p> <p>mid2: 中间点第二个轴坐标，相对起始点距离</p> <p>end1: 结束点第一个轴坐标，相对起始点距离</p> <p>end2: 结束点第二个轴坐标，相对起始点距离</p> <p>坐标位置请确保正确，否则实际运动轨迹会错误。</p>
适用控制器	通用
例子	<p>BASE(0,1)</p> <p>ATYPE=1,1            '设为脉冲轴类型</p> <p>UNITS=100,100</p> <p>DPOS=0,0</p> <p>SPEED=100,100</p>

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

MOVECIRC2ABS -- 三点画弧-绝对

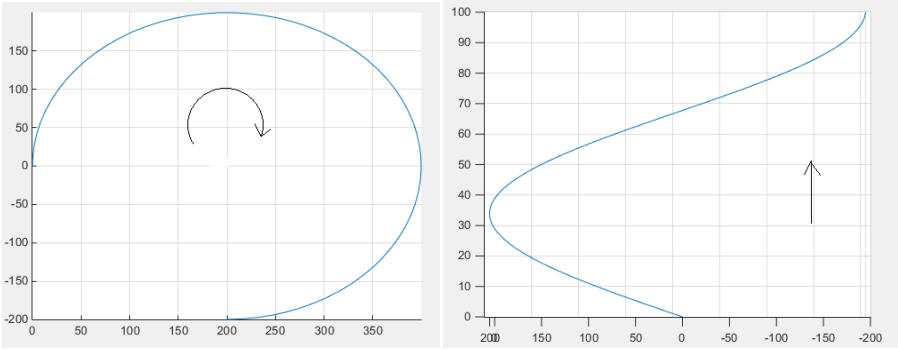
类型	多轴运动指令
描述	<div>圆弧插补， 三点画弧， 绝对运动。</div> <div>BASE 第一轴和第二轴进行圆弧插补， 绝对移动方式。自定义速度的连续插补运动可以使用 SP 后缀的指令， 见*SP 描述。</div> <div>注意： 不能用此指令进行整圆插补运动， 整圆使用 MOVECIRC 相对圆弧， 或连续使用两条此类指令。</div>
语法	<div>MOVECIRC2ABS(mid1, mid2, end1, end2)</div> <div>mid1： 中间点第一个轴坐标， 绝对位置</div> <div>mid2： 中间点第二个轴坐标， 绝对位置</div> <div>end1：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

	<div>DPOS=0,0</div> <div>SPEED=100,100</div> <div>ACCEL=1000,1000</div> <div>DECEL=1000,1000</div> <div>TRIGGER'自动触发示波器</div> <div>MOVE(100,100)'先运动 100,100 位置</div> <div>MOVECIRC2ABS(200,200,300,100)'三点画半圆，绝对坐标</div> <div>插补轨迹</div> <div>DPOS(0)垂直刻度 200</div> <div>DPOS(1)垂直刻度 200</div> <div><div><div>1 DPOS[0]</div><div>2 DPOS[1]</div></div><div><div>Min:0.00</div><div>Min:0.00</div></div><div><div>Max:300.00</div><div>Max:199.99</div></div></div>
相关指令	<a href="#">MOVECIRC2</a> ， <a href="#">MOVECIRCABS</a> ， <a href="#">*SP</a>

MHELICAL -- 圆心螺旋

类型	多轴运动指令						
描述	<div>螺旋插补，圆心画弧，相对运动。</div> <div>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。</div> <div>可完整螺旋一圈，从 Z 方向看为一整圆。</div>						
语法	<div>MHELICAL(end1,end2,centre1,centre2,direction,distance3,[mode])</div> <div>end1: 结束点第一个轴运动坐标，相对于起始点</div> <div>end2: 结束点第二个轴运动坐标，相对于起始点</div> <div>centre1: 圆心第一个轴运动坐标，相对于起始点</div> <div>centre2 : 圆心第一个轴运动坐标，相对于起始点</div> <div>direction: 0-逆时针，1-顺时针</div> <div>distance3: 第三个轴运动距离</div> <div>mode: 第三轴的速度计算</div> <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>	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
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	<div>BASE(0,1,2)</div> <div>ATYPE=1,1,1            '设为脉冲轴类型</div> <div>UNITS=100,100,100</div> <div>DPOS=0,0,0</div> <div>SPEED=100,100,100       '主轴速度</div> <div>ACCEL=1000 ,1000,1000    '主轴加速度</div> <div>DECEL=1000 ,1000,1000</div> <div>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</div> <div>插补轨迹</div> <div></div>
相关指令	<a href="#">MHELICAL2</a> ， <a href="#">MHELICALABS</a> ， <a href="#">*SP</a>

MHELICALABS -- 圆心螺旋-绝对

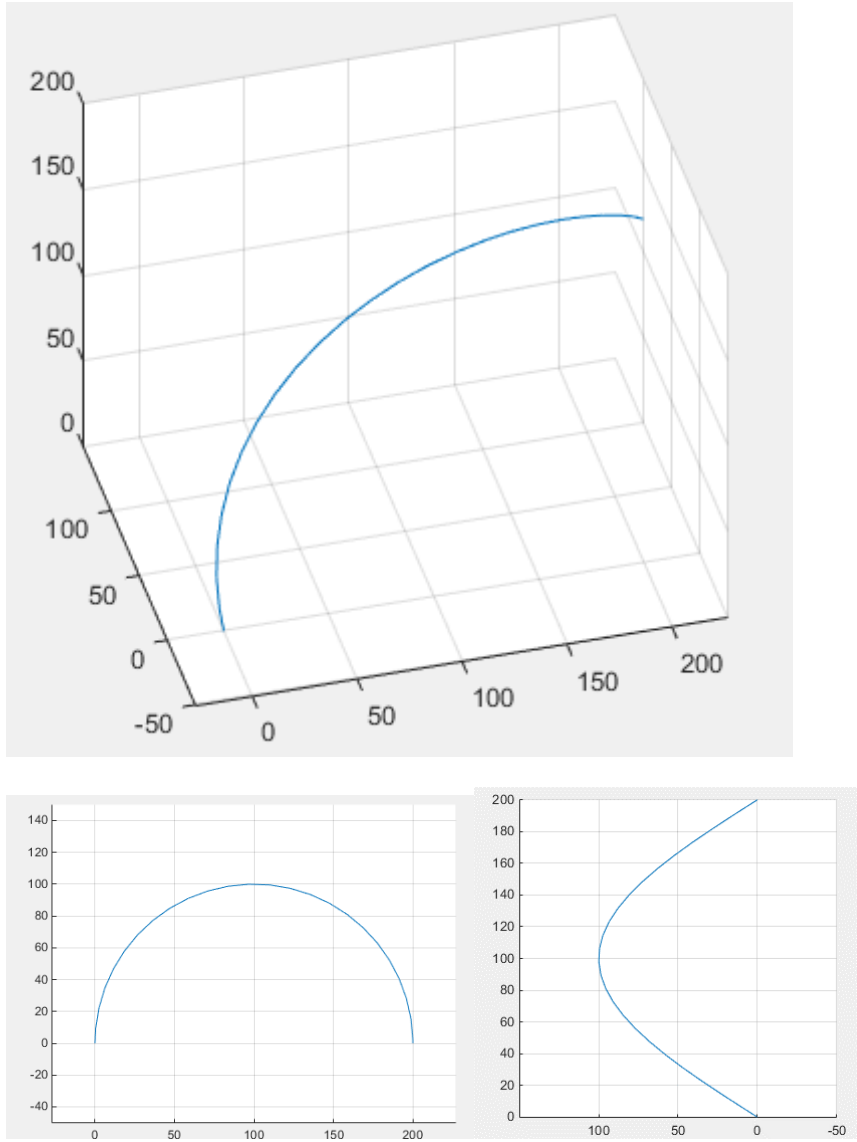
类型	多轴运动指令						
描述	<div>螺旋插补，圆心画弧，绝对运动。</div> <div>BASE 第一轴和第二轴进行圆弧插补，第三轴进行螺旋，绝对移动方式。</div> <div>自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。</div> <div>可完整螺旋一圈，从 Z 方向看为一个整圆。</div>						
语法	<div>MHELICALABS(end1,end2,centre1,centre2,direction,distance3,[mode])</div> <div>end1: 第一个轴运动坐标</div> <div>end2: 第二个轴运动坐标</div> <div>centre1: 第一个轴运动圆心</div> <div>centre2 : 第二个轴运动圆心</div> <div>direction: 0-逆时针，1-顺时针</div> <div>distance3: 第三个轴运动坐标</div> <div>mode: 第三轴的速度计算：</div> <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>	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
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	<div>BASE(0,1,2)</div> <div>ATYPE=1,1,1            '设为脉冲轴类型</div> <div>UNITS=100,100,100</div> <div>DPOS=0,0,0</div> <div>SPEED=100,100,100       '主轴速度</div> <div>ACCEL=1000 ,1000,1000   '主轴加速度</div> <div>DECEL=1000 ,1000,1000</div> <div>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</div> <div>插补轨迹</div> <div></div> <div></div> <div></div>

MHELICAL2 -- 三点螺旋

类型	多轴运动指令
----	--------

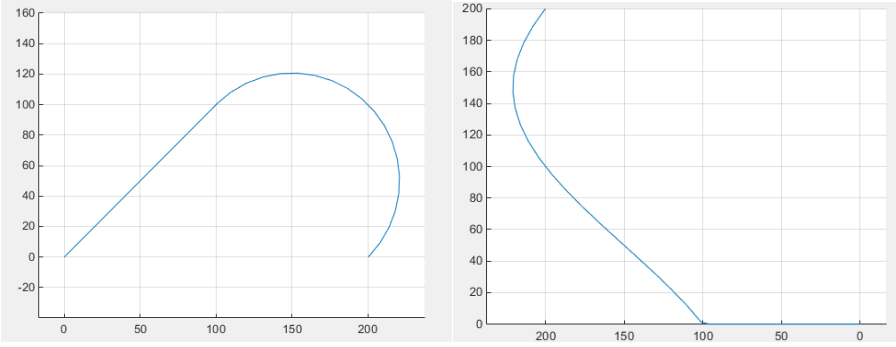
描述	<p>螺旋插补，三点画弧，相对运动。</p> <p>BASE 第一轴和第二轴进行圆弧插补，第三轴进行螺旋，相对移动方式。</p> <p>自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。</p> <p>无法螺旋一圈，需要螺旋一圈请使用 MHELICAL 或 MHELICALABS。</p>						
语法	<p>MHELICAL2(mid1, mid2, end1, end2, distance3,[mode])</p> <p>mid1: 中间点第一个轴坐标，相对起始点距离</p> <p>mid2: 中间点第二个轴坐标，相对起始点距离</p> <p>end1: 结束点第一个轴坐标，相对起始点距离</p> <p>end2: 结束点第二个轴坐标，相对起始点距离</p> <p>distance3: 第三个轴运动距离，相对起始点距离</p> <p>mode: 第三轴的速度计算</p> <table border="1"> <tr> <th>值</th><th>描述</th></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> <p>坐标位置请确保正确，否则实际运动轨迹会错误。</p>	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
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						
适用控制器	通用						
例子	<p>BASE(0,1,2)</p> <p>ATYPE=1,1,1            '设为脉冲轴类型</p> <p>UNITS=100,100,100</p> <p>DPOS=0,0,0</p> <p>SPEED=100,100,100       '主轴速度</p> <p>ACCEL=1000,1000,1000    '主轴加速度</p> <p>DECEL=1000,1000,1000</p> <p><b>MHELICAL2</b>(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</p> <p>插补轨迹</p>						

	
相关指令	<a href="#">MHELICAL2ABS</a> , <a href="#">MHELICAL</a> , <a href="#">*SP</a>

MHELICAL2ABS -- 三点螺旋-绝对

类型	多轴运动指令
描述	<p>螺旋插补，三点画弧，绝对运动。</p> <p>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 <b>MHELICAL</b> 或 <b>MHELICALABS</b>。</p>
语法	<p>MHELICAL2ABS(mid1, mid2, end1, end2, distance3,[mode])</p> <p>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</p>

	<p>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</p> <p>mode: 第三轴的速度计算</p> <table><tr><th>值</th><th>描述</th></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> <p>坐标位置请确保正确, 否则实际运动轨迹会错误。</p>	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
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						
适用控制器	通用						
例子	<p>BASE(0,1,2)</p> <p>ATYPE=1,1,1            '设为脉冲轴类型</p> <p>UNITS=100,100,100</p> <p>DPOS=0,0,0</p> <p>SPEED=100,100,100       '主轴速度</p> <p>ACCEL=1000 ,1000,1000   '主轴加速度</p> <p>DECEL=1000 ,1000,1000</p> <p>MOVE(100,100)           '运动到 (100,100)</p> <p><b>MHELICAL2ABS</b>(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</p> <p>插补轨迹</p>						

	
相关指令	<a href="#">MHELICAL2</a> , <a href="#">MHELICALABS</a> , <a href="#">*SP</a>

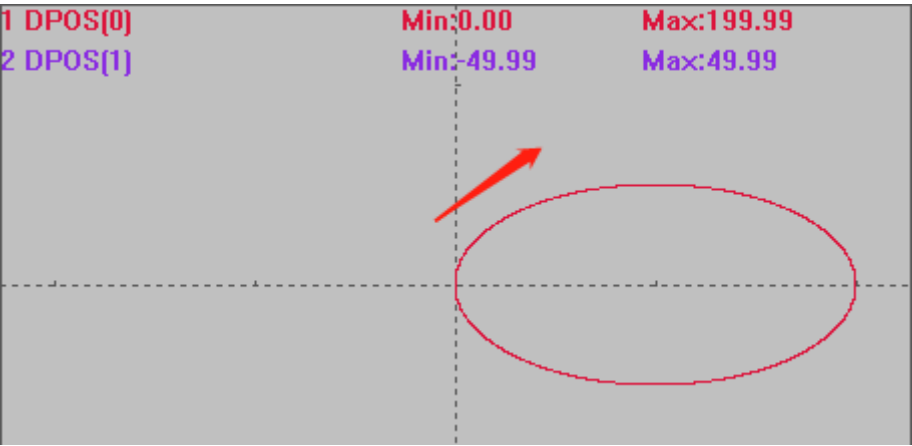
MECLIPSE -- 椭圆

类型	多轴运动指令						
描述	<p>椭圆插补，中心画弧，相对运动，可选螺旋。</p> <p>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。</p> <p>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。</p> <p>可画整椭圆。</p> <p>只能画长轴（短轴）与 X 平行或垂直的椭圆。</p>						
语法	<p>MECLIPSE (end1, end2, centre1, centre2, direction, adis, bdis[, end3])</p> <p>end1: 终点第一个轴运动坐标，相对于起始点</p> <p>end2: 终点第二个轴运动坐标，相对于起始点</p> <p>centre1: 中心第一个轴运动坐标，相对于起始点</p> <p>centre2: 中心第二个轴运动坐标，相对于起始点</p> <p>direction: 0-逆时针，1-顺时针</p> <table><tr><td>值</td><td>描述</td></tr><tr><td>0</td><td>逆时针</td></tr><tr><td>1</td><td>顺时针</td></tr></table> <p>adis: 第一轴的椭圆半径，半长轴或者半短轴都可</p> <p>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</p> <p>end3: 第三个轴的运动距离，需要螺旋时填入</p> <p>坐标位置请确保正确，否则实际运动轨迹会错误。</p>	值	描述	0	逆时针	1	顺时针
值	描述						
0	逆时针						
1	顺时针						
适用控制器	通用						
例子	<p>例一 不进行螺旋</p> <p>RAPIDSTOP(2)</p> <p>WAIT IDLE(0)</p> <p>BASE(0,1,2)</p> <p>ATYPE=1,1,1        '设为脉冲轴类型</p> <p>UNITS=100,100,100</p> <p>SPEED=100,100,100    '主轴速度</p>						

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

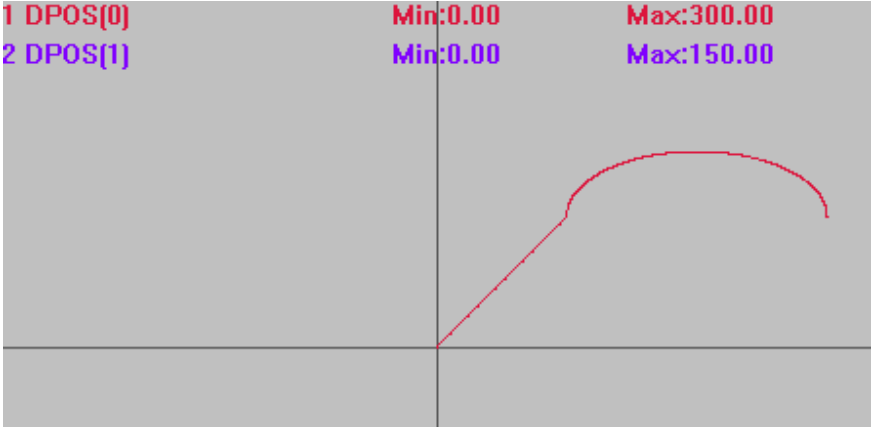
插补轨迹

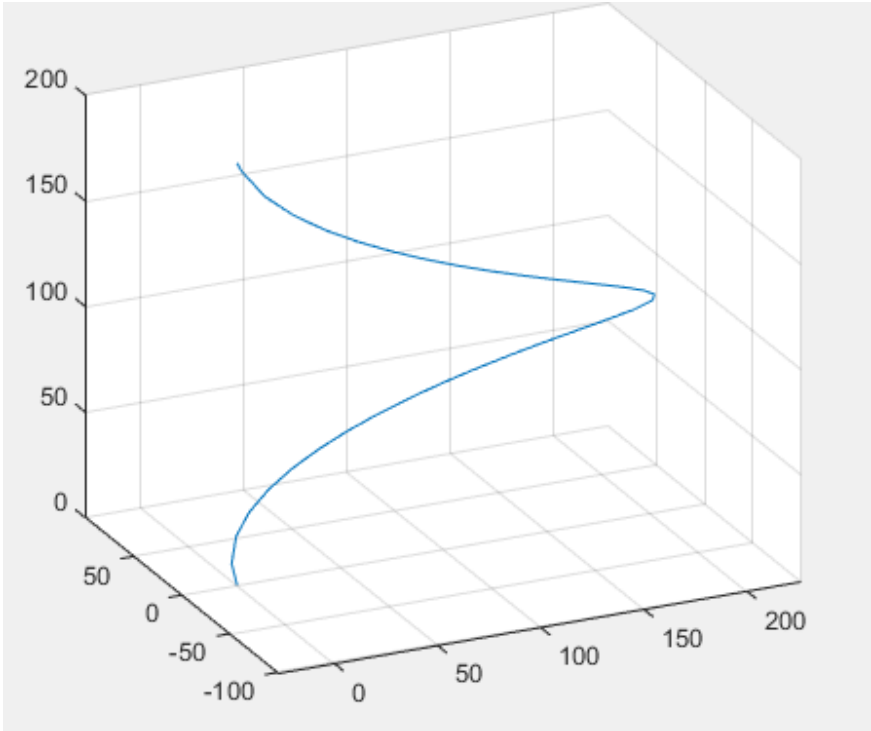
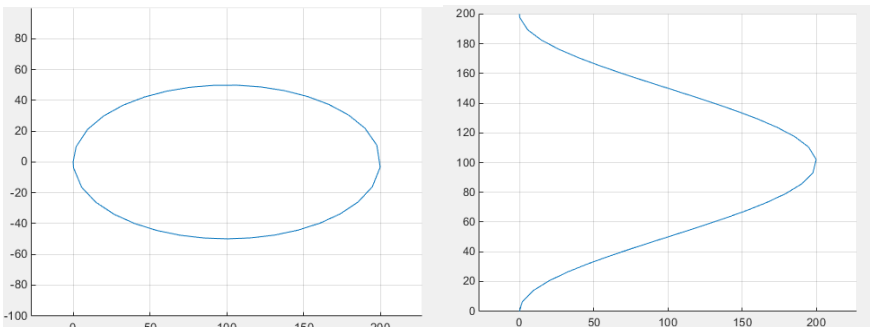
相关指令	<a href="#">MECLIPSEABS</a> , <a href="#">*SP</a>

MECLIPSEABS -- 椭圆-绝对

类型	多轴运动指令
描述	<p>椭圆插补，中心画弧，绝对运动，可选螺旋。</p> <p>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。</p> <p>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。</p> <p>可画整椭圆。</p>
语法	<p>MECLIPSEABS(end1, end2, centre1, centre2, direction, adis, bdis[, end3])</p> <p>end1: 终点第一个轴运动坐标</p> <p>end2: 终点第二个轴运动坐标</p> <p>centre1: 中心第一个轴运动坐标</p> <p>centre2: 中心第二个轴运动坐标</p> <p>direction: 0-逆时针, 1-顺时针</p>



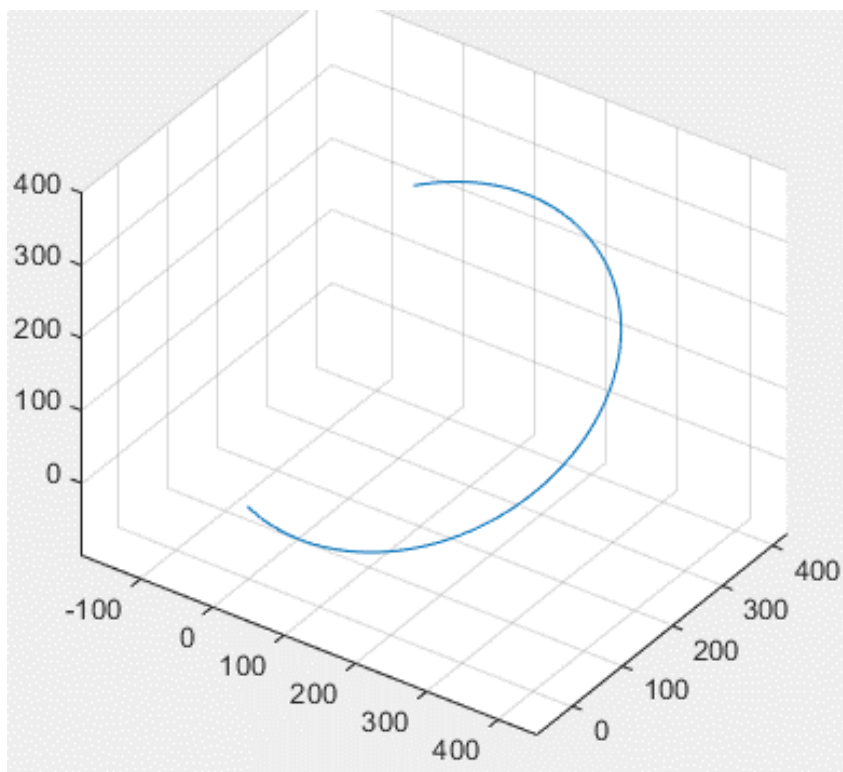
	<table><tr><th>值</th><th>描述</th></tr><tr><td>0</td><td>逆时针</td></tr><tr><td>1</td><td>顺时针</td></tr></table> <p>adis: 第一轴的椭圆半径, 半长轴或者半短轴都可</p> <p>bdis: 第二轴的椭圆半径, 半长轴或者半短轴都可, AB 相等时自动为圆弧或螺旋</p> <p>end3: 第三个轴的运动坐标, 需要螺旋时填入</p> <p>坐标位置请确保正确, 否则实际运动轨迹会错误。</p>	值	描述	0	逆时针	1	顺时针
值	描述						
0	逆时针						
1	顺时针						
适用控制器	通用						
例子	<p>例一 不进行螺旋</p> <p>BASE(0,1,2)</p> <p>ATYPE=1,1,1           '设为脉冲轴类型</p> <p>UNITS=100,100,100</p> <p>DPOS=0,0,0</p> <p>SPEED=100,100,100       '主轴速度</p> <p>ACCEL=1000,1000,1000   '主轴加速度</p> <p>DECEL=1000,1000,1000</p> <p>TRIGGER               '自动触发示波器</p> <p>MOVE(100,100)</p> <p><b>MECLIPSEABS</b>(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</p> <p>插补轨迹</p> <p>DPOS(0)垂直刻度 150</p> <p>DPOS(1)垂直刻度 150</p> <div><div>1 DPOS[0]</div><div>2 DPOS[1]</div><div><div>Min:0.00</div><div>Min:0.00</div><div>Max:300.00</div><div>Max:150.00</div></div></div> <p>例二 进行螺旋</p> <p>BASE(0,1,2)</p> <p>ATYPE=1,1,1           '设为脉冲轴类型</p> <p>UNITS=100,100,100</p> <p>DPOS=0,0,0</p> <p>SPEED=100,100,100       '主轴速度</p> <p>ACCEL=1000,1000,1000   '主轴加速度</p>						

	<p>DECEL=1000,1000,1000</p> <p><b>MECLIPSEABS</b>(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</p> <p>插补轨迹</p>  
相关指令	<a href="#">MECLIPSE</a> ， <a href="#">*SP</a>

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	<p>MSPHERICAL(end1,end2,end3,centre1,centre2,centre3,mode[,distance4] [,distance5])</p> <p>end1: 第 1 个轴运动距离参数 1</p> <p>end2: 第 2 个轴运动距离参数 1</p> <p>end3: 第 3 个轴运动距离参数 1</p>

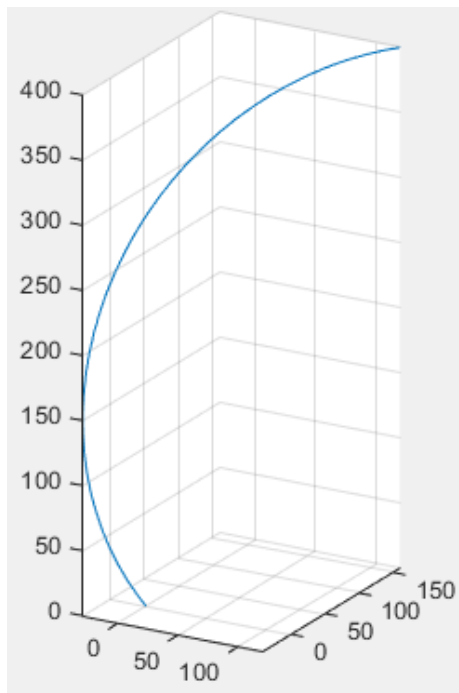
	<p>centre1: 第 1 个轴运动距离参数 2</p> <p>centre2: 第 2 个轴运动距离参数 2</p> <p>centre3 : 第 3 个轴运动距离参数 2</p> <p>mode: 指定前面参数的意义</p> <table><tr><th>值</th><th>描述</th></tr><tr><td>0</td><td>当前点, 中间点, 终点三点定圆弧 距离参数 1 为终点距离, 距离参数 2 为中间点距离</td></tr><tr><td>1</td><td>当前点, 圆心, 终点定圆弧 <b>走最短的圆弧</b> 距离参数 1 为终点距离, 距离参数 2 为圆心的距离</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 <b>先走最短的圆弧, 再继续走完整圆</b> 距离参数 1 为终点距离, 距离参数 2 为圆心的距离</td></tr></table> <p>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</p> <p>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</p> <p>坐标位置请确保正确, 否则实际运动轨迹会错误。</p>	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 <b>走最短的圆弧</b> 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 <b>先走最短的圆弧, 再继续走完整圆</b> 距离参数 1 为终点距离, 距离参数 2 为圆心的距离
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 <b>走最短的圆弧</b> 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 <b>先走最短的圆弧, 再继续走完整圆</b> 距离参数 1 为终点距离, 距离参数 2 为圆心的距离										
适用控制器	通用										
例子	<p>BASE(0,1,2)</p> <p>ATYPE=1,1,1            '设为脉冲轴类型</p> <p>UNITS=100,100,100</p> <p>DPOS=0,0,0</p> <p>SPEED=100,100,100       '主轴速度</p> <p>ACCEL=1000 ,1000,1000    '主轴加速度</p> <p>DECEL=1000 ,1000,1000</p> <p>使用圆心为(120,160,150), 半径 250 的圆演示 4 种 mode 的运动轨迹</p> <p>mode 0:</p> <p><b>MSPHERICAL</b>(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</p> <p>插补轨迹</p>										



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

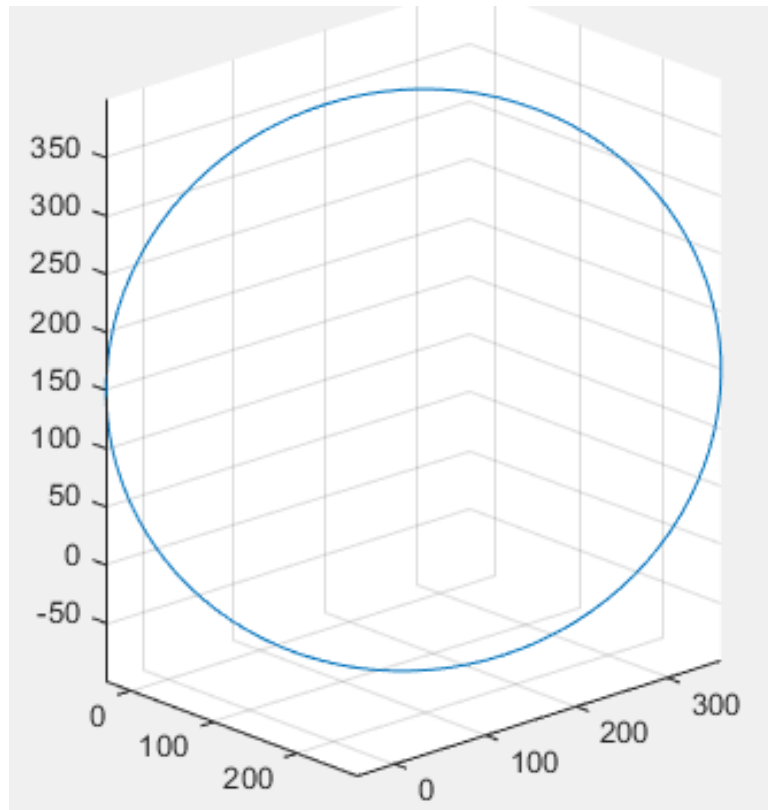


mode 2:

**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

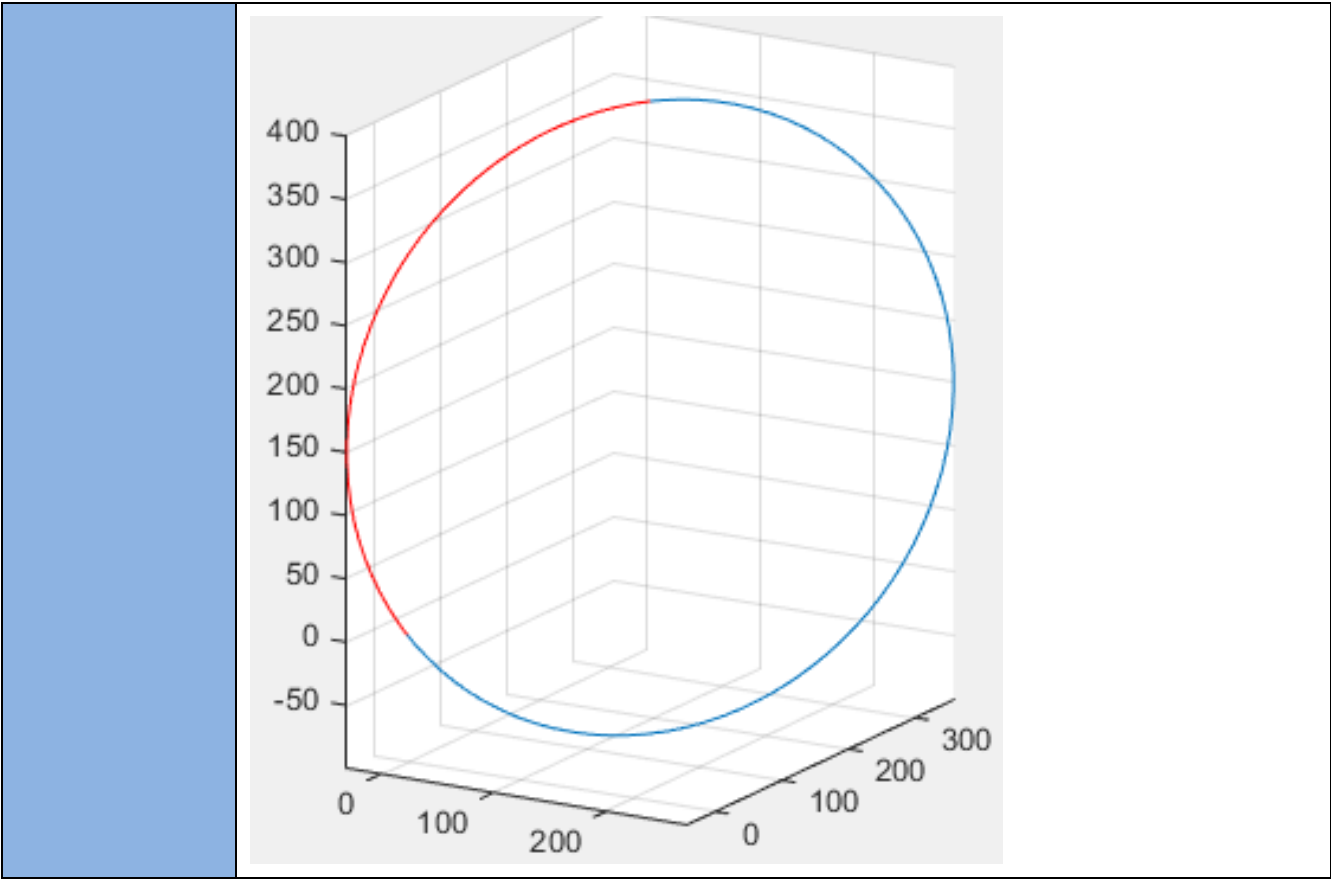
插补轨迹



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



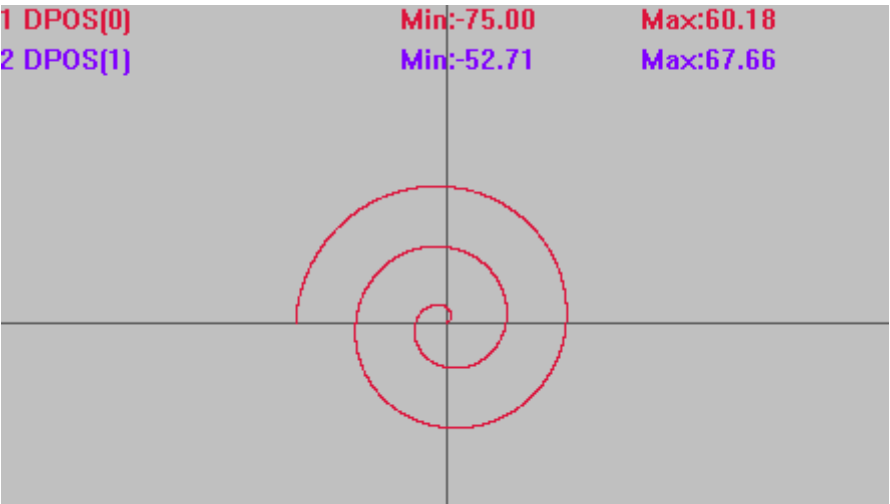
MOVESPIRAL -- 渐开线圆弧

类型	多轴运动指令
描述	<p>渐开线圆弧插补运动，相对移动方式，可选螺旋。</p> <p>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。</p> <p>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。</p>
语法	<p>MOVESPIRAL(centre1,centre2,circles,pitch[,distance3][,distance4])</p> <p>centre1: 圆心的第 1 轴相对距离</p> <p>centre2: 圆心的第 2 轴相对距离</p> <p>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</p> <p>pitch: 每圈的扩散距离，可以为负</p> <p>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</p> <p>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</p>
适用控制器	通用
例子	<p>BASE(0,1,2)</p> <p>ATYPE=1,1,1                    '设为脉冲轴类型</p> <p>UNITS=100,100,100</p>

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

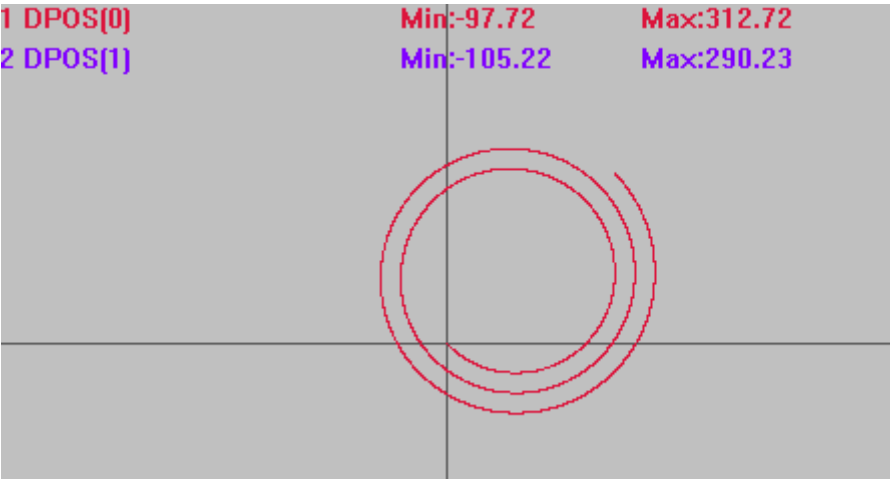
例一 从起点扩散  
**MOVESPIRAL**(0,0,2.5,30) '此时以起始位置为中心，逆时针旋转 2.5 圈，每圈扩散 30

插补轨迹  
DPOS(0)垂直刻度 100  
DPOS(1)垂直刻度 100

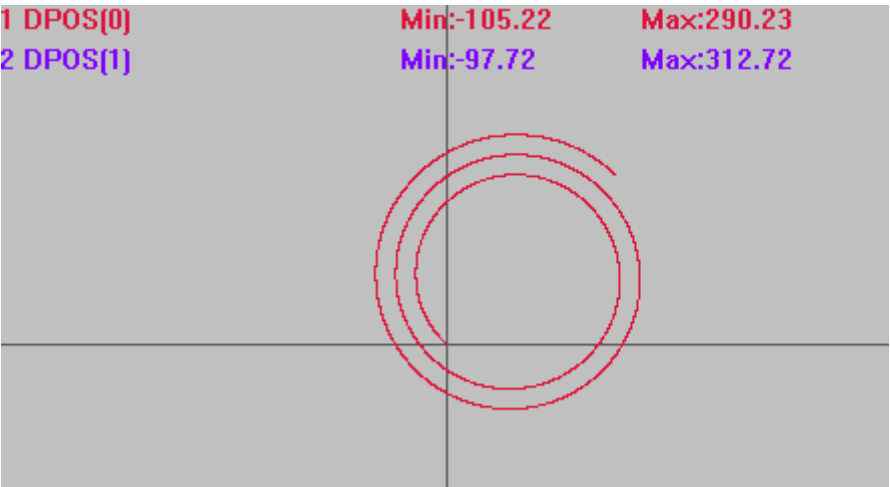


例二 不螺旋  
**MOVESPIRAL** (100,100,2.5,30) '起始半径 100，以(100,100)为圆心，逆时针旋转 2.5 圈，每圈向外扩散 30

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



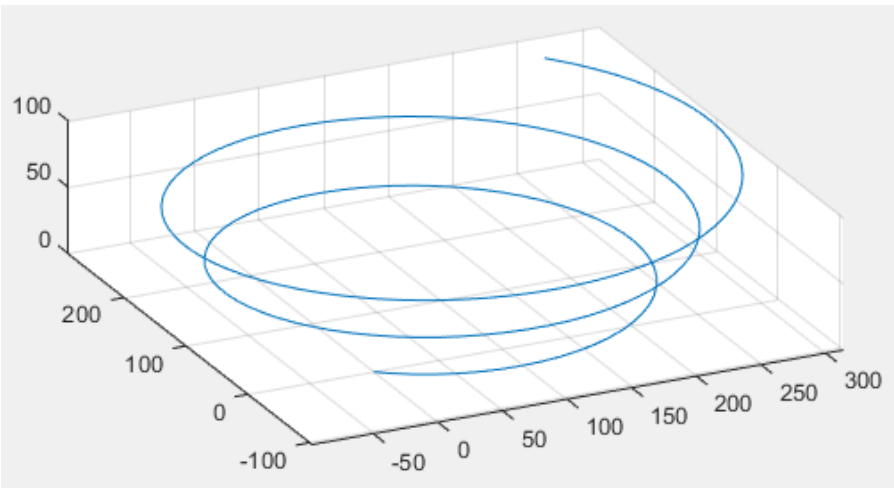
**MOVESPIRAL** (100,100,-2.5,30) '旋转圈数为负数时(-2.5)，顺时针旋转



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

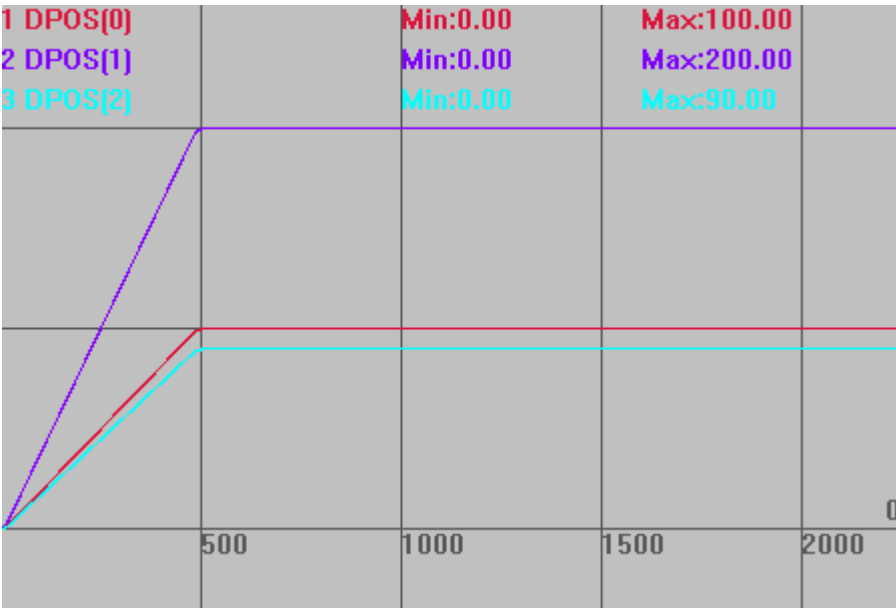
类型	特殊运动指令
描述	<p>样条插补运动，相对/绝对运动。</p> <p>样条运动的控制点位置提前写入 TABLE 数组。</p> <p>此运动没有 SP 指令，自定义速度的连续插补运动通过 CORNER_MODE 的 BIT8 来设置。</p>
语法	<p><b>MOVESPLINE</b> (axes,mode ,dtendcontrol4, dtcontrol2, dtcontrol3)</p> <p>axes: 插补的轴个数</p> <p>mode: 模式, 0-使用 3 阶贝塞尔样条</p>



	<p>dtendcontrol4: 第 4 个控制点存储 table 索引, 对贝塞尔曲线就是终点</p> <p>dtcontrol2: 第 2 个控制点存储 table 索引</p> <p>dtcontrol3: 第 3 个控制点存储 table 索引</p> <p>对贝塞尔曲线, 第 1 个控制点就是当前位置。</p>
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	<p>BASE(0,1)</p> <p>DPOS=0,0</p> <p>ATYPE=1,1            '设为脉冲轴类型</p> <p>SPEED=100,100        '主轴速度</p> <p>ACCEL=1000,1000      '主轴加速度</p> <p>DECEL=1000,1000</p> <p>TRIGGER</p> <p>CORNER_MODE=2 + 256   '设置 SP 运动, 使用 FORCE_SPEED</p> <p>FORCE_SPEED=100</p> <p>TABLE(0, 100, 100)   'TABLE(0)(1)存储第二个控制点, 相对起点距离</p> <p>TABLE(10, 200, -100) 'TABLE(10)(11)存储第三个控制点, 相对起点距离</p> <p>TABLE(20, 300, 0)    'TABLE(20)(21)存储第四个控制点, 终点距离</p> <p><b>MOVESPLINE</b>(2, 0, 20, 0, 10)   '2 轴相对样条插补</p> <p>插补轨迹</p> <p>DPOS(0)垂直刻度 200</p> <p>DPOS(1)垂直刻度 200</p>
相关指令	<a href="#">CORNER_MODE</a>

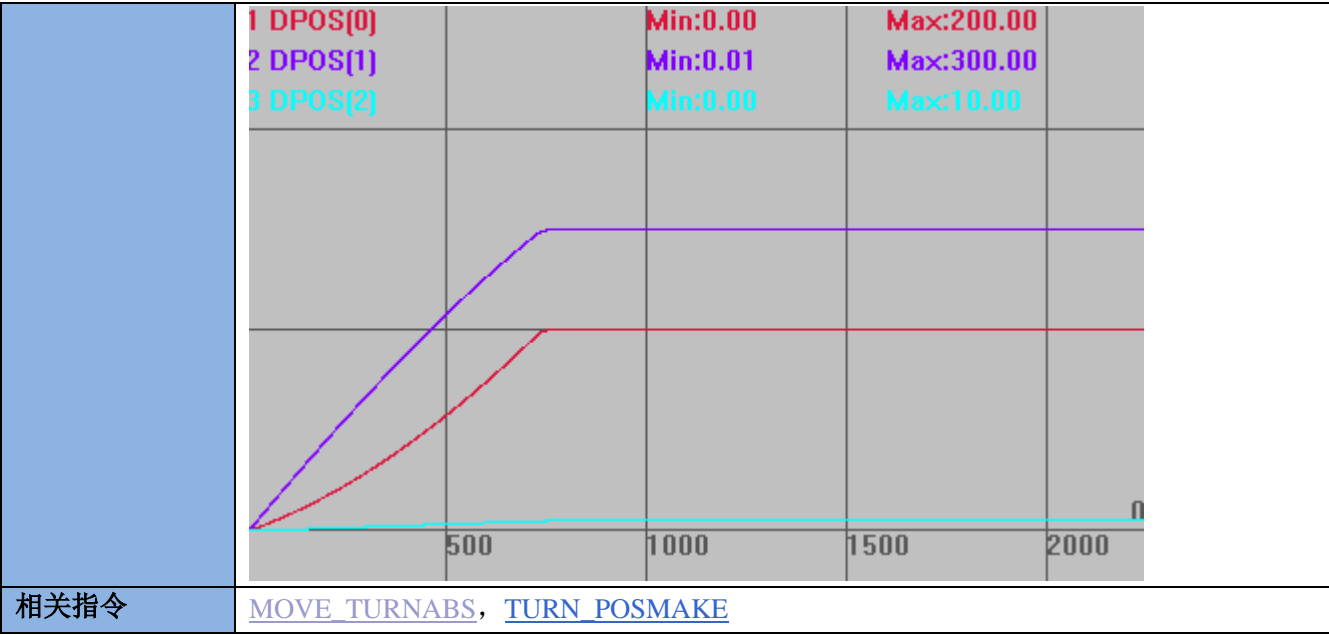
MOVE\_TURNABS -- 旋转台插补

类型	多轴运动指令
描述	<p>旋转的插补运动, 保证在旋转台上是直线运动。</p> <p>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)。</p>

	<div>旋转的参数存储在 <b>TABLE</b> 表里面，依次存储 <b>R</b> 轴号，<b>R</b> 轴一圈脉冲数，<b>X</b> 轴号，<b>Y</b> 轴号，<b>X</b> 圆心，<b>Y</b> 圆心。</div> <div>自定义速度的连续插补运动可以使用 <b>SP</b> 后缀的指令，见*<b>SP</b> 描述。</div> <div>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 <b>frame11/17</b>。</div>															
语法	<div>MOVE_TURNABS(tablenum,position1[,position2[,position3[,position4...]]])</div> <div>tablenum: 存储旋转参数的 table 编号</div> <div>position1: 第一个轴运动坐标</div> <div>position2: 下一个轴运动坐标</div>															
适用控制器	通用															
例子	<div>BASE(0,1,2)</div> <div>ATYPE=1,1,1            '设为脉冲轴类型</div> <div>UNITS=100,100,100</div> <div>DPOS=0,0,0</div> <div>SPEED=100,100,100       '主轴速度</div> <div>ACCEL=1000 ,1000,1000   '主轴加速度</div> <div>DECEL=1000 ,1000,1000   '主轴减速度</div> <div>TABLE(0, 3, 3600, 0,1, 0,0) '设置旋转台的参数</div> <div>TRIGGER</div> <div>MOVE_TURNABS(0,100,200,90) '直线插补至目标位置</div> <div>WAIT IDLE                '等待运动停止</div> <div>插补轨迹</div> <div>DPOS(0)垂直刻度 100</div> <div>DPOS(1)垂直刻度 100</div> <div>DPOS(2)垂直刻度 100</div> <div><table><tr><td>1 DPOS[0]</td><td></td><td>Min:0.00</td><td>Max:100.00</td><td></td></tr><tr><td>2 DPOS[1]</td><td></td><td>Min:0.00</td><td>Max:200.00</td><td></td></tr><tr><td>3 DPOS[2]</td><td></td><td>Min:0.00</td><td>Max:90.00</td><td></td></tr></table></div>	1 DPOS[0]		Min:0.00	Max:100.00		2 DPOS[1]		Min:0.00	Max:200.00		3 DPOS[2]		Min:0.00	Max:90.00	
1 DPOS[0]		Min:0.00	Max:100.00													
2 DPOS[1]		Min:0.00	Max:200.00													
3 DPOS[2]		Min:0.00	Max:90.00													
相关指令	<a href="#">MCIRC_TURNABS</a>															

## MCIRC\_TURNABS -- 旋转台插补-绝对

类型	多轴运动指令
描述	<p>旋转的插补运动，保证在旋转台上是圆弧运动。</p> <p>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）。</p> <p>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。</p> <p>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。</p>
语法	<p>MCIRC_TURNABS(tablenum, refpos1, refpos2, mode,end1, end2 [, dis3, dis4, dis5])</p> <p>tablenum: 存储旋转参数的 table 编号</p> <p>refpos1: 第一个轴参考点，绝对位置</p> <p>refpos2: 第二个轴参考点，绝对位置</p> <p>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</p> <p>end1: 第一个轴结束点，绝对位置</p> <p>end2: 第二个轴结束点，绝对位置</p> <p>dis3: 螺旋轴的结束位置</p>
适用控制器	通用
例子	<pre> 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 </pre>



MOVESMOOTH -- 倒圆角

类型	多轴运动指令
描述	<p>空间直线倒圆运动。</p> <p>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。</p> <p>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。</p> <p>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。</p>
语法	<p>MOVESMOOTH (end1, end2, end3, next1, next2, next3, radius)</p> <p>end1: 第 1 个轴运动绝对坐标</p> <p>end2: 第 2 个轴运动绝对坐标</p> <p>end3: 第 3 个轴运动绝对坐标</p> <p>next1: 第 1 个轴下一个直线运动绝对坐标</p> <p>next2: 第 2 个轴下一个直线运动绝对坐标</p> <p>next3: 第 3 个轴下一个直线运动绝对坐标</p> <p>radius: 插入圆弧的半径，当过大的时候自动缩小</p>
适用控制器	通用
例子	<p>BASE(0,1,2)</p> <p>ATYPE=1,1,1            '设为脉冲轴类型</p> <p>UNITS=100,100,100</p> <p>DPOS=0,0,0</p> <p>SPEED=100,100,100       '主轴速度</p> <p>ACCEL=1000 ,1000,1000   '主轴加速度</p> <p>DECEL=1000 ,1000,1000   '主轴减速度</p> <p>TRIGGER                 '自动触发示波器</p> <p>MOVESMOOTH (0,100,0,100,100,0,50) '加入圆弧后，实际运动到了'(50,100,0)</p>

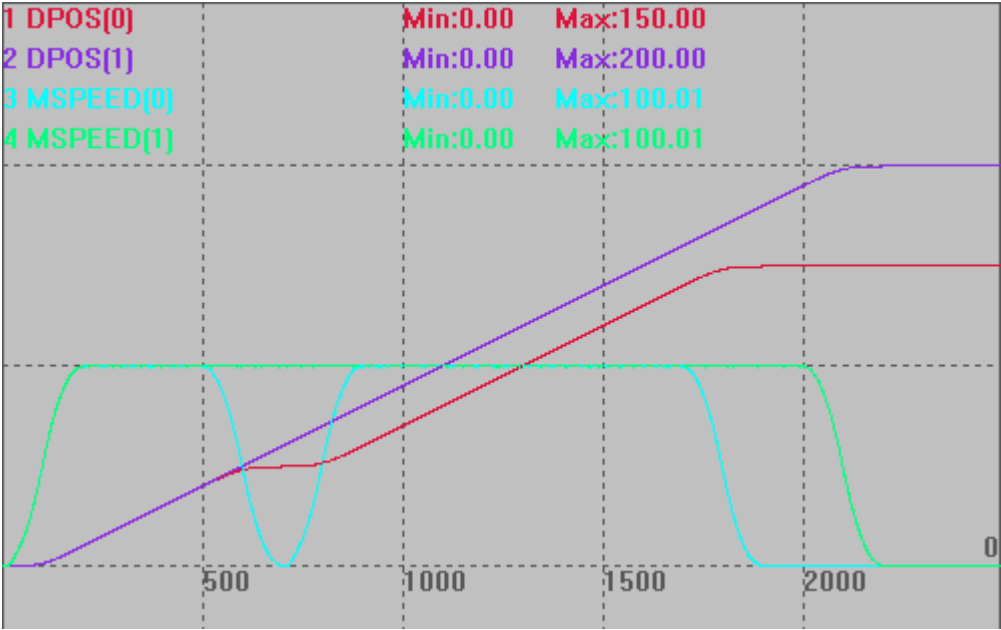
	<div><div>MOVEABS(100,100,0)</div><div>插补轨迹</div><div>DPOS(0)垂直刻度 100</div><div>DPOS(1)垂直刻度 100</div><div><div><div>1 DPOS[0]</div><div>2 DPOS[1]</div></div><div><div>Min:0.00</div><div>Min:0.01</div></div><div><div>Max:100.00</div><div>Max:100.00</div></div></div></div>
--	---

MOVEMODIFY2 -- 修改多轴运动位置

类型	多轴运动指令
描述	<div>修改上一个运动的目标位置。</div> <div>MOVEMODIFY2 是以各自轴的速度与相关速度参数运行的。</div> <div>前面没有运动时与 MOVEABS 效果一样，但不会进入运动缓冲。</div> <div>连续插补时使用 MOVEMODIFY2 会破坏速度连续性。</div> <div>MOVEMODIFY2 同时对多轴运动时不一定为直线插补。</div>
语法	<div>MOVEMODIFY (distance1, distance2,...)</div> <div>distance1: 第一个轴的运动距离</div> <div>distance2: 第二个轴的运动距离</div>
适用控制器	通用
例子	<div>例一</div> <div>BASE(0,1)</div> <div>ATYPE=1,1            '设为脉冲轴类型</div> <div>DPOS=0,0</div> <div>SPEED = 100,100</div> <div>ACCEL=1000,1000    '加速度设置</div> <div>DECEL=1000,1000</div> <div>SRAMP=100,100</div> <div>TRIGGER</div> <div>MOVE(200)</div> <div>MOVEMODIFY2(50,200)    '取消 MOVE(200)，强制定位新的位置 50,200</div>

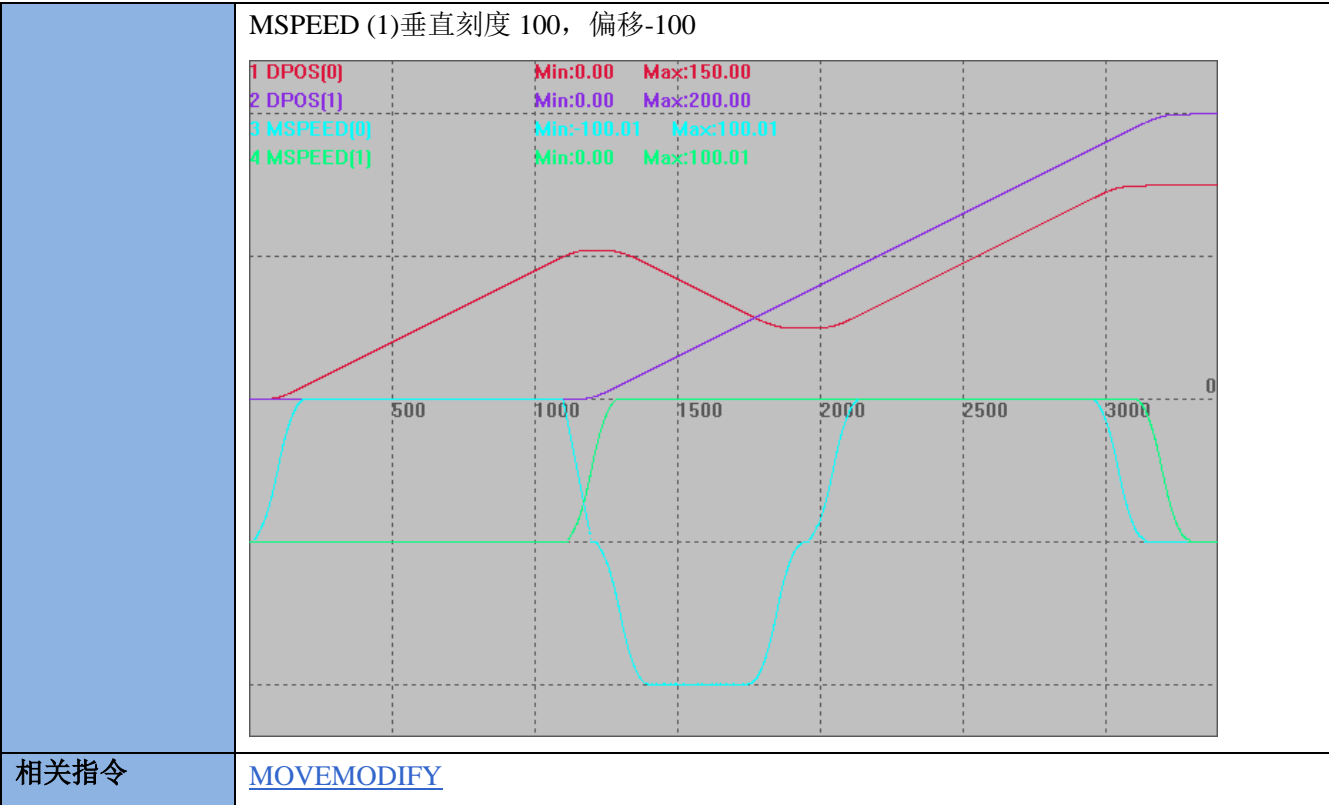
MOVE(100) '最终位置(50+100,200)  
END

运动轨迹  
DPOS(0)垂直刻度 100  
DPOS(1)垂直刻度 100  
MSPEED(0)垂直刻度 100  
MSPEED (1)垂直刻度 100



例二  
BASE(0,1)  
ATYPE=1,1 '设为脉冲轴类型  
DPOS=0,0  
SPEED = 100,100  
ACCEL=1000,1000 '加速度设置  
DECEL=1000,1000  
SRAMP=100,100  
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  
DPOS(0)垂直刻度 100  
DPOS(1)垂直刻度 100  
MSPEED(0)垂直刻度 100， 偏移-100



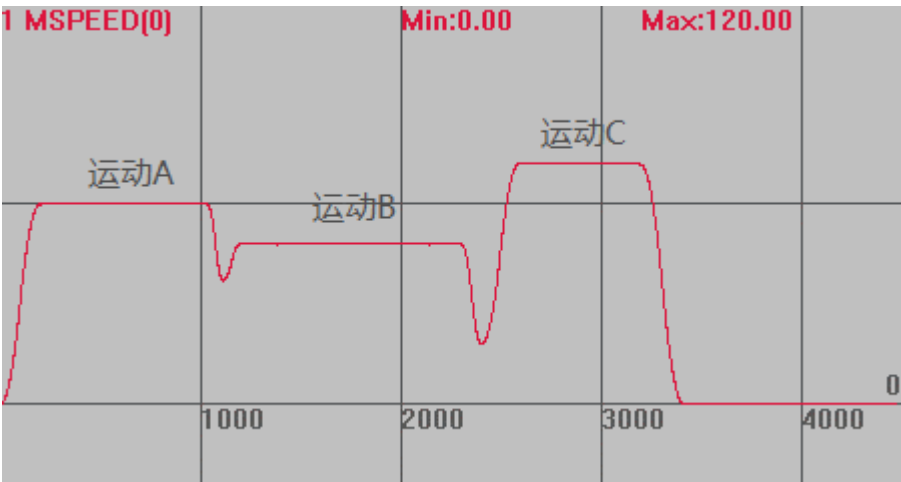
\*SP -- 运动单独速度

类型	多轴运动指令
描述	<p>用于设置每段运动的运行速度、起始速度和终止速度。</p> <p>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。</p>
语法	<p>SP 指令有：MOVESP，MOVEABSSP，MOVECIRCSP，MOVECIRABSSP，MHELICALSP，MHELICALABSSP，MECLIPSESP，MECLIPSEABSSP，MSPHERICALSP。</p> <p>FORCE_SPEED 、ENDMOVE_SPEED 和 STRATMOVE_SPEED 会随 SP 运动指令写入运动缓存区。</p>
适用控制器	通用
例子	<p>例一</p> <p>BASE(0)</p> <p>ATYPE=1</p> <p>UNITS=100</p> <p>DPOS=0</p> <p>ACCEL=1000</p> <p>DECEL=1000</p>

```
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

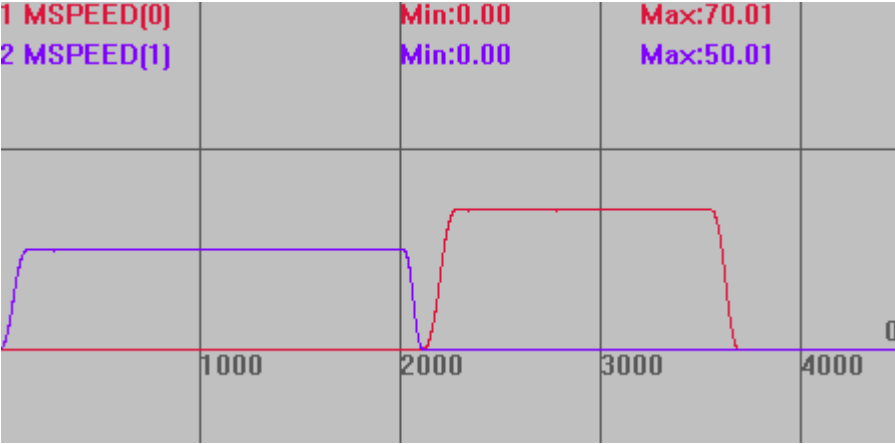


不使用 SP 指令时，运行速度为 SPEED，使用 SP 指令后，运行速度为 FORCE\_SPEED。当 STARTMOVE\_SPEED 和 ENDMOVE\_SPEED 同时设置时，STARTMOVE\_SPEED 生效。

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
```



	<div>ENDMOVE_SPEED = 1000      '设置一个较大的结束速度</div> <div>FORCE_SPEED= 50      '每段速度受 speed 限制，将 speed 设置为一个较大值即可</div> <div>TRIGGER      '自动触发示波器</div> <div>MOVESP(0,100)      '该段速度为 50units/s</div> <div>FORCE_SPEED = 70      '每段速度受 speed 限制，将 speed 设置为一个较大值即可</div> <div>MOVESP(100,0)      '该段速度为 70units/s</div> <div><div>速度轨迹</div><div>MSPEED(0)垂直刻度 100</div><div>MSPEED(1)垂直刻度 100</div><div></div></div>
相关指令	<a href="#">FORCE_SPEED</a> ， <a href="#">ENDMOVE_SPEED</a> ， <a href="#">STRATMOVE_SPEED</a>

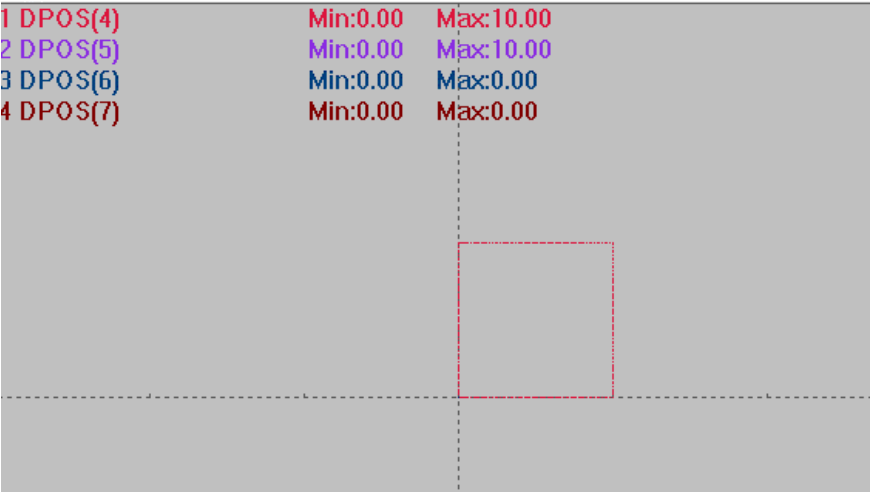
MOVESCAN -- 振镜运动

类型	运动指令
描述	<div>不带加减速的运动指令，支持 us 级别的时间控制。</div> <div>使用 FORCE_SPEED 与矢量距离直接计算出运行时间。</div> <div>支持 MOVESCANABS 绝对运动。</div> <div>需要振镜控制器固件版本 20180714 以上。</div> <div><div>拐角延时、ZSMOOTH 在此运动下意义为最大的拐角延时、实际的延时时间在 DECEL_ANGLE 与 STOP_ANGLE 之间线性分布。</div><div>CORNER_MODE 的 bit1 设置是否使用拐角延时。</div><div>可以与 MOVE_WAIT、MOVE_OP 一起实现 us 级别的时间控制。</div><div>非振镜轴也可以使用，但要自己分段控制速度来做加减速。</div></div>
语法	<div>MOVESCAN(pos1[,pos2] [,pos3]...)</div> <div>pos1: 第一个轴运动距离</div> <div>pos2: 下一个轴运动距离</div>
适用控制器	振镜控制器
例子	<div>例一</div> <div>BBASE(4,5)</div> <div>AXIS_ZSET=2      '开启精准输出</div>

```
TRIGGER
CORNER_MODE=0      '无拐角延时
MOVE_PAUSE(3)      '强制暂停
MOVE_OP(0,1)
FORCE_SPEED=200
MOVESCANABS(0,0)
MOVESCANABS(10,0)   '100us 的时间
MOVESCANABS(10,10)  '100us 的时间
MOVESCANABS(0,10)   '100us 的时间
MOVESCANABS(0,0)    '100us 的时间
MOVE_DELAY(0.25)    '延时 250us
MOVE_OP(0,0)        '450us 以后输出
MOVE_RESUME
END
```

振镜轴 XY 模式下合成轨迹如下

DPOS(4)垂直刻度 10  
DPOS(5)垂直刻度 10



例二

```
BASE(4,5)
AXIS_ZSET=2
CORNER_MODE=2      '拐角延时
ZSMOOTH=100        '拐角最大延时 100us
DECEL_ANGLE = 25 * (PI/180)  '设置开始减速拐角，弧度
STOP_ANGLE = 90 * (PI/180)  '设置结束减速拐角，弧度
MOVE_PAUSE(3)
MOVE_OP(0,1)
FORCE_SPEED=10000
MOVESCAN(1,0)
MOVESCAN(0,1)      '前面会加 100us 延时
MOVE_DELAY(0.25)
```

	MOVE_OP(0,0) MOVE_RESUME	'550us 以后输出
相关指令	<a href="#">MOVE</a>	

## 7.3. 特殊运动指令

### MOVE\_PAUSE -- 运动暂停

类型	特殊运动指令									
描述	<b>BASE 轴运动暂停。</b> 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 <b>AXISSTATUS</b> 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 <b>VMOVE</b> 、同步运动指令等。									
语法	<b>MOVE_PAUSE (mode)</b> <b>mode:</b> 暂停方式 <table><tr><td>0( 缺省 )</td><td>暂停当前运动。</td></tr><tr><td>1</td><td>在当前运动完成后正准备执行下一条运动指令时暂停</td></tr><tr><td>2</td><td>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 <b>MARK</b> 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</td></tr><tr><td>3</td><td>强制暂停，<b>IDLE</b> 模式下也可以进入暂停状态 <b>20170513 固件版本增加此功能</b></td></tr></table>		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 <b>MARK</b> 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， <b>IDLE</b> 模式下也可以进入暂停状态 <b>20170513 固件版本增加此功能</b>
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 <b>MARK</b> 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， <b>IDLE</b> 模式下也可以进入暂停状态 <b>20170513 固件版本增加此功能</b>									
适用控制器	通用									
例子	<b>BASE(0)</b> <b>DPOS=0</b> <b>SPEED=100</b>  例一 <b>mode 0</b> <b>MOVE(1000)</b> '当前运动 <b>MOVEABS(-100)</b> '缓冲运动 <b>MOVE_PAUSE(0)</b> '模式 0，暂停当前运动 <b>?DPOS(0)</b> '打印结果，0 '此时当前运动只运行了极短时间，扫描到 <b>MOVE_PAUSE</b> 时直接暂停  例二 <b>mode 1</b> <b>MOVE(1000)</b> '当前运动 <b>MOVEABS(-100)</b> '缓冲运动 <b>MOVE_PAUSE(1)</b> '模式 1，先完成当前运动再暂停 <b>WAIT IDLE</b>									

	?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 <b>MOVE_PAUSE(2)</b> '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	<a href="#">MOVE_MARK</a> , <a href="#">MOVE_RESUME</a> , <a href="#">AXISSTATUS</a>

## MOVE\_RESUME -- 运动恢复

类型	特殊运动指令
描述	当 <b>BASE</b> 轴暂停时, 从暂停处继续运动。 可以通过 <b>AXISSTATUS</b> 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) <b>MOVE_RESUME</b> '继续运行 WAIT IDLE ?DPOS(0) '打印结果, 200
相关指令	<a href="#">MOVE_PAUSE</a> , <a href="#">AXISSTATUS</a>

MOVE\_PT -- 单位时间距离

类型	特殊运动指令
描述	<p>在一段时间内驱动电机运动设置的距离。</p> <p>一般是 PC 每个周期计算好对应的坐标，然后传给控制器。</p> <p>运动时的速度=(DIS/TICKS)*1000 units/s。</p> <p>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。</p>
语法	<p>MOVE_PT (ticks, dis1,dis2···)</p> <p>ticks: 时间的长度。ticks 会自己不断减 1，1ticks 大约 1ms</p> <p>dis1: 运动的距离</p> <p>SERVO_PERIOD 为 1000us 的控制器 1 TICKS 为 1ms(不同 SERVO_PERIOD 的 TICKS 时间不一样)</p>
适用控制器	通用
例子	<p>例一</p> <pre>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</pre> <p>插补速度 DPOS(0)垂直刻度 100 MSPEED(0)垂直刻度 2000</p>

相关指令	<a href="#">MOVE_PTABS</a>
------	----------------------------

## MOVE\_PTABS -- 单位时间距离绝对

类型	特殊运动指令
描述	<p>在一段时间内驱动电机运动到某个位置。</p> <p>一般是 PC 每个周期计算好对应的坐标，然后传给控制器。</p> <p>运动时的速度=(DIS/TICKS)*1000 units/s。</p> <p><b>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。</b></p>
语法	<p>MOVE_PTABS(ticks,dis1,dis2,...)</p> <p>ticks: 时间的长度。ticks 会自己不断减 1, 1ticks 大约 1ms</p> <p>dis1: 运动的绝对位置</p>
适用控制器	通用
例子	<p>例一</p> <pre> BASE(0,1) DPOS=0,0 <b>MOVE_PTABS</b> (3, 20,20)      '3tick 时间内运动到到 20,20 WAIT IDLE PRINT *DPOS                  '打印结果, 20,20 </pre> <p>例二</p> <pre> RAPIDSTOP(2) WAIT IDLE(0)  BASE(0) ATYPE=1 UNITS=100 SPEED=100 ACCEL=1000 DECEL=1000 DPOS = 0  SetSine                      '调用函数，产生 SINE 曲线 TRIGGER                      '自动触发示波器  FOR i=0 TO 100     <b>MOVE_PTABS</b> (1, TABLE(i))      '在 1TICK 内，运动 TABLE 距离 NEXT WAIT IDLE(0) PRINT DPOS(0)                '打印结果 500 END </pre>

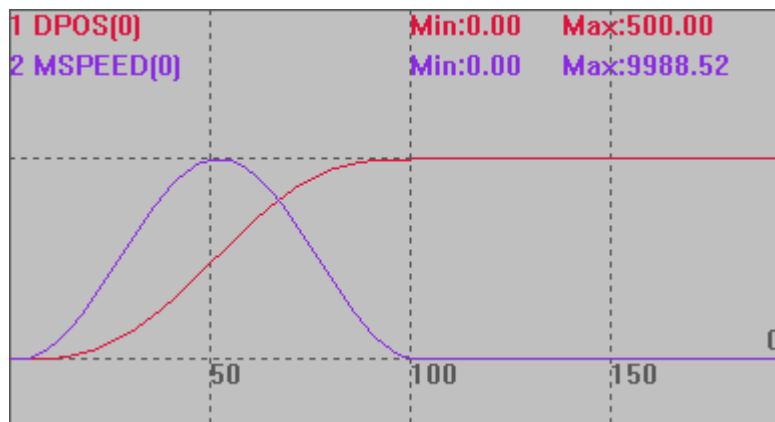
```

GLOBAL SUB SetSine()           '计算小段位移
    LOCAL num_p,scale         '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) '存储运动参数
    NEXT
END SUB

```

DPOS(0)竖直刻度 500

MSPEED(0)竖直刻度 10000

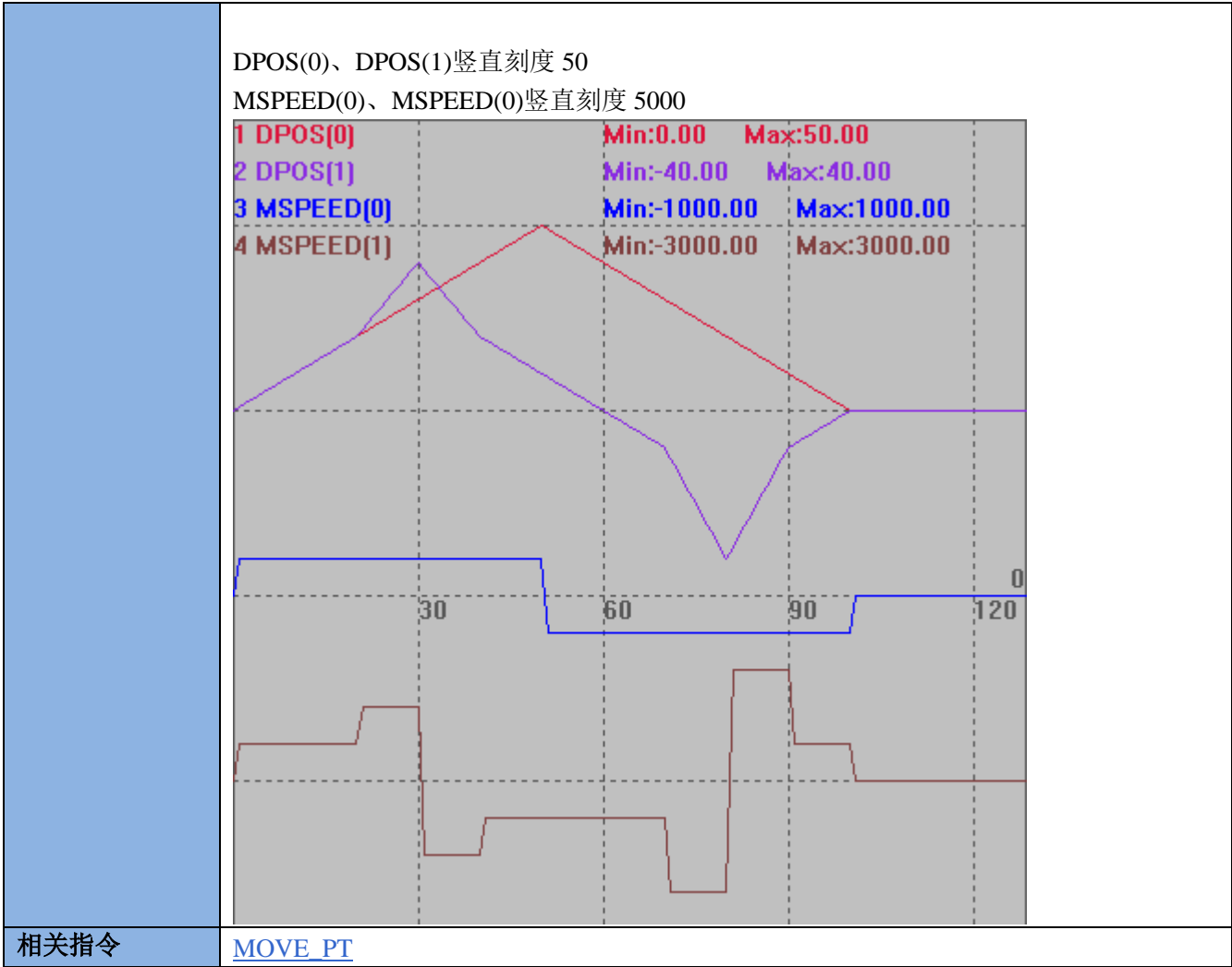


例二

```

RAPIDSTOP(2)
WAIT IDLE(0)
WAIT IDLE(1)
BASE(0,1)
ATYPE=1,1
UNITS=100,100
DPOS=0,0
TRIGGER
MOVE_PTABS (10,10,10) '10tick 时间内运动到绝对位置(10,10)
MOVE_PTABS (10,20,20)
MOVE_PTABS (10,30,40)
MOVE_PTABS (10,40,20)
MOVE_PTABS (10,50,10)
MOVE_PTABS (10,40,0)
MOVE_PTABS (10,30,-10)
MOVE_PTABS (10,20,-40)
MOVE_PTABS (10,10,-10)
MOVE_PTABS (10,0,0)
END

```

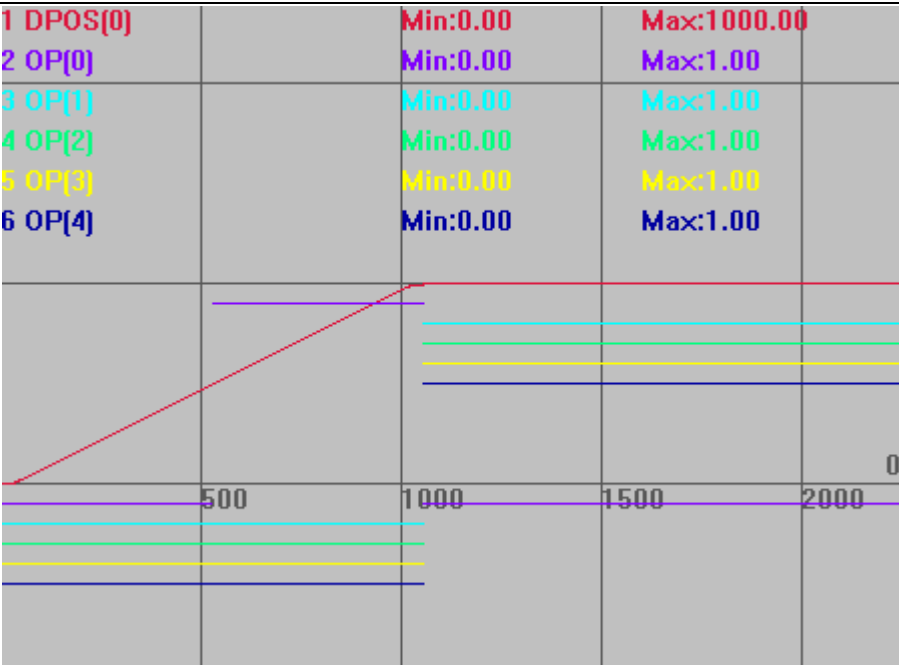


MOVE\_OP -- 缓冲输出

类型	特殊运动指令
描述	<p><b>BASE 轴运动缓冲加入一个输出口操作。</b></p> <p>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。</p> <p>普通模式，误差在一个扫描周期，所有控制器都可使用。</p> <p>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。</p> <ol style="list-style-type: none"> <li>只有支持硬件比较输出的 OP 口才支持精准输出功能。</li> <li>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）</li> <li>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。</li> </ol>



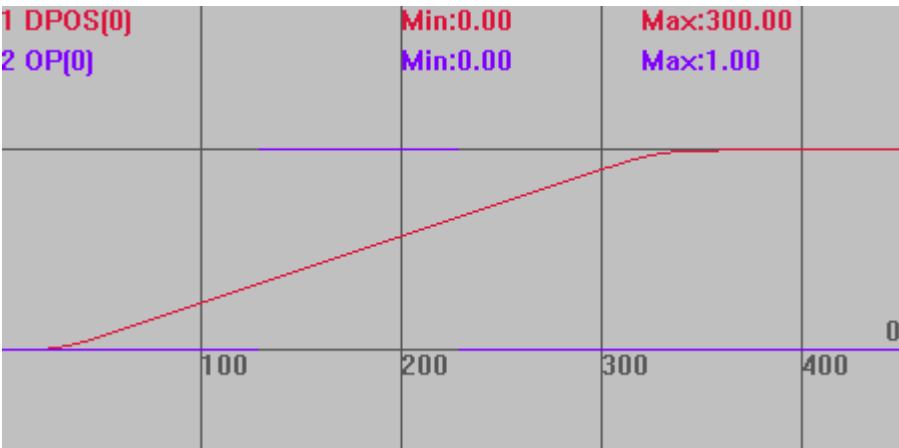
	4. <b>MOVE_OP</b> 精准功能基于 <b>BASE</b> 主轴，多个轴插补时，与 <b>BASE</b> 主轴 <b>ATYPE</b> 类型不一样的从轴，无法保证精准位置输出。
语法	<p>语法一: <b>MOVE_OP</b> ([ionum],value)  ionum: 输出编号，从 0 开始  value: 输出状态，多个输出口操作时按位来指明多个口状态</p> <p>语法二: <b>MOVE_OP</b> (ionum1, ionum2,value[,mask])  ionum1: 要操作的第一个输出通道  ionum2: 要操作的最后一个输出通道  value: 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</p>
适用控制器	通用
例子	<p>例一：普通模式使用</p> <p><b>BASE</b>(0)  <b>UNITS</b>=100  <b>DPOS</b>=0  <b>SPEED</b>=200  <b>ACCEL</b>=1000  <b>DECEL</b>=1000  <b>TRIGGER</b>                    '自动触发示波器  <b>MOVE</b>(500)  <b>MOVE_OP</b> (0,ON)            '等待上条运动完成后，OUT0 口输出信号  <b>MOVE</b>(500)  <b>MOVE_OP</b> (0,OFF)           '等待上条运动完成后，OUT0 口关闭信号  <b>MOVE_OP</b>(1,4,15)           'OUT1-4 口输出信号，15 对应二进制 1111</p> <p>轨迹曲线，为方便查看，进行了竖直偏移  <b>DPOS</b>(0)垂直刻度 1000  <b>OP</b>(0)垂直刻度 1,偏移-0.1  <b>OP</b>(1)垂直刻度 1,偏移-0.2  <b>OP</b>(2)垂直刻度 1,偏移-0.3  <b>OP</b>(3)垂直刻度 1,偏移-0.4  <b>OP</b>(4)垂直刻度 1,偏移-0.5</p>



例二：精准位置输出模式

```
BASE(0)
UNITS=100
DPOS=0
SPEED=200
ACCEL=1000
DECEL=1000
TRIGGER          '自动触发示波器
ATYPE=1
MERGE=1
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
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)

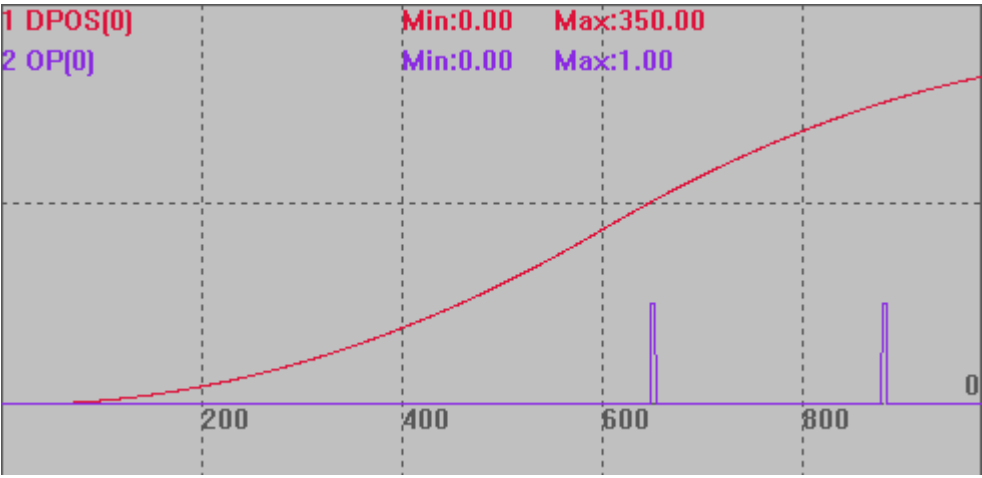
轨迹曲线
DPOS(0)垂直刻度 300
OP(0)垂直刻度 1
```



```
例三：编码器精准位置输出模式
20170505 以上固件版本支持
DIM opnum
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 = 2 '实际输出时间延迟 2ms
HW_TIMER(0)

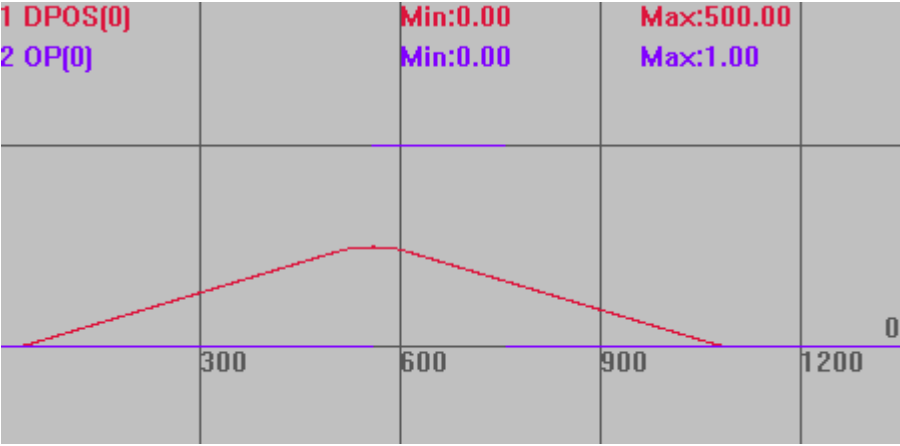
OP(opnum,0) '初始化 OP
HW_TIMER(2, 10000, 5000, 1, 0, opnum) '输出变为 on 后, 5000us 后切换为 off
MOVE(200)
MOVE_OP(opnum,1) '输出 on 后靠 HW_TIMER 来延时关闭，延时 5ms 后关闭
MOVE(100)
MOVE_OP(opnum,1)
MOVE(50)
END

轨迹曲线
DPOS(0)垂直刻度 200
OP(0)垂直刻度 2
```

	
相关指令	<a href="#">OP</a> , <a href="#">MOVE_OP2</a> 精准位置输出模式 <a href="#">SYSTEM_ZSET</a> , <a href="#">AXIS_ZSET</a>

## MOVE\_OP2 -- 缓冲输出 2

类型	特殊运动指令
描述	<b>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。</b>  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 <b>MOVE_OP2</b> 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) <b>MOVE_OP2</b> (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

	<div>OP(0)垂直刻度 1</div> <div><div><div>1 DPOS[0]</div><div>2 OP[0]</div></div><div><div>Min:0.00</div><div>Min:0.00</div></div><div><div>Max:500.00</div><div>Max:1.00</div></div></div> <div></div>
相关指令	<a href="#">MOVE_OP</a> , <a href="#">OP</a>

MOVE\_TABLE -- 缓冲 Table

类型	特殊运动指令
描述	<b>BASE 轴运动缓冲加入一个 TABLE。</b> 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	<div>BASE(0)</div> <div>UNITS=100</div> <div>DPOS=0</div> <div>SPEED=200</div> <div>ACCEL=1000</div> <div>DECEL=1000</div> <div>TABLE(0)=0               '先令 TABLE(0)=0</div> <div>?TABLE(0)               '打印确认 TABLE(0)的值</div> <div>                          '打印结果，0</div> <div>TRIGGER                '自动触发示波器</div> <div>MOVE(500)</div> <div><b>MOVE_TABLE(0, 60)</b>       '等待运动完成后，TABLE(0)赋值 60</div> <div>MOVE(500)</div> <div>WAIT IDLE</div> <div>?TABLE(0)               '打印修改后 TABLE(0)的值，打印结果，60</div> <div> 轨迹曲线</div> <div>DPOS(0)垂直刻度 1000</div>

	<div>TABLE(0)垂直刻度 100</div> <div><div><div>1 DPOS[0]</div><div>2 TABLE[0]</div></div><div><div>Min:0.00</div><div>Min:0.00</div></div><div><div>Max:1000.00</div><div>Max:60.00</div></div></div> <div></div>
相关指令	<a href="#">TABLE</a>

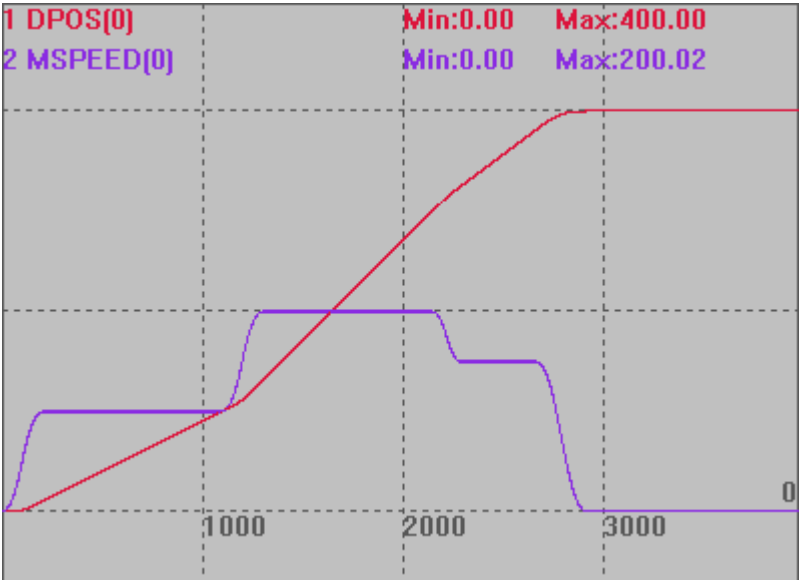
MOVE\_PARA -- 缓冲参数

类型	特殊运动指令
描述	<b>BASE 轴运动缓冲修改参数。</b> 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	<div>例一：修改 SPEED</div> <div>BASE(0)       '选择轴 0</div> <div>ATYPE=1</div> <div>SPEED=100</div> <div>PRINT SPEED    '打印结果 100</div> <div>MOVE_PARA(speed ,0,200)   '修改轴 0 的 SPEED 参数值为 200</div> <div>DELAY(1000)</div> <div>PRINT SPEED    '打印结果 200</div> <div>例二：单轴变速</div> <div>BASE(0)       '选择轴 0</div> <div>UNITS=1000</div> <div>ATYPE=1</div> <div>SPEED=100</div> <div>ACCEL =1000</div> <div>DECEL =1000</div> <div>SRAMP =100</div>

```
DPOS = 0
MERGE=ON
TRIGGER

MOVE(100)
MOVE_PARA(SPEED,0,200)
MOVE(200)
MOVE_PARA(SPEED,0,150)
MOVE(100)
END
```

垂直刻度均为 200



例三：多轴变速

```
RAPIDSTOP(2)
WAIT IDLE(0)
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
SRAMP=100,100           'S 曲线

MERGE=ON                 '开启连续插补
CORNER_MODE=2+8+32       '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
```

```
ZSMOOTH=2
FORCE_SPEED=100      '等比减速时起作用
TRIGGER              '自动触发示波器

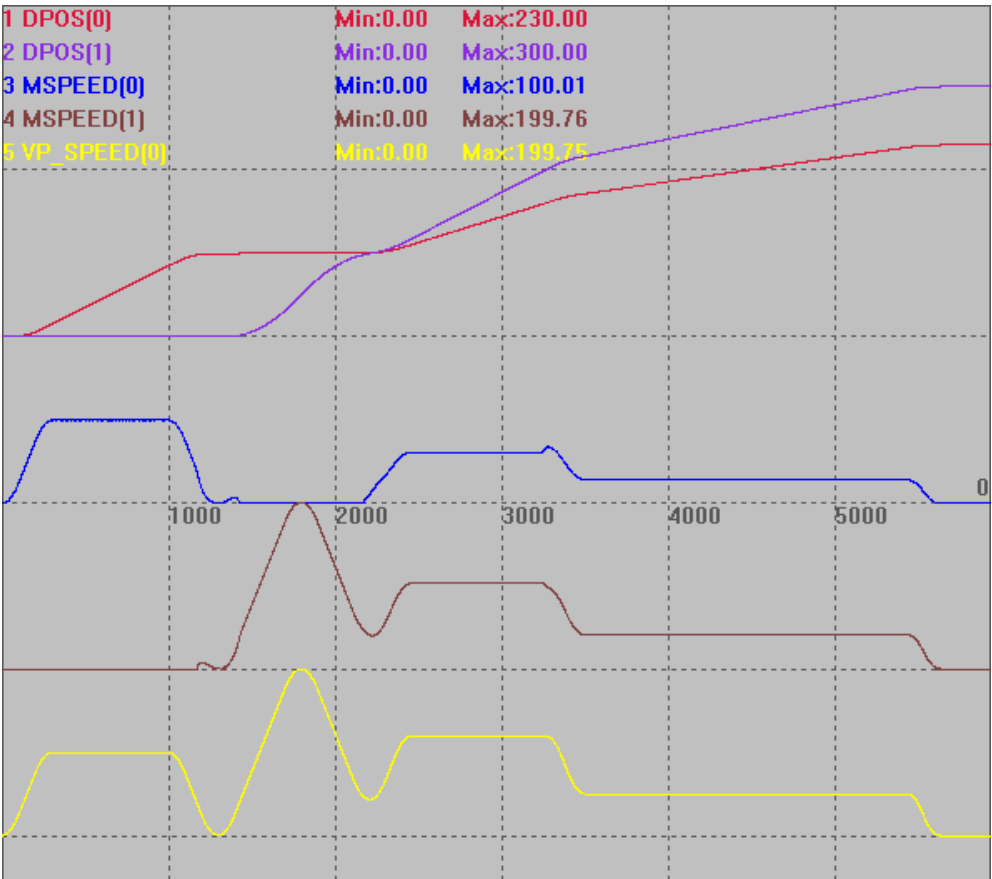
MOVE(100,0)

MOVE_PARA(SPEED,0,200)
MOVE(0,100)          '运动角度大于 45° ， 完全减速

MOVE_PARA(SPEED,0,120)
MOVE(60,100)         '运动角度 30.96° 处于 15° ~45° ， 等比减速

MOVE_PARA(SPEED,0,50)
MOVE(70,100)         '运动角度 4.03° 小于 15° ， 不减速
END
```

垂直刻度均为 200



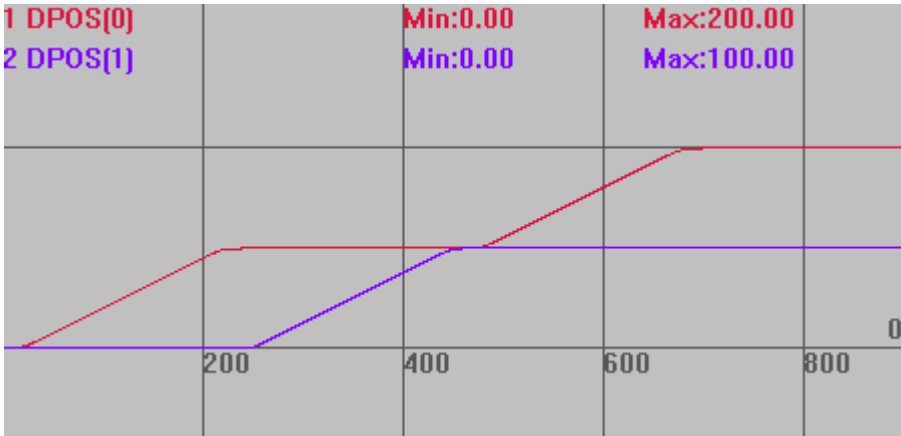


## MOVE\_PWM -- 缓冲 PWM

类型	特殊运动指令
描述	<p><b>BASE 轴运动缓冲操作 PWM。</b></p> <p>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。</p> <p>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。</p>
语法	<p>MOVE_PWM(pwmindex,duty[,freq])</p> <p>pwmindex: pwm 编号</p> <p>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</p> <p>freq: 频率，缺省为 1KHz，硬件最大为 1MHz，软件最大为 2KHz</p>
适用控制器	20170503 以上固件
例程	<pre> RAPIDSTOP(2) WAIT IDLE TRIGGER TICKS=0 BASE(0) SPEED = 1000 MOVE(10) <b>MOVE_PWM</b>(0, 0.111, 2000)      '轴 0 运行到 10 时，操作 PWM0 MOVE_DELAY(111) <b>MOVE_PWM</b>(0, 0.333) MOVE_DELAY(111) <b>MOVE_PWM</b>(0, 0.555, 3000) MOVE(100) WHILE NOT IDLE     <b>MOVE_PWM</b>(0, 0, 1000)      '关闭 PWM     ? -TICKS, PWM_FREQ(0), PWM_DUTY(0)     WA 10 WEND         </pre>
相关指令	<a href="#">PWM_DUTY</a> ， <a href="#">PWM_FREQ</a>

## MOVE\_SYNMOVE -- 缓冲触发其他轴

类型	特殊运动指令
描述	<p><b>BASE 轴运动缓冲触发其他轴运动，当前轴等待。</b></p> <p>此指令的 MTYPE 与 MOVE_DELAY 一致。</p>
语法	<p>MOVE_SYNMOVE(axisnum,dis[,ifsp])</p> <p>axisnum: 需要同步运动的轴号</p> <p>dis: 相对运动距离</p>

	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	<p> 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)  <b>MOVE_SYNMOVE</b>(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 </p> <p>运动轨迹</p> <p>DPOS(0)垂直刻度 200</p> <p>DPOS(1)垂直刻度 200</p> 
相关指令	<a href="#">MOVE_ASYNMOVE</a>

## MOVE ASYNMOVE -- 缓冲触发其他轴 2

类型	特殊运动指令
描述	<b>BASE</b> 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) <b>MOVE_ASYNCMOVE</b> (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	<a href="#">MOVE_SYNMOVE</a>

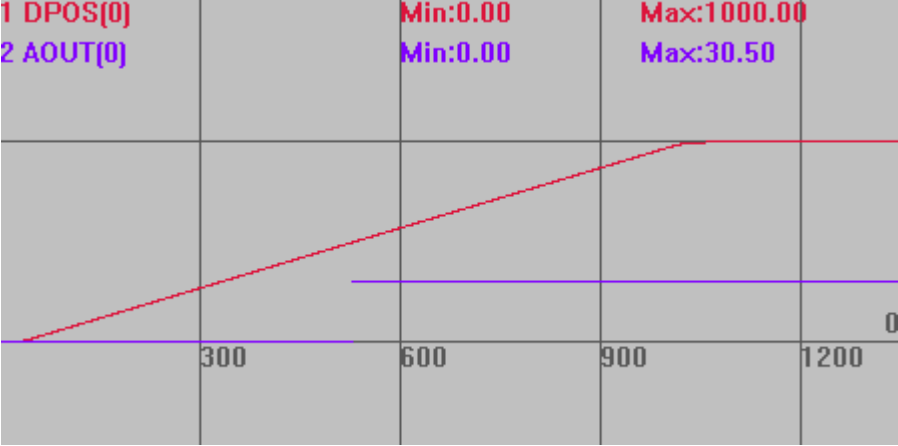
## MOVE TASK -- 缓冲开启任务

类型	特殊运动指令
描述	<p><b>BASE 轴运动缓冲加入启动 TASK。</b></p> <p>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。</p> <p>当任务已经启动时，会报错，但不影响程序执行。</p>
语法	<p>MOVE_TASK(tasknum, label)</p> <p>tasknum: 任务号</p> <p>label: 函数名或标号</p>
适用控制器	通用
例子	BASE(0)

	<pre> DPOS=0 UNITS=100 ACCEL=1000 DECEL=1000 SPEED=100 ACCEL=1000 DECEL=1000 MOVE(100) <b>MOVE_TASK</b>(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 </pre>
相关指令	<a href="#">RUNTASK</a> , <a href="#">RUN</a>

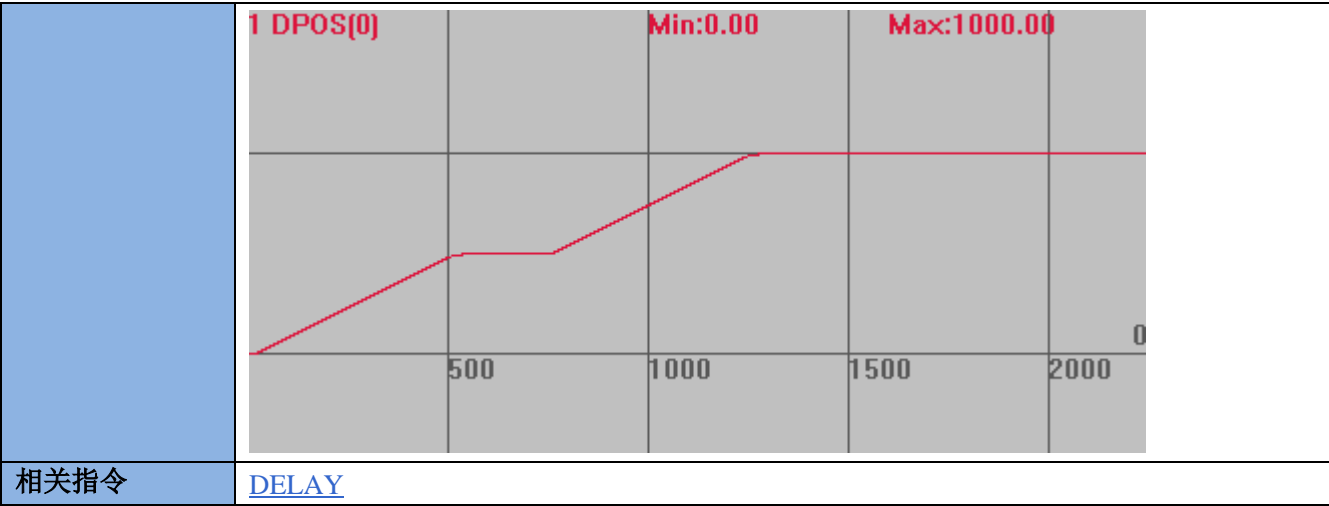
## MOVE\_AOUT -- 缓冲模拟量输出

类型	特殊运动指令
描述	<b>BASE</b> 轴运动缓冲加入一个 <b>AOUT</b> 指令。 这个指令 <b>LOAD</b> 执行时不做任何运动, 只修改 <b>AOUT</b> 值。此指令的 <b>MTYPE</b> 与 <b>MOVE_OP</b> 一致。
语法	<pre>MOVE_AOUT(danum, value)</pre> <p>danum: DA 编号 value: 要修改的值</p>
适用控制器	通用, 实际只有包含 DA 通道的控制器生效
例子	<pre> 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) <b>MOVE_AOUT</b>(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 </pre>

	<div>运动轨迹</div> <div>DPOS(0)垂直刻度 1000</div> <div>AOUT(0)垂直刻度 100</div> <div><div><div>1 DPOS[0]</div><div>2 AOUT[0]</div></div><div><div>Min:0.00</div><div>Min:0.00</div></div><div><div>Max:1000.00</div><div>Max:30.50</div></div><div></div></div>
相关指令	<a href="#">AOUT</a>

## MOVE\_DELAY -- 缓冲延时

类型	特殊运动指令
描述	<b>BASE 轴运动缓冲加入一个延时。</b> 这个指令 LOAD 执行时不做任何运动，只延时指定时间。 <b>前面的运动指令结束时速度会自动将为 0。</b>
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	<div>BASE(0)</div> <div>UNITS=100</div> <div>DPOS=0</div> <div>SPEED=200</div> <div>ACCEL=2000</div> <div>DECEL=2000</div> <div>TRIGGER                   '自动触发示波器</div> <div>MOVE(500)</div> <div><b>MOVE_DELAY</b>(1000)       '两个 MOVE 中间等待 1 秒</div> <div>MOVE(500)</div> <div>运动轨迹</div> <div>DPOS(0)垂直刻度 1000</div>



MOVE\_WAIT -- 缓冲等待

类型	特殊运动指令
描述	<b>BASE 轴运动缓冲加入一个条件判断。</b> 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	<b>MOVE_WAIT</b> (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	<pre>BASE(0) UNITS=100 DPOS=0 SPEED=200 ACCEL=2000 DECEL=2000 TRIGGER          '自动触发示波器 MOVE(500) <b>MOVE_WAIT</b>(IN, 0, 0, 1) '等待 IN(0)有信号，才执行下一条运动缓冲 MOVE(500)</pre> <p>运动轨迹 DPOS(0)垂直刻度 1000</p>

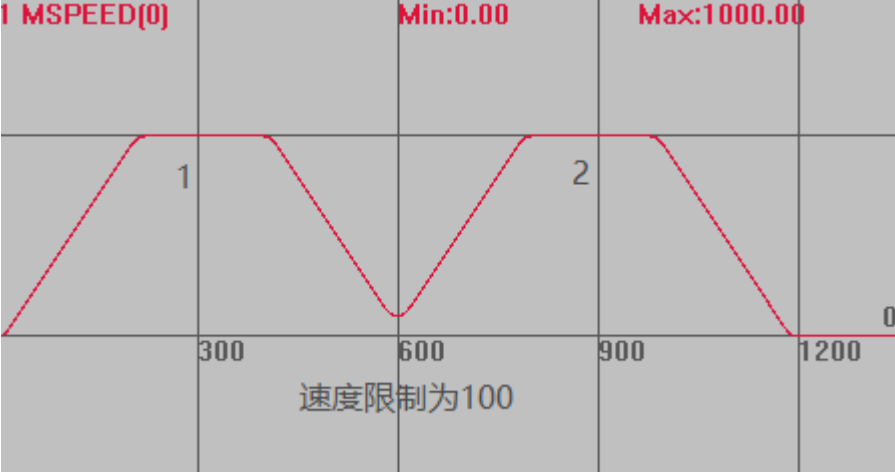
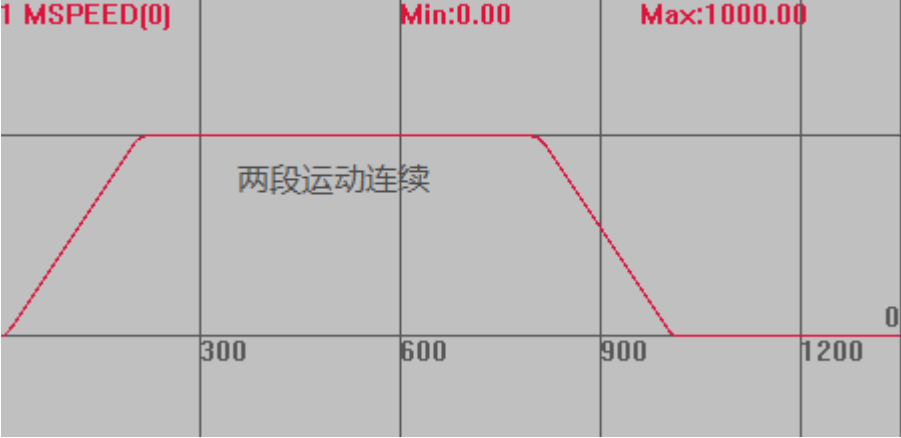
	<p>IN(0)垂直刻度 1</p>
相关指令	<a href="#">WAIT UNTIL</a>

## 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	<a href="#">CANCEL</a>

## 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

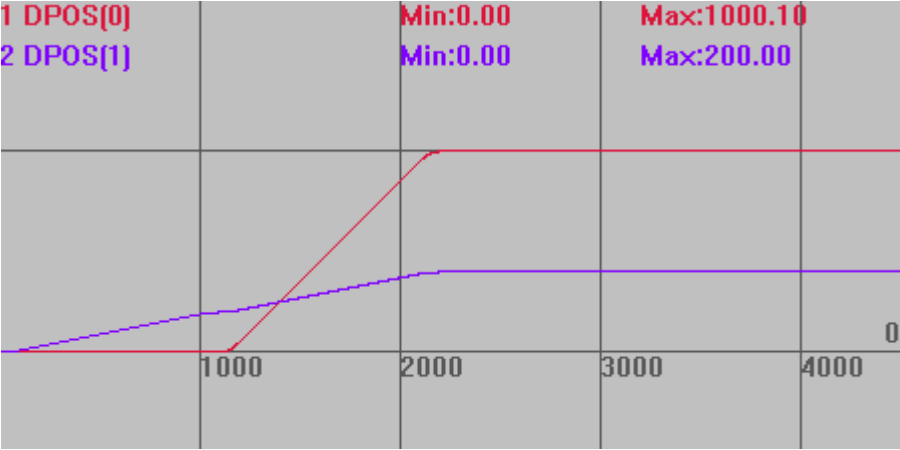
	<div><div>MERGE=ON'开启连续插补，多段运动间速度连续</div><div>TRIGGER'自动触发示波器</div><div>MOVE(2000)</div><div>MOVELIMIT(100)'在两个运动的中间降速到 100</div><div>MOVE(2000)</div><div>用 MOVELIMIT 限制速度时，插补速度</div><div>MSPEED(0)垂直刻度 1000</div><div></div><div>不限制速度时</div><div></div></div>
相关指令	<a href="#">CORNER_MODE</a>

7.4. 同步运动指令

CONNECT -- 同步运动

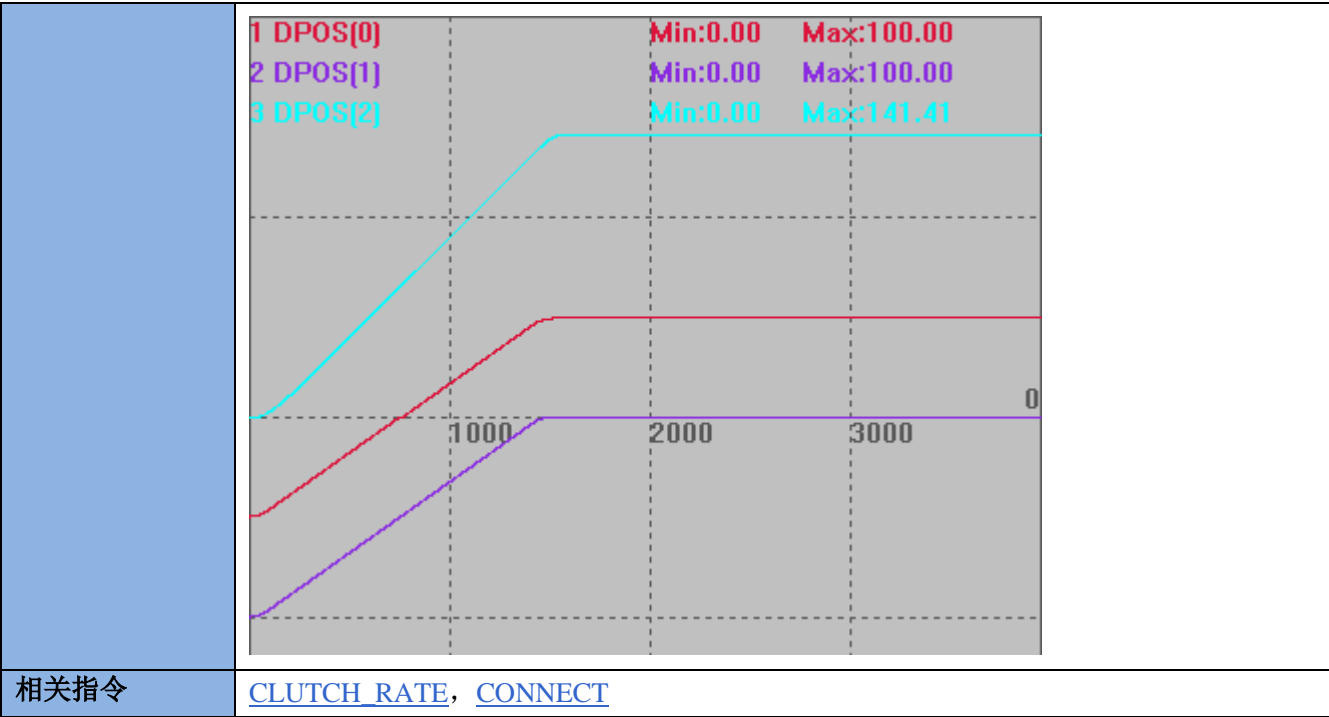
类型	同步运动指令
描述	将当前轴的目标位置与 driving_axis 轴的测量位置通过电子齿轮连接。 连接的是脉冲个数，要考虑不同轴 UNITS 的比例。取消连接时用 CANCEL。



	<p>假设连接轴 0 的 UNIST 为 100，被连接轴 1 的 UNITS 为 10。</p> <p>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。</p> <p>比率可以通过重复调用指令动态变化。</p> <p>常用于手轮使用。</p>
语法	<p>CONNECT (ratio, driving_axis)</p> <p>ratio: 比率，可正可负，注意是脉冲个数的比例</p> <p>driving_axis: 连接轴的轴号，手轮时为编码器轴</p>
适用控制器	通用
例子	<p>RAPIDSTOP(2)</p> <p>WAIT IDLE(0)</p> <p>WAIT IDLE(1)</p> <p>BASE(0,1)</p> <p>ATYPE=1,1</p> <p>UNITS=10,100</p> <p>DPOS=0,0</p> <p>SPEED=100,100</p> <p>ACCEL=1000,1000</p> <p>DECEL=1000,1000</p> <p>TRIGGER '自动触发示波器</p> <p>MOVE(100) AXIS(1) '轴 1 运动 100，此时轴 0 不动</p> <p>WAIT IDLE(1)</p> <p>CONNECT(1,1) AXIS(0) '轴 0 连接到轴 1，比例为 1</p> <p>MOVE(100) AXIS(1) '轴 1 运动 100，轴 0 运动 1000</p> <p>运动轨迹</p> <p>DPOS(0)垂直刻度 1000</p> <p>DPOS(1)垂直刻度 500</p> 
相关指令	<a href="#">CLUTCH_RATE</a> , <a href="#">CONNPATH</a>

## CONNPATH -- 同步运动 2

类型	同步运动指令
描述	<p>将当前轴的目标位置与 <b>driving_axis</b> 轴的插补矢量长度通过电子齿轮连接。 需要连接到插补运动的主轴才能与插补矢量长度建立连接，连接到从轴的效果与 CONNECT 相同。</p> <p>连接的是脉冲个数，要考虑不同轴 <b>UNITS</b> 的比例。取消连接时用 <b>CANCEL</b>。 比率可以通过重复调用指令动态变化。</p>
语法	<p>CONNPATH (ratio, driving_axis)</p> <p>Ratio: 比率，可正可负，注意是脉冲个数的比例</p> <p>driving_axis: 连接轴的轴号，手轮时为编码器轴</p>
适用控制器	通用
例子	<pre> RAPIDSTOP(2) WAIT IDLE(0) WAIT IDLE(1) BASE(0,1,2) DPOS=0,0,0 ATYPE=1,1,1 UNITS=100,100,100 SPEED=100,100,100 ACCEL=1000,1000,1000 DECEL=1000,1000,1000 TRIGGER '自动触发示波器 CONNPATH(1,0) AXIS(2) '轴 2 连接到插补运动的主轴轴 0，比例为 1 MOVE(100,100) '插补运动  运动轨迹 DPOS(0)垂直刻度 100，偏移-50 DPOS(1)垂直刻度 100，偏移-50 DPOS(2)垂直刻度 100，无偏移 </pre>



[CLUTCH\\_RATE](#), [CONNECT](#)

CAM -- 凸轮表运动

类型	同步运动指令
描述	<p>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。</p> <p>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。</p> <p>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。</p> <p>TBALE 数据需要手动设置，第一个数据为引导点，建议设为 0。</p> <p>TABLE 数据*table multiplier 这个比例=发出的脉冲数。</p> <p>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。</p>
语法	<p>CAM(start point, end point, table multiplier, distance)</p> <p>start point: 起始点 TABLE 编号，存储第一个点的位置</p> <p>end point: 结束点 TABLE 编号</p> <p>table multiplier: 位置乘以这个比例，一般设为脉冲当量值</p> <p>distance: 参考运动的距离</p> <p>总时间=distance/轴 speed</p>
适用控制器	通用
例子	<p>例一：</p> <p>RAPIDSTOP(2)</p> <p>WAIT IDLE(0)</p>

```

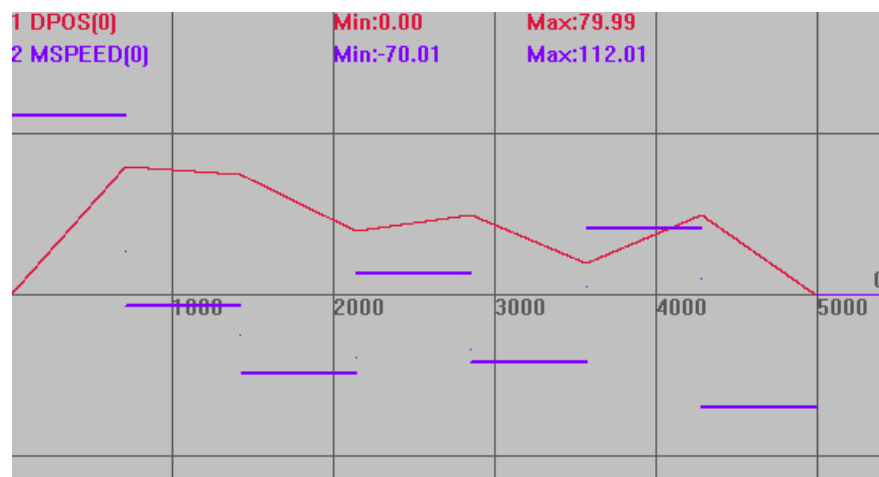
WAIT IDLE(1)
BASE(0)
UNITS=100
DPOS=0
ACCEL=1000
DECEL=1000
SPEED=100
TABLE(10,0,80,75,40,50,20,50,0) 'table 从 10 开始存数据
TRIGGER '自动触发示波器
CAM(10,17,100,500) '运动轨迹为 table10 到 17，运动总时间为 500/100=5s

```

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

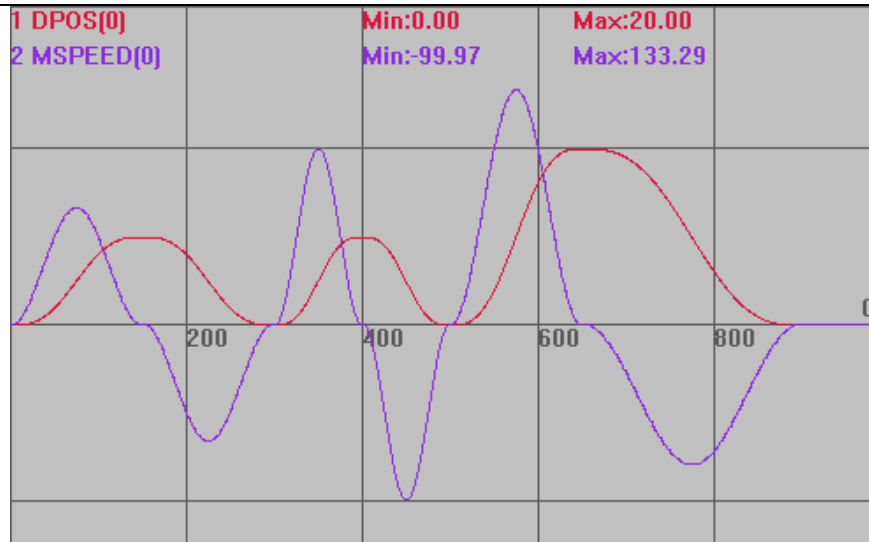
CAM(0,100,-m,SPEED\*t)

WAIT IDLE

插补轨迹

DPOS(0)垂直刻度 20

MSPEED(0)垂直刻度 100



例三：连续凸轮

RAPIDSTOP(2)

WAIT IDLE(0)

BASE(0)

UNITS=100

DPOS=0

ACCEL=1000

DECEL=1000

SPEED=100

DIM rad,x,deg

FOR deg= 0 TO 360 STEP 1 '构造 0-360 度的凸轮数据

rad = deg \* 2 \* PI / 360

x = deg \* 2 + 10000 \* ( 1 - COS (rad))

TABLE (deg ,x)

print deg,x

NEXT deg

TRIGGER '自动触发示波器

CAM(0,360,100,100) '运动轨迹为 table 0 到 360，运动总时间为 100/100=1s

CAM(0,360,100,200) '运动轨迹为 table 0 到 360，运动总时间为 200/100=2s

CAM(0,360,100,300) '运动轨迹为 table 0 到 360，运动总时间为 300/100=3s

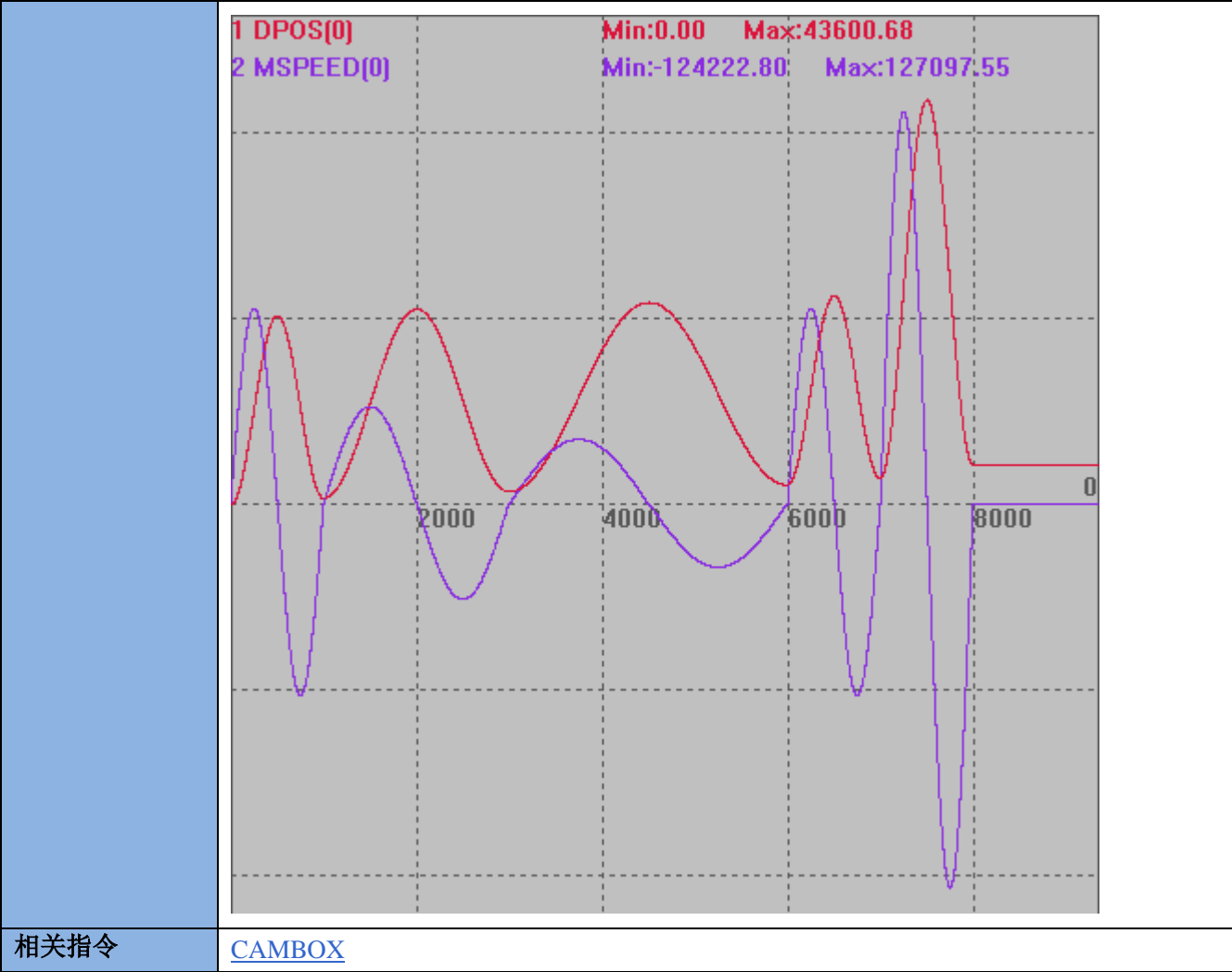
WAIT UNTIL REMAIN\_BUFFER(1) > 0 '等待缓冲空间空了再存运动指令

CAM(0,360,100,100) '运动轨迹为 table 0 到 360，运动距离为 100\*table 数据/units(0)

CAM(0,360,200,100) '运动轨迹为 table 0 到 360，运动总时间 200\*table 数据/units(0)

DPOS(0)垂直刻度 20000

MSPEED(0)垂直刻度 60000

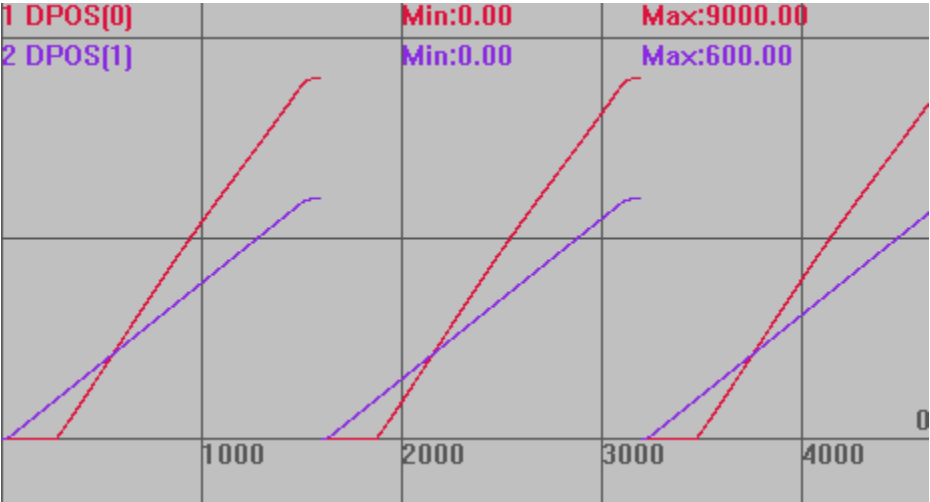
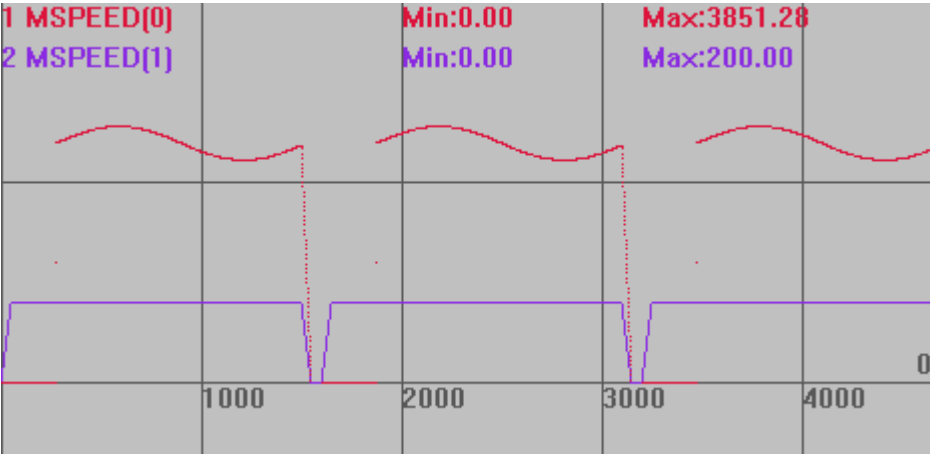


CAMBOX -- 跟随凸轮表运动

类型	同步运动指令
描述	<p>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。</p> <p>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。</p> <p>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。</p> <p>Table 数据需要手动设置，第一个数据为引导点，建议设为 0。</p> <p>Table 数据*table multiplier 这个比例=发出的脉冲数。</p> <p>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。</p>
语法	<p>CAMBOX(start_point, end_point, table_multiplier, link_distance , link_axis[, link_options][, link_pos][, link_offpos])</p> <p>start point: 起始点 TABLE 编号，存储第一个点的位置</p> <p>end point: 结束点 TABLE 编号</p>

	<p>table multiplier: 位置乘以这个比例, 一般设为脉冲当量值</p> <p>link_distance: 参考轴运动的距离</p> <p>link_axis: 参考轴轴号</p> <p>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</p> <table border="1"> <thead> <tr> <th>位</th><th>意义</th></tr> </thead> <tbody> <tr> <td>bit0</td><td>当参考轴 <b>MARK</b> 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 <b>REP_OPTION=1</b>, 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 <b>MARKB</b> 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> </tbody> </table> <p>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</p> <p>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</p>	位	意义	bit0	当参考轴 <b>MARK</b> 信号事件触发时, 当前轴与参考轴开始进行连接运动	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 <b>REP_OPTION=1</b> , 可以取消重复)	bit4	从中间某个位置启动, 配合掉电中断实现恢复凸轮	bit5	只有参考轴的正向运动才连接	bit8	当参考轴 <b>MARKB</b> 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
位	意义														
bit0	当参考轴 <b>MARK</b> 信号事件触发时, 当前轴与参考轴开始进行连接运动														
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 <b>REP_OPTION=1</b> , 可以取消重复)														
bit4	从中间某个位置启动, 配合掉电中断实现恢复凸轮														
bit5	只有参考轴的正向运动才连接														
bit8	当参考轴 <b>MARKB</b> 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														
适用控制器	通用														
例子	<pre> 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 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)     </pre>														



	<div>WAIT UNTIL IDLE AND IDLE(1) '等待运动停止</div> <div>DELAY(100) '延时</div> <div>ENDIF</div> <div>WEND</div> <div>END '停止当前任务</div> <div>运动轨迹</div> <div>DPOS(0)垂直刻度 5000</div> <div>DPOS(1)垂直刻度 500</div> <div></div> <div>速度曲线</div> <div>MSPEED(0)垂直刻度 3000</div> <div>MSPEED(1)垂直刻度 500</div> <div></div>
相关指令	<a href="#">CAM</a>

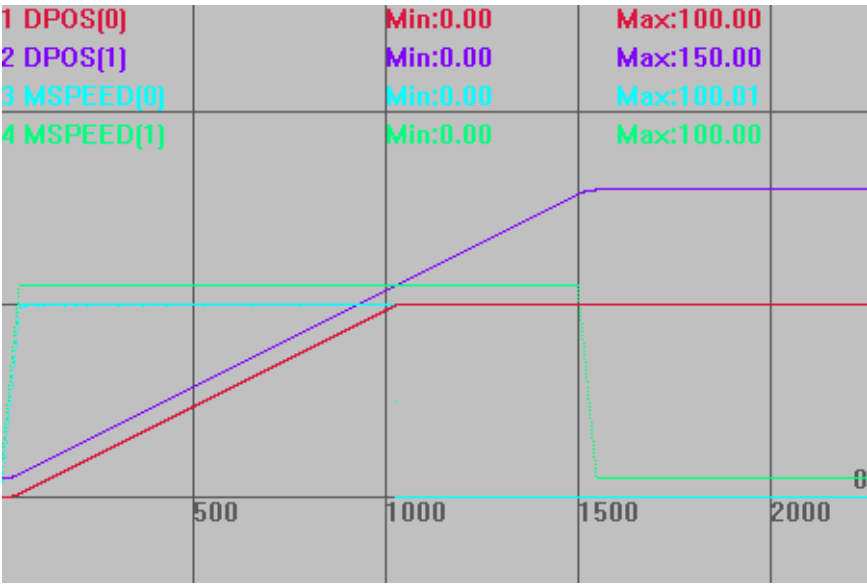
MOVELINK -- 自动凸轮

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。

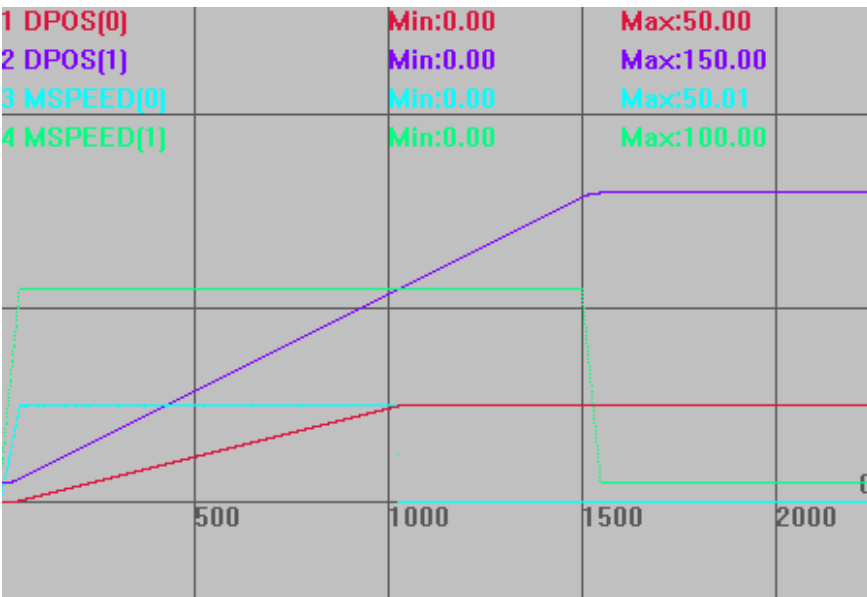
	<p>被连接轴为参考轴，连接轴为跟随轴。</p> <p>连接轴的距离分成 3 个阶段应用于参考轴的运动，分别是加速部分、匀速部分和减速部分。</p> <p>在加速和减速阶段为了与速度匹配，link distance（基本轴运动距离）必须是 distance（跟随轴运动距离）的两倍。</p> <p>请确保指令传递的距离参数*units 是整数个脉冲，否则出现浮点数会导致运动有细微误差。</p>																								
语法	<p>MOVELINK (distance, link dist, link acc, link dec, link axis[,link options] [,link pos][,link offpos])</p> <p>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</p> <p>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</p> <p>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</p> <p>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</p> <p>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</p> <p>link axis: 参考轴的轴号</p> <p>link options: 连接模式选项，不同的二进制位代表不同的意义</p> <table><tr><th>模式</th><th>位</th><th>描述</th></tr><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>8</td><td>位 3</td><td>当设置时,采用 S 曲线加减速。20170502 以上固件支持</td></tr><tr><td>16</td><td>位 4</td><td>从中间某个位置启动，配合掉电中断实现恢复跟随</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> <p>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</p> <p>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</p>	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
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																							
适用控制器	通用																								
例子	<p>例一：</p> <p>RAPIDSTOP(2)</p> <p>WAIT IDLE(0)</p> <p>WAIT IDLE(1)</p> <p>BASE(0,1) '轴 0 为跟随轴，轴 1 为参考轴</p> <p>UNITS=100,100</p> <p>ATYPE=1,1</p> <p>DPOS=0,0</p> <p>SPEED=100,100</p>																								

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

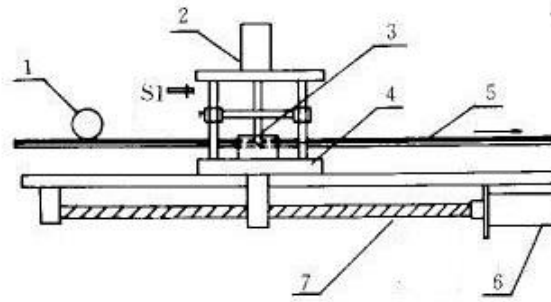


图1 飞剪系统原理图

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
```

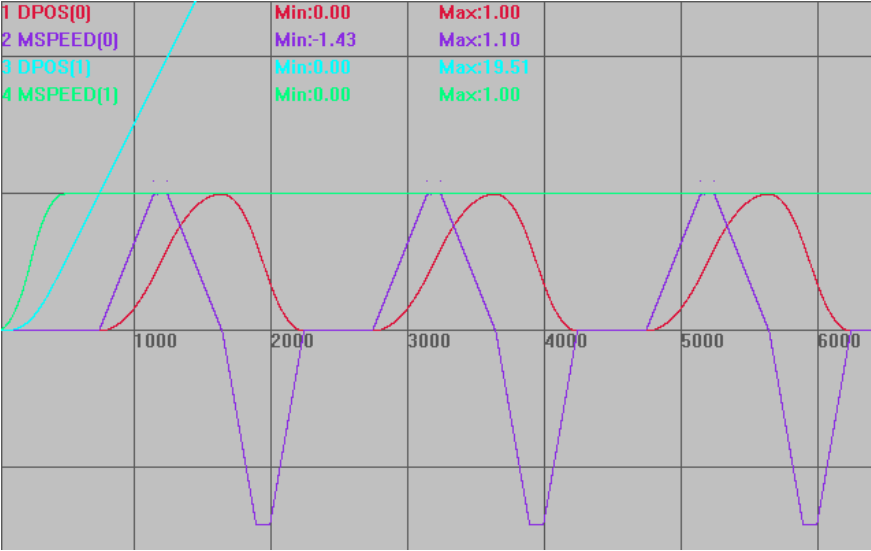
运动轨迹和速度曲线:

DPOS(0)垂直刻度 1, 无偏移

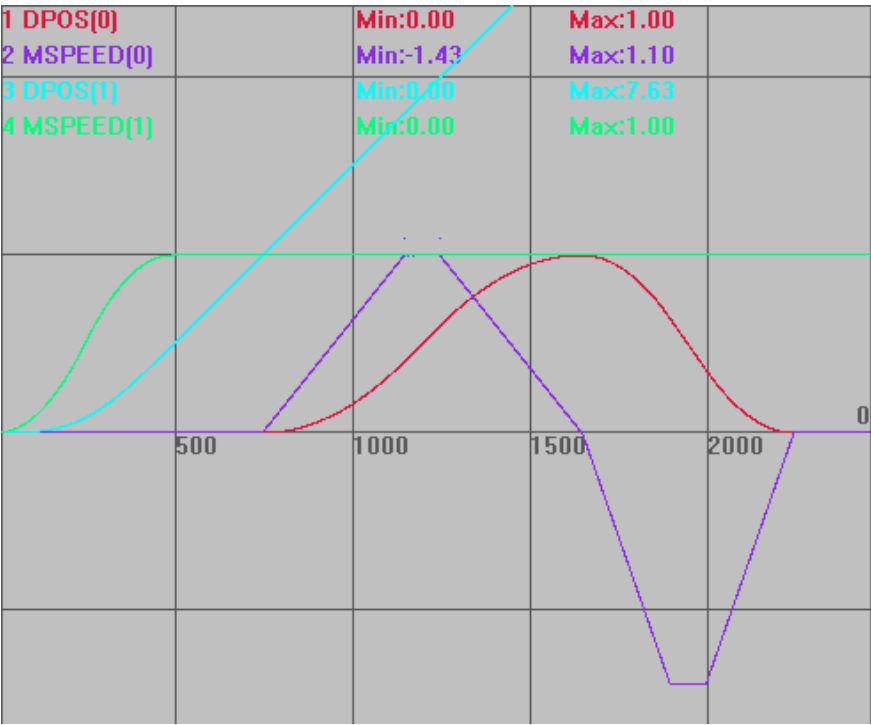
MSPEED(0)垂直刻度 1, 无偏移

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:  $0.4(\text{加速阶段}) + 0.2(\text{跟随同步}) + 0.4(\text{减速阶段}) = 1\text{m}$  单位, 然后-1m 返回运动。

型材(参考轴)的运动距离:  $1 + 0.8 + 0.2 + 0.8 + 1.2 = 4\text{m}$  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)

```

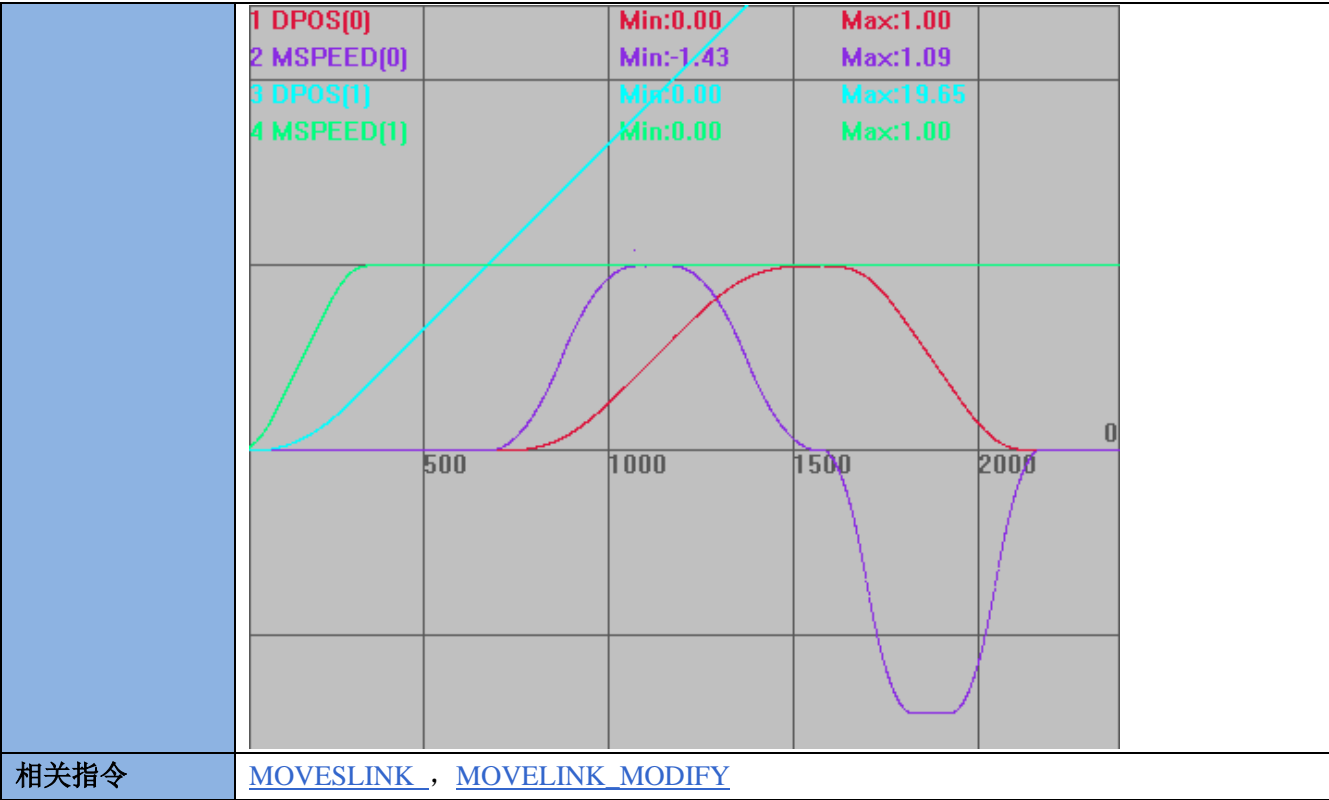
WAIT IDLE(0)
WAIT IDLE(1)
DATUM(0)
BASE(0,1)
UNITS=10000,10000
ATYPE=1,1
DPOS=0,0
SPEED=1,1      "型材运行速度 1m/s,60m/min
ACCEL=2,2
DECEL=2,2
SRAMP=200,200

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

```



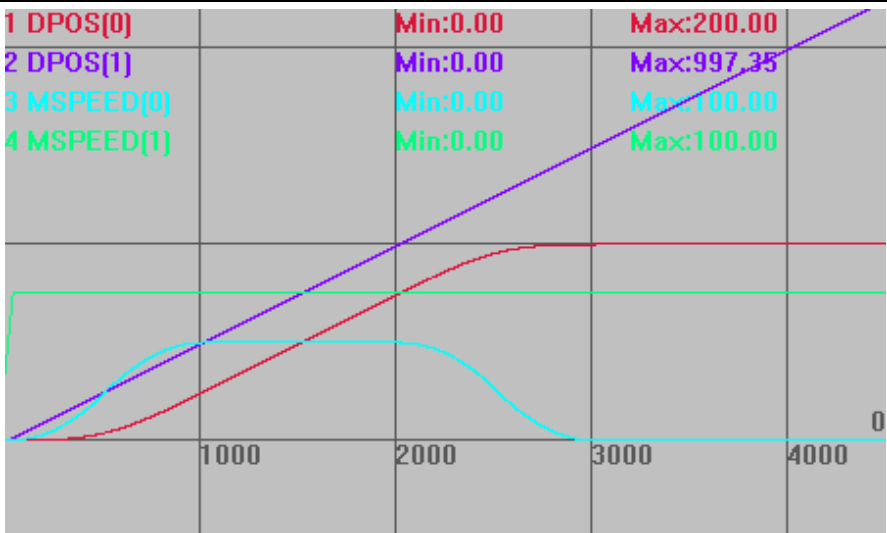
相关指令 [MOVESLINK](#) , [MOVELINK\\_MODIFY](#)

MOVESLINK -- 自动凸轮 2

类型	同步运动指令					
描述	<p>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。</p> <p>被连接轴为参考轴，连接轴为跟随轴。</p> <p>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。</p> <p>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。</p>					
语法	<p>MOVESLINK (distance,link dist,start sp,end sp,link axis[,link options][,link pos][, link offpos])</p> <p>[, 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。</p> <p>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</p> <p>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</p> <p>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</p> <p>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</p> <p>注：当 start sp = end sp = distance/link dist 时，匀速运动</p> <p>link axis: 参考轴的轴号</p> <p>link options: 连接模式选项，不同的二进制位代表不同的意义</p> <table><tr><td>模式</td><td>位</td><td>描述</td></tr></table>			模式	位	描述
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)
	16	位 4	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
	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。			
适用控制器	通用		
例子	<p>功能与 MOVELINK 相同, 仅是参数设置区别</p> <p>例一:</p> <p>RAPIDSTOP(2)</p> <p>WAIT IDLE(0)</p> <p>WAIT IDLE(1)</p> <p>DATUM(0)</p> <p>BASE(0,1)</p> <p>UNITS=100,100</p> <p>ATYPE=1,1</p> <p>DPOS=0,0</p> <p>SPEED=100,100</p> <p>ACCEL=2000,2000</p> <p>DECEL=2000,2000</p> <p>TRIGGER '自动触发示波器</p> <p>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</p> <p>MOVESLINK(100,100,1,1,1) AXIS(0) '轴 0 跟踪轴 1 运动, 匀速 100units</p> <p>MOVESLINK(50,100,1,0,1) AXIS(0) '轴 0 跟踪轴 1 运动, 减速到 0</p> <p>VMOVE(1) AXIS(1)</p> <p>运动轨迹和速度曲线</p> <p>DPOS(0)垂直刻度 200, 无偏移</p> <p>DPOS(1)垂直刻度 200, 无偏移</p> <p>MSPEED(0)垂直刻度 200, 无偏移</p> <p>MSPEED(1)垂直刻度 200, 偏移 50</p>		





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

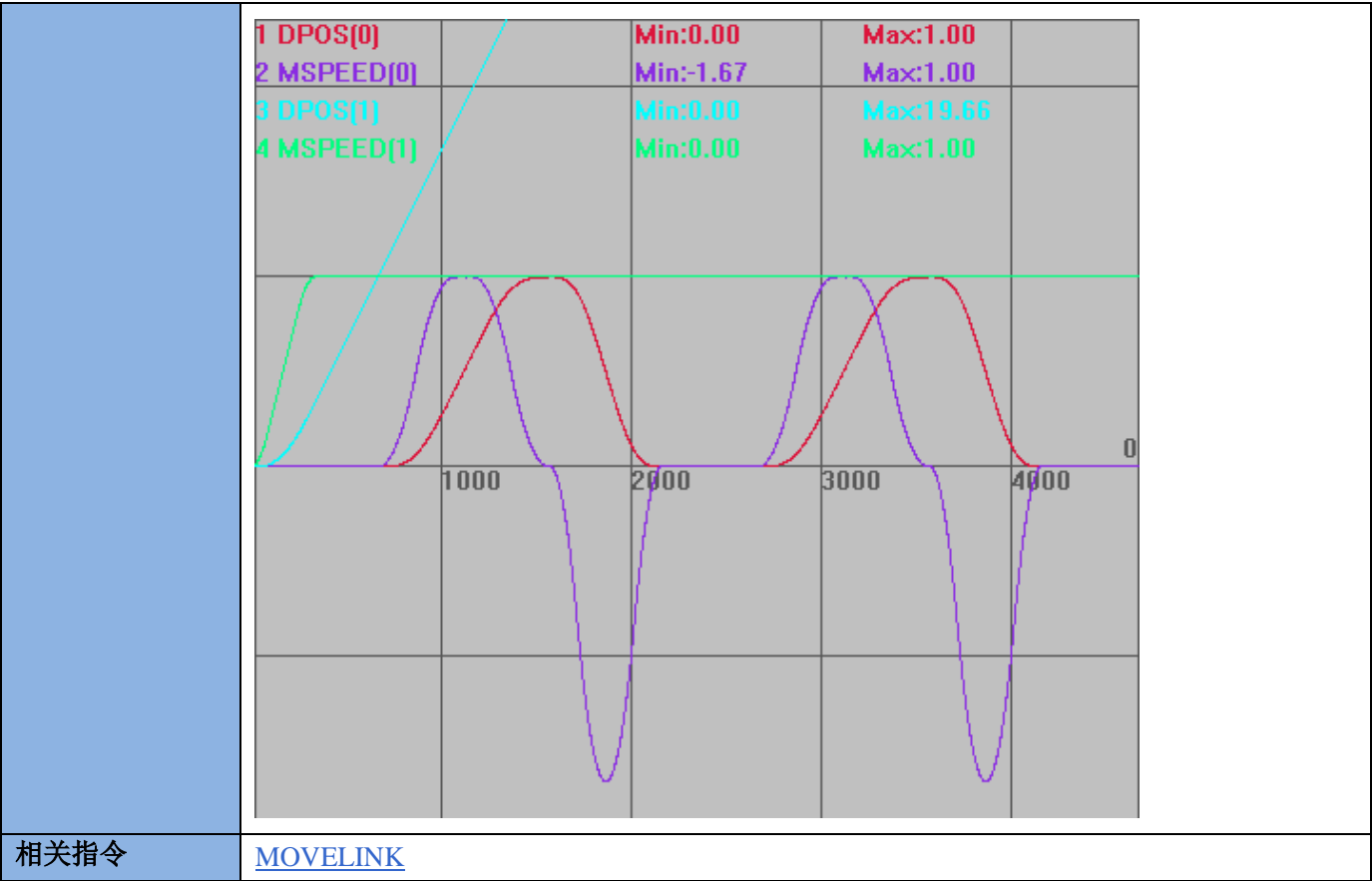
运动轨迹和速度曲线

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

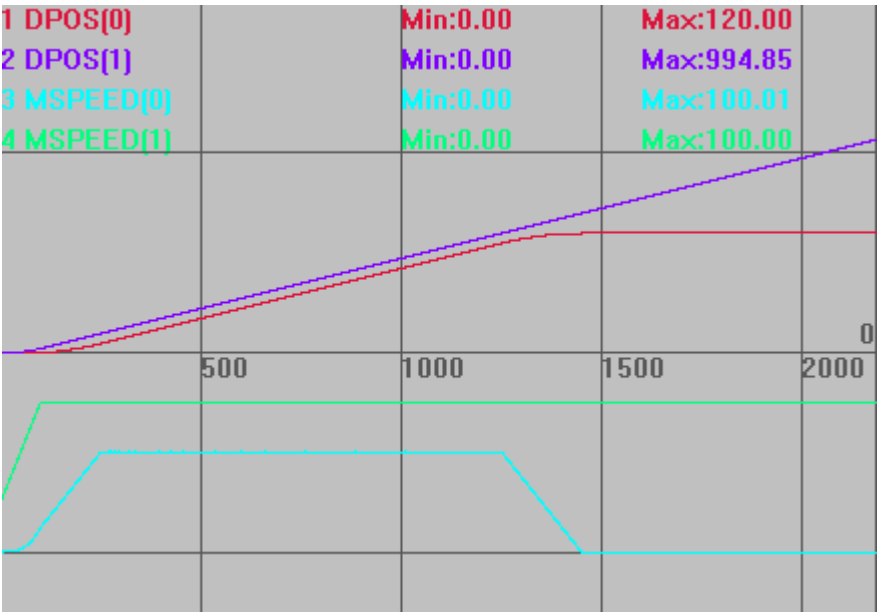


MOVELINK\_MODIFY -- 同步距离修改

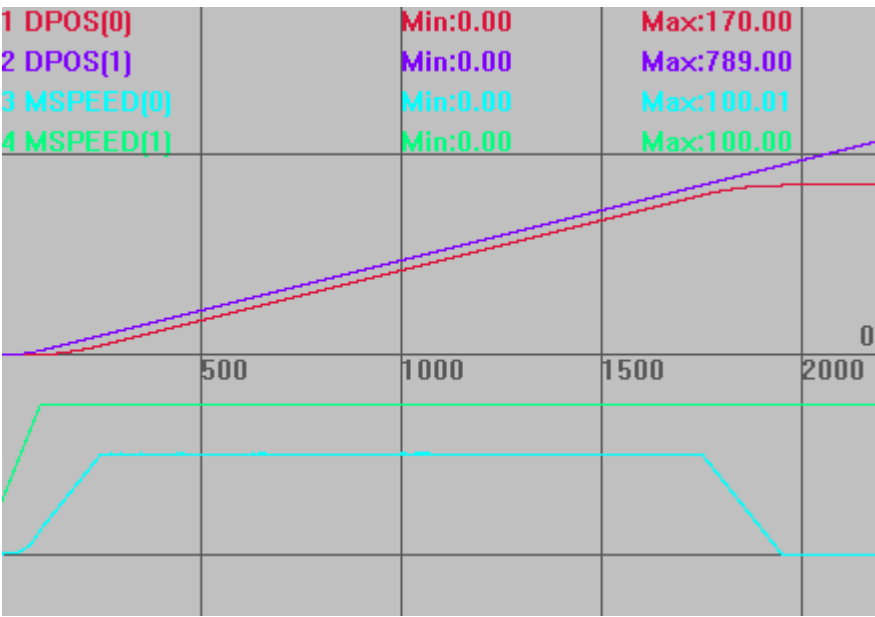
类型	轴参数
描述	相对修改 MOVELINK 指令的同步区长度。 带运动缓冲，只在同步段后设置生效。
语法	VAR1 = MOVELINK_MODIFY, MOVELINK_MODIFY = expression
适用控制器	20160926 以后固件版本支持。
例子	例一： RAPIDSTOP(2) WAIT IDLE(0) WAIT IDLE(1) BASE(0,1) UNITS=100,100 ATYPE=1,1 DPOS=0,0 SPEED=100,100 ACCEL=1000,1000 DECEL=1000,1000 TRIGGER 未修改同步距离 <div>'自动触发示波器'</div>

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

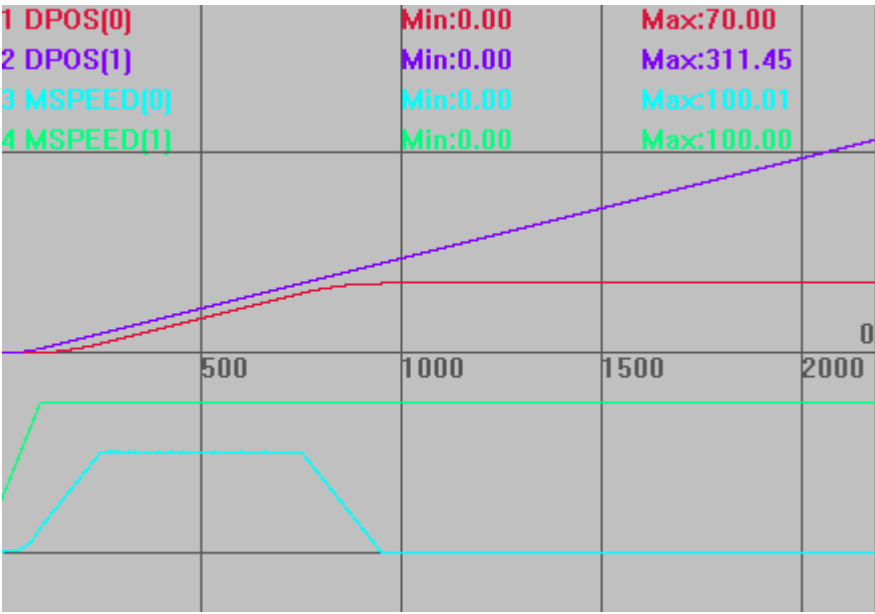


其他条件同上，增加同步距离  
MOVELINK(10,20,20,0,1) '工作台加速阶段  
MOVELINK(100,100,0,0,1) '同步阶段 100  
**MOVELINK\_MODIFY = 50** '修改同步段为 100+50  
MOVELINK(10,20,0,20,1) '减速阶段



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)

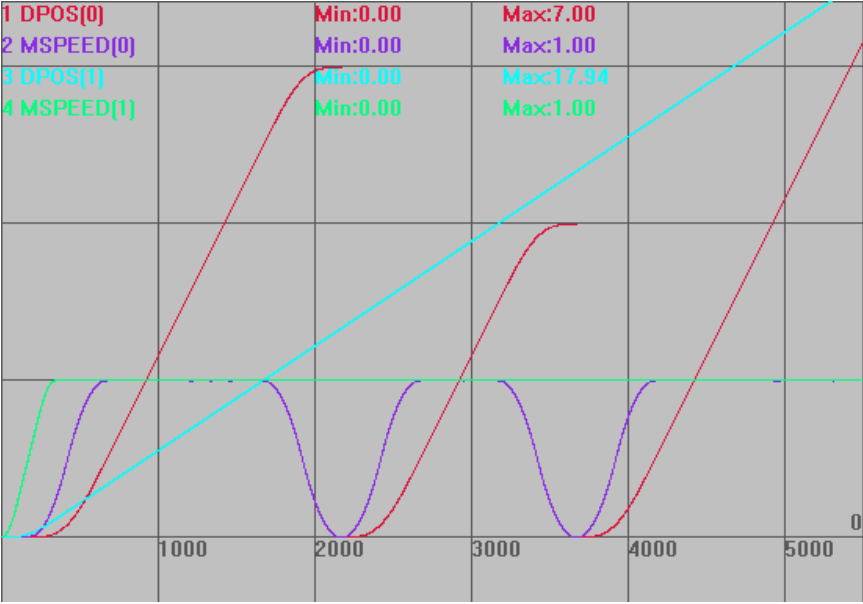
BASE(0)

DPOS=0

'MOVELINK\_MODIFY=0 '先清空

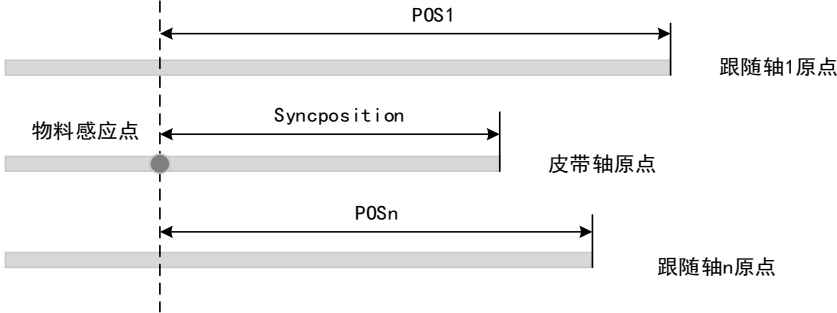
MOVELINK(3,4,1,1,1,8) AXIS(0)

WAIT UNTIL MPOS(0)>1 '等待跟随轴距离>2

	<div>MOVELINK_MODIFY=1           '把跟随轴距离增加 1</div> <div>WAIT UNTIL MOVELINK_MODIFY=0   '等待同步偏移完成</div> <div>WEND</div> <div>运动轨迹和速度曲线</div> <div>DPOS(0)垂直刻度 1，无偏移</div> <div>MSPEED(0)垂直刻度 1，无偏移</div> <div>DPOS(1)垂直刻度 3，无偏移</div> <div>MSPEED(1)垂直刻度 1，无偏移</div> <div></div>
相关指令	<a href="#">MOVELINK MOVELINK -- 自动凸轮</a>

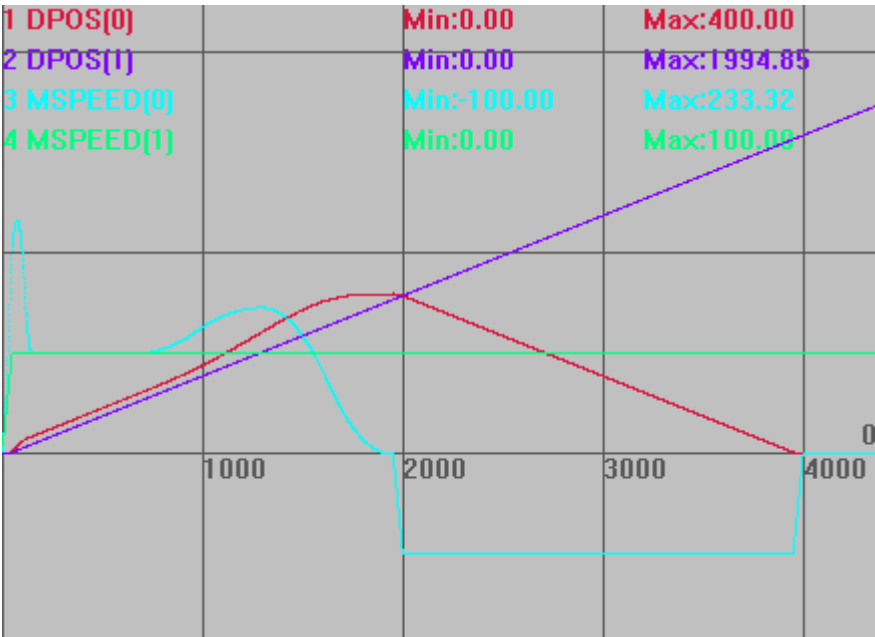
MOVESYNC -- 同步运动

类型	运动设置指令				
描述	<div>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。</div> <div>要求皮带轴与 BASE 跟随轴的长度单位是一样的。</div> <div>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。</div> <div>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。</div> <div>此指令属于凸轮指令，不支持运动暂停。</div>				
语法	<div>MOVESYNC(mode,synctime,syncposition,syncaxis,pos1[,pos2, pos3...])</div> <div>mode: 模式</div> <table><tr><th>模式</th><th>描述</th></tr><tr><td>-1</td><td>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</td></tr></table>	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
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				

	<table><tr><td>-2</td><td>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</td></tr><tr><td>0</td><td>BASE 第 1 个轴（x）跟随皮带轴物体</td></tr><tr><td>10</td><td>BASE 第 2 个轴（y）跟随皮带轴物体</td></tr><tr><td>20</td><td>BASE 第 3 个轴跟随皮带轴物体</td></tr></table> <p>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</p> <p>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</p> <p>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</p> <p>syncaxis：皮带轴轴号，-1 表示没有皮带轴，直接运动到 pos1 的位置</p> <p>pos1：皮带轴物体被感应到时的 BASE 第 1 个轴绝对位置</p> <p>posn：皮带轴物体被感应到时的 BASE 第 n 个轴绝对位置</p> 	-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
-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								
适用控制器	4 系列 170601 以上固件支持。								
例子	<p>例一：皮带取料</p> <p>RAPIDSTOP(2)</p> <p>WAIT IDLE(0)</p> <p>WAIT IDLE(1)</p> <p>BASE(0,1)</p> <p>DPOS=0,0</p> <p>UNITS=100,100</p> <p>ATYPE=1,1</p> <p>SPEED=100,100</p> <p>ACCEL=1000,1000</p> <p>DECEL=1000,1000</p> <p>TRIGGER</p> <p><b>MOVESYNC(0, 0, 100, 1, 120)</b> '同步到皮带物体上</p> <p>MOVE_OP(1,1) '下降, 如果 Z 轴下降,可以用 MOVESYNC 运动指令代替</p> <p>MOVE_OP(0,1) '开吸嘴</p> <p><b>MOVESYNC(0, 1000, 100, 1, 120 )</b> '继续同步 1s</p> <p>MOVE_OP(1,0) '上升</p>								

**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

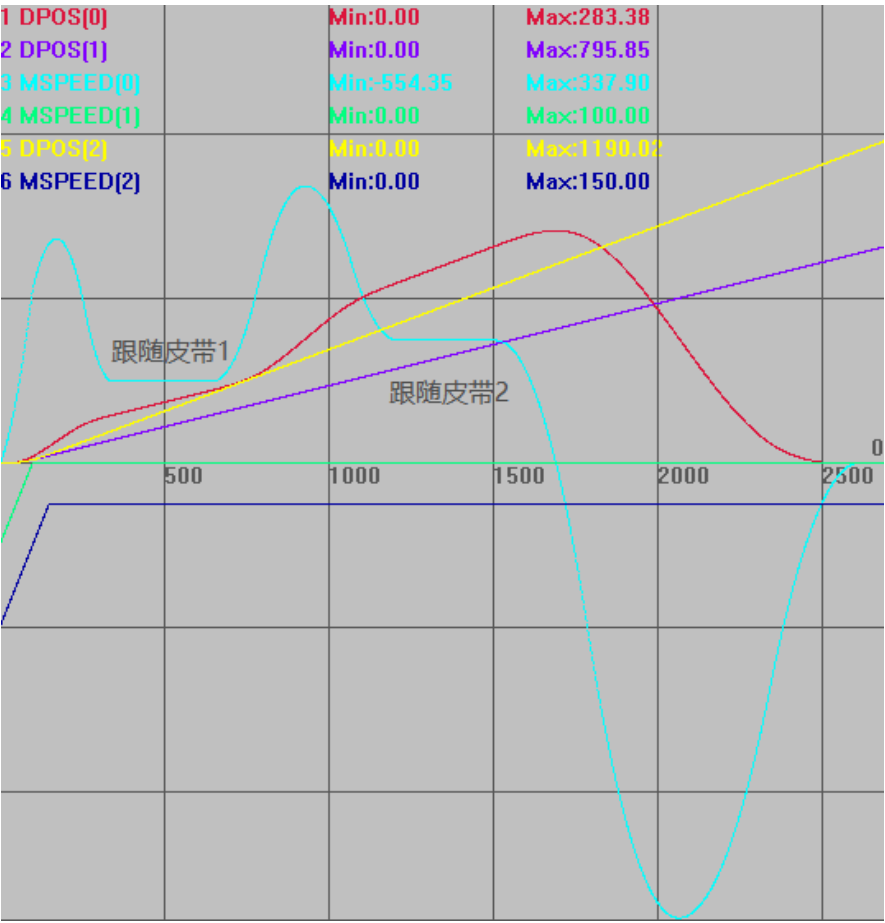


例二：皮带取料，放另外的皮带上  
**RAPIDSTOP(2)**  
**WAIT IDLE(0)**  
**WAIT IDLE(1)**  
**BASE(0,1,2)**  
**DPOS=0,0,0**  
**UNITS=100,100,100**  
**ATYPE=1,1,1**  
**SPEED=1000,100,150** '设置不同速度  
**ACCEL=1000,1000,1000**  
**DECEL=1000,1000,1000**  
**TRIGGER**  
**MOVESYNC(0, 0, 50, 1, 80 )** '同步到皮带物体上  
**MOVE\_OP(0,1)** '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)

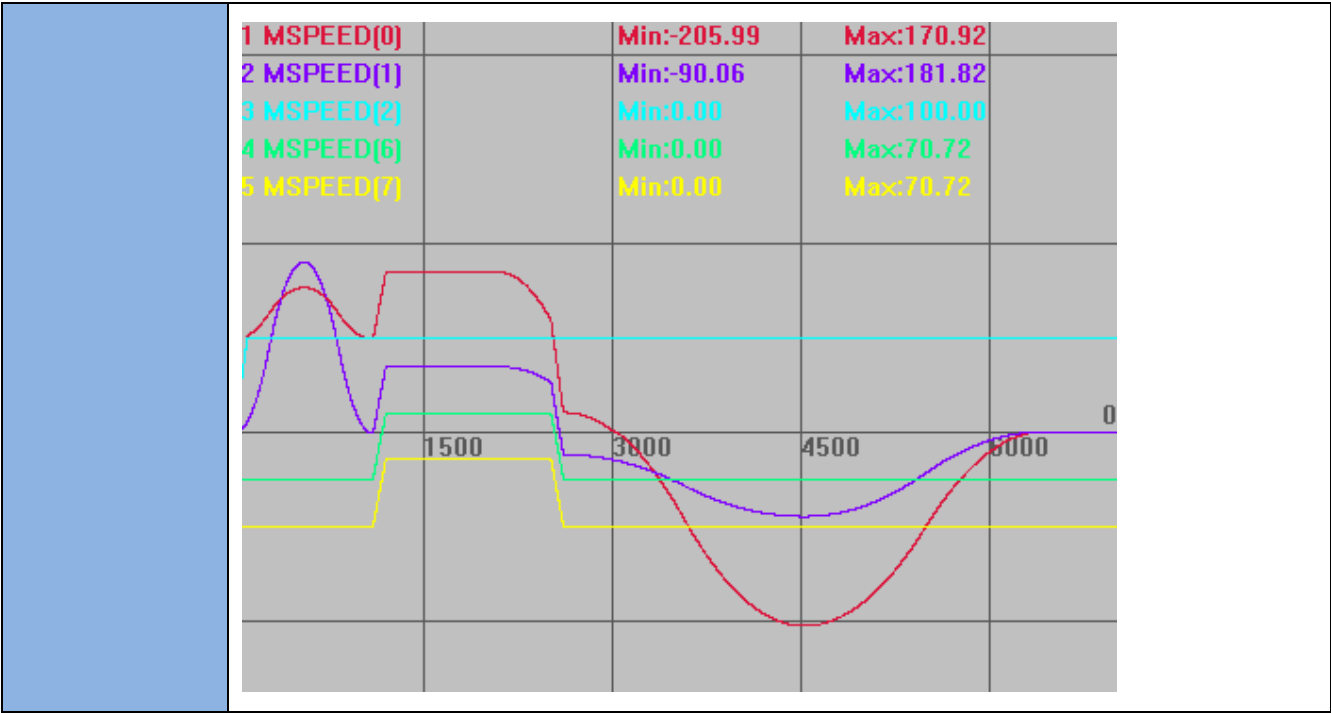
```

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

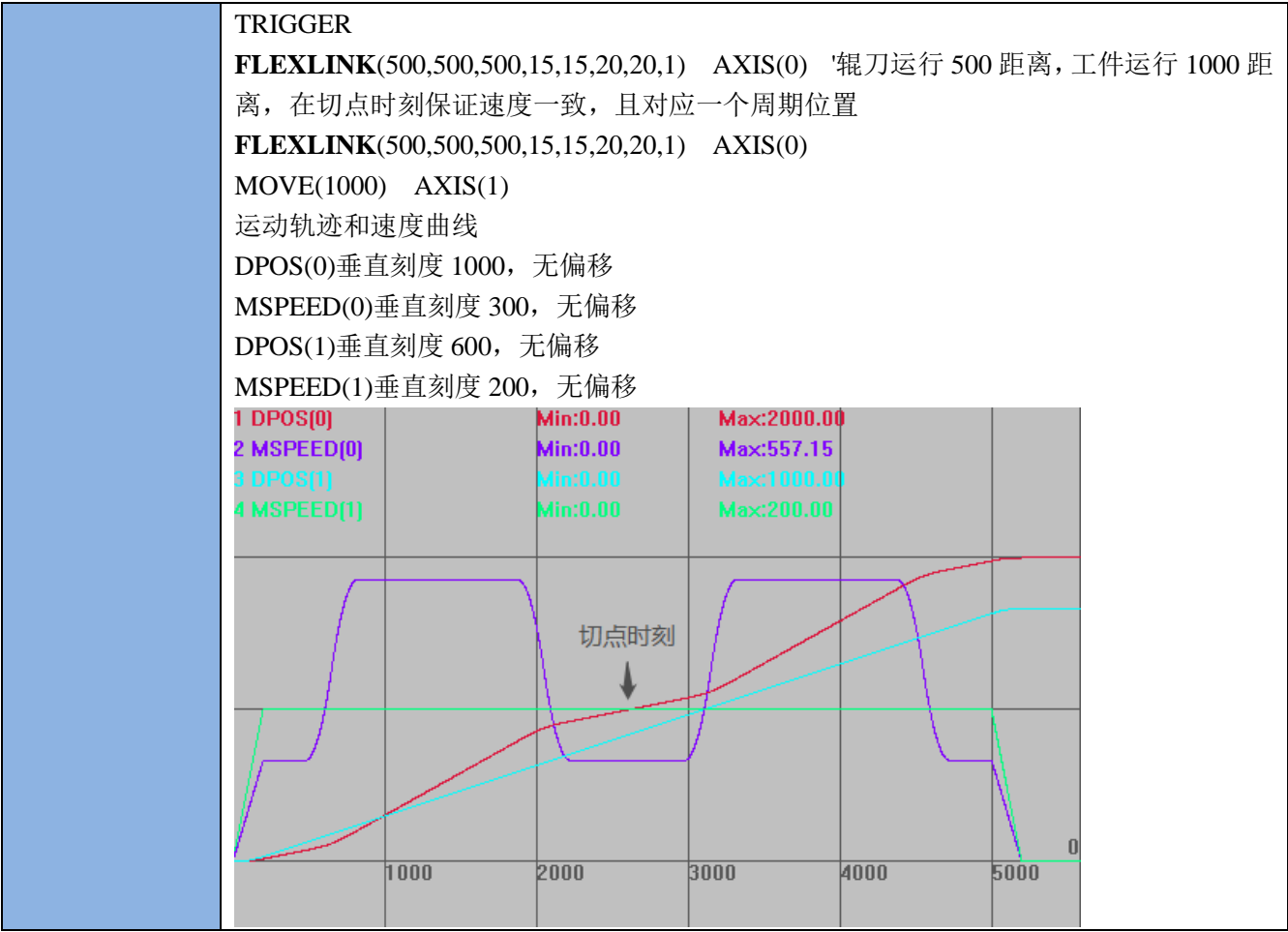
```



FLEXLINK -- 激励运动

类型	同步运动指令
描述	<p>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。</p>
语法	<p>FLEXLINK(base_dist, excite_dist, link_dist, base_in, base_out, excite_acc, excite_dec, link_axis, link_options, [start_pos], [link_offpos])</p> <p>base_dist: 跟随轴匀速运动距离</p> <p>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</p> <p>link_dist: 整个指令过程，跟随轴运动完成，主轴移动的距离</p> <p>base_in: 激励开始前，跟随轴运动距离占 base_dist 的百分比</p>

	<p>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)</p> <p>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</p> <p>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</p> <p>(以上 4 个参数在 excite_dist 不为 0 时生效)</p> <p>link_axis: 主轴轴号</p> <p>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</p> <table><tr><th>位</th><th>描述</th></tr><tr><td>bit0</td><td>当参考轴锁存信号事件 MARK 触发</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>凸轮中间启动</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>markb 启动</td></tr></table> <p>start_pos: 绝对位置触发</p> <p>link_offpos: 凸轮中间启动</p>	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
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														
适用控制器	<p>4 系列 20170518 以上固件版本支持。</p> <p>3 系列 20161212 以上固件版本支持。</p>														
例子	<p>轮切应用</p> <p>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。</p> <p>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。</p> <div><div><div>辊刀</div><div>切点</div><table><tr><td>匀速段</td><td>加速段</td><td>匀速段</td><td>匀速段</td><td>加速段</td><td>匀速段</td></tr></table><div>工件</div></div></div> <p>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</p>	匀速段	加速段	匀速段	匀速段	加速段	匀速段								
匀速段	加速段	匀速段	匀速段	加速段	匀速段										



## 7.5. 运动设置指令

### CLUTCH\_RATE -- 连接速度

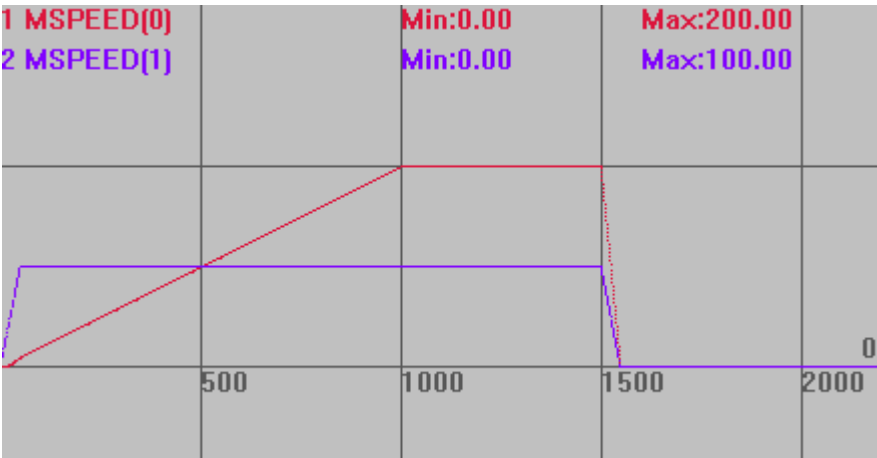
类型	轴参数
描述	<p><b>connect</b> 连接的速度, 默认值 1000000。</p> <p>用于定义连结率从 0 到设置倍率的改变时间, ratio/秒, 见例一。</p> <p>设置值如果不能远大于 CONNECT 连接比例的话, 实际连接比例会减小, 见例一位移曲线图。</p> <p>当设置为 0 时, 根据跟随轴的速度/加速度参数来跟踪连接, 比较适合于手轮运动 (当速度不够高时可能导致要持续运动一段时间才能结束)。</p>
语法	CLUTCH_RATE= value
适用控制器	通用
例子	<p>例一</p> <p>BASE(0,1)</p> <p>ATYPE=1,1</p> <p>DPOS=0,0</p>

```
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

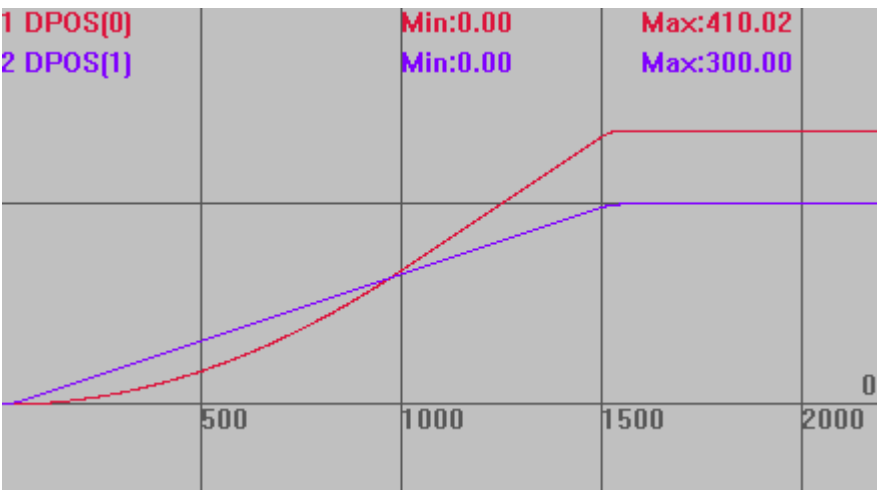
MSPEED(1)垂直刻度 200



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
```

	<div><div><div><div><div>SPEED=100,100</div><div>ACCEL=500,500</div><div>DECEL=500,500</div><div>CLUTCH_RATE = 0</div><div>TRIGGER</div><div>CONNECT(2,1) AXIS(0)</div><div>MOVE(500)AXIS(1)</div></div><div><div>'根据跟随轴的速度/加速度来跟踪连接，不一定同步'</div><div>'自动触发示波器'</div><div>'此时连接时间根据跟随轴速度加速度确定，本例中 0.2s'</div></div></div></div><div><div>速度曲线</div><div>MSPEED(0)垂直刻度 100</div><div>MSPEED(1)垂直刻度 100</div><div><div><div>1 MSPEED[0]</div><div>2 MSPEED[1]</div></div><div><div>Min:0.00</div><div>Min:0.00</div></div><div><div>Max:100.00</div><div>Max:100.00</div></div></div><div></div></div><div><div>位移曲线</div><div>DPOS(0)垂直刻度 1000</div><div>DPOS(1)垂直刻度 1000</div><div><div><div>1 DPOS[0]</div><div>2 DPOS[1]</div></div><div><div>Min:0.00</div><div>Min:0.00</div></div><div><div>Max:1000.00</div><div>Max:500.00</div></div></div><div></div></div></div>
相关指令	<a href="#">CONNECT</a>

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	<p>ENCODER_RATIO(mpos_count, input_count[, mode])</p> <p>mpos_count: 分子，不要超过 65535</p> <p>input_count: 分母，不要超过 65535</p> <p>mode: 模式</p> <table border="1"> <tr> <th>模式</th><th>描述</th></tr> <tr> <td>1</td><td>AB 1X 模式</td></tr> <tr> <td>2</td><td>AB 2X 模式</td></tr> <tr> <td>4</td><td>AB 4X 模式</td></tr> </table> <p>必须先设置 ATYPE，再调用 mode 设置。 ZMC406 20170502 以后固件支持。</p>	模式	描述	1	AB 1X 模式	2	AB 2X 模式	4	AB 4X 模式
模式	描述								
1	AB 1X 模式								
2	AB 2X 模式								
4	AB 4X 模式								
适用控制器	通用								
例子	<p><b>ENCODER_RATIO(4,1)</b>    '编码器 4 倍输入，等同于 ENCODER_RATIO(1,1,4)</p> <p><b>ENCODER_RATIO(1,-1)</b>    '编码器输入切换方向，等同于 ENCODER_RATIO(-1,1)</p>								
相关指令	<a href="#">PP_STEP</a> ， <a href="#">ENCODER</a>								

## STEP\_RATIO -- 电机齿轮比

类型	运动设置指令
描述	<p>设置步进输出齿轮比，缺省(1,1)。范围 1-65535。</p> <p>设置为负值可修改电机方向，但一般不建议。脉冲电机可用 <a href="#">INVERT_STEP</a> 指令；总线电机最好在驱动器上修改。</p> <p>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。</p>
语法	<p>STEP_RATIO(output_count, input_count)</p> <p>output_count: 分子，不要超过 65535</p> <p>input_count: 分母，不要超过 65535</p>
适用控制器	通用
例子	<b>STEP_RATIO(16,1)</b> '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	<p>BACKLASH(enable [,dist[,speed,acceleration]])</p> <p>enable: 使能与否</p> <p>dist: 距离，units 单位</p> <p>speed: 反向间隙速度</p> <p>acceleration: 反向间隙加速度</p>



适用控制器	通用
例子	<div><div><div>例一</div><div><div>BACKLASH(0)</div><div>'关闭反向间隙功能</div></div><div><div>BACKLASH(1,0.1)</div><div>'设置反向间隙 0.1mm</div></div></div><div><div>例二</div><div><div>RAPIDSTOP(2)</div><div>WAIT IDLE(0)</div></div><div><div>BASE(0)</div><div>ATYPE=5</div><div>UNITS = 1000</div><div>SPEED = 100</div><div>ACCEL = SPEED * 10</div><div>DECEL = SPEED * 10</div><div>SRAMP = 0</div><div>DPOS = 0</div><div>MPOS = 0</div></div><div><div>BACKLASH(0)</div><div>TRIGGER</div><div>'关闭反向间隙</div></div><div><div>'应用反向间隙参数</div><div>BACKLASH(1, 10, 50, 100)</div><div>'10mm 补偿</div><div>MOVE(200)</div><div>MOVE(-100)</div><div>'反向时开始补偿</div><div>END</div></div></div><div><div><div>1 DPOS[0]</div><div>Min:0.00</div><div>Max:200.00</div></div><div><div>2 MSPEED[0]</div><div>Min:304.00</div><div>Max:204.00</div></div><div><div>3 MPOS[0]</div><div>Min:0.00</div><div>Max:210.00</div></div></div><div></div></div>

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	<div>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 <b>PITCHSET(1,300,100,8,0)</b> 'MPOS=300 时，开始补偿 8 个点，间隔 100 TRIGGER MOVE(1500)</div> <table><tr><th>开始补偿点位置（MPOS）</th><th>补偿值（脉冲个数）</th></tr><tr><td>300</td><td>-</td></tr><tr><td>400</td><td>10 个脉冲</td></tr><tr><td>500</td><td>20 个脉冲</td></tr><tr><td>600</td><td>30 个脉冲</td></tr><tr><td>700</td><td>40 个脉冲</td></tr><tr><td>800</td><td>-10 个脉冲（反向补偿脉冲）</td></tr><tr><td>900</td><td>-20 个脉冲（反向补偿脉冲）</td></tr><tr><td>1000</td><td>-30 个脉冲（反向补偿脉冲）</td></tr><tr><td>1100</td><td>-40 个脉冲（反向补偿脉冲）</td></tr></table>		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）
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）																					

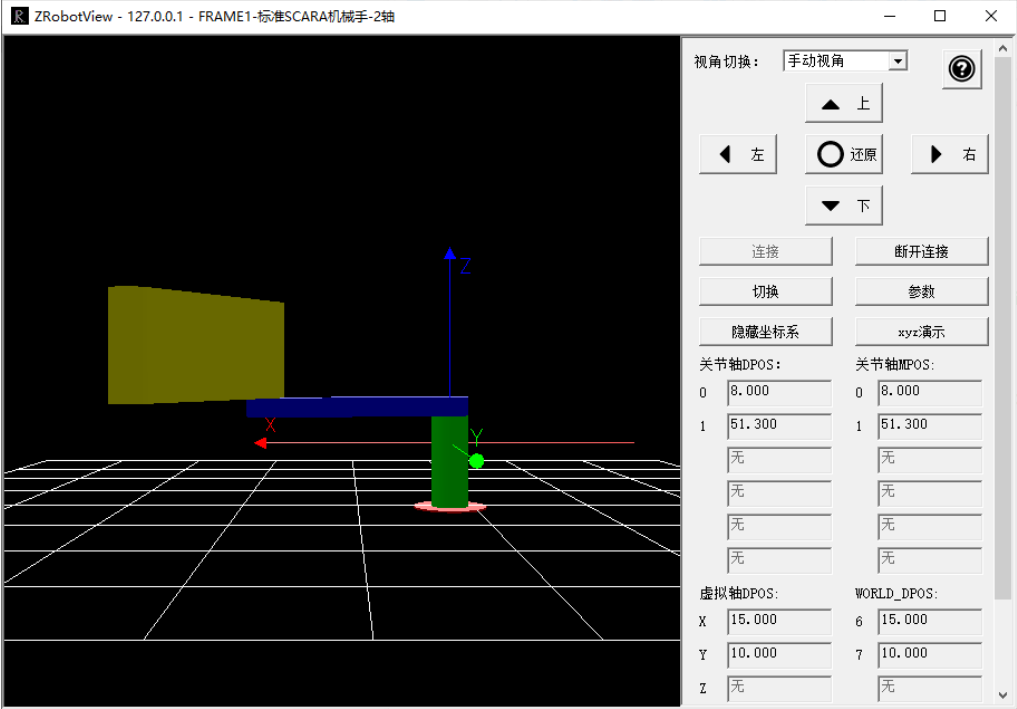
## 7.6. 机械手指令

### CONNFRAME -- 机械手逆解



类型	同步运动指令
描述	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。

	<p>⚠ 当关节轴告警等出错时，此运动会被 CANCEL。</p> <p>⚠ 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。</p> <p>⚠ 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。</p> <p>⚠ 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。</p> <p>⚠ 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。</p> <p>⚠ 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。</p>																																
语法	<p>CONNFRAME(frame, tablenum, viraxis0, viraxis1,[...])</p> <p>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）</p> <p>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</p> <p>viraxis0: 虚拟坐标系第一个轴</p> <p>viraxis1: 虚拟坐标系第二个轴</p> <p>[...]: 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</p> <p><b>FRAME 机械结构一览表</b></p> <p>详细说明请查看《正运动机械手指令说明》文档。</p> <p>其他特殊的机械结构如需支持请联系厂家。</p> <table><tr><th>frame</th><th>结构类型</th></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><tr><td>11</td><td>旋转台</td></tr><tr><td>17</td><td>双旋转台</td></tr><tr><td>18</td><td>偏移旋转台</td></tr><tr><td>19</td><td>偏移双旋转台</td></tr><tr><td>3</td><td>码垛机械手</td></tr><tr><td>103</td><td>码垛变形，喷涂机械手</td></tr><tr><td>5</td><td>旋转伸缩机械手</td></tr><tr><td>15</td><td>XY 滑台</td></tr><tr><td>102</td><td>2 轴 delta</td></tr></table>	frame	结构类型	1	标准 SCARA 机械手	101	SCARA+摆动，虚拟轴定义 4 个	105	SCARA+摆动，虚拟轴定义 5 个	106	特殊 SCARA	107	特殊 SCARA	108	特殊 5 轴 SCARA	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
frame	结构类型																																
1	标准 SCARA 机械手																																
101	SCARA+摆动，虚拟轴定义 4 个																																
105	SCARA+摆动，虚拟轴定义 5 个																																
106	特殊 SCARA																																
107	特殊 SCARA																																
108	特殊 5 轴 SCARA																																
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	<div>DIM L1,L2</div> <div>L1=10<div>'第一个关节轴长度</div></div> <div>L2=10<div>'第二个关节轴长度</div></div> <div>BASE(0,1)<div>'关节轴号是 0，1</div></div> <div>ATYPE=1,1</div> <div>UNITS=10,10<div>'这里以度为单位</div></div> <div>DPOS=0,0<div>'设置关节轴的位置，此处要根据实际情况来修改</div></div> <div>BASE(6,7)<div>'虚拟轴号 6，7</div></div> <div>ATYPE=0,0<div>'设置为虚拟轴</div></div> <div>TABLE(0,L1,L2,3600,3600)<div>'参数存储从 TABLE(0)开始存储，电机一圈 360 个脉冲</div></div> <div>UNITS=1000,1000<div>'此脉冲当量要提前设置，中途不能变化</div></div> <div>BASE(0,1)</div> <div>CONNFRAME(1,0,6,7)<div>'建立逆解，轴 0/1 的坐标根据轴 6/7 来计算运动关节轴</div></div> <div>WAIT LOADED</div> <div>BASE(6,7)</div> <div>MOVEABS(15,10)<div>'虚拟轴发送运动指令</div></div> <div>END</div> <div>连接机械手仿真工具查看运行效果如下图：</div>			

	<div><div>ZRobotView - 127.0.0.1 - FRAME1-标准SCARA机械手-2轴</div><div></div></div>
相关指令	<a href="#">CONNREFRAME</a>

CONNREFRAME -- 机械手正解

类型	同步运动指令
描述	<p>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。</p> <p>此指令为 CONNFRAME 的反运动指令。</p> <p>当虚拟轴 CONNREFRAME 运动 LOAD 时，关节轴的 CONNFRAME 运动会自动被 CANCEL。</p> <p>当关节轴的 CONNFRAME 运动 LOAD 时，虚拟轴 CONNREFRAME 运动会自动被 CANCEL。</p>
语法	<p>CONNREFRAME(frame, tablenum, axis0, axis1,[...])</p> <p>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)</p> <p>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</p> <p>axis0: 关节坐标系第一个轴</p> <p>axis1: 关节坐标系第二个轴</p> <p>[...]: 关节坐标系第 N 个轴</p> <p>BASE 轴与参数里面的轴的位置相反。</p>
适用控制器	通用
例子	DIM L1,L2

	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	<a href="#">CONNFRAME</a>	

## FRAME -- 机械手类型

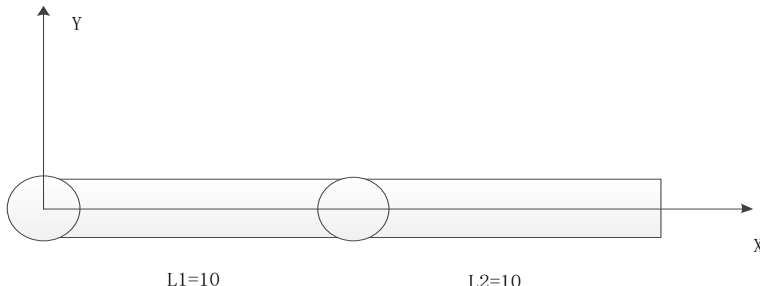
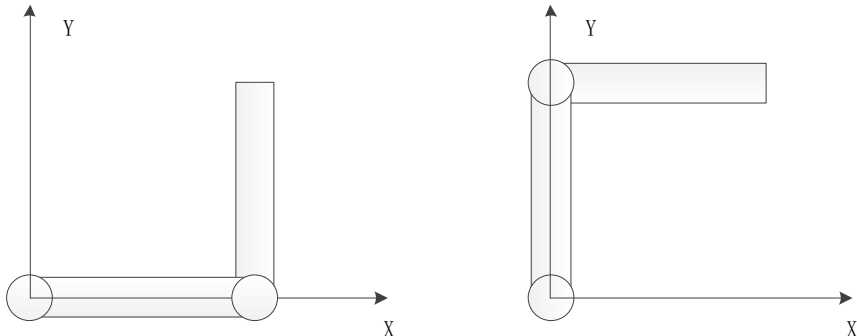
类型	机械手指令
描述	选择机械手类型，请查看正运动机械手手册。
相关指令	<a href="#">CONNFRAME</a>

## 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	<a href="#">BASE</a>

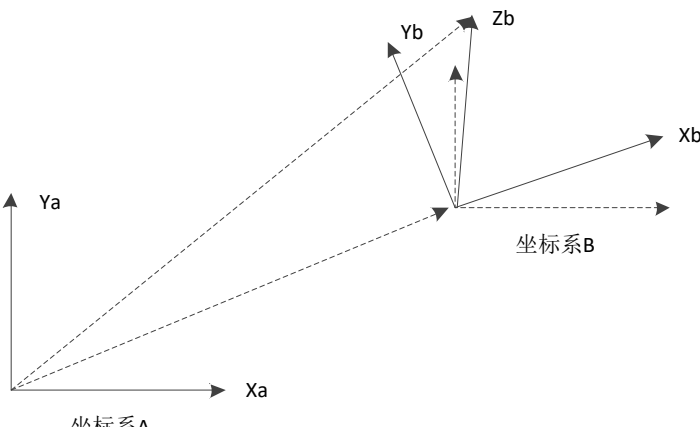
## FRAME\_TRANS2 -- 正逆解坐标转换

类型	机械手计算指令
描述	坐标转换函数。 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。																							
语法	<div>FRAME_TRANS2(tablein, tableout, dir)</div> <div>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</div> <div>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</div> <div>dir: 模式选择</div> <table><tr><th>模式</th><th>类型</th><th>描述</th></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>	模式	类型	描述	0	逆解	虚拟轴到关节轴，无姿态，自动使用当前的姿态	1	正解	关节轴到虚拟轴，无姿态输出	2	逆解	输入虚拟轴坐标，最后加上姿态	3	正解	输出虚拟轴坐标，输出最后一个位置填姿态								
模式	类型	描述																						
0	逆解	虚拟轴到关节轴，无姿态，自动使用当前的姿态																						
1	正解	关节轴到虚拟轴，无姿态输出																						
2	逆解	输入虚拟轴坐标，最后加上姿态																						
3	正解	输出虚拟轴坐标，输出最后一个位置填姿态																						
适用控制器	通用																							
例子	<div>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。</div> <div>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。</div> <div></div> <div>虚拟轴坐标 (10,10) 时，有两种姿态 关节轴 (0,90) 和关节轴 (90,-90)</div> <div></div> <div>坐标转换表</div> <table><tr><th>状态</th><th>BASE 轴</th><th>输入 X、Y、姿态</th><th>使用指令模式</th><th>输出关节坐标</th></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 rowspan="2">frame_trans2(100,200,1)</td><td>table(200,20,0,0)</td></tr><tr><td>table(100,0,90)</td><td>table(200,10,10,0)</td></tr></table>	状态	BASE 轴	输入 X、Y、姿态	使用指令模式	输出关节坐标	逆解	虚拟轴	table(100,20,0,0)	frame_trans2(100,200,0)	table(200,0,0)	table(100,10,10,1)	table(200,0,90)	table(100,10,10,1)	frame_trans2(100,200,2)	table(200,90,-90)	输入关节坐标	使用指令模式	输出 X、Y、姿态	table(100,0,0)	frame_trans2(100,200,1)	table(200,20,0,0)	table(100,0,90)	table(200,10,10,0)
状态	BASE 轴	输入 X、Y、姿态	使用指令模式	输出关节坐标																				
逆解	虚拟轴	table(100,20,0,0)	frame_trans2(100,200,0)	table(200,0,0)																				
		table(100,10,10,1)		table(200,0,90)																				
		table(100,10,10,1)	frame_trans2(100,200,2)	table(200,90,-90)																				
		输入关节坐标	使用指令模式	输出 X、Y、姿态																				
		table(100,0,0)	frame_trans2(100,200,1)	table(200,20,0,0)																				
		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)

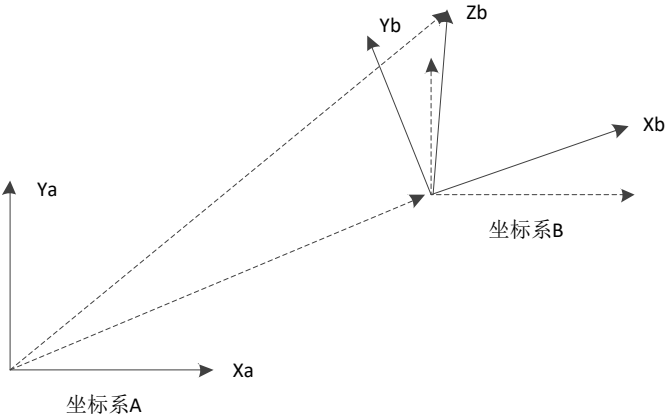
## FRAME\_ROTATE -- 工件坐标系变换

类型	机械手计算指令
描述	<p>用于对工件坐标系进行平移和旋转。</p> <p>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。</p> <p>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。</p> <p>使用时需控制器当前存在机械链接。</p> <p>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。</p> <p>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。</p>
语法	<p>FRAME_ROTATE(X,Y,Z,RX,RY,RZ)</p> <p>X: 坐标系 B 沿 X<sup>^</sup>的平移距离  Y: 坐标系 B 沿 Y<sup>^</sup>的平移距离  Z: 坐标系 B 沿 Z<sup>^</sup>的平移距离  RX: 坐标系 B 沿 X<sup>^</sup>的旋转的角度  RY: 坐标系 B 沿 Y<sup>^</sup>的旋转的角度  RZ: 坐标系 B 沿 Z<sup>^</sup>的旋转的角度</p>  <p>坐标系A</p> <p>坐标系B</p> <p>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。</p>
适用控制器	通用



例程	<p>例一 以 FRAME=2, DETLA 为例, 对其绕 X 轴旋转 90 度。</p> <pre> 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) <b>FRAME_ROTATE</b>(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) <b>FRAME_ROTATE</b>(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 </pre>
相关指令	<a href="#">FRAME_ROTATE2</a>

FRAME\_ROTATE2 -- 坐标系变换计算

类型	机械手计算指令						
描述	<p>手动计算坐标系旋转后的坐标值。</p> <p>使用时需控制器当前存在机械链接。</p> <p>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。</p> <p>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。</p>						
语法	<p>FRAME_ROTATE2(tablein, tableout, dir[, x,y,z[, rx,ry,rz]])</p> <p>ret = FRAME_ROTATE2(tablein, tableout, dir[, x,y,z[, rx,ry,rz]])</p> <p>tablein: 转换前，填写的坐标存储 table 位置</p> <p>tableout: 转换后，输出的坐标存储 table 位置</p> <p>dir: 方向选择</p> <table><tr><td>值</td><td>描述</td></tr><tr><td>0</td><td>DPOS 到 WORLD_DPOS</td></tr><tr><td>1</td><td>WORLD_DPOS 到 DPOS</td></tr></table> <p>X: 坐标系 B 沿 X^的平移距离</p> <p>Y: 坐标系 B 沿 Y^的平移距离</p> <p>Z: 坐标系 B 沿 Z^的平移距离</p> <p>RX: 坐标系 B 沿 X^的旋转的角度</p> <p>RY: 坐标系 B 沿 Y^的旋转的角度</p> <p>RZ: 坐标系 B 沿 Z^的旋转的角度</p> <p>[, x,y,z[, rx,ry,rz]]: 不填时使用当前的缺省 rotate 参数</p> <p>Ret: 返回成功与否；-1-成功，0-未成功</p> 	值	描述	0	DPOS 到 WORLD_DPOS	1	WORLD_DPOS 到 DPOS
值	描述						
0	DPOS 到 WORLD_DPOS						
1	WORLD_DPOS 到 DPOS						
适用控制器	通用						
例程	<p>例一 以 FRAME=2，DETLA 为例；对其绕 X 轴旋转 90 度。</p> <p>BASE(0,1,2)</p>						

```

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

```

	<pre> 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 </pre>
相关指令	<a href="#">FRAME_ROTATE</a>

## 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	<p><b>关节轴直线插补。</b></p> <p>机械手关节插补运动，机械手末端直线运动到指定坐标。</p> <p><b>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。</b></p>
语法	<p>MOVER_L(distance1 [,distance2 [,distance3 [,distance4...]]])</p> <p>distance1: 第一个轴运动距离</p> <p>distance2: 下一个轴运动距离</p>
适用控制器	4 系列 20170511 以上固件支持
例程	<pre> 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 <b>MOVER_LABS</b>(90, 0)   '末端直线运动 WAIT IDLE            '等待运动停止 </pre>

	PRINT *DPOS
相关指令	<a href="#">MOVER_C</a> , <a href="#">MOVER_C3</a>

## MOVER\_C/MOVER\_CABS -- 关节轴平面圆弧

类型	运动指令										
描述	<p>关节轴直接走圆弧插补运动。</p> <p>此指令正解模式下使用。</p> <p>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。</p>										
语法	<p>MOVER_C/MOVER_CABS (end1,end2,centre1,centre2,mode,[dis1,...,disn])</p> <p>end1: 第 1 个轴运动距离参数 1</p> <p>end2: 第 2 个轴运动距离参数 1</p> <p>centre1: 第 1 个轴运动距离参数 2</p> <p>centre2: 第 2 个轴运动距离参数 2</p> <p>mode: 模式</p> <table border="1"> <thead> <tr> <th>值</th><th>描述</th></tr> </thead> <tbody> <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> </tbody> </table> <p>dis1- disn: 螺旋轴的距离</p>	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。
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。										
适用控制器	4 系列 20170511 以上固件支持										
例程	<pre> 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) </pre>										
相关指令	<a href="#">MOVER_L</a> , <a href="#">MOVER_C3</a>										

## MOVER\_C3/MOVER\_C3ABS -- 关节轴空间圆弧

类型	运动指令										
描述	<p>关节轴直接走空间圆弧插补运动。</p> <p>此指令正解模式下使用。</p> <p>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。</p>										
语法	<p>MOVER_C3 (endx, endy, endz, midx, midy, midz, mode[, dis1][,dis2][,dis3])</p> <p>end1: 第 1 个轴运动距离参数 1</p> <p>end2: 第 2 个轴运动距离参数 1</p> <p>end3: 第 3 个轴运动距离参数 1</p> <p>centre1: 第 1 个轴运动距离参数 2</p> <p>centre2: 第 2 个轴运动距离参数 2</p> <p>centre3: 第 3 个轴运动距离参数 2</p> <p>mode: 模式</p> <table border="1"> <thead> <tr> <th>值</th><th>描述</th></tr> </thead> <tbody> <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> </tbody> </table> <p>dis1- disn: 螺旋轴的距离</p>	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。
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。										
适用控制器	4 系列 20170511 以上固件支持										
例程	<pre> 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)    'CONNREFRAME 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) </pre>										
相关指令	<a href="#">MOVER_L</a> , <a href="#">MOVER_C</a>										

## FRAME\_CAL -- 参数矫正

类型	机械手计算指令
描述	<p>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。</p> <p>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。</p> <p>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。</p> <p>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。</p> <p>FRAME_CAL 只做计算，指令返回值为-1 计算成功，返回 0 计算失败。</p> <p><b>FRAME_CAL 的 BASE 轴必须是在 FRAME 状态的轴。</b></p>
语法	<p>FRAME_CAL(tablein,space,groups,tableaux, zeroout, [tableout2])</p> <p>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</p> <p>space: 每两个点之间间隔的 table 元素</p> <p>groups: 点数</p> <p>tableaux: 辅助参数的 table 编号，部分 frame 需要</p> <p>zeroout: 计算出来理论零点时关节轴的绝对坐标的 table 编号</p> <p>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</p>
适用控制器	通用
例程	查看机械手指令说明手册章节



## 第八章 程序结构与流程指令

### 8.1. 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

## 8.2. 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	<b>CONST</b> CVARNAME = value CVARNAME: 常量名 value: 常量值
适用控制器	通用
例子	例一 <b>CONST</b> MAX_VALUE = 100000      '定义常量 TABLE(0)=MAX_VALUE          'table(0)赋值 10000  例二 <b>GLOBAL CONST</b> MAX_AXIS=6      '定义总轴数
相关指令	<a href="#">DIM</a>

### 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	<b>DIM</b> 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	<b>DIM</b> ARRAY1(100)      '定义数组 ARRAY1 <b>DIM</b> 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	<a href="#">CONST</a> , <a href="#">LOCAL</a> , <a href="#">GLOBAL</a>

## LOCAL -- 局部定义

类型	语法指令
描述	<p>定义局部变量，局部变量。</p> <p>局部变量主要用于 SUB 中。</p> <p>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。</p> <p>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。</p>
语法	<pre>SUB subA()     LOCAL  localname      'localname 局部变量名     ..... ENDSUB</pre>
适用控制器	通用
例子	<pre>SUB aaa()     LOCAL  v1              '定义局部变量 V1     v1=100 END SUB</pre>
相关指令	<a href="#">DIM</a> , <a href="#">GLOBAL</a>

## GLOBAL -- 全局定义

类型	语法指令
描述	<p>定义全局变量，数组；定义全局 SUB 过程。</p> <p>全局定义的量可以在整个项目的任意程序文件中使用。</p>
语法	<p>语法 1: GLOBAL VAR1</p> <p>语法 2: GLOBAL SUB SUB1()</p> <p>语法 3: GLOBAL CONST CVARNAME = value</p> <p>参数：</p> <p>VAR1: 变量名称</p> <p>SUB1: 过程名称</p> <p>CVARNAME: 常量名称</p> <p>value: 常量值</p>
适用控制器	通用
例子	<pre>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</pre>
相关指令	<a href="#">DIM</a> , <a href="#">LOCAL</a>

## 8.3. 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位置。 <b>谨慎对超长数组进行操作，特别是数组 TABLE。</b>
语法	DMINS arrayname(pos, size) arrayname: 数组名称 pos: 数组索引 size: 要修改的个数，注意加上 pos 不要超过数组大小
适用控制器	通用
例子	<pre> 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，原有的所有元素都向后移动一个位置 aa(0) = 10      '为插入元素赋值 ?*aa           '打印插入后的数组所有元素 </pre>
相关指令	<a href="#">DMDEL</a> ， <a href="#">DMCPY</a>

### 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。 <b>不要一次修改超过 500 元素。</b>
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	<pre> 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 </pre>

	?*aaa          '打印，其中 10、11、12、13、14 为 4，其余为 0
相关指令	<a href="#">DMINS</a> ， <a href="#">DMCPY</a>

## 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。 <b>谨慎对超长数组进行操作，特别是数组 TABLE。</b>
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 <b>DMDEL</b> 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	<a href="#">DMINS</a> ， <a href="#">DMCPY</a>

## 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。 <b>谨慎对超长数组进行操作，特别是数组 TABLE。</b>
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 <b>DMCPY</b> 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	<a href="#">DMINS</a> , <a href="#">DMDEL</a>
------	---

## DMSET -- 数组赋值

类型	语法指令
描述	数组区域赋值。 谨慎对超长数组进行操作，特别是数组 <b>TABLE</b> 。
语法	DMSET arrayname(pos, size, data) pos: 起始索引 size: 长度 data: 设置的数值
适用控制器	通用
例子	<pre> <b>DMSET</b>  TABLE(0,10,2)    '数组区域赋值 FOR i=0 TO 9     PRINT "TABLE",i, TABLE(i)    '打印数组 NEXT <b>DMSET</b>  TABLE(0,10,3)    '数组区域赋值 FOR i=0 TO 9     PRINT  "TABLE",i, TABLE(i)    '打印数组 NEXT </pre>
相关指令	<a href="#">DMINS</a> , <a href="#">DMDEL</a>

## 8.4. 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 <b>GLOBAL</b> 描述以定义全局使用的 SUB 过程。
语法	<pre> SUB label([para1] [,para2]...) ... END SUB </pre> <p>参数:</p> <p>label: 过程名称，不能与现有的关键词冲突</p> <p>para1: 过程调用时传入的参数，自动作为 <b>LOCAL</b> 局部变量</p> <p>para2: 过程调用时传入的参数，自动作为 <b>LOCAL</b> 局部变量</p>
适用控制器	通用
例子	<pre> SUB sub1()    '定义过程 SUB1，只能在当前文件中使用 ?1 </pre>

	<pre> ... 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 </pre>
相关指令	<a href="#">SUB PARA</a> ， <a href="#">SUB IFPARA</a>

## 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	<pre> 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 </pre>
相关指令	<a href="#">SUB</a> ， <a href="#">SUB IFPARA</a>

## 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	<pre> AAA(0,100)    '传入 num1,num2 AAA(,100)     '只传入 num2 END SUB AAA(NUM1,NUM2) </pre>

	<pre> 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 </pre>
相关指令	<a href="#">SUB</a> , <a href="#">SUB PARA</a>

## GOSUB/CALL -- SUB 调用

类型	程序结构
描述	<p><b>SUB 过程调用</b>，只能调用本文件的 SUB 过程或全局 SUB 过程。</p> <p>直接调用 SUB 过程时，可以省掉 GOSUB 语句。</p> <p>SUB 过程没有参数传递时，可以省掉括号()。</p> <p>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。</p>
语法	<pre> GOSUB/CALL  label             label: SUB 过程名 </pre>
适用控制器	通用
例子	<pre> '主程序 main:     <b>GOSUB</b> 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 </pre>



## GSUB -- 自定义子函数-G 代码格式

类型	语法指令
描述	<p>用户自定义 <b>GSUB</b> 过程。</p> <p>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。</p>
语法	<pre>GSUB  label([char1] [,char2]...)</pre> <p>...</p> <pre>END SUB</pre> <p>参数：</p> <p>Label: 过程名称，不能与现有的关键词冲突</p> <p>char1: 过程调用时传入的字母参数，自动作为 LOCAL 局部变量</p> <p>char2: 过程调用时传入的字母参数，自动作为 LOCAL 局部变量</p> <p><b>字母参数只能为单字符</b></p>
适用控制器	通用
例子	<pre>G01 X100 Y100 Z100 U100      '调用 G01</pre> <pre>END                          '主程序结束</pre> <pre>GLOBAL GSUB G01(X, Y, Z, U)    '定义 GSUB 过程 G01</pre> <p>...</p> <pre>END SUB</pre>
相关指令	<a href="#">GSUB_PARA</a> , <a href="#">GSUB_IFPARA</a>

## 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	<pre>GSUB_PARA(char)</pre> <p>char: GSUB 定义时传入的字母参数</p>
适用控制器	通用
例子	<pre>GSUB AAA(X,Y,Z)</pre> <p>    ?GSUB_PARA(X)   '调用 AAA 时，打印传递的 X 值</p> <p>    ?GSUB_PARA(Y)   '打印传递的 Y 值</p> <p>    ?GSUB_PARA(Z)   '打印传递的 Z 值</p> <pre>END SUB</pre>
相关指令	<a href="#">GSUB</a> , <a href="#">GSUB_IFPARA</a>

## 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	<pre> 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 </pre>
相关指令	<a href="#">GSUB -- 自定义子函数-G 代码格式</a> , <a href="#">GSUB PARA</a>

## 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	<p>用户 SUB 过程返回或返回值。</p> <p>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。</p> <p>不同任务的返回值不同。</p>
语法	RETURN
适用控制器	通用
例子	<pre> CALL sub1 ?RETURN    '结果为 111 END        '主程序结束 </pre>

	SUB sub1() <b>RETURN</b> 111     '返回 111 END SUB
--	--

## 8.5. 结构体定义指令

### STRUCTURE -- 结构体定义

类型	语法指令
描述	结构体定义。  5 系列控制器 20180327 以上固件支持。 4 系列控制器 fast 版本 20190107 以上固件支持。
语法	Structure 结构名称 Dim 成员 1 名称 [As 数据类型 1] ... .. Dim 成员 n 名称[(数组长度)] [As 数据类型 1] End Structure  数据类型只支持结构体，每个元素都同数组元素，占用一个数组元素空间。 结构体不能递归。  结构变量定义： DIM 变量名 AS 结构名 DIM 结构数组名[(数组长度)] AS 结构名  GLOBAL 变量名 AS 结构名 GLOBAL 结构数组名[(数组长度)] AS 结构名  预留功能： LOCAL 变量名 AS 结构名  支持使用 FLASH_WRITE, FLASH_READ 指令读写结构体定义的变量和数组。 FLASH_WRITE id, 结构变量 FLASH_WRITE id, 结构数组 FLASH_WRITE id, 结构数组(index) FLASH_WRITE id, 结构数组(index).item FLASH_WRITE id, 结构数组(index).item 数组(index) FLASH_READ 同上  支持使用数组操作指令操作结构体数组。 DMINs 结构数组(index) [,numes ] DMINs 结构数组(index).item 数组(index) [,numes ] DMDEL 同上

	<p>DMCPY 结构数组 1(index1), 结构数组 2(index2) [,size]</p> <p>DMSET 只支持对最后一层的数组进行操作, 不能对结构数组赋值。 DMSET 结构变量.item 数组(index, size, data) DMADD 同上</p>
适用控制器	通用
例子	<pre> '声明结构体 AA GLOBAL Structure ClassAA     DIM AA_val1          '成员变量     DIM AA_array(10)     '成员数组 END Structure  '声明结构体 BB GLOBAL Structure ClassBB     DIM BB_val1 AS ClassAA  '成员变量为结构体 END Structure  '创建结构体变量 GLOBAL Class1 AS ClassAA GLOBAL Class2 AS ClassBB  Class1.AA_val1=123 ?Class1.AA_val1  class1.AA_array="abc" ?class1.AA_array  Class2.BB_val1.AA_val1=567 ?Class2.BB_val1.AA_val1  Class2.BB_val1.AA_array="zxc" ?Class2.BB_val1.AA_array  AA_val1=8 FLASH_WRITE 0,AA_val1 AA_val1=123 FLASH_READ 0,AA_val1 ?AA_val1  END </pre>
相关指令	<a href="#">DIM</a> , <a href="#">GLOBAL</a> , <a href="#">UNION</a>

## UNION -- 共用体定义

类型	语法指令
描述	<p>共用体定义。</p> <p>5 系列控制器 20180327 以上固件支持。 4 系列控制器 fast 版本 20190107 以上固件支持。</p>
语法	<p>UNION 结构名称</p> <p>    Dim 成员 1 名称 [As 数据类型 1]</p> <p>    ... ..</p> <p>    Dim 成员 n 名称[(数组长度)] [As 数据类型 1]</p> <p>END UNION</p> <p>结构变量定义：</p> <p>DIM 变量名 AS 结构名</p> <p>DIM 结构数组名[(数组长度)] AS 结构名</p> <p>GLOBAL 变量名 AS 结构名</p> <p>GLOBAL 结构数组名[(数组长度)] AS 结构名</p> <p>预留功能：</p> <p>LOCAL 变量名 AS 结构名</p> <p>支持使用 FLASH_WRITE, FLASH_READ 指令读写结构体定义的变量和数组。</p> <p>FLASH_WRITE id, 结构变量</p> <p>FLASH_WRITE id, 结构数组</p> <p>FLASH_WRITE id, 结构数组(index)</p> <p>FLASH_WRITE id, 结构数组(index).item</p> <p>FLASH_WRITE id, 结构数组(index).item 数组(index)</p> <p>FLASH_READ 同上</p> <p>支持使用数组操作指令操作结构体数组。</p> <p>DMINS 结构数组(index) [,numes ]</p> <p>DMINS 结构数组(index).item 数组(index) [,numes ]</p> <p>DMDEL 同上</p> <p>DMCPY 结构数组 1(index1), 结构数组 2(index2) [,size]</p> <p>DMSET 只支持对最后一层的数组进行操作，不能对结构数组赋值。</p> <p>DMSET 结构变量.item 数组(index, size, data)</p> <p>DMADD 同上</p>
适用控制器	通用
例子	<p>'声明共用体 AA</p> <p>GLOBAL UNION ClassAA</p> <p>    DIM AA_val1               '成员变量</p> <p>    DIM BB_val1               '成员变量</p>

	<pre> DIM AA_array(4)      '成员数组 END UNION  '创建共用体变量 GLOBAL Class1 AS ClassAA  Class1.AA_val1=123 ?Class1.AA_val1      '结果: 123  Class1.BB_val1=456 ?Class1.BB_val1      '结果: 456 ?Class1.AA_val1      '结果: 456 END </pre>
相关指令	<a href="#">DIM</a> , <a href="#">GLOBAL</a> , <a href="#">STRUCTURE</a>

## 8.6. 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	<pre> a=100 <b>GOTO</b> 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 </pre>

### ON GOSUB -- 条件跳转

类型	程序结构
描述	当 <b>expression</b> 条件为真时，调用过程 label。
语法	<pre> ON expression  GOSUB  label       expression: 判断条件       label: 跳转 sub、label 名称 </pre>

适用控制器	通用
例子	<pre> a=100 <b>ON</b> a&gt;10 <b>GOSUB</b> label1    'a&gt;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 </pre>

## ON GOTO -- 条件跳转 2

类型	程序结构
描述	条件跳转，当 <b>expression</b> 条件为真时跳转，不会压栈。
语法	<pre> <b>ON</b> expression <b>GOTO</b> label expression: 判断条件 label: 跳转 sub、label 名称 </pre>
适用控制器	通用
例子	<pre> a=100 <b>on</b> a&gt;10 <b>goto</b> label1 a=1000 END                        '主程序结束  label1: PRINT a END                        'goto 跳转无法 return 返回 </pre>

## 8.7. 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	<pre> <b>IF</b> &lt;condition1&gt; <b>THEN</b>     commands <b>ELSEIF</b> &lt;condition2&gt; <b>THEN</b>     commands <b>ELSE</b> </pre>

	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	<a href="#">THEN</a> , <a href="#">ENDIF</a>

## THEN -- 条件判断结构

类型	程序结构
描述	参见: <b>IF</b>
适用控制器	通用

## ENDIF -- 条件判断结构

类型	程序结构
描述	参见: <b>IF</b>
适用控制器	通用

## ELSEIF -- 条件判断结构

类型	程序结构
描述	参见: <b>IF</b>
适用控制器	通用



## 8.8. 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 <b>BASIC</b> 语法。
语法	<p>FOR variable=start TO end [STEP increment]            commands  NEXT variable</p> <p>参数：</p> <p>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</p> <p>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。</p>
适用控制器	通用
例子	<pre>LOCAL a <b>FOR</b> a=1 TO 100   '1 循环至 100     PRINT a       '打印 a <b>NEXT</b></pre>
相关指令	<a href="#">TO</a> , <a href="#">STEP</a> , <a href="#">NEXT</a>

### TO -- for 循环结构

类型	程序结构
描述	参见: <b>FOR</b>
适用控制器	通用

### STEP -- for 循环结构

类型	程序结构
描述	参见: <b>FOR</b>
适用控制器	通用

## NEXT -- for 循环结构

类型	程序结构
描述	参见: <b>FOR</b>
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 <b>WHILE</b> IN(4)=OFF '直到输入 4 有效, 退出循环 a=a+1 PRINT a DELAY(1000) <b>WEND</b>

## WEND -- while 循环结构

类型	程序结构
描述	参见: <b>WHILE</b>
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 <b>EXIT FOR</b> '必须采用这种方式, 否则 IF 和 ENDIF 不匹配 <b>NEXT</b>

## 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	<pre> a=0 <b>REPEAT</b>           '循环执行下面语句     PRINT a     a=a+1     DELAY(1000) <b>UNTIL</b> IN(4)=ON    '直到输入 4 有效           </pre>

## UNTIL -- 条件结构

类型	程序结构
描述	参见: <b>REPEAT</b> 、 <b>WAIT</b>
适用控制器	通用

## 8.9. 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	<b>DELAY</b> (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。

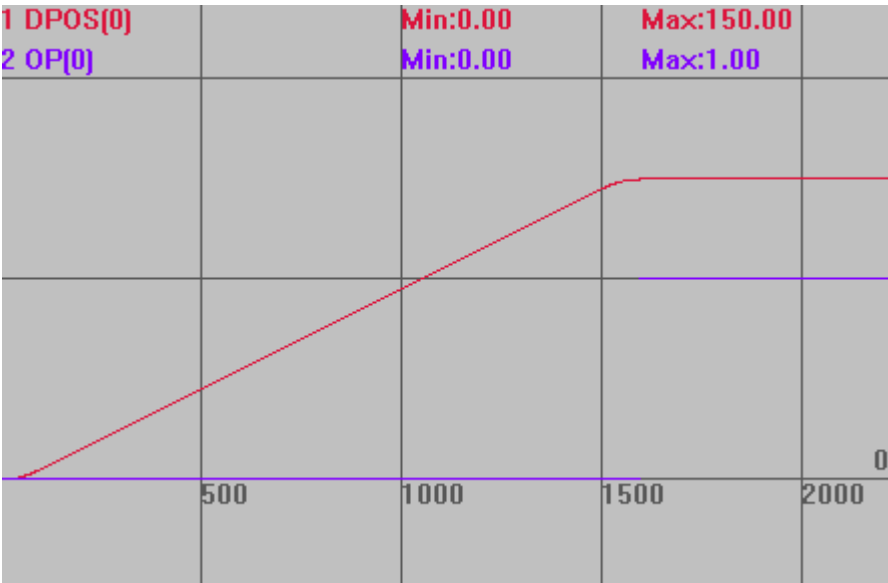
语法	WAIT UNTIL condition1 [and condition2 or condition3 ...] 可以通过逻辑运算操作多个条件。
适用控制器	通用
例子	<p>例一</p> <p><b>WAIT UNTIL</b> DPOS(0) &gt; 0 '等待轴 0 位置超过 0</p> <p>例二 与 TICKS 一起使用</p> <p>TICKS=2000 'ticks 设为 2000</p> <p><b>WAIT UNTIL</b> TICKS&lt;0 '等待 2s</p> <p>?"执行下一步"</p> <p>例三 与逻辑条件一起使用</p> <p><b>WAIT UNTIL</b> IDLE(0)=-1 AND IDLE(1)=-1 AND IDLE(2)=-1</p> <p>'等待轴 0、1、2 都停止</p>

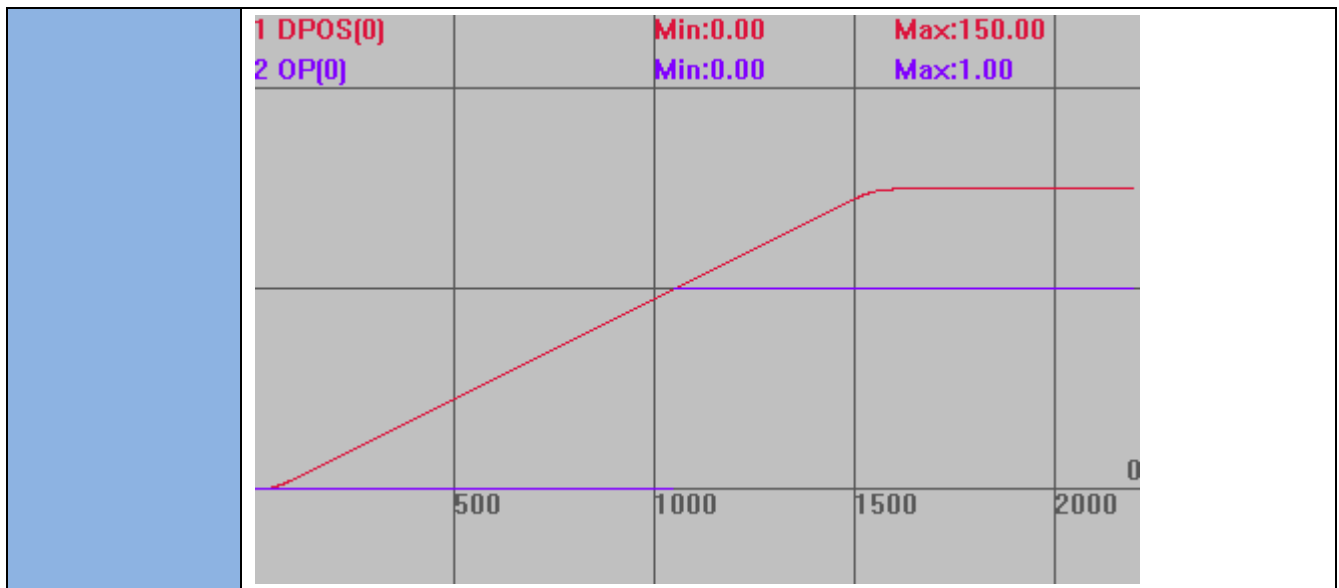
## WAIT IDLE -- 等待轴停止

类型	语法指令
描述	等待 <b>BASE</b> 轴运动完成， <b>BASE</b> 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	<p>例一</p> <p>BASE(0,1)</p> <p>MOVE(100,100)</p> <p><b>WAIT IDLE</b> '等待当前插补运动结束</p> <p>例二</p> <p>BASE(0,1)</p> <p>MOVE(100,100)</p> <p>BASE(2,3)</p> <p>MOVE(200,200)</p> <p><b>WAIT UNTIL IDLE(0) AND IDLE(1) AND IDLE(2) AND IDLE(3)</b></p> <p>'等待轴 0, 1, 2, 3 停止</p> <p>?"运动完成"</p>

## WAIT LOADED -- 等待轴缓冲空

类型	语法指令
描述	等待 <b>BASE</b> 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	<div>与 WAIT IDLE 的区别</div> <div>BASE(0)</div> <div>ATYPE=1</div> <div>UNITS=100</div> <div>DPOS=0</div> <div>SPEED=100</div> <div>ACCEL=1000</div> <div>MERGE=1</div> <div>TRIGGER</div> <div>MOVE(100)     '当前运动</div> <div>MOVE(50)     '缓冲运动，此时缓冲区只有这一条运动</div> <div>              '当本条运动执行时，缓冲区就已经清空</div> <div>WAIT IDLE     '使用 wait idle 时，要等到所有运动完成才能往下执行</div> <div>OP(0,ON)     '打开 op0</div> <div><div>运动轨迹</div><div>DPOS(0)垂直刻度 100</div><div>OP(0)垂直刻度 1</div><div><table><tr><td>1 DPOS[0]</td><td></td><td>Min:0.00</td><td>Max:150.00</td><td></td></tr><tr><td>2 OP[0]</td><td></td><td>Min:0.00</td><td>Max:1.00</td><td></td></tr></table></div><div>WAIT LOADED   '使用 wait loaded 时，运动缓冲空即可往下执行</div><div>OP(0,ON)</div></div>	1 DPOS[0]		Min:0.00	Max:150.00		2 OP[0]		Min:0.00	Max:1.00	
1 DPOS[0]		Min:0.00	Max:150.00								
2 OP[0]		Min:0.00	Max:1.00								



## 8.10. ZINDEX 指针指令

### ZINDEX\_LABEL -- 建立指针索引

类型	语法指令
描述	建立索引指针，便于后续调用指针内容。
语法	Pointer = zindex_label(subname) subname: 数组或 SUB 名称
适用控制器	通用
例子	DIM arr1(100) '定义数组 arr1(0,1) '对数组 0 地址赋值 1 Pointer = ZINDEX_LABEL(arr1) '建立索引指针 PRINT ZINDEX_ARRAY(Pointer) (0) '访问数组，打印数组第一位数据，结果 1
相关指令	<u>ZINDEX_CALL</u> , <u>ZINDEX_ARRAY</u> , <u>ZINDEX_VAR</u>

### ZINDEX\_CALL -- 访问 SUB 函数

类型	语法指令
描述	通过索引指针来调用 SUB 函数。
语法	ZINDEX_CALL(zidnex) (subpara, ...) zidnex: 通过 ZINDEX_LABEL 生成的索引指针 subpara: sub 的参数调用
适用控制器	通用

例子	Pointer = ZINDEX_LABEL(sub1)    '建立索引指针 ZINDEX_CALL(Pointer) (2)        '调用函数 SUB    sub1(a) PRINT a END SUB
相关指令	<a href="#">ZINDEX_LABEL</a>

## ZINDEX\_ARRAY -- 访问数组

类型	语法指令
描述	通过索引指针来访问数组。
语法	var = ZINDEX_ARRAY (pointer)(index) pointer: 通过 ZINDEX_LABEL 生成的指针索引 index: 数组索引
适用控制器	通用
例子	DIM    arr1(100)        '定义数组 arr1(0,1)               '对数组 0 地址赋值 1 Pointer = ZINDEX_LABEL(arr1)        '建立索引指针 PRINT   ZINDEX_ARRAY(Pointer) (0)   '访问数组，打印数组第一位数据，结果 1
相关指令	<a href="#">ZINDEX_LABEL</a>

## ZINDEX\_VAR -- 访问变量

类型	语法指令
描述	通过索引指针来访问变量。
语法	ZINDEX_VAR(zindex) zidnex: 通过 ZINDEX_LABEL 生成的索引指针  zindex= ZINDEX_LABEL(varname) ZINDEX_VAR(zindex)=value VAR2 = ZINDEX_VAR(zindex)
适用控制器	通用
例子	global   gTestVar global   VarAdd1 VarAdd1=ZINDEX_LABEL(gTestVar) ZINDEX_VAR(VarAdd1)=10 ?ZINDEX_VAR(VarAdd1)
相关指令	<a href="#">ZINDEX_LABEL</a>

## ZINDEX\_STRUCT -- 访问结构体

类型	语法指令
描述	获取结构变量的指针后，通过指针来访问结构变量或数组。
语法	<pre> zindex = ZINDEX_LABEL(structvarname) ZINDEX_STRUCT(structname,index).item = var var = ZINDEX_STRUCT(structname,index).item         </pre> <p>zindex: 通过 ZINDEX_LABEL 生成的索引指针  structvarname: 结构体变量名称  structname: 结构体名称  Item: 结构体成员</p>
适用控制器	<p>结构体指针功能只有特殊固件版本支持：  <b>5 系列控制器 20180327 以上固件支持。</b>  <b>4 系列控制器 fast 版本 20190107 以上固件支持。</b></p>
例子	<pre> GLOBAL Structure ClassAA      '结构体声明     DIM AA_val1               '成员变量     DIM AA_array(10)          '成员数组 END Structure  GLOBAL Class1 AS ClassAA      '结构体全局变量定义  GLOBAL gStructureAdd Class1.AA_array(0,1,2,3)      '结构体数组赋值 ?Class1.AA_array(0)           '结果: 1  gStructureAdd = ZINDEX_LABEL(Class1)      '建立结构体索引指针 ?ZINDEX_STRUCT(ClassAA,gStructureAdd).AA_array(0)      '结果: 1  ZINDEX_STRUCT(ClassAA,gStructureAdd).AA_array(0)= 10 ?ZINDEX_STRUCT(ClassAA,gStructureAdd).AA_array(0)      '结果: 10 END         </pre>
相关指令	<a href="#">ZINDEX_LABEL</a>



## 第九章 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。

### 9.1. 任务启停指令

#### RUN -- 启动文件任务

类型	任务指令
描述	<p>新建一个任务来执行控制器上的一个文件。</p> <p>重复启动同一任务会报错。</p> <p>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。</p> <p>多任务操作指令有：</p> <ul style="list-style-type: none"><li>END：当前任务正常结束</li><li>STOP：停止指定文件</li><li>STOPTASK：停止指定任务</li><li>HALT：停止所有任务</li><li>RUN：启动文件执行</li><li>RUNTASK：启动任务在一个 SUB 上执行</li></ul>
语法	<p>RUN "filename"[, tasknum]</p> <p>filename：程序文件名，bas 文件可不加扩展名</p> <p>tasknum：任务号，缺省查找第一个有效的</p>
适用控制器	通用
例子	<b>RUN "aaa", 1</b> '启动任务 1 运行 aaa.bas 文件
相关指令	<a href="#">RUNTASK</a>

#### RUNTASK -- 启动 SUB 任务

类型	任务指令
描述	<p>把一个 SUB 过程或是标号作为一个新的任务执行。</p> <p>重复启动同一任务会报错。</p>
语法	<p>RUNTASK tasknum, label</p> <p>tasknum：任务号</p> <p>label：自定义 SUB 过程（不能带参数）或标号</p>
适用控制器	通用

例子	<b>RUNTASK</b> 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 <b>WHILE</b> 1 <b>PRINT</b> *mpos <b>DELAY</b> (1000) <b>WEND</b> <b>END</b>
相关指令	<a href="#">RUN</a>

## END -- 结束

类型	任务指令
描述	<b>当前任务结束。</b> 当一个文件中有主程序和 <b>SUB</b> 过程时，一定要在主程序结束后写 <b>END</b> ，如果不写 <b>END</b> ，程序执行完主程序后，就会依次再执行 <b>SUB</b> 子程序，直到子程序的 <b>END SUB</b> 才停止。
适用控制器	通用
相关指令	<a href="#">RUN</a> , <a href="#">RUNTASK</a>

## STOP -- 停止文件任务

类型	任务指令
描述	<b>程序强制停止，操作文件。</b> 再启动任务前都要停止任务。 使用 <b>STOP</b> 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 <b>STOPTASK</b> 指令。
语法	<b>STOP</b> 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	<b>RUN</b> aaa, 1    '执行 aaa.bas <b>STOP</b> aaa, 1    '停止任务 1
相关指令	<a href="#">STOPTASK</a> , <a href="#">HALT</a>

## 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	<a href="#">STOP</a> , <a href="#">HALT</a>

## 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	<a href="#">STOP</a> , <a href="#">STOPTASK</a>

## 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	<a href="#">PAUSETASK</a>

## 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	<a href="#">RESUMETASK</a>

## 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	<a href="#">PAUSETASK</a>

## 9.2. 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	<a href="#">FILE3_ONRUN</a>

## 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	<a href="#">FILE3_RUN</a>

## 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	<a href="#">FILE3_LINE</a> , <a href="#">FILE3_RUN</a>

## 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	<a href="#">FILE3_RUN</a> , <a href="#">FILE3_GOTO</a>

### 9.3. 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 <b>BASE</b> 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 <b>BASE_MOVE</b> 参数。 <b>MOVE</b> 、 <b>MOVEABS</b> 、 <b>MOVECIRC</b> 、 <b>MOVE_OP</b> 、 <b>MOVE_TASK</b> 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	<b>BASE_MOVE=2</b> '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 <b>MERGE(2)=ON</b> '轴 2 启动连续 <b>SPEED(2)=100</b> <b>ACCEL(2)=1000</b> <b>BASE(0,1)</b> <b>MOVE(100,100)</b> '轴 0/1 插补, 轴 2 也强制作为主轴参与, 但运动距离为 0 <b>MOVE_OP(1,1)</b> <b>BASE(1)</b> <b>MOVE(100)</b> '1 轴运动 100, 也使用轴 2 为主轴 <b>BASE_MOVE=-1</b> '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 <b>PROC_STATUS</b> (0)      '打印任务 0 状态 在线命令输入 >>>PRINT <b>PROC_STATUS</b> (0) 输出: 1
相关指令	<a href="#">PROC</a>

## 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 <a href="#">AXIS</a>
适用控制器	通用
例子	例一 完整写法 PRINT <b>PROC_STATUS PROC</b> (1)      '打印任务 1 运行状态  例二 简写 PRINT <b>PROC_STATUS</b> (1)      '打印任务 1 运行状态
相关指令	<a href="#">PROCNUMBER</a>

## 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 <b>PROC</b> 来修正。
语法	VAR1 = <b>PROCNUMBER</b>
适用控制器	通用
例子	PRINT <b>PROCNUMBER</b> '打印当前任务号
相关指令	<a href="#">PROC</a>

## 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	<a href="#">PROC</a>

## PROC\_PROGRESS -- 任务指令进度

类型	任务状态
描述	任务指令进度，FILE 指令使用，范围 0-100。 每个任务都有。 FILE 指令 LOAD_ZAR 时，可以通过 HMI 界面显示进度。 注意：FILE 指令执行时，只有 HMI 任务能同时扫描到，FILE 指令不要直接由 HMI 任务来驱动。
语法	VAR1 = PROC_LINE(tasknum) tasknum: 任务号
适用控制器	通用
相关指令	<a href="#">PROC</a>

## PROC\_PRIORITY -- 任务优先级

类型	任务状态
描述	任务优先级，范围 1-10，10 最高，默认值为 1，建议只改一个任务。 每个任务都有。
语法	VAR1 = PROC_PRIORITY(tasknum) 或 PROC_PRIORITY(tasknum)=value tasknum: 任务号
适用控制器	通用
例子	PROC_PRIORITY(5)=3     '任务 5 优先级为 3 PROC_PRIORITY =2
相关指令	<a href="#">PROC</a>

## 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	<a href="#">PROC</a> , <a href="#">ERROR_SET</a>

## 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	<a href="#">ERROR_LINE</a>

## 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	<a href="#">TIME_TICKUS</a>

## 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	<b>TIME_TICKUS=0</b> DELAY(1)                '延时 1 毫秒 ?TIME_TICKUS        '打印结果：1000，单位微秒
相关指令	<a href="#">TICKS</a>

## 第十章 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 > 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	=	按位非	Not
负号	-	不等于	<>	按位与	And
乘	*	小于	<	按位或	Or 或
除	/	大于	>	按位异或	Xor
整除	\	小于等于	<=	按位同或	Eqv
求余	Mod 或 %	大于等于	>=		
加	+				
减	-				
左移位	<<				
右移位	>>				

### 10.1. 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	<p>expression1 &lt;&lt; expression2</p> <p>expression1: 任意有效的表达式</p> <p>expression2: 任意有效的表达式</p> <p>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。</p>
适用控制器	通用
例子	<p>例一 直接操作数值</p> <p>在线命令输入</p> <p>&gt;&gt;PRINT 8&lt;&lt;1 '二进制左移一位</p> <p>输出: 16</p> <p>在线命令输入</p> <p>&gt;&gt;PRINT 8&lt;&lt;2 '二进制左移两位</p> <p>输出: 32</p> <p>例二 操作变量、寄存器</p> <p>DIM bb</p> <p>bb=8</p> <p>MODBUS_REG(0)=8</p> <p>PRINT bb&lt;&lt;1,bb&lt;&lt;2</p> <p>PRINT MODBUS_REG(0)&lt;&lt;1,MODBUS_REG(0)&lt;&lt;2</p> <p>例三 优先级比较</p> <p>&gt;&gt;PRINT 8&lt;&lt;1+1 '此时二进制左移 2 位</p> <p>输出: 32</p> <p>&gt;&gt;PRINT (8&lt;&lt;1)+1 '此时二进制左移 1 位</p> <p>输出: 17</p>

## >> -- 右移位

类型	运算符
描述	右移位。

语法	<p>expression1 &gt;&gt; expression2</p> <p>expression1: 任意有效的表达式</p> <p>expression2: 任意有效的表达式</p> <p>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。</p>
适用控制器	通用
例子	<p>例一 直接操作数值</p> <p>在线命令输入</p> <p>&gt;&gt;PRINT 8&gt;&gt;1 '二进制右移一位</p> <p>输出: 4</p> <p>在线命令输入</p> <p>&gt;&gt;PRINT 8&gt;&gt;2 '二进制右移两位</p> <p>输出: 2</p> <p>例二 操作变量、寄存器</p> <p>DIM bb</p> <p>bb=8</p> <p>MODBUS_REG(0)=8</p> <p>PRINT bb&gt;&gt;1,bb&gt;&gt;2</p> <p>PRINT MODBUS_REG(0)&gt;&gt;1,MODBUS_REG(0)&gt;&gt;2</p> <p>例三 优先级比较</p> <p>&gt;&gt;PRINT 8&gt;&gt;1+1 '此时二进制右移 2 位</p> <p>输出: 2</p> <p>&gt;&gt;PRINT (8&gt;&gt;1)+1 '此时二进制右移 1 位</p> <p>输出: 5</p>

## MOD -- 求余数

类型	运算符
描述	求余数。
语法	<p>expression1 MOD expression2</p> <p>expression1: 任意有效的表达式，取整数部分。</p> <p>expression2: 任意有效的表达式，取整数部分。</p>
适用控制器	通用
例子	<p>在线命令输入</p> <p>&gt;&gt;PRINT 10 MOD (1+2)</p> <p>输出: 1</p>

## 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

## 10.2. 比较运算指令

### = -- 比较/赋值运算

类型	运算符
描述	<b>比较运算符</b> : 如果表达式 1 等于表达式 2, 返回 TRUE, 否则返回 FALSE <b>赋值运算符</b> : 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 '变量 aaa 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	<p>ON MODBUS_BIT(0)&lt;&gt;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</p> <p>label1:</p> <p>PRINT 11      '打印 11</p>

## > -- 大于

类型	运算符
描述	如果表达式 1 大于表达式 2，返回 <b>TRUE</b> ，否则返回 <b>FALSE</b> 。
语法	<p>expression1 &gt; expression2</p> <p>expression1: 任意有效的表达式</p> <p>expression2: 任意有效的表达式</p>
适用控制器	通用
例子	<p>WAIT UNTIL MPOS&gt;100</p> <p>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。</p> <p>例一</p> <p>DIM q      '定义变量</p> <p>q= 2&gt;1      '2 大于 1，所以返回 true</p> <p>PRINT q      '打印返回值</p> <p>例二</p> <p>DIM a      '定义变量</p> <p>a=0      '变量赋值</p> <p>REPEAT      '循环执行</p> <p>    a=a+1      '加一</p> <p>    ?a      '打印</p> <p>    DELAY(200)      '延时</p> <p>UNTIL a&gt;10      '条件判断，直到 a 里面的值大于 10 时，停止循环执行</p>

## >= -- 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 <b>TRUE</b> ，否则返回 <b>FALSE</b> 。
语法	<p>expression1 &gt;= expression2</p> <p>expression1: 任意有效的表达式</p> <p>expression2: 任意有效的表达式</p>



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)。		

## 10.3. 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	结果
		0	0
		0	1
		1	0
		1	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		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	<p>在线命令输入          &gt;&gt;PRINT 1 XOR 1          输出: 0</p> <p>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</p>		

## 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		结果
	0	0	1
	0	1	0
	1	0	0
	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		

## 10.4. 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

## 10.5. 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= <b>SQR</b> (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= <b>LN</b> (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= <b>LOG</b> (100) PRINT a      '结果是 2

## 10.6. 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。 对 VR 寄存器可设置的位范围是 0-24。
语法	命令语法: SET_BIT(bit#, vr#) <b>直接操作 VR 寄存器</b> bit#: 位编号: 0-24 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。

	<p>函数语法: <code>ret = SET_BIT(bit#,int)</code></p> <p>ret: 操作结果</p> <p>bit#: 位编号: 0-24</p> <p>int: 要操作的表达式, 取整数部分</p> <p>函数语法使用时返回操作后的结果, 操作对象的值不变。</p>
适用控制器	通用
例子	<p>例一 命令语法</p> <p><code>VR(23)=0.333</code></p> <p><code>SET_BIT(0,23)</code>      'VR(23)的第 0 位将置为 1, 并清除小数部分</p> <p><code>?VR(23)</code>              '结果为 1</p> <p>例二 函数语法</p> <p><code>DIM a,b</code></p> <p><code>a=0.333</code></p> <p><code>b=0</code></p> <p><code>b=SET_BIT(0,a)</code>      '设置 a 的第 0 位的, 结果赋值到 b, 并清除小数</p> <p><code>PRINT a,b</code>            '打印结果 0.333,1, a 没有改变, b 为 1</p>
相关指令	<a href="#">CLEAR_BIT</a> , <a href="#">READ_BIT</a> , <a href="#">READ_BIT2</a>

## CLEAR\_BIT -- 按位置 0

类型	数学指令或函数
描述	<p>位操作, 只对整数, 修改对应位为 0。</p> <p>分为命令语法和函数语法。</p> <p>对 VR 寄存器可设置的位范围是 0-24。</p>
语法	<p>命令语法: <code>CLEAR_BIT(bit#,vr#)</code> <b>直接操作 VR 寄存器</b></p> <p>bit#: 位编号: 0-24</p> <p>vr#: 要操作的 VR 变量编号, 整数部分</p> <p>命令语法使用时没有返回值, 直接操作, 修改操作对象的值。</p> <p>函数语法: <code>ret = CLEAR_BIT(bit#,int)</code></p> <p>ret: 操作结果</p> <p>bit#: 位编号: 0-24</p> <p>int: 要操作的表达式, 取整数部分</p> <p>函数语法使用时返回操作后的结果, 操作对象的值不变。</p>
适用控制器	通用
例子	<p>例一 命令语法</p> <p><code>VR(23)=3.333</code></p> <p><code>CLEAR_BIT(0,23)</code>      'VR(23)的第 0 位将被清除(设置为 0)</p> <p><code>?VR(23)</code>              '打印结果 2, 小数被去除</p> <p>例二 函数语法</p>



	DIM a,b a=3.333 b=0 b= <b>CLEAR_BIT</b> (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	<a href="#">SET_BIT</a> , <a href="#">READ_BIT</a> , <a href="#">READ_BIT2</a>

## 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 <a href="#">READ_BIT2</a> 。 对 VR 寄存器可设置的位范围是 0-24。
语法	ret = <b>READ_BIT</b> (bit#, vr#) ret: 读取结果: 1 或 0 bit#: 位编号: 0-24 vr#: 要操作的 VR 变量编号
适用控制器	通用
例子	VR(23)=3.333 PRINT <b>READ_BIT</b> (0,23)    '读取 VR(23)的第 0 位，结果为 1
相关指令	<a href="#">SET_BIT</a> , <a href="#">CLEAR_BIT</a> , <a href="#">READ_BIT2</a>

## 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 = <b>READ_BIT2</b> (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= <b>READ_BIT2</b> (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	<a href="#">SET_BIT</a> , <a href="#">CLEAR_BIT</a> -- <a href="#">按位置 0</a> , <a href="#">READ_BIT</a>

## 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 = <b>IEEE_IN</b> (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	<p>例一 VAR = <b>IEEE_OUT</b>(VR(1),2)        '提取 VR(1)的第二个字节</p> <p>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:</p>  <table border="1"> <thead> <tr> <th>监视内容</th><th>值</th></tr> </thead> <tbody> <tr> <td>var0</td><td>66</td></tr> <tr> <td>var1</td><td>246</td></tr> <tr> <td>var2</td><td>233</td></tr> <tr> <td>var3</td><td>121</td></tr> <tr> <td>vr(2)</td><td>123.4560</td></tr> </tbody> </table>	监视内容	值	var0	66	var1	246	var2	233	var3	121	vr(2)	123.4560
监视内容	值												
var0	66												
var1	246												
var2	233												
var3	121												
vr(2)	123.4560												

## \$ -- 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

## 10.7. 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: $\geq 0$ , 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

## 10.8. 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

## 10.9. 高级运算指令

### CRC16 -- CRC 检验计算

类型	数学函数
描述	<b>CRC16 CCITT 计算。</b>
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 个数据存储存储在 TABLE CRCVALUE = <b>CRC16</b> (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	<b>对 TABLE 存储的点坐标进行平滑调整。</b>
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 $\text{半径} = (\text{速度平方}) / \text{拐弯加速度}$ 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) <b>DTSMOOTH</b> (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	<p>B_SPLINE(type, data_start, points, data_out, ratio)</p> <p>type: 类型, 目前只支持 1-B 样条</p> <p>data_start: 图形数据在 TABLE 中的起始位置</p> <p>points: 图形数据的个数</p> <p>data_out: 平滑后的图形数据在 TABLE 中起始位置</p> <p>ratio: B_SPLINE 函数的平滑比率, 平滑后的个数为 points * ratio</p> <p>新增加样条控制点的自动计算功能, 此功能配合 MOVESPLINE 样条曲线运动使用, 4 系列以上产品支持, 4 系列控制器固件版本 20170621</p> <p>B_SPLINE(type, axes, dtstartpos, dtendpos, dtlastpos, dtnexpos, dtoutcontrol1, dtoutcontrol2)</p> <p>type:</p> <ol style="list-style-type: none"> <li>1 兼容原来的功能</li> <li>11 为连续线段的第一条线段计算样条拟合的控制点.</li> <li>12 为连续线段的中间线段计算样条拟合的控制点.</li> <li>13 为连续线段的最后一条线段计算样条拟合的控制点.</li> </ol> <p>axes: 参与样条插补的轴数</p> <p>dtstartpos: 线段起点坐标位于的 table 数组索引, 多个轴连续存储不同的 table, 下同</p> <p>dtendpos: 线段终点坐标位于的 table 数组索引</p> <p>dtlastpos: 线段起点的前面点的坐标索引, 用于计算参考, 第一条线段此参数无用</p> <p>dtnexpos: 线段终点的后面点的坐标索引, 用于计算参考, 最后一天线段此参数无用</p> <p>dtoutcontrol1: 输出样条的控制点数据, 贝塞尔的第 1 个控制点(起点也作为控制点除外)</p> <p>dtoutcontrol2: 输出样条的控制点数据, 贝塞尔的第 2 个控制点</p> <p>起点、dtoutcontrol1、dtoutcontrol2、终点一起构成贝塞尔的 4 个控制点。</p>
适用控制器	通用
例子	<p>例一 type1</p> <p><b>B_SPLINE</b>(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</p> <p>例二 新增模式</p> <p><b>TABLE</b>(0,0,0,0,100,100,100,200,100) 'XY 两轴连续 4 点的坐标数据</p> <p><b>B_SPLINE</b>(11, 2, 0, 2, -1, 4, 100,200) '第 1 条线段</p>

	?TABLE(100),TABLE(101),TABLE(200),TABLE(201) <b>B_SPLINE</b> (12, 2, 2, 4, 0, 6, 100,200)      '第 2 条线段 ?TABLE(100),TABLE(101),TABLE(200),TABLE(201) <b>B_SPLINE</b> (13, 2, 4, 6, 2, 6, 100,200)      '第 3 条线段 ?TABLE(100),TABLE(101),TABLE(200),TABLE(201)
--	--

## 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	<a href="#">MCIRC_TURNABS</a>

## 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	<p>功能 2: 计算空间圆弧或直线上一定距离后的点的位置。</p> <p><b>Table 表参数按三点画圆模式填写其他两个点参数。</b></p> <p><b>填写相对坐标，返回也是相对的位置。</b></p> <p>语法: ZCUSTOM(2,tableend,tablemid,tableout,mode,vectdis)</p> <p>    tableend: 存储圆弧终点的 table 索引</p> <p>    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</p> <p>    tableout: 输出计算数据的 table 索引</p> <p>    mode: 模式</p> <table border="1"> <thead> <tr> <th>值</th><th>描述</th></tr> </thead> <tbody> <tr> <td>1</td><td>相对起点的空间圆弧弧长</td></tr> <tr> <td>2</td><td>相对终点的空间圆弧弧长</td></tr> <tr> <td>3</td><td>相对起点的直线距离，此时 tablemid 不起作用</td></tr> <tr> <td>4</td><td>相对终点的直线距离，此时 tablemid 不起作用</td></tr> <tr> <td>5</td><td>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</td></tr> </tbody> </table> <p>    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</p>	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
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												

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

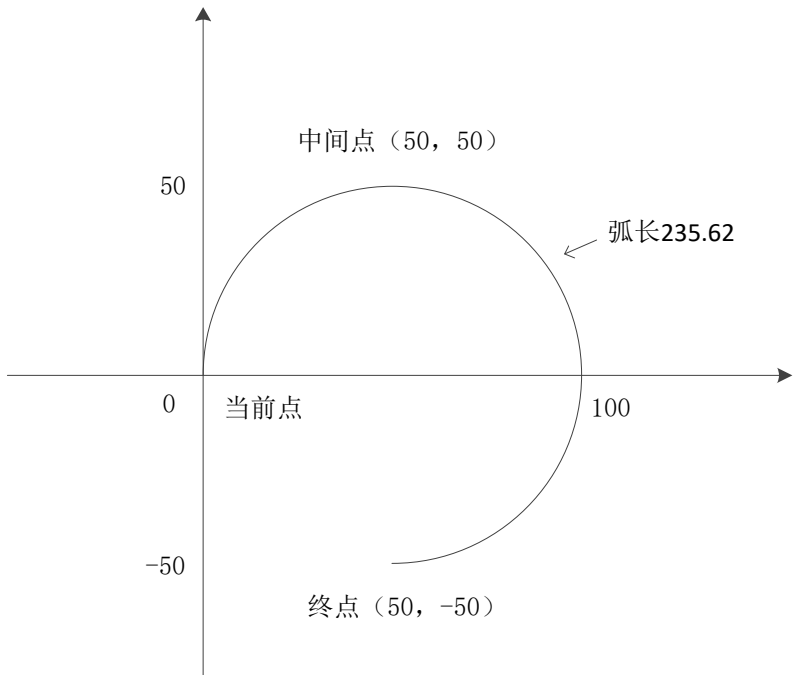
base\_in: 激励开始前, 跟随轴运动距离占 base\_dist 的百分比

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

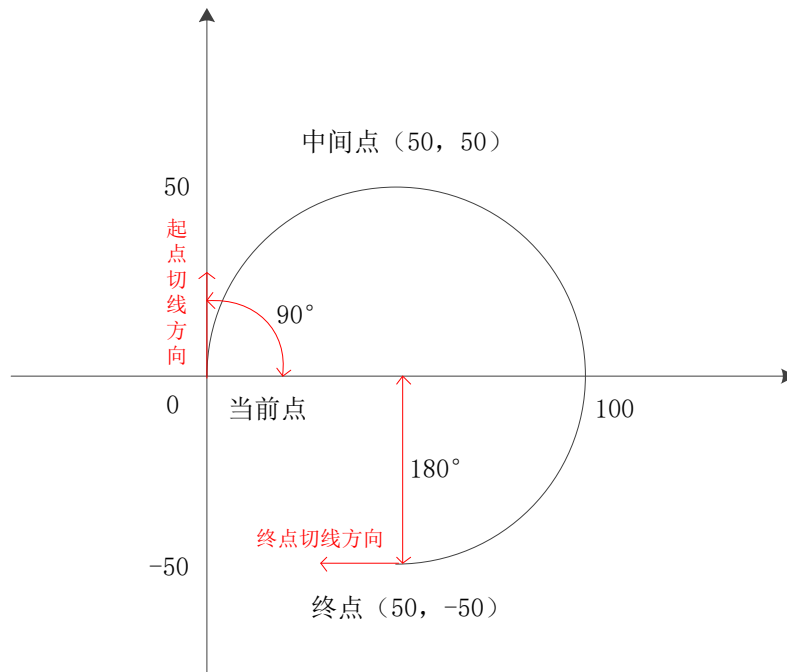
excite\_dec: 激励过程中, 跟随轴减速阶段运动距离占 excite\_dist 的百分比, excite\_dist 为负值时, 为加速阶段

distancemoved: 从轴已经运动脉冲数

	<p>tableout: 输出 Table 索引, 输出对应主轴的位置, 如多个解返回第一个</p> <p>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。</p> <p>语法: ZCUSTOM (10, dtzero, dtx, dty, dtout)</p> <p>dtzero: 工件零点在原来坐标系上的位置</p> <p>dtx: 工件坐标系 X 轴上的点在原来坐标系上的位置</p> <p>dty: 工件坐标系 Y 轴上的点在原来坐标系上的位置</p> <p>dtout: 输出 TABLE 索引, 分别存储: X, Y, Z, RX, RY, RZ</p>
适用控制器	通用
例子	<p>例一 功能 2 使用</p>  <p>TABLE(0,50,-50,0) '设置终点坐标, 相对位置</p> <p>TABLE(3,50,50,0) '设置中间点坐标, 相对位置</p> <p>'模式 1, 相对起点</p> <p><b>ZCUSTOM(2,0,3,10,1,78.54)</b> '相对起点位置顺时针弧长为 78.54 的圆上点</p> <p>?TABLE(10),TABLE(11),TABLE(12) '输出相对坐标为 0,0,0</p> <p><b>ZCUSTOM(2,0,3,10,1,-78.54)</b> '相对起点位置逆时针弧长为 78.54 的圆上点</p> <p>?TABLE(10),TABLE(11),TABLE(12) '输出相对坐标为 50,-50,0</p> <p>'模式 2, 相对终点</p> <p><b>ZCUSTOM(2,0,3,10,2,78.54)</b> '相对终点位置顺时针弧长为 78.54 的圆上点</p> <p>?TABLE(10),TABLE(11),TABLE(12) '输出坐标为 50,-50,0</p> <p><b>ZCUSTOM(2,0,3,10,2,-78.54)</b> '相对终点位置逆时针弧长为 78.54 的圆上点</p> <p>?TABLE(10),TABLE(11),TABLE(12) '输出坐标为 100,0,0</p> <p>'模式 5, 计算此时三点画圆的圆心、弧度、弧长</p>

**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 ( $1.571=90 \times \pi / 180$ )  
 ?TABLE(12),TABLE(13) '输出终点 -3.142,0 ( $-3.142=-180 \times \pi / 180$ )

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)

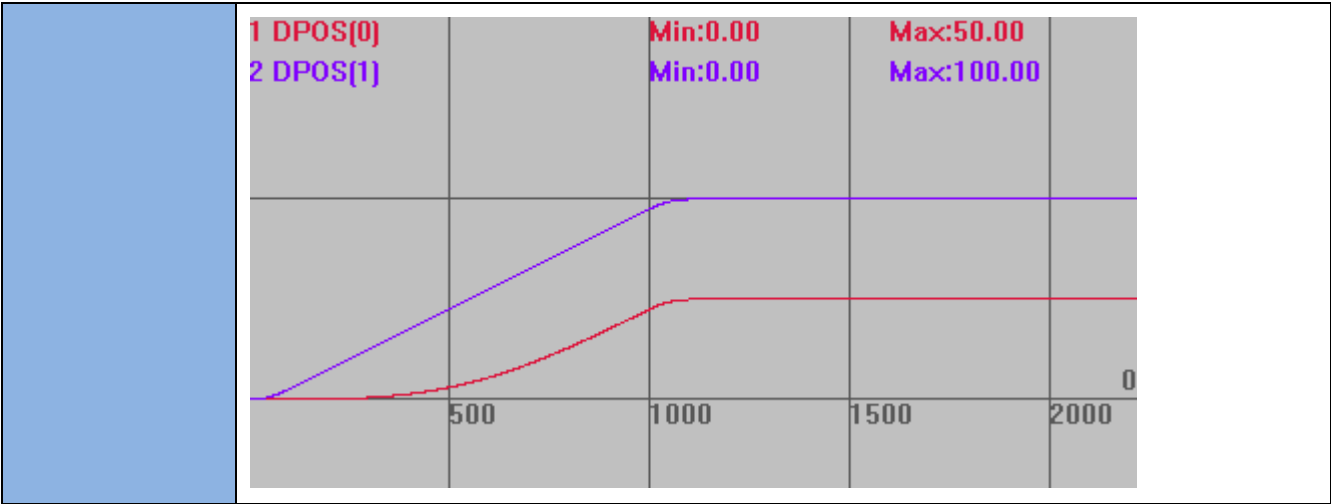
**ZCUSTOM(8,50,100,0,1,25,10)** '根据建立连接的参数，计算从轴运动 25 时主轴的位置

?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	<p>对存储 D 寄存器(MODBUS 寄存器)的 64 位数进行计算。</p> <p>一个 64 位有符号整数占用 4 个寄存器(小端模式)。</p> <p>只操作 MODBUS 寄存器，不处理 VR 映射等。</p> <p>4 系列控制器 20170629 固件以上版本支持。</p>																																																										
语法	<p>ZMATH64(opmode, dindex1, dindex2)</p> <p>opmode: 操作编号</p> <p>dindex1、dindex2: MODBUS 寄存器编号</p> <table><tr><th>操作编号</th><th>执行操作</th><th>说明</th></tr><tr><td>1</td><td>64 位整数加法</td><td>D64(dindex1) += D64(dindex2)</td></tr><tr><td>2</td><td>64 位整数减法</td><td>D64(dindex1) -= D64(dindex2)</td></tr><tr><td>3</td><td>64 位整数乘法</td><td>D64(dindex1) *= D64(dindex2)</td></tr><tr><td>4</td><td>64 位整数除法</td><td>D64(dindex1) /= D64(dindex2)</td></tr><tr><td>5</td><td>64 位整数余</td><td>D64(dindex1) %= D64(dindex2)</td></tr><tr><td></td><td></td><td></td></tr><tr><td>11</td><td>64 位整数读取</td><td>D64(dindex1) = D32(dindex2)</td></tr><tr><td>12</td><td>64 位整数转换</td><td>D32(dindex1) = D64(dindex2)</td></tr><tr><td>13</td><td>64 位整数读取</td><td>D64(dindex1) = D32IEEE(dindex2)</td></tr><tr><td>14</td><td>64 位整数转换</td><td>D32IEEE(dindex1) = D64(dindex2)</td></tr><tr><td></td><td></td><td></td></tr><tr><td>15</td><td>赋值</td><td>D64(dindex1) = double64(dindex2)</td></tr><tr><td>16</td><td>赋值</td><td>double64(dindex1) = D64(dindex2)</td></tr><tr><td>17</td><td>赋值</td><td>double64 (dindex1) = D32IEEE (dindex2)</td></tr><tr><td>18</td><td>赋值</td><td>D32IEEE (dindex1) = double64 (dindex2)</td></tr><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></table>		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)				21	double 加法	double64 (dindex1) += double64 (dindex2)	22	double 减法	double64 (dindex1) -= double64 (dindex2)
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)																																																									
21	double 加法	double64 (dindex1) += double64 (dindex2)																																																									
22	double 减法	double64 (dindex1) -= double64 (dindex2)																																																									

	23	double 乘法	double64 (dindex1) *= double64 (dindex2)
	24	double 除法	double64 (dindex1) /= double64 (dindex2)
	25	double 余，求小数	double64 (dindex1) %= double64 (dindex2)
	<p>D32IEEE 表示浮点数存储，同 MODBUS_IEEE。</p> <p>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。</p>		
适用控制器	通用		
例子	<p>MODBUS_LONG(0)=100</p> <p>MODBUS_LONG(8)=20</p> <p><b>ZMATH64</b>(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</p> <p>?MODBUS_LONG(0)                      '打印结果，100</p> <p>?MODBUS_LONG(8)                      '打印结果，120</p>		
相关指令	<u>MODBUS_IEEE</u> , <u>MODBUS_LONG</u> , <u>MODBUS_REG</u>		

## 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	<p>MODBUS_LONG(0)=100</p> <p>MODBUS_LONG(8)=200</p> <p>ZMATH64(16, 0, 8)                      '64 位赋值</p> <p>?MODBUS_DOUBLE(0)                      '打印结果，200</p> <p>?MODBUS_LONG(0)                      '打印结果，0</p> <p>?MODBUS_LONG(8)                      '打印结果，200</p>
相关指令	<u>ZMATH64</u>


# 第十一章 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。

## 11.1. 轴选择

### BASE -- 轴选择/轴组选择

类型	轴参数	
描述	<p>选择要设置参数、参与运动的轴。</p> <p>缺省值依次为：0，1，2...</p> <p>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。</p> <p>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。</p> <p>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。</p> <p>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。</p>	
语法	<p>BASE(axis&lt;,second axis&gt;&lt;,third axis&gt;...)</p> <p>axis: 第一个轴</p> <p>second axis: 下一个轴</p> <p>...</p> <p>参数最多为控制器支持的轴数，参照对应控制器硬件手册。</p>	
适用控制器	通用	
例子	<p>例一</p> <div><div>BASE(0,1,2,3)</div><div>SPEED=100,10,20,30</div><div>MOVE(100,100,100,100)</div></div> <div><div>'轴列表选择为：0,1,2,3</div><div>'此时轴 0,1,2,3 设置了对应速度，但是插补运动时，只主轴轴 0 的速度 100 起作用</div><div>'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</div></div> <p>例二</p>	



	<b>BASE(1)</b>	'轴列表选择为: 1
	<b>MOVE(100,100,100)</b>	'轴 1,2,3 做插补运动
	例三	
	<b>BASE(0,2,5)</b>	'轴列表选择为: 0,2,5
	<b>MOVE(100,100,100)</b>	'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	<p>例一</p> <p>BASE(0) MOVE(1000) <b>AXIS(1)</b> '此时强制轴 1 运动 1000units MOVE(100) '轴 0 运动 100units</p> <p>例二</p> <p>BASE(1) UNITS <b>AXIS(0)</b>=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</p>
相关指令	<a href="#">BASE</a>

## 11.2. 基本参数指令

### UNITS -- 脉冲当量

类型	轴参数
描述	<p>脉冲当量，指定每单位发送的脉冲数，支持 5 位小数精度。</p> <p>脉冲当量一般设为电机走 1mm 或 1°的所需脉冲数。</p> <p>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。</p> <p>比如：UNITS=10 对应 DPOS=3000，MPOS=3000 修改 UNITS=100 后 对应 DPOS=300，MPOS=300</p>
语法	可读：VAR1 = UNITS      或 VR1=UNITS(轴号)

	可写: $UNITS = \text{expression}$ 或 $UNITS(\text{轴号}) = \text{expression}$
适用控制器	通用
例子	<p><b>如何设置</b></p> <p>假设电机 <math>U=3600</math> 脉冲转一圈, 丝杠一圈螺距 <math>P=2\text{mm}</math></p> <p>电机转 <math>1^\circ</math> 对应的 <math>UNITS</math>:</p> <p><math>UNITS=U/360=3600/360=10</math>, 此时 <math>MOVE(1)</math>, 电机转 <math>1^\circ</math></p> <p>工作台走 <math>1\text{mm}</math> 对应的 <math>UNITS</math>:</p> <p><math>UNITS=U/P=3600/2=1800</math>, 此时 <math>MOVE(1)</math>, 工作台走 <math>1\text{mm}</math></p> <p>机台存在减速比时, 要把减速比算上, 假设减速比 <math>i=2:1</math></p> <p><math>UNITS=U*i/P=3600*2/2=3600</math></p> <p><b>编程使用</b></p> <p><math>BASE(0,1,2)</math>                      '选择轴 0, 轴 1, 轴 2</p> <p><math>UNITS=10,100,1000,30</math>      '轴 0 设为 10, 轴 1 设为 100, 轴 2 设为 1000, 轴 3 设为 30</p> <p>与 <math>BASE</math> 联合使用时, 数据多出 <math>BASE</math> 选择的轴数, 依次往后设置轴。</p> <p>数据少于 <math>BASE</math> 选择的轴数, 就只设置对应的轴。</p> <p><math>UNITS(2)=100</math>                      '直接设置轴 2, 与 <math>BASE</math> 选择轴无关</p>

## ATYPE -- 轴类型

类型	轴参数																				
描述	<p><b>轴功能类型设置, 只能设置为轴具备的特性</b> (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)。</p> <p>最好是在程序初始化的时候就设置好 <math>ATYPE</math>。</p> <p><math>ZCAN</math> 扩展轴要先设置 <math>AXIS\_ADDRESS</math>, 并在设置后延迟 2 个 tick 再调用运动指令, <b>受总线带宽限制, <math>ZCAN</math> 扩展轴不要设置超过 2 个。</b></p> <p>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 <math>ZMC206</math> 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。</p>																				
语法	<p><math>VAR1 = ATYPE, ATYPE = \text{expression}</math></p> <table border="1"> <thead> <tr> <th>ATYPE 类型</th><th>描述</th></tr> </thead> <tbody> <tr> <td>0</td><td>虚拟轴</td></tr> <tr> <td>1</td><td>脉冲方向方式的步进或伺服</td></tr> <tr> <td>2</td><td>模拟信号控制方式的伺服</td></tr> <tr> <td>3</td><td>正交编码器</td></tr> <tr> <td>4</td><td>脉冲方向输出+正交编码器输入</td></tr> <tr> <td>5</td><td>脉冲方向输出+脉冲方向编码器输入</td></tr> <tr> <td>6</td><td>脉冲方向方式的编码器</td></tr> <tr> <td>7</td><td>脉冲方向方式步进或伺服+EZ 信号输入</td></tr> <tr> <td>8</td><td><math>ZCAN</math> 扩展脉冲方向方式步进或伺服</td></tr> </tbody> </table>	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
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
	20	振镜类型，带振镜状态反馈 振镜连接不上 AXISSTATUS 的 bit2 会置位，ENCODER 返回原始的发送位置，脉冲单位 ZMC408SCAN 支持
	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
	22	振镜轴类型，带振镜位置反馈 振镜连接不上 AXISSTATUS 的 bit2 会置位，振镜报警 AXISSTATUS 的 bit3 会置位 MPOS 返回反馈位置，做了反矫正处理，ENCODER 返回原始的反馈位置脉冲单位 ZMC408SCAN 支持
	24	远程编码器轴类型 ZHD500X 上手轮使用，需要控制器 5 系列 20180404 以上固件版本支持
	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，并设置速度限制
	65	ECAT 周期位置模式，需支持 EtherCAT
	66	ECAT 周期速度模式，需支持 EtherCAT Profile 要设置为 20 或以上
	67	ECAT 周期力矩模式，需支持 EtherCAT PROFILE 要设置为 30 或以上
	70	ECAT 自定义操作，只读取编码器，需支持 EtherCAT
	电机模式 INVERT_STEP 指令设置，默认脉冲方向	
适用控制器	通用	
例子	<p>例一 脉冲类型</p> <p>BASE(0,1)</p> <p><b>ATYPE</b> = 1,1                    '轴 0,1 设为脉冲控制类型</p> <p>UNITS=100,100                '脉冲当量设为 100</p> <p>SPEED=100,100                '速度 100 units/s</p> <p>ACCEL=1000,1000              '加速度 1000 units/s/s</p> <p>DECEL=1000                    '减速度 1000 units/s/s</p> <p>MOVE(100,100)                '直线插补</p> <p>例二 EtherCAT 总线控制</p> <p>SLOT_SCAN(0)                '总线扫描</p> <p>BASE(0)</p>	

	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	'单轴使能
	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)	
	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	'单轴使能
	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
	例五 远程编码器轴	
	BASE(axisnum)	
	AXIS_ADDRESS = lcd 编号	
	ATYPE=24	
相关指令	<a href="#">AXIS_ADDRESS</a> ， <a href="#">INVERT_STEP</a>	

AXIS\_ADDRESS -- 轴地址设置

类型	轴参数
描述	<p>扩展轴时的轴地址配置。</p> <p><b>1.ZCAN 扩展轴时</b>，扩展板上带 8 位拨码开关（V1.3 以上硬件版本）。</p> <p>受总线带宽限制，ZCAN 扩展轴不要设置超过 2 个。</p> <p>必须先设置 <b>AXIS_ADDRESS</b>，再设置 ZCAN 扩展轴 <b>ATYPE</b> 类型，修改后必须重新设置 <b>ATYPE</b>。见例一。</p>

	<table><tr><td>位 1-4</td><td>CAN 地址拨码，组合值 0-15</td></tr><tr><td>位 5-6</td><td>CAN 速度拨码，不同组合值速度不同</td></tr><tr><td>位 7</td><td>特殊功能预留</td></tr><tr><td>位 8</td><td>120 欧姆电阻拨码，拨 ON 时电阻接通</td></tr></table>	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
	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。								
	<table><tr><td>位 0-15</td><td>驱动器编号加 1，0-自动指定</td></tr><tr><td>位 16-31</td><td>SLOT 编号（多槽位时）</td></tr></table>	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）				
	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									
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>	位 0-15	映射的本地脉冲轴号	位 16-31	高 16 位全设为 1（等同于十进制下高 16 位= -1）					
位 0-15	映射的本地脉冲轴号								
位 16-31	高 16 位全设为 1（等同于十进制下高 16 位= -1）								
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									
语法	VAR1 = AXIS_ADDRESS, AXIS_ADDRESS = expression								
适用控制器	通用								
例子	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 100units/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								

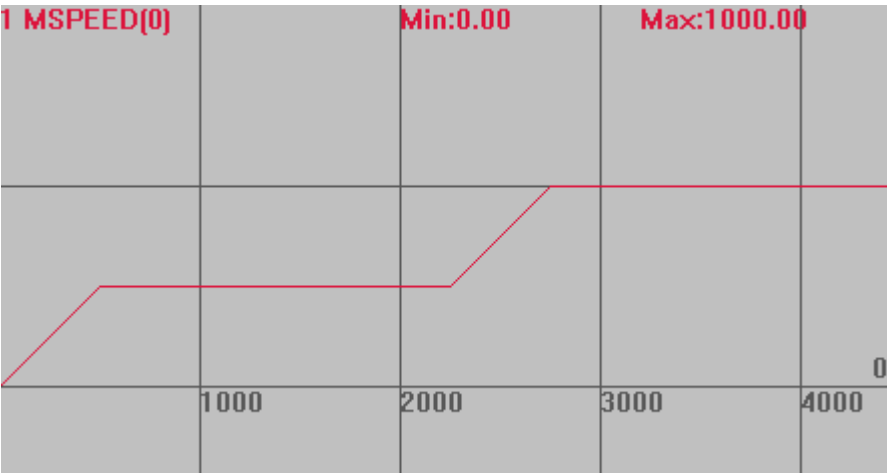
	<p>ATYPE(0)=65                    '设置为 Ecat 类型</p> <p>ATYPE(1)=65</p> <p>ATYPE(2)=65</p> <p>例三 EtherCAT 自动映射轴号</p> <p><b>AXIS_ADDRESS</b> (0)=0        '自动指定 slot0 的驱动器，按照接线顺序，从轴 0 开始依次映射轴号(不建议使用，建议例二)</p> <p>ATYPE(0)=65                    '轴 0 配置为 ECAT 模式</p> <p>例四 EtherCAT 控制器修改脉冲轴号</p> <p>'修改前，操作轴 0，对应控制器上的轴 0 接口</p> <p>BASE(16)                    '重映射的轴号</p> <p>ATYPE(16)=0</p> <p>BASE(0)                    '要修改的本地脉冲轴号</p> <p>ATYPE(0)=0                    '先将本地脉冲轴 0 设为虚拟轴</p> <p><b>AXIS_ADDRESS</b> (16)= (-1&lt;&lt;16)+0        '绑定本地脉冲轴 0，高 16 位=-1</p> <p>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</p> <p>例五 振镜轴号重映射</p> <p>ATYPE(4)=0</p> <p>ATYPE(5)=0</p> <p>BASE(X)                    '希望映射的轴号</p> <p><b>AXIS_ADDRESS</b> = (-1&lt;&lt;16)+4        '重映射第一个振镜轴</p> <p>ATYPE = 21</p> <p>BASE(Y)                    '希望映射的轴号</p> <p><b>AXIS_ADDRESS</b> = (-1&lt;&lt;16)+5        '重映射第二个振镜轴</p> <p>ATYPE = 21</p>
相关指令	<a href="#">ATYPE</a>

## AXIS\_ENABLE -- 单轴使能

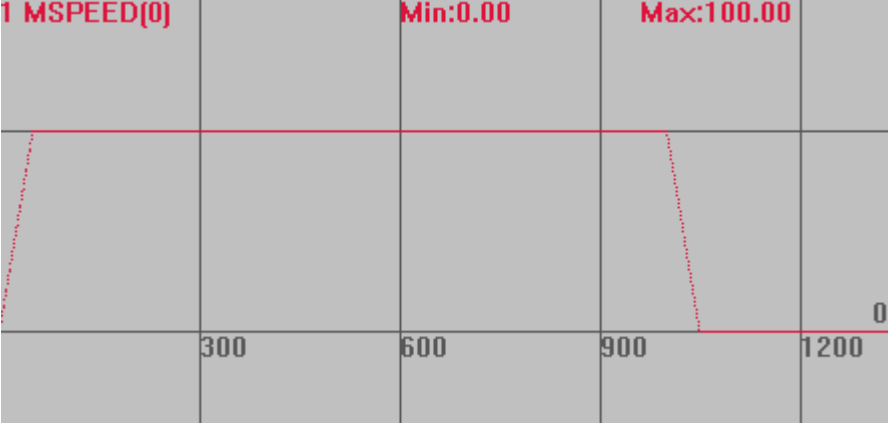
类型	轴参数
描述	<p>各轴使能设置。</p> <p>EtherCAT 总线轴需设置，还必须设置 WDOG=1 总使能。</p>
语法	AXIS_ENABLE = 1/0    1 使能，0 关闭
适用控制器	通用
例子	<b>AXIS_ENABLE</b> (0) = 1        '打开轴 0 使能
相关指令	<a href="#">WDOG</a>

## 11.3. 速度参数指令

### SPEED -- 运动速度

类型	轴参数
描述	<p>轴速度，单位为 <b>units/s</b>。</p> <p>若脉冲当量设为 1mm 脉冲数，SPEED 速度设 100，则每秒移动 100mm。</p> <p>当多轴运动时，作为插补运动的速度。</p> <p>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。</p> <p>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)。</p>
语法	VAR1 = SPEED, SPEED = expression
适用控制器	通用
例子	<p>BASE(0)</p> <p>UNITS=100        '脉冲当量</p> <p><b>SPEED</b> =500     '设置 0 轴速度 500units/s</p> <p>ACCEL=1000      '加速度 1000units/s/s</p> <p>DPOS=0          '坐标清 0</p> <p>TRIGGER         '自动触发示波器</p> <p>VMOVE(1)        '持续运动</p> <p>WAIT UNTIL DPOS(0)&gt;1000    '等待轴 0 运行到 1000</p> <p><b>SPEED</b>=1000     '速度改为 1000</p> <p>速度曲线</p> <p>MSPEED(0)垂直刻度 1000</p> 
相关指令	<a href="#">FORCE_SPEED</a> , <a href="#">SPEED_RATIO</a>

ACCEL -- 加速度

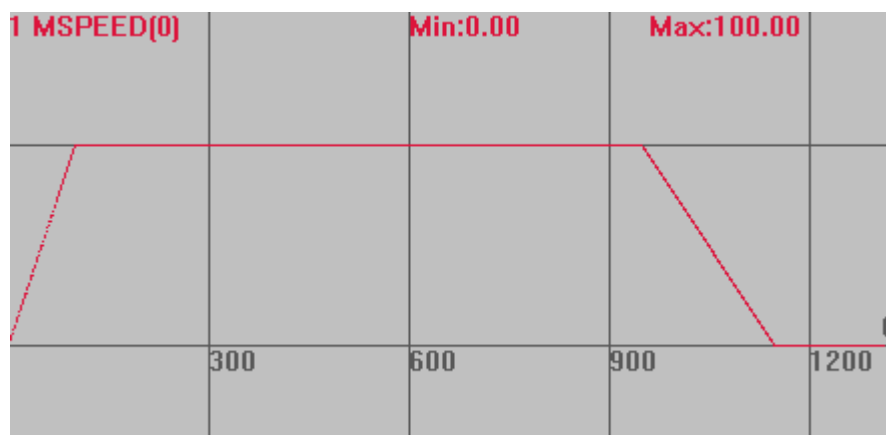
类型	轴参数
描述	<p>轴加速度，单位为 <b>units/s/s</b>。</p> <p>SPEED 速度设 100，加速度设 1000，加速到 100 时间为 100/1000 秒。</p> <p>当多轴运动时，插补运动的加速度为主轴加速度。</p> <p>建议运动前设置好加速度和减速度，运动中不要修改，运动中调整会导致速度曲线变化。</p>
语法	<p>可读：VAR1 = ACCEL(轴号)</p> <p>可写 ACCEL(轴号) = expression</p>
适用控制器	通用
例子	<p>例一</p> <p>BASE(1,2,3,4)     'BASE 选择轴</p> <p>ACCEL=100, 100, 100, 100     '设置轴 1，2，3，4 的加速度</p> <p>ACCEL(2)=200     '设置轴 2 的加速度</p> <p>例二</p> <p>BASE(0)</p> <p>UNITS=100     '脉冲当量</p> <p>DPOS=0     '坐标清 0</p> <p>ACCEL=2000</p> <p>DECEL=2000</p> <p>SPEED=100</p> <p>MOVE(100)</p> <p>加速过程的速度曲线</p> <p>MSPEED(0)垂直刻度 100</p> <div></div> <p>变更加减速之后的速度曲线</p> <p>ACCEL=500</p> <p>DECEL=500</p>



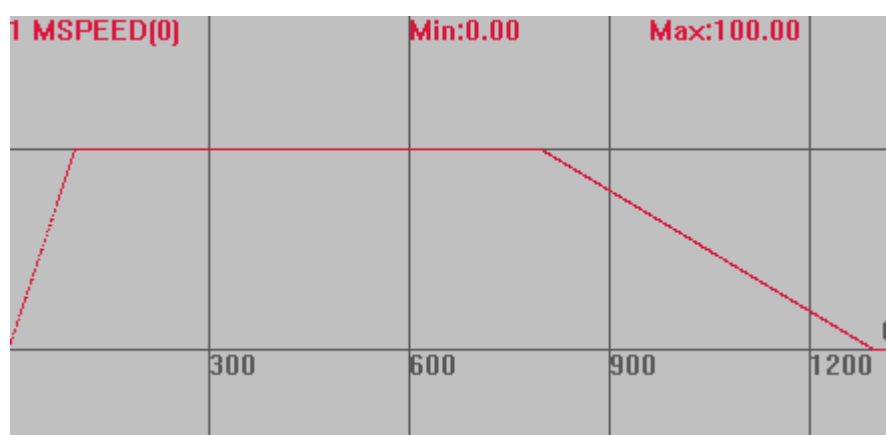


DECEL -- 减速度

类型	轴参数
描述	<p>轴减速度，单位为 <b>units/s/s</b>。</p> <p>当多轴运动时，作为插补运动的减速度。</p> <p>当设置为 0 时，自动等于加速度 <b>ACCEL</b> 值，进行对称的加减速。</p> <p>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。</p>
语法	VAR1 = DECEL, DECEL = expression
适用控制器	通用
例子	<p>例一</p> <p>BASE(1,2,3,4)</p> <p><b>DECEL</b> =100, 100, 100, 100     '设置轴 1，2，3，4 的减速度</p> <p><b>DECEL</b> (0)=200                 '设置轴 0 的减速度</p> <p>PRINT DECEL (0)</p> <p>例二</p> <p>BASE(0)                         '选择轴 0</p> <p>UNITS=100                     '脉冲当量</p> <p>DPOS=0                         '坐标清 0</p> <p>SPEED=100                     '设置速度为 100 units/s</p> <p>ACCEL=1000</p> <p><b>DECEL</b>=500                     '减速度为 500units/s/s</p> <p>TRIGGER                         '自动触发示波器</p> <p>MOVE(200)                     '移动 200units</p> <p>速度曲线</p> <p>MSPEED(0)垂直刻度 100</p>



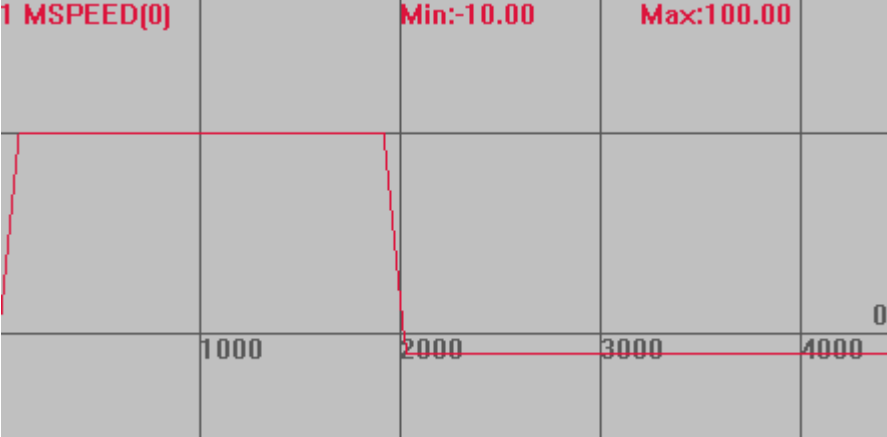
### DECEL=200 时的速度曲线



相关指令	ACCEL, FASTDEC
------	----------------

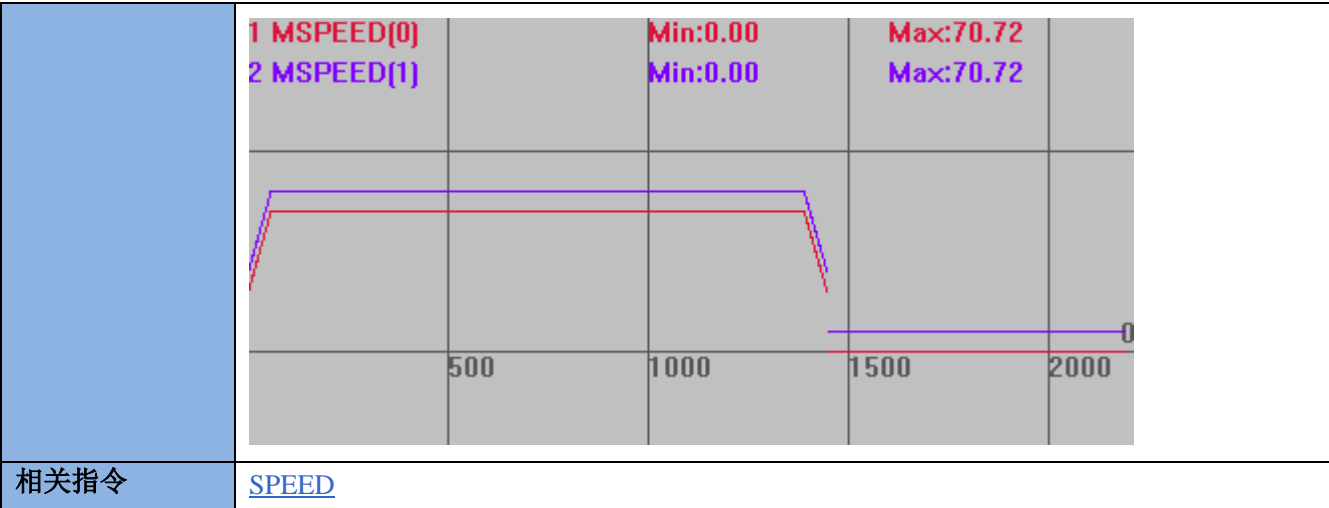
## CREEP -- 爬行速度

类型	轴参数
描述	轴回零时爬行速度，用于原点搜寻，单位为 <b>units/s</b> 。
语法	VAR1 = CREEP, CREEP = expression
适用控制器	通用
例子	<p>BASE(0)  DPOS=0  UNITS=100  ACCEL=1000  DECEL=1000  SPEED = 100            '运行速度设置为 100units/s  <b>CREEP</b> = 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</p>

	<div>DATUM(3) '先按速度 100 反找回零，遇原点后再按速度 10 反向离开原点</div> <div>速度曲线</div> <div>MSPEED(0)垂直刻度 100</div> <div></div>
相关指令	<a href="#">DATUM</a>

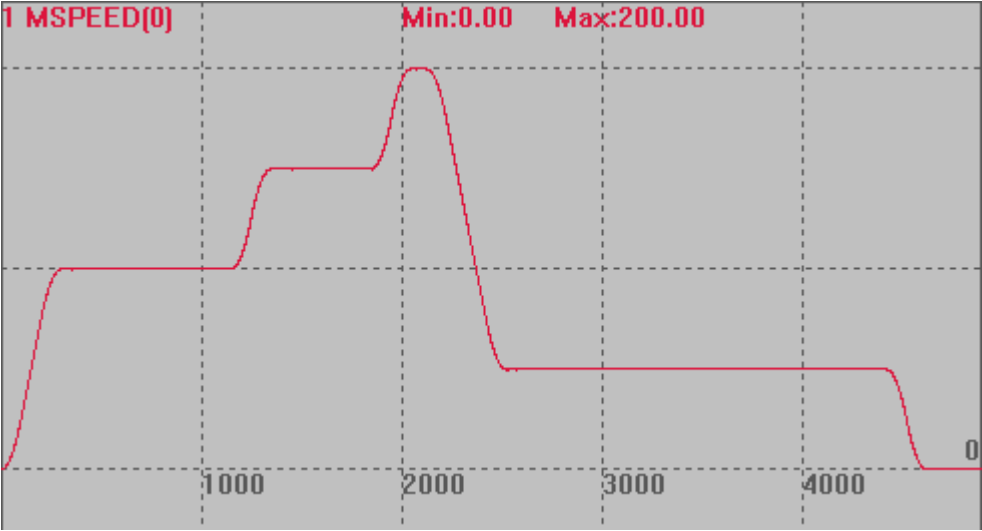
LSPEED -- 起始速度

类型	轴参数
描述	<div>轴起始速度，同时用于停止速度，缺省 0，单位为 units/s。</div> <div>当多轴运动时，作为插补运动的起始速度。</div> <div>当需要追求效率时，可以考虑设置起始速度。</div>
语法	VAR1 = LSPEED, LSPEED = expression
适用控制器	通用
例子	<div>例一</div> <div>BASE(0,1) '选择轴 0 为主轴</div> <div>DPOS=0,0</div> <div>UNITS=100,100 '脉冲当量 100</div> <div>SPEED=100,100 '主轴速度 100units/s</div> <div>ACCEL=1000,1000</div> <div>DECEL=1000,1000</div> <div>LSPEED=40 '起始速度 40units/s</div> <div>TRIGGER '自动触发示波器</div> <div>MOVE(100,100) '各轴运动距离</div> <div>速度曲线</div> <div>MSPEED(0)垂直刻度 100，无偏移</div> <div>MSPEED(1)垂直刻度 100，偏移 10</div>



FORCE\_SPEED -- SP 速度

类型	轴参数
描述	<p>自定义速度的 SP 运动的强制速度，单位为 <b>units/s</b></p> <p>这个参数被带入运动缓冲。</p> <p>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)。</p> <p>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。</p>
语法	VAR1 = FORCE_SPEED, FORCE_SPEED = expression
适用控制器	通用
例子	<p>BASE(0)</p> <p>DPOS=0</p> <p>ATYPE=1</p> <p>UNITS=100                   '脉冲当量 100</p> <p>ACCEL=500</p> <p>DECEL=500</p> <p>SPEED = 100                '速度 100units/s</p> <p><b>FORCE_SPEED=150</b>        '自定义速度 150units/s</p> <p>SRAMP=100                 'S 曲线</p> <p>MERGE=ON</p> <p>TRIGGER                    '自动触发示波器</p> <p>MOVE(100)                 '不带 SP 使用 SPEED 速度</p> <p>MOVESP(100)               '速度为 150</p> <p><b>FORCE_SPEED=200</b></p> <p>MOVESP(100)               '速度为 200</p> <p><b>FORCE_SPEED=50</b></p> <p>MOVESP(100)               '速度为 50</p>

	<div>END</div> <div>速度曲线：</div> <div>MSPEED(0)垂直刻度 100</div> <div></div>
相关指令	<div>*SP SPEED -- 运动速度</div>

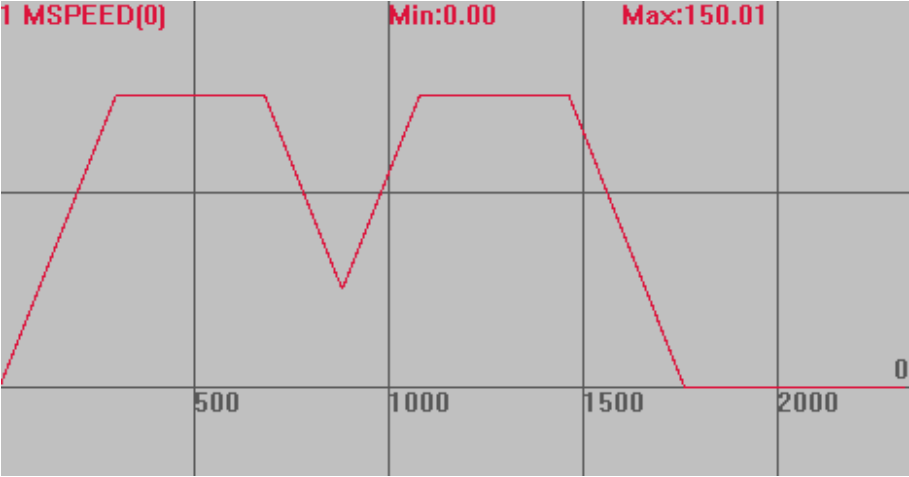
STARTMOVE\_SPEED -- SP 运动开始速度

类型	轴参数
描述	<div>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。</div> <div>只有使用了带 SP 的运动指令才生效。</div> <div>不使用时请设置较大值。控制器默认值 1000。</div>
语法	VAR1 =STARTMOVE_SPEED, STARTMOVE_SPEED = expression
适用控制器	通用
例子	<div>RAPIDSTOP(2)</div> <div>WAIT IDLE(0)</div> <div>BASE(0) '选择 XY 轴</div> <div>DPOS = 0</div> <div>MPOS = 0</div> <div>ATYPE=1 '脉冲方式步进或伺服</div> <div>UNITS = 100 '脉冲当量</div> <div>SPEED = 100</div> <div>ACCEL = 200</div> <div>DECEL = 200</div> <div>SRAMP=100 'S 曲线</div> <div>MERGE = ON '启动连续插补</div> <div>TRIGGER</div> <div>'第一段</div>

	<div><div><div><div><div>FORCE_SPEED = 30</div><div>STARTMOVE_SPEED = 1000</div><div>ENDMOVE_SPEED = 1000</div><div>MOVESP(40)</div></div><div><div>'第一段速度 30'</div><div>'不设置，默认值 1000'</div><div>'不设置，默认值 1000'</div></div></div><div><div>'第二段</div><div>FORCE_SPEED = 50</div><div>STARTMOVE_SPEED = 20</div><div>ENDMOVE_SPEED = 40</div><div>MOVESP(50)</div></div><div><div>'第二段速度 50'</div><div>'第二段的起始速度 20'</div><div>'结束速度 40'</div></div></div><div><div>'第三段</div><div>FORCE_SPEED = 60</div><div>STARTMOVE_SPEED = 1000</div><div>ENDMOVE_SPEED = 1000</div><div>MOVESP(60)</div><div>END</div></div><div><div>'第三段速度 60'</div><div>'不设置，默认值 1000'</div><div>'不设置，默认值 1000'</div></div></div> <div><div>速度和位置曲线</div><div>MSPEED(0)垂直刻度 100</div><div>DPOS(0)垂直刻度 100，偏移-100</div><div></div></div>
--	--

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。

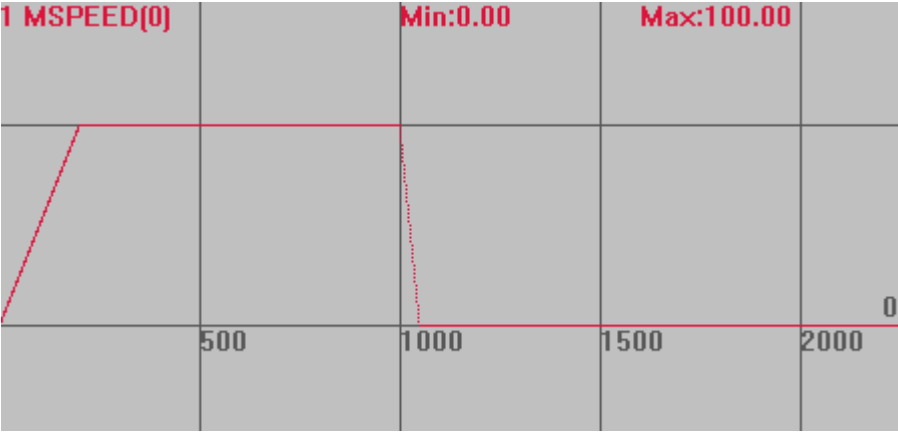
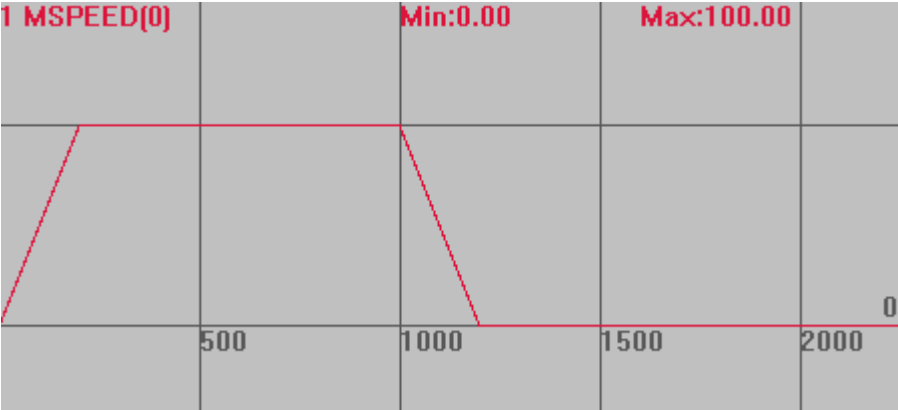
	只有使用了带 SP 的运动指令才生效。 不使用时请设置较大值。控制器默认值 1000。
语法	VAR1 = ENDMOVE_SPEED, ENDMOVE_SPEED = expression
适用控制器	通用
例子	<div>BASE(0) DPOS=0 UNITS=100 MERGE=1'开启连续插补 SPEED=100 ACCEL=500 DECEL=500 FORCE_SPEED=150'限制速度为 150units/s <b>ENDMOVE_SPEED=50</b>'强制结束速度为 50units/s TRIGGER'自动触发示波器 MOVESP(100) MOVESP(100)</div> <div>速度曲线 MSPEED(0)垂直刻度 100</div> <div></div> <div>不使用 ENDMOVE_SPEED 限制速度时</div>



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	<a href="#">DECEL</a>

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 <b>units/s</b> 。
语法	VAR1 = MSPEED
适用控制器	通用
相关指令	<a href="#">UNITS</a> ， <a href="#">VP_SPEED</a>

SPEED\_RATIO -- 速度比例

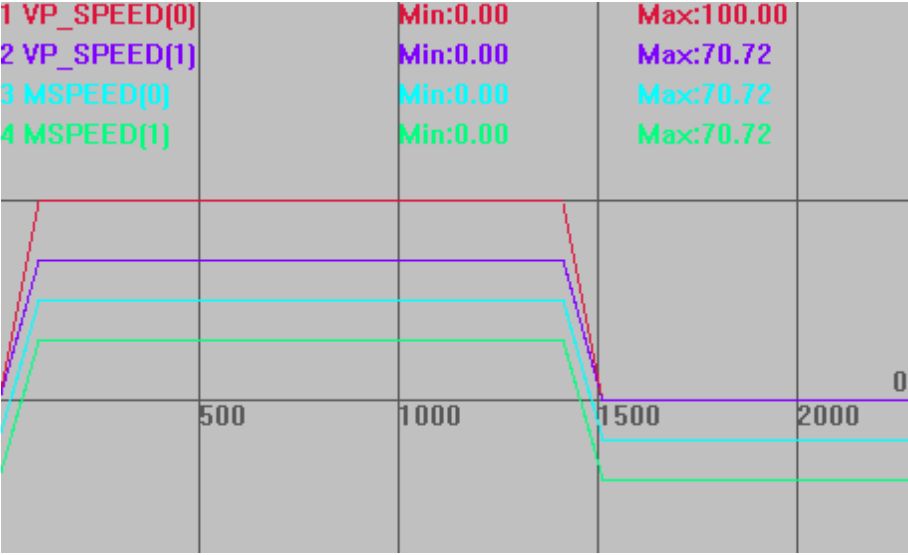
类型	轴参数
描述	轴速度比例， <b>0-1</b> 。 轴实际运动速度为 SPEED*SPEED_RATIO。 用于运动中根据加减速的平滑速度改变。
语法	SPEED_RATIO(轴号) = value

	<p>value: 比例, 取值范围 0-1</p> <p>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。</p> <p>在线命令不带轴号时, 默认对轴 0 起作用。</p>
适用控制器	最新固件支持
例子	<p>RAPIDSTOP(2)</p> <p>WAIT IDLE</p> <p><b>SPEED_RATIO</b> = 1</p> <p>TRIGGER</p> <p>BASE(0)                                '选择轴 0</p> <p>DPOS=0</p> <p>UNITS=100</p> <p>SPEED = 100</p> <p>ACCEL = 1000</p> <p>DECEL = 1000</p> <p>MERGE = ON</p> <p>SRAMP = 50</p> <p>MOVE(100)</p> <p>DELAY(500)                                '等待 0.5s</p> <p><b>SPEED_RATIO</b> = 0.5                        '速度降为 50</p> <p>WAIT UNTIL VP_SPEED &lt; 80</p> <p>DELAY(100)                                '等待 0.1s</p> <p><b>SPEED_RATIO</b> = 0.3                        '速度降为 30</p> <p>END</p> <p>速度曲线</p> <p>VP_SPEED(0)垂直刻度 100</p> 
相关指令	<a href="#">FORCE_SPEED</a> , <a href="#">SPEED</a>

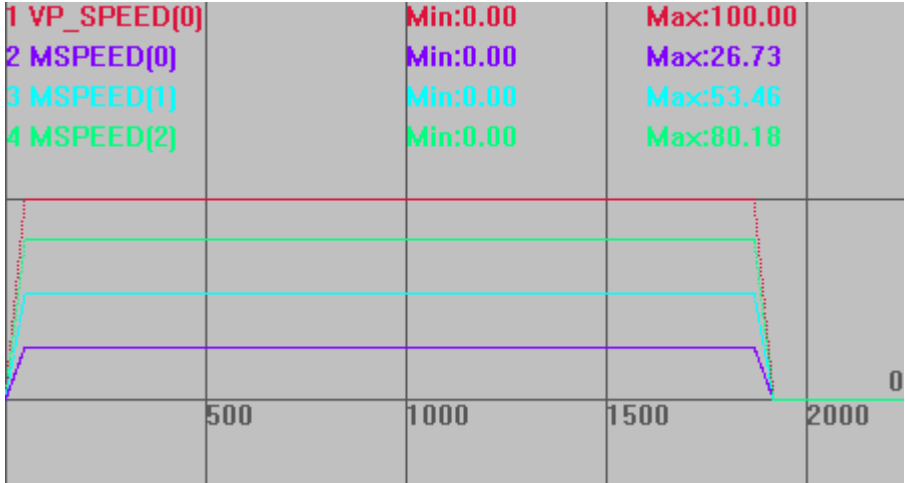
SRAMP -- 加减速曲线

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	<div>BASE(0) '选择轴 0</div> <div>DPOS=0</div> <div>UNITS=100 '脉冲当量 100</div> <div>SPEED=100 '速度 1000units/s</div> <div>ACCEL=1000 '加速度 1000units/s/s</div> <div>DECEL=1000 '加速度 1000units/s/s</div> <div>SRAMP=100 'S 曲线时间 100ms</div> <div>TRIGGER '自动触发示波器</div> <div>MOVE(100) '运动 100units</div> <div>速度曲线</div> <div>MSPEED(0)垂直刻度 100</div> <div></div> <div>SRAMP=0 时</div> <div></div>		
相关指令	<a href="#">ACCEL</a> , <a href="#">DECEL</a>		

VP\_SPEED -- 当前运动速度

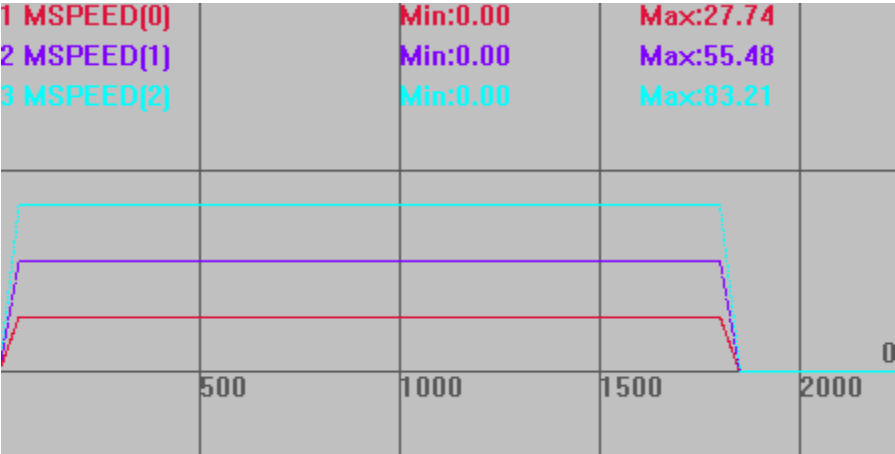
类型	轴状态
描述	<p>返回轴当前运动的速度，单位为 <b>units/s</b>。</p> <p>当多轴运动时，主轴返回的是插补运动的速度，不是主轴的分速度。</p> <p>非主轴返回的是相应的分速度，与 <b>MSPEED</b> 效果一致。</p> <p>VP_SPEED 在默认情况下是为显示多轴合成速度设计的，是没有负值的，除非把 <b>SYSTEM_ZSET</b> 的 bit0 的值设置为 0，就可以用来显示单轴的命令速度，可正可负。</p>
语法	VAR1 = VP_SPEED
适用控制器	通用
例子	<p>BASE(0,1) DPOS=0,0                '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</p> <p>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</p> 
相关指令	<a href="#">MSPEED</a> ， <a href="#">SPEED</a>

## INTERP\_FACTOR -- 插补速度计算

类型	轴参数
描述	<p>插补时轴是否参与速度计算，缺省参与计算（1）。</p> <p>此参数只对直线的任意轴和螺旋的第三个轴起作用。</p> <p>运动后请及时还原，否则会导致后续运动不正确。</p> <p>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。</p> <p>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。</p>
语法	<p>INTERP_FACTOR = 0/1</p> <p>0-不参与计算，1-参与计算</p>
适用控制器	通用
例子	<p>例一：全部参与速度计算</p> <p>BASE(0,1,2) '选择轴 0 为主轴</p> <p>DPOS=0,0,0</p> <p>ATYPE=1,1,1</p> <p>UNITS=100,100,100 '脉冲当量 100</p> <p>SPEED=100,100,100 '主轴速度 100units/s</p> <p>ACCEL=1000,1000,1000</p> <p>DECEL=1000,1000,1000</p> <p><b>INTERP_FACTOR=1,1,1</b> '3 轴都参与速度计算</p> <p>TRIGGER '自动触发示波器</p> <p>MOVE(100,200,300) '各轴分运动距离</p> <p>此时根据合成运动速度 100 可以计算出各轴分速度，如下图。</p> <p>VP_SPEED(0)垂直刻度 100</p> <p>MSPEED(0)垂直刻度 100</p> <p>MSPEED(1)垂直刻度 100</p> <p>MSPEED(2)垂直刻度 100</p> 

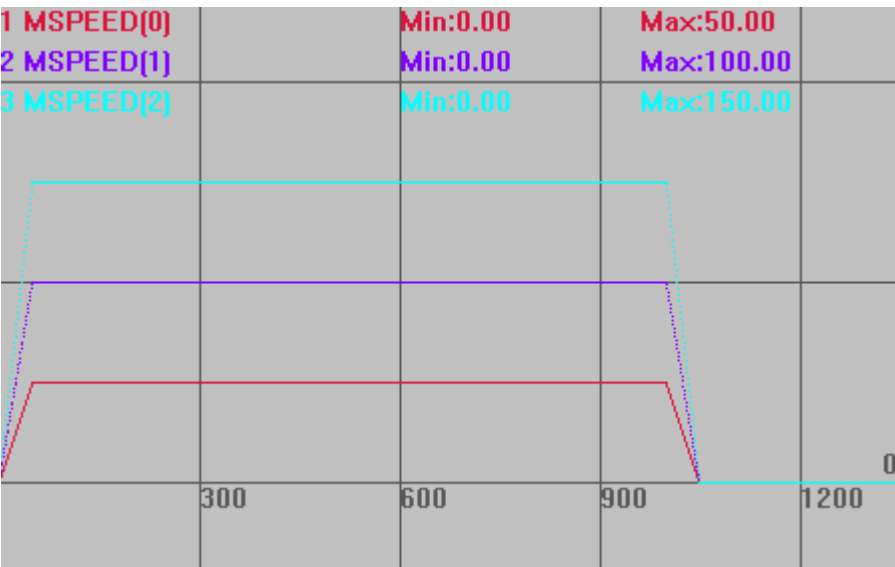
例二：部分轴不参与速度计算  
**INTERP\_FACTOR=0,1,1** '轴 0 不参与速度计算

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](#)

## 11.4. 轴状态查看指令

### MTYPE -- 当前运动类型

类型	轴状态	
描述	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。	
语法	VAR1 = MTYPE	
	MTYPE	运动指令类型
	0	IDLE (没有运动)
	1	MOVE
	2	MOVEABS
	3	MHELICAL
	4	MOVECIRC
	5	MOVEMODIFY
	6	MOVESP
	7	MOVEABSSP
	8	MOVECIRCSP
	9	MHELICALSP
	10	FORWARD, VMOVE(1)
	11	REVERSE, VMOVE(-1)
	12	DATUMING
	13	CAM
	14	FWD_JOG
	15	REV_JOG
	20	CAMBOX
	21	CONNECT
	22	MOVELINK
	23	CONNPATH
	25	MOVESLINK
	26	MOVESPIRAL
	27	MECLIPSE, MECLIPSEABS, MECLIPSESP, MECLIPSEABSSP
	28	MOVE_OP/MOVE_OP2, MOVE_TABLE, MOVE_TASK MOVE_PARA, MOVE_PWM, MOVE_ASYNMOVE, MOVE_AOUT
	29	MOVE_DELAY, MOVE_WAIT, MOVE_SYNMOVE
	31	MSPHERICAL, MSPHERICALSP
	32	MOVE_PT
	33	CONNFRAME
	34	CONNREFRAME

适用控制器	通用
例子	<pre> 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 </pre>
相关指令	<a href="#">NTYPE</a> , <a href="#">REMAIN BUFFER</a>

## 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	<pre> 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 </pre>
相关指令	<a href="#">MTYPE</a>

## 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
	19	精准输出缓冲溢出	524288	80000h
	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	<p>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</p> <p>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</p> <p>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。</p>			
相关指令	<a href="#">AXIS_STOPPREASON</a>			

## IDLE -- 运动状态

类型	轴状态
描述	<p>轴运动状态判断，只能判断是运动中还是停止。</p> <p>运动中，返回 0，运动结束，返回-1。</p> <p>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。</p>

语法	VAR1 = IDLE
适用控制器	通用
例子	<p>例一</p> <pre>IF IDLE(0) THEN      '如果轴 0 停止     BASE(1)     MOVE(100) ENDIF</pre> <p>例二</p> <pre>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</pre>
相关指令	<a href="#">LOADED</a> , <a href="#">WAIT IDLE</a>

## ADDAX\_AXIS -- 叠加轴号

类型	轴状态
描述	<b>ADDAX</b> 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	<pre>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</pre>
相关指令	<a href="#">ADDAX</a>

## 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	<p>写 0 清除, 按位锁存, 意义与 AXISSTATUS 相同。</p> <p>20150731 以上版本支持。</p>
适用控制器	通用
例子	<pre>IF AXIS_STOPREASON AND (512+1024) THEN     PRINT "axis el stoped" ENDIF</pre>
相关指令	<a href="#">AXISSTATUS</a>

## 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	<a href="#">CAMBOX</a> ， <a href="#">MOVELINK</a> ， <a href="#">CONNECT</a>	

## 11.5. 运动前瞻指令

### CORNER\_MODE -- 拐角设置

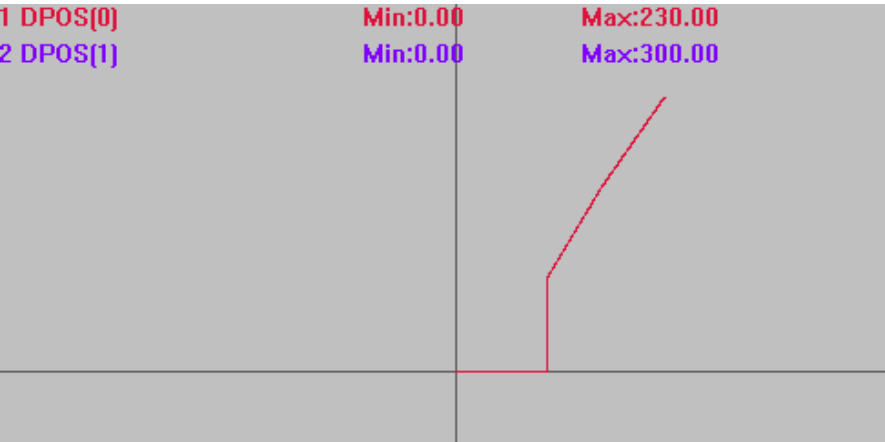
类型	轴参数																						
描述	拐角减速等模式设置。																						
语法	CORNER_MODE = mode mode: 不同的位代表不同的意义，位可以同时使用。 <table border="1"> <thead> <tr> <th>位</th><th>值</th><th>描述</th></tr> </thead> <tbody> <tr> <td>0</td><td>1</td><td>预留</td></tr> <tr> <td>1</td><td>2</td><td> <b>自动拐角减速</b>            按 ACCEL,DECEL 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  <b>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</b> </td></tr> <tr> <td>2</td><td>4</td><td>预留</td></tr> <tr> <td>3</td><td>8</td><td> <b>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</b>            此参数在 MOVE 函数调用前修改生效  <b>限制速度按 FORCE_SPEED</b>  <b>限速= FORCE_SPEED * 实际半径/FULL_SP_RADIUS</b>            限速半径 FULL_SP_RADIUS 设置         </td></tr> <tr> <td>4</td><td>16</td><td>预留</td></tr> <tr> <td>5</td><td>32</td><td> <b>自动倒角设置</b>            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         </td></tr> </tbody> </table>		位	值	描述	0	1	预留	1	2	<b>自动拐角减速</b> 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 <b>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</b>	2	4	预留	3	8	<b>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</b> 此参数在 MOVE 函数调用前修改生效 <b>限制速度按 FORCE_SPEED</b> <b>限速= FORCE_SPEED * 实际半径/FULL_SP_RADIUS</b> 限速半径 FULL_SP_RADIUS 设置	4	16	预留	5	32	<b>自动倒角设置</b> 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
位	值	描述																					
0	1	预留																					
1	2	<b>自动拐角减速</b> 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 <b>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</b>																					
2	4	预留																					
3	8	<b>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</b> 此参数在 MOVE 函数调用前修改生效 <b>限制速度按 FORCE_SPEED</b> <b>限速= FORCE_SPEED * 实际半径/FULL_SP_RADIUS</b> 限速半径 FULL_SP_RADIUS 设置																					
4	16	预留																					
5	32	<b>自动倒角设置</b> 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

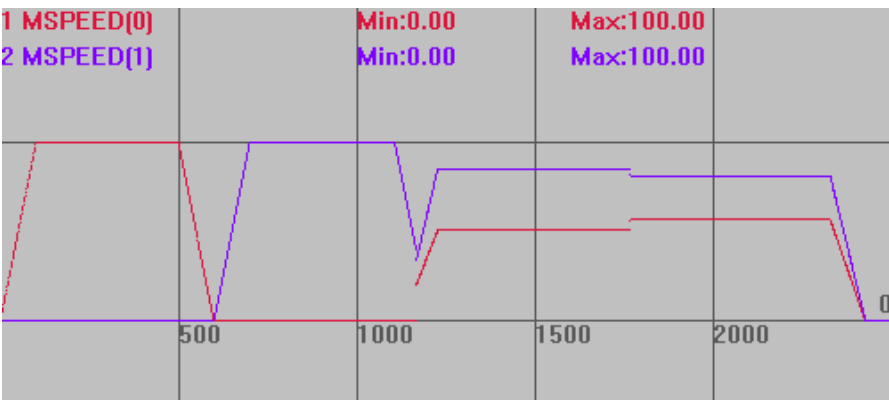
为了方便说明每个位的功能，例程都是单独位设置，实际应用时可以多位同时使用。  
比如 CONNER\_MODE=2+8，打开自动拐角减速和小圆限速

```
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

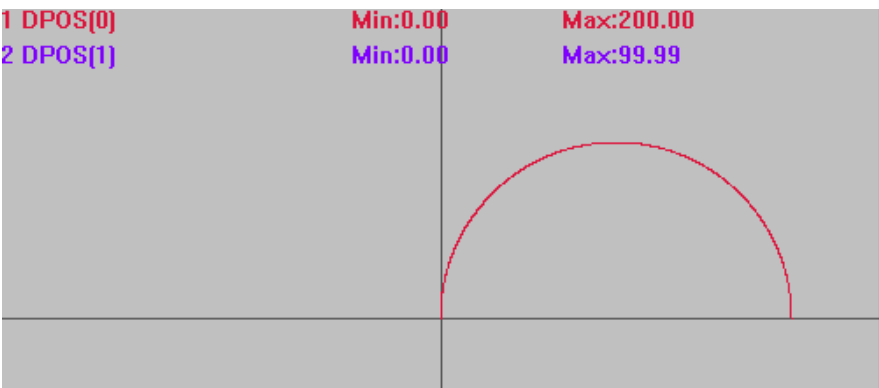
TRIGGER '自动触发示波器

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

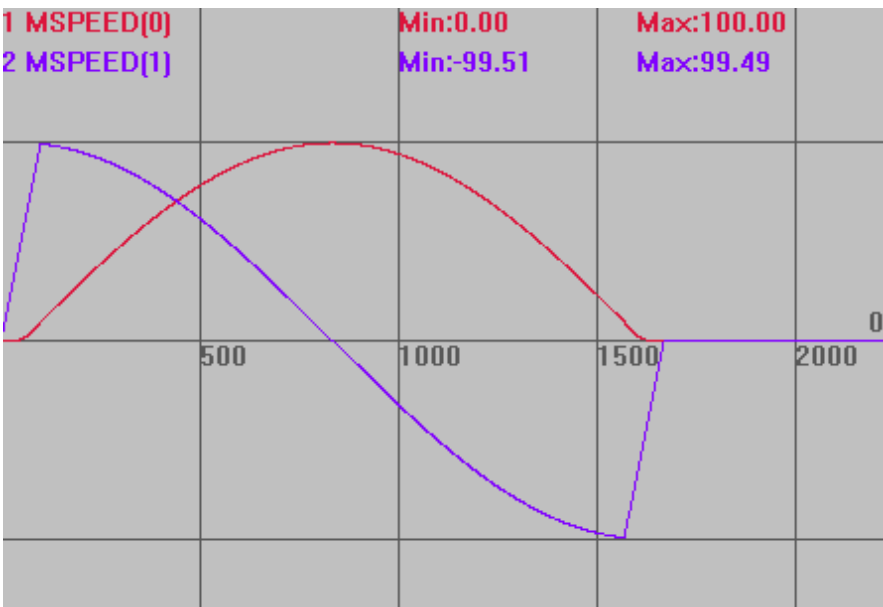
DPOS(1)垂直刻度 100



速度曲线

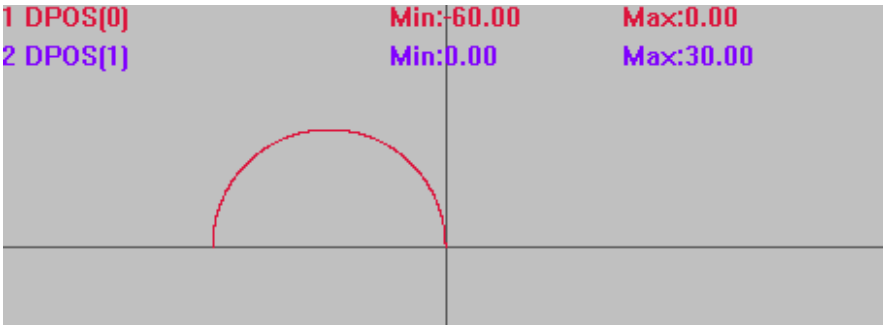
MSPEED(0)垂直刻度 100

MSPEED(1)垂直刻度 100

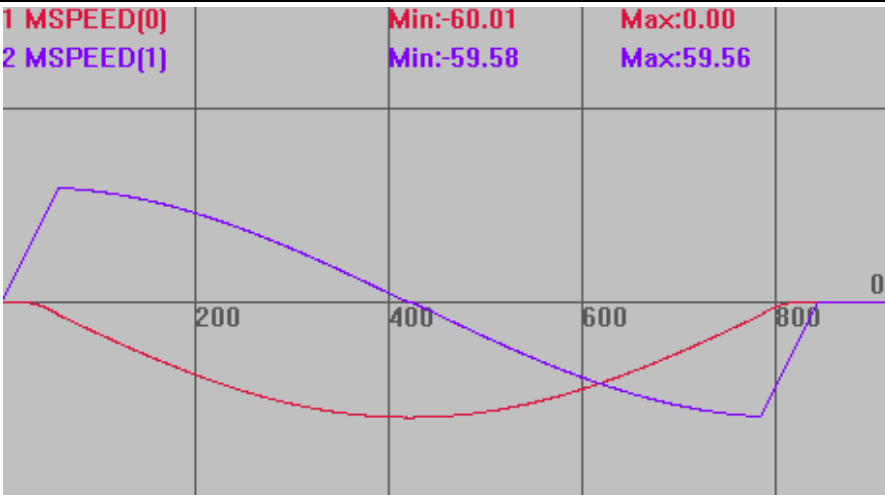


'半径小于限制值时，限速=FORCE\_SPEED \* 实际半径/FULL\_SP\_RADIUS  
MOVECIRC(-60,0,-30,0,0)      '此时速度 60=120\*30/60

轨迹曲线  
DPOS(0)垂直刻度 100  
DPOS(1)垂直刻度 100

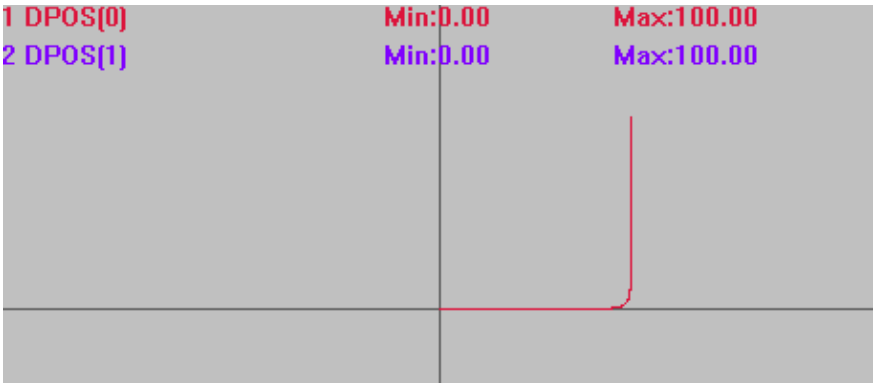


速度曲线  
MSPEED(0)垂直刻度 100  
MSPEED(1)垂直刻度 100



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

	<div><div>1 DPOS{0}</div><div>2 DPOS{1}</div></div> <div><div>Min:0.00</div><div>Min:0.00</div></div> <div><div>Max:100.00</div><div>Max:100.00</div></div>
--	---

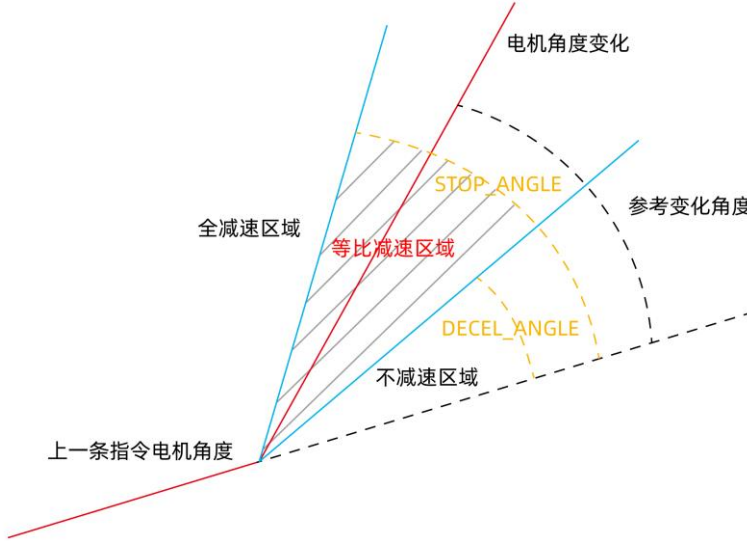
DECEL\_ANGLE -- 拐角减速开始

类型	轴参数
描述	<p>开始减速的角度，单位是弧度。</p> <p>减速拐角参考速度以 <b>FORCE_SPEED</b> 速度为参考，一定要设置合理的 <b>FORCE_SPEED</b>。</p> <p>角度转换弧度公式：角度值*(PI/180)</p> <p>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。</p> <p>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。</p> <div></div> <p>下一插补运动轨迹处于下方时取绝对值。</p> <p>直线与圆弧相连时按照圆弧的切线方向计算角度。</p> <p>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。</p>
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

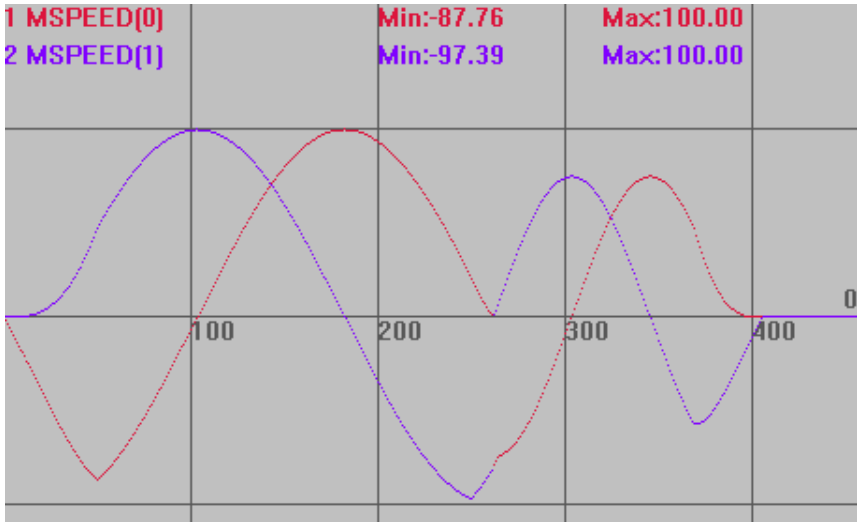


	<b>DECEL_ANGLE</b> = 25 * (PI/180) '设置开始减速拐角，弧度 <b>STOP_ANGLE</b> = 45 * (PI/180) '设置结束减速拐角，弧度 <b>FORCE_SPEED</b> = <b>SPEED</b> '一定要设置 <b>FORCE_SPEED</b>
相关指令	<a href="#">STOP_ANGLE</a> ， <a href="#">CORNER_MODE</a> -- 拐角设置

STOP\_ANGLE -- 拐角减速结束

类型	轴参数
描述	<p>停止减速到的角度，单位是弧度。</p> <p>减速拐角参考速度以 <b>FORCE_SPEED</b> 速度为参考，一定要设置合理的 <b>FORCE_SPEED</b>。</p> <p>角度转换弧度公式：角度值*(PI/180)</p> <p>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。</p> <p>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。</p>  <p>下一插补运动轨迹处于下方时取绝对值。</p> <p>直线与圆弧相连时按照圆弧的切线方向计算角度。</p> <p>与 <b>DECEL_ANGLE</b> 一起使用，当实际运动的角度处于 <b>DECEL_ANGLE</b>（上限）与 <b>STOP_ANGLE</b>（下限）时才会减速。</p>
语法	<b>VAR1</b> = <b>STOP_ANGLE</b> , <b>STOP_ANGLE</b> = expression
适用控制器	通用
例子	<p>参考 <b>CORNER_MODE</b> 例程一</p> <p><b>CORNER_MODE</b>=2</p> <p><b>DECEL_ANGLE</b> = 25 * (PI/180)</p> <p><b>STOP_ANGLE</b> = 90 * (PI/180)</p> <p><b>FORCE_SPEED</b>=<b>SPEED</b> '一定要设置 <b>FORCE_SPEED</b></p>
相关指令	<a href="#">DECEL_ANGLE</a> ， <a href="#">CORNER_MODE</a>

## FULL\_SP\_RADIUS -- 限速半径

类型	轴参数
描述	<p>小圆限速的最大圆弧半径，单位是 <b>units</b>。</p> <p>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。</p> <p><math>VP\_SPEED = FORCE\_SPEED * radius / FULL\_SP\_RADIUS</math>。</p> <p>设置自动倒角时为倒角的参考半径。</p>
语法	VAR1 = FULL_SP_RADIUS, FULL_SP_RADIUS = expression
适用控制器	通用
例子	<p>BASE(0,1)                    '选择轴 0 为主轴</p> <p>DPOS=0,0                    '坐标清 0</p> <p>UNITS=100,100</p> <p>ATYPE=1,1                    '轴类型设置</p> <p>SPEED=100,100                '主轴速度 100units/s</p> <p>ACCEL=1000,1000              '加速度 1000units/s/s</p> <p>DECEL=1000,1000              '加速度 1000units/s/s</p> <p>FORCE_SPEED=150              '自定义速度 150units/s</p> <p>CORNER_MODE=8                '开启小圆限速</p> <p><b>FULL_SP_RADIUS=8</b>            '限速半径 8</p> <p>TRIGGER                        '自动触发示波器</p> <p>MOVECIRC(10,10,0,10,1)      '圆弧半径 10，不限速，按 speed=100 运行</p> <p>MOVECIRC(4,4,0,4,1)         '圆弧半径 4，限速，速度按 <math>4/8*150=75</math> 运行</p> <p>速度曲线</p> <p>MSPEED(0)垂直刻度 100</p> <p>MSPEED(1)垂直刻度 100</p>  <p>1 MSPEED[0]                    Min:-87.76                    Max:100.00</p> <p>2 MSPEED[1]                    Min:-97.39                    Max:100.00</p>
相关指令	<a href="#">CORNER_MODE</a> <a href="#">CORNER_MODE</a> -- 拐角设置, <a href="#">FORCE_SPEED</a> , <a href="#">SPLIMIT_RADIUS</a>

## SPLIMIT\_RADIUS -- 限速值

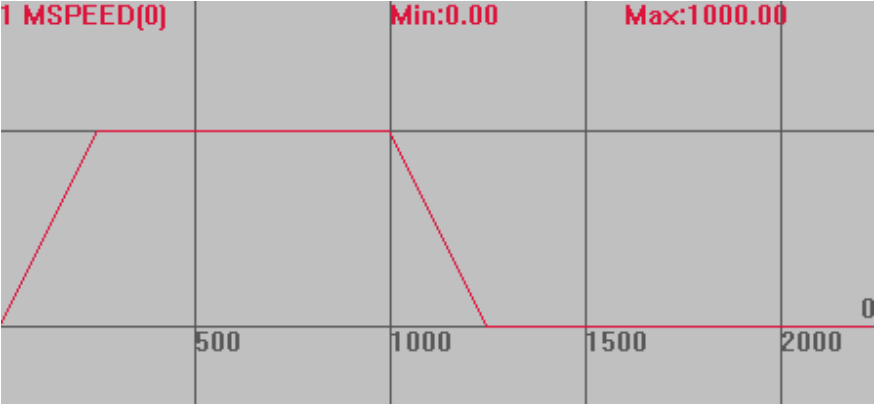
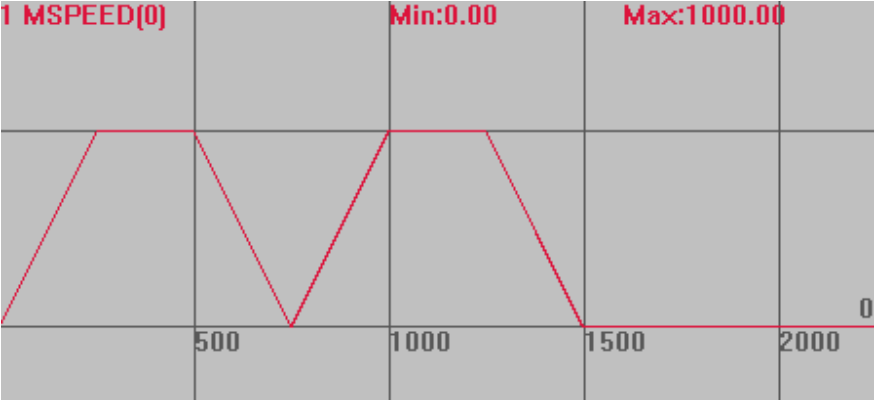
类型	轴参数
描述	小圆限速的最小速度，单位是 <b>units</b> 。 当小于 LSPEED 时，按 LSPEED 值。
语法	VAR1 = SPLIMIT_RADIUS, SPLIMIT_RADIUS = expression
适用控制器	4 系列控制器 170518 以上固件版本
相关指令	<a href="#">CORNER_MODE</a> , <a href="#">FULL SP RADIUS</a>

## ZSMOOTH -- 倒角半径

类型	轴参数
描述	参考倒角半径， <b>CORNER_MODE</b> 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 CORNER_MODE 例三
相关指令	<a href="#">CORNER_MODE</a>

## 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

	<div>ACCEL=1000</div> <div>DECEL=1000</div> <div>MERGE=ON</div> <div>TRIGGER</div> <div>MOVE(1000)</div> <div>MOVE(1000)</div> <div>速度曲线</div> <div>MSPEED(0)垂直刻度 100</div> <div></div> <div>MERGE=OFF 时</div> <div>MSPEED(0)垂直刻度 100</div> <div></div>
相关指令	<div><a href="#">*SP</a>, <a href="#">CORNER MODE</a></div>

11.6. 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 TRUE。 此指令无法判断轴是否停止，判断轴停止使用 IDLE。
语法	VAR1 = LOAED

适用控制器	通用
相关指令	<a href="#">IDLE</a> , <a href="#">WAIT LOADED</a>

## 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	<a href="#">LOADED</a> , <a href="#">LIMIT_BUFFERED</a> , <a href="#">REMAIN_BUFFER</a>

## 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) <b>MTYPE 缺省为最复杂的运动，如空间圆弧。</b>
适用控制器	通用
例子	<pre> DIM movetime          '定义变量 movetime = 0 WHILE movetime &lt; 100   '条件循环     IF REMAIN_BUFFER(1) &gt; 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) </pre>
相关指令	<a href="#">LOADED</a> , <a href="#">LIMIT_BUFFERED</a>

## 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 <b>MARK</b> 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	<b>MOVE_MARK</b> =1            '设置为标号 1 MOVE(100) <b>MOVE_MARK</b> =1            '设置为标号 1 MOVE(100) <b>MOVE_MARK</b> =2            '设置为标号 2 MOVE(200) MOVE_PAUSE (2)            'MOVE(200)前暂停
相关指令	<a href="#">MOVE_CURMARK</a>

## 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 <b>MOVE_MARK</b> 标号。
语法	VAR1 = MOVE_CURMARK
适用控制器	通用
例子	<b>MOVE_MARK</b> =1 MOVE(100) <b>MOVE_MARK</b> =2 MOVE(200) <b>MOVE_MARK</b> =3 MOVE(300) WAIT UNTIL <b>MOVE_CURMARK</b> = 2    '等待 MOVE(200)开始执行的时候，开输出口 1 OP(1,ON)
相关指令	<a href="#">MOVE_MARK</a>

## 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：

	<pre>&gt;&gt;?LIMIT_BUFFERED 4096</pre> <p>修改运动缓冲个数限制值:</p> <pre>&gt;&gt;LIMIT_BUFFERED=2000 &gt;&gt;?LIMIT_BUFFERED 2000</pre>
相关指令	<a href="#">REMAIN_BUFFER</a> , <a href="#">MOVES_BUFFERED</a>

## 11.7. 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	<b>DPOS(0) = 0</b> '轴 0 坐标偏移至 0 <b>?*DPOS</b> '命令行查看当前 DPOS，打印结果，000000000000
相关指令	<a href="#">MPOS</a> , <a href="#">ENDMOVE</a> , <a href="#">OFFPOS</a> , <a href="#">DEFPOS</a>

### 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	<b>MPOS(0) = 50</b> 'MPOS 偏移至 50 <b>?*MPOS</b> '打印结果，500000000000
相关指令	<a href="#">DPOS</a> , <a href="#">ENDMOVE</a>

### 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	<p>例一</p> <pre> 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 </pre> <p>例二</p> <p>与 OFFPOS 相对改变不同，DEFPOS 是改变为绝对位置</p> <pre> 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) </pre>
相关指令	<a href="#">DPOS OFFPOS -- 偏移位置</a>

## 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	<p>例一 相对偏移位置</p> <pre> BASE(0) MOVEABS(1000) WAIT IDLE OFFPOS = -1000      '坐标偏移 1000 </pre>



	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	<a href="#">DPOS</a> ， <a href="#">DEFPOS</a>

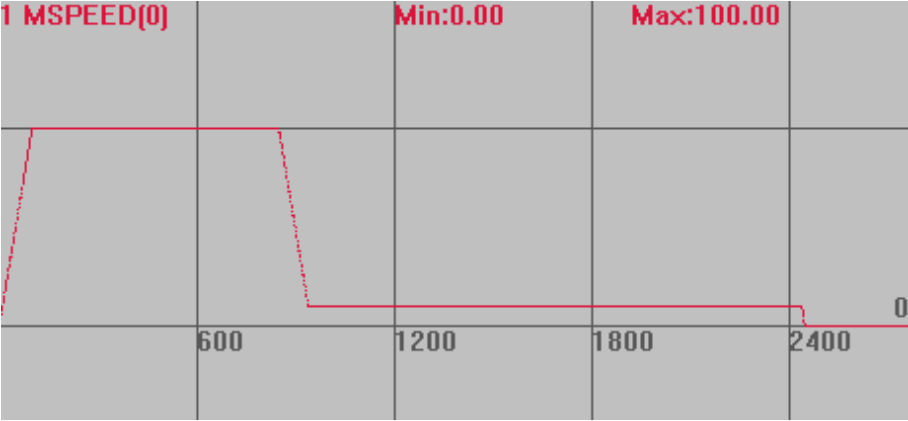
## ENDMOVE -- 当前运动目标位置

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	<a href="#">DPOS</a> ， <a href="#">MPOS</a> ， <a href="#">ENDMOVE_BUFFER</a>

## 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	<p>例一 单轴</p> <p><b>VECTOR_MOVED=0</b>        '手动清 0</p> <p>MOVE(100)</p> <p>WAIT IDLE</p> <p><b>? VECTOR_MOVED</b>        '打印出轴 0 运动距离，结果，100</p> <p>例二 多轴</p> <p>BASE(0,1)</p> <p>DPOS=0,0</p> <p><b>VECTOR_MOVED=0</b>        '轴 0 轴组合成矢量运动距离手动清 0</p> <p>BASE(2,3)</p> <p>DPOS=0,0</p> <p><b>VECTOR_MOVED=0</b>        '轴 2 轴组合成矢量运动距离手动清 0</p> <p>BASE(0,1)</p> <p>MOVE(-300,-400)</p> <p>WAIT IDLE(0)</p> <p><b>? VECTOR_MOVED</b>        '打印出轴 0 轴组合成矢量运动距离，结果，500</p> <p>MOVE(300,400)</p> <p>WAIT IDLE(0)</p> <p><b>? VECTOR_MOVED</b>        '打印出轴 0 轴组合成矢量运动距离，结果，1000</p> <p>BASE(2,3)</p> <p>MOVE(30,-40)</p> <p>WAIT IDLE(2)</p> <p><b>? VECTOR_MOVED</b>        '打印出轴 2 轴组合成矢量运动距离，结果，50</p> <p>MOVE(30,40)</p> <p>WAIT IDLE(2)</p> <p><b>? VECTOR_MOVED</b>        '打印出轴 2 轴组合成矢量运动距离，结果，100</p>
相关指令	<a href="#">ENDMOVE</a>

## 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	<p>           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 <b>REMAIN</b>&lt;20 '等待剩余距离小于 20            SPEED=10 '修改速度         </p> <p>速度曲线 MSPEED(0)垂直刻度 100</p> 
相关指令	<a href="#">VECTOR_BUFFERED</a>

## 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	<p>           BASE(0) '选择轴 0            UNITS=100 '脉冲当量 100         </p>

	<p>SPEED=100                    '速度 100units/s</p> <p>ACCEL=1000                   '加速度 1000units/s/s</p> <p>MOVE(100)                    '当前运动 100units</p> <p>MOVE(300)                    '缓冲运动 300units</p> <p>MOVE(-1000)                   '缓冲运动-1000units</p> <p>?VECTOR_BUFFERED    '返回剩余运动距离，结果，1400</p>
相关指令	<a href="#">REMAIN</a>

## ENDMOVE\_BUFFER -- 缓冲最终位置

类型	轴状态
描述	<p>轴当前和缓冲的所有运动的最终目标位置。</p> <p>可用来进行绝对和相对位置的转换，见例二。</p> <p>对 <b>VMOVE</b>，<b>DATUM</b> 等非固定距离的运动，<b>ENDMOVE_BUFFER</b> 并不准确，或时刻在变化。</p> <p>使用了循环坐标指令 <b>REP_OPTION</b> 后，<b>ENDMOVE_BUFFER</b> 根据 <b>REP_OPTION</b> 模式按 <b>REP_DIST</b> 设置值依次递减，即最小精度是 <b>REP_DIST</b>（模式 1）或 2 倍 <b>REP_DIST</b>（模式 0），见例三。</p>
语法	VAR1 = ENDMOVE_BUFFER
适用控制器	通用
例子	<p>例一</p> <pre> 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 </pre> <p>例二 相对绝对转换</p> <p>使用 <b>ENDMOVE_BUFFER</b> 与相对运动指令可实现绝对运动的功能，常用于 <b>MSPHERICAL</b> 等只有相对模式的指令。</p> <pre> 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 </pre>

	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	<a href="#">DPOS</a> , <a href="#">MPOS</a> , <a href="#">ENDMOVE</a>

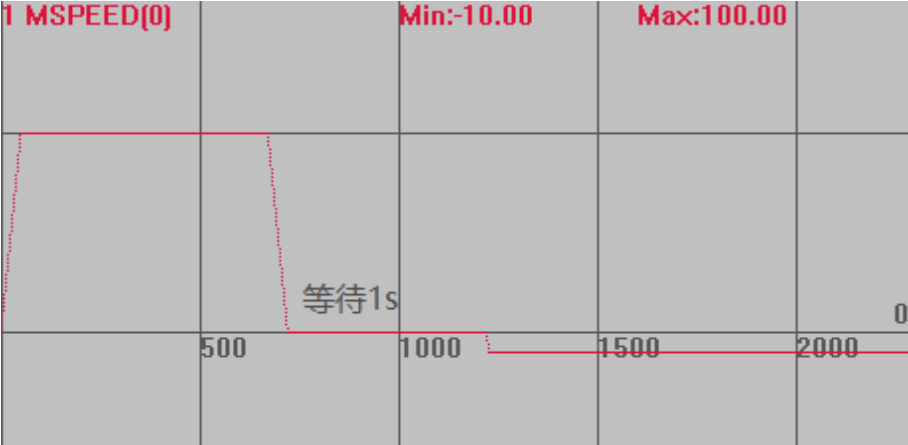
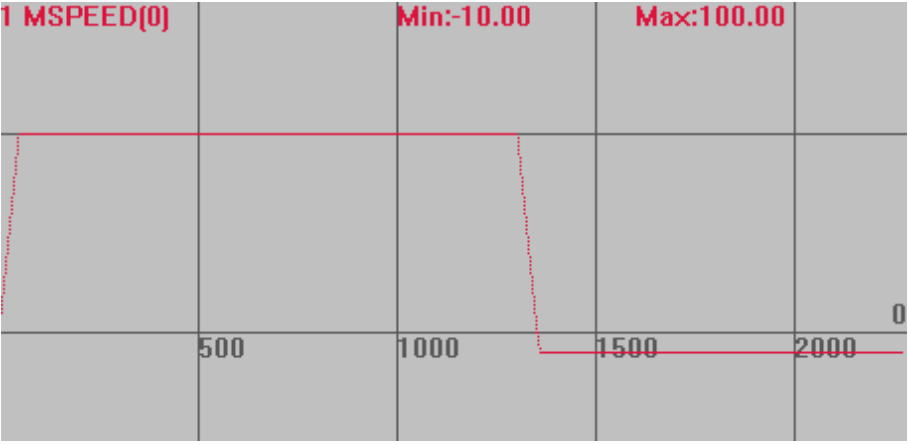
## 11.8. 原点回零指令

### 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	<a href="#">DATUM</a> , <a href="#">FWD_IN</a> , <a href="#">REV_IN</a> , <a href="#">INVERT_IN</a>

### HOMEWAIT -- 回零反找延时

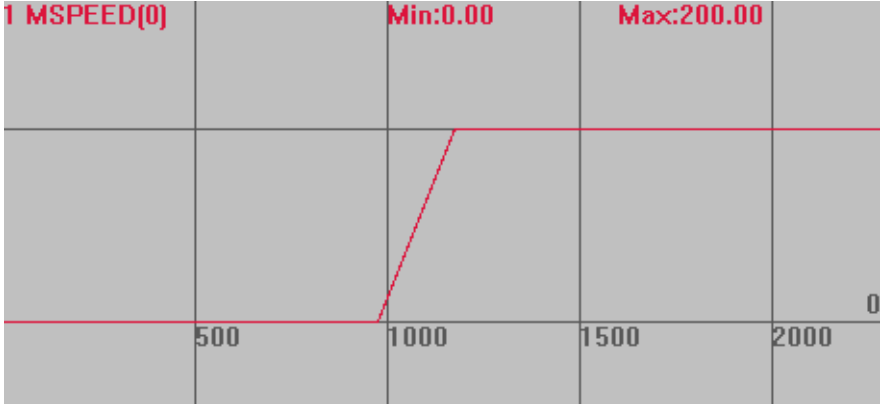
类型	轴参数
描述	此参数设置等待的时间，单位是毫秒。 对脉冲方式的伺服驱动器，回原点运动时，当反找时要等待一定时间。 控制器默认值为 2ms。

语法	VAR1 = HOMEWAIT, HOMEWAIT = expression
适用控制器	通用
例子	<div>BASE(0) '选择轴 0</div> <div>DPOS=0 '坐标清 0</div> <div>UNITS=100</div> <div>ATYPE=1</div> <div>SPEED=100 '找原点速度 100units/s</div> <div>ACCEL=1000,1000 '加速度 1000units/s/s</div> <div>DECEL=1000,1000 '加速度 1000units/s/s</div> <div>CREEP=10 '爬行速度 10units/s</div> <div>DATUM_IN=0 '原点信号设为输入 IN0</div> <div>INVERT_IN(0,ON) '反转信号</div> <div><b>HOMEWAIT=1000</b> '设置反找等待 1s</div> <div>TRIGGER '自动触发示波器</div> <div>DATUM(3) '正向找原点</div> <div><p>速度曲线</p><p>碰到 IN0 口时，会先等待 1s，然后在反向爬行</p><p>MSPEED(0)垂直刻度 100</p><p>HOMEWAIT=2 时，几乎不停止</p><p>MSPEED(0)垂直刻度 100</p></div>

相关指令	<a href="#">DATUM</a>
------	-----------------------

11.9. JOG 运动指令

FAST\_JOG -- 映射点动输入

类型	轴参数										
描述	<p>快速点动的输入的编号，-1 为无效。</p> <p>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。</p> <p>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)。</p>										
语法	VAR1 = FAST_JOG, FAST_JOG = expression										
适用控制器	通用										
例子	<p>BASE(0)                    '选择轴 0</p> <p>DPOS=0                    '坐标清 0</p> <p>UNITS=100</p> <p>ATYPE=1</p> <p>SPEED=100                '设置速度为 100 units/s</p> <p>ACCEL=500                '加速度为 500units/s/s</p> <p>JOGSPEED=200            '点动速度设为 200units/s</p> <p><b>FAST_JOG(0)=0</b>           '轴 0 的快速输入设为 IN0 口</p> <p>FWD_JOG(0)=1            '正向点动开关设为 IN1 口</p> <p>INVERT_IN(0,ON)        '反转电平</p> <p>INVERT_IN(1,ON)</p> <p>TRIGGER                   '自动触发示波器</p> <p>速度曲线</p> <p>IN0 无输入时，按下 IN1 并保持，轴速度为 JOGSPEED=200</p> <p>MSPEED(0)垂直刻度 200</p>  <table border="1"><caption>速度曲线数据表</caption><thead><tr><th>时间 (ms)</th><th>速度 (units/s)</th></tr></thead><tbody><tr><td>0</td><td>0</td></tr><tr><td>1000</td><td>0</td></tr><tr><td>1000</td><td>200</td></tr><tr><td>2000</td><td>200</td></tr></tbody></table>	时间 (ms)	速度 (units/s)	0	0	1000	0	1000	200	2000	200
时间 (ms)	速度 (units/s)										
0	0										
1000	0										
1000	200										
2000	200										





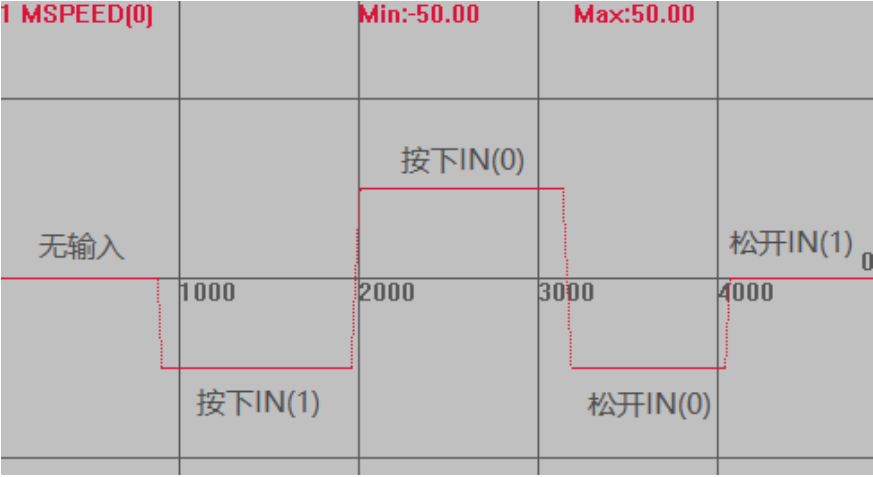
	<div> <div>1 MSPEED[0]</div> <div>2 IN[0]</div> </div> <div> <div>Min:0.00</div> <div>Min:0.00</div> </div> <div> <div>Max:100.00</div> <div>Max:1.00</div> </div>	
相关指令	<a href="#">REV_JOG</a> , <a href="#">JOGSPEED</a> , <a href="#">FAST_JOG</a>	

## REV\_JOG -- 映射负向 JOG 输入

类型	轴参数
描述	<p>负向 JOG 输入对应的输入口编号, -1 无效。</p> <p>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)。</p> <p>当有信号输入时, 对应轴按照 JOGSPEED 速度负向运动。</p> <p>当 REV_JOG 和 FWD_JOG 同时有信号输入时, 轴按照 FWD_JOG 运行。</p>
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	<a href="#">FWD_JOG</a> , <a href="#">JOGSPEED</a> , <a href="#">FAST_JOG</a>


## JOGSPEED -- JOG 速度

类型	轴参数
描述	<p><b>JOG 时的速度, 单位为 units/s。</b></p> <p>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。</p>
语法	JOGSPEED= value, VAR1=JOGSPEED
适用控制器	通用
例子	<p>例一</p> <p>BASE(0)                '选择轴 0</p> <p>DPOS=0                '坐标清 0</p> <p>UNITS=100             '脉冲当量</p> <p>SPEED =100            '主轴速度 100units/s</p> <p>ACCEL=1000            '加速度 1000units/s/s</p>

	<div>DECEL=1000        '减速度 1000units/s/s</div> <div>TRIGGER            '自动触发示波器</div> <div>JOGSPEED=50      'JOG 速度 50</div> <div>FWD_JOG=0        '输入 IN0 作为正向 JOG 开关</div> <div>REV_JOG=1         '输入 IN1 作为负向 JOG 开关</div> <div>INVERT_IN(0,ON)   '反转信号</div> <div>INVERT_IN(1,ON)</div> <div><div>输入 0 口有信号输入时，轴 0 正向运行，速度为 50。</div><div>输入 1 口有信号输入时，轴 0 负向运行，速度为 50。</div><div>同时有信号输入时，轴 0 正向运行。</div></div> <div><div>速度曲线</div><div>MSPEED(0)垂直刻度 100</div><div></div></div>
相关指令	<a href="#">REV_JOG</a> ， <a href="#">FWD_JOG</a> ， <a href="#">FAST_JOG</a>

FHOLD\_IN -- 映射保持输入

类型	轴参数
描述	<div>保持输入对应的输入点编号，-1 无效。</div> <div>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。</div> <div>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)。</div> <div>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。</div>
语法	VAR1 = FHOLD_IN, FHOLD_IN = expression
适用控制器	通用
例子	BASE(0)            '选择轴 0

	<div>DPOS=0'坐标清 0</div> <div>UNITS=100</div> <div>ATYPE=1</div> <div>ACCEL=500'加速度为 500units/s/s</div> <div>DECEL=500</div> <div>FORCE_SPEED=200'设置速度为 200 units/s</div> <div>FHSPEED=200'保持速度设为 100units/s</div> <div><b>FHOLD_IN</b>(0)=0'轴 0 的保持输入设为 IN0 口</div> <div>INVERT_IN(0,ON)'反转电平</div> <div>TRIGGER'自动触发示波器</div> <div>MOVESP(10000)'运动</div> <div>当 IN0 无输入时，轴以 FORCE_SPEED=200 的速度运动</div> <div>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</div> <div>速度曲线</div> <div>MSPEED(0)垂直刻度 200</div> <div></div>
相关指令	<a href="#">FHSPEED</a>

FHSPEED -- 保持速度

类型	轴参数
描述	轴保持速度，在 <b>FHOLD_IN</b> 被按下保持时的速度，单位为 <b>units/s</b> 。 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	<a href="#">FHOLD_IN</a> , <a href="#">SPEED</a>

## 11.10. 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(轴号)
适用控制器	通用
例子	?*ENCODER        '打印各轴编码器值，打印结果，000000000000
相关指令	<a href="#">MPOS</a> , <a href="#">ENCODER_RATIO</a>

### ENCODER\_STATUS -- 编码器状态

类型	轴状态		
描述	编码器的 <b>EA EB EZ</b> 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	?* <b>ENCODER_STATUS</b> '打印全部轴的编码器状态		
相关指令	<a href="#">ATYPE</a> , <a href="#">MPOS</a>		

### ENCODER\_FILTER -- 编码器滤波

类型	轴参数
描述	内置编码器轴滤波设置，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。 缺省 1-不滤波，0.5-2 个周期的滤波，0.25-4 个周期的滤波。 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	<a href="#">ENCODER_RATIO</a>

## 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	<a href="#">ENCODER_RATIO</a>

## 11.11. 锁存相关指令

### REGIST -- 锁存

类型	位置锁存指令						
描述	<p><b>REGIST 指令用来锁存轴的测量反馈位置。</b></p> <p>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。</p> <p>4 系列及以上控制器支持 4 锁存通道。</p> <p>锁存输入口 432：2 个，432N：4 个，412：8 个。</p> <p>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。</p> <p>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。</p> <p>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。</p> <p>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。</p> <p>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。</p>						
语法	<p>语法一</p> <p>REGIST(mode)</p> <p>mode: 锁存方式</p> <p>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。</p> <p>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。</p> <table border="1"> <thead> <tr> <th>值</th><th>描述</th></tr> </thead> <tbody> <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> </tbody> </table>	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
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

## 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
	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
	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	<p>例一 锁存脉冲轴 0 的输入信号 R0 上跳沿时的位置, 并打印。</p> <pre> 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 </pre> <p>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。</p> <pre> 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 </pre> <p>例三 锁存总线轴 0 的输入信号 R2/R3 边沿时的位置, 并打印。</p> <pre> BASE(0) REG_INPUTS = \$1000  '映射锁存通道 R3-R0 对应到输入口 1,0,0,0 REGIST(imode) IF imode = 18 OR imode = 19 THEN  '通道 R2 锁存     WAIT UNTIL MARKC  '探针 1 </pre>	

```

?"模式",Imode,"锁存位置 REG_POSC",REG_POSC
ELSEIF imode = 20 OR imode = 21 THEN           '通道 R3 锁存
    WAIT UNTIL MARKD           '等待锁存触发
    ?"模式",Imode,"锁存位置 REG_POSD",REG_POSD
ENDIF

```

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

```

例六 总线驱动器锁存，需要配置 DRIVE\_PROFILE 支持带探针的模式

```

BASE(iaxis) '选择需要锁存位置的轴号
REGIST(imode) '锁存模式
IF imode = 3 OR imode = 4 THEN
    WAIT UNTIL MARK '探针 1
    ?"模式",imode,"锁存位置 REG_POS",REG_POS

```



	<pre> DELAY(100) ELSEIF imode = 14 OR imode = 15 THEN     WAIT UNTIL MARKB          '等待锁存触发     ?"模式",imode,"锁存位置 REG_POSB",REG_POSB     DELAY(100) ELSEIF imode = 11 THEN     WAIT UNTIL MARK OR MARKB    '等待锁存触发     IF MARK THEN         ?"模式",Imode,"锁存位置 REG_POS",REG_POS         WAIT UNTIL MARKB         ?"锁存位置 REG_POSB",REG_POSB     ELSEIF MARKB THEN         ?"模式",Imode,"锁存位置 REG_POSB",REG_POSB         WAIT UNTIL MARK         ?"锁存位置 REG_POSB",REG_POS     ENDIF     DELAY(100) ENDIF </pre>
相关指令	<a href="#">MARK</a> , <a href="#">MARKB</a> , <a href="#">REG_POS</a> , <a href="#">REG_POSB</a>

## 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 都对输入口 1		
相关指令	<a href="#">REGIST</a>		

## 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 <b>MARK</b> '等待触发
相关指令	<a href="#">REG_POS</a> , <a href="#">REGIST</a>

## 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 <b>MARKB</b> '等待触发
相关指令	<a href="#">REG_POSB</a> , <a href="#">REGIST</a>

## 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，MARKC 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 <b>MARKC</b> '等待触发

相关指令	<a href="#">REG_POSC</a> , <a href="#">REGIST</a>
------	---

## 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	<a href="#">REG_POSD</a> , <a href="#">REGIST</a>

## 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	<a href="#">REGIST</a> , <a href="#">CLOSE_WIN</a>

## 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	<a href="#">REGIST</a> , <a href="#">OPEN_WIN</a>

## 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	<a href="#">REGIST</a> , <a href="#">MARK</a>	

## 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	<a href="#">REGIST</a> , <a href="#">MARKB</a>	

## 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	<a href="#">REGIST</a> , <a href="#">MARKC</a>	

## 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	<a href="#">REGIST</a> , <a href="#">MARKD</a>

## 11.12. 限位参数指令

### FS\_LIMIT -- 正向软限位设置

类型	轴参数
描述	<p>正向软限位位置，单位是 <b>units</b>。</p> <p>当 FS_LIMIT 大于 REP_DIST 时，参数不起作用，正向软限位被禁止。</p> <p>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。</p> <p>FS_LIMIT 的值默认为 200000000。</p> <p><b>软限位无法作为 DATUM 回零时的限位信号参考。</b></p>
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 <b>FS_LIMIT=200</b> '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。 <b>FS_LIMIT=2000000</b> '取消正向软限位设置
相关指令	<a href="#">RS_LIMIT</a> , <a href="#">FWD_IN</a> , <a href="#">REV_IN</a>

### RS\_LIMIT -- 负向软限位设置

类型	轴参数
描述	<p>负向软限位位置，单位是 <b>units</b>。</p> <p>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。</p>

	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 <b>DATUM</b> 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	<b>RS_LIMIT</b> = -50      '设置负向软限位 50units <b>RS_LIMIT</b> = - 2000000      '取消负向软限位设置
相关指令	<a href="#">FS_LIMIT</a> , <a href="#">FWD_IN</a> , <a href="#">REV_IN</a>

## 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 <b>FASTDEC</b> 。 输入 <b>OFF</b> 时，认为有信号输入，要相反效果可以用 <b>INVERT_IN</b> 反转电平( <b>ECI</b> 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 <b>FWD_IN</b> =6,7,8,9      '分别设置正向限位开关 <b>INVERT_IN</b> (6,ON)      '反转信号 <b>INVERT_IN</b> (7,ON) <b>INVERT_IN</b> (8,ON) <b>INVERT_IN</b> (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	<a href="#">REV_IN</a> ., <a href="#">FS_LIMIT</a> , <a href="#">FASTDEC</a>

## 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 <b>FASTDEC</b> 。 输入 <b>OFF</b> 时，认为有信号输入，要相反效果可以用 <b>INVERT_IN</b> 反转电平( <b>ECI</b> 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	<a href="#">FWD_IN</a> , <a href="#">RS_LIMIT</a> , <a href="#">FASTDEC</a>


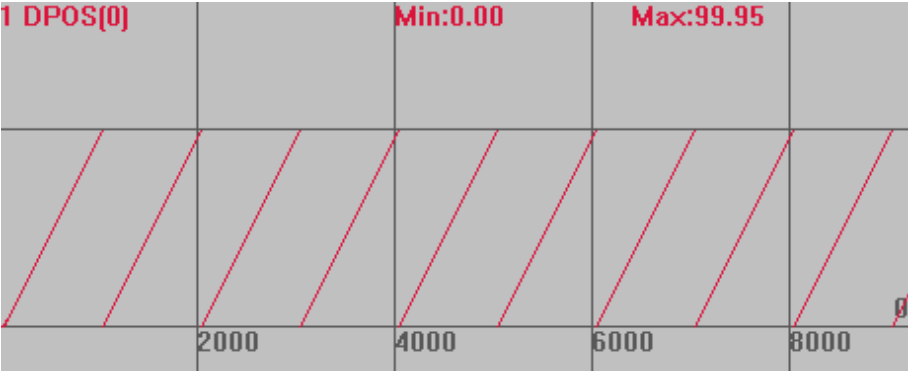
## ALM\_IN -- 映射报警输入

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	<a href="#">DATUM_IN</a> , <a href="#">FWD_IN</a> , <a href="#">REV_IN</a> , <a href="#">INVERT_IN</a> , <a href="#">FASTDEC</a>

## 11.13. 限幅参数指令

### REP\_OPTION -- 坐标循环模式

类型	轴参数															
描述	<p>坐标重复设置。</p> <p>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。</p> <p>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。</p> <p>相对运动不受影响。</p> <p>REP_DIST 超过参数支持的最大数值，就会把 REP_OPTION 的 bit4 置 1。</p>															
语法	<p>VAR1 = REP_OPTION，REP_OPTION = opt</p> <p>opt: 按位来表示不同的意义</p> <table><tr><td>位</td><td>值</td><td>描述</td></tr><tr><td>0</td><td>1</td><td>0-循环范围 - REP_DIST 到 +REP_DIST 1-循环范围为 0 到 +REP_DIST</td></tr><tr><td>1</td><td>2</td><td>写 1 禁止 CAMBOX 和 MOVELINK 的重复运动，禁止生效后还原为 0</td></tr><tr><td>2</td><td>4</td><td>预留</td></tr><tr><td>4</td><td>16</td><td>1-不使用 REP_DIST，0-使用 REP_DIST</td></tr></table>	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
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	<a href="#">CAMBOX</a> , <a href="#">MOVELINK</a> , <a href="#">REP_DIST</a>	



## 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	<a href="#">REP_OPTION</a>

## 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	<a href="#">MPOS</a> , <a href="#">DPOS</a>

## 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	<a href="#">FE</a> , <a href="#">FE_RANGE</a>

## 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	<a href="#">FE</a> , <a href="#">FE_LIMIT</a>

## 11.14. 高级设置指令

### INVERT\_STEP -- 脉冲模式设置

类型	轴参数																																																				
描述	<p>伺服/步进脉冲输出模式设置。</p> <p>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的支持。</p> <p>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。</p>																																																				
语法	<p>INVERT_STEP = mode</p> <p>mode: 模式选择，缺省 0</p> <p>低 8 位（位 0-位 7）表示的模式值如下：</p> <p>0-3 脉冲方向模式，脉冲线+方向线</p> <p>4-7 双脉冲方式（或称 CW/CCW），正向脉冲线+负向脉冲线</p> <p>8-9 AB 输出(部分控制器定制)</p> <p>各个模式对应的电平如下：</p> <table><tr><th rowspan="2">模式值</th><th rowspan="2">描述</th><th colspan="2">松下设置参考</th><th>三菱设置参考</th></tr><tr><th>Pr0.06</th><th>Pr0.07</th><th>PA13</th></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> <p>高 8 位（位 8-位 15）表示方向变化保护时间，单位微秒：0-255</p> <p>常用模式为 0、2、6、7。</p> <p>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。</p>					模式值	描述	松下设置参考		三菱设置参考	Pr0.06	Pr0.07	PA13	0	脉冲/方向（脉冲正逻辑）（正向）	0	3	××01h	1	脉冲/方向（脉冲负逻辑）（正向）	/	/	××11h	2	脉冲/方向（脉冲正逻辑）（负向）	1	3	××01h	3	脉冲/方向（脉冲负逻辑）（负向）	/	/	××11h	4	双脉冲（方向负逻辑）（正向）	/	/	××10h	5	双脉冲（方向负逻辑）（负向）	/	/	××10h	6	双脉冲（方向正逻辑）（正向）	1	1	××00h（默认）	7	双脉冲（方向正逻辑）（负向）	0(默认)	1(默认)	××00h（默认）
模式值	描述	松下设置参考		三菱设置参考																																																	
		Pr0.06	Pr0.07	PA13																																																	
0	脉冲/方向（脉冲正逻辑）（正向）	0	3	××01h																																																	
1	脉冲/方向（脉冲负逻辑）（正向）	/	/	××11h																																																	
2	脉冲/方向（脉冲正逻辑）（负向）	1	3	××01h																																																	
3	脉冲/方向（脉冲负逻辑）（负向）	/	/	××11h																																																	
4	双脉冲（方向负逻辑）（正向）	/	/	××10h																																																	
5	双脉冲（方向负逻辑）（负向）	/	/	××10h																																																	
6	双脉冲（方向正逻辑）（正向）	1	1	××00h（默认）																																																	
7	双脉冲（方向正逻辑）（负向）	0(默认)	1(默认)	××00h（默认）																																																	
适用控制器	通用																																																				
例子	<p>设置为脉冲方向模式：</p> <p>INVERT_STEP = 256*100+0    '设置 100 微秒的保护时间，模式为 0</p> <p>设置为双脉冲模式：</p> <p>INVERT_STEP = 256*100+6    '设置 100 微秒的保护时间，模式为 6</p> <p>查询脉冲模式设置：</p>																																																				

	<p>在线命令栏输入如下指令查询。</p> <p>?INVERT_STEP(0)     '打印轴 0 的脉冲模式设置值</p> <p>?*INVERT_STEP     '打印全部轴的脉冲模式设置值</p>
相关指令	<a href="#">STEP_RATIO</a>

## MAX\_SPEED -- 脉冲频率限制

类型	轴参数
描述	<p>脉冲输出的最高频率限制。</p> <p>一旦发现超过此设置值会强制，并且设置 AXISSTATUS。</p> <p>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）。</p> <p>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。</p>
语法	MAX_SPEED = value
适用控制器	通用
例子	<p><b>MAX_SPEED</b> AXIS(n)=4000000     '设置轴 n 脉冲速度限制 4000000</p> <p>BASE(6)</p> <p>ATYPE=3</p> <p><b>MAX_SPEED</b> =500000     '启动编码器滤波</p>
相关指令	<a href="#">AXISSTATUS</a>

## AXIS\_ZSET -- 精准 op 设置

类型	轴参数
描述	<p>对轴启用 MOVEOP 精准输出功能，作用在轴组的主轴上。</p> <p>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。</p> <p>设置参数：</p> <p>bit0: 1-VP_SPEED 缺省使用插补速度，0-VP_SPEED 使用单轴的速度</p> <p>bit1: 1-使用 MOVE_OP 精确输出功能， 0- MOVE_OP 原来的方式</p> <p>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</p> <p>多个编码器轴插补时，使用 BASE 运动主轴的设置</p>
语法	<p>可读: VALUE=AXIS_ZSET</p> <p>可写: AXIS_ZSET=VALUE</p>
适用控制器	20170517 以上固件版本
例程	<p>例一 开启精准输出</p> <p>BASE(0)</p> <p>ATYPE=1</p>

	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	<a href="#">SYSTEM_ZSET</a> ， <a href="#">MOVE_OP</a>

## AXIS\_MODE -- connect 运动保持

类型	轴参数
描述	<p>设置 BIT1=1，防止限位和软限位导致 CONNECT 运动退出。</p> <p>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。</p> <p><b>固件版本 20170616 及以上支持此功能。</b></p>
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
```

	1 DPOS[0]		Min:0.00	Max:1000.00
	2 DPOS[1]		Min:-499.95	Max:500.05
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	<a href="#">CONNECT</a> ， <a href="#">FS_LIMIT</a> ， <a href="#">RS_LIMIT</a>			

## MOVEOP\_DELAY -- 缓冲输出延时

类型	轴参数
描述	<b>BASE 轴缓冲信号延时输出。</b>  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
例程	MOVEOP_DELAY = 2 '实际输出时间延迟 2ms
相关指令	<a href="#">MOVE_OP</a> ， <a href="#">HW_PSWITCH</a> ， <a href="#">HW_PSWITCH2</a>

## DAC -- 总线轴模拟量控制

类型	轴参数
描述	<b>伺服轴 DA 直接控制，速度或力矩模式下支持。</b>  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



此时电机将以 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

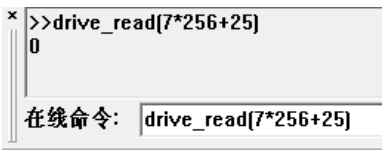
[大小]：带符号 32bit

[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~7FFFFFFh



此时单位为【r/min】

例二 Rtex 力矩控制

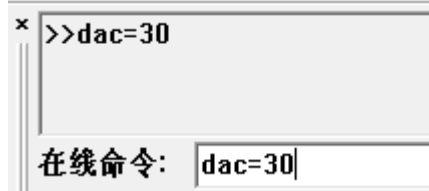
请先将驱动器参数 pr6.47 的第一位置为 0，关闭 2 自由度控制模式；参数 pr3.17 设置速度限制，如下图。（参照松下 Rtex 手册）

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		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	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
		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				

		(最大过速度等级)的 最小设定速度进行限制	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/(Pr9.06×2%)

### 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



	<p>ENDIF</p> <p>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。</p>
相关指令	<a href="#">SERVO</a>

## 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	<a href="#">AXISSTATUS</a> , <a href="#">WD OG</a>

## ZSCAN\_CORRECT -- 振镜矫正

类型	轴参数
描述	矫正振镜轴参数。
语法	<p>ZSCAN_CORRECT(ixy,imode,imaxline,imaxrow,x1,y1,x2,y2,tableindex)</p> <p>ixy: 值为 0 或 1，两个振镜选择；0-第一个振镜，1-第二个振镜</p> <p>imode: 0-关闭矫正功能；1-使用分区矫正，table 输入测量得到的实际位置；2-使用分区矫正，table 输入需要运行到的脉冲位置，210701 增加此功能</p> <p>imaxline: 行数，Y 方向的点数为行数，数据越多精度越高</p> <p>imaxrow: 列数，X 方向的点数为列数</p> <p>x1,y1: 理论的左下角的位置</p> <p>x2,y2: 理论的右上角的位置</p> <p>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，注意此 XY 是实际的物理轴，第一个物理轴为 X，第二个为 Y，与轴号映射的虚拟轴轴号无关，填入坐标为振镜实际的脉冲位置(支持小数)(XY2 协议坐标范围是-32768-32767)</p>
适用控制器	带振镜轴控制器
例子	<p>TABLE(0, -40.6,-41.2)</p> <p>TABLE(2, 0,-41)</p> <p>TABLE(4, 41,-42)</p> <p>TABLE(6, -41,0)</p> <p>TABLE(8, 0,0)</p> <p>TABLE(10, 41.2,0)</p> <p>TABLE(12, -40.4,41.2)</p> <p>TABLE(14, 0,41.2)</p> <p>TABLE(16, 41,42.4)</p>

	FOR i=0 TO 17 TABLE(i) = TABLE(i)*500      '输入的都是脉冲坐标 NEXT  ZSCAN_CORRECT(0,1,3,3,-20000,-20000,20000,20000,0)
--	--

## 11.15. 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	<a href="#">P_GAIN</a> , <a href="#">I_GAIN</a> , <a href="#">OV_GAIN</a> , <a href="#">VFF_GAIN</a>

### 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	<a href="#">P_GAIN</a> , <a href="#">D_GAIN</a> , <a href="#">OV_GAIN</a> , <a href="#">VFF_GAIN</a>

### 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	<a href="#">P_GAIN</a> , <a href="#">D_GAIN</a> , <a href="#">I_GAIN</a> , <a href="#">VFF_GAIN</a>

### 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	<a href="#">I_GAIN</a> , <a href="#">D_GAIN</a> , <a href="#">OV_GAIN</a> , <a href="#">VFF_GAIN</a>

## 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	<a href="#">P_GAIN</a> , <a href="#">I_GAIN</a> , <a href="#">D_GAIN</a> , <a href="#">OV_GAIN</a>

## 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	<a href="#">WDOG</a>

## TRANS\_DPOS

类型	轴状态
描述	预留

## 第十二章 输入输出相关指令

### 12.1. 输入相关指令

#### IN -- 输入口

类型	输入输出函数
描述	<p>读取输入，没有参数时返回 <b>0-31</b> 的状态。</p> <p>读取的是 INVERT_IN 翻转以后的状态。</p> <p>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。</p> <p>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。</p>
语法	<p>IN([channel1],[channel2])</p> <p>channel1: 要读取的起始输入通道</p> <p>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</p>
适用控制器	通用
例子	a=IN(1)      '读取通道 1 的输入
相关指令	<a href="#">OP</a> , <a href="#">INVERT IN</a>

#### AIN -- 模拟量输入

类型	输入输出函数
描述	<p>读取模拟输入，返回 AD 转换模块的刻度值。</p> <p>12 位刻度值范围 0~4095 对应 0~10V 电压。</p> <p>16 位刻度值范围 0~65536 对应 0~10V 电压。</p> <p>ZAI0 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。</p> <p>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。</p>
语法	<p>VAR=AIN(channel)</p> <p>channel: 模拟输入通道 0-127</p>
适用控制器	通用
例子	<p>a=AIN(1)      '读取通道 1 的 AD</p> <p>a=AIN(1) *10 /4095      '通道 1 的电压值</p>
相关指令	<a href="#">AOUT</a>

## 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	<a href="#">IN</a>

## 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	<a href="#">AIN</a>

## 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	<a href="#">ENCODER</a>

## 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	<b>INVERT_IN(1,ON)</b> '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) <b>FWD_IN(0)=1</b> '输入 IN1 作为轴 0 的正向限位信号
相关指令	<a href="#">IN</a>

## IN\_SCAN -- 扫描输入变化

类型	输入输出函数
描述	<p>扫描输入变动，返回值 <b>1(TRUE)</b>-变动，<b>0(FALSE)</b>-没有变动。</p> <p>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 <b>IN_EVENT</b> 读取变动的具体情况，使用的是 <b>INVERT_IN</b> 翻转以后的状态。</p> <p>固件版本 <b>20140214</b> 以后的才提供这个支持，扫描范围有宽度限制。 <b>00x 系列控制器只能在单个任务中使用。</b></p>
语法	<b>VAR1=IN_SCAN([channel1][,channel2])</b> 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	<pre> WHILE 1 IF IN_SCAN(0,23) THEN           '扫描 IN0-23 口电平变化     IF IN_EVENT(0) &gt; 0 THEN      'IN0 上升沿触发         PRINT "IN0 UP", IN_BUFF(0)     ELSEIF IN_EVENT(0) &lt; 0 THEN  'IN0 下降沿触发         PRINT "IN0 DOWN", IN_BUFF(0)     ENDIF ENDIF WEND </pre>
相关指令	<a href="#">IN_EVENT</a> , <a href="#">SCAN_EVENT</a> , <a href="#">IN_BUFF</a>

## IN\_EVENT -- 读取输入变化

类型	输入输出函数
描述	<p>读取输入的变化情况。</p> <p>1-上升沿，-1-下降沿，0-没有变化。 此函数必须与 <b>IN_SCAN</b> 配合使用。</p>
语法	<b>VAR1 = IN_EVENT(IONUM)</b>
适用控制器	通用

例子	参见 IN_SCAN
相关指令	<a href="#">IN_SCAN</a> , <a href="#">SCAN_EVENT</a>

## SCAN\_EVENT -- 检测变化

类型	输入输出函数
描述	<p>检测表达式的内容变化。</p> <p>OFF- ON 返回 1, ON-OFF 返回-1, 不变返回 0。</p> <p>不要在循环内或者多任务调用同一个 SUB 内的 SCAN_EVENT。</p> <p>150810 之后固件版本支持, 之前版本用 IN_EVENT 和 IN_SCAN。</p>
语法	<p>ret = SCAN_EVENT (expression)</p> <p>expression: 任意有效的表达式, 结果会转成 BOOL 类型</p>
适用控制器	通用
例子	<p>例一 输入信号扫描</p> <pre> WHILE 1   IF  SCAN_EVENT(IN(0)) &gt; 0 THEN      'IN0 上升沿触发     PRINT "IN0 ON"   ELSEIF  SCAN_EVENT(IN(0))&lt;0 THEN    'IN0 下降沿触发     PRINT "IN0 OFF"   ENDIF WEND </pre> <p>例二 寄存器、变量扫描</p> <pre> WHILE 1   IF  SCAN_EVENT(TABLE(0)) &gt; 0 THEN    'TABLE0 上升沿触发     PRINT "TABLE0 ON"   ELSEIF  SCAN_EVENT(TABLE(0))&lt;0 THEN  'TABLE0 下降沿触发     PRINT "TABLE0 OFF"   ENDIF WEND </pre> <p>在线命令操作 table(0), 打印相关结果</p>
相关指令	<a href="#">IN_SCAN</a> , <a href="#">IN_EVENT</a>

## IN\_BUFF -- 读取输入缓冲

类型	输入输出函数
描述	<p>读取 IN_SCAN 缓冲的当前输入, 没有参数时返回 0-31 的状态。</p> <p>读取的是 INVERT_IN 翻转以后的状态。</p>
语法	<p>IN_BUFF([channel1],[channel2])</p> <p>channel1: 要读取的起始输入通道, 必须是 IN_SCAN 的输入范围</p>

	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	<a href="#">IN_SCAN</a>

## 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	<b>INFILTER= 5</b> 在干扰严重场合加大滤波时间

## 12.2. 输出相关指令

### OP -- 输出口

类型	输入输出指令和函数
描述	<p>输出或读取输出状态。</p> <p>当在表达式中使用时, 自动为函数语法。</p> <p>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。</p> <p>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。</p> <p>最多可操作 32 个输出口。</p>
语法	<p>OP([ionum],value)</p> <p>或 OP(ionum1, ionum2,value[,mask])</p> <p>或 OP([firstnum],[finalnum])</p> <p>ionum: 输出编号, 从 0 开始</p> <p>value: 输出状态, 多个输出口操作时按位来指明多个口状态</p> <p>ionum1: 要操作的第一个输出通道</p> <p>ionum2: 要操作的最后一个输出通道</p> <p>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</p> <p>firstnum: 输出编号, 从 0 开始</p> <p>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</p>
适用控制器	通用
例子	<p>例一 单个操作</p> <p>'翻转输出口 0</p>



	<pre> 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) </pre>
相关指令	<a href="#">READ_OP</a> , <a href="#">MOVE_OP</a>

## AOUT -- 模拟量输出

类型	输入输出指令或函数
描述	<p><b>模拟通道输出。</b></p> <p>12 位刻度值范围 0~4095 对应 0~10V 电压。</p> <p>16 位刻度值范围 0~65536 对应 0~10V 电压。</p>
语法	<pre>AOUT(channel) = value</pre> <p>channel: 模拟输出通道 0-63</p> <p>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。</p>
适用控制器	通用
例子	<pre> AOUT(1) = 0    '关闭输出 DA 通道 1 AOUT(1) = 4095 'DA1 口输出 10V 电压 </pre>
相关指令	<a href="#">AIN</a>

## 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	<a href="#">OP</a>

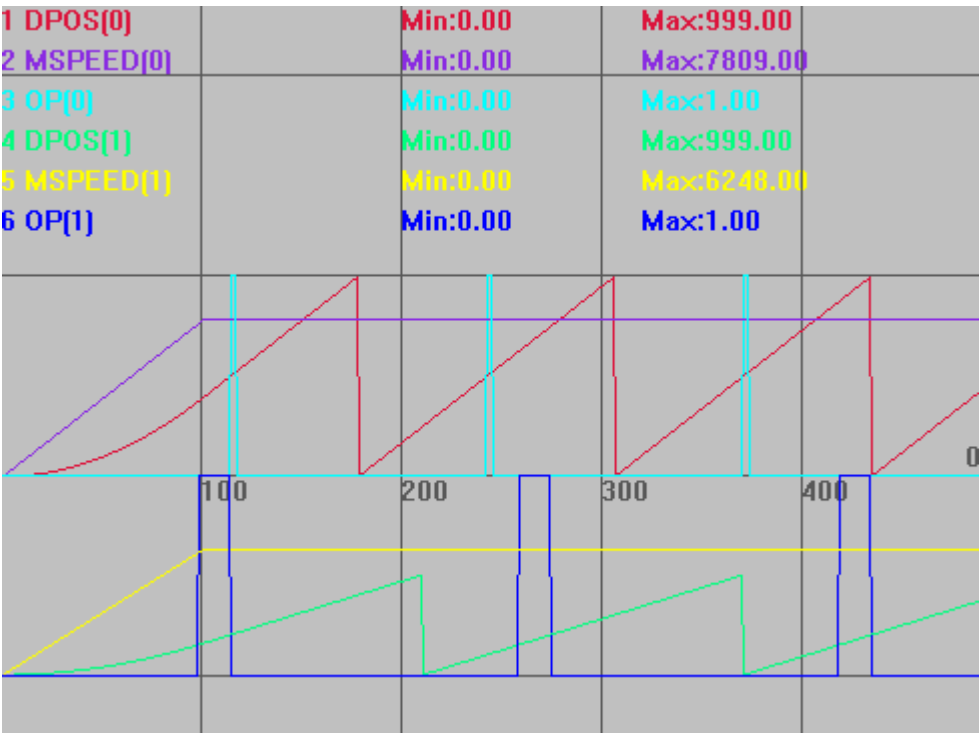
## 12.3. 位置比较输出指令

### 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 <b>PSWITCH</b> 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 <b>ATYPE</b> 为 4 时才是比较反馈位置(MPOS)，默认出厂的 <b>ATYPE</b> 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

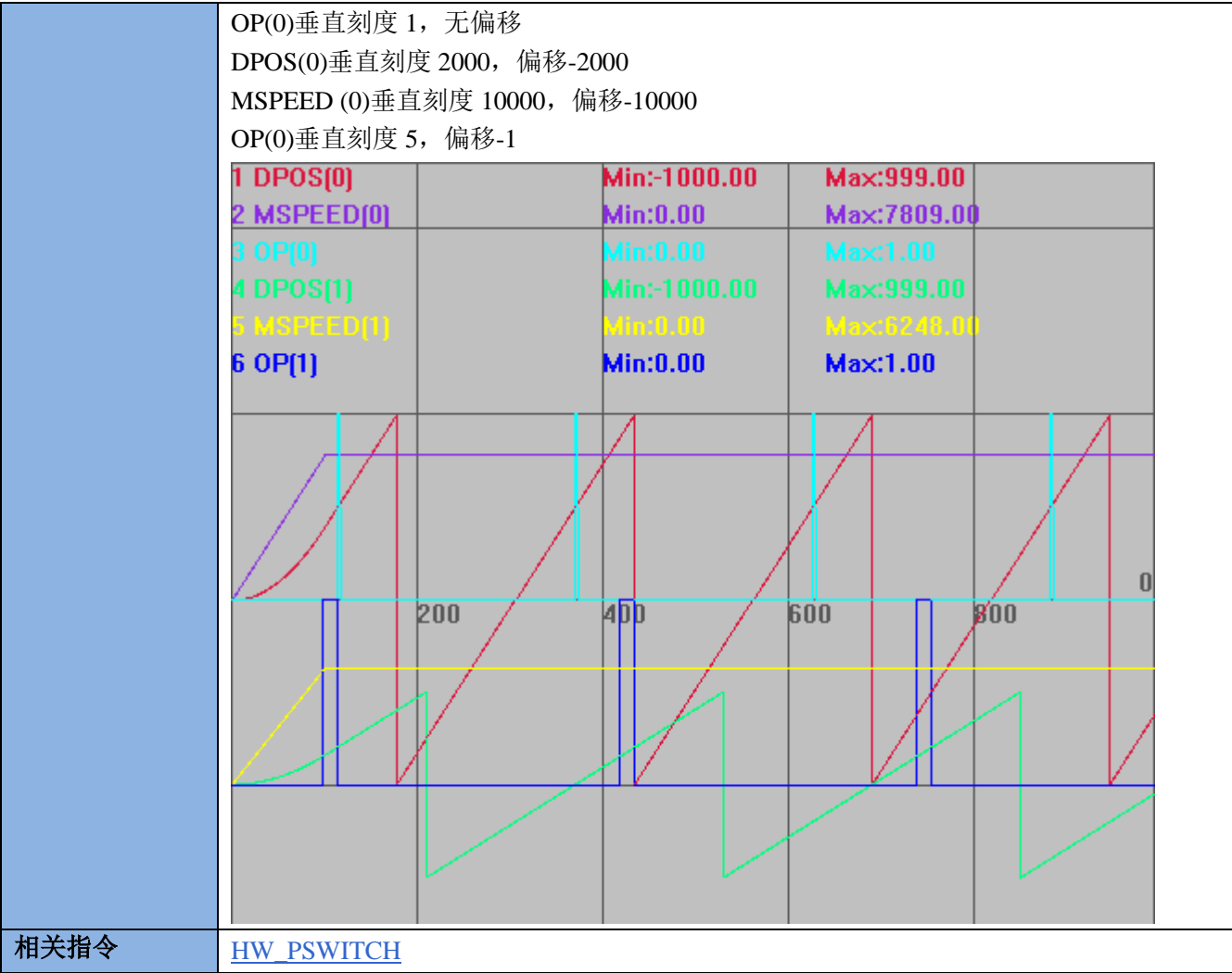
```
ATYPE=1,1      '脉冲方式步进或伺服
DPOS = 0,0
UNITS = 1,1     '脉冲当量
SPEED = 10000,10000
ACCEL=SPEED(0)*10,SPEED(1)*10
DECEL=SPEED(0)*10,SPEED(1)*10
REP_OPTION=1,1   '设置坐标循环范围为 0 到+ REP_DIST
REP_DIST=1000,1000
TRIGGER
MOVE(10000,8000)
PSWITCH(0,ON,0,0,ON,500,520)
PSWITCH(1,ON,1,1,ON,300,400)
END
```

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

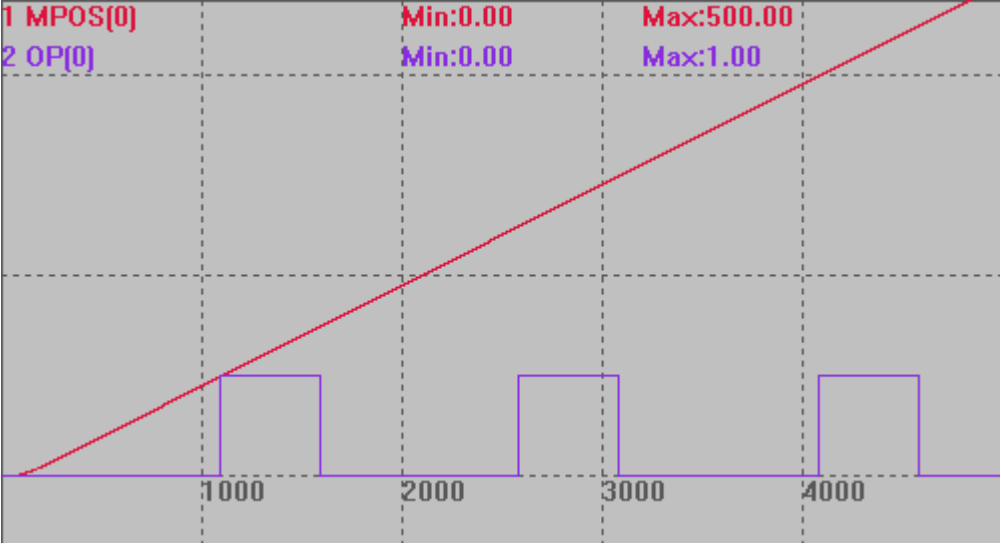


相关指令 [HW\\_PSWITCH](#)

## HW\_PSWITCH -- 硬件位置比较输出

类型	轴指令
描述	<p>硬件比较输出，不同的轴对应不同的输出口。</p> <p>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。</p> <p>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。</p> <p>每个比较点触发都会使得当前输出口电平翻转。</p> <p><b>HW 缓冲数 1024 个，可以连续调用 1024 个 HW 指令。</b></p> <p><b>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。</b></p> <p><b>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。</b></p> <p><b>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。</b></p>

	使用脉冲型电机时只有 <b>ATYPE</b> 为 4 时才是比较反馈位置(MPOS)，默认出厂的 <b>ATYPE</b> 为 1 或 7 比较的是命令位置(DPOS)。
语法	<p>HW_PSWITCH(mode, direction, reserve, tablestart, tableend)            Buff = HW_PSWITCH([axisnum])</p> <p>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</p> <p>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。</p>
适用控制器	<p>4 系列及以上产品, 4 系列固件 20170704 及以上            ZMC420SCAN 不支持 HW_PSWTICH</p>
例子	<p>以下例程测试环境: ZMC432, 固件: 20170709 (仿真器无法运行此指令)。</p> <p>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  <b>HW_PSWITCH(2)</b> '停止并删除没有完成的比较点  <b>HW_PSWITCH(1, 1, 0, 0, 5)</b> '启动比较输出            TRIGGER            MOVEABS(500)</p> <p>比较输出图            MPOS(0)垂直刻度 200            OP(0)垂直刻度 2</p>

	
相关指令	<a href="#">PSWITCH</a> , <a href="#">HW_PSWITCH2</a>

## HW\_TIMER -- 硬件定时

类型	特殊命令
描述	<p>硬件定时器，用于硬件比较输出后一段时间后还原电平。</p> <p><b>HW_TIMER</b> 只有 1 个，每次调用会强制停止之前的调用。</p> <p><b>ZMC420SCAN</b> 每个输出口的 <b>HW_TIMER</b> 功能独立。</p> <p>振镜控制器 20170709 以上固件版本增加功能：</p> <p>OP 和 MOVE_OP 操作会关闭正在进行的 <b>HW_TIMER</b> 脉冲，这样可以使用 <b>HW_TIMER</b> 来实现类似 PWM 的功能，OP 输出打开脉冲输出，下一个 OP 输出关闭脉冲输出，当使用 MOVE_OP 精准输出时，可以实现精准的 PWM 输出无限脉冲功能。</p> <p>使用 ?*<b>HW_TIMER</b> 可以看到还有多少脉冲剩余。</p>
语法	<p><b>HW_TIMER(mode, cyclonetime, optime, reptimes, opstate, opnum )</b></p> <p>done = <b>HW_TIMER_DONE</b></p> <p>mode: 0 停止, 2-启动</p> <p>cyclonetime: 周期时间, us 单位</p> <p>optime: 有效时间, us 单位</p> <p>reptimes: 重复次数, 启动模式, reptimes =0 时, 软关闭 <b>HW_TIMER</b>, 原来的脉冲没有完成的, 会继续输出完成</p> <p>opstate: 输出缺省状态, 输出口变为非此状态后开始计时</p> <p>opnum: 输出口编号, 必须能硬件比较输出的口</p> <p>振镜控制器 20170710 以上版本增加 mode 3, 类似 0, 当前脉冲完成后不再输出, 但后面的 OP 仍会继续触发脉冲。</p>
适用控制器	3 系列部分、4 系列及以上产品, 4 系列固件 20170704 及以上
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

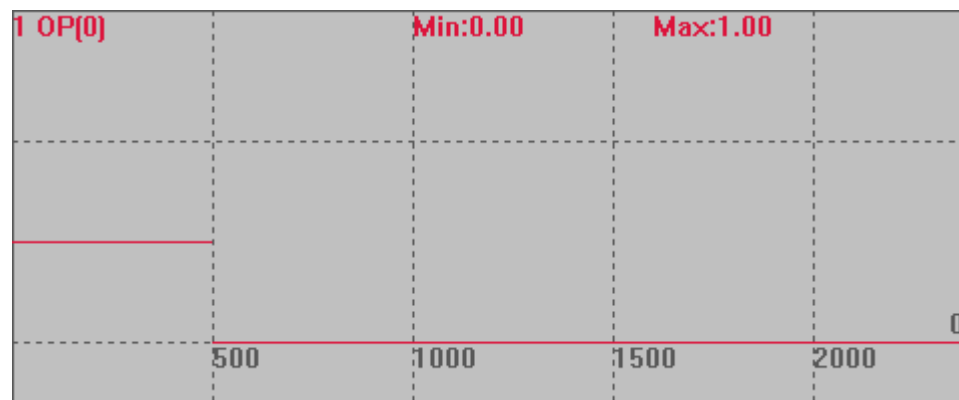
TRIGGER

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

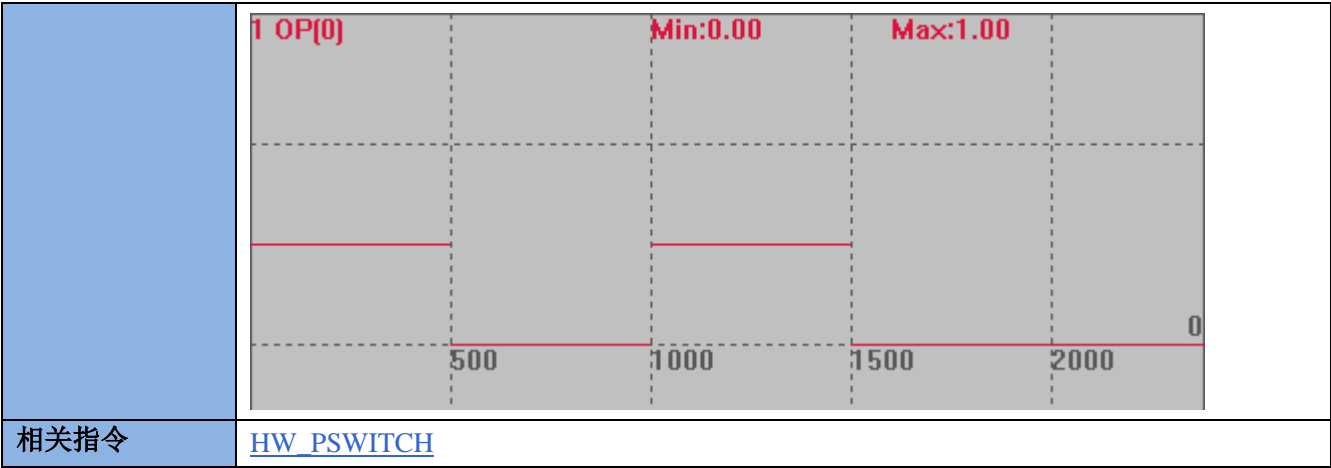
TRIGGER

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，后半周期关闭。



HW\_PSWITCH2 -- 总线硬件位置比较输出

类型	轴指令
描述	<p>总线硬件位置比较输出，也支持脉冲轴，必须使用特定的输出口。</p> <p>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。</p> <p>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。</p> <p>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。</p> <p><b>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。</b></p> <p><b>每个系统周期内只能比较一次，系统周期通过 SERVO_PERIOD 查询。</b></p> <p><b>TABLE 位置数据在所有比较点完成前不要修改。</b></p> <p><b>脉冲轴和总线轴均支持此指令。</b></p> <p><b>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)。</b></p>
语法	<p>命令语法：HW_PSWITCH2(mode, [...])</p> <p>函数语法：Buff = HW_PSWITCH2([axisnum])</p> <p><b>Mode=1:单轴比较，见例一</b></p> <p>HW_PSWITCH2(1,opnum,opstate,tablestart,tableend[,direction])</p> <p>mode: 1-启动比较器</p> <p>opnum: 对应的输出口</p> <p>opstate: 第一个比较点的输出状态</p> <p>tablestart: 第一个比较点绝对坐标所在 TABLE 编号</p> <p>tableend: 最后一个比较点绝对坐标所在 TABLE 编号</p> <p>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</p> <p><b>Mode=2:清除比较点</b></p>



**HW\_PSWITCH2(2)**

mode: 2-停止并删除没完成的比较点

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=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 **VECTOR\_MOVED** 坐标所在 TABLE 编号

tableend: 最后一个比较点 **VECTOR\_MOVED** 坐标所在 TABLE 编号

**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 **VECTOR\_MOVED** 当前运动距离

**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 **VECTOR\_MOVED** 当前运动距离

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 **VECTOR\_MOVED** 当前运动距离

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

**Mode=7: 与 HW\_TIMER 一起使用****HW\_PSWITCH2(7, opnum, opstate, tablestart, tableend [, optimeus, optimes, cyctimeus])**

mode: 7-启动比较器，opstate 不翻转，方便与 **HW\_TIMER** 配合使用。

opnum: 对应的输出口

	<p>opstate: 第一个比较点的输出状态</p> <p>tablestart: 第一个比较点 VECTOR_MOVED 坐标所在 TABLE 编号</p> <p>tableend: 最后一个比较点 VECTOR_MOVED 坐标所在 TABLE 编号 [与 hwtimer 并用时, 可以动态调整 hwtimer 参数]</p> <p>optimeus: 动态调整 HW_TIMER 的有效时间</p> <p>optimes: 动态调整 HW_TIMER 的触发脉冲数, 0-不输出</p> <p>cyctimeus: 动态调整 HW_TIMER 的脉冲周期时间</p> <p>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。</p> <p><b>以下为 2D, 3D 比较模式</b></p> <p>4 系列 170706 以后固件版本支持, 运行轨迹必须顺序通过 fifo 的点。</p> <p><b>2D 比较: 每 2 个 table 存储一个点</b> 多个点比较, 每次输出翻转状态。 HW_PSWITCH2(25, opnum, opstate, maxerr, num, tablepos) 注意: 此模式下 maxerr 偏差参数不能写 0。</p> <p>与 HW_TIMER 复用 HW_PSWITCH2(26, opnum, opstate, maxerr, num, tablepos, [ophvertimeus, ophwtimes, hwcycetimeus])</p> <p><b>3D 比较: 每 3 个 TABLE 存储一个点</b> 多个点比较, 每次输出翻转状态。 HW_PSWITCH2(35, opnum, opstate, maxerr, num, tablepos)</p> <p>与 HW_TIMER 复用 HW_PSWITCH2(36, opnum, opstate, maxerr, num, tablepos, [ophvertimeus, ophwtimes, hwcycetimeus])</p> <p>参数:</p> <ul style="list-style-type: none"> <li>mode: 25, 26, 35, 36 多维的比较模式</li> <li>opnum: 对应的输出口</li> <li>opstate: 第一个比较点的输出状态</li> <li>maxerr: 比较位置每个轴左右的脉冲偏差, 进入偏差范围后开始比较</li> <li>num: table 里面存储的比较点个数</li> <li>tablepos: 第一个比较点坐标所在 table 编号 [与 hwtimer 并用时, 可以动态调整 hwtimer 参数]</li> <li>ophvertimeus: 脉冲时间</li> <li>ophwtimes: 脉冲个数</li> <li>hwcycetimeus: 脉冲周期</li> </ul>
适用控制器	3 系列部分、4 系列及以上产品, 4 系列固件 20170704 及以上

## 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

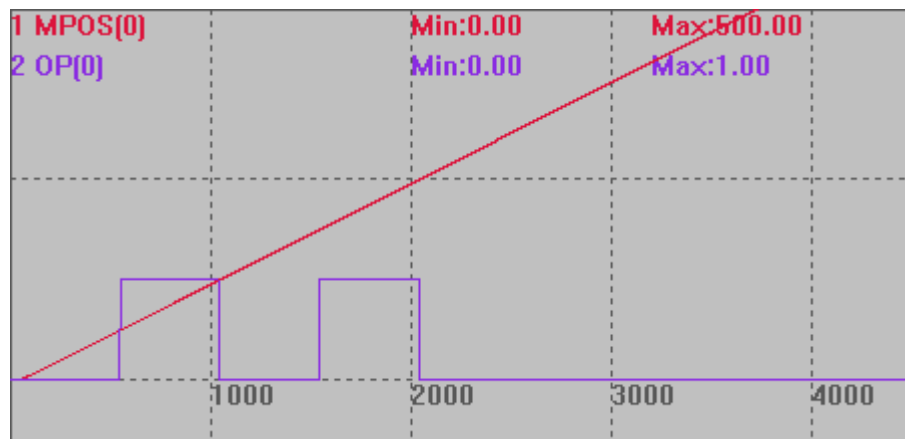
MOVE(500)

### 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=120 '设置矢量起始位置，会影响比较点

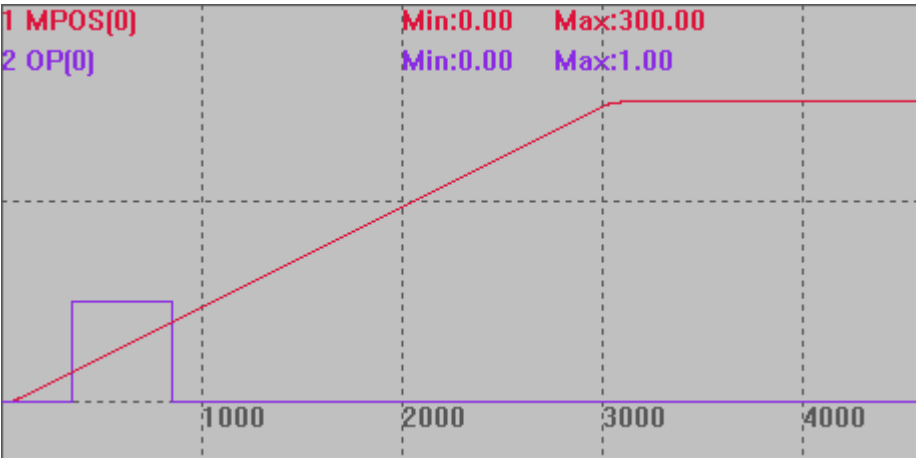
HW\_PSWITCH2(2) '停止并删除没有完成的比较点

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 120 处开始，只比较了后面两个点，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 XY 加工模式的 PSO 应用

```
RAPIDSTOP(2)
WAIT IDLE(0)
WAIT IDLE(1)
BASE(0,1)
ATYPE=1,1
SPEED=100,100
ACCEL=1000,1000
DECEL=1000,1000
SRAMP=100,100
DPOS=0,0
MPOS=0,0
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
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(300,400)
END
```

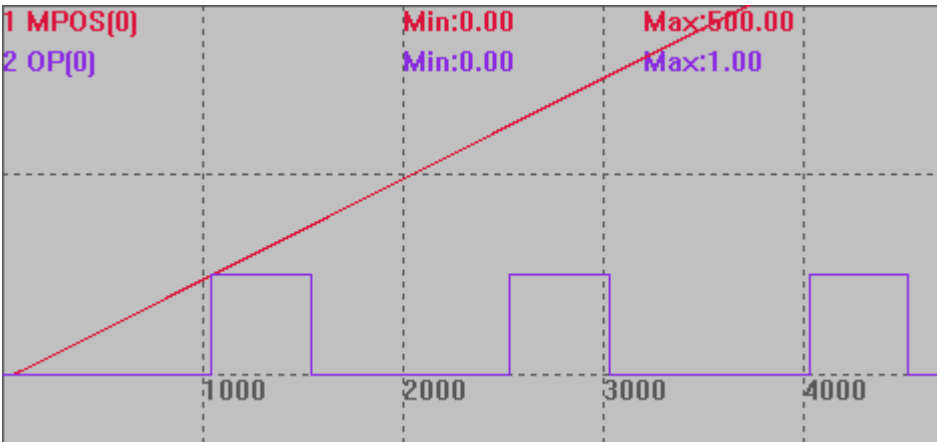
比较输出图：按 XY 轴插补位置比较  
VECTOR\_MOVED(0)垂直刻度 200  
OP(0)垂直刻度 2



例四 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

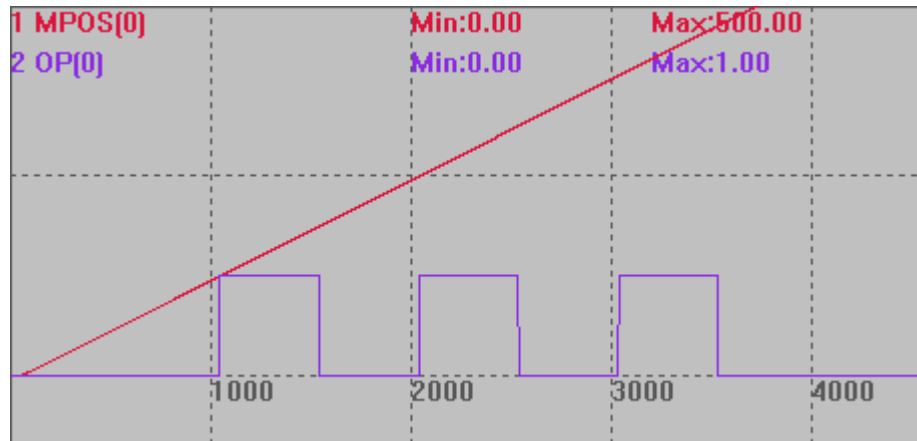
MOVE(500)

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



例五 二维硬件位置比较输出，模式 25

BASE(0,1) '选择 XY 轴

UNITS=1000,1000

SPEED=100,100

ACCEL=10000,10000

DECEL=10000,10000

'将当前位置设置为 0 点

DPOS=0,0

MPOS=0,0

'提前写入位置比较点 XY 坐标到 table 10~49 中

TABLE(10, 10,0,12,0, 20,0,22,0, 30,0,32,0, 40,0,42,0, 50,0,52,0, 52,10,52,12, 52,20,52,22, 52,30,52,32, 52,40,52,42, 52,50,52,52)

GLOBAL pointNum '比较点数

```
pointNum=20

GLOBAL startX,startY '起点
startX=0
startY=0

GLOBAL midX,midY '中点
midX=52
midY=0

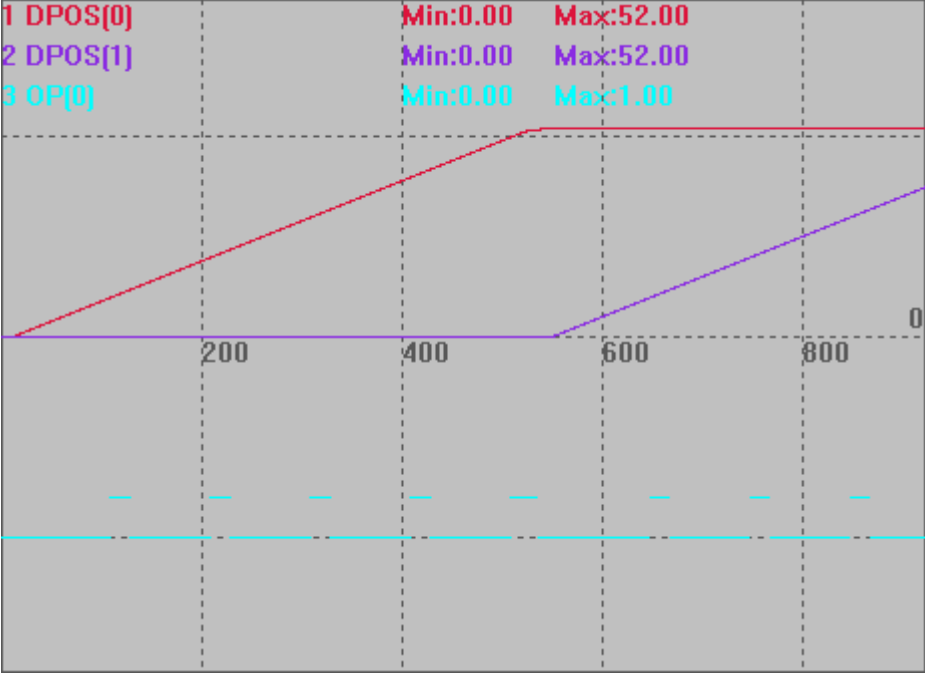
GLOBAL endX,endY '终点
endX=52
endY=52

WHILE 1
    IF TABLE(0)=1 THEN '比较脉冲位置,非精准输出
        WAIT IDLE
        ?"比较脉冲位置,非精准输出"
        '设置参数
        SYSTEM_ZSET=1
        AXIS_ZSET=1,1

    ELSEIF TABLE(0)=2 THEN '比较脉冲位置,精准输出
        WAIT IDLE
        ?"比较脉冲位置,精准输出"
        '设置参数
        SYSTEM_ZSET=3
        AXIS_ZSET=3,3

    ELSEIF TABLE(0)=3 THEN '比较编码器位置,非精准输出
        WAIT IDLE
        ?"比较编码器位置,非精准输出"
        '设置参数
        SYSTEM_ZSET=17
        AXIS_ZSET=17,17

    ELSEIF TABLE(0)=4 THEN '比较编码器位置,精准输出
        WAIT IDLE
        ?"比较编码器位置,精准输出"
        '设置参数
        SYSTEM_ZSET=19
        AXIS_ZSET=19,19
    ENDIF
```

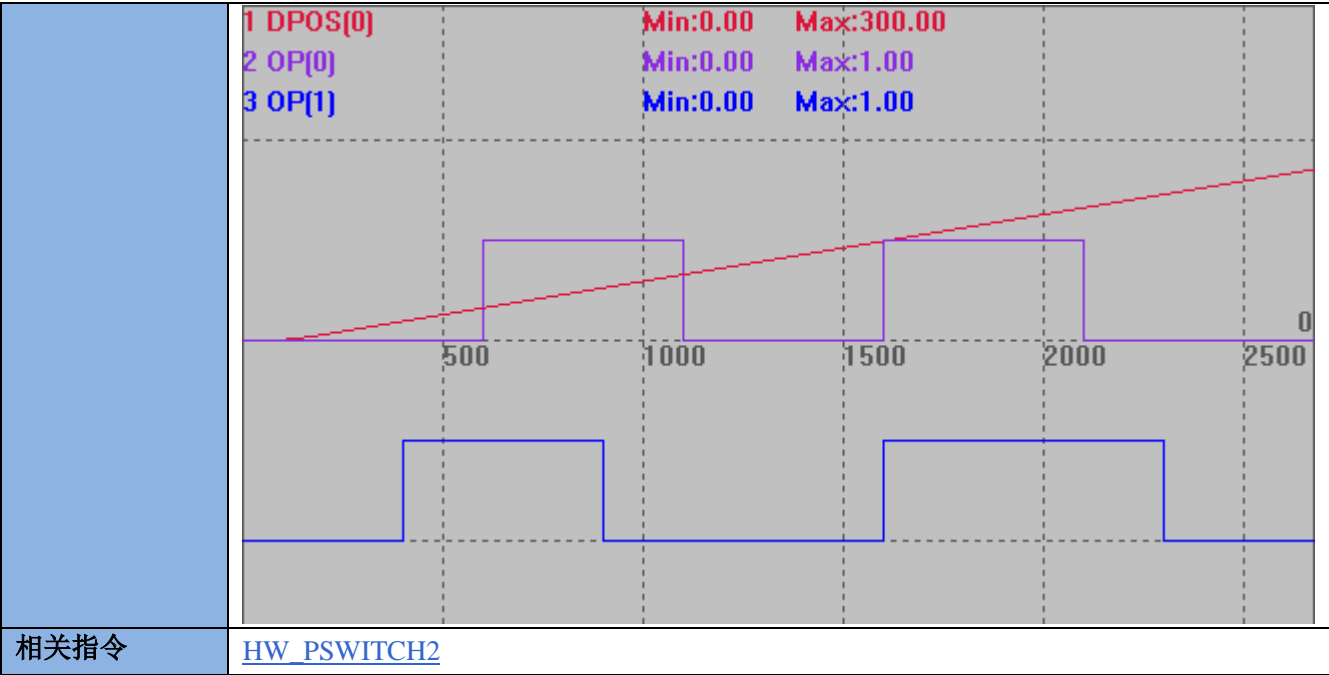
	<pre>IF TABLE(0)&lt;&gt;0 AND IDLE(0)=-1 THEN   TABLE(0)=0   ?"启动"    HW_PSWITCH2(2)          '先清除所有的比较   HW_PSWITCH2(25,0,1,10,pointNum,10)  '写入比较，操作输出口 0    WA 10   TRIGGER          '启动示波器    MOVEABS(startX,startY)    '开始运动   MOVEABS(midX,midY)   MOVEABS(endX,endY) ENDIF WEND END</pre> 
相关指令	<a href="#">PSWITCH</a> , <a href="#">HW_PSWITCH</a> , <a href="#">MOVEOP_DELAY</a> , <a href="#">REG_INPUTS</a>

HW\_PS2AXISNUM -- 设置 PS2 轴号

类型	轴参数
描述	设置 <b>HW_PSWITCH2</b> 指令实际操作的轴号，缺省-1 表示不修改。 用于把没有使用的轴的 <b>HW_PSWITCH2</b> 缓冲重复利用起来，指向当前运动的主轴，，可以对当前的主轴同时做多个比较。

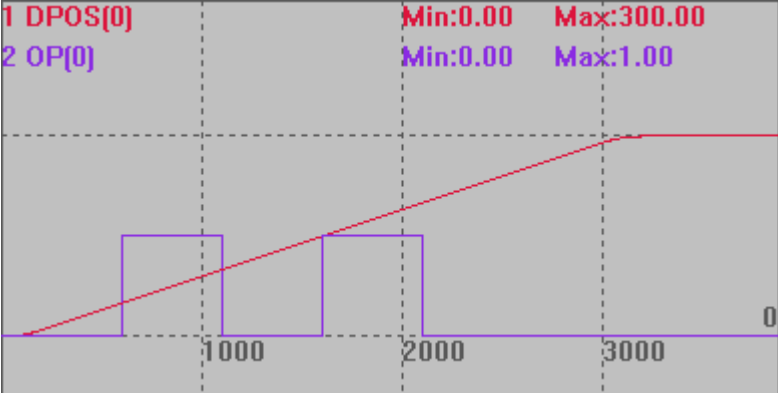


语法	HW_PS2AXISNUM(axisnum1)=axisnum2 axisnum1: 缓冲轴号 axisnum2: 实际操作的轴号
适用控制器	4 系列 170705 以后固件版本支持
例子	<p>以下例程测试环境：ZMC432，固件：20170709（仿真器无法运行此指令）。</p> <pre> RAPIDSTOP(2) WAIT IDLE(0) WAIT IDLE(1) BASE(0,1) ATYPE=1,1 UNITS=100,100 SPEED=100,100 ACCEL=500,500 DPOS=0,0 TRIGGER OP(0, OFF) OP(1, OFF)  HW_PS2AXISNUM(1)=0  '使用轴 1 的缓冲区，比较轴 0 的位置 VECTOR_MOVED =0 TABLE(0,50,100,150,200)  '第一个比较点坐标设置 TABLE(10,30,80,150,220)  '第一个比较点坐标设置  HW_PSWITCH2(1, 0, 1, 0, 3,1)          '第 1 个比较 HW_PSWITCH2(1, 1, 1, 10, 13,1)  AXIS(1)  '第 2 个比较，使用 1 轴的比较缓冲，实际 比较 0 轴的位置 MOVE(300) WAITIDLE(0) HW_PS2AXISNUM(1)=-1  '取消 END </pre>



HW\_PS2COUNTS -- PS2 比较的点数

类型	轴状态
描述	HW_PSITCH2 指令实际已经比较的点数，HW_PSITCH2(2)时清 0。
语法	HW_PS2AXISNUM(axisnum1)=axisnum2 axisnum1：缓冲轴号 axisnum2：实际操作的轴号
适用控制器	4 系列 170706 以后固件版本支持
例子	<p>以下例程测试环境：ZMC432，固件：20170711（仿真器无法运行此指令）。</p> <p>RAPIDSTOP(2) WAIT IDLE(0)</p> <p>BASE(0) ATYPE=1 UNITS=100 SPEED=100 ACCEL=500 DPOS=0 MPOS=0 OP(0,OFF) TABLE(0,50,100,150,200) '比较点坐标设置</p> <p>HW_PSITCH2(2) '停止并删除没有完成的比较点 HW_PSITCH2(1, 0, 1, 0, 3,1) '比较 4 个点，操作输出口 0</p>

	<p>TRIGGER '触发示波器 MOVE(300) ?HW_PS2COUNTS '打印 0, 还没到比较点 WAIT IDLE(0) ?HW_PS2COUNTS '打印 4, 比较了 4 个点 END</p> 
相关指令	<a href="#">HW_PSWITCH2</a>



## 12.4. PMW 控制指令

### PWM\_FREQ -- PWM 频率

类型	PWM 控制函数
描述	<b>PWM 频率设置或读取。</b> 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	<a href="#">PWM_DUTY</a> , <a href="#">MOVE_PWM</a>

### PWM\_DUTY -- pwm 占空比

类型	PWM 控制函数
描述	<b>PWM 占空比设置或读取。</b> 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




	<p>一定要在 PWM 开关之前调整。</p> <p>占空比指有效电平占整个周期的比例。</p> <p>一个周期中先输出有效电平，再输出无效电平。</p> <p>占空比为0.5 </p> <p>减小占空比 </p> <p>PWM 的实际输出受输出口的控制，必须输出口打开，PWM 才能输出，否则输出被屏蔽掉。可以先开 PWM 功能，然后再开输出口，这样实现激光电源的首脉冲抑制功能。</p>
语法	<p><code>PWM_DUTY(index, duty)</code>    <code>PWM_DUTY(index)=duty</code></p> <p>index: 编号，从 0 开始</p> <p>duty: 占空比，0-1，当设置 0 的时候，PWM 关闭</p>
适用控制器	支持 PWM 功能的控制器才支持此函数。
例子	<p><code>PWM_DUTY(0)=0.5</code></p> <p><code>?PWM_DUTY(0)</code></p> <p>打印结果 0.5</p>
相关指令	<a href="#">PWM_FREQ</a> , <a href="#">MOVE_PWM</a>

## 第十三章 通讯相关指令

### 13.1. 串口通讯指令

#### SETCOM -- 串口配置

类型	系统指令																								
描述	<p>串口的配置。</p> <p>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。</p> <p>ZMC00x 系列不支持 MODBUS 主端。</p>																								
语法	<p>SETCOM (baudrate,databits,stopbits,parity,port[,mode] [,variable] [,timeout])</p> <p>baudrate: 串口波特率 9600 19200 4800 115200 38400(缺省) 57600 128000 256000</p> <p>databits: 数据位数 8</p> <p>stopbits: 停止位，只能设置 0/1/2</p> <p>parity: 是否校验:</p> <table border="1"> <tr> <th>值</th><th>描述</th></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> <p>port: 串口 PORT 编号 0-1，参见 PORT 描述，不同的控制器不一样</p> <p>mode: 协议</p> <table border="1"> <tr> <th>值</th><th>描述</th></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> <p>variable: 寄存器选择，0-VR，1-TABLE，2-系统 MODBUS 寄存器</p> <table border="1"> <tr> <th>值</th><th>描述</th></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 不会有问題</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，</td></tr> </table>	值	描述	0 (缺省)	无校验	1	奇校验	2	偶校验	值	描述	0	RAW 数据模式，无协议，此时可以使用 GET #, PRITNT #	4 (缺省)	MODBUS 从端 (16 位整数)	14	MODBUS 主端 (16 位整数)	15	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)	值	描述	0	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 不会有问題	1	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，
值	描述																								
0 (缺省)	无校验																								
1	奇校验																								
2	偶校验																								
值	描述																								
0	RAW 数据模式，无协议，此时可以使用 GET #, PRITNT #																								
4 (缺省)	MODBUS 从端 (16 位整数)																								
14	MODBUS 主端 (16 位整数)																								
15	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)																								
值	描述																								
0	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 不会有问題																								
1	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
	2（缺省）	系统 MODBUS 寄存器,此时 VR 与 MODBUS 寄存器是两片独立区间
	3	VR_INT 模式, 此时一个 VR_INT 映射到两个 MODBUS_REG
	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	<p>例一 mode0 RAW 模式</p> <pre>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</pre> <p>松下 A6 伺服编码器读取参照第十三章<a href="#">简易例程</a></p> <p>例二 MODBUS 通讯设置</p> <pre>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</pre> <p>具体例程参照 MODBUSM_DES 指令例程</p> <p>例三 直接字符命令方式</p> <pre>setcom(38400, 8,1,0,0,15) '设置串口 0 为直接字符串命令方式</pre> <p>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。</p> <div><div><input type="checkbox"/> HEX发送 <input checked="" type="checkbox"/> 发送新行</div><div>字符串输入框: <input type="text" value="发送"/></div><div>units=100</div></div> <p>发送前 UNITS=10000 发送后 UNITS=100</p> <p>一定要换行发送, 否则控制器报错</p>	

	<div> <div>×</div> <div>Online command line not ended over time.</div> </div>	
	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	<a href="#">ADDRESS</a> , <a href="#">PROTOCOL</a> , <a href="#">MODBUSM_DES</a> , <a href="#">PORT</a>	

**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- 255，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	<a href="#">SETCOM</a> ， <a href="#">PORT</a> ， <a href="#">PROTOCOL</a>

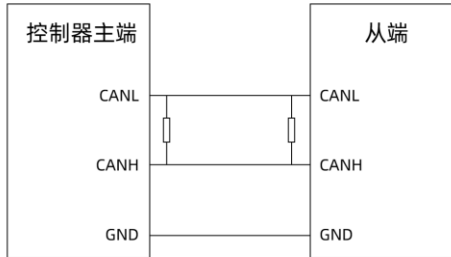
## COM UNUSED -- 指定串口

类型	系统参数
描述	按位指定串口是否被 ZBASIC 使用。  bit0: 串口 1, 值设置 1 表示 ZBASIC 不使用 bit1: 串口 2, 值设置 1 表示 ZBASIC 不使用
语法	COM_UNUSED = value

适用控制器	5 系列控制器以上
例子	COM_UNUSED=1            'ZBASIC 不使用 RS232 串口
相关指令	<a href="#">SETCOM</a> , <a href="#">PORT</a> , <a href="#">PROTOCOL</a>

## 13.2. CAN 通讯指令

### CAN -- CAN 通讯

类型	系统指令										
描述	<p>直接通过 CAN 总线收发数据。</p> <p>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)。</p> <p>较新的固件版本才支持这个功能，如不支持请联系厂家。</p> <p>接线如下图所示：</p>  <p>CANL - CANL CANH - CANH</p> <p>CANL 和 CANH 的首尾两端分别连接 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。</p>										
语法	<p>CAN(channel, function, tablenum)</p> <p>channel: CAN 通道, 0 表示第一个通道, -1 表示缺省通道</p> <p>function: 功能号</p> <table border="1"> <thead> <tr> <th>值</th><th>描述</th></tr> </thead> <tbody> <tr> <td>6</td><td>接收, 没有数据时, identifier&lt;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&lt;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> </tbody> </table> <p>tablenum: 数据存储的 TABLE 位置</p> <p>6、7 模式时依次存储：</p>	值	描述	6	接收, 没有数据时, identifier<0	7	发送	16 需要升级固件	带扩展支持接收, 没有数据时, identifier<0	17 需要升级固件	发送扩展数据, 普通数据使用 7 发送
值	描述										
6	接收, 没有数据时, identifier<0										
7	发送										
16 需要升级固件	带扩展支持接收, 没有数据时, identifier<0										
17 需要升级固件	发送扩展数据, 普通数据使用 7 发送										



	<p><b>identifier:</b> 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。bit11 表示是否远程帧。</p> <p><b>bytes:</b> 数据区域字节数, 最多为 8。</p> <p><b>data:</b> 数据区域, 字节 (0-FF)。</p> <p>16、17 模式时依次存储:</p> <p><b>identifier:</b> 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。bit11 表示是否远程帧。</p> <p><b>identifier extend:</b> 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。</p> <p><b>bytes:</b> 数据区域字节数, 最多为 8。</p> <p><b>data:</b> 数据区域, 字节 (0-FF)。</p> <p>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          低 18 位为 110001001100000000, 对应 10 进制 201472          所以使用 CAN17 模式时, identifier=1595, Identifier extend=201472</p>
适用控制器	通用
例子	<p>例一</p> <p>'发送端:</p> <p>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</p> <p>CAN(0,7,0) '发送</p> <p>'接收端:</p> <p>CANIO_ADDRESS=1 '设置为非主控, 此参数设置一次即可</p> <p>CAN(0,6,0) '接收</p> <p>?TABLE(0)</p> <p>例二</p> <p>'发送端</p> <p>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</p> <p>CAN(0,17,0) '发送</p> <p>'接收端:</p> <p>CANIO_ADDRESS=1 '设置为非主控, 此参数设置一次即可</p> <p>CAN(0,16,0) '接收</p> <p>?TABLE(0)</p>
相关指令	<a href="#">CANIO_ADDRESS</a> , <a href="#">CANIO_STATUS</a> , <a href="#">CANIO_ENABLE</a>

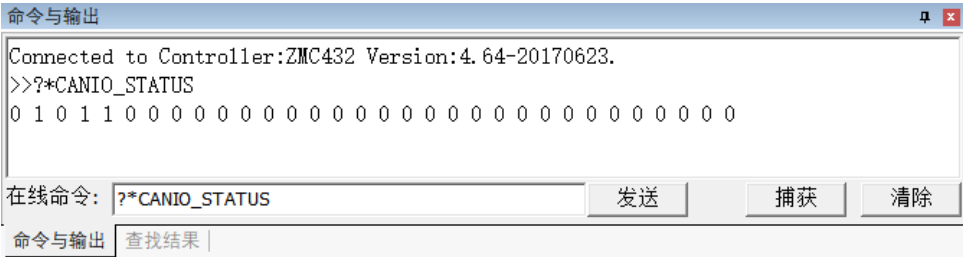
## CANIO\_ADDRESS -- CAN 通讯设置

类型	系统参数										
描述	<p>控制器上 CAN 的 CANID 和速度的设置。</p> <p>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。</p> <p>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。</p> <table border="1"> <tr> <th>高 8 位值</th><th>CAN 速度</th></tr> <tr> <td>0</td><td>500 Kbps（缺省值）</td></tr> <tr> <td>1</td><td>250Kbps</td></tr> <tr> <td>2</td><td>125Kbps</td></tr> <tr> <td>3</td><td>1 Mbps</td></tr> </table> <p>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。</p>	高 8 位值	CAN 速度	0	500 Kbps（缺省值）	1	250Kbps	2	125Kbps	3	1 Mbps
高 8 位值	CAN 速度										
0	500 Kbps（缺省值）										
1	250Kbps										
2	125Kbps										
3	1 Mbps										
语法	CANIO_ADDRESS = value										
适用控制器	通用										
例子	<p>CANIO_ADDRESS = 32      '设置主端，CAN 波特率 500Kbps</p> <p>CANIO_ADDRESS = 32+ 256      '设置主端，CAN 波特率为 250Kbps</p> <p>CANIO_ADDRESS = 32+ 512      '设置主端，CAN 波特率为 125Kbps</p> <p>CANIO_ADDRESS = 32+ 768      '设置主端，CAN 波特率为 1Mbps</p> <p>CANIO_ADDRESS = 1      '设置 CANID 为 1，此时为从端，500 Kbps，不能连接 IO 板</p> <p>CANIO_ADDRESS = 1 +256      '设置 CANID 为 1，此时为从端，250 Kbps，作为 ZCAN 从站使用</p> <p>CANIO_ADDRESS = 1 +512      '设置 CANID 为 1，此时为从端，125 Kbps，作为 ZCAN 从站使用</p> <p>CANIO_ADDRESS = 1 +768      '设置 CANID 为 1，此时为从端，1 Mbps，作为 ZCAN 从站使用</p> <p>CANIO_ADDRESS = 3      '设置 CANID 为 3，此时为从端，500 Kbps，作为 ZCAN 从站使用</p> <p>CANIO_ADDRESS = 8 +256      '设置 CAN ID 为 8，此时为从端，250 Kbps，作为 ZCAN 从站使用</p> <p>CANIO_ADDRESS = 16 +512      '设置 CAN ID 为 16，此时为从端，125 Kbps，作为 ZCAN 从站使用</p> <p>CANIO_ADDRESS = 31 +768      '设置 CAN ID 为 31，此时为从端，1Mbps，作为 ZCAN 从站使用</p> <p>具体使用参考第十七章的 <a href="#">CAN 通讯例子</a></p>										
相关指令	<a href="#">CANIO_ENABLE</a> ， <a href="#">CAN</a> ， <a href="#">CANIO_STATUS</a>										

## 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	<a href="#">CANIO_ADDRESS</a>

## 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: 该扩展模块的拨码 ID
适用控制器	通用
例子	<p>例一</p> <p><b>?*CANIO_STATUS</b>                    '输出所有 IO 板的状态</p> <p>打印结果如下图：表示接入了三个扩展模块，拨码 ID 的组合值分别为 1，3，4。</p>  <p>例二</p> <p><b>IF CANIO_STATUS(1)=0 THEN</b>   '判断 IO 板的连接状态</p> <p>    <b>PRINT</b>   "扩展模块 1 没有连接好"</p> <p><b>ENDIF</b></p>
相关指令	<a href="#">CAN</a> , <a href="#">CANIO_ENABLE</a> , <a href="#">CANIO_INFO</a> <a href="#">CANIO_ADDRESS</a> -- <a href="#">CAN 通讯设置</a>

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	<div>CANIO_INFO(canid, isel [, moduleid]) canid: 扩展模块的拨码 ID isel: 读取的参数<table><tr><td>0</td><td>正运动</td></tr><tr><td>1</td><td>DEVICE, 设备编号</td></tr><tr><td>2</td><td></td></tr><tr><td>3</td><td></td></tr><tr><td>4</td><td></td></tr><tr><td>IO 的个数</td><td>描述</td></tr><tr><td>10</td><td>IN 个数, 整体的</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>增加</td><td></td></tr><tr><td>16</td><td>模块数</td></tr><tr><td>17</td><td>子模块的类型号, 必须后面带 moduleid id</td></tr><tr><td>预留</td><td></td></tr><tr><td>20</td><td>子模块输入</td></tr><tr><td>21</td><td>子模块输出</td></tr><tr><td>22</td><td>子模块 AIN</td></tr><tr><td>23</td><td>子模块 AOUT</td></tr></table> moduleid: ZMIO 子模块的信息, 部分 isel 参数选择需要填这个参数</div>	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
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																																				
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	<a href="#">CAN</a> , <a href="#">CANIO_ENABLE</a> , <a href="#">CANIO_STATUS</a> <a href="#">CANIO_ADDRESS</a> -- <a href="#">CAN 通讯设置</a>																																				

13.3. 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


	<p>语法 2: GET #PORT, ARRAY[(startindex)] [,maxchars]</p> <p>语法 3: charesget = GET #PORT, VARIABLE</p> <p>语法 4: charesget = GET #PORT, ARRAY[(startindex)] [,maxchars]</p> <p>port: 通道号</p> <p>variable: 存放的变量名</p> <p>startindex: 存放数组的起始地址</p> <p>maxchars: 存放的最多数量</p> <p>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。</p> <p>语法 3、4 会返回读取到的字节数。</p> <p>20150522 版本以前只支持语法 1。</p> <p>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。</p> <p>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。</p> <p>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。</p>
适用控制器	通用
例子	<p>例一</p> <pre>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</pre> <p>例二</p> <pre>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 &gt; 0 THEN     ARRAY1(CHARES) = 0        '设置结束 0     PRINT ARRAY1               '字符串打印出来 ENDIF</pre>
相关指令	<a href="#">PORT</a> , <a href="#">SETCOM</a> , <a href="#">PRINT #</a>

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	<p>OPEN #PORT, "mode", portnum [, ipaddress]</p> <p>port: 通讯通道，参见 PORT 描述，选择自定义网络通道</p> <p>mode: 讯主从，“TCP_CLIENT” - 从，“TCP_SERVER” - 主，“UDP_CLIENT” - UDP 从，“UDP_SERVER” - UDP 主</p>

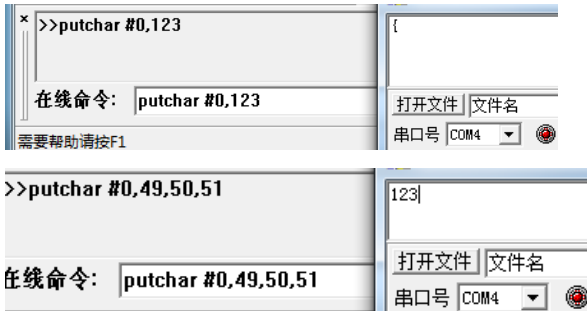
	<p>portnum: TCP 或 UDP 端口号, 主端为本地端口号, 从端为对方端口号</p> <p>ipaddress: 对方 IP 地址, 字符串, 从端的时候要提供</p> <p>UDP_SERVER 必须先接收对方的数据, 才能发送回数据(除非用 PORT_TARGET 先强制指定对方)。</p> <p>UDP_CLIENT 本地端口号随机, 必须先发送给对方, 对方才能知道端口号, 此模式时不是指定对方的包会丢弃掉。</p> <p>UDP 自定义通讯需要 4 系列控制器 20170628 以上固件版本; XPLC 系列控制器 20170702 以上固件版本。</p>
适用控制器	通用
例子	<p>例一</p> <p><b>OPEN #11, "TCP_SERVER", 10</b> '主端设置</p> <p><b>OPEN #10, "TCP_CLIENT", 10, "192.168.1.112"</b> '从端设置</p> <p>例二</p> <p><b>OPEN #10, "UDP_SERVER", 1000</b> 'UDP 主端设置</p> <p><b>OPEN #11, "UDP_CLIENT", 60000, "192.168.0.120"</b> 'UDP 从端设置</p> <p>详细例程参考<a href="#">自定义网口通讯</a></p>
相关指令	<a href="#">PORT</a> , <a href="#">PORT_TARGET</a> , <a href="#">SETCOM</a> , <a href="#">PRINT #</a>

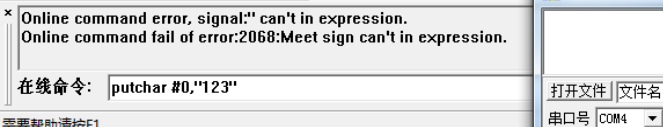

## PRINT # -- 输出字符串

类型	系统指令
描述	<p>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。</p> <p>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:</p> 

		
		
	<p>PRINT #发送数组为 ASCII 码，遇 0 停止，如下所示：</p>	
		
语法	PRINT #PORT, "字符串" port: 通道号	
适用控制器	通用	
例子	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 <b>PRINT</b> #0,VAR                   '从通道 0 输出数组数据	
相关指令	<a href="#">PUTCHAR #</a> , <a href="#">GET #</a> , <a href="#">PORT</a> , <a href="#">SETCOM</a>	

## PUTCHAR # -- 输出字符

类型	系统指令
描述	<p>从 <b>RAW</b> 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。</p> <p>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：</p> 

	 <p>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：</p> 												
语法	<p>PUTCHAR #PORT, 字符</p> <p>port: 通道号</p> <p>PUTCHAR #PORT, ARRAY(INDEX, NUMES)</p> <p>port: 通道号</p> <p>index: 开始输出的位置</p> <p>numes: 输出的字节个数, 二进制方式</p>												
适用控制器	通用												
例子	<table border="0"> <tr> <td>DIM VAR1, ARRAY1(10)</td> <td>'定义数组</td> </tr> <tr> <td>VAR1 = \$FE</td> <td>'数组 1 赋值</td> </tr> <tr> <td>ARRAY1 = "ABCDEFGHJI"</td> <td>'数组 2 赋值</td> </tr> <tr> <td>SETCOM(38400,8,1,0,0,0)</td> <td>'开启 RAW 方式</td> </tr> <tr> <td>PUTCHAR #0, VAR1</td> <td>'从通道 0 输出数组 1 数据</td> </tr> <tr> <td>PUTCHAR #0, ARRAY1</td> <td>'从通道 0 输出数组 2 数据</td> </tr> </table>	DIM VAR1, ARRAY1(10)	'定义数组	VAR1 = \$FE	'数组 1 赋值	ARRAY1 = "ABCDEFGHJI"	'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
DIM VAR1, ARRAY1(10)	'定义数组												
VAR1 = \$FE	'数组 1 赋值												
ARRAY1 = "ABCDEFGHJI"	'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	<a href="#">PORT</a> , <a href="#">SETCOM</a> , <a href="#">GET #</a>												

## 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	<pre>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</pre>
相关指令	<a href="#">OPEN # CANIO_ADDRESS -- CAN 通讯设置</a>

## 13.4. 打印输出指令

### PRINT -- 打印信息

类型	打印输出函数
描述	<p>打印信息到 PC 端 ZDevelop 软件的输出窗口。</p> <p>别名：？</p> <p>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。</p>
语法	<pre>PRINT expression, "string" 或 ? expression, "string"</pre> <p>Expression: 有效表达式</p> <p>*SET: 打印所有参数值</p> <p>*TASK: 打印任务信息</p> <p>任务正常时只打印任务状态</p> <p>任务出错时还会打印出错误任务号，具体错误行</p> <p>*MAX: 打印所有规格参数</p> <p>*FILE: 打印程序文件信息</p> <p>*SETCOM: 打印当前串口的配置信息</p> <p>*BASE: 打印当前任务的 BASE 列表（140123 以后版本支持）</p> <p>*数组名: 打印数组的所有元素，数组长度不能太长</p> <p>*参数名: 打印一个所有轴的单个参数</p> <p>?*ETHERCAT: 打印 EtherCAT 总线连接设置状态</p> <p>?*RTEX: 打印 Rtex 总线连接设置状态</p> <p>?*FRAME: 打印机械手参数，需要 161022 及以上固件支持</p> <p>?*SLOT: 打印出控制器槽位口信息（RTEX 口，EtherCAT 口）</p> <p>?*PORT: 打印所有 PORT 通讯口</p>
适用控制器	通用
例子	<p>例一</p> <p>在线命令输入</p> <p>&gt;&gt;<b>PRINT</b> 1+2</p> <p>输出：3</p> <p>例二</p>

	<p>在线命令输入</p> <pre>&gt;&gt;PRINT *task      '打印所有任务状态</pre> <p>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.</p> <p>例三</p> <p>在线命令输入</p> <pre>&gt;&gt; ?*mpos</pre> <p>输出: 21872.400 0 0 0 0 0 0 0 0 0 0</p>
相关指令	<a href="#">TRACE</a>

## ERRSWITCH -- 信息输出设置

类型	系统参数												
描述	调试输出开关，控制 <b>TRACE WARN ERROR</b> 调试输出语句是否真的输出信息。												
语法	<p>ERRSWITCH=switch</p> <p>switch: 调试输出的开关</p> <table border="1"> <thead> <tr> <th>值</th><th>描述</th></tr> </thead> <tbody> <tr> <td>0</td><td>TRACE WARN ERROR 指令全部不输出</td></tr> <tr> <td>1</td><td>只输出 ERROR 指令</td></tr> <tr> <td>2</td><td>输出 WARN ERROR 指令</td></tr> <tr> <td>3</td><td>TRACE WARN ERROR 指令全部输出</td></tr> <tr> <td>4</td><td>TRACE WARN ERROR 指令全部输出，以及运动指令监控</td></tr> </tbody> </table>	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
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	<p>例一</p> <pre><b>ERRSWITCH</b> = 3</pre> <p>'打印全输出，此时 WARN（报警）、ERROR（错误）、TRACE 信息都会输出</p> <p>例二</p> <pre><b>ERRSWITCH</b> = 0</pre> <p>'不输出，控制器错误信息和报警信息都不输出</p> <pre>TRACE DPOS(0)</pre> <p>'此时无法用 trace 打印出轴 0 的 dpos</p> <pre>?DPOS(0)</pre> <p>'可以用 print 打印，结果，0</p> <p>例三</p> <pre><b>ERRSWITCH</b> = 4</pre> <p>'输出所有信息，以及运动指令监控</p> <pre>&gt;&gt;MOVE(111)</pre> <pre>MOVE(111)AXIS(0)TASK(23)</pre> <p>'打印当前操作的轴及运动的任务</p>												
相关指令	<a href="#">TRACE</a> ， <a href="#">WARN</a> ， <a href="#">ERROR</a>												

## 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 <b>TRACE</b> "DPOS(0) =" DPOS(0) 打印结果 0
相关指令	<a href="#">ERRSWITCH</a> , <a href="#">PRINT</a> , <a href="#">WARN</a> , <a href="#">ERROR</a>

## 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	<a href="#">ERRSWITCH</a> , <a href="#">PRINT</a> , <a href="#">TRACE</a> , <a href="#">ERROR</a>

## 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	<a href="#">ERRSWITCH</a> , <a href="#">PRINT</a> , <a href="#">TRACE</a> , <a href="#">WARN</a>

## 13.5. 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	<p>PORT (portnum) portnum: 通道号</p> <p>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。</p> <p><b>ZMC00x 系列</b></p> <table border="1"> <tr> <th>通道号</th><th>协议</th></tr> <tr> <td>0</td><td>串口 A</td></tr> <tr> <td>1</td><td>串口 B</td></tr> </table> <p><b>ZMC1-2xx 系列</b>，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。</p> <table border="1"> <tr> <th>通道号</th><th>协议</th></tr> <tr> <td>0</td><td>RS232 串口</td></tr> <tr> <td>1</td><td>RS485 串口</td></tr> <tr> <td>2</td><td>以太网接口，连接 1</td></tr> <tr> <td>3</td><td>以太网接口，连接 2</td></tr> <tr> <td>10</td><td>自定义网络通讯通道 1</td></tr> <tr> <td>11</td><td>自定义网络通讯通道 2</td></tr> </table> <p><b>ZMC3xx 系列</b>：带 3 个串口。</p> <table border="1"> <tr> <th>通道号</th><th>协议</th></tr> <tr> <td>0</td><td>RS232 串口</td></tr> <tr> <td>1</td><td>RS485 串口</td></tr> <tr> <td>2</td><td>RS422 串口</td></tr> <tr> <td>3</td><td>以太网接口，连接 1</td></tr> <tr> <td>4</td><td>以太网接口，连接 2</td></tr> <tr> <td>10</td><td>自定义网络通讯通道 1</td></tr> <tr> <td>11</td><td>自定义网络通讯通道 2</td></tr> </table> <p><b>ZMC4xx 系列</b>：带 2 个串口。</p> <table border="1"> <tr> <th>通道号</th><th>协议</th></tr> <tr> <td>0</td><td>RS232 串口</td></tr> <tr> <td>1</td><td>RS485 串口</td></tr> <tr> <td>2</td><td>以太网接口，连接 1</td></tr> <tr> <td>3</td><td>以太网接口，连接 2</td></tr> <tr> <td>10</td><td>自定义网络通讯通道 1</td></tr> <tr> <td>11</td><td>自定义网络通讯通道 2</td></tr> <tr> <td>20</td><td>网络控制器互联通道，不支</td></tr> </table>	通道号	协议	0	串口 A	1	串口 B	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	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	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
通道号	协议																																																				
0	串口 A																																																				
1	串口 B																																																				
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																																																				
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																																																				
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	<div>查看控制器通道</div> <div>&gt;&gt;?*PORT</div> <div>Port:0-COM.</div> <div>Port:1-COM.</div> <div>Port:2-ETH.</div> <div>Port:3-ETH.</div> <div>Port:4-ETH.</div> <div>Port:5-ETH.</div> <div>Port:6-ETH.</div> <div>Port:7-ETH.</div> <div>Port:10-ECUSTOM.</div> <div>Port:11-ECUSTOM.</div> <div>Port:12-ECUSTOM.</div> <div>Port:13-ECUSTOM.</div> <div>Port:14-ECUSTOM.</div> <div>Port:15-ECUSTOM.</div> <div>Port:20-CONNECT.</div> <div></div> <div>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。</div>	
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	<p>返回当前通道的状态，串口总是返回 1。</p> <p>MODBUS_TCP 链接上以后，PORT_STATUS 会变为 1。</p> <p>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。</p> <p>新固件版本提供此功能。</p>					
语法	<p>VAR1 = PORT_STATUS(port)</p> <p>port: 通道号</p> <table><tr><td>值</td><td>协议</td></tr><tr><td>0</td><td>没有连接</td></tr></table>		值	协议	0	没有连接
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	<a href="#">PORT</a> , <a href="#">SETCOM</a> , <a href="#">ADDRESS</a>			

## 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。 <b>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。</b>
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	<a href="#">PORT</a>

## 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 <table border="1"> <thead> <tr> <th>值</th><th>协议</th></tr> </thead> <tbody> <tr> <td>0</td><td>RAW 数据格式，无协议</td></tr> <tr> <td>3</td><td>MODBUS 协议，控制器为从端（缺省）</td></tr> <tr> <td>14</td><td>MODBUS 协议，控制器为主端</td></tr> <tr> <td>15</td><td>直接命令执行方式</td></tr> </tbody> </table>	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
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	<a href="#">PORT</a> , <a href="#">SETCOM</a> , <a href="#">ADDRESS</a>										

## ETH\_MODE -- 网口模式设置

类型	通道参数
描述	网口模式设置，数组类型，每个网口独立设置。
语法	ETH_MODE(isel) = imode

	<p>isel: 0- 第一个网口, 1-第二个网口(EtherCAT)</p> <p>imode: 0 - 非 eth 模式, 传统模式, 兼容性好, 抗干扰稍差</p> <p>1 - eth 模式, 抗干扰性好, 多个链接时需要配合 PC 上防掉线 DLL, 标准 DLL 只能链接一个 (有极少数触摸屏或 EtherCAT 驱动器不兼容此模式)</p> <p>2 - 自动模式(新固件缺省值), 如果有检测到网络没有连上, 会自动在 0/1 模式中切换</p> <p>20170720 以上带 ETH 的固件版本支持。</p>
适用控制器	通用
相关指令	<a href="#">PORT</a>

## 13.6. 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。	
	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
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 <b>MODBUS_BIT</b> (100) = 1      'bit100 赋值 1 VAR = <b>MODBUS_BIT</b> (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“ <a href="#">MODBUS 寄存器</a> ”。		
	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 = MODBUS_IEEE(100)    'ieee100 赋值给变量 VAR		
相关指令	<a href="#">MODBUS_REG</a> ， <a href="#">MODBUS_LONG</a> ， <a href="#">MODBUS_STRING</a>		

## MODBUS\_LONG -- 字寄存器-32 位整型

类型	MODBUS 字寄存器		
描述	<p>修改或者读取字寄存器，32 位整型。触摸屏一般叫 4x 寄存器。</p> <p>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“<a href="#">MODBUS 寄存器</a>”。</p> <p>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。</p>		
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 <b>MODBUS_LONG</b> (100) = 100        'long100 赋值 100 VAR = <b>MODEBUS_LONG</b> (100)    'long100 赋值给 VAR		
相关指令	<a href="#">MODBUS_REG</a> , <a href="#">MODBUS_IEEE</a> , <a href="#">MODBUS_STRING</a>		



## 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“ <a href="#">MODBUS 寄存器</a> ”。		
	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	<a href="#">MODBUS_IEEE</a> ， <a href="#">MODBUS_LONG</a> ， <a href="#">MODBUS_STRING</a>		

## 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“ <a href="#">MODBUS 寄存器</a> ”。		
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) <b>MODBUS_STRING</b> (0, 8) = "abc"    'string0 开始保存字符串，共 8 字节 print <b>MODBUS_STRING</b> (0, 8)        '打印保存的字符串，打印结果，abc ARR = <b>MODBUS_STRING</b> (0,8)        '字符换赋值给数组		
相关指令	<a href="#">MODBUS_REG</a> ， <a href="#">MODBUS_LONG</a> ， <a href="#">MODBUS_IEEE</a>		

## MODBUSM\_DES -- modbus 通讯连接

类型	通讯指令
描述	<p>设置或读取 MODBUS 主端的对方。</p> <p>通讯等待时只会阻塞当前这个任务，不影响其他任务。</p> <p>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。</p>
语法	<p>MODBUSM_DES (address[,port],[timer],[resendset])</p> <p>ADDRESS1 = MODBUSM_DES([port])</p> <p>address: 对端的 modbus 协议站号</p> <p>port: 当前 modbus 主通讯的 port 号</p> <p>timer: 消息超时时间设置，缺省 1000ms。</p> <p>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。</p> <p>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。</p> <p>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。</p>
适用控制器	通用
例子	<p>例一 多主端-多从端通讯</p> <p>SETCOM(38400,8,1,0,0,14,2,1000) '设置串口 0 为 modbus 主端，通讯等待 1s</p> <p>SETCOM(38400,8,1,0,1,14,2,1000) '设置串口 1 为 modbus 主端，通讯等待 1s</p> <pre> 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 &lt;&gt; 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 &lt;&gt; 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   </pre>

	<pre> MODBUSM_REGGET(60,10,60) '对端寄存器复制到本地 WAIT UNTIL MODBUSM_STATE &lt;&gt; 1 '等待消息结束 ELSE <b>MODBUSM_DES</b>(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 &lt;&gt; 1 '等待消息结束 ENDIF ?"通道 1 状态=", MODBUSM_STATE '打印通讯状态 ENDIF WEND </pre>
相关指令	<a href="#">SETCOM</a> , <a href="#">PROTOCOL</a> , <a href="#">PORT</a> , <a href="#">MODBUSM_DES2</a>

## MODBUSM\_DES2 -- 控制器间网口通讯

类型	通讯指令
描述	<p>控制器间网口通讯，也可作为 MODBUS_TCP 主端通讯。  ?<b>*PORT</b> 可以显示出当前可用的网络链接通道。</p> <pre> &gt;&gt;?*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. </pre> <p>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。</p> <p>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。</p>
语法	<p><b>MODBUSM_DES2</b> (id,port,"desipaddress",[timer],[resendset])</p> <p>id: 对方控制器的 MODBUS 从端 ID, 缺省 1</p> <p>port: 支持两种模式，?<b>*PORT</b> 确认通道号及模式  ETH 时，作为 MODBUS_TCP 主端通道</p>

	<p>CONNECT 时, 做为控制器互联通道</p> <p>desipaddress: 字符串, 对方控制器的 IP 地址</p> <p>timer: 消息超时时间设置, 缺省 1000ms</p> <p>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。</p> <p>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。</p> <p>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。</p>
适用控制器	4 系列产品, 20170117 及以上固件版本支持
例子	<p>例一 MODBUS 主端通讯连接建立</p> <p><b>MODBUS_TCP 主端通讯不用考虑丢包。</b></p> <p><b>MODBUSM_DES2(1,4,"192.168.0.12")</b> '按站号和 IP 与从端通讯使用控制器通道 4, ?*port 确认</p> <pre> 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) &lt;&gt; 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 </pre> <p>例二 控制器间通讯连接建立</p> <p><b>网络环境不好时, 互连通讯有很小的丢包可能。</b></p> <p><b>MODBUSM_DES2(\$fe,20,"192.168.0.25",10)</b> '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</p> <pre> MODBUSM_REGSET(0,10,0) '本地寄存器复制到对端 WAIT UNTIL MODBUSM_STATE &lt;&gt; 1 '等待消息结束 IF MODBUSM_STATE&lt;&gt;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 ENDIF WAIT UNTIL MODBUSM_STATE &lt;&gt; 1 '等待消息结束 IF MODBUSM_STATE&lt;&gt;0 THEN   MODBUSM_REGGET(20,10,20) '出错重发一次 </pre>

	ENDIF END
相关指令	<a href="#">ADDRESS</a> , <a href="#">PORT</a>

## MODBUSM\_STATE -- modbus 通讯状态

类型	通讯状态										
描述	<b>MODBUS 主通讯状。</b> <table border="1"> <thead> <tr> <th>值</th><th>描述</th></tr> </thead> <tbody> <tr> <td>0</td><td>状态正常</td></tr> <tr> <td>1</td><td>等待应答中</td></tr> <tr> <td>2</td><td>等待超时</td></tr> <tr> <td>3</td><td>应答错误</td></tr> </tbody> </table>	值	描述	0	状态正常	1	等待应答中	2	等待超时	3	应答错误
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	<a href="#">PROTOCOL</a> , <a href="#">PORT</a> , <a href="#">SETCOM</a>										

## 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	<a href="#">ADDRESS</a> , <a href="#">PROTOCOL</a> , <a href="#">PORT</a> , <a href="#">SETCOM</a>

## 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: 本地系统 MODBUS 寄存器起始编号
适用控制器	通用
例子	<p>台达绝对值编码器读取</p> <pre> 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 <b>MODBUSM_REGGET(98,2,300)</b> TICKS = 1000  WHILE (MODBUS_LONG(300) AND TICKS &gt; 0) '等到 P0-49 变成 0 或 1 秒超时，即更新完成或者不成功     <b>MODBUSM_REGGET(98,2,300)</b> WEND  IF TICKS &lt; 0 THEN     PRINT "伺服更新不成功"     flag_abs = 1     RETURN ENDIF  <b>MODBUSM_REGGET(100,6,310)</b> 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 </pre>

	<pre> ENDIF  IF flag_abs = 0 THEN  '正确     dpos(0) = -total_pul/units(0)    '测试出来的脉冲个数是反的，取反还原坐标 ENDIF END </pre>
相关指令	<a href="#">ADDRESS</a> , <a href="#">PROTOCOL</a> , <a href="#">PORT</a> , <a href="#">SETCOM</a>

## 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	<pre> 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 </pre>
适用控制器	通用
例子	<b>MODBUSM_3XGET(0,10,0)</b> '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	<a href="#">ADDRESS</a> , <a href="#">PROTOCOL</a> , <a href="#">PORT</a> , <a href="#">SETCOM</a>

## 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	<pre> 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 </pre>
适用控制器	通用
例子	<b>MODBUSM_BITSET (0,10,0)</b> '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	<a href="#">ADDRESS</a> , <a href="#">PROTOCOL</a> , <a href="#">PORT</a> , <a href="#">SETCOM</a>

## 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: 本地系统 MODBUS 寄存器起始编号
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 复制到本地寄存器 0-9
相关指令	<a href="#">ADDRESS</a> , <a href="#">PROTOCOL</a> , <a href="#">PORT</a> , <a href="#">SETCOM</a>

## 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: 本地系统 MODBUS 寄存器起始编号
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 复制到本地寄存器 0-9
相关指令	<a href="#">ADDRESS</a> , <a href="#">PROTOCOL</a> , <a href="#">PORT</a> , <a href="#">SETCOM</a>

## 13.7. 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	<a href="#">SEND_CMD</a>



## SEND\_CMD -- send 命令

类型	通讯指令
描述	<p>主控制器给从控制器发送 ZMC_DIRECTCOMMAND 命令，结果通过 SEND_RESULT 查看。</p> <p>发送 BASIC 内容：cmdstring (参数列表)</p> <p>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。</p>
语法	<p>SEND_CMD(cmdstring,可选参数列表)</p> <p>cmdstring: 命令字符串</p> <p>可选参数列表: 个数可变，不带参数的时候，不添加括号</p>
适用控制器	4 系列以上控制器支持，支持版本号 20170618
例子	<p>SEND_CMD("MOVE",DIS1,DIS2,DIS3)</p> <p>SEND_CMD("MOVEABS",DIS1)</p>
相关指令	<a href="#">SEND_RESULT</a>

## SEND\_CMDAXIS -- send 命令

类型	通讯指令
描述	<p>主控制器给从控制器发送 ZMC_DIRECTCOMMAND 命令，结果通过 SEND_RESULT 查看。</p> <p>发送 BASIC 内容：cmdstring (参数列表)，AXIS(iaxis)</p> <p>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。</p>
语法	<p>SEND_CMDAXIS (cmdstring,iaxis,可选参数列表)</p> <p>cmdstring: 命令字符串</p> <p>iaxis: 轴号</p> <p>可选参数列表: 个数可变，不带参数的时候，不添加括号</p>
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	<a href="#">SEND_RESULT</a> ， <a href="#">SEND_CMD</a>

## 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

	<p>查看。 发送 BASIC 内容: cmdstring (参数列表)=value</p> <p>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。</p>
语法	<p>SEND_ASSIGN (cmdstring,value,可选参数列表)</p> <p>cmdstring: 命令字符串</p> <p>value: 赋值的内容</p> <p>可选参数列表: 个数可变, 不带参数的时候, 不添加括号</p>
适用控制器	4 系列以上控制器支持, 支持版本号 20170618
例子	<p>SEND_ASSIGN("DPOS",0,0) '生成 DPOS(0)=0</p> <p>SEND_ASSIGN("DPOS(1)",0,0) '生成 DPOS(1)=0</p>
相关指令	<a href="#">SEND_RESULT</a>

## SEND\_QUERY -- send 命令

类型	通讯指令
描述	<p>主控制器给从控制器发送 ZMC_DIRECTCOMMAND 命令, 结果通过 SEND_RESULT 查看。</p> <p>发送 BASIC 内容: cmdstring(参数列表)。</p> <p>不带参数的时候, 不添加括号。</p> <p>接收的内容根据 SEND_QUERYSET 的设置填到 TABLE 里面。</p> <p>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。</p>
语法	<p>SEND_QUERY (cmdstring,可选参数列表)</p> <p>cmdstring: 命令字符串</p> <p>可选参数列表: 个数可变, 不带参数的时候, 不添加括号</p>
适用控制器	4 系列以上控制器支持, 支持版本号 20170618
例子	<p>SEND_QUERYSET(0,1)</p> <p>SEND_QUERY("?dpos",0) 'table(0)保存控制器 DPOS(0)的内容</p> <p>SEND_QUERY("?REMAIN_BUFFER(1)AXIS(0)",0) '发送字符串内容 ("?REMAIN_BUFFER(1) AXIS(0)")</p> <p>SEND_QUERYSET(0,2)</p> <p>SEND_QUERY("?dpos(0),dpos(1)") '从控制器会返回两个数据</p>
相关指令	<a href="#">SEND_RESULT</a> , <a href="#">SEND_QUERYSET</a>

## 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	<a href="#">SEND_RESULT</a> , <a href="#">SEND_QUERY</a>

## 13.8. 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	<a href="#">MODBUSM_STATE</a> , <a href="#">SEND_PERCENT</a>

### 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	<a href="#">MODBUSM STATE</a> , <a href="#">SEND PERCENT</a>

## 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	<a href="#">MODBUSM STATE</a> , <a href="#">SEND PERCENT</a>

## 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	<a href="#">MODBUSM STATE</a> , <a href="#">SEND_PERCENT</a>

## SEND\_PERCENT -- 查询指令进度

类型	通讯指令
描述	<b>SEND</b> 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 <b>0-100</b> 。  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	<a href="#">SEND_ZAR</a> , <a href="#">SEND_FLASH</a> , <a href="#">SEND_FILE</a>

## 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	<a href="#">SEND_ZAR</a> , <a href="#">CONTROL</a>

## 第十四章 系统相关指令

所有的日期时间参数或指令，LOCK 后不能修改。

### 14.1. 控制器加密指令

#### APP\_PASS -- 密码

类型	系统指令
描述	<p>控制器应用密码。</p> <p>下载 ZAR 包时可以选择校验这个密码，校验错误将不能下载。</p> <p>LOCK 后不能修改 APP_PASS。</p> <p><b>APP_PASS 采用不可逆算法加密，一旦忘记，将无法获知。</b></p>
语法	<p>APP_PASS(pass)</p> <p>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。</p>
适用控制器	通用
例子	APP_PASS(Zmotion)
相关指令	<a href="#">LOCK</a>

#### LOCK -- 锁定控制器

类型	系统指令
描述	<p>锁定控制器，不让对控制器操作。</p> <p>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。</p> <p>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。</p> <p><b>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。</b></p>
语法	<p>LOCK (pass)</p> <p>pass: 字母或数字，“_”等少数特殊符号，总共不要超过 16 个字符</p>
适用控制器	通用
例子	LOCK(passwd)     '锁定控制器，密码为 passwd
相关指令	<a href="#">UNLOCK</a>

## UNLOCK -- 解锁控制器

类型	系统指令
描述	解锁控制器。 <b>LOCK</b> 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	<a href="#">LOCK</a>

## 14.2. 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。 <b>仿真器无法修改日期。</b>
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	<a href="#">DATES</a> ， <a href="#">RTC DATE</a>

### DATES -- 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	DATES
适用控制器	通用
例子	DATE=27:2:13 在线命令输入 >>PRINT DATES 输出：27:02:2013
相关指令	<a href="#">DATE</a> ， <a href="#">RTC DATE</a>

## DAY -- 系统星期

类型	系统参数
描述	<p>设置系统时钟的星期，0-6，0 表示星期天，掉电保存。</p> <p>仿真器无法修改日期。</p> <p>DAY 不随 DATE 和 RTC_DATE 改变而改变，两者独立。</p>
语法	VAR=DAY， DAY=expression
适用控制器	通用
例子	<p>例一</p> <p><b>DAY = 3</b></p> <p>例二</p> <p>在线命令输入</p> <p>&gt;&gt;PRINT DAY</p> <p>输出：3</p>
相关指令	<a href="#">DAYS</a>

## DAYS -- 系统星期

类型	字符串函数
描述	<p>字符串函数，返回 DAY 设置星期。</p> <p>DAYS 不随 DATE 和 RTC_DATE 改变而改变，两者独立。</p>
语法	DAYS
适用控制器	通用
例子	<p>在线命令输入</p> <p>&gt;&gt;PRINT DAYS</p> <p>输出：Wednesday</p>
相关指令	<a href="#">DAY</a>

## RTC\_DATE --系统日期

类型	系统参数
描述	<p>设置或者获取系统日期，掉电保存，从 2000 年 1 月 1 日起。</p> <p>注意与 DATE 指令的表示格式是不一样的，返回值类型为整数。</p> <p>仿真器无法修改日期。</p>
语法	RTC_DATE = YYYYMMDD 或 YYMMDD
适用控制器	通用
例子	<p><b>RTC_DATE = 20130227</b></p> <p>在线命令输入</p>



	<pre>&gt;&gt;PRINT  RTC_DATE</pre> <p>输出: 20130227</p> <pre> DIM gRTC_Date,CurDate,curYear,curMonth,curDay ?RTC_DATE gRTC_Date = RTC_DATE CurDate=gRTC_Date mod 20000000 ?CurDate curYear=int(CurDate/10000) ?curYear curMonth=int((CurDate-curYear*10000)/100) ?curMonth curDay=CurDate-curYear*10000-curMonth*100 ?curDay </pre>
相关指令	<a href="#">DATE</a> , <a href="#">DATE\$</a>

## 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。 <b>仿真器无法修改日期。</b>
语法	TIME = hh:mm:ss
适用控制器	通用
例子	<p>例一</p> <pre>TIME= 11:14:40</pre> <p>例二</p> <p>在线命令输入</p> <pre>&gt;&gt;PRINT  TIME</pre> <p>输出: 40541</p>
相关指令	<a href="#">TIMES</a> , <a href="#">RTC TIME</a>

## TIMES -- 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	TIMES
适用控制器	通用
例子	<p>在线命令输入</p> <pre>&gt;&gt;PRINT  TIMES</pre>

	输出: 11:29:46
相关指令	<a href="#">TIME</a> , <a href="#">RTC_TIME</a>

## 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	<a href="#">TIME</a> , <a href="#">TIMES</a>

## 14.3. 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	<a href="#">AXIS_ENABLE</a>

### 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

	<b>DISABLE_GROUP(0,5,1)</b> '轴 0,5,1 设置一组 <b>DISABLE_GROUP(4,2)</b> '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	<a href="#">WDOG</a> ， <a href="#">AXIS_ENABLE</a>

## 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	<a href="#">MOTION_ERROR</a>

## 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	<a href="#">ERROR_AXIS</a>

## 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。  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: 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	<pre> <b>ERROR_SET</b>(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 </pre>

## RADIUS\_ERRSET -- 圆弧插补检查

类型	系统参数
描述	圆弧插补圆心方向半径检查设置。
语法	<p>RADIUS_ERRSET=mode</p> <p>mode : 0- 缺省值，不检查</p> <p>1- 起始点结束点半径不一样(相差超过 2 个脉冲)，自动纠正。</p> <p>2- 不一致，返回 1006 错误</p> <p>2022 年 1 月 11 日添加支持.</p>
适用控制器	通用
例子	<pre> RADIUS_ERRSET=2      '圆弧指令坐标不对报错 1006，圆弧指令不运行  MOVECIRC(200,0,98,0,1)      '画圆弧 RADIUS_ERRSET=1      '圆弧指令坐标不对自动纠正，提示 Center error, radius:98.000000 radiusend:102.000000 diff.  ?RADIUS_ERRSET      '设置查询 </pre>
相关指令	<a href="#">MOVECIRC</a>

## 14.4. IP 参数指令

### IP\_ADDRESS -- IP 地址

类型	系统参数，自动存储到 FLASH
描述	<p><b>控制器 IP 地址。</b></p> <p>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。</p> <p>修改立刻生效，使用网口连接时，修改后网口会断开，需重连。</p> <p>单网卡连接控制器，网卡与控制器处于同一网段即可。</p> <p>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。</p>
语法	IP_ADDRESS = dot.dot.dot.dot
适用控制器	通用
例子	<p>在线命令输入</p> <pre>&gt;&gt;IP_ADDRESS=192.168.0.26 &gt;&gt;PRINT IP_ADDRESS</pre> <p>输出：436250816</p> <p>具体转换过程如下：</p> <p>四段转换为二进制</p> <pre>192 -- 1100 0000 168 -- 1010 1000 0 -- 0000 0000 26 -- 0001 1010</pre> <p>二进制重新组合</p> <pre>26      0      168      192 0001 1010  0000 0000  1010 1000  1100 0000</pre> <p>转化为十进制</p> <pre>436250816</pre>
相关指令	<a href="#">IP_GATEWAY</a> ， <a href="#">IP_NETMASK</a>

### IP\_ADDRESS2 -- IP 地址 2

类型	系统参数，自动存储到 FLASH
描述	<p><b>5 系列控制器第二个网口的 IP 地址。</b></p> <p>5 系列控制器支持，出厂默认第二个网口 IP 为 192.168.1.11，读取时以 32 位整数返回。</p> <p>修改立刻生效，使用网口连接时，修改后网口会断开，需重连。</p> <p>单网卡连接控制器，网卡与控制器处于同一网段即可。</p>

	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_ADDRESS2 = dot.dot.dot.dot
适用控制器	5 系列控制器以上
例子	在线命令输入 >>IP_ADDRESS=192.168.1.26
相关指令	<a href="#">IP_ADDRESS</a>

## 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	<a href="#">IP_NETMASK</a> ， <a href="#">IP_ADDRESS</a>

## 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	<a href="#">IP_GATEWAY</a> ， <a href="#">IP_ADDRESS</a>

## IP\_IFDHCP -- IP 自动获取

类型	系统参数
----	------

描述	是否使用 <b>DHCP</b> 设置，缺省是不使用。 自动获取 IP，设置了这个后 ZDevelop 的固定 IP 没用，只能自动获取，此参数修改后要重启。
语法	IP_IFDHCP = 1 或 0      1-自动获取 IP; 0-使用固定 IP
适用控制器	通用
例子	在线命令输入 >>IP_IFDHCP =1
相关指令	<a href="#">IP_ADDRESS</a>

## IP\_IFDHCP2 -- IP 自动获取 2

类型	系统参数
描述	<b>5 系列控制器的第二个网口</b> 是否使用 <b>DHCP</b> 设置，缺省是不使用。 自动获取 IP，设置了这个后 ZDevelop 的固定 IP 没用，只能自动获取，此参数修改后要重启。
语法	IP_IFDHCP2 = 1 或 0      1-自动获取 IP; 0-使用固定 IP
适用控制器	5 系列控制器
例子	在线命令输入 >>IP_IFDHCP2 =1
相关指令	<a href="#">IP_ADDRESS</a>

## 14.5. 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	<p>?*VERSION_DATE '打印出固件版本 结果:20161122</p>  <p>在线命令: ?version_date “控制器状态” - “SoftVersion” 查看</p>

## 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	<p>?*VERSION '打印出软件版本 结果:4.3300</p>  <p>在线命令: ?version “控制器状态” - “SoftVersion” 查看</p>

## 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	<p>在线命令输入 &gt;&gt;PRINT ID_HARDWARE '打印控制器型号 输出: 464</p>  <p>在线命令: print id hardware “控制器状态” - “HardVersion” 查看</p>
相关指令	<a href="#">CONTROL</a>

## 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	<div>在线命令输入</div> <div>&gt;&gt;PRINT CONTROL '打印控制器型号</div> <div>输出：464</div> <div><div><div>&gt;&gt;print CONTROL</div><div>464</div></div><div>在线命令: print CONTROL</div></div> <div>“控制器状态” - “SoftType” 查看</div> <div><div>控制器状态</div><div><div>VirtualAxes: 64</div><div>RealAxes: 64</div><div>Taskes: 26</div><div>Files/3Files: 62/1</div><div>Modbus0x Bits: 8000</div><div>Modbus4x Regs: 8000</div><div>VR Regs: 8000</div><div>TABLE Regs: 320000</div><div>RomSize: 31250KB</div><div>FlashSize: 258200KB</div><div>SoftType: ZMC464</div><div>SoftVersion: 4.330-20181122</div><div>IpAddress: 192.168.0.49</div><div>HardVersion: 464-0</div><div>ControllerID: 161200004</div></div></div>
----	---

SYSTEM\_ZSET -- 控制器设置

类型	系统参数
描述	<div>控制器设置。</div> <div>设置参数：</div> <div>bit0: 1-VP_SPEED 缺省使用插补速度， 0-VP_SPEED 使用单轴的速度</div> <div>bit1: 1-使用 MOVE_OP 精确输出功能,， 0-MOVE_OP 普通方式</div> <div>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</div> <div>bit7: 1-总线时钟优化， 0-脉冲轴时钟优化</div> <div>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。</div> <div>20170505 以上固件版本支持这个位设置。</div> <div>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。</div> <div>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。</div>

	总线时钟优化可在总线启动后，通过?*ETHERCAT 命令来查看是否有使用 >>?*ethercat Slot:0 contain 1 nodes. <b>dc:ecat-sensitive</b> .Lostcount:0-0
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	<a href="#">MOVE_OP</a> ， <a href="#">AXIS_ZSET_MOVE_OP</a> -- <a href="#">缓冲输出</a>

## LEDOUT -- 控制器指示灯

类型	系统指令
描述	操作控制器指示灯。 <b>电源指示灯无法操作。</b>
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	<b>LEDOUT</b> (1,0)     '关闭运行灯 <b>LEDOUT</b> (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 <b>PRINT SERIAL_NUMBER</b> '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。

	<pre> ?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 </pre>
--	---

## SERVO\_PERIOD -- 总线通讯周期

类型	系统参数
描述	<p>总线伺服通讯周期。</p> <p>缺省 1000 微秒，修改功能预留，目前只读。</p> <p>4 系列控制器标准固件版本 20170713、4 系列控制器 fast 固件版本 20190106、5 系列控制固件版本 20180307 增加支持用户自行修改周期的功能，必须在控制器支持的周期范围内修改，修改后要重启才生效。</p> <p><b>Rtex 驱动使用时请按以下设置</b></p> <p>控制器周期 500us 时，将驱动器的 P7.20 设置为 3，P7.21 设置为 1。</p> <p>控制器周期 1000us 时，将驱动器的 P7.20 设置为 6，P7.21 设置为 1。</p> <p>查看 Rtex 驱动器设置请看例子。</p>
语法	value=SERVO_PERIOD
适用控制器	通用
例子	<p>在线命令打印控制器通讯周期和 <b>Rtex 驱动器设置</b></p> <pre> &gt;&gt;&gt;PRINT  SERVO_PERIOD      '打印伺服更新周期，结果 1000 &gt;&gt;&gt;DRIVE_READ(7*256+20)AXIS(0) '打印 Rtex 轴 0 驱动器周期设置 &gt;&gt;&gt;DRIVE_READ(7*256+21)AXIS(0) '打印 Rtex 轴 0 驱动器周期比值设置 </pre> <p>部分固件修改周期：</p> <pre> SERVO_PERIOD=500 SERVO_PERIOD=1000 </pre>
相关指令	<a href="#">SERVO</a>

## 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	<p>num = SYS_ZFEATURE (code)</p> <p>num: 返回值</p> <p>code: 获取数据类型</p> <ul style="list-style-type: none"> <li>0- 最大虚拟轴数</li> <li>1- 支持电机个数</li> <li>2- 本体输入个数</li> <li>3- 本体输出个数</li> <li>4- 本体模拟输入 AIN</li> <li>5- 本体模拟输出 AOUT</li> <li>6- PWM 个数</li> <li>7- BASIC 任务数，不包含中断任务</li> <li>8- 总线槽位数</li> <li>9- 3 次文件个数</li> <li>10- 串口链接数</li> <li>11- 网络链接数</li> <li>12- 网络自定义链接数 0-不支持</li> <li>13- 网络互联主端数，0-不支持，（从端总是支持）</li> <li>14- FLASH 块数</li> <li>15- FLASH 块大小</li> <li>16- VR 个数</li> <li>17- MODBUS_BIT 个数</li> <li>18- MODBUS_REG 个数</li> <li>19- 定时器个数</li> <li>20- 数组空间</li> <li>21- 虚拟最大输入个数，对应 PLC 的 X 寄存器个数</li> <li>22- 虚拟最大输出个数，对应 PLC 的 Y 寄存器个数</li> <li>23- 虚拟最大 AIN 个数</li> <li>24- 虚拟最大 AOUT 个数</li> <li>25- PLC 计数器</li> <li>26- PLC S 寄存器</li> <li>27- PLC V 寄存器</li> <li>28- PLC Z 寄存器</li> <li>29- PLC L 寄存器</li> <li>30- HMI 个数，（包括网络 HMI 与本体 HMI）</li> <li>31- 本体自带 HMI 个数</li> </ul>	
适用控制器	通用	
例子	<p>?SYS_ZFEATURE (0)            '打印最大轴数</p> <p>?SYS_ZFEATURE (17)        '打印 MODBUS_BIT 个数</p>	
相关指令	/	

## 14.6. 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 <b>table</b> 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	<a href="#">TSIZE</a>

### 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	<a href="#">TABLE</a>

### 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。



14.7. 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。 示波器开启连续采集之后，不要在 Basic 里调用 TRIGGER 指令，建议手动触发采集。
语法	TRIGGER
适用控制器	通用
例子	<b>TRIGGER</b> '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	<a href="#">SCOPE</a>

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 <b>TRIGGER</b> 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 <b>SCOPE</b> (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 <b>TRIGGER</b> '开始抓取 <b>MOVE</b> (10000)
相关指令	<a href="#">TRIGGER</a> , <a href="#">SCOPE_POS</a>

## SCOPE\_POS -- 采样点数

类型	系统指令
描述	只读，返回当前 <b>SCOPE</b> 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 <b>SCOPE</b> (ON,10,0,1000,DPOS(0),MSPEED(0)) '采样设置 <b>TRIGGER</b> '开始抓取 <b>MOVE</b> (10000) <b>WHILE</b> 1 <b>?SCOPE_POS</b> '返回当前已采样保存的点数 <b>WEND</b>
相关指令	<a href="#">SCOPE</a>

## 14.8. 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	<b>CLEAR()</b> 清除 VR 全部数据
相关指令	<a href="#">VR</a>

### 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。 <b>保存浮点数，与 VR_INT 和 VRSTRING 共用一个空间。</b>
语法	VR(index) = value , VAR1 = VR(index)
适用控制器	通用
例子	<b>VR(0) = 10.58</b> aaa = <b>VR(0)</b> ?aaa 打印结果 10.5800
相关指令	<a href="#">VR_INT</a> , <a href="#">VRSTRING</a>

### 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。 <b>只能保存整型，与 VR 和 VRSTRIG 共用一个空间。</b>
语法	VR_INT(index) = value , VAR1 = VR_INT(index)
适用控制器	通用
例子	<b>VR_INT(0) = 10.58</b> aaa = <b>VR_INT(0)</b> ?aaa 打印结果 10，仅保留整数部分
相关指令	<a href="#">VR</a> , <a href="#">VRSTRING</a>



## 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	<a href="#">VR</a> , <a href="#">VR_INT</a>

# 第十五章 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。

## 15.1. U 盘相关指令

### FILE -- U 盘文件操作


类型	文件指令																												
描述	加载、搜索控制器或 U 盘文件。 根据对应字符串选择功能。 U 盘读取文件系统支持 fat32 和 fat16，不支持 ntfs。 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。																												
语法	value = FILE "function" ,...																												
	"LOAD_ZAR"	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。																											
	"LOAD_TCF"	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></table>		TABLE0	标志位		TABLE1	总点数		TABLE2	胶头编号		TABLE100	点类型	第一个点	TABLE101	X 坐标		TABLE102	Y 坐标		TABLE103	Z 坐标		TABLE104	预留		TABLE105	点类型
TABLE0	标志位																												
TABLE1	总点数																												
TABLE2	胶头编号																												
TABLE100	点类型	第一个点																											
TABLE101	X 坐标																												
TABLE102	Y 坐标																												
TABLE103	Z 坐标																												
TABLE104	预留																												
TABLE105	点类型	第二个点																											

		TABLE106	X 坐标		
		TABLE107	Y 坐标		
		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","fileorddir" 删除指定文件。 file: 文件名全称，带扩展名。 支持文件夹的删除，可以带盘符 A、C				

		C 盘，表示 FLASH 目录，不带盘符缺省认为是 FLASH 目录 A 盘，表示 U 盘
	"DELETE"	FILE "DELETE","filename" 删除 U 盘指定文件。 filename: 文件名全称，带扩展名。
	"COPY_FROM"	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
	"COPY_TO"	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。
	"FLASH_COPY"	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
	"MAKE_DIR"	FILE "MAKE_DIR" "路径"
function: 功能选择		
适用控制器	通用，使用 U 盘功能时要求控制器带 U 盘接口	
例子	<p>例一 下载 zar 升级程序</p> <pre> DIM result                                '定义变量 IF U_STATE=TRUE THEN                      'U 盘插入判断     result = <b>FILE</b> "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         <b>FILE</b>"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 </pre> <p>例二 查找 zar 升级程序</p> <pre> <b>FILE</b> "find_next",10                      '查找下一个 zar 文件存储结果到 vrstring(10) <b>FILE</b> "find_prev",20                     '查找上一个 zar 文件存储结果 vrstring(20) </pre> <p>例三 FLASH 与 U 盘数据互相拷贝</p> <pre> DIM a,aa(8) a=10 FOR i=0 TO 7     aa(i)=i NEXT WHILE 1     IF SCAN_EVENT(IN(0))&gt; 0 THEN         FLASH_WRITE 1,a aa     </pre>	

	<pre>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))&gt; 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</pre> <p>例四 读取/删除 U 盘文件</p> <p><b>FILE</b> "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</p> <p><b>FILE</b> "DELETE" ,"sd0.bin"  '删除 U 盘上名称为 sd0.bin 的文件</p> <p>00.txt 文件内容：ZMOTION</p> <p>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。</p> <table><tr><th>寄存器名</th><th>值</th></tr><tr><td>DT(200)</td><td>7.000</td></tr><tr><td>DT(201)</td><td>90.000</td></tr><tr><td>DT(202)</td><td>77.000</td></tr><tr><td>DT(203)</td><td>79.000</td></tr><tr><td>DT(204)</td><td>84.000</td></tr><tr><td>DT(205)</td><td>73.000</td></tr><tr><td>DT(206)</td><td>79.000</td></tr><tr><td>DT(207)</td><td>78.000</td></tr><tr><td>DT(208)</td><td>0.000</td></tr><tr><td>DT(209)</td><td>0.000</td></tr></table>	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
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	<p>检查 U 盘是否插入。</p> <p>有插入返回 TRUE，否则返回 FALSE。</p> <p>不具备外部存储接口的控制器不支持此命令。</p> <p>U 盘里面的文件不要放太多。</p> <p>U 盘读取文件系统支持 fat32 和 fat16，不支持 ntfs。</p>
语法	Val=U_STATE
适用控制器	带 U 盘接口
例子	<pre>?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</pre>

相关指令	<a href="#">U_READ</a> , <a href="#">U_WRITE</a>
------	--

## U\_READ -- 从 U 盘读取

类型	存储指令
描述	<p>从外部存储器读取数据到变量或数组里面。</p> <p>不具备外部存储接口的控制器不支持此命令。</p> <p>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。</p> <p>U 盘读取文件系统支持 fat32 和 fat16，不支持 ntfs。</p>
语法	<p>U_READ sect_num, [varname] [,arrayname] [,arrayname(a)][, arrayname(a,length)]</p> <p>sect_num: 文件编号，对应到 SD【filenum】.BIN</p> <p>varname: 变量名</p> <p>arrayname: 数组名，可以为 TABLE, VR</p> <p>a: 操作的数组索引</p> <p>length: 操作的数组元素个数</p>
适用控制器	带 U 盘接口
例子	<pre>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</pre>
相关指令	<a href="#">U_WRITE</a> , <a href="#">U_READ2</a> , <a href="#">U_STATE</a>

## U\_READDBL -- 从 U 盘读取-double

类型	存储指令
描述	<p>从外部存储器读取数据到变量或数组里面。</p> <p>使用方法同 U_READ，区别是 U_READ 读取类型是 float，32 位，U_READDBL 读取的数据类型是 double，64 位。</p> <p>不具备外部存储接口的控制器不支持此命令。</p> <p>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。</p> <p>U 盘读取文件系统支持 fat32 和 fat16，不支持 ntfs。</p>
语法	<p>U_READDBL sect_num, [varname] [,arrayname] [,arrayname(a)][, arrayname(a,length)]</p> <p>sect_num: 文件编号，对应到 SD【filenum】.BIN</p> <p>varname: 变量名</p> <p>arrayname: 数组名，可以为 TABLE, VR</p> <p>a: 操作的数组索引</p> <p>length: 操作的数组元素个数</p>
适用控制器	带 U 盘接口的 4 系列及以上控制器，20190128 及以上固件支持

例子	IF U_STATE = TRUE THEN 'U 盘已插入 <b>U_READDBL</b> 1, VAR, TABLE(0), ARRAY2(1) '读取 U 盘文件 SD1 数据 ENDIF
相关指令	<a href="#">U_WRITE</a> , <a href="#">U_READ2</a> , <a href="#">U_STATE</a>

## U\_READ2 -- 从 U 盘读取 2

类型	存储指令
描述	<p>从外部存储器读取数据到变量或数组里面，支持设定文件读取的起始位置。</p> <p>不具备外部存储接口的控制器不支持此命令。</p> <p>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。</p> <p>U 盘读取文件系统支持 fat32 和 fat16，不支持 ntfs。</p>
语法	<p><b>U_READ2</b> sect_num, start_num[, varname][, arrayname][, arrayname(a)][, arrayname(a, length)]</p> <p>sect_num: 文件编号，对应到 SD【filenum】.BIN</p> <p>start_num: 文件内读取的起始位置</p> <p>varname: 变量名</p> <p>arrayname: 数组名，可以为 TABLE, VR</p> <p>a: 操作的数组索引</p> <p>length: 操作的数组元素个数</p>
适用控制器	带 U 盘接口
例子	IF U_STATE = TRUE THEN 'U 盘已插入 <b>U_READ2</b> 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	<a href="#">U_READ</a> , <a href="#">U_WRITE</a> , <a href="#">U_STATE</a>

## U\_READ2DBL -- 从 U 盘读取 2-double

类型	存储指令
描述	<p>从外部存储器读取数据到变量或数组里面，支持设定文件读取的起始位置。</p> <p>使用方法同 U_READ2，区别是 U_READ2 读取类型是 float，32 位，U_READ2DBL 读取的数据类型是 double，64 位。</p> <p>不具备外部存储接口的控制器不支持此命令。</p> <p>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。</p> <p>U 盘读取文件系统支持 fat32 和 fat16，不支持 ntfs。</p>
语法	<p><b>U_READ2DBL</b> sect_num, start_num [, varname] [, arrayname] [, arrayname(a)] [, arrayname(a, length)]</p>

	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 盘接口的 4 系列及以上控制器, 20190128 及以上固件支持
例子	<pre>IF U_STATE = TRUE THEN  'U 盘已插入     U_READ2DBL  1,10,VAR, TABLE(0), ARRAY2(1)  '读取 U 盘文件 SD1 数据从     SD1 位置 10 开始 ENDIF</pre>
相关指令	<a href="#">U_READ</a> , <a href="#">U_WRITE</a> , <a href="#">U_STATE</a>

## U\_READDSB -- DSB 文件读取

类型	存储指令																																																							
描述	<b>DSB 文件读取。</b> 不具备外部存储接口的控制器不支持此命令。 U 盘读取文件系统支持 fat32 和 fat16, 不支持 ntfs。																																																							
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 <table border="1"> <thead> <tr> <th>编号</th><th>内容</th><th></th></tr> </thead> <tbody> <tr> <td>0</td><td>标志位: 标明一行的变量数(暂无用)</td><td></td></tr> <tr> <td>1</td><td>针迹数:</td><td></td></tr> <tr> <td>2--29</td><td>文件名:28 个字符</td><td></td></tr> <tr> <td>30</td><td>换色次数:</td><td></td></tr> <tr> <td>31</td><td>+X:</td><td></td></tr> <tr> <td>32</td><td>-X:</td><td></td></tr> <tr> <td>33</td><td>+Y:</td><td></td></tr> <tr> <td>34</td><td>-Y:</td><td></td></tr> <tr> <td>35</td><td>AX: +</td><td></td></tr> <tr> <td>36</td><td>AY: -</td><td></td></tr> <tr> <td>37</td><td>MX: +</td><td></td></tr> <tr> <td>38</td><td>MY: +</td><td></td></tr> <tr> <td>39</td><td>PD:</td><td></td></tr> <tr> <td>40</td><td>L_limt</td><td>左边界坐标</td></tr> <tr> <td>41</td><td>R_limt</td><td>右边界坐标</td></tr> <tr> <td>42</td><td>U_limt</td><td>上边界坐标</td></tr> <tr> <td>43</td><td>D_limt</td><td>下边界坐标</td></tr> </tbody> </table>		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
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		
适用控制器	带 U 盘接口			
相关指令	<a href="#">U_READ</a> ， <a href="#">U_WRITE</a> ， <a href="#">U_STATE</a>			

## U\_WRITE -- 输出到 U 盘

类型	存储指令
描述	<p>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。</p> <p>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。</p>
语法	<p><code>U_WRITE sect_num, [varname] [,arrayname] [,arrayname(a)] [,arrayname(a,length)]</code></p> <p>sect_num: 文件编号,对应到 SD【filenum】.BIN</p> <p>varname: 变量名</p> <p>arrayname: 数组名,可以为 TABLE, VR</p> <p>a: 操作的数组索引</p> <p>length: 操作的数组元素个数</p>
适用控制器	带 U 盘接口
例子	<pre>IF U_STATE = TRUE THEN    'U 盘已插入     U_WRITE 0, TABLE(0, 10)    'TABLE0-10 的数据写入 U 盘文件 SD0 ENDIF</pre>
相关指令	<a href="#">U_READ</a> , <a href="#">U_STATE</a>

## U\_WRITEDBL -- 输出到 U 盘-double

类型	存储指令
描述	<p>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。</p> <p>使用方法同 U_WRITE,区别是 U_WRITE 输出类型是 float, 32 位, U_WRITEDBL 输出的数据类型是 double, 64 位。 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</p>

	以通过 PC 事先做好文件，然后用 U_READDBL 指令读取。
语法	<p>U_WRITEDBL sect_num, [,varname] [,arrayname] [,arrayname(a)] [,arrayname(a,length)]</p> <p>sect_num: 文件编号，对应到 SD【filenum】.BIN</p> <p>varname: 变量名</p> <p>arrayname: 数组名，可以为 TABLE，VR</p> <p>a: 操作的数组索引</p> <p>length: 操作的数组元素个数</p>
适用控制器	带 U 盘接口的 4 系列及以上控制器，20190128 及以上固件支持
例子	<pre>IF U_STATE = TRUE THEN    'U 盘已插入     U_WRITEDBL 0, TABLE(0, 10) 'TBAL0-10 的数据写入 U 盘文件 SD0 ENDIF</pre>
相关指令	<a href="#">U_READ</a> ， <a href="#">U_STATE</a>

## STICK\_READ -- U 盘读取到 table

类型	存储指令						
描述	<p>拷贝外部存储器的数据到 TABLE 里面。</p> <p>value=TRUE 表示操作成功，否则操作失败。</p> <p>建议使用 U_READ。</p> <p>不具备外部存储接口的控制器不支持此命令。</p> <p>U 盘读取文件系统支持 fat32 和 fat16，不支持 ntfs。</p>						
语法	<p>value = STICK_READ (filenum, table_start [,format])</p> <p>filenum: 文件号，对应到 SD【filenum】</p> <p>table_start: 开始操作的起始 TABLE 号</p> <p>format: 写入文件的格式</p> <table border="1"> <thead> <tr> <th>值</th><th>描述</th></tr> </thead> <tbody> <tr> <td>0 (缺省)</td><td>float 格式存储，.BIN</td></tr> <tr> <td>1</td><td>文本格式存储，.CSV</td></tr> </tbody> </table>	值	描述	0 (缺省)	float 格式存储，.BIN	1	文本格式存储，.CSV
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	<b>STICK_READ</b> (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	<a href="#">U_READ</a> ， <a href="#">STICK_READVR</a>						

## STICK\_WRITE -- table 输出到 U 盘

类型	存储指令
描述	<p>拷贝 TABLE 的数据到外部存储器里面。</p> <p>value=TRUE 表示操作成功，否则操作失败。</p> <p>建议使用 U_WRITE。</p>

	不具备外部存储接口的控制器不支持此命令。						
语法	<p>value = STICK_WRITE(filenum, table_start [,length [,format]])</p> <p>filenum: 文件号, 对应到 SD【filenum】</p> <p>table_start: 开始操作的起始 TABLE 号</p> <p>length: 要操作的 TABLE 元素个数, 缺省 128</p> <p>format: 写入文件的格式</p> <table border="1"> <thead> <tr> <th>值</th><th>描述</th></tr> </thead> <tbody> <tr> <td>0 (缺省)</td><td>float 格式存储, .BIN</td></tr> <tr> <td>1</td><td>文本格式存储, .CSV</td></tr> </tbody> </table>	值	描述	0 (缺省)	float 格式存储, .BIN	1	文本格式存储, .CSV
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	<b>STICK_WRITE(0,0,128,0)</b> '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	<a href="#">U_WRITE</a> , <a href="#">STICK_WRITEVR</a>						

## STICK\_READVR -- U 盘读取到 vr

类型	存储指令						
描述	<p>拷贝外部存储器的数据到 VR 里面。</p> <p>value=TRUE 表示操作成功, 否则操作失败。</p> <p>建议使用 U_READ。</p> <p>不具备外部存储接口的控制器不支持此命令。</p> <p>U 盘读取文件系统支持 fat32 和 fat16, 不支持 ntfs。</p>						
语法	<p>value = STICK_READVR (filenum, vr_start [,format])</p> <p>filenum: 文件号, 对应到 SD【filenum】</p> <p>vr_start: 开始操作的起始 VR 号</p> <p>format: 文件的格式</p> <table border="1"> <thead> <tr> <th>值</th><th>描述</th></tr> </thead> <tbody> <tr> <td>0 (缺省)</td><td>float 格式存储</td></tr> <tr> <td>1</td><td>文本格式存储</td></tr> </tbody> </table>	值	描述	0 (缺省)	float 格式存储	1	文本格式存储
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	<b>STICK_READVR (0,20,0)</b> '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	<a href="#">U_READ</a> , <a href="#">STICK_READ</a>						

## STICK\_WRITEVR -- vr 输出到 U 盘

类型	存储指令
描述	<p>拷贝 VR 的数据到外部存储器里面。</p> <p>value=TRUE 表示操作成功, 否则操作失败。</p>

	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 <table border="1"> <thead> <tr> <th>值</th><th>描述</th></tr> </thead> <tbody> <tr> <td>0 (缺省)</td><td>float 格式存储, .BIN</td></tr> <tr> <td>1</td><td>文本格式存储, .CSV</td></tr> </tbody> </table>	值	描述	0 (缺省)	float 格式存储, .BIN	1	文本格式存储, .CSV
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	<a href="#">U_WRITE</a> , <a href="#">STICK_WRITE</a>						

## 15.2. 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。 注意在运动中不要操作 FLASH, 对运动执行会有影响。
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

	<p>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</p> <p>MODBUS_LONG(1)=123456      '使用 MODBUS_REG(1)和 MODBUS_REG(2)存储</p> <p>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)</p> <p>MODBUS_LONG(1)=100</p> <p>FLASH_READ 1, MODBUS_REG(1,2)</p> <p>?MODBUS_REG(1)      '打印结果-7616</p> <p>?MODBUS_REG(2)      '打印结果 1</p> <p>?MODBUS_LONG(1)      '打印结果 123456</p>
相关指令	<a href="#">FLASHVR</a> , <a href="#">FLASH_READ</a>

## FLASH\_WRITEDBL -- flash 存储-double

类型	存储指令
描述	<p>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。</p> <p>使用方法同 FLASH_WRITE，区别是 FLASH_WRITE 存储类型是 float，32 位，FLASH_WRITEDBL 存储的数据类型是 double，64 位。</p> <p>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。</p> <p>内部 FLASH 有存储次数限制，不要随意循环操作。</p> <p>注意在运动中不要操作 FLASH，对运动执行会有影响。</p>
语法	<p>FLASH_WRITEDBL sect_num [, varname] [, arrayname] [, arrayname(a)] [, arrayname(a,length)]</p> <p>sect_num: flash 块编号，不同类型不一样</p> <p>varname: 变量名</p> <p>arrayname: 数组名，可以为 TABLE, VR, MODBUS</p> <p>a: 操作的数组索引</p> <p>length: 操作的数组元素个数</p>
适用控制器	4 系列及以上控制器，20190128 及以上固件支持
例子	FLASH_WRITEDBL 1,table(0,4)      '从 table(0)-table(4)，共 5 个元素 写入 FLASH 块 1
相关指令	<a href="#">FLASHVR</a> , <a href="#">FLASH_READ</a>

## FLASH\_READ -- flash 读取

类型	存储指令
描述	<p>从内部 FLASH 读取数据到变量，或数组里面。</p> <p>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。</p> <p>读取未被写入过的 Flash 块时，会提示 Warn file:"BASIC1.BAS" line:5 task:0,</p>

	File:C:\SD10.BIN open error, not load., 不影响使用。 注意在运动中不要操作 <b>FLASH</b> , 对运动执行会有影响。
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	<a href="#">FLASH_READDBL</a> , <a href="#">FLASHVR</a> , <a href="#">FLASH_WRITE</a>

## FLASH\_READDBL -- flash 读取-double

类型	存储指令
描述	从内部 <b>FLASH</b> 读取数据到变量, 或数组里面。  使用方法同 FLASH_READ, 区别是 FLASH_READ 存储类型是 float, 32 位, FLASH_READDBL 存储的数据类型是 double, 64 位。 内部 <b>FLASH</b> 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 Warn file:"BASIC1.BAS" line:5 task:0, File:C:\SD10.BIN open error, not load., 不影响使用。 注意在运动中不要操作 <b>FLASH</b> , 对运动执行会有影响。
语法	FLASH_READDBL 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	4 系列及以上控制器, 20190128 及以上固件支持
例子	FLASH_READDBL 1, table(6,5) '读 FLASH 块 1 数据, 位置 6 开始, 读取 5 个, 放入 table(6)-table(10), 共 5 个元素
相关指令	<a href="#">FLASH_READ</a> , <a href="#">FLASHVR</a> , <a href="#">FLASH_WRITE</a>

## FLASH\_READ2 -- flash 读取 2

类型	存储指令
描述	从内部 <b>FLASH</b> 指定位置读取数据到变量, 或数组里面。

	<p>内部 <b>FLASH</b> 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。</p> <p>读取未被写入过的 Flash 块时，会提示 Warn file:"BASIC1.BAS" line:5 task:0, File:C:\SD10.BIN open error, not load., 不影响使用。</p> <p>注意在运动中不要操作 <b>FLASH</b>，对运动执行会有影响。</p>
语法	<p>FLASH_READ2 sect_num start_num [, varname] [, arrayname] [, arrayname(a)] [, arrayname(a,length)]</p> <p>sect_num: flash 块编号，不同类型不一样</p> <p>start_num: 文件内读取的起始位置</p> <p>varname: 变量名</p> <p>arrayname: 数组名，可以为 TABLE, VR</p> <p>a: 操作的数组索引</p> <p>length: 操作的数组元素个数</p>
适用控制器	通用
例子	<pre>FOR i=0 TO 10     TABLE(i)=120+i NEXT  FLASH_WRITE 1, TABLE(0,10) '数据写入 flash  FOR i=0 TO 11     TABLE(i)=0 NEXT  FLASH_READ2 1,2, TABLE(4,5) '从 table(2)开始依次读出数据,读 5 个数据依次放到 table(4)-table(8)  FOR i=0 TO 11     ?"TABLE",i,TABLE(i) NEXT END</pre>
相关指令	<a href="#">FLASH_READ</a> , <a href="#">FLASH_WRITE</a>

## FLASH\_READ2DBL -- flash 读取 2-double

类型	存储指令
描述	<p>从内部 <b>FLASH</b> 指定位置读取数据到变量，或数组里面。</p> <p>使用方法同 FLASH_READ2DBL，区别是 FLASH_READ2DBL 存储类型是 float，32 位，FLASH_READ2DBL 存储的数据类型是 double，64 位。</p> <p>内部 <b>FLASH</b> 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。</p> <p>读取未被写入过的 Flash 块时，会提示 Warn file:"BASIC1.BAS" line:5 task:0, File:C:\SD10.BIN open error, not load., 不影响使用。</p>

	注意在运动中不要操作 <b>FLASH</b> ，对运动执行会有影响。
语法	<p>FLASH_READ2DBL sect_num start_num [, varname] [, arrayname] [, arrayname(a)] [, arrayname(a,length)]</p> <p>sect_num: flash 块编号，不同类型不一样</p> <p>start_num: 文件内读取的起始位置</p> <p>varname: 变量名</p> <p>arrayname: 数组名，可以为 TABLE, VR</p> <p>a: 操作的数组索引</p> <p>length: 操作的数组元素个数</p>
适用控制器	4 系列及以上控制器，20190128 及以上固件支持
例子	<pre> FOR i=0 TO 10     TABLE(i)=120+i NEXT  FLASH_WRITEDBL 1, TABLE(0,10) '数据写入 flash  FOR i=0 TO 11     TABLE(i)=0 NEXT  FLASH_READ2 DBL 1,2, TABLE(4,5) '从 table(2)开始依次读出数据,读 5 个数据依次 放到 table(4)-table(8)  FOR i=0 TO 11     ?"TABLE",i,TABLE(i) NEXT END </pre>
相关指令	<a href="#">FLASH_READDBL</a> ， <a href="#">FLASH_READ2</a> ， <a href="#">FLASH_WRITE</a>

## FLASHVR -- 拷贝 RAM 的数据

类型	存储指令
描述	<p>拷贝 <b>RAM</b> 的数据到 <b>FLASH</b> 里面。</p> <p><b>ZMC00x</b>（除了 005）、<b>ECI</b> 全系列控制器把 <b>TABLE</b> 数据存储到了 <b>FLASH</b> 的最后一个块，用 <b>FLASH_WRITE</b>，<b>FLASH_READ</b> 指令也可以操作到这个块，要避免冲突。</p> <p>其他系列控制器把 <b>TABLE</b> 数据存储到另一个独立的存储区域，无法操作。</p> <p>注意在运动中不要操作 <b>FLASH</b>，对运动执行会有影响。</p>



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
适用控制器	通用	
例子	<b>FLASHVR (-1)</b> 'table 存到 flash 块, 重新上电后再从 flash 读回 table	
相关指令	<a href="#">FLASH_WRITE</a>	

## FLASH\_SECTSIZE -- flash 变量数

类型	系统状态函数
描述	读取内部 <b>FLASH</b> 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	<a href="#">FLASH_SECTES</a>

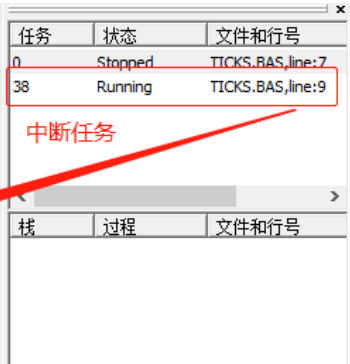
## FLASH\_SECTES -- flash 块数

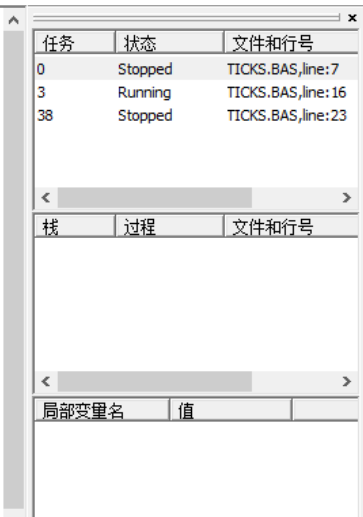
类型	系统状态函数
描述	读取控制器内部 <b>FLASH</b> 的总块数。 不同控制器块数不同。 对 ZMC00x 系列控制器, FLASHVR 会使用用最后一个块, 要避免冲突。
语法	value = FLASH_SECTES
适用控制器	通用
例子	?FLASH_SECTES '打印出 flash 块数 打印结果 128 'ZMC005
相关指令	<a href="#">FLASH_SECTSIZE</a>

# 第十六章 中断相关指令

## 16.1. 三种中断指令

### INT\_ENABLE -- 中断总开关

类型	系统参数						
描述	<p>中断总开关。</p> <p>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。</p> <p>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。</p> <p>解决办法：</p> <p>(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。</p> <p>(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。</p>						
语法	<div><div>INT_ENABLE = switch</div><table><tr><th>值</th><th>描述</th></tr><tr><td>0（缺省）</td><td>关闭</td></tr><tr><td>1</td><td>打开</td></tr></table></div>	值	描述	0（缺省）	关闭	1	打开
值	描述						
0（缺省）	关闭						
1	打开						
适用控制器	通用						
例子	<div><div><p><b>错误示范，中断堵塞</b></p><p>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。</p><pre>1  INT_ENABLE=1      ' 开启中断 2  TIMER_START(0,1000)    ' 定时器0开启，1000ms后执行一次 3  TIMER_START(1,1100)    ' 定时器1开启，1100ms后执行一次 4 5  END 6 7  GLOBAL SUB ONTIMER0() ' 中断处理函数 8      DELAY 10000 ' 假设大量的堵塞性代码 9      WAIT UNTIL IN(0) &lt;&gt; 0    ' 第一个中断 10 11  END SUB 12 13 GLOBAL SUB ONTIMER1() ' 中断处理函数 14     ' 第二个中断 15  END SUB</pre></div><div><p><b>正确示范</b></p><p>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。</p></div></div>						

<pre>1  INT_ENABLE=1          '开启中断 2  TIMER_START(0,1000)    '定时器0开启, 1000ms后执行一次 3  TIMER_START(1,1100)    '定时器1开启, 1100ms后执行一次 4 5  END 6 7  GLOBAL SUB ONTIMER0()  '中断处理函数 8  '创建一个新任务来处理自己的复杂任务 9  '就不会堵塞其他中断的响应速度 10 11  RUNTASK 3, MyIntHandler() 12  END SUB 13 14  GLOBAL SUB MyIntHandler() 15  DELAY 1000 '假设大量的堵塞性代码 16  WAIT UNTIL IN(0) &lt;&gt; 0 17  ?"第一个中断" 18  END SUB 19 20  GLOBAL SUB ONTIMER1()  '中断处理函数 21  ?"第二个中断" 22  END SUB 23</pre>	
<p>对应代码如下：</p> <pre>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) &lt;&gt; 0     ?"第一个中断" END SUB  GLOBAL SUB ONTIMER1()  '中断处理函数     ?"第二个中断" END SUB</pre>	
相关指令	<a href="#">ONPOWEROFF</a> , <a href="#">ONTIMERn</a>

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

	<pre> 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 </pre>
相关指令	<a href="#">INT_ENABLE</a>

## INT\_ONn -- 外部输入中断 SUB

类型	中断
描述	<p>外部输入中断程序入口，上升沿触发，必须是全局的 SUB 过程。</p> <p>必须支持 PLC 功能的固件才可使用。</p> <p>中断 IN 口 0~31。</p>
语法	<pre> GLOBAL SUB INT_ONn()    n 是 IN 编号 ... END SUB </pre>
适用控制器	支持 PLC 功能的固件
例子	<pre> INT_ENABLE=1    '开启中断 END  GLOBAL SUB INT_ON0 () '中断程序     PRINT "输入 IN0 上升沿触发" END SUB </pre>
相关指令	<a href="#">INT_OFFn</a> , <a href="#">INT_ENABLE</a>

## INT\_OFFn -- 外部输入中断 SUB

类型	中断
描述	<p>外部输入中断程序入口，下降沿触发，必须是全局的 SUB 过程。</p> <p>必须支持 PLC 功能的固件才可使用。</p> <p>中断 IN 口 0~31。</p>
语法	<pre> GLOBAL SUB INT_OFFn()    n 是 IN 编号 ... END SUB </pre>
适用控制器	支持 PLC 功能的固件
例子	<pre> INT_ENABLE=1    '开启中断 END  GLOBAL SUB INT_OFF0 () '中断程序 </pre>

	PRINT "输入 IN0 下降沿触发" END SUB
相关指令	<a href="#">INT_ONn</a> , <a href="#">INT_ENABLE</a>

## ONTIMERn -- 定时器中断 SUB

类型	中断
描述	<p>定时器中断程序入口，必须是全局的 SUB 过程。</p> <p>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 max_timer。</p> <pre>max_task:26 max_timer:128  max_loopnest:8 max_callstack:8</pre> <p>在线命令: ?*max</p>
语法	<pre>GLOBAL SUB ONTIMERn()    'n 是定时器编号 ... END SUB</pre>
适用控制器	通用
例子	<pre>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</pre>
相关指令	<a href="#">INT_ENABLE</a> , <a href="#">TIMER_START</a>

## INT\_CYCLE -- 中断周期执行

类型	系统参数
描述	<p>中断周期执行 BASIC 功能，每个 SERVO_PERIOD 执行一次。</p> <p>4 系列以上，20170630 以上版本开放。</p>
语法	<p>命令语法: INT_CYCLE(function, taskid [, subname])</p> <p>参数描述:</p> <ul style="list-style-type: none"> <li>function: 1-启动, 2-停止</li> <li>taskid: 使用的 BASIC 任务号, BASIC 本身不能使用</li> <li>subname: 周期执行的 SUB 名称, 程序必须足够精简</li> </ul> <p>函数语法: var = INT_CYCLE(function, taskid)</p> <p>参数描述:</p>

	<p>function:</p> <p>3 -返回状态: 1-使能, 0-停止</p> <p>4 -返回时间, 上次的执行时间 us</p> <p>5 -返回时间, 执行最长时间 us</p> <p>6 -返回时间最长限制 us</p> <p>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</p> <p>8 -返回错误行号</p> <p>taskid: 使用的 BASIC 任务号</p>
适用控制器	通用
例子	<pre> DIM times INT_CYCLE(1,1,intisr) END  GLOBAL SUB intisr()     times=times+1     MOVE_PT(1,1)  '每周运行 END SUB </pre>
相关指令	<a href="#">INT_ENABLE</a>

## 16.2. 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	<p>value = TIMER_IFEND (timernum)</p> <p>返回 0: 定时器正在定时, 即未执行对应中断程序。</p> <p>返回 1: 定时器定时完成, 开始执行对应中断程序。</p> <p><b>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。</b></p>
适用控制器	通用
例子	<pre> 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 </pre>
相关指令	<a href="#">ONTIMERn</a> , <a href="#">TIMER_START</a>

## 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	<p>TIMER_START(timernum, time_ms)</p> <p>timernum: 编号, 0-定时器个数减 1</p> <p>time_ms: 定时器长度, 单位毫秒</p> <p>time 100 及以上为累积性计时器。</p>
适用控制器	通用
例子	参考 TIMER_IFEND 例程
相关指令	<a href="#">ONTIMERn</a> , <a href="#">TIMER_IFEND</a>

## 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	<p>TIMER_STOP (timernum)</p> <p>timernum: 编号, 0-定时器个数减 1</p>
适用控制器	通用
例子	<pre> 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 <b>TIMER_STOP(0)</b>        '停止计时器 0 ?TIMER_IFEND(0)        '打印结果, 0, 未运行 </pre>
相关指令	<a href="#">ONTIMERn</a> , <a href="#">TIMER_IFEND</a>

# 第十七章 总线相关指令

## 17.1. 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。

## 17.2. 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。



	<p>Rtex 扫描失败会直接报错 6209。</p> <p>遇到不支持的设备类型，<b>RETURN</b> 会返回 0。</p> <p><b>Rtex</b> 控制器未连接设备时会报错，<b>EtherCAT</b> 控制器不会。</p> <p>扫描后可以使用 <b>NODE</b> 指令读取相关设备信息，并使用 <b>DRIVE</b> 相关指令配置。</p>
语法	<p><b>SLOT_SCAN</b> (slot)</p> <p>slot: 控制器 EtherCAT 槽位号或 RTEX 槽位号, 0-缺省</p>
适用控制器	带 EtherCAT 接口或 RTEX 接口
例子	<pre> aa:                                '标记 aa <b>SLOT_SCAN</b>(0)                      '总线扫描 ? <b>RETURN</b>                          '打印返回值, -1 成功, 0 失败 <b>IF RETURN THEN</b>     ?<b>NODE_COUNT</b>(0)                '扫描成功, 返回当前连接的设备个数 <b>ELSE</b>     ?"扫描失败"     <b>DELAY</b> (1000)                  '等待 1s     <b>GOTO</b> aa                       '扫描失败, 跳转到 aa, 重新扫描 <b>ENDIF</b> </pre>
相关指令	<a href="#">SLOT_START</a> , <a href="#">SLOT_STOP</a>

## SLOT\_START -- 总线开启

类型	总线指令
描述	<p><b>总线启动。</b></p> <p>通过 <b>RETURN</b> 返回成功与否。返回-1 启动成功, 0 启动失败。</p> <p>需在总线扫描 <b>SLOT_SCAN</b> 成功后再执行。</p> <p>执行前, 先设置好 <b>AXIS_ADDRESS</b>, <b>ATYPE</b>, <b>DRIVE_PROFILE</b></p>
语法	<p><b>SLOT_START</b> (slot [,opstate])</p> <p>slot: 控制器 EtherCAT 槽位号或 RTEX 槽位号, 0-缺省</p> <p>opstate: 启动到的 EtherCAT 状态, 4-SAFEOP, 8- OP(缺省)</p> <p><b>NODE_PDOBUFF</b> 指令才使用</p>
适用控制器	带 EtherCAT 接口或 RTEX 接口
例子	<pre> aa:                                '标记 aa <b>SLOT_SCAN</b>(0)                      '总线扫描 <b>IF RETURN THEN</b>                    '扫描成功, 进入轴设置     <b>AXIS_ADDRESS</b>(0)=1              '第一个驱动器映射到轴 0     <b>ATYPE</b>(0)=65                    '轴类型 65, 位置控制     <b>DRIVE_PROFILE</b>(0)=0            'PDO 周期扫描设置     <b>SLOT_START</b>(0)                  '开启总线 <b>ELSE</b>     ?"扫描失败"     <b>DELAY</b> (1000)                  '等待 1s     <b>GOTO</b> aa                       '扫描失败, 跳转到 aa, 重新扫描 <b>ENDIF</b> </pre>
相关指令	<a href="#">SLOT_STOP</a> , <a href="#">SLOT_SCAN</a> , <a href="#">NODE_PDOBUFF</a>

## SLOT\_STOP -- 总线停止

类型	总线指令
描述	<b>总线停止。</b> 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	<pre> 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))&gt;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         </pre>
相关指令	<a href="#">SLOT_START</a> ， <a href="#">SLOT_SCAN</a>

## ?\*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	<b>?*SLOT</b> 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	<p><b>?*ETHERCAT</b> 打印结果：</p> <pre>Slot:0 contain 1 nodes. Lostcount:0-0. Node:0 status:1 manid:7595h productid:0h axes:1 Alstate:8 Node_profile:0. BindAxis:0 Drive_profile:0 Controlword:fh drive_status:1237h Drive_mode:8h target:ffffe067h encode:ffffe068h.</pre> <p>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  Axes: 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  Encoder: 编码器位置</p>
相关指令	<a href="#">PRINT</a>

## ?\*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	<p><b>?*RTEX</b> 打印结果：</p> <pre>Slot:0 contain 1 nodes. Lostcount:0-0. Node:0 status:1 manid:616e6150h devicetype:31h axes:1 Alstate:1. BindAxis:0 Drive_profile:0 Controlword:80h drive_status:3c1h target:ffffe5ch encode:ffffe5ch.</pre> <p>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</p>

	Status: 设备连接状态, 参考 NODE_STATUS Mainid: 厂商 ID Productid: 设备 ID Axes: 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 Encoder: 编码器位置
相关指令	<a href="#">PRINT</a>

## ZTEST -- EtherCAT 总线信息查询

类型	EtherCAT 辅助指令
描述	调试时使用, 可以查看多种信息。
语法	?*ETHERCAT
适用控制器	带 EtherCAT 接口
例子	<p>例一 查询当前 PDO 与关键数据字典</p> <pre> ztest(30,10,nodeid)     nodeid = 设备编号, 0--(n-1)  &gt;&gt;ztest(30,10,0) TestDriver_ecat para1:10,para2:0! reg:1c12:0 value:0x1 reg:1c12:1 value:0x1600 reg:1600:0 value:0x3 reg:1600:1 value:0x60400010 reg:1600:2 value:0x607a0020 reg:1600:3 value:0x60600008 reg:1c13:0 value:0x1 reg:1c13:1 value:0x1a00 reg:1a00:0 value:0x2 reg:1a00:1 value:0x60410010 reg:1a00:2 value:0x60640020 reg:6040:0 value:0xf reg:6041:0 value:0x1237 reg:6060:0 value:0x8 reg:6061:0 value:0x8 reg:6064:0 value:0x10ac4 reg:607a:0 value:0x10ac4 reg:603f:0 value:0x0 </pre> <p>例二 设备 AL 状态查询, 1-init, 2-preop, 4-safeop, 8-op          查询所有 ECAT 合并的 AL 状态。</p> <pre> &gt;&gt;ztest(30,1) TestDriver_ecat para1:1,para2:0! </pre>

	<p>al:0x8 code:0x0. alctrl:0x8</p> <p>ztest(30,2,nodeid) nodeid = 设备编号, 0---(n-1) 单个的 AL 状态查询。</p> <p>&gt;&gt;ztest(30,2,0) TestDriver_ecat para1:2,para2:0! al:0x8 code:0x0. alctrl:0x8</p> <p>例三 丢包查询 &gt;&gt;ztest(30,12) TestDriver_ecat para1:12,para2:0! Slot:0 contain 1 nodes. Lostcount:0-0. 第一个数据: 没有应答次数 第二个数据: 时钟冲突次数</p> <p>例四 查询是否支持指定设备: ztest (30,20,厂商 ID, 产品 ID, 版本号)</p> <p>&gt;&gt;ztest(30,20,\$41b,0,11) Id:0x41b ProductCode:0x0 version:0xb support. &gt;&gt;ztest(30,20,\$41b,145,11) Id:0x41b ProductCode:0x91 version:0xb not support.</p>
相关指令	<a href="#">PRINT</a>

17.3. 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	<p>通过设备号和槽位号进行 SDO 写入。</p> <p>通过 RETURN 返回成功与否，-1 写入成功，0 写入失败。</p> <p>需连接好设备，扫描总线后才能执行。</p> <p>只有可写的数据字典才能写入。</p>								
语法	<p>SDO_WRITE (slot, node, index, subindex ,type ,value)</p> <p>slot: 槽位号 0-缺省</p> <p>node: 设备编号 0-</p> <p>index: 数据字典编号，前面可加"\$"表示 16 进制，如\$6060</p> <p>subindex: 子编号</p> <p>type: 数据类型</p> <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></table>	1	boolean	2	integer 8	3	integer 16	4	integer 32
1	boolean								
2	integer 8								
3	integer 16								
4	integer 32								

	<table border="1"> <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> <p>value: 数据值</p>	5	unsigned 8	6	unsigned 16	7	unsigned 32
5	unsigned 8						
6	unsigned 16						
7	unsigned 32						
适用控制器	带 EtherCAT 接口						
例子	<pre>SLOT_SCAN(0) IF NODE_COUNT(0)&gt;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</pre>						
相关指令	<a href="#">SDO_WRITE_AXIS</a> , <a href="#">SDO_READ</a>						

## SDO\_WRITE\_AXIS -- 数据字典写入

类型	总线指令，仅 EtherCAT 可用														
描述	<p>通过轴号 SDO 写入。</p> <p>通过 RETURN 返回成功与否，-1 写入成功，0 写入失败。</p> <p>需连接好设备，扫描总线后才能执行。</p> <p>只有可写的数据字典才能写入。</p>														
语法	<p>SDO_WRITE_AXIS(axis, index, subindex ,type ,value)</p> <p>axis: 轴号</p> <p>index: 数据字典编号，前面可加"\$"表示 16 进制，如\$6060</p> <p>subindex: 子编号</p> <p>type: 数据类型</p> <table border="1"> <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> <p>value: 数据值</p>	1	boolean	2	integer 8	3	integer 16	4	integer 32	5	unsigned 8	6	unsigned 16	7	unsigned 32
1	boolean														
2	integer 8														
3	integer 16														
4	integer 32														
5	unsigned 8														
6	unsigned 16														
7	unsigned 32														
适用控制器	带 EtherCAT 接口														
例子	<pre>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)&gt;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</pre>														
相关指令	<a href="#">SDO_WRITE</a> , <a href="#">SDO_READ_AXIS</a>														

## SDO\_READ -- 数据字典读取

类型	总线指令，仅 EtherCAT 可用														
描述	<p>通过设备号和槽位号进行 SDO 读取。</p> <p>通过 RETURN 返回成功与否，-1 写入成功，0 写入失败</p> <p>需连接好设备，扫描总线后才能执行。</p> <p>只有可读的数据字典才能读取。</p>														
语法	<p>SDO_READ (slot, node, index, subindex ,type, tablenum)</p> <p>slot: 槽位号 0-缺省</p> <p>node: 设备编号 0-</p> <p>index: 数据字典编号，前面可加"\$"表示 16 进制，如\$6060</p> <p>subindex: 子编号</p> <p>type: 数据类型</p> <table border="1"> <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> <p>tablenum: 读取的数据存储的 TABLE 位置</p>	1	boolean	2	integer 8	3	integer 16	4	integer 32	5	unsigned 8	6	unsigned 16	7	unsigned 32
1	boolean														
2	integer 8														
3	integer 16														
4	integer 32														
5	unsigned 8														
6	unsigned 16														
7	unsigned 32														
适用控制器	带 EtherCAT 接口														
例子	<pre> SLOT_SCAN(0) IF NODE_COUNT(0)&gt;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 </pre>														
相关指令	<a href="#">SDO_READ_AXIS</a> ， <a href="#">SDO_WRITE</a>														

## SDO\_READ\_AXIS -- 数据字典读取

类型	总线指令，仅 EtherCAT 可用						
描述	<p>通过轴号进行 SDO 读取。</p> <p>通过 RETURN 返回成功与否，-1 写入成功，0 写入失败</p> <p>需连接好设备，扫描总线后才能执行。</p> <p>只有可读的数据字典才能读取。</p>						
语法	<p>SDO_READ_AXIS(axis, index, subindex ,type, tablenum)</p> <p>axis: 轴号</p> <p>index: 数据字典编号，前面可加"\$"表示 16 进制，如\$6060</p> <p>subindex: 子编号</p> <p>type: 数据类型</p> <table border="1"> <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> </table>	1	boolean	2	integer 8	3	integer 16
1	boolean						
2	integer 8						
3	integer 16						

	<table border="1"> <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> <p>tablename: 读取的数据存储的 TABLE 位置</p>	4	integer 32	5	unsigned 8	6	unsigned 16	7	unsigned 32
4	integer 32								
5	unsigned 8								
6	unsigned 16								
7	unsigned 32								
适用控制器	带 EtherCAT 接口								
例子	<p>正确的连接了一个 EtherCAT 轴设备再使用例程</p> <pre> SLOT_SCAN(0)           '总线扫描 IF NODE_COUNT(0)&gt;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   <b>SDO_READ_AXIS</b> (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 </pre>								
相关指令	<a href="#">SDO_READ</a> ， <a href="#">SDO_WRITE_AXIS</a>								

## 17.4. 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	<p>只读: var = NODE_COUNT (slot)</p> <p>slot: 槽位号, 0-缺省</p> <p>可以直接打印出来, 见例一 可以直接作为数据, 见例二</p>
适用控制器	带 EtherCAT 接口或 RTEX 接口
例子	<p>例一</p> <pre> SLOT_SCAN(0) ? <b>NODE_COUNT</b>(0)      '打印出 0 槽口连接的设备数 </pre> <p>例二</p> <pre> SLOT_SCAN(0) IF <b>NODE_COUNT</b>(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 </pre>
相关指令	<a href="#">NODE_INFO</a>



## 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	<a href="#">NODE_INFO</a>	

## 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	<pre>SLOT_SCAN(0) IF NODE_COUNT(0)&gt;0 THEN     ? NODE_AXIS_COUNT (0,0)      '打印出设备 0 带电机个数 ENDIF</pre>	
相关指令	<a href="#">NODE_INFO</a>	

## 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 RTEEX 接口
例子	SLOT_SCAN(0) IF NODE_COUNT(0)>0 THEN <b>NODE_IO</b> (0,0)=32           '设置设备 0 的 IO 起始编号为 32 ? <b>NODE_IO</b> (0,0)           '打印出设备 0 的 IO 起始编号 ENDIF
相关指令	<a href="#">NODE_AIO</a>

## 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]) 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 border="1" data-bbox="427 1205 1209 1312"> <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>	0	缺省, 同时设置 AIN、AOUT; 读取时只读 AIN	3	AIN	4	AOUT
0	缺省, 同时设置 AIN、AOUT; 读取时只读 AIN						
3	AIN						
4	AOUT						
适用控制器	带 EtherCAT 接口或 RTEEX 接口						
例子	SLOT_SCAN(0) IF NODE_COUNT(0)>0 THEN <b>NODE_AIO</b> (0,0,3)=3           '设置设备 0 的 AIN 起始编号为 3 ? <b>NODE_AIO</b> (0,0,3)           '打印出设备 0 的 AIN 起始编号 ENDIF						
相关指令	<a href="#">NODE_IO</a>						

## NODE\_INFO -- 设备信息

类型	EtherCAT 总线指令
描述	总线设备的信息读取。 必须总线扫描后才能读取。
语法	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	
	值	描述
	0	VENDER, 厂商编号
	1	DEVICE, 设备编号
	2	VERSION, 版本
	3	ALIAS, 别名, 一般用来识别驱动器
	4	预留
	IO 的个数	描述
	10	IN 个数
	11	OP 个数
	12	AIN 个数
	13	AOUT 个数
	14	预留
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_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	
相关指令	<a href="#">SLOT_SCAN</a>	

## NODE\_PROFILE -- PDO 预设置

类型	总线指令
描述	总线设备的 <b>profile</b> 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	<p>特殊 EtherCAT 设备的 PDO 支持。</p> <p>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。</p> <p>需要启动以后才能修改，可以先用 SDO_START 启动到 SAFEOP，然后设置初始化 PDO</p>

	状态., 再启动到 OP。														
语法	<p>命令语法: NODE_PDOBUFF (slot, node, index, subindex ,type)</p> <p>函数语法: Buff = NODE_PDOBUFF (slot, node, index, subindex ,type)</p> <p>slot: 槽位号 0-缺省</p> <p>node: 设备编号 0-</p> <p>index: 数据字典编号, 前面可加"\$"表示 16 进制, 如\$6060</p> <p>subindex: 子编号</p> <p>type: 数据类型</p> <table border="1"> <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>	1	boolean	2	integer 8	3	integer 16	4	integer 32	5	unsigned 8	6	unsigned 16	7	unsigned 32
1	boolean														
2	integer 8														
3	integer 16														
4	integer 32														
5	unsigned 8														
6	unsigned 16														
7	unsigned 32														
适用控制器	带 EtherCAT 接口, 4 系列产品, 20170508 以上版本支持														
例子	<p>&gt;&gt;NODE_PDOBUFF(0,0, \$6040, 0, 3) = 15</p> <p>&gt;&gt;?NODE_PDOBUFF(0,0, \$6041, 0, 3)</p>														
相关指令	<a href="#">SDO_WRITE</a> , <a href="#">SDO_READ</a>														

## NODE\_PRESET -- 设备预配置

类型	EtherCAT 总线指令
描述	<p>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。</p> <p>总线启动后不能再修改。</p> <p>预设置后, 可以通过 NODE_STATUS 判断外设是否挂上。</p> <p>预设值的类型与实际不一样, 将不能启动总线。</p> <p>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。</p> <p>固件 20160601 以上版本支持。</p>
语法	<p>命令语法 1: NODE_PRESET (slot, node, manuid, productid)</p> <p>命令语法 2: NODE_PRESET (slot, -1) 清除所有预设置</p> <p>函数语法 1: VALUE = NODE_PRESET (slot, node) 返回是否有预设置</p> <p>函数语法 1: VALUE = NODE_PRESET (slot) 返回预设值的最大个数</p> <p>slot: 槽位号, 0-缺省</p> <p>node: 设备编号, 0-</p> <p>manuid: 厂商 ID 编号, 参考 NODE_INFO</p> <p>productid: 设备 ID 编号, 参考 NODE_INFO</p>
适用控制器	带 EtherCAT 接口或 RTEX 接口
例子	<p><b>NODE_PRESET(0,-1)</b>清除原来的预设置</p> <p><b>NODE_PRESET(0,0, \$83, 5)</b> '设置第一个 NODE 为 OMRON 驱动器</p> <p>SLOT_SCAN(0)</p> <p>? "SCAN RESULT:", RETURN, "MAX", NODE_COUNT(0) '此时会自动显示总数为 1</p> <p>FOR i= 0 TO NODE_COUNT(0) -1</p> <p>? "node", i</p>

	? "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	<a href="#">NODE_STATUS</a>

## 17.5. 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 <b>ATYPE</b> （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	<pre> SLOT_SCAN(0) ... IF NODE_COUNT(0)&gt;0 THEN     <b>DRIVE_MODE</b>(0)=8      '轴 0 设备设置为位置控制模式     ? <b>DRIVE_MODE</b>(0)      '打印轴 0 的控制模式 ENDIF         </pre> '轴使能过程，参考第十七章 <a href="#">简易例程</a>

### 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 <b>ATYPE</b> （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, 0x60b90010, 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



	<pre> {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 </pre>
语法	<pre> 可读: var= DRIVE_PROFILE(axis) 可写: DRIVE_PROFILE(axis)= value axis: 轴号 </pre>



适用控制器	带 EtherCAT 接口或 RTEX 接口
例子	<pre> SLOT_SCAN(0) IF NODE_COUNT(0)&gt;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 </pre>
相关指令	<a href="#">ATYPE</a> , <a href="#">NODE_PROFILE</a>

## DRIVE\_CW\_MODE -- 驱动器设置

类型	轴参数
描述	<p><b>驱动器设置参数</b></p> <p>必须设置正确的 <b>ATYPE</b>（设置为 65/66/67）以后才能操作这个参数。</p> <p>部分版本为使用方便，可以直接操作 <b>MODE</b>。</p> <p>RTEX 的控制字不要随便修改。</p> <p>0-控制器自动调整 <b>DRIVE_CONTROLWORD</b> 参数,此时 <b>DRIVE_CONTROLWORD</b> 指令无效。</p> <p>1- 可以手动调整，此时可以使用 <b>DRIVE_CONTROLWORD</b> 指令。</p>
语法	<p>可读: var= DRIVE_CW_MODE(axis)</p> <p>可写: DRIVE_CW_MODE(axis)=value</p> <p>axis: 轴号</p>
适用控制器	带 EtherCAT 接口或 RTEX 接口
相关指令	<a href="#">ATYPE</a> , <a href="#">DRIVE_CONTROLWORD</a>

## DRIVE\_CONTROLWORD -- 驱动器控制字

类型	轴参数
描述	<p><b>驱动控制字，按位操作</b></p> <p>必须设置正确的 <b>ATYPE</b>（设置为 65/66/67）以后才能操作这个参数。</p> <p>对于 EtherCAT 驱动器， 对应数据字典 0x6040</p> <p>EtherCAT 控制器 <b>ATYPE=65</b> 时控制字会根据 <b>WDOG/AXIS_ENABLE</b> 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。</p>

	<table><tr><th rowspan="3">command</th><th colspan="5">bits of the controlword</th><th rowspan="3">PDS transitions</th></tr><tr><th>bit 7</th><th>bit 3</th><th>bit 2</th><th>bit 1</th><th>bit 0</th></tr><tr><th>fault reset</th><th>enable operation</th><th>quick stop</th><th>enable voltage</th><th>switch on</th></tr><tr><td>shut down</td><td>0</td><td>-</td><td>1</td><td>1</td><td>1</td><td>2,6,8</td></tr><tr><td>switch on</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>3</td></tr><tr><td>switch on+enable opetation</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>3+4 (*1)</td></tr><tr><td>enable opetation</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>4,16</td></tr><tr><td>disable voltage</td><td>0</td><td>-</td><td>-</td><td>0</td><td>-</td><td>7,9,10,12</td></tr><tr><td>quick stop</td><td>0</td><td>-</td><td>0(*2)</td><td>1</td><td>-</td><td>7,10,11</td></tr><tr><td>disable operation</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>5</td></tr><tr><td>fault reset</td><td></td><td>-</td><td>-</td><td>-</td><td>-</td><td>15</td></tr></table>	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
	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																																																																				
<p>对于 Rtex 驱动器</p> <p>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。</p> <table><tr><td>bit7</td><td>bit6</td><td>bit5</td><td>bit4</td><td>bit3</td><td>bit2</td><td>bit1</td><td>bit0</td></tr><tr><td>Servo_On</td><td>0</td><td>0</td><td>Gain_SW</td><td>TL_SW</td><td>HM_Ctrl</td><td>0</td><td>0</td></tr><tr><td>bit15</td><td>bit14</td><td>bit13</td><td>bit12</td><td>bit11</td><td>bit10</td><td>bit9</td><td>bit8</td></tr><tr><td>Hard_Stop</td><td>Smooth_Stop</td><td>Pause</td><td>0</td><td>SL_SW</td><td>0</td><td>EX-OUT2</td><td>EX-OUT1</td></tr></table>							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																																				
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	<p>可读：var=DRIVE_CONTROLWORD(axis)</p> <p>可写：DRIVE_CONTROLWORD(axis)=value</p> <p>axis: 轴号</p>																																																																									
适用控制器	带 EtherCAT 接口或 RTEX 接口																																																																									
例子	<p>例一 EtherCAT</p> <p>SLOT_SCAN(0)</p> <p>IF NODE_COUNT(0)&gt;0 THEN</p> <p>    AXIS_ADRESS(0)=1</p> <p>    ATYPE(0)=65    </p>																																																																									

	<b>DRIVE_CONTROLWORD(0)=128</b> '伺服使能 ENDIF
相关指令	<a href="#">ATYPE</a> , <a href="#">DRIVE CW MODE</a>

## DRIVE\_STATUS -- 驱动器状态

类型	轴状态																																																											
描述	<p>驱动器的当前状态，按位判断状态。</p> <p>必须设置正确的 <b>ATYPE</b>（EtherCAT 总线设置为 65/66/67，RTEX 总线设置为 50/51/52）以后才能读取。</p> <p>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</p> <table><tr><th>状态字</th><th colspan="2">PDS state</th></tr><tr><td>xxxx xxxx x0xx 0000 b</td><td>not ready to switch on</td><td>初始化 未完成状态</td></tr><tr><td>xxxx xxxx x1xx 0000 b</td><td>switch on disabled</td><td>初始化 完成状态</td></tr><tr><td>xxxx xxxx x01x 0001 b</td><td>ready to switch on</td><td>主电路电源 off 状态</td></tr><tr><td>xxxx xxxx x01x 0011 b</td><td>switch on</td><td>伺服使能 off/伺服准备</td></tr><tr><td>xxxx xxxx x01x 0111 b</td><td>operation enabled</td><td>伺服使能 on</td></tr><tr><td>xxxx xxxx x00x 0111 b</td><td>quick stop active</td><td>快速停止</td></tr><tr><td>xxxx xxxx x0xx 1111 b</td><td>fault reaction active</td><td>异常（报警）判断</td></tr><tr><td>xxxx xxxx x0xx 1000 b</td><td>fault</td><td>异常（报警）状态</td></tr></table> <p>其他位意义请查看对应驱动器手册说明。</p> <p>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。</p> <table><tr><td>bit7</td><td>bit6</td><td>bit5</td><td>bit4</td><td>bit3</td><td>bit2</td><td>bit1</td><td>bit0</td></tr><tr><td>Servo_Active</td><td>Servo_Ready</td><td>Alarm</td><td>Warning</td><td>Torque_Limited</td><td>Homing_Complete</td><td>In_Progress</td><td>In_Position</td></tr><tr><td>bit15</td><td>bit14</td><td>bit13</td><td>bit12</td><td>bit11</td><td>bit10</td><td>bit9</td><td>bit8</td></tr><tr><td>SI-MON5 /E-STOP</td><td>SI-MON4/ EX-SON</td><td>SI-MON3/E XT3/STOP</td><td>SI-MON2/E XT2/RET</td><td>SI-MON1/ EXT1</td><td>HOME</td><td>POT/NOT</td><td>NOT/POT</td></tr></table>	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	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
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																																																										
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 ? <b>DRIVE_STATUS</b> (0)                    '打印轴 0 设备的状态 ENDIF
相关指令	<a href="#">ATYPE</a> , <a href="#">DRIVE_PROFILE</a>

## DRIVE\_IO -- 驱动器 IO

类型	轴参数																										
描述	直接配置 <b>DRIVE</b> 的 <b>IO</b> 的起始编号，输入输出都是同样的编号。 <b>DRIVE_PROFILE</b> 支持读取驱动器 IO 时起作用。																										
语法	可读: var= <b>DRIVE_IO</b> (axis) 可写: <b>DRIVE_IO</b> (axis)=value 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 border="1" data-bbox="379 969 1093 1435"> <tr> <th>DRIVE_IO+以下编号</th><th>伺服功能</th></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>	DRIVE_IO+以下编号	伺服功能	输入点		0	NOT/POT	1	POT/NOT	2	HOME	3	SI-MON1/EXT1	4	SI-MON2/EXT2	5	SI-MON3/EXT3	6	SI-MON4/EX-SON	7	SI-MON5/E-STOP	输出点		0	EX-OUT1	1	EX-OUT2
DRIVE_IO+以下编号	伺服功能																										
输入点																											
0	NOT/POT																										
1	POT/NOT																										
2	HOME																										
3	SI-MON1/EXT1																										
4	SI-MON2/EXT2																										
5	SI-MON3/EXT3																										
6	SI-MON4/EX-SON																										
7	SI-MON5/E-STOP																										
输出点																											
0	EX-OUT1																										
1	EX-OUT2																										
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 <a href="#">简易例程</a> IF NODE_COUNT(0)>0 THEN <b>DRIVE_IO</b> (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																										
相关指令	<a href="#">NODE_IO</a> , <a href="#">DRIVE_PROFILE</a>																										

## 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 <b>ATYPE</b> （设置为 65/66/67）与 <b>DRIVE_PROFILE</b> 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	<pre> SLOT_SCAN(0) IF NODE_COUNT(0)&gt;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     ? <b>DRIVE_TORQUE</b> (0)   '打印轴 0 的力矩 ENDIF </pre>
相关指令	<a href="#">ATYPE</a> , <a href="#">DRIVE_PROFILE</a>

## DRIVE\_FE -- 驱动器误差

类型	轴状态
描述	<p>驱动器的当前误差超限，对应数据字典 0x60F4。 必须设置正确的 <b>ATYPE</b>（设置为 65/66/67）以后才能设置。</p> <p>ZMC408SCAN 的 ATYPE=22 带反馈位置时有效，此时返回驱动器的接收和反馈位置偏差。</p>
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	<pre> SLOT_SCAN(0)           '总线扫描 IF NODE_COUNT(0)&gt;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     ?<b>DRIVE_FE</b> (0)       '读取轴 0 设备的跟踪误差值 ENDIF </pre>
相关指令	<a href="#">DRIVE_FE_LIMIT</a>

## DRIVE\_FE\_LIMIT -- 驱动器误差限制

类型	EtherCAT 轴参数
描述	<p>驱动器的当前误差超限设置。 必须设置正确的 <b>ATYPE</b>（设置为 65/66/67）以后才能设置。预留。</p>
语法	<p>可读：var=DRIVE_FE_LIMIT(axis)</p> <p>可写：DRIVE_FE_LIMIT(axis)= value axis: 轴号</p>

适用控制器	带 EtherCAT 接口或 RTEX 接口
相关指令	<a href="#">DRIVE_FE</a>

## DRIVE\_CLEAR -- 清除报警

类型	总线指令								
描述	操作当前 <b>BASE</b> 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: <table border="1"> <thead> <tr> <th>值</th><th>描述</th></tr> </thead> <tbody> <tr> <td>0</td><td>清除当前告警</td></tr> <tr> <td>1</td><td>清除历史告警</td></tr> <tr> <td>2</td><td>清除外部输入告警</td></tr> </tbody> </table>	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
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 <b>DRIVE_CLEAR</b> (0)            '清除当前告警 ENDIF								
相关指令	<a href="#">DRIVE_READ</a> ， <a href="#">DRIVE_WRITE</a>								

## 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 <b>BASE</b> 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	箱位位置 1, 指令单位	0: CH1 箱位的电机的实际位置
	\$0A	箱位位置 2, 指令单位	0: CH2 箱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 <pre>IF NODE_COUNT(0)&gt;0 THEN   BASE(0)                '选择轴 0   <b>DRIVE_READ</b>(7*256+11,0) '读取 Pr7.11 参数数值, 保存到 vr(0)   ?vr(0)                  '打印 ENDIF</pre> 例二 <pre>IF NODE_COUNT(0)&gt;0 THEN   BASE(0)                '选择轴 0   <b>DRIVE_READ</b>(\$10000+\$01) '读取厂商名, 直接打印出来 ENDIF</pre>		
相关指令	<a href="#">DRIVE_WRITE</a> , <a href="#">DRIVE_CLEAR</a>		

## 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	描述	
	1	参数分类*256 + 参数编号（Pr7.20=7*256+20）	
	2	特殊参数=128，当前参数写入 EEPROM（此时 value=1）	
	3	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		设置为 1 时，根据 RTEX 通信指令 SL_SW 的数值进行选择
				0	1	
			[0]	Pr3.21		
			1	Pr3.21	Pr3.22	
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.回零第位模式

	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	<div>总线开启后才能使用</div> <div>例一</div> <div>IF NODE_COUNT(0)&gt;0 THEN</div> <div>BASE(0) '选择轴 0</div> <div>DRIVE_WRITE(7*256+11,6) '写 Pr7.11 参数为 6</div> <div>DRIVE_READ(7*256+11,0) '读取 Pr7.11 参数数值，保存到 vr(0)</div> <div>?vr(0) '打印，打印值 6</div> <div>ENDIF</div> <div>例二</div> <div>IF NODE_COUNT(0)&gt;0 THEN</div> <div>BASE(0) '选择轴 0</div> <div>DRIVE_WRITE(128,1) '将修改的参数写入 EEPROM</div> <div>ENDIF</div>							
相关指令	<a href="#">DRIVE_READ</a> ， <a href="#">DRIVE_CLEAR</a>							

## 第十八章 简易例程

### 18.1. 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，字母、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

```



```
        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

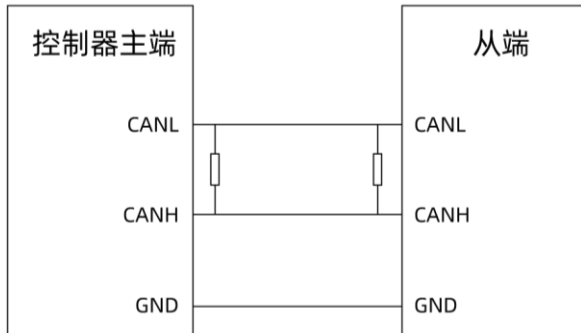
'延时
GLOBAL GSUB G04(P)
    TRACE "G04 entered"
    IF sub_ifpara(0) THEN
        DELAY (sub_para(0))
    ELSE
        ENDIF
END SUB

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
```

## 18.2. 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 间各连接一个 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
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,"aaaaaaaaa"
        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(VALUE) '转成字符串输出
```



## 18.3. 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
```

## 第十九章 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。

### 19.1. 常见问题列表

现象	排查方法
电机不动	<a href="#">手动运动调试</a>
触摸屏寄存器数据值不对	<a href="#">寄存器查看</a>
没有按照程序预期运行	<a href="#">断点调试+打印程序信息</a>
输入输出不起作用	<a href="#">IO 口快速检测</a>
机台抖动大	<a href="#">示波器抓取</a>

### 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)

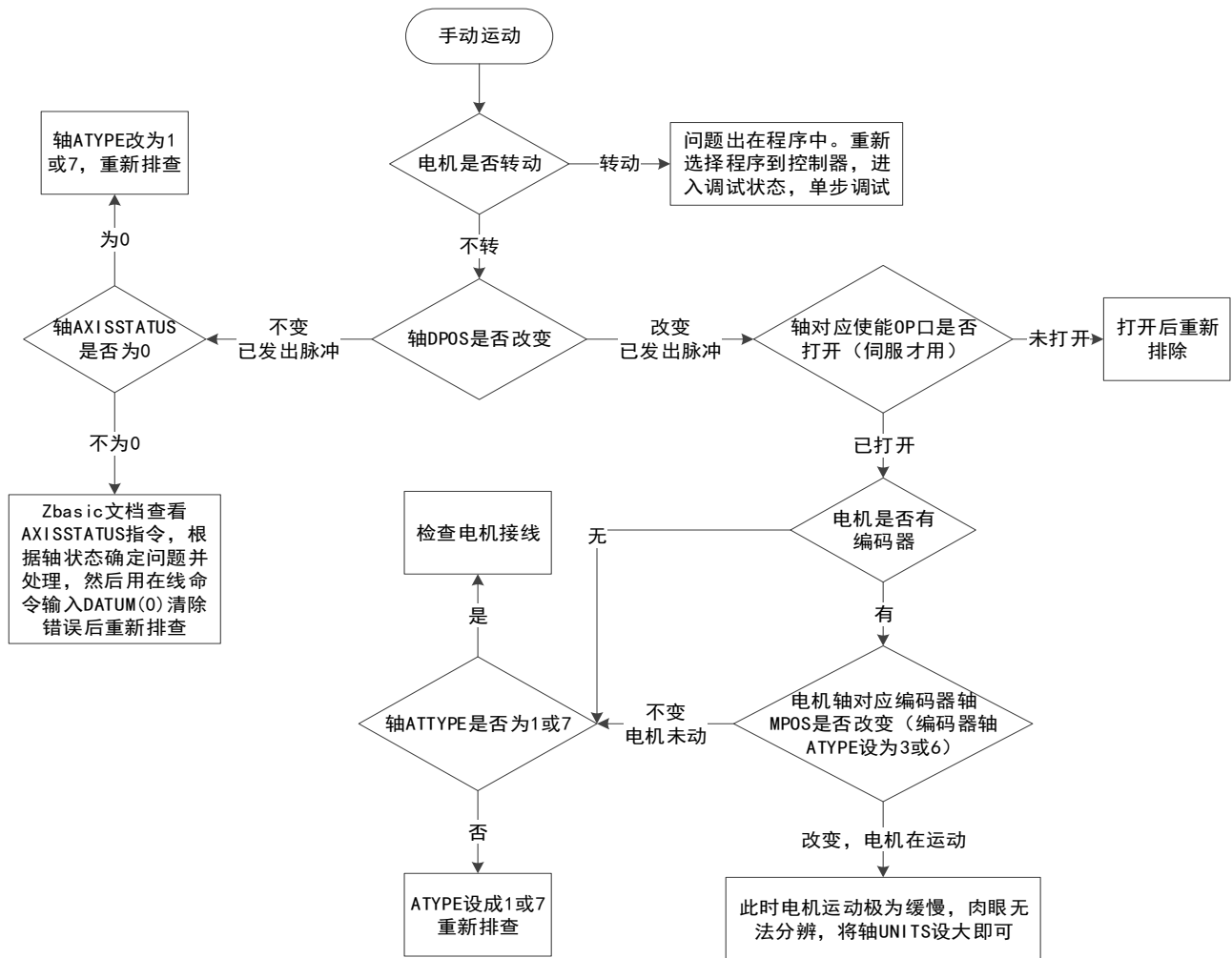
信号名称	记号	设定值	
		常开 (ON)	常闭 (NC)
无效	-	00h	设定不可
正方向驱动禁止输入	POT	01h	81h
负方向驱动禁止输入	NOT	02h	82h

## 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。

## 19.2. 解决办法

### 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。

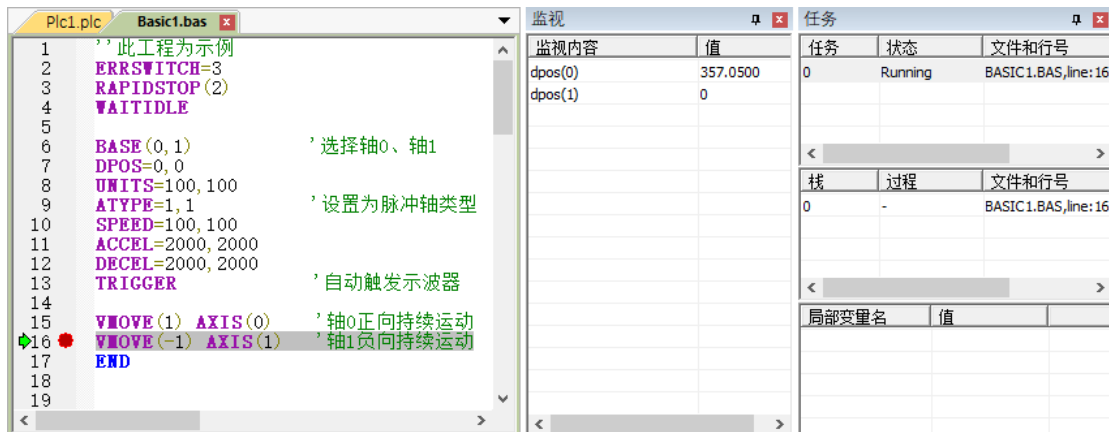
检查轴参数 AXISSTATUS 是否报警。

## 断点调试

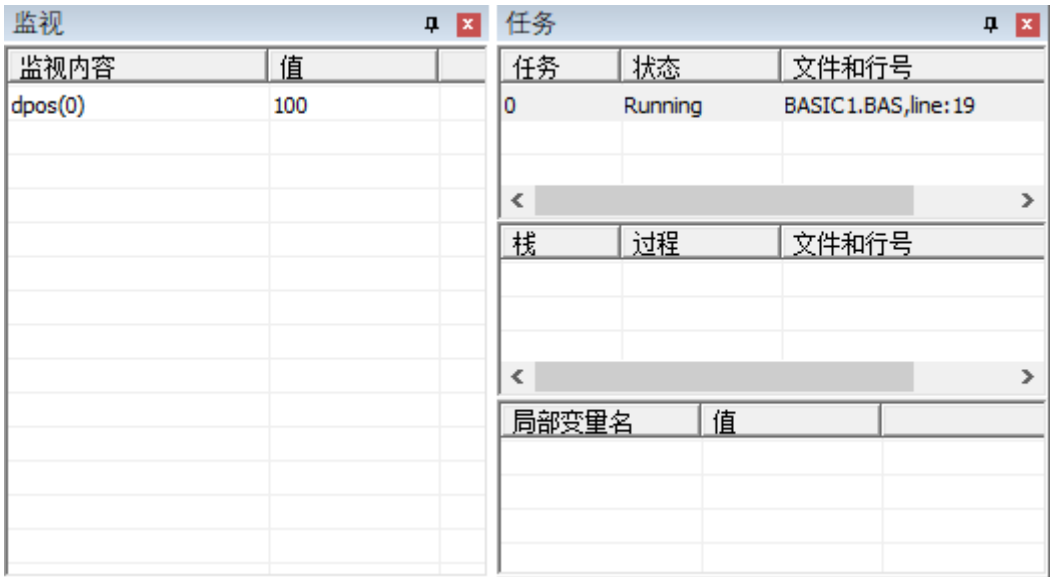
断点调试可以查看程序运行的具体过程，主要用于判断程序逻辑错误。配合监视内容可以查看程序每执行一步对寄存器、变量、数组等的影响。调试程序 and 控制器程序必须一致。

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](#)”章节。

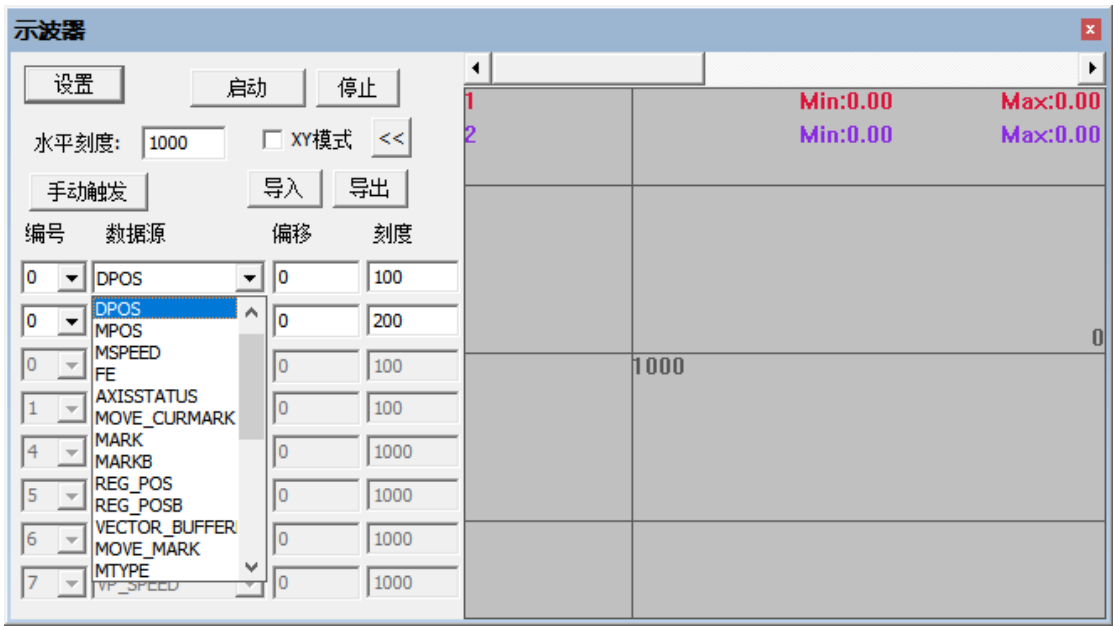


此时可以查看个任务运行情况、监视内容、子函数调用过程、子函数局部变量值。



示波器抓取

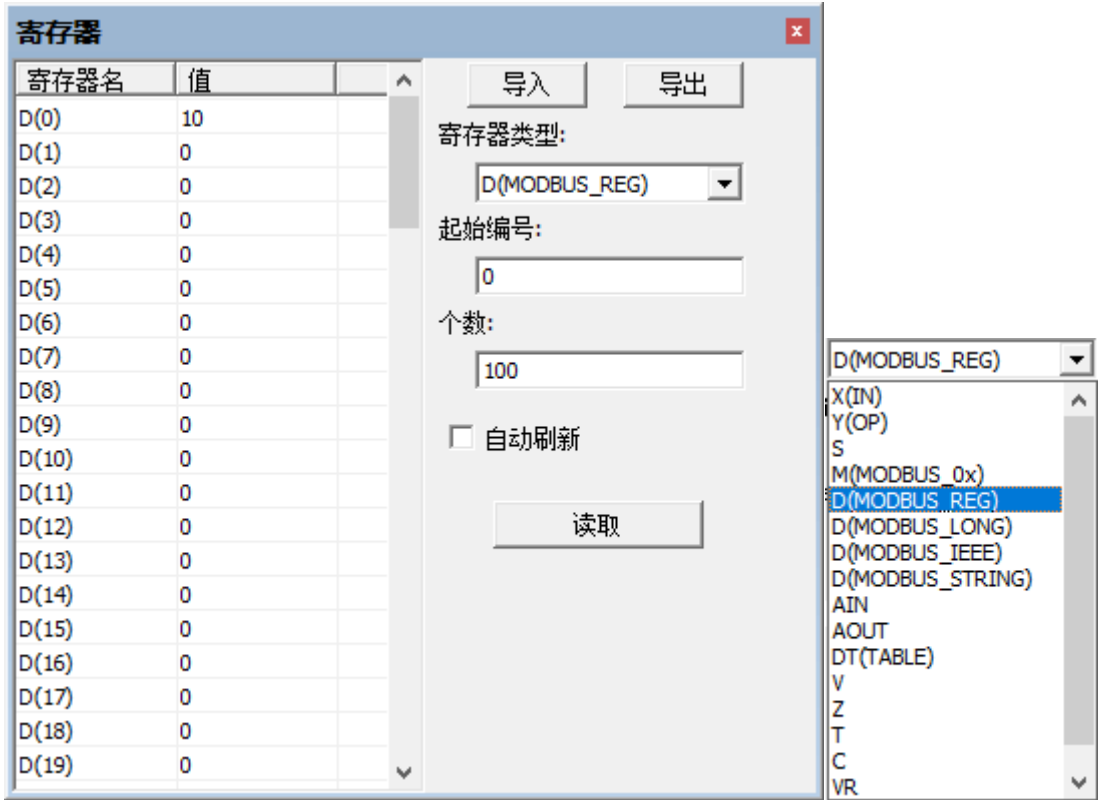
示波器可以抓取各类数据，数据具体种类查看“视图” - “示波器” - “数据源”。  
示波器抓取一般用来判断电机实际运行速度和实际位置。



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。

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。

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。

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
15
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

```

## 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（输入口和输出口的 EGND 端子内部是通的），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，不接电源无法输出。

如果检测扩展板 IO，先确认扩展模块与控制器能成功通讯（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。

## 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。

轴参数						
轴选择	参数选择					
	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 指令查看轴的各种状态。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。

3、AXIS\_STOPREASON 指令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。

## 附录一 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	
227	固件不支持	
228	文件打开失败	
229	创建网络连接失败	
230	bind 失败	
231	文件读取失败	
232	文件写入失败	
233	连接被加密	
234	固件检查错误	
235	文件删除错误	
236	路径错误	
237	文件关闭错误	
260	硬件错误, 出现下面错误时, led 灯会闪烁	
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	
289	复位太快	
290	驱动程序初始化出错	
291	fpga 错误	
292	mmap 资源不够	
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	检查是否不同类型变量定义重名引起，例如 2002 为局部变量未定义
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	在线命令阻塞时间过长	
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	
2131	数据类型没有定义	
2132	结构定义要在使用代码前面	
2133	静态定义的不能动态删除	
2134	需要数组类型	
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	
2200	函数回调没有完成	

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	MODBUS 从端返回不支持的功能码	
3422	MODBUS 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	
4512	末端不能相连	
4513	输出类型必须最末端	
4514	不支持的类型	
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	
5008	HMI 文件内容错误	
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	
5040	元件属性丢失	
5041	没有输入显示元件,无法输入	
5042	状态数过多	
5043	不支持的绘图属性	
5044	远程通讯设备名称错误	

5045	远程通讯数据没有更新	
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, 驱动器返回错误	数据字典读写错误或写入了驱动器不支持的数据
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	
6056	PDO 读写内容没有找到	
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	
6212	RTEX 设备类型错误	
6213	RTEX 消息超时	
6214	RTEX SDO 消息错误	
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	
20029	PCI 卡程序连太多	
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	

## 附录二 模块扩展

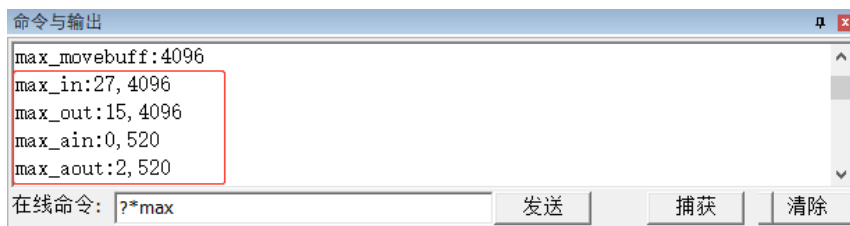
当控制器自身的轴资源、IO 资源不够用时，可采用扩展模块来扩展，可以扩展脉冲轴、数字量输入输出、模拟量输入输出这三种类型。只有带脉冲轴接口的扩展模块才支持扩展脉冲轴数，总线轴不可扩展。

**IO 数字量扩展：**4 系列及以上的 ZMC 控制器 IO 点数可扩展至 4096 点。

**AIO 模拟量扩展：**4 系列及以上的 ZMC 控制器 AIO 点数可扩展至 520 点。

**ZCAN 总线轴扩展：**只能扩展 4 个脉冲轴，不建议使用过多轴扩展板，可选用支持脉冲轴数较多的控制器型号。

控制器可扩展的最大 IO 点数可在硬件手册或“命令与输出”窗口输入 `?*max` 打印查看。



扩展模块按连接方式可分为 ZCAN 总线扩展模块和 EtherCAT 总线扩展模块两类，两类总线的扩展接线与资源映射方法不同。

按产品系列划分，可分为 ZCAN 扩展模块，EtherCAT 扩展模块、ZMIO300 扩展模块三大类，ZMIO300 系列的通讯模块可选 CAN 通讯模块或 EtherCAT 通讯模块。

所有的控制器型号都包含 CAN 总线接口，EtherCAT 接口只有支持 EtherCAT 总线的控制器型号支持。

扩展模块与控制器接线完成后，扩展的 IO 和轴资源还需要操作映射才能使用，CAN 总线扩展与 EtherCAT 总线扩展的映射方法不同，映射时需要映射的编号在整个控制系统中不得重复，当控制器或扩展模块的 IO 编号范围重复时，只有一个有效。

## ZCAN 扩展模块

### 扩展接线

CAN 总线上连接了多个 CAN 扩展模块时，全部 CAN 通讯模块的 CANL 和 CANH 端口分别接到一起，CAN 总线的左右两端分别接入一个 120 电阻。

扩展模块 CAN 端子：

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

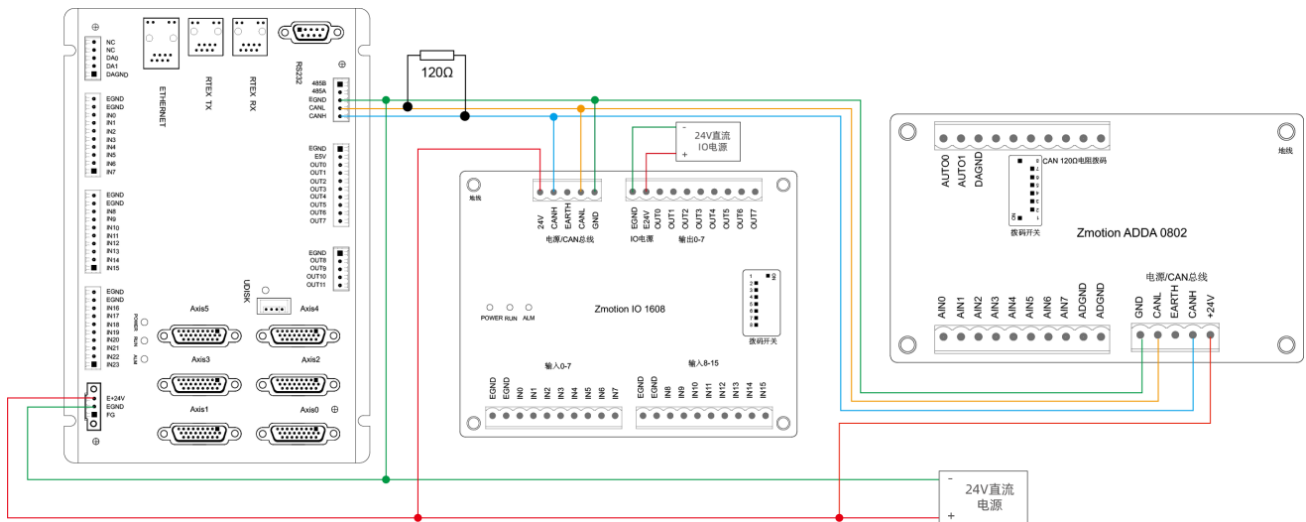
控制器连接 CAN 扩展模块的接线方法如下图所示，在控制器的 CANL 和 CANH 之间接入一个 120 欧

姆电阻，并将最后一个 CAN 通讯模块的拨码开关第 8 位拨为 ON(表示 CANL 与 CANH 端口之间接入一个 120 欧姆的电阻)，其他模块的第 8 位拨码开关无需操作，只需操作最末端的扩展模块。

CAN 通讯必须保证对应 GND 相连，或是控制器主电源和扩展模块主电源用同一个电源，防止扩展模块烧坏。

ZCAN 扩展接线参考如下：ZMC432+ZIO1608M+ZAI00802M，CAN 扩展时建议使用双绞屏蔽线，屏蔽层接地。

**ZIO 扩展模块为双电源供电，需要主电源和 IO 电源，不接 IO 电源无法使用；ZAI0 扩展模块为单电源供电只需要主电源。**



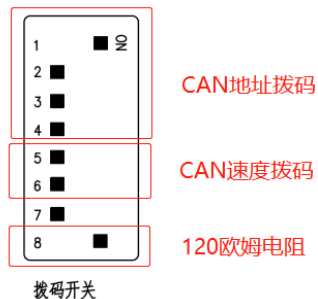
## 资源映射

ZCAN 扩展模块扩展的资源需要映射后才能使用，IO 映射采用扩展模块上自带的拨码开关设置，轴映射采用 AXIS\_ADDRESS 指令映射轴号。

数字量 IO 和模拟量的映射编号规则略有不同，参见下文说明。

## IO 映射

ZCAN 扩展板一般带 8 位拨码开关，拨 ON 生效，如下图所示，拨码含义如下：



1-4: 4 位 CAN ID 用于 ZCAN 扩展模块 IO 地址映射，对应值 0-15；

5-6: CAN 通讯速度，对应值 0-3，可选四种不同的速度；

7: 预留；

8: 120 欧姆电阻，拨 ON 表示 CANL 和 CANH 之间接入了 120 欧电阻。

拨码 1-4 选择 CAN 地址，控制器根据 CAN 拨码地址来设定对应扩展模块的 IO 编号范围，拨码每位 OFF 时对应值 0，ON 时对应值 1，地址组合值=拨码 4×8+拨码 3×4+拨码 2×2+拨码 1。

拨码开关必须在上电之前拨好，上电后重新拨码无效，需再次上电才生效。

不同地址对应数字量 IO 编号分配情况如下表，数字量起始 IO 映射编号从 16 开始，按 16 的倍数递增。

拨码 1-4 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

不同地址对应模拟量编号分配情况如下表，模拟量 AD 起始 IO 映射编号从 8 开始，按 8 的倍数递增。  
模拟量 DA 起始 IO 映射编号从 4 开始，按 4 的倍数递增。

拨码 1-4 组合值	起始 AD 编号	结束 AD 编号	起始 DA 编号	结束 DA 编号
0	8	15	4	7
1	16	23	8	11
2	24	31	12	15
3	32	39	16	19
4	40	47	20	23
5	48	55	24	27
6	56	63	28	31
7	64	71	32	35
8	72	79	36	39
9	80	87	40	43
10	88	95	44	47
11	96	103	48	51
12	104	111	52	55
13	112	119	56	59
14	120	127	60	63
15	128	135	64	67



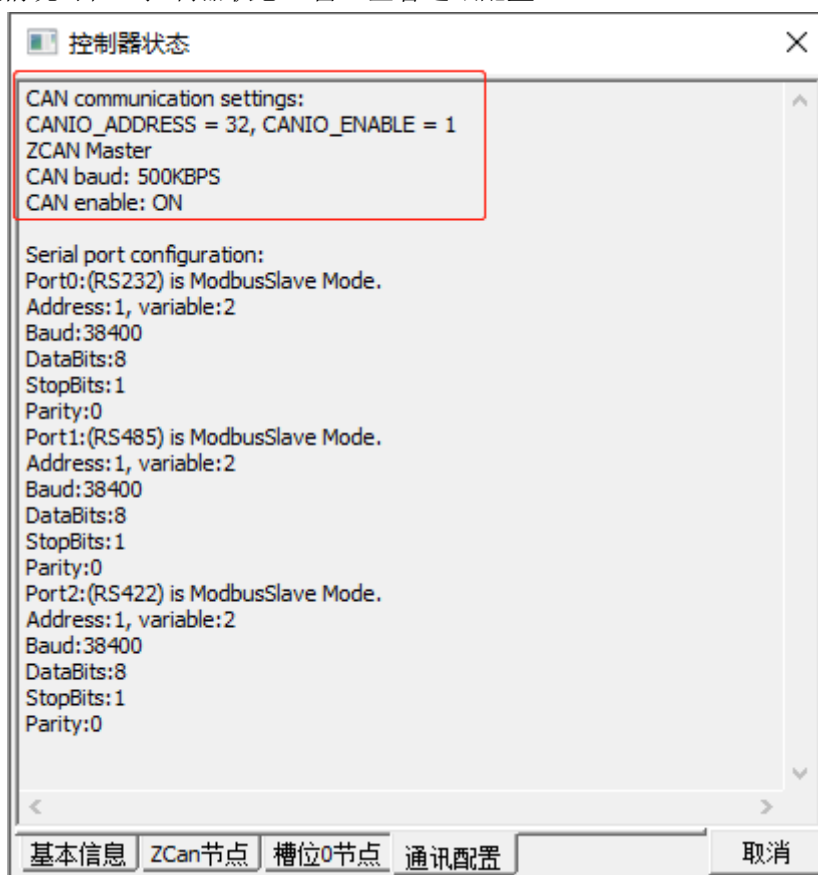
拨码 5-6 选择 CAN 总线通讯速度，速度组合值=拨码 6×2+拨码 5×1，组合值范围 0-3，对应的速度如下表：

拨码 5-6 组合值	CANIO_ADDRESS 高 8 位值	CAN 通讯速度
0	0（对应十进制 128）	500KBPS（缺省值）
1	1（对应十进制 256）	250KBPS
2	2（对应十进制 512）	125KBPS
3	3（对应十进制 768）	1MBPS

控制器端通过 CANIO\_ADDRESS 指令设置 CAN 通讯速度，同样也是有四种速度参数可供选择，需要与组合值对应的扩展模块的通讯速度一致才可以互相通讯。

CANIO\_ADDRESS 指令还可以设置 CAN 通讯的主从端，缺省值 32，做主端，设置为其他值便是做从端。

CAN 通讯配置情况可在“控制器状态”窗口查看通讯配置。



#### 拨码开关设置注意事项：

扩展模块拨码开关根据当前已包含 IO 点数的 IN 和 OP 最大者（外部 IO 接口数+脉冲轴内的 IO 接口数）。

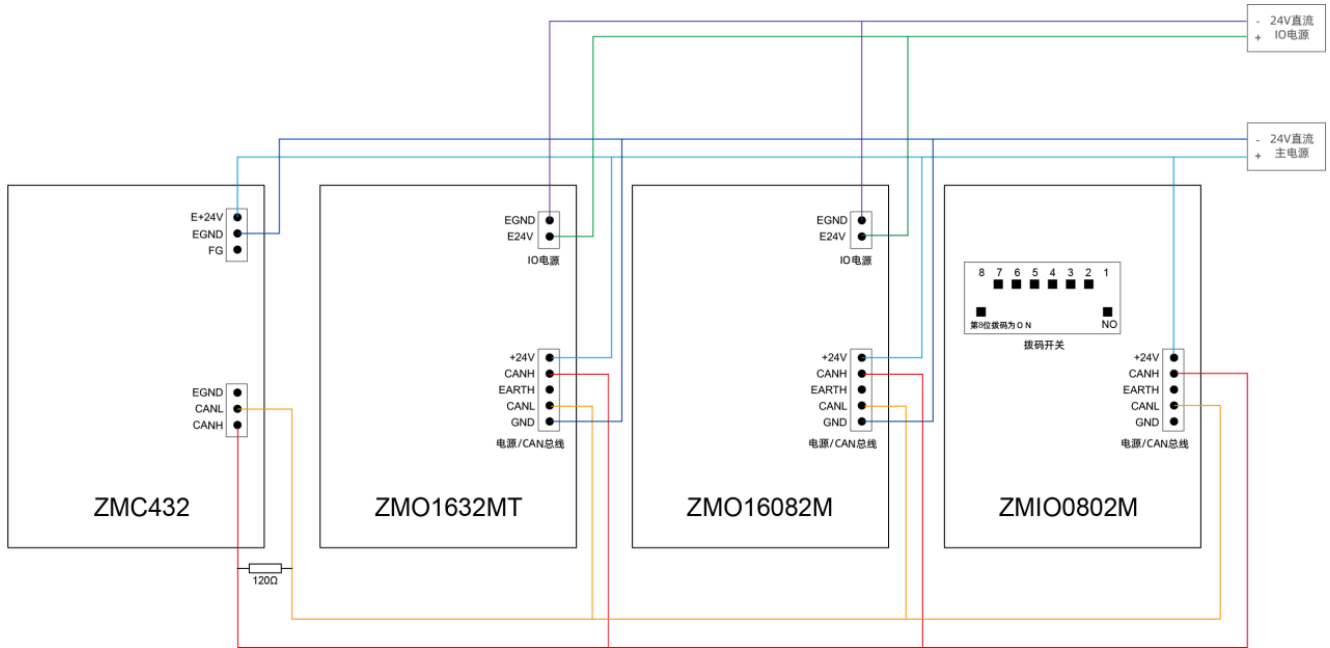
如控制器本身包含 28 个 IN，16 个 OP，那么第一个扩展模块设置的起始地址应超过最大值 28，按 IO 映射规则应将地址拨码设置为组合值 1（二进制组合值 0001，从右往左对应拨码 1-4，此时拨码 1 置 ON，其他置 OFF），此时扩展模块上的 IO 编号为 32-47，其中，29-31 空缺出来的 IO 编号舍去不用。

后续的扩展模块则依次按 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整个控制系统的 IO 编号均不重复。

### ZCAN 扩展模块 IO 映射配置示例:

控制模块配置: 1 个 ZMC432+1 个 ZIO1632MT+1 个 ZIO16082M+1 个 ZAIO0802M



CAN 接线方法参见上图, 正确设置每个模块的拨码 ID, 并将最后一个扩展模块的第八位拨码拨为 ON (表示 CANL 和 CANH 之间接入 120 欧姆电阻), 使用 ZDevelop 软件连接上控制器, 打开“控制器”-“控制器状态”窗口, 查看 ZCAN 节点信息, 可以看到 CAN 总线连接的全部设备的信息。

ZIO1632 的 CAN ID 设置为 1, 扩展的数字量输入 IO 编号为 32-47 共 16 个, 扩展的数字量输出 IO 编号为 32-63 共 32 个。

ZIO16082 的 CAN ID 设置为 3, 扩展的数字量输入 IO 编号为 64-79 共 16 个, 扩展的数字量输出 IO 编号为 64-71 共 8 个, 除此之外还带两个脉冲轴。

ZAIO0802 的 CAN ID 设置为 4, 扩展的模拟量输入 AD 编号为 40-47 共 8 个, 扩展的模拟量输出 DA 编号为 20-21 共 2 个。

控制器状态							
CanID	硬件ID	轴数	输入	输出	AD	DA	
Local	432-0(ZMC432)	32	30(0-29)	18(0-17)	0	2(0-1)	
1	48(ZIO1632)	0	16(32-47)	32(32-63)	0	0	
3	26(ZIO16082)	2	16(64-79)	8(64-71)	0	0	
4	10(ZAIO0802)	0	0	0	8(40-47)	2(20-21)	
<div> <div>基本信息</div> <div>ZCan节点</div> <div>槽位0节点</div> <div>通讯配置</div> </div> <div>取消</div>							

### 轴映射

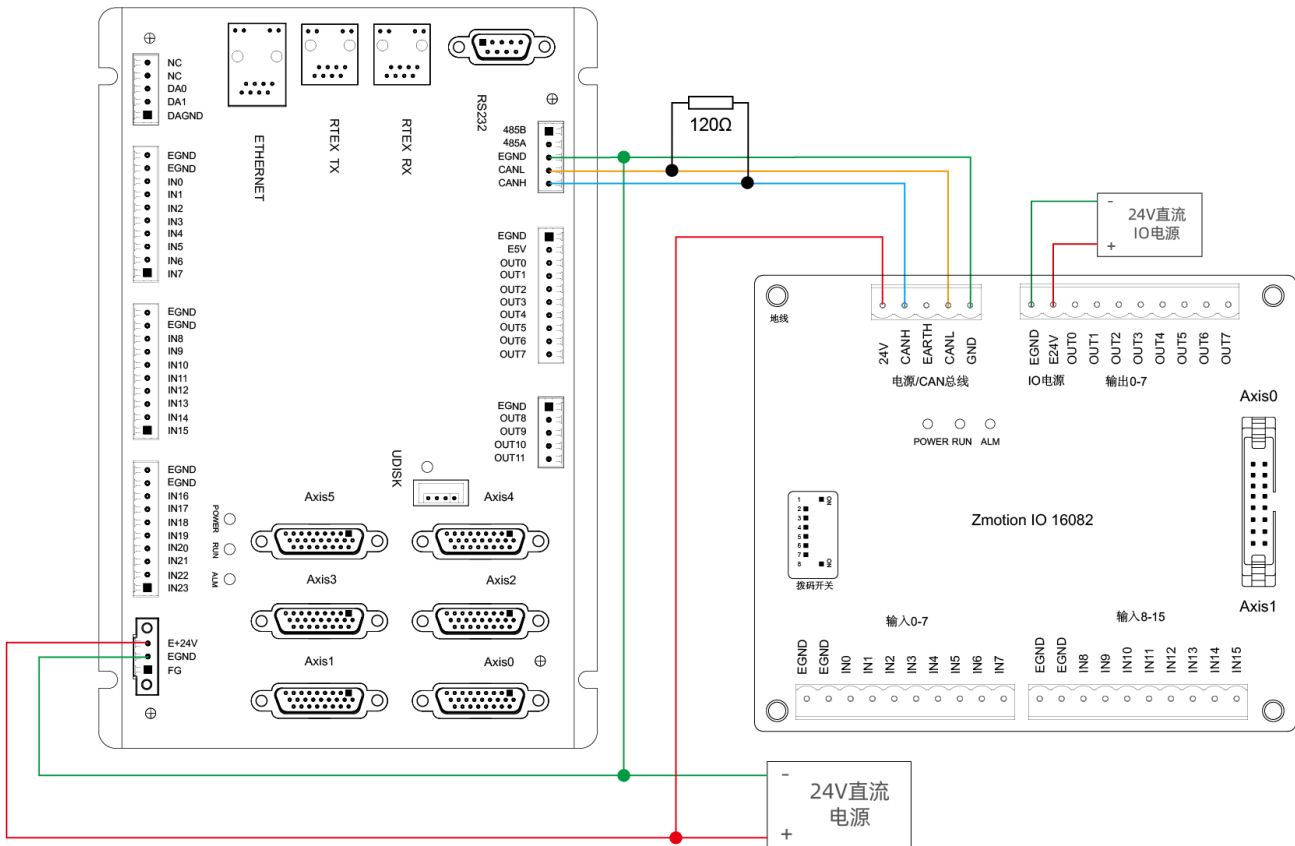
CAN 总线扩展方式扩展脉冲轴时, 可选 ZIO16082M, 扩展两个脉冲轴。

扩展轴需要进行轴映射操作, 采用 AXIS\_ADDRESS 指令映射, 映射规则如下:

AXIS\_ADDRESS(轴号)=(32\*0)+ID '扩展模块的本地轴接口 0

AXIS\_ADDRESS(轴号)=(32\*1)+ID '扩展模块的本地轴接口 1

ID 为扩展模块 1-4 位地址拨码的组合值。



映射完成设置 ATYPE 等轴参数后就可以使用扩展轴，示例：

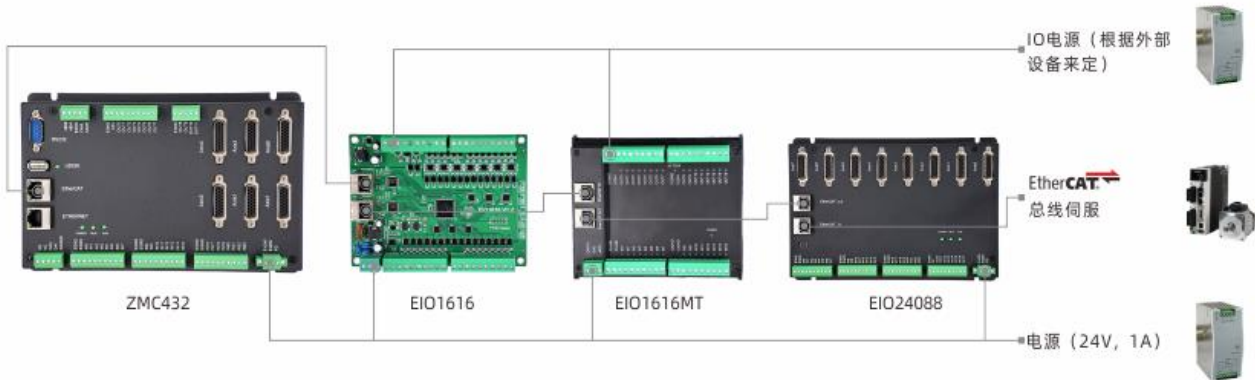
ATYPE(6)=0	'设为虚拟轴
AXIS_ADDRESS (6)=1+(32*0)	'ZCAN 扩展模块 ID 为 1 的轴号 0 映射到轴 6
ATYPE(6)=8	'ZCAN 扩展轴类型，脉冲方向方式步进或伺服
UNITS(6)=100	'脉冲当量 100
SPEED(6)=100	'速度 100units/s
ACCEL(6)=1000	'加速度 1000units/s^2
MOVE(100) AXIS(6)	'扩展轴运动 100units

## EtherCAT 扩展模块

### 扩展接线

EtherCAT 扩展模块接线只需将各个模块的 EtherCAT 口相互连接即可，EIO 系列扩展板带两个 EtherCAT 接口，EtherCAT 0 口接主控制器，EtherCAT 1 口接下级扩展板或驱动设备，不可混用。

EIO 扩展接线参考如下：ZMC432+EIO1616+EIO1616MT+EIO24088。



## 资源映射

EtherCAT 总线上的 IO 映射采用 NODE\_IO 指令(数字量)、NODE\_AIO 指令(模拟量)设置，轴映射采用 AXIS\_ADDRESS 指令映射轴号。

槽位号和设备号按照与控制器的连接顺序，从 0 开始自行编号。

## IO 映射

NODE\_IO 指令设置设备的数字量 IO 起始编号，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设置，NODE\_AIO 指令使用与 NODE\_IO 指令基本相同。

语法：

NODE\_IO(slot, node)=iobase

slot: 槽位号，0-缺省

node: 设备编号，编号从 0 开始

ioBASE: 映射 IO 起始编号，设置结果只会是 8 的倍数

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。0-缺省，同时设置 AIN、AOUT，读取时只读 AIN；3-AIN；4-AOUT

IO 映射示例：ZMC432 控制器上依次连接两个 EtherCAT 扩展模块，配置：1 个 ZMC432+1 个 EIO1616MT+1 个 ZMIO-4AD。

```

SLOT_SCAN(0)           '扫描总线
IF NODE_COUNT(0)>0 THEN '判断槽位 0 上是否有设备
    NODE_IO(0,0)=32     '设置槽位 0 接口设备 0 的 IO 起始编号为 32
    NODE_AIO(0,1,3)=8   '设置槽位 0 接口设备 1 的 AIN 起始编号为 8
ENDIF
    
```

## 轴映射

总线轴需要进行轴映射操作，采用 `AXIS_ADDRESS` 指令映射，操作方法如下：

`AXIS_ADDRESS(轴号)=(槽位号<<16)+驱动器编号+1`

轴映射写在总线初始化程序中，扫描总线之后，开启总线之前。

示例：

`AXIS_ADDRESS (0)=(0<<16)+0+1` '第一个 ECAT 驱动器，驱动器编号 0，绑定为轴 0

`AXIS_ADDRESS (2)=(0<<16)+1+1` '第二个 ECAT 驱动器，驱动器编号 1，绑定为轴 2

`AXIS_ADDRESS (1)=(0<<16)+2+1` '第三个 ECAT 驱动器，驱动器编号 2，绑定为轴 1

`ATYPE(0)=65` '设置为 ECAT 轴类型， 65-位置 66-速度 67-转矩

`ATYPE(1)=65`

`ATYPE(2)=65`

## 附录三 触摸屏通讯

### 控制器与触摸屏通讯简介

控制器与触摸屏一般通过串口或网口连接，控制器的串口和网口采用 MODBUS 协议，只要支持 MODBUS 通讯协议的触摸屏都可以与正运动控制器连接使用，也可使用正运动自主研发的 ZHD 系列触摸屏，ZHD400X 触摸屏如下图。



控制器使用 MODBUS 协议与第三方触摸屏通讯时，此时需要将数据放在 MODBUS 寄存器内进行传递，控制器搭配 ZHD 系列触摸屏编程更为灵活、自由。

控制器的 MODBUS 地址与其他厂家的触摸屏地址映射关系有所不同，控制器与触摸屏 modbus 寄存器地址关系如下：

控制器的 MODBUS 地址从 0 开始，在与威纶触摸屏通讯时，地址都是从 0 开始，所以是一一对应。

控制器 MODBUS\_BIT(0)对应威纶触摸屏 MODBUS\_0X\_0，布尔型。

控制器 MODBUS\_REG(0)对应威纶触摸屏 MODBUS\_4X\_0，字寄存器。

在与昆仑通态触摸屏通讯时，昆仑通态地址从 1 开始，控制器地址从 0 开始，所以触摸屏地址加 1。

控制器 MODBUS\_BIT(0)对应昆仑通态触摸屏 MODBUS\_0X\_1，布尔型。

控制器 MODBUS\_REG(0)对应昆仑通态触摸屏 MODBUS\_4X\_1，字寄存器。

控制器端程序可使用 ZDevelop 软件支持的 Basic 语言或 PLC 梯形图编程，ZHD 系列触摸屏的程序则用 ZDevelop 软件的 HMI 语言编程。

MODBUS 寄存器的说明参见前面章节“寄存器”→“[MODBUS](#)”。

## 控制器与触摸屏连接

控制器连接触摸屏使用的一般流程：

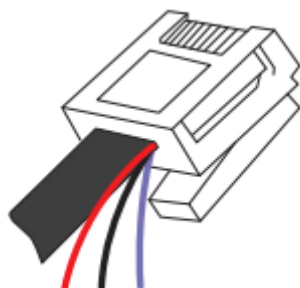
1. 控制器端的程序使用 ZDevelop 软件编写完成下载到控制器内。
2. 触摸屏端的程序使用对应的编程软件编写完成后下载到触摸屏保存。
3. 程序下载完成之后，选择串口或网口连接触摸屏与控制器脱机运行

### 连接 ZHD 系列触摸屏

正运动控制器连接正运动 ZHD300X 和 ZHD400X 系列触摸屏使用十分便捷，触摸屏程序可下载到控制器，下面是 ZHD400X 的使用方法，ZHD300X 和 ZHD400X 不同之处在于，前者 RS232 串口通讯，后者网口通讯，其他配置方法均相同。

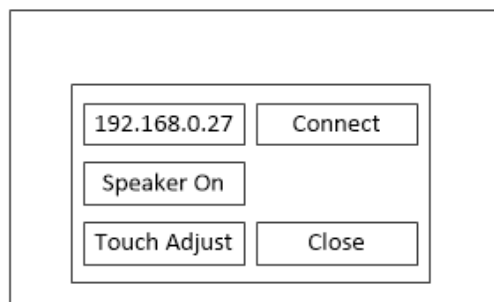
ZHD400X 触摸屏配一根网线，如下图，将此网线连接到控制器的 EtherNET 网口，网线水晶头边上引出三根线，分别是示教盒电源线和急停信号线，红色为 24V 电源正极，黑色为 24V 电源负极，紫色为急停信号线。

触摸屏和控制器的电源可共用一个。



#### 触摸屏连接控制器使用步骤：

1. 先使用 ZDevelop 软件编辑好 HMI 程序，连接控制器，将程序下载到 ROM 掉电保存，就可以断开控制器和 ZDevelop 的连接。然后给触摸屏上电。
2. 直接使用配发的连接线将 ZHD400X 接到控制器的网口上，然后在屏上四个角，按画 Z 字顺序点击，连续 2 次，唤醒屏幕，弹出可以弹出设置窗口，可以进行触摸校正，控制器 IP 修改等。

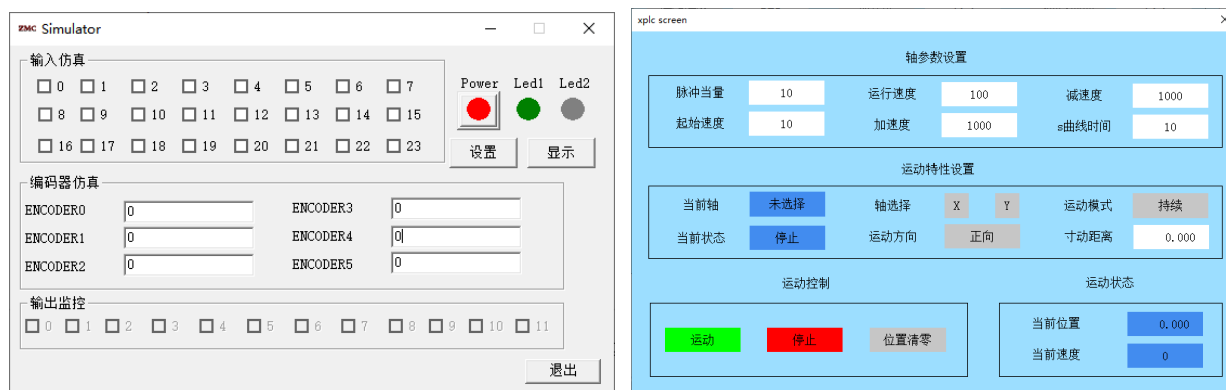


3. 设置窗口如下，在弹出的窗口上自动获取到当前所连的控制器 IP 的地址，确认 IP 无误，点击 Connect 即可连接使用，此时触摸屏显示起始基本窗口的内容。

4. 若没有实物触摸屏，可将 HMI 程序下载到仿真器，在 XPLC screen 平台仿真。



连接仿真器下载程序之后，点击“显示”按钮即可弹出仿真界面。



## 连接第三方触摸屏

支持标准 MODBUS 协议的触摸屏都可以与正运动控制器通讯，将通讯数据放在 MODBUS 寄存器中传递，支持通过串口或网口连接到控制器。

触摸屏和控制器建立通讯连接的时候，主要在触摸屏端操作连接，连接时对应的串口或网口参数要匹配。

通讯时可用寄存器类型有如下几种：MODBUS\_BIT（布尔型），MODBUS\_REG(16 位整型，MODBUS\_LONG(32 位整型)，MODBUS\_IEEE(32 位浮点型)，MODBUS\_STRING(8 位字节型)。

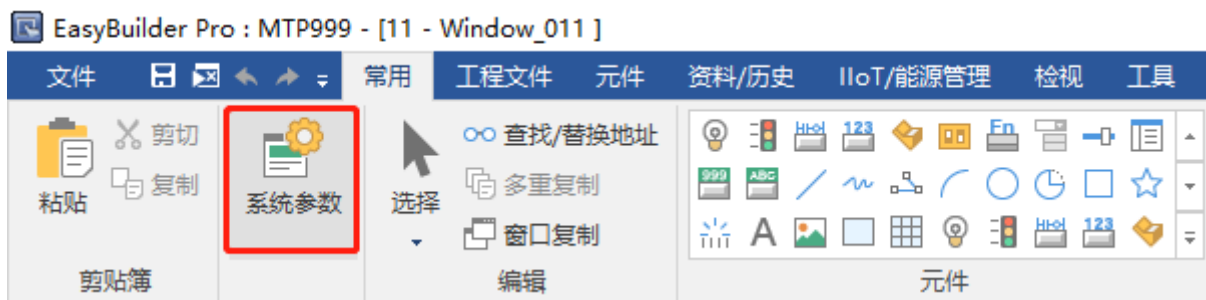
控制器和第三方触摸屏的通讯实例：以控制器和威纶屏通讯为例展开触摸屏的使用说明。

### 1. 下载控制器程序

控制器端的程序使用 ZDevelop 软件编写完成并下载到控制器内。详细步骤参见“新建项目运行程序”章节。

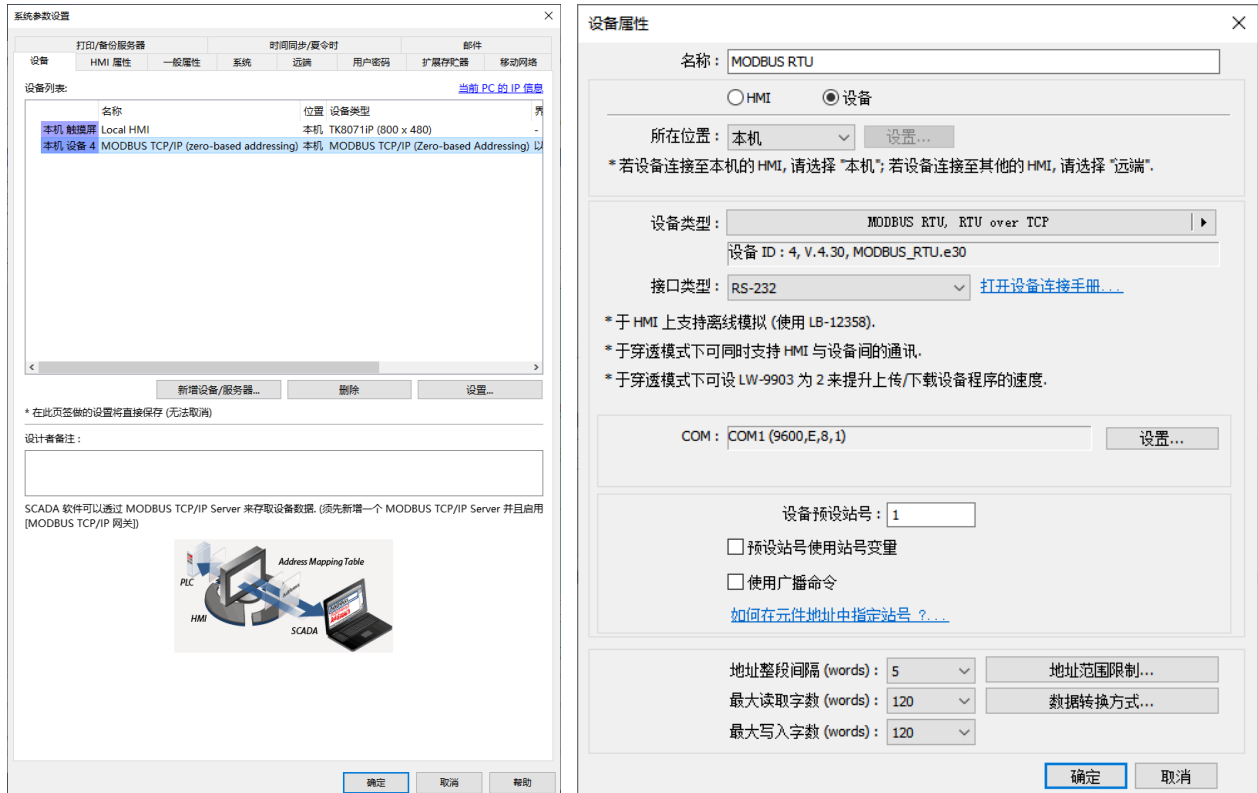
### 2. 下载触摸屏程序

威纶触摸屏端的程序使用 EasyBuilder 编程软件编写，程序编程完成后，打开“系统参数设置”窗口，如下图。



#### 1) 添加要与触摸屏连接的设备

设备列表里会显示本机触摸屏和本机设备，若有本机设备双击该行，若没有本机设备，点击下图“新建设备/服务器...”，弹出设备属性窗口。



## 2) 设置设备属性

如上图所示，选择设备类型，先选 MODBUS IDA 通讯协议，再根据触摸屏与控制器的实际连接方式选择。

串口通讯和网口通讯所选的设备类型不同，详见后续说明。

### 若采用串口连接：

设备类型：选择模式 MODBUS RTU(Zero-BASEd Addressing)；

接口类型：选择串口类型（RS485 或 RS232）；

COM：通讯端口设置匹配的波特率等参数，如下图，此时参数必须与连接到控制器的端口参数一致，设置完成确认关闭系统参数设置窗口。



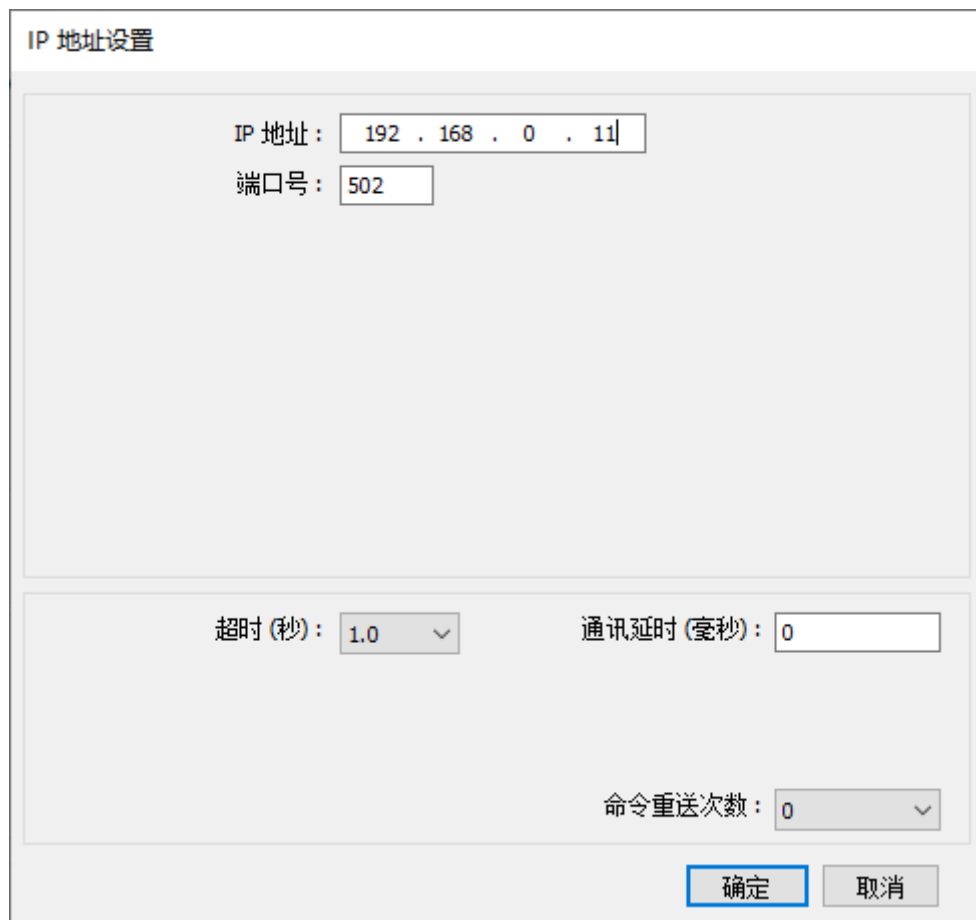


若采用网口连接:

设备类型: 选择模式 MODBUS TCP/IP(Zero-BASEd Addressing), 接口类型自动改为以太网;

IP: 填入当前要连接的控制器的 IP 地址和端口号, 如下图;

设置完成确认关闭系统参数设置窗口。



IP 地址设置对话框，包含以下配置项：

- IP 地址: 192 . 168 . 0 . 11
- 端口号: 502
- 超时 (秒): 1.0
- 通讯延时 (毫秒): 0
- 命令重送次数: 0
- 底部按钮: 确定、取消

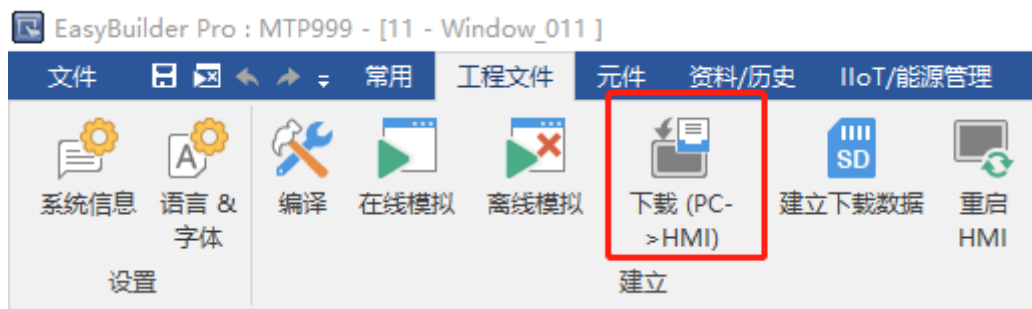
系统参数设置完成后, 编译写好的组态程序, 点击“编译”, 打开如下编译窗口。



点击右下角“开始编译”按钮, 编译成功打印信息提示, “开始编译”按钮变成“编译”按钮, 程序不正确是编译窗口能打印出错误信息, 修改程序知道编译成功。



程序编译成功, 将触摸屏连接到 PC 下载程序。



点击下载按钮, 弹出下方界面, 通过以太网将程序下载到触摸屏, 下载完成后, 程序已写入触摸屏, 可以断开触摸屏与 PC 的连接。

### 3. 触摸屏和控制器通讯

在控制器端的程序成功下载到控制器后, 和触摸屏端的程序成功下载到触摸屏之后, 可与 PC 断开连接, 连接触摸屏与控制器, 此时触摸屏与控制器就可以相互通信了。

#### 4. 控制器与触摸屏脱机仿真

若没有控制器或触摸屏，可采用仿真器仿真，ZDevelop 程序下载到仿真器内，只支持网口连接仿真，按照上面的步骤，EasyBuilder 软件的系统参数设置时选择设备类型为 MODBUS IDA—MODBUS TCP/IP(Zero-BASEd Addressing)，IP 地址填入仿真器 IP：127.0.0.1，选择“在线模拟”即可连接控制器程序与组态程序进行仿真。



点击在线模拟之后自动开始编译，编译结果正确打开如下触摸屏仿真界面，此时可以操作。编译不成功会报错错误信息。

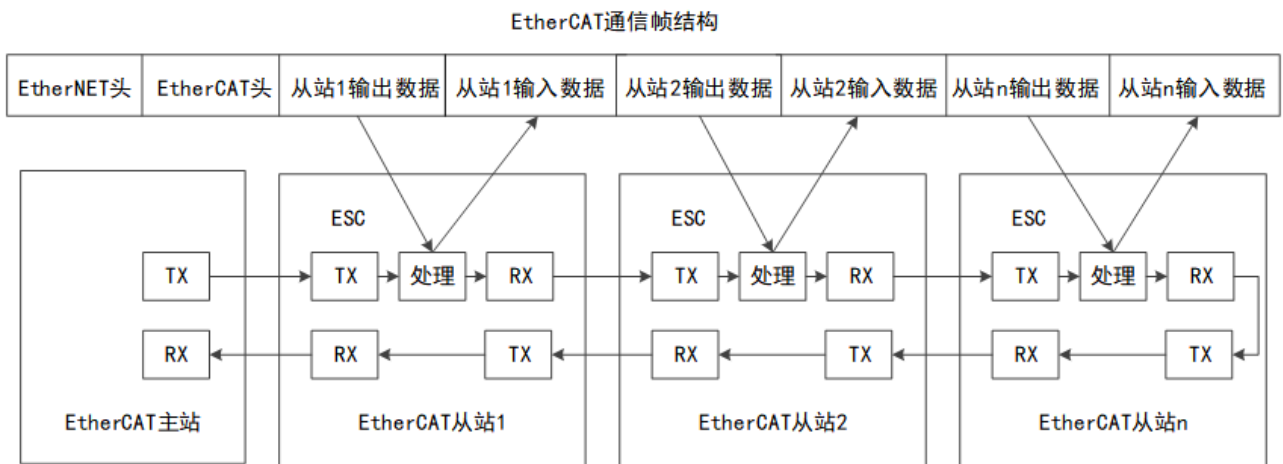
触摸屏仿真界面：



## 附录四 ETHERCAT 通讯

EtherCAT 总线是基于以太网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。

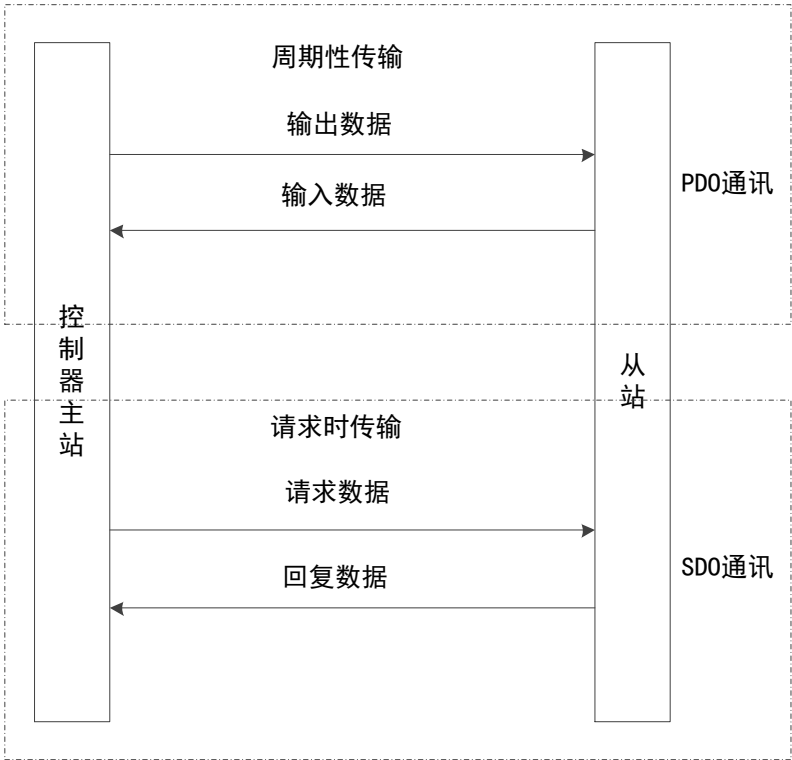
EtherCAT 充分利用了以太网的全双工特性，使用主从模式介质访问控制。EtherCAT 网络和普通以太网有明显不同，同一个 EtherCAT 网络内，只有一个 EtherCAT 主站，另外 EtherCAT 从站有专门处理 EtherCAT 通讯数据的芯片 ESC(EtherCAT Slave Controller)。ESC 芯片可以在 EtherCAT 数据帧通过时，取出主站发送给该从站的数据，并将该从站需要传送给主站的数据插入到 EtherCAT 数据帧中，网络中的最后一个 EtherCAT 从站 ESC 自动闭环，将处理过的报文依次返回给主站，数据传输示意图如下图所示。



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)

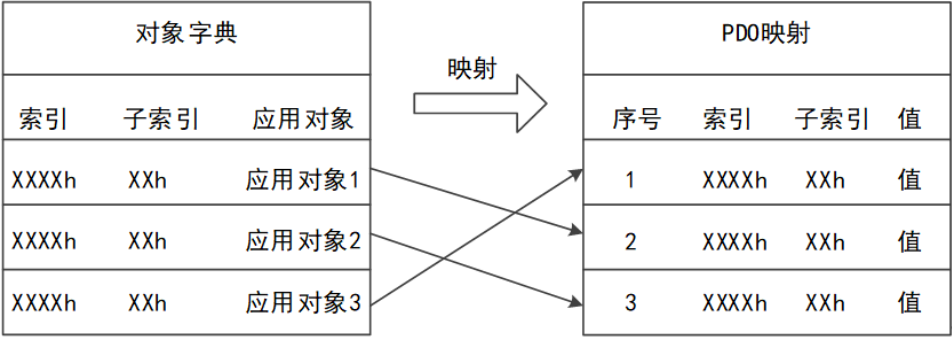
PDO 全名为 Process Data Object，指在 EtherCAT 总线网络中周期的进行主站与从站的数据交互的功能，PDO 数据用于周期性数据读取和控制，读写速度快。主站和从站通过 PDO 进行数据交换时，一方发送数据后，另一方不需要应答。控制器通过运动指令控制 EtherCAT 从站时，控制器和从站之间通过 PDO 方式进行数据交换。

EtherCAT 初始化过程中必须进行驱动器 PDO 配置，DRIVE\_PROFILE 指令配置驱动器的 PDO 列表，目前提供约 20 几种配置选择，每种配置包含哪些数据字典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。

PDO 列表可以看作一个数组空间，每个数组元素存放了不同的功能码，PDO 在一个周期中执行这些功能码对应的操作，这些功能码就叫做数据字典，数据字典用 4 位 16 进制数来表示，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。

PDO 中的所有传送数据必须由对象字典中映射进来：  
配置完后 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。数据字典的读写可 PDO 列表的自定义配置通过指令 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；客户端和服务端均可以主动终止 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。

## 数据字典

EtherCAT 通讯操作对象字典，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，多个数据对象组合成一个数据字典，又称 PDO 列表。

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。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

索引 1600h~17FFh 用以 RxPDO 映射的设定，设定完成分配到 1C12h，索引 1A00h~1BFFh 用于 TxPDO 映射的设定，设定完成分配到 1C13h。

常用数据字典参考：

索引	子索引	名称	数据范围	数据类型	读写	PDO	控制模式	EEPROM
6040h	00h	控制字	0-65535	U16	RW	RxPDO	全部	NO
6041h	00h	状态字	0-65535	U16	RO	RxPDO	全部	NO
6060h	00h	控制模式设置	-128~127	I8	RW	RxPDO	全部	YES
6061h	00h	控制模式查看	-128~127	I8	RO	TxPDO	全部	NO
6071h	00h	目标力矩	-32768~32767	I16	RW	RxPDO	tq, cst	YES
6072h	00h	最大力矩	0-65535	U16	RW	RxPDO	全部	YES
6077h	00h	实际力矩	-32768~32767	I16	RO	TxPDO	全部	NO
607Ah	00h	目标位置	-2147483648~2147483647	I32	RW	RxPDO	pp, csp	NO
607Eh	00h	电机极性	0-255	U8	RW	NO	全部	YES
6091h	01h	电子齿轮比分子	1~4294967295	U32	RW	NO	全部	YES
	02h	电子齿轮比分母	1~4294967295	U32	RW	NO	全部	YES
6098h	00h	回零模式	-128~127	I8	RW	RxPDO	hm	YES
60FDh	00h	数字输入	0~4294967295	U32	RO	TxPDO	全部	NO
60FDh	01h	数字输出	0~4294967295	U32	RW	RxPDO	全部	YES
60FFh	00h	目标速度	-2147483648~2147483647	I32	RW	RxPDO	pv, csv	NO

不同位的值表示的功能参见驱动器手册的描述去设置，部分参数设置完成需要写入到驱动器的掉电存储器中，并重启驱动器生效。

# 附录五 RTEX 总线

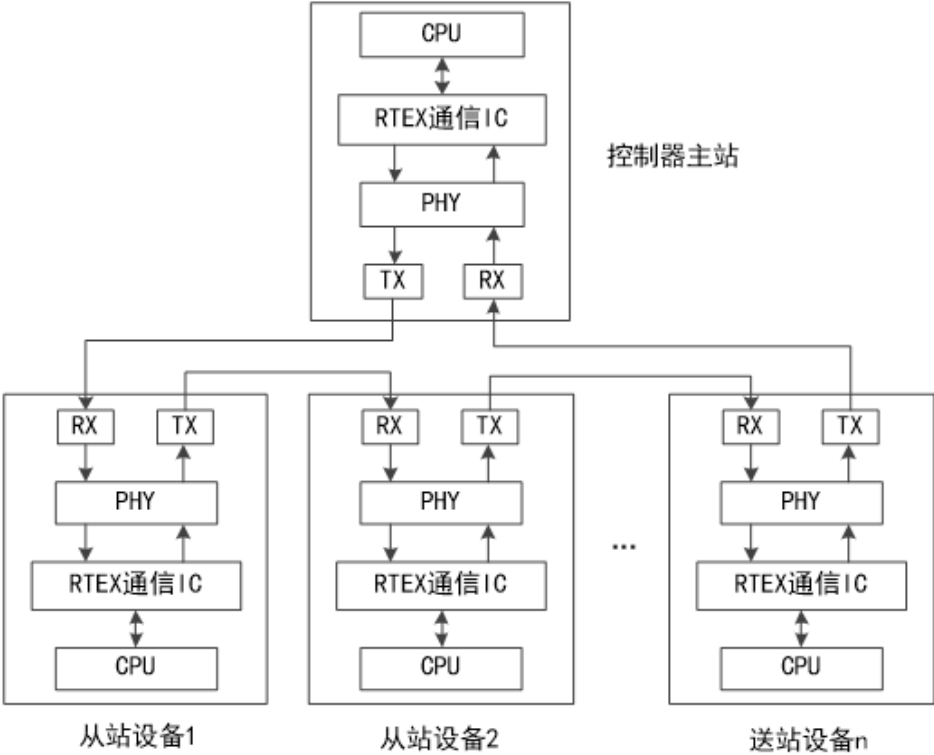
正运动控制器部分型号支持 RTEX 总线,支持 RTEX 总线轴、EtherCAT 总线轴、脉冲轴联合插补,RTEX 总线是松下自主开发出高速网络化的 RTEX 总线,适用于小系统的实时总线,小型设备组成的流水线更具有快速应变的优势。

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。

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。



RTEX 参数读写使用 DRIVE\_READ 指令和 DRIVE\_WRITE 指令操作下方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 <sup>23</sup>	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 <sup>30</sup>	—	设定电子齿轮比的分子
0	10	C	电子齿轮分母	0~2 <sup>30</sup>	—	设定电子齿轮比分母
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 PSL/NSL 会 ON (=1)
7	12	A	负向软件限位值	-1073741823~ 1073741823	指令单位	必须正侧软件限位值 > 负侧软件限位值
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，否则会发 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）