leetro

专业运动控制

# P1 系列　PC_Based 脉冲式运动控制卡
# 编程手册

成都乐创自动化技术股份有限公司

# 目录

# 版权申明

## 成都乐创自动化技术股份有限公司

## 保留所有权利



| | |
|---|---|
|  | 通电或正在工作中的机器有危险！<br>使用者有责任在及其中设计应对型的出错处理及安全保护机制，乐创技术没有义务和责任对由此造成的、附带的或相应产生的损失负责。 |

# 联系方式

官方网站：http://www.leetro.com

微信公众号：cdleetro

服务热线：400-990-0289

技术支持：support@leetro.com

总部研发：成都市高新区科园南二路 1 号大一孵化园 8 幢 B 座

东莞销售：东莞市松山湖园区科技四路 2 号御豪轩大厦 1 栋 610

苏州销售：苏州市高新区狮山路 28 号苏州高新广场 1102

# 更新记录

| 版本号 | 修订日期 | 更新要点 |
|---|---|---|
| V1.0 | 2022 年 11 月 29 日 | 新建 |
| V1.1 | 2023 年 4 月 26 日 | 更新部分函数 |
| | | |
| | | |
| | | |

# 1. 前言

## 1.1. 手册用途

本手册作为 P1 系列产品使用技术资料，为自动化设备厂商的软件开发工程师构建 PC_Based 式机器自动化系统而设计。开发者利用 P1 SDK，在 Windows 操作系统的高级语言开发环境（如 VS2017）下，通过调用 DLL 接口函数，实现自动化设备运动轴和 I/O 的具体控制功能。

用户阅读及利用本手册，应基本掌握计算机高级语言（如 C++,C#）编程知识和编程工具的使用方法。同时应具备一定的自动化设备运动控制调试经验，对初次接触运动控制卡的人员，应掌握运动控制系统的基本知识后再实施代码工程的设计与开发。

## 1.2. 内容指引

- 为方便用户快速定位各编程指令的说明，本手册第 2 章建立了指令汇总。
- 第 3 章介绍如何在 windows 下开发自动化系统，如是熟练开发者可跳过第 3 章。
- 第 4 章介绍使用使用 P1 控制卡需要做的配置，作为使用 P1 控制卡的前提，第 4 章必要阅读。
- 其他章节为 P1 手册提供的各项功能，具体使用场景及说明请阅读相关章节。
- 在实际使用中，如对手册资料及产品编程需要进一步帮助，可联系我司销售人员或电邮至 support@leetro.com 寻求解决。

## 1.3. 关联资料

| 名称 | 说明 |
| --- | --- |
| 《用户手册-P1 系列运动控制卡》 | 详细介绍 P1 系列产品如何安装，接线及电气调试 |
| 《FAQ 手册-P1 系列运动控制卡》 | 专项介绍 P1 系列产品常用功能及常见问题排查方案 |
| 《产品资料-P1 系列运动控制卡》 | 对 P1 产品的综合性介绍，用以产品选型 |

# 2. 编程指令汇总

本章汇总了 P1 控制卡提供的所有编程指令，用户可以方便分类查找这些指令，每条指令都提供了超链接，可以快速定位到指令的详细说明。

| 系统初始化/退出 | |
|---|---|
| auto_set | 进入控制卡 |
| init_board | 对控制卡系统初始化 |
| keep_card_status | 关闭应用程序时保存卡状态或不保存卡状态 |
| exit_set | 退出控制卡 |
| **轴配置** | |
| set_outmode | 设置轴输出脉冲模式 |
| set_home_mode | 设置轴回零模式 |
| set_encoder_mode | 设置编码器反馈参数 |
| set_maxspeed | 设置轴最大速度 |
| enable_handwheel | 设置某个轴的编码器反馈具备手轮控制 |
| **I/O 配置** | |
| enable_el | 使能轴限位输入功能 |
| enable_org | 使能轴原点输入功能 |
| enable_alm | 使能轴报警输入功能 |
| set_el_logic | 设置对 EL 输入信号的响应逻辑 |
| set_org_logic | 设置对 ORG 输入信号的响应逻辑 |
| set_alm_logic | 设置对 A-ALM 输入信号的响应逻辑 |
| enable_card_alm | 使能 C-ALM 输入功能 |
| set_card_alm_logic | 设置对 C-ALM 输入信号的响应逻辑 |
| set_input_filter | 设置通用输入口软件滤波时间 |
| set_special_input_filter | 设置专用输入口软件滤波时间 |
| set_gelnputLogic | 设置对通用输入信号的响应逻辑 |
| **配置文件的方式** | |
| load_configfile | 加载控制卡配置文件 |
| **查询轴的状态** | |
| check_status | 查询单个轴的工作状态信息 |
| get_cur_dir | 查询单个轴的运动方向 |
| check_done | 查询单个轴是否处于运动中 |
| check_limit | 查询轴是否处于限位状态（限位输入被配置为有效时） |
| check_home | 查询轴是否处于原点状态（原点输入被配置为有效时） |
| check_alarm | 查询轴是否处于报警状态（报警输入被配置为有效时） |

| 查询专用输入的状态 | |
|---|---|
| check_card_alm | 查询卡报警专用输入口的状态 |
| check_sfr | 查询单卡上各专用输入口的状态（≤32 点时） |
| check_sfr_ex | 查询单卡上各专用输入口的状态（＞32 点时） |
| check_sfr_bit | 查询单卡上单个专用输入口的状态 |
| checkin_axis_INP | 查询轴伺服定位完成专用输入口的状态 |
| checkin_axis_SRDY | 查询轴伺服准备好专用输入口的状态 |
| read_axis_RST | 查询轴当前的报警清除的输出状态 |
| read_axis_SEVERON | 查询轴当前的使能输出状态 |
| **查询通用 I/O 的状态** | |
| checkin_byte | 查询卡上多个通用输入口的状态 |
| checkin_bit | 查询卡上单个通用输入口的状态 |
| checkin_ai | 查询卡上的模拟量输入 |
| get_geInputLogicStatus | 查询卡上单个通用输入点的逻辑状态 |
| check_outport_status | 查询单卡上多个通用输出口的状态 |
| **IO 控制** | |
| outport_bit_RST | 对伺服驱动器进行报警清除 |
| outport_bit_SEVERON | 对伺服驱动器进行轴使能或禁止 |
| outport_bit_CL | 对伺服驱动器的计数器进行清零 |
| outport_byte | 打开或关闭多个通用输出口 |
| outport_bit | 打开或关闭单个通用输出口 |
| **JOG 运动的编程** | |
| con_vmove | 以常速模式启动单轴 JOG 运动 |
| con_vmove2 | 以常速模式启动两轴 JOG 运动 |
| con_vmove3 | 以常速模式启动三轴 JOG 运动 |
| con_vmove4 | 以常速模式启动四轴 JOG 运动 |
| fast_vmove | 以变速模式启动单轴 JOG 运动 |
| fast_vmove2 | 以变速模式启动两轴 JOG 运动 |
| fast_vmove3 | 以变速模式启动三轴 JOG 运动 |
| fast_vmove4 | 以变速模式启动四轴 JOG 运动 |
| **定位运动的编程** | |
| set_conspeed | 设置单轴常速模式下的速度 |
| set_s_curve | 设置轴 S 型速度模式 |
| set_profile | 设置轴 T 型变速模式下的速度 |
| set_s_section | 设置轴 S 型变速模式下的速度 |
| con_pmove | 以常速模式启动单轴定位运动 |
| con_pmove_to | 以常速模式启动单轴定位运动（绝对式） |
| con_pmove2 | 以常速模式启动两轴定位运动 |

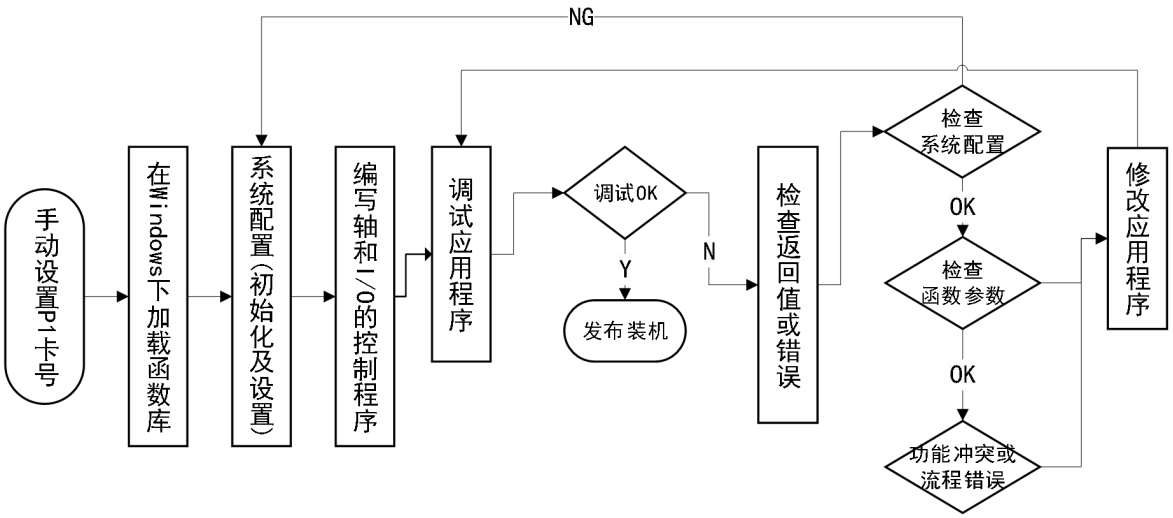
| | |
|---|---|
| con_pmove3 | 以常速模式启动三轴定位运动 |
| con_pmove4 | 以常速模式启动四轴定位运动 |
| fast_pmove | 以变速模式启动单轴定位运动 |
| fast_pmove_to | 以变速模式启动单轴定位运动（绝对式） |
| fast_pmove2 | 以变速模式启动两轴定位运动 |
| fast_pmove3 | 以变速模式启动三轴定位运动 |
| fast_pmove4 | 以变速模式启动四轴定位运动 |
| set_abs_pos | 设置单轴绝对位置 |
| reset_pos | 对单个轴位置清零 |
| get_abs_pos | 获取单个轴当前的位置 |
| get_rate | 获取单个轴当前的速度 |
| **直线插补运动的编程** | |
| set_vector_conspeed | 设置常速模式下直线运动的矢量速度 |
| set_vector_profile | 设置 T 型变速模式下直线运动的矢量速度 |
| con_line2 | 以常速模式启动两轴直线插补运动 |
| con_line3 | 以常速模式启动三轴直线插补运动 |
| con_line4 | 以常速模式启动四轴直线插补运动 |
| con_line2_to | 以常速模式启动两轴直线插补运动（绝对式） |
| con_line3_to | 以常速模式启动三轴直线插补运动（绝对式） |
| con_line4_to | 以常速模式启动四轴直线插补运动（绝对式） |
| fast_line2 | 以变速模式启动两轴直线插补运动 |
| fast_line3 | 以变速模式启动三轴直线插补运动 |
| fast_line4 | 以变速模式启动四轴直线插补运动 |
| fast_line2_to | 以变速模式启动两轴直线插补运动（绝对式） |
| fast_line3_to | 以变速模式启动三轴直线插补运动（绝对式） |
| fast_line4_to | 以变速模式启动四轴直线插补运动（绝对式） |
| **回零运动的编程** | |
| con_hmove | 以常速模式启动单轴回零运动 |
| con_hmove2 | 以常速模式启动两轴回零运动 |
| con_hmove3 | 以常速模式启动三轴回零运动 |
| con_hmove4 | 以常速模式启动四轴回零运动 |
| fast_hmove | 以变速模式启动单轴回零运动 |
| fast_hmove2 | 以变速模式启动两轴回零运动 |
| fast_hmove3 | 以变速模式启动三轴回零运动 |
| fast_hmove4 | 以变速模式启动四轴回零运动 |
| **运动轴的制动** | |
| sudden_stop | 对单个轴急停制动 |
| sudden_stop2 | 对两个轴同时急停制动 |
| sudden_stop3 | 对三轴轴同时急停制动 |
| sudden_stop4 | 对四个周同时急停制动 |

| | |
|---|---|
| decel_stop | 对单个轴缓停制动 |
| decel_stop2 | 对两个轴缓停制动 |
| decel_stop3 | 对三个轴缓停制动 |
| decel_stop4 | 对四个轴缓停制动 |
| move_pause | 运动轴暂停 |
| move_resume | 运动轴恢复（暂停后） |
| **位置比较输出** | |
| set_compare_outport | 设置轴对应的位置比较输出口 |
| start_comparePulse | 在 HSIO 口直接输出一个脉冲或者电平信号 |
| start_compareData | 设置非等间距模式的位置比较输出 |
| start_compareLinear | 设置等间距位置比较输出输出 |
| get_compareStatus | 读取已输出的位置比较信号数量，即比较过的点数 |
| compare_stop | 取消位置比较输出 |
| **高速位置锁存** | |
| enable_lock_enc | 使能高速锁存功能 |
| get_locked_flag | 查询是否发生锁存 |
| get_locked_encoder | 查询到控制器已锁存编码器值后，读取锁存值 |
| **反向间隙补偿** | |
| start_backlash | 启动反向间歇补偿 |
| end_backlash | 结束反向间隙补偿 |
| set_backlash | 设置反向间隙补偿 |
| **螺距误差补偿** | |
| set_leadscrew_comp | 设置螺距补偿参数 |
| enable_leadscrew_comp | 设置螺距补偿的使能与禁止 |
| **运动变速/变位** | |
| change_speed | 运动中变速度 |
| change_pos | 动态改变目标位置，目标位置为相对位置 |
| change_pos_to | 动态改变目标位置，目标位置为绝对位置 |
| **手轮功能** | |
| enable_handwheel | 使能轴为手轮模式 |
| **系统信息查询** | |
| get_max_axe | 获取总轴数 |
| get_board_num | 获取总卡数 |
| get_axe | 获取控制卡的轴数 |
| get_lib_ver | 获取函数库文件版本号 |
| get_sys_ver | 获取第一张卡的驱动程序版本 |
| get_sys_ver_ex | 获取控制卡驱动程序版本号 |
| get_card_ver | 获取控制卡固件程序版本号 |
| get_sn | 获取控制卡的序列号 |
| **参数设置查询** | |

| | |
|---|---|
| get_profile | 获取设定的单轴变速参数 |
| get_vector_conspeed | 获取设定的矢量常速参数 |
| get_vector_profile | 获取设定的矢量变速参数 |
| check_IC | 查询卡号是否设置正确 |
| read_input_filter | 查询通用输入口软件滤波设置 |
| read_special_input_filter | 查询专用输入口软件滤波设置 |
| **错误信息查询** | |
| get_err | 获取控制卡最近 10 条错误的信息 |
| get_last_err | 获取控制卡最进 1 条错误的信息 |
| reset_err | 清除控制卡的错误信息 |
| **自定义存储区操作** | |
| write_password_flash | 设置控制卡加密用户通用存储区的密码 |
| read_password_flash | 获取控制卡加密用户通用存储区的密码 |
| clear_password_flash | 清除控制卡加密用户通用存储区设置的密码 |
| write_flash | 写控制卡非加密通用存储区 |
| read_flash | 读控制卡非加密通用存储区 |
| clear_flash | 将指定片区数据擦写 bit 位置 1 |
| **其他** | |
| set_dir | 设置轴信号方向的电平 |
| outport_byte_ex | 空操作（兼容以前版本） |
| check_exoutport_status | 空操作（兼容以前版本） |

# 3. 应用程序开发

## 3.1. 总体开发流程




## 3.2. 在 Windows 下加载函数库

### 3.2.1在 Visual C#中的使用

(1) 启动 Visual Studio 2017，新建 C#工程；

(2) 将安装路径 C:\Program Files (x86)\Leetro\P1 Assistant\SDK\Common\Lib\x86（根据要编写工艺软件的运行平台选择 32 位或 64 位的文件夹目录）下所有.dll 文件；

（LtDrvPkgP1.dll,LtMcLogi6.dll,LtMcLv1P1.dll,LtMcP1.dll,LtMCServerClientP1.dll）复制到工程所在目录“..\bin\debug\”或“..\bin\release\”文件夹中；

(3) 将安装路径 C:\Program Files (x86)\Leetro\P1 Assistant\SDK\Common\Src 下所有.cs 文件（LtMCB.cs,LtMcLv1P1.cs,LtMcP1.cs,LtMCServerClientP1.cs）复制到工程文件夹；

(4) 在工程添加->添加现有项下，将步骤 3 复制的所有文件添加到工程；

(5) 在工程文件中加入命名空间：using Leetro;

(6) 至此，用户可以在 Visual Studio 2017 中使用 C#模块调用函数库中的任何函数，开始编写应用程序。

### 3.2.2在 Visual C++ 中的使用

(1)启动 Visual Studio 2017，新建一个 MFC 工程；

(2)将安装路径 C:\Program Files (x86)\Leetro\P1 Assistant\SDK\Common\Lib\x86（根据情况选择 32 位或 64 位）下所有.lib 文件复制到工程文件夹；

(3)将安装路径 C:\Program Files (x86)\Leetro\P1 Assistant\SDK\Common\Src 下所有.h 文件复制到工程文件夹；

(4)在工程添加->添加现有项下，将步骤 2、3 复制的所有文件添加到工程；

(5)将安装路径 C:\Program Files (x86)\Leetro\P1 Assistant\SDK\Common\Lib\x86（根据要编写工艺软件的运行平台选择 32 位或 64 位的文件夹目录）下所有.dll 文件（LtDrvPkgP1.dll,LtMcLogi6.dll,LtMcLv1P1.dll,LtMcP1.dll,LtMCServerClientP1.dll）复制到工程所在目录“..\bin\debug\”或“..\bin\release\”文件夹中；

(6)在应用程序文件中加入函数库头文件的声明，#include “LtMcLv1P1.h”；

(7)至此，用户可以在 Visual C++中调用函数库中的任何函数，开始编写应用程序。

# 4. 系统配置

P1 系列控制卡的各项编程操作，需要一系列的系统配置才能保证控制卡有效工作。系统配置主要包括系统初始化，轴的基础设置及 I/O 的配置项参数的设置以及与机器安全控制必要的设置。未执行必要设置或设置错误时，在应用程序调用库函数时将会报错，或给设备带来安全控制隐患。

指令调用注意事项：P1 控制卡运动指令以立即方式执行。立即方式指不等上一条运动指令控制的所有轴运动完毕即开始下一条运动指令的执行。若用在多条不同的轴运动指令连续执行时必须在每条运动指令后使用 check_done 判断轴处于停止状态后，才能对该轴发出下一条运动指令，若控制轴尚在运动，系统将抛弃后面的运动指令，造成某些指令无法执行。

控制卡初始化完成后，运动控制卡的各项参数恢复到下列的初始值：

- 各轴默认参数：

  脉冲输入模式：Pul/Dir（脉冲/方向）；

  编码器模式：A/B 90 度相位差，4 倍频；

  回零模式：原点有效立即停止；

  快速速度曲线：梯形；

  默认绝对位置值：0；

- 速度参数：

  点对点运动：1）常速：2000 pps　　2）快速：低速 2000pps、高速 8000pps、上升/下降加速度 80000pps

  直线插补运动：1）常速：2000 pps　2）快速：低速 2000pps、高速 8000pps、上升/下降加速度 80000pps

  速度限制：最大速度 100000pps

- 专用输入：

  Org（轴原点）：使能，低电平有效

  EL± （轴限位）：使能，低电平有效

  Alm（轴报警）：使能，低电平有效

  Card_Alm（板卡报警）：使能，低电平有效

- 通用输入：默认使能，低电平有效。

- 通用输出：默认输出高电平。

## 4.1. 相关指令列表

| 系统初始化/退出 | |
|---|---|
| auto_set | 进入控制卡 |
| init_board | 对控制卡系统初始化 |
| keep_card_status | 关闭应用程序时保存卡状态或不保存卡状态 |
| exit_set | 退出控制卡 |

| 轴配置 | |
|---|---|
| set_outmode | 设置轴输出脉冲模式 |
| set_home_mode | 设置轴回零模式 |
| set_encoder_mode | 设置编码器反馈参数 |
| set_maxspeed | 设置轴最大速度 |
| enable_handwheel | 设置某个轴的编码器反馈具备手轮控制 |
| I/O 配置 | |
| enable_el | 使能轴限位输入功能 |
| enable_org | 使能轴原点输入功能 |
| enable_alm | 使能轴报警输入功能 |
| set_el_logic | 设置对 EL 输入信号的响应逻辑 |
| set_org_logic | 设置对 ORG 输入信号的响应逻辑 |
| set_alm_logic | 设置对 A-ALM 输入信号的响应逻辑 |
| enable_card_alm | 使能 C-ALM 输入功能 |
| set_card_alm_logic | 设置对 C-ALM 输入信号的响应逻辑 |
| set_input_filter | 设置通用输入口软件滤波时间 |
| set_special_input_filter | 设置专用输入口软件滤波时间 |
| set_gelnputLogic | 设置对通用输入信号的响应逻辑 |
| 配置文件的方式 | |
| load_configfile | 加载控制卡配置文件 |

## 4.2. 系统初始化/退出

### 4.2.1应用说明

控制器初始化是正常调用其他指令的基础，本文后续的所有指令都需要在初始化成功后才能正常调用。P1 有 2 个系统初始化相关的指令，作为其他所有指令的前置指令，若要使用 P1 各项功能，在应用程序入口时应首先调用指令 auto_set 函数自动检测控制卡，执行正确返回卡上轴数。然后调用 init_board 函数为运动控制卡分配软硬件资源，调用成功后才能使用控制卡其它指令，否则不能使用控制卡函数资源。当应用程序退出时，应调用 exit_set 释放控制卡所占用的 PC 资源。

## 4.2.2调用示例

**控制器初始化及退出**

```
int mRtn;
mRtn = auto_set(); //自动设置
if (mRtn <= 0)
{
    return mRtn;//返回错误代码
}
else
{
    //自动设置成功，获得轴数
}
mRtn = init_board(); //初始化板卡
if (mRtn <= 0)
{
    return mRtn;//返回错误代码
}
else
{
    //初始化成功，获得卡数
}
//结束应用程序
mRtn = exit_set(); //退出控制卡
```

## 4.2.3指令详解

**4.2.3.1 auto_set**

指令原型：int auto_set();

```
//-----------------------------------------------------
// 功能描述：初始化控制卡，检测用户设置的板卡号是否正确。
// 输入参数：无
// 输出参数：无
// 返回值：>0 - 成功，表示可使用总轴数； <=0 - 失败
// 注意事项：系统调用板卡初始化函数“auto_set”后，假如返回值显示初始化失败。调用“get_err”“get_last_err”等函数可读取错误信息。然后根据错误信息提示解决问题后，再重新调用该函数。
//-----------------------------------------------------
```

**4.2.3.2 init_board**

指令原型：int init_board();

```
//-----------------------------------------------------
// 功能描述：必须调用init_board函数检测函数库、驱动程序、控制器固件版本号，初始化控制器所有寄存器为默认状态，init_board应在auto_set之后调用，为了防止用户误操作，auto_set和init_board只能调用一次。执行正确返回计算机内安装的控制器卡数。
// 输入参数：无
// 输出参数：无
// 返回值：>0 - 成功，表示总卡数； <=0 - 失败
// 注意事项：系统调用板卡初始化函数“auto_set”后，假如返回值显示初始化失败。调用“get_err”“get_last_err”等函数可读取错误信息。然后根据错误信息提示解决问题后，再重新调用该函数。
//-----------------------------------------------------
```

#### 4.2.3.3 keep_card_status

指令原型：int keep_card_status();

```
//-------------------------------------------------
// 功能描述：关闭应用程序时保存卡状态或不保存卡状态
// 输入参数：  cardno ---- 卡号
              flag ---- 保持标志，1(保持卡状态) ---- 0(不保存卡状态)
// 输出参数：无
// 返回值：0 - 成功；  1 - 失败
注意事项：
1)在未使用该函数前，默认不保存卡状态。
2)该函数可保存的状态：数字量输出状态、轴专用输出状态，轴输出模式、编码器位置、编码器模式(1/2/4倍频)、编码器计数方向、指令脉冲位置。
3)如果不保存状态：在调用auto_set后，数字量输出低、轴专用输出状态为低，轴输出模式为脉冲+方向、编码器位置清0、编码器模式为AB相90°相位差4倍频、指令脉冲位置清0。
//-------------------------------------------------
```

#### 4.2.3.4 exit_set

指令原型：int exit_set();

```
//-------------------------------------------------
// 功能描述：退出动态链接库需要调用exit_set函数退出函数库
// 输入参数：无
// 输出参数：无
// 返回值：0 - 成功；  <0 - 失败
//-------------------------------------------------
```

#### 4.2.3.5 enable_handwheel

指令原型：　int　enable_handwheel (int Axis,　int Mul, int Mode)

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// Mul---- 手轮跟随倍率，规定设为1，10，100
// Mode------ 使能控制标记，1-使能手轮控制，0-取消手轮控制
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：设置是否使能轴为手轮模式
// 注意事项：
//-------------------------------------------------
```

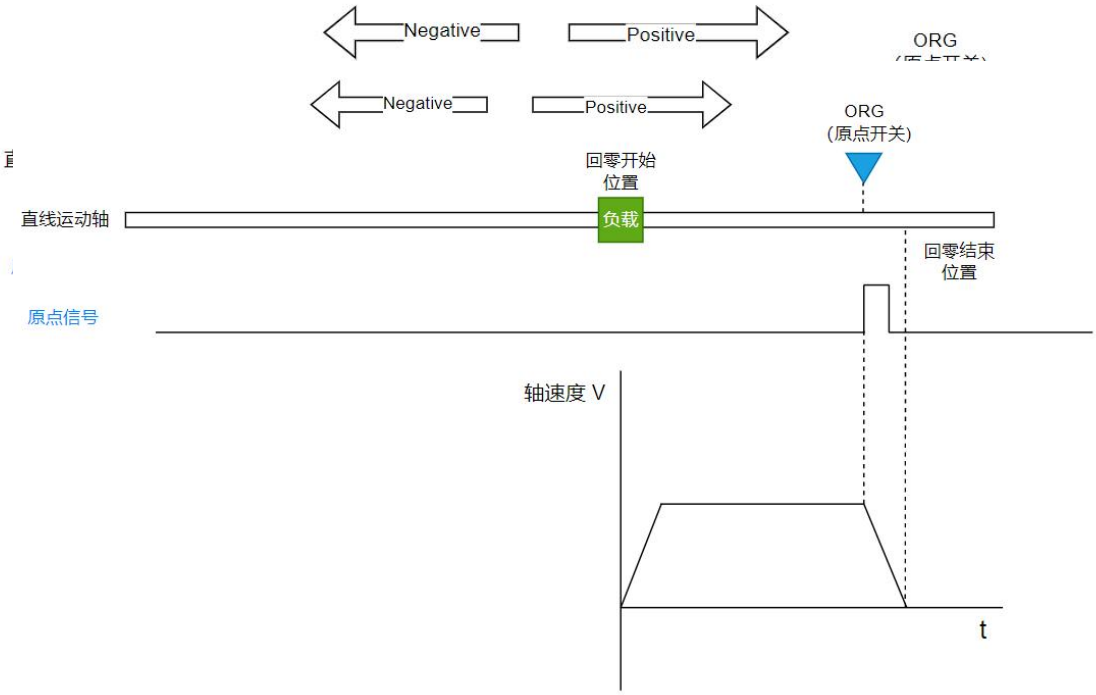
## 4.3. 轴配置

### 4.3.1应用说明

在控制卡正常初始化以后，调用本章节的函数对轴脉冲输出的类型、轴的回零模式、轴的最大速度及编码器反馈参数，是否配置手脉功能各项进行集中配置。

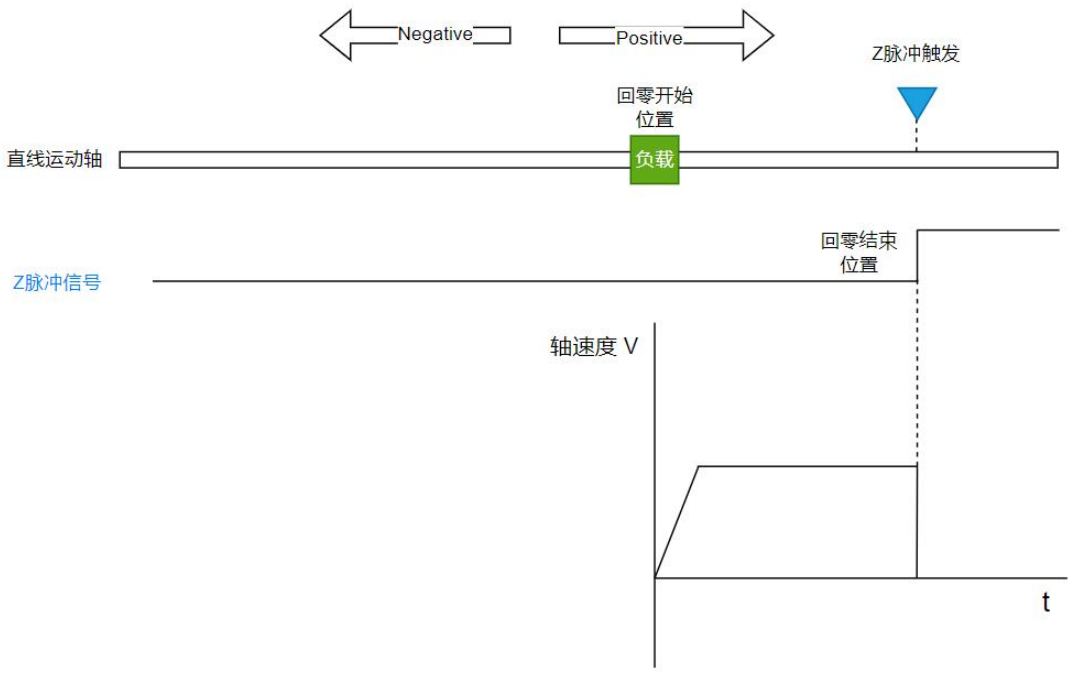
P1 控制卡提供了 4 种回零模式，分别为：

- 模式 0：指定轴检测到原点信号立即停止（下图以正向举例）

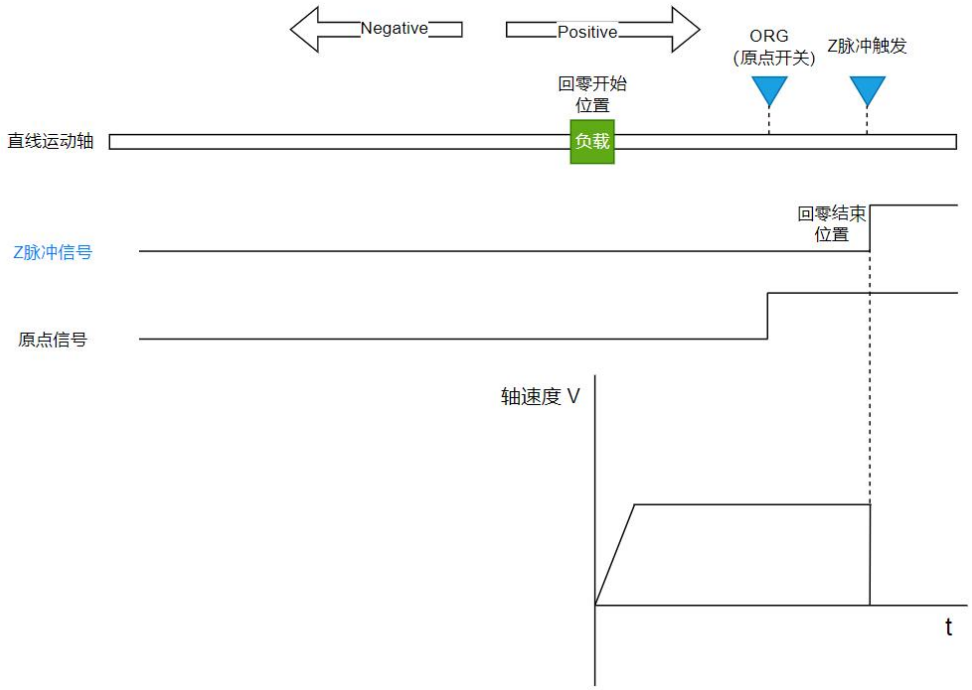



■ 模式 1：指定轴以梯形速度运动时，遇到轴 Org 信号有效时立即减速，到达梯形低速时停止（下图以正向举例）

■ 模式 2：检测到指定轴检测到 Z 脉冲信号立即停止（下图以正向举例）



■ 模式 3：指定轴检测到原点信号和 Z 脉冲信号同时有效时立即停止（下图以正向举例），此种回零模式一定要让原点触发的信号宽度足够，确保能触发一次 Z 脉冲信号。




**set_maxspeed 说明:**

用户设置一个轴的最大输出脉冲频率，该频率主要用于确定运动控制器脉冲输出倍率。

P1 系列运动控制器的输出脉冲频率由两个变量控制：速度寄存器值和倍率，两者的乘积即输出的脉冲频率。由于运动控制器内部倍率寄存器长度是有限的，为 13bit 位宽（最大值为 8191，最小值为 1，最小分辨单位为 1），所以速度寄存器能表示的速度值范围就是 1~8191。如果要达到大于 8191 的输出脉冲频率，就必须设置倍率，该倍率由最大速度除以 8191 得到。

倍率设置会影响速度分辨率和最大速度。比如当设置最大速度为 8MHz 的时候，倍率为 8M/8191 = 976，此时内部寄存器单位 1（最小值）就表示 976Hz，不能跑到低于 976Hz 以下的频率值，即速度的分辨率降低了（分辨率指寄存器单位 1 表示的实际速度）。为了解决这个问题，调用 set_maxspeed 设置当前运动能达到的最大输出脉冲频率，如 set_maxspeed（1，100）。设置后倍率将被重新设置，为（100/8191）=0.012，这样就能提高速度分辨率。虽然速度分辨率提高了，但最高速度有限制了，最高速不能高于 100Hz。

注意：P1 系列的最高分辨率（即最小倍率）可以达到 0.01，此时控制器的最大输出脉冲频率只能达到 82Hz。

set_maxspeed 的另一个用法即是需要在运动中途修改运行速度时，最大能调整的速度为最后一次调用该函数设定的最大速度。因为该函数会改变倍率，如果在运动过程中倍率改变了会导致速度突变造成冲击，所以为了保证调速稳定，运动过程中倍率应该稳定。该函数设置的最大速度即为整个运动，包含调速能达到的最高速度。

在缺省情况下，init_board 函数将所有轴设置其最大速度，即 100000（0.1MHz）。使用时可按照实际

输出速度进行设置以获得比较好的速度分辨率。最大输出脉冲频率范围：81.91~ 8000000 Hz。

set_maxspeed 指令适用于点位运动指令、回零运动指令、连续运动指令、插补运动指令。

## 4.3.2调用示例

**参数设置**

```
set_outmode(1, 1, 0);//设置 1 号轴为脉冲＋方向模式
set_outmode(2, 0, 0);//设置 2 号轴为 CW/CCW 模式
set_home_mode(1, 0);//设置 1 号轴回零方式为检测到原点信号立即停止
set_home_mode(2, 1);//设置 2 号轴回零方式为检测到原点信号减速停止
set_encoder_mode(1, 0, 4, 0, 0);//设置 1 号轴为 A/B 90 度相位差，4 倍频,B 向超前为正
set_encoder_mode(2, 1, 0, 0, 0);//设置 2 号轴为增减脉冲模式，倍频无效,B 向超前为正
set_maxspeed(1, 200000);//设置1号轴的最大速度为200000
```

## 4.3.3指令详解

### 4.3.3.1 set_outmode

指令原型：int set_outmode(int Axis, int Mode, int Logic);

```
//-------------------------------------------------
// 功能描述：运动控制器提供了两种脉冲输出模式：“脉冲+方向”（Pul+）和“正负脉冲”，默认脉冲输出方式为“脉冲+方向”。调用set_outmode指令，可将脉冲输出模式设置为“正负脉冲”方式。
// 输入参数：
    Axis --- 轴号
    Mode --- 模式： 0 (CW/CCW) - 1 (Pul/Dir)
    Logic --- 暂不使用，设置为 0
// 输出参数：无
// 返回值：=0 - 成功， =-1 - 失败
//-------------------------------------------------
```

### 4.3.3.2 set_home_mode

指令原型：int set_home_mode(int Axis, int origin_Mode);

```
//-------------------------------------------------
// 功能描述：设置轴的回原点模式
// 输入参数：
    Axis --- 轴号
    Mode --- 回零模式
    0 ---检测到原点信号立即停止；
    1 ---以梯形速度运动时，检测到原点信号有效时立即减速，到达梯形低速时停止；
    2 ---检测到Z脉冲信号立即停止；
    3 ---检测到原点信号和Z脉冲信号同时有效时立即停止；
// 输出参数：无；
// 返回值：=0 - 成功， =-1 - 失败
// 注意事项：MPC1400和MPC1800，回零模式只限于0，1。MPC3400和MPC3800的回零模式支持0，1，2，3
//-------------------------------------------------
```

### 4.3.3.3 set_encoder_mode

指令原型：int set_encoder_mode(int Axis, int Mode, int Multip, int CountUnit,int CountDir);

```
//-------------------------------------------------
// 功能描述：设置编码器模式。默认值为，MPC3400、MPC3800可设置。
// 输入参数：
    Axis---- 轴号
```

```
    Mode----编码器反馈模式：0--A/B向 90度相位差。1--增减脉冲模式
    multip --- 倍频数：1, 2, 4（仅在A/B 90度相位差模式下生效）
    CountUnit---未使用
    CountDir----计数方向，0--B向超前为正，1--A向超前为正。
// 输出参数：无
// 返回值：0 - 成功；  =-1 - 失败
注意事项：
1）初始化后默认是A/B向 90度相位差，4倍频，计数方向为B向超前模式。
2）在设置为增减脉冲模式后，倍频参数无效。
//-------------------------------------------------
```

#### 4.3.3.4 set_maxspeed

指令原型：int set_maxspeed(int Axis, double Speed);

```
//-------------------------------------------------
// 功能描述：设置轴最大运行速度，控制运行时速度的精度
// 输入参数：
    Axis ---- 轴号
    Speed ---- 最大速度（正值），范围10~8M pps
//注意：该函数未调用时，最大速度为0.1M
// 输出参数：无
// 返回值：0 - 成功；  <0 - 失败
//-------------------------------------------------
```

## 4.4. IO 配置

### 4.4.1应用说明

在控制卡正常初始化以后，调用本章节的函数对专用输入口（轴：限位，原点，报警；控制卡：报警）进行使能、逻辑电平和滤波参数进行配置。对通用输入口的逻辑电平和滤波参数进行配置。

### 4.4.2调用示例

#### 设置专用 IO

```
    set_outmode(1, 1, 0); //设置一轴脉冲输出模式为脉冲+方向模式
    set_home_mode(1, 0); //设置一轴回原点模式为 Org 信号有效立即停止
    enable_el(1, 1); //使能一轴限位信号
    enable_org(1, 1); //使能一轴原点信号
    enable_alm(1, 0); //禁止一轴报警信号
    set_el_logic(1, 0); //设置一轴限位信号为低电平有效，此时如果一轴限位信号接口有
    //低电平输入，会触发轴的限位状态，一轴轴对应方向的运动会停止
    set_org_logic(1, 0); //设置一轴原点信号为低电平有效，此时如果原点信号接口有低
    //电平输入，会触发轴的原点状态
```

### 4.4.3指令详解

#### 4.4.3.1 enable_el

指令原型：int enable_el(int Axis, int Flag);

```
//-------------------------------------------------
// 功能描述：  运动控制器为每个控制轴提供了正负限位接口。使能轴限位信号，当某轴的限位开关触发时，运动控制器
```

```
将立即停止该方向运动，以保障系统安全。不使能时，可以将其当作普通输入口来使用。
// 输入参数：
    Axis --- 轴号
    Flag ---- 使能标志： 1 （使能） -- 0 （不使能）
// 输出参数：无
// 返回值：=0 - 成功， =-1 - 失败
//--------------------------------------------------
```

#### 4.4.3.2 enable_org

指令原型：int enable_org(int Axis, int Flag);

```
//--------------------------------------------------
// 功能描述： 运动控制器为每个控制轴提供了原点信号输入接口。使能轴原点信号，当某轴作回原点运动时，若原点开关触发，运动控制器将自动停止运动。原点信号只对回原点运动有效。不使能时，可以将其当作普通输入口来使用。
// 输入参数：
    Axis --- 轴号
    Flag ---- 使能标志： 1 （使能） -- 0 （不使能）
// 输出参数：无
// 返回值：=0 - 成功， =-1 - 失败
//--------------------------------------------------
```

#### 4.4.3.3 enable_alm

指令原型：int enable_alm(int Axis, int Flag);

```
//--------------------------------------------------
// 功能描述： 运动控制器每个控制轴提供了轴报警信号输入接口。使能轴报警信号，当运动控制器处于运动状态时，若报警开关触发，运动控制器对应轴将自动停止运动。不使能时，可以将其当作普通输入口来使用。
// 输入参数：
    Axis --- 轴号
    Flag ---- 使能标志： 1 （使能） -- 0 （不使能）
// 输出参数：无
// 返回值：=0 - 成功， =-1 - 失败
//--------------------------------------------------
```

#### 4.4.3.4 set_el_logic

指令原型：int set_el_logic(int Axis, int Flag);

```
//--------------------------------------------------
// 功能描述： 设置轴限位信号有效电平。默认为低电平有效。
// 输入参数：
    Axis --- 轴号
    Flag ---- 有效电平： 1 （高电平） -- 0 （低电平）
// 输出参数：无
// 返回值：=0 - 成功， =-1 - 失败
//--------------------------------------------------
```

#### 4.4.3.5 set_org_logic

指令原型：int set_org_logic(int Axis, int Flag);

```
//--------------------------------------------------
// 功能描述： 设置轴原点信号有效电平。默认为低电平有效。
// 输入参数：
    Axis --- 轴号
    Flag ---- 有效电平： 1 （高电平） -- 0 （低电平）
// 输出参数：无
// 返回值：=0 - 成功， =-1 - 失败
//--------------------------------------------------
```

#### 4.4.3.6 set_alm_logic

指令原型：int set_alm_logic(int Axis, int Flag);

```
//--------------------------------------------------
// 功能描述： 设置轴报警信号有效电平。默认为低电平有效。
// 输入参数：
    Axis --- 轴号
    Flag ---- 有效电平： 1 （高电平） -- 0 （低电平）
// 输出参数：无
// 返回值：=0 - 成功， =-1 - 失败
//--------------------------------------------------
```

#### 4.4.3.7 enable_card_alm

指令原型：int enable_card_alm(int Axis, int Flag);

```
//--------------------------------------------------
// 功能描述： 运动控制器提供了一个板卡报警信号输入接口。使能卡报警信号，当运动控制器处于运动状态时，若报警开关触发，运动控制器将自动停止所有运动。不使能时，可以将其当作普通输入口来使用。
// 输入参数：
    Axis --- 轴号
    Flag ---- 使能标志： 1 （使能） -- 0 （不使能）
// 输出参数：无
// 返回值：=0 - 成功， =-1 - 失败
//--------------------------------------------------
```

#### 4.4.3.8 set_card_alm_logic

指令原型：int  set_card_alm_logic(int Card, int Flag)

```
//--------------------------------------------------
// 功能：设置卡报警信号有效电平
//参数：
 Card ----- 卡号
 Flag ---- 有效电平： 1 （高电平） -- 0 （低电平）
//返回值： 0（正确）----- -1（错误）
//--------------------------------------------------
```

#### 4.4.3.9 set_input_filter

指令原型：int set_input_filter(int Card, int BitNo, int Time)

```
//--------------------------------------------------
// 功能描述： 设置通用输入口的滤波时间。只有当电平信号持续触发大于滤波时间，运动控制卡才认为接受到实际通用输入信号。默认值是1000us
// 输入参数：
    Card---- 卡号
    BitNo ----通用输入口位标记，范围 1 ~ 32
    Time ----读取的滤波时间，单位：us，范围0~65000// 输出参数：无
// 返回值：=0 - 成功， =-1 - 失败
//--------------------------------------------------
```

#### 4.4.3.10 set_special_input_filter

指令原型：int set_special_input_filter(int Card,int BitNo,int Time);

```
//-------------------------------------------------------
// 功能描述：设置专用输入口的滤波时间。只有当电平信号持续触发大于滤波时间，运动控制卡才认为接受到实际专用输入信号。默认值是1000us。
// 输入参数：
     Card---- 卡号
     BitNo ----专用输入口位标记，范围 1 ~ 64
     Time ----设置的滤波时间，单位：us，范围0~65000
//输出参数：无
// 返回值：=0 - 成功，  =-1 - 失败
//-------------------------------------------------------
```

#### 4.4.3.11 set_geInputLogic

指令原型：int set_geInputLogic(int Card,int BitNo,int Time);

```
//-------------------------------------------------------
// 功能：设置通用输入口的有效逻辑电平
// 参数：
    Card ---- 卡号
    BitNo ---- 通用输入口位标记，范围 1 ~ 32
    Flag ---- 0：低电平有效，1：高电平有效
//返回值：  0（正确）----- -1（错误）
//-------------------------------------------------------
```

#### 4.4.3.12 load_configfile

指令原型：int load_configfile(int cardno,const char* FileName);

```
//-------------------------------------------------------
// 功能：设置通用输入口的有效逻辑电平
// 参数：
    Cardno ---- 卡号
    FileName ----配置文件路径
               参数文件名+后缀：相对路径
               完整描述参数文件的路径+文件名+后缀：绝对路径
    返回值：0(成功) ---- -1(失败)
//-------------------------------------------------------
    注意事项：1)使用相对路径时，参数文件与程序必须在同一目录下
              2)参数文使用P1ConfigTools.exe生成
              3)注意参数文件与卡的对应关系
              4)下载系统配置信息到运动控制器，在init_board后调用
              5)函数返回失败可能是卡号与卡类型不匹配或者轴正在运动中
```

# 5. 状态查询

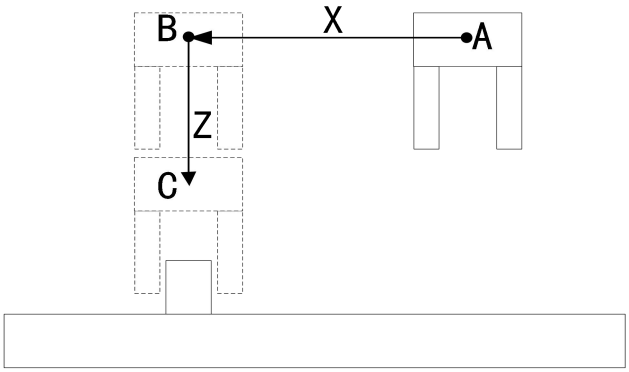
## 5.1. 相关指令列表

| 查询轴的状态 | |
|---|---|
| check_status | 查询单个轴的工作状态信息 |
| get_cur_dir | 查询单个轴的运动方向 |
| check_done | 查询单个轴是否处于运动中 |
| check_limit | 查询轴是否处于限位状态（限位输入被配置为有效时） |
| check_home | 查询轴是否处于原点状态（原点输入被配置为有效时） |
| check_alarm | 查询轴是否处于报警状态（报警输入被配置为有效时） |
| **查询专用输入的状态** | |
| check_card_alm | 查询卡报警专用输入口的状态 |
| check_sfr | 查询单卡上各专用输入口的状态（<32 点时） |
| check_sfr_ex | 查询单卡上各专用输入口的状态（>=32 点时） |
| check_sfr_bit | 查询单卡上单个专用输入口的状态 |
| checkin_axis_INP | 查询轴伺服定位完成专用输入口的状态 |
| checkin_axis_SRDY | 查询轴伺服准备好专用输入口的状态 |
| read_axis_RST | 查询轴当前的报警清除的输出状态 |
| read_axis_SEVERON | 查询轴当前的使能输出状态 |
| **查询通用 I/O 的状态** | |
| checkin_byte | 查询卡上多个通用输入口的状态 |
| checkin_bit | 查询卡上单个通用输入口的状态 |
| checkin_ai | 查询卡上的模拟量输入 |
| get_gelnputLogicStatus | 查询卡上单个通用输入点的逻辑状态 |
| check_outport_status | 查询单卡上多个通用输出口的状态 |

## 5.2. 查询轴的状态

### 5.2.1应用说明

在设备流程运动过程中，通常一条运动指令执行完毕后才能运行另一条运动指令。例如，当设备通过 XZ 轴的运动移动夹爪去抓取固定位置元件时，点位运动指令移动 X 轴控制夹爪从 A 移动到 B 点，通过调用 check_done 查询指令，判断 X 轴是否运动停止，当 X 轴停止后才能通过点位运动指令移动 Z 控制夹爪从 B 点下降到 C 点，调用 check_done 查询指令，判断 Z 轴是否运动停止，当 Z 轴停止后在操作 IO 控制指令抓取元件，完成整个流程。




抓取运动使用的相关指令：

| 指令函数 | 指令说明 |
| --- | --- |
| fast_pmove | 控制 X 轴从 A 移动到 B |
| check_done | 判断 X 轴运动停止 |
| fast_pmove | 控制 Z 轴从 B 移动到 C |
| check_done | 判断 Z 轴运动停止 |
| outport_bit | 控制夹爪气缸抓取 |

## 5.2.2调用示例

**轴抓取运动**

```
set_s_curve(1, 0); //设置 1 轴梯形速度模式
set_s_curve(2, 0); //设置 2 轴梯形速度模式
set_maxspeed(1, 5000); //设置 1 轴最大速度为 5000
set_maxspeed(2, 5000); //设置 2 轴最大速度为 5000
set_profile(1, 100, 5000, 2000, 2000); //设置 1 轴初速、高速、加速度和减速度
set_profile(1, 80, 3000, 2000, 2000); //设置 2 轴初速、高速、加速度和减速度
fast_pmove(1, -2000);//1 轴以梯形快速运动方式从 A 运动到 B
int rtn = check_done(1);//判断 1 轴是否停止
if (rtn == 0) //运动完成，可以往下运动
fast_pmove(2, -1000);//2 轴以梯形快速运动方式从 B 运动到 C
rtn = check_done(2);//判断 2 轴是否停止
if (rtn == 0) //运动完成，可以往下运动
outport_bit(1, 1, 0);//打开输出口1控制夹爪抓取
```

## 5.2.3指令详解

**5.2.3.1 check_status**

指令原型：int  check_status(int Axis);

```
//-----------------------------------------------------
// 功能描述：获取指定轴工作状态信息。
// 输入参数:
// Axis - 轴号，取值：1~32
// 输出参数：无
// 返回值：正值（指定轴的状态值）-成功，-1-失败
// 功能描述：获取指定轴工作状态信息。
// 注意：查看轴专用输入信号（ORG、EL、ALM 等）的有效状态时，必须确认所查看的信号已经使能，否则查询到的状态不对。
//-----------------------------------------------------
```

| Check_status 返回值说明 | | |
|---|---|---|
| Bit 位 | 状态含义 | 状态说明 |
| D31-D28 | 保留 | |
| D27 | 锁存信号触发标志 | 0：未触发锁存 1:已触发锁存 |
| D26 | 手轮功能使能 | 0：无效　　1：有效 |
| D24-D25 | 保留 | |
| D23 | 原点信号 | 0：无效　　1：有效 |
| D22 | 正限位有效 | 0：无效　　1：有效 |
| D21 | 负限位有效 | 0：无效　　1：有效 |
| D20 | Z 脉冲有效 | 0：无效　　1：有效 |
| D19 | 总报警有效 | 0：无效　　1：有效 |
| D18 | 轴报警有效 | 0：无效　　1：有效 |
| D17 | 总停止 | 0：无效　　1：有效 |
| D16 | 限位停止 | 0：无效　　1：有效 |
| D15 | 报警停止 | 0：无效　　1：有效 |
| D14 | 回零方式 3 停止 | 0：无效　　1：有效 |
| D13 | 回零方式 2 停止 | 0：无效　　1：有效 |
| D12 | 回零方式 1 停止 | 0：无效　　1：有效 |
| D11 | 回零方式 0 停止 | 0：无效　　1：有效 |
| D1-D10 | 保留 | |
| D0 | Hold | 0：运动状态　　1：停止状态 |

#### 5.2.3.2 get_cur_dir

指令原型：int　get_cur_dir(int Axis);

```
//-------------------------------------------------
// 功能描述：获取指定轴当前运动方向
// 输入参数：
//      Axis - 轴号，取值：1~32
// 输出参数：
// 返回值：0-停止，1-正向，-1-负向，-2-失败
//-------------------------------------------------
```

#### 5.2.3.3 check_done

指令原型：int　check_done(int Axis);

```
//-------------------------------------------------
// 功能描述：获取轴当前运动状态。
// 输入参数：
//      Axis - 轴号，取值：1~32
// 输出参数：无
// 返回值：0-停止，1-运动，-1-失败
// 注意：在调用下一条运动指令时，必须先调用此 API 确认当前轴已经处于停止状态，否则不执行后续运动指令
//-------------------------------------------------
```

#### 5.2.3.4 check_limit

指令原型：int　check_limit(int Axis);

```
//-------------------------------------------------
// 功能描述：获取指定轴当前限位信号有效状态。
// 输入参数：
//      Axis - 轴号，取值：1~32
// 输出参数：无
// 返回值：0-无效，1-正限位有效，-1-负限位有效，-3-失败
// 注意：该条指令只能在使能轴限位信号后使用才能返回正确状态。
//-------------------------------------------------
```

**5.2.3.5 check_home**

```
指令原型：int  check_home(int Axis);
//-------------------------------------------------
// 功能描述：获取指定轴当前原点信号有效状态。
// 输入参数：
//      Axis - 轴号，取值：1~32
// 输出参数：无
// 返回值：0-无效，1-有效，-3-失败
// 注意事项：该条指令只能在使能轴原点信号后使用才能返回正确状态。
//-------------------------------------------------
```

**5.2.3.6 check_alarm**

```
指令原型：int  check_alarm(int Axis);
//-------------------------------------------------
// 功能描述：获取指定轴当前报警信号有效状态。
// 输入参数:
//      Axis - 轴号，取值：1~32
// 输出参数：无
// 返回值：0-无效，1-有效，-3-失败
// 注意事项：该条指令只能在使能轴报警信号后使用才能返回正确状态。
//-------------------------------------------------
```

# 5.3. 查询专用输入口状态

## 5.3.1应用说明

P1 系列控制卡根据控制对象不同分为步进驱动控制卡和伺服驱动控制卡。其中步进驱动控制卡专用输入包含**报警信号**、**原点信号**、**限位信号**；伺服驱动控制卡专用输入包含**报警信号**、**原点信号**、**限位信号**、**报警信号**、**伺服到位信号**、**伺服就绪信号**；以及**控制卡报警信号**。

P1 系列控制卡提供了获取控制卡报警输入状态、伺服轴定位完成状态、伺服轴就绪状态、以及控制卡全部专用输入状态的指令接口。

## 5.3.2调用示例

**获取专用输入状态**

```
    if (check_card_alarm(1) != 0)//检测 1 号卡报警状态
    {
        //处理 1 号卡报警
    }
    if (checkin_axis_SRDY(1) != 1)//检测 1 轴是否就绪
    {
        //未就绪，等待伺服就绪信号
    }
    if (checkin_axis_INP(1) != 1)//检测 1 轴是否定位完成
    {
        //定位未完成，等待伺服定位完成
    }
    int status = check_sfr(1); //获取 1 卡全部专用输入状态（MPC1400/3400 有效）
```

```
unsigned int nstatus[2] = { 0,0 }; //8 轴卡需要 2 个 int 变量描述 8 轴的专用输入状态
int numb = 0;
check_sfr_ex(2, &numb, nstatus); //获取指定 2 卡全部专用输入状态（MPC1800/3800 有效）
```

## 5.3.3指令详解

### 5.3.3.1 check_card_alarm

指令原型：int  check_card_alarm(int Card);

```
//-----------------------------------------------------
// 功能描述：查询卡报警专用输入口的状态
// 输入参数：
//      Card - 卡号，取值：1~4
// 输出参数：无
// 返回值：0-无效，1-有效，-3-失败（MPC1616不适用该函数，也会返回-3）
//-----------------------------------------------------
```

### 5.3.3.2 check_sfr

指令原型：int  check_sfr(int Card);

```
//-----------------------------------------------------
// 功能描述：查询4轴卡专用输入口的状态
// 输入参数：
//      Card - 卡号，取值：1~4
// 输出参数：无
// 返回值：非负（专用输入口状态）-成功，  -1-失败
// 注意事项：该指令用于MPC1400/MPC3400控制卡
//-----------------------------------------------------
```

Check_sfr 返回值说明

| Bit0 | bit1 | bit2 | bit3 | bit4 | bit5 | bit6 | bit7 | bit8 | bit9 |
|---|---|---|---|---|---|---|---|---|---|
| 1轴原点 | 1轴正限位 | 1轴负限位 | 1轴报警 | 1轴Z脉冲 | 1轴伺服定位完成 | 1轴伺服就绪 | 2轴原点 | 2轴正限位 | 2轴负限位 |
| bit10 | bit11 | bit12 | bit13 | bit14 | bit15 | bit16 | bit17 | bit18 | bit19 |
| 2轴报警 | 2轴Z脉冲 | 2轴伺服定位完成 | 2轴伺服就绪 | 3轴原点 | 3轴正限位 | 3轴负限位 | 3轴报警 | 3轴Z脉冲 | 3轴伺服定位完成 |
| bit20 | bit21 | bit22 | bit23 | bit24 | bit25 | bit26 | bit27 | bit28 | bit29 |
| 3轴伺服就绪 | 4轴原点 | 4轴正限位 | 4轴负限位 | 4轴报警 | 4轴Z脉冲 | 4轴伺服定位完成 | 4轴伺服就绪 | 卡1报警 | 卡2报警 |
| bit30 | bit31 | | | | | | | | |
| 保留 | 保留 | | | | | | | | |

### 5.3.3.3 check_sfr_ex

指令原型：int  check_sfr_ex(int Card, int* Num, int* Status)

```
//-----------------------------------------------------
// 功能描述：查询8轴卡专用输入口的状态
// 输入参数：
//      Card   - 卡号，取值：1~4
// 输出参数：
//      Num    - 返回专用输入口状态的个数；
//      Status - 返回专用输入口状态的数组，个数由Num决定，目前返回最大个数为2，使用时
//               需定义一个长度为2的数组；
// 返回值：非负-成功，  -1-失败
// 注意事项：该指令用于MPC1800/MPC3800控制卡
//-----------------------------------------------------
```

| Check_sfr_ex 输出参数 status 说明 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Status[0] | | | | | | | | | |
| bit0 | bit1 | bit2 | bit3 | bit4 | bit5 | bit6 | bit7 | bit8 | bit9 |
| 1轴原点 | 1轴正限位 | 1轴负限位 | 1轴报警 | 1轴Z脉冲 | 1轴伺服定位完成 | 1轴伺服就绪 | 2轴原点 | 2轴正限位 | 2轴负限位 |
| bit10 | bit11 | bit12 | bit13 | bit14 | bit15 | bit16 | bit17 | bit18 | bit19 |
| 2轴报警 | 2轴Z脉冲 | 2轴伺服定位完成 | 2轴伺服就绪 | 3轴原点 | 3轴正限位 | 3轴负限位 | 3轴报警 | 3轴Z脉冲 | 3轴伺服定位完成 |
| bit 20 | bit21 | bit22 | bit23 | bit24 | bit25 | bit26 | bit27 | bit28 | bit29 |
| 3轴伺服就绪 | 4轴原点 | 4轴正限位 | 4轴负限位 | 4轴报警 | 4轴Z脉冲 | 4轴伺服定位完成 | 4轴伺服就绪 | 卡1报警 | 卡2报警 |
| bit30 | bit31 | | | | | | | | |
| 保留 | 保留 | | | | | | | | |
| Status[1] | | | | | | | | | |
| bit0 | bit1 | bit2 | bit3 | bit4 | bit5 | bit6 | bit7 | bit8 | bit9 |
| 5轴原点 | 5轴正限位 | 5轴负限位 | 5轴报警 | 5轴Z脉冲 | 5轴伺服定位完成 | 5轴伺服就绪 | 6轴原点 | 6轴正限位 | 6轴负限位 |
| bit10 | bit11 | bit12 | bit13 | bit14 | bit15 | bit16 | bit17 | bit18 | bit19 |
| 6轴报警 | 6轴Z脉冲 | 6轴伺服定位完成 | 6轴伺服就绪 | 7轴原点 | 7轴正限位 | 7轴负限位 | 7轴报警 | 7轴Z脉冲 | 7轴伺服定位完成 |
| bit20 | bit21 | bit22 | bit23 | bit24 | bit25 | bit26 | bit27 | bit28 | bit29 |
| 7轴伺服就绪 | 8轴原点 | 8轴正限位 | 8轴负限位 | 8轴报警 | 8轴Z脉冲 | 8轴伺服定位完成 | 8轴伺服就绪 | 保留 | 保留 |
| bit30 | bit31 | | | | | | | | |
| 保留 | 保留 | | | | | | | | |

#### 5.3.3.4 check_sfr_bit

指令原型：int  check_sfr_bit(int Card, int BitNo)

```
//-------------------------------------------------
//功能：读取指定卡上某位专用输入口电平状态（ORG、EL、轴ALM、Z信号和板卡 ALARM信号）
//参数：
Card ---- 卡号
BitNo ---- 专用输入口位标记，范围 1 ~ 64(BitNo为1时对应1轴原点，BitNo为33时对应5轴原点)
//返回值： 1（高电平）---- 0（低电平）----- -1（错误）
//-------------------------------------------------
```

#### 5.3.3.5 checkin_axis_INP

指令原型：int  checkin_axis_INP(int Axis);

```
//-------------------------------------------------
// 功能描述：查询轴伺服定位完成专用输入口的物理电平状态
// 输入参数：
//      Axis - 轴号，取值：1~32
// 输出参数：无
// 返回值：0-低电平，1-高电平
// 注意事项：该指令只对MPC3400/MPC3800控制卡有效
//-------------------------------------------------
```

#### 5.3.3.6 checkin_axis_SRDY

指令原型：int  checkin_axis_SRDY(int Axis);

```
//-------------------------------------------------
// 功能描述：查询轴伺服就绪专用输入口的物理电平状态
// 输入参数：
//      Axis - 轴号，取值：1~32
```

```
// 输出参数：无
// 返回值：0-低电平，1-高电平
// 注意事项：该指令只对MPC3400/MPC3800控制卡有效
//---------------------------------------------------
```

#### 5.3.3.7 read_axis_RST

指令原型：int  read_axis_RST(int Axis, int* Status);

```
//---------------------------------------------------
// 功能描述：读取轴当前的报警清除输出状态
// 输入参数：
//      Axis - 轴号，取值：1~32
// 输出参数：
       Status ---- 报警清除输出状态，1(无输出) ---- 0(输出低电平)
// 返回值：0-低电平，1-高电平
// 注意事项：该指令只对MPC3400/MPC3800控制卡有效
//---------------------------------------------------
```

#### 5.3.3.8 read_axis_SEVERON

指令原型：int  read_axis_SEVERON(int Axis, int* Status);

```
//---------------------------------------------------
// 功能描述：读取轴当前的使能输出状态
// 输入参数：
//      Axis - 轴号，取值：1~32
// 输出参数：
       Status ---- 使能输出状态，1(无输出) ---- 0(输出低电平)
// 返回值：0-低电平，1-高电平
// 注意事项：该指令只对MPC3400/MPC3800控制卡有效
//---------------------------------------------------
```

## 5.4. 查询通用 I/O 的状态

### 5.4.1应用说明

P1 系列控制卡根据控制卡型号不同，对应的通用 I/O 数量也各不相同。

| 名称 | 说明 |
|---|---|
| MPC1400 | 提供带光电隔离 12 路通用输入口和 12 路通用输出口、2 路模拟量输入 |
| MPC3400 | 提供带光电隔离 16 路通用输入口和 16 路通用输出口 |
| MPC1800 | 提供带光电隔离 24 路通用输入口和 24 路通用输出口、4 路模拟量输入 |
| MPC3800 | 提供带光电隔离 32 路通用输入口和 32 路通用输出口 |
| MPC1616 | 提供带光电隔离 16 路通用输入口和 16 路通用输出口 |

### 5.4.2调用示例

获取输入状态

```
int status = checkin_bit(1, 1);//获取 1 号卡通用输入口 1 状态
status = checkin_byte(1);//获取 1 号卡全部通用输入状态
int value = 0;
```

```
    checkin_ad(1, 1, &value);//读取模拟量输入口 1 的模拟量值
    status = get_geInputLogicStatus(1, 1); //获取卡 bit1 通用输入状态
    status = check_outport_status(1); //获取指定 4 轴卡全部专用输入状态
```

## 5.4.3指令详解

### 5.4.3.1 checkin_byte

指令原型：int　checkin_byte(int Card);

```
//-------------------------------------------------
// 功能描述：查询单卡上多个通用输入口的实际状态
// 输入参数：
//      Card - 卡号，取值：1~4
// 输出参数：无
// 返回值：非负（通用输入口状态）-成功，-1-失败(当卡为MPC3800时也返回-1)
// 注意事项：控制卡全部通用输入状态按bit定义，每bit位对应一个输入口（当卡为MPC3800时使用checkin_byte_ex）
//-------------------------------------------------
```

checkin_byte 返回值说明

| 型号 | 通用输入口 bit | bit 状态 |
|---|---|---|
| MPC1400 | bit1-bit12 | 0-低电平，1-高电平 |
| MPC3400 | bit1-bit16 | 0-低电平，1-高电平 |
| MPC1800 | bit1-bit24 | 0-低电平，1-高电平 |
| MPC1616 | bit1-bit16 | 0-低电平，1-高电平 |

### 5.4.3.2 checkin_bit

指令原型：int　checkin_bit(int Card, int BitNo);

```
//-------------------------------------------------
// 功能描述：查询单个通用输入口的实际状态
// 输入参数：
//      Card   - 卡号，取值：1~4
//      BitNo -  通用输入口位标记，取值：1~32
// 输出参数：无
// 返回值：0-低电平，1-高电平，-1-失败
//-------------------------------------------------
```

### 5.4.3.3 checkin_ai

指令原型：int　checkin_ai(int Card, int BitNo, int *Value);

```
//-------------------------------------------------
// 功能描述：查询用户通用的模拟量输入
// 输入参数：
//      Card    - 卡号，取值：1~4
//      BitNo  -  模拟量输入口位标记，取值：1~4
// 输出参数：
//      Value  -  读取到的模拟量输入口值
// 注意事项：读取值0-4095对应-10.24V到+10.24V
// 返回值：0-成功，-1-失败
//-------------------------------------------------
```

### 5.4.3.4 get_geInputLogicStatus

指令原型：int　get_geInputLogicStatus(int Card, int BitNo);

```
//-------------------------------------------------
// 功能描述：获取单卡上通用输入逻辑状态
// 输入参数：
//      Card    - 卡号，取值：1~4
```

```
//      BitNo  - 通用输入口位标记，取值：1~32
// 输出参数：无
// 返回值：0-无效，1-有效，-1-失败
//------------------------------------------------
```

#### 5.4.3.5 check_outport_status

指令原型：int  check_outport_status(int Card);

```
//------------------------------------------------
// 功能描述：查询单卡上多个通用输出口状态
// 输入参数：
//      Card - 卡号，取值：1~4
// 输出参数：无
// 返回值：非负（通用输出口状态）-成功，-1-失败(当卡为 MPC3800 时也返回-1)
// 注意事项：控制卡全部通用输出状态按bit定义，每bit位对应一个输出口（当卡为MPC3800时使用
check_outport_status_ex）
//------------------------------------------------
```

check_outport_ststus 返回值说明

| 型号 | 通用输出口 bit | bit 状态 |
|---|---|---|
| MPC1400 | bit1-bit12 | 0-低电平，1-无输出 |
| MPC3400 | bit1-bit16 | 0-低电平，1-无输出 |
| MPC1800 | bit1-bit24 | 0-低电平，1-无输出 |
| MPC3800 | bit1-bit32 | 0-低电平，1-无输出 |
| MPC1616 | bit1-bit16 | 0-低电平，1-无输出 |

# 6. I/O 口控制

## 6.1. 相关指令列表

| IO 控制 | |
|---|---|
| outport_bit_RST | 对伺服驱动器进行报警清除 |
| outport_bit_SEVERON | 对伺服驱动器进行轴使能或禁止 |
| outport_bit_CL | 对伺服驱动器的计数器进行清零 |
| outport_byte | 打开或关闭多个通用输出口 |
| outport_bit | 打开或关闭单个通用输出口 |

## 6.2. 应用说明

P1 系列控制卡可以调用输出控制指令实现对控制器的输出口的控制。其中 MP3400 和 MPC3800 伺服驱动控制卡提供了伺服报警清除、伺服轴使能 2 路专用伺服输出控制的指令接口。

应用场景：

红绿黄三色灯用于显示设备报警、加工、待机 3 种状态，分别对应 3 个通用输出口，在应用软件编程中，调用指令 outport_bit 控制相应的输出口用于设备状态显示。

当有伺服轴组成的设备时，在应用软件编程中，会调用伺服专用输出指令对伺服驱动进行控制，通过调用指令 outport_bit_SEVERON 输出使能信号控制伺服驱动器使能；调用指令 outport_bit_RST 输出信号控制驱动器清除之前的报警状态（只能清除驱动不需要重启的报警状态）。

## 6.3. 调用示例

三色灯及伺服使能控制

```
#define BITNO_RED    1
#define BITNO_GREEN  2
#define BITNO_YELLO  3
    if (check_alarm(1) != 0)//检测 1 轴报警状态
    {
        outport_bit(1, BITNO_RED, 1);//三色灯输出红灯报警
    }
    else
    {
        outport_bit_RST(1, 1);//清除伺服报警
        outport_bit(1, BITNO_YELLO, 1);//三色灯输出黄灯
        //等待一段时间
        outport_bit_SEVERON(1, 1);//输出 1 轴伺服使能
    }
```

# 6.4. 指令详解

#### 6.4.1.1 outport_bit_RST

指令原型：int  outport_bit_RST(int Axis, int status);

```
//-------------------------------------------------
// 功能描述：对伺服驱动器进行报警清除
// 输入参数：
//      Axis   - 轴号，取值：1~32
//      state - 输出口状态，0-低电平，1-高电平
// 输出参数：无
// 返回值：0-成功，-1-失败
// 注意事项：该指令只对MPC3400/MPC3800控制卡有效
//-------------------------------------------------
```

#### 6.4.1.2 outport_bit_SEVERON

指令原型：int  outport_bit_SEVERON(int Axis, int status);

```
//-------------------------------------------------
// 功能描述：对伺服驱动器进行轴使能或禁止
// 输入参数：
//      Axis  - 轴号，取值：1~32
//      state - 输出口状态，0-低电平，1-高电平
// 输出参数：无
// 返回值：0-成功，-1-失败
// 注意事项：该指令只对MPC3400/MPC3800控制卡有效
//-------------------------------------------------
//-------------------------------------------------
```

#### 6.4.1.3 outport_bit_CL

指令原型：int  outport_bit_CL(int Axis, int status);

```
//-------------------------------------------------
// 功能描述：对伺服驱动器的计数器进行清零
// 输入参数：
//      Axis  - 轴号，取值：1~32
//      state - 输出口状态，0-低电平，1-高电平
// 输出参数：无
// 返回值：0-成功，-1-失败
//-------------------------------------------------
```

#### 6.4.1.4 outport_byte

指令原型：int  outport_byte(int cardno, int data);

```
//-------------------------------------------------
// 功能描述：打开或关闭多个通用输出口
// 输入参数：
//      cardno - 卡号，取值：1~5
//      data   - 输出口状态，0-输出低电平，1-无输出
// 输出参数：无
// 返回值：0-成功，-1-失败
//-------------------------------------------------
```

#### 6.4.1.5 outport_bit

指令原型：int　outport_bit(int Card, int BitNo, int Status);

```
//-------------------------------------------------
// 功能描述：打开或关闭单个通用输出口
// 输入参数：
//      Card - 卡号，取值：1~5
//      BitNo  - 通用输出口位标记，取值：1~32
//      Status - 输出口状态，0-输出低电平，1-无输出
// 输出参数：无
// 返回值：0-成功，-1-失败
//-------------------------------------------------
```

# 7. 轴运动控制

## 7.1. 相关指令列表

| JOG 运动的编程 | |
|---|---|
| con_vmove | 以常速模式启动单轴 JOG 运动 |
| con_vmove2 | 以常速模式启动两轴 JOG 运动 |
| con_vmove3 | 以常速模式启动三轴 JOG 运动 |
| con_vmove4 | 以常速模式启动四轴 JOG 运动 |
| fast_vmove | 以变速模式启动单轴 JOG 运动 |
| fast_vmove2 | 以变速模式启动两轴 JOG 运动 |
| fast_vmove3 | 以变速模式启动三轴 JOG 运动 |
| fast_vmove4 | 以变速模式启动四轴 JOG 运动 |
| **定位运动的编程** | |
| set_conspeed | 设置单轴常速模式下的速度 |
| set_s_curve | 设置轴 S 型速度模式 |
| set_profile | 设置轴 T 型变速模式下的速度 |
| set_s_section | 设置轴 S 型变速模式下的速度 |
| con_pmove | 以常速模式启动单轴定位运动 |
| con_pmove_to | 以常速模式启动单轴定位运动（绝对式） |
| con_pmove2 | 以常速模式启动两轴定位运动 |
| con_pmove3 | 以常速模式启动三轴定位运动 |
| con_pmove4 | 以常速模式启动四轴定位运动 |
| fast_pmove | 以变速模式启动单轴定位运动 |
| fast_pmove_to | 以变速模式启动单轴定位运动（绝对式） |
| fast_pmove2 | 以变速模式启动两轴定位运动 |
| fast_pmove3 | 以变速模式启动三轴定位运动 |
| fast_pmove4 | 以变速模式启动四轴定位运动 |
| set_abs_pos | 设置单轴绝对位置 |
| reset_pos | 对单个轴位置清零 |
| get_abs_pos | 获取单个轴当前的位置 |
| get_rate | 获取单个轴当前的速度 |
| **直线插补运动的编程** | |
| set_vector_conspeed | 设置常速模式下直线运动的矢量速度 |
| set_vector_profile | 设置 T 型变速模式下直线运动的矢量速度 |
| con_line2 | 以常速模式启动两轴直线插补运动 |
| con_line3 | 以常速模式启动三轴直线插补运动 |
| con_line4 | 以常速模式启动四轴直线插补运动 |
| con_line2_to | 以常速模式启动两轴直线插补运动（绝对式） |

| | |
|---|---|
| con_line3_to | 以常速模式启动三轴直线插补运动（绝对式） |
| con_line4_to | 以常速模式启动四轴直线插补运动（绝对式） |
| fast_line2 | 以变速模式启动两轴直线插补运动 |
| fast_line3 | 以变速模式启动三轴直线插补运动 |
| fast_line4 | 以变速模式启动四轴直线插补运动 |
| fast_line2_to | 以变速模式启动两轴直线插补运动（绝对式） |
| fast_line3_to | 以变速模式启动三轴直线插补运动（绝对式） |
| fast_line4_to | 以变速模式启动四轴直线插补运动（绝对式） |
| **回零运动的编程** | |
| con_hmove | 以常速模式启动单轴回零运动 |
| con_hmove2 | 以常速模式启动两轴回零运动 |
| con_hmove3 | 以常速模式启动三轴回零运动 |
| con_hmove4 | 以常速模式启动四轴回零运动 |
| fast_hmove | 以变速模式启动单轴回零运动 |
| fast_hmove2 | 以变速模式启动两轴回零运动 |
| fast_hmove3 | 以变速模式启动三轴回零运动 |
| fast_hmove4 | 以变速模式启动四轴回零运动 |
| **运动轴的制动** | |
| sudden_stop | 对单个轴急停制动 |
| sudden_stop2 | 对两个轴同时急停制动 |
| sudden_stop3 | 对三轴轴同时急停制动 |
| sudden_stop4 | 对四个周同时急停制动 |
| decel_stop | 对单个轴缓停制动 |
| decel_stop2 | 对两个轴缓停制动 |
| decel_stop3 | 对三个轴缓停制动 |
| decel_stop4 | 对四个轴缓停制动 |
| move_pause | 运动轴暂停 |
| move_resume | 运动轴恢复（暂停后） |

# 7.2. JOG 运动的编程

## 7.2.1应用说明

JOG 运动是指各轴按各自设定的速度、加速度、运动方向运动，直到有相应方向的限位、报警信号，或者调用停止指令才停止的运动。

应用场景关联：JOG 运动操作简单，按住某个轴 JOG 按键，该轴一直运动，松开按键该轴停止运动。1.在设备安装完毕后，JOG 运动常用于确认轴的运动方向是否正确，点击正方向点动按钮后松开按钮，查看轴是否朝正方向运动。2.工艺试制过程中，JOG 运动常用于示教，通过 JOG 运动控制某个轴移动到目标位置，记录目标位置为加工点。

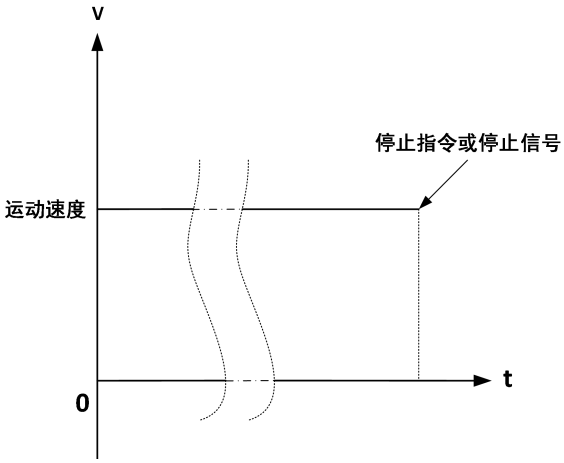



图 1

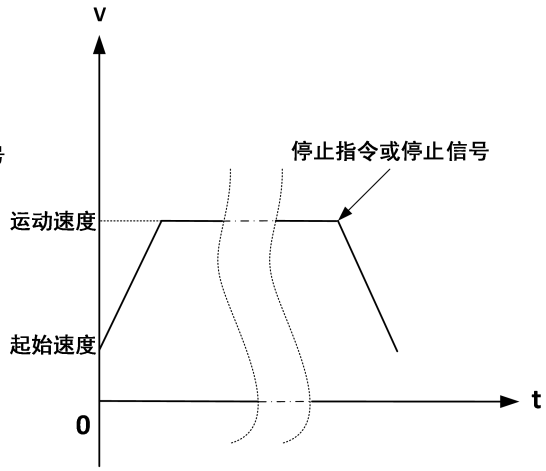



图 2

图 1 和图 2 分别为常速 JOG 运动和变速 JOG 运动的速度时间曲线。

## 7.2.2调用示例

Jog 运动

```
set_maxspeed(1, 10000); //设置 1 轴最大速度
set_profile(1, 100, 10000, 8000,8000);//设置 1 轴初速、高速、加速度
fast_vmove(1, 1); //启动 1 轴正向连续运动（带梯形加减速）
//一段时间后运动到 1 轴目标位置
sudden_stop(1);//停止 1 轴运动
set_conspeed(2, 5000);//设置 2 轴常速运动速度
con_vmove(2, -1);//启动 2 轴负向连续运动（不带梯形加减速）
//一段时间后运动到 2 轴目标位置
sudden_stop(2);//停止2轴运动
```

## 7.2.3指令详解

7.2.3.1 con_vmove

指令原型：int  con_vmove(int Axis, int Dir);

```
//-------------------------------------------------
// 输入参数：
// Axis - 轴号，1~32号轴；
// Dir - 运动方向（ 1：正向 -1：负向）
// 输出参数：无
// 返回值：0-成功，-1-失败
// 功能描述：以常速模式启动单轴JOG运动
// 注意事项：运动速度由指令“set_conspeed”设置
//-------------------------------------------------
```

#### 7.2.3.2 con_vmove2

指令原型：int　con_vmove2(int Axis1, int Dir1,int Axis2, int Dir2);

```
//-------------------------------------------------
// 输入参数
//      Axis1 - 轴号，1~32号轴
//      Dir1 - 运动方向（ 1：正向 -1：负向）
//      Axis2 - 轴号，1~32号轴
//      Dir2 - 运动方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：以常速模式启动两轴JOG运动
// 注意事项：运动速度由指令“set_conspeed”设置
//-------------------------------------------------
```

#### 7.2.3.3 con_vmove3

指令原型：int　con_vmove3(int Axis1, int Dir1,int Axis2, int Dir2,int Axis3, int Dir3);

```
//-------------------------------------------------
// 输入参数
//      Axis1 - 轴号，1~32号轴
//      Dir1 - 运动方向（ 1：正向 -1：负向）
//      Axis2 - 轴号，1~32号轴
//      Dir2 - 运动方向（ 1：正向 -1：负向）
//      Axis3 - 轴号，1~32号轴
//      Dir3 - 运动方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：以常速模式启动三轴JOG运动
// 注意事项：运动速度由指令“set_conspeed”设置
//-------------------------------------------------
```

#### 7.2.3.4 con_vmove4

指令原型：int　con_vmove4(int Axis1, int Dir1,int Axis2, int Dir2,int Axis3, int Dir3,int Axis4, int Dir4);

```
//-------------------------------------------------
// 输入参数
//      Axis1 - 轴号，1~32号轴
//      Dir1 - 运动方向（ 1：正向 -1：负向）
//      Axis2 - 轴号，1~32号轴
//      Dir2 - 运动方向（ 1：正向 -1：负向）
//      Axis3 - 轴号，1~32号轴
//      Dir3 - 运动方向（ 1：正向 -1：负向）
//      Axis4 - 轴号，1~32号轴
//      Dir4 - 运动方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：以常速模式启动四轴JOG运动
// 注意事项：运动速度由指令“set_conspeed”设置
//-------------------------------------------------
```

#### 7.2.3.5 fast_vmove

指令原型：int　fast_vmove(int Axis, int Dir);

```
//-------------------------------------------------
```

```
// 输入参数
// Axis ---- 轴号
// Dir - 运动方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：以变速模式启动单轴JOG运动
// 注意事项：运动速度由指令“set_profile”设置
//-------------------------------------------------
```

#### 7.2.3.6 fast_vmove2

指令原型：int  fast_vmove2(int Axis1, int Dir1,int Axis2, int Dir2);

```
//-------------------------------------------------
// 输入参数
//      Axis1 - 轴号，1~32号轴；
//      Dir1 - 运动方向（ 1：正向 -1：负向）
//      Axis2 - 轴号，1~32号轴；
//      Dir2 - 运动方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：以变速模式启动两轴JOG运动
// 注意事项：运动速度由指令“set_profile”设置
//-------------------------------------------------
```

#### 7.2.3.7 fast_vmove3

指令原型：int  fast_vmove2(int Axis1, int Dir1,int Axis2, int Dir2,int Axis3, int Dir3);

```
//-------------------------------------------------
// 输入参数
//      Axis1 - 轴号，1~32号轴；
//      Dir1 - 运动方向（ 1：正向 -1：负向）
//      Axis2 - 轴号，1~32号轴；
//      Dir2 - 运动方向（ 1：正向 -1：负向）
//      Axis3 - 轴号，1~32号轴；
//      Dir3 - 运动方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：以变速模式启动三轴JOG运动
// 注意事项：运动速度由指令“set_profile”设置
//-------------------------------------------------
```

#### 7.2.3.8 fast_vmove4

指令原型：int  fast_vmove2(int Axis1, int Dir1,int Axis2, int Dir2,int Axis3, int Dir3,int Axis4, int Dir4);

```
//-------------------------------------------------
// 输入参数
//      Axis1 - 轴号，1~32号轴；
//      Dir1 - 运动方向（ 1：正向 -1：负向）
//      Axis2 - 轴号，1~32号轴；
//      Dir2 - 运动方向（ 1：正向 -1：负向）
//      Axis3 - 轴号，1~32号轴；
//      Dir3 - 运动方向（ 1：正向 -1：负向）
//      Axis4 - 轴号，1~32号轴；
//      Dir4 - 运动方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：以变速模式启动四轴JOG运动
// 注意事项：运动速度由指令“set_profile”设置
```

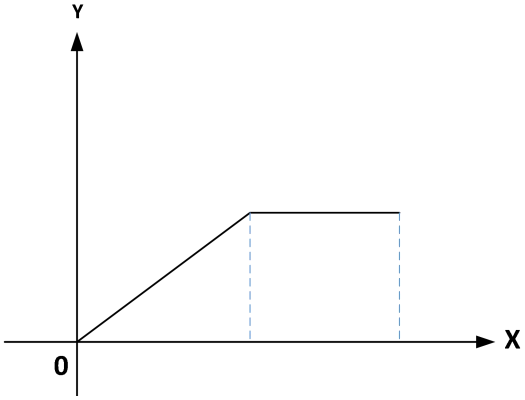
```
//-----------------------------------------
```

## 7.3. 定位运动的编程

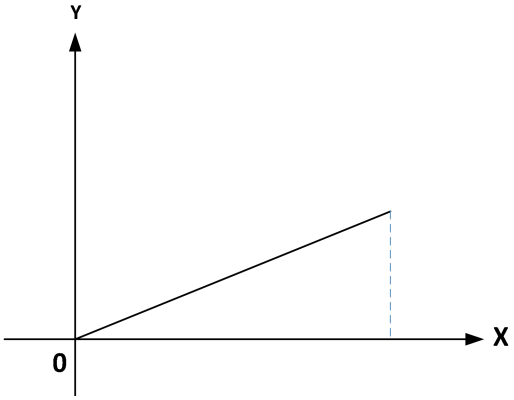
### 7.3.1应用说明

应用场景关联：定位运动的编程，往往与设备某个控制对象在加工过程的快速定位控制要求关联。现实中的快速定位场景，主要是设备的某个或某几个轴同时启动，按各自能达到最快速度运动到指定位置。多轴时，其特性是运动同时启动但不一定同时达到。区别于直线插补运动，各轴速度曲线时序详见下图：X 轴为轴 1，Y 轴为轴 2。X 轴从起始位置 0 运动到 2000，Y 轴从起始位置 0 运动到 1000。

绘制两种运动模式下，XY 平面上的运动路径图。

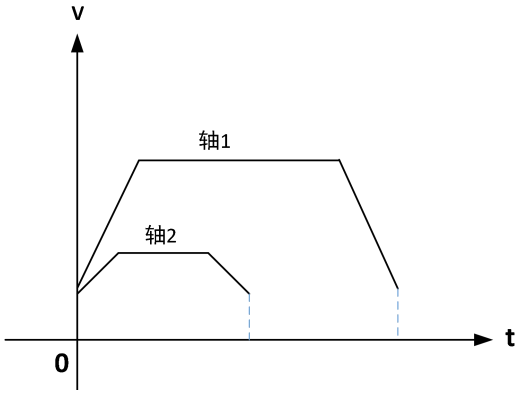



**两轴定位运动路径图**

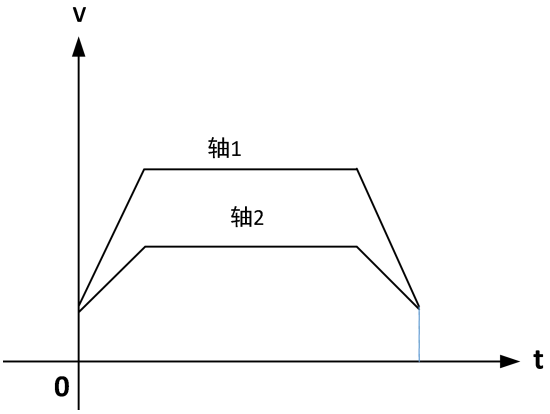



**两轴插补路径图**

分别绘制两轴定位运动与两轴直线插补的速度曲线图对比。




**两轴定位运动速度时间图**




**两轴插补运动速度时间图**

在插补运动中，2 轴同时启动，并同时到达终点。X、Y 轴的运动速度之比为 ΔX：ΔY，二轴合成的矢量速度为：

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

插补 3 轴运动中 3 轴的速度比为 ΔX：ΔY：ΔZ，三轴合成的矢量速度为：

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

插补 4 轴运动中 4 轴的速度比为 ΔX：ΔY：ΔZ：ΔU，四轴合成的矢量速度为：

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

## 7.3.2调用示例

**两轴定位运动**

```
set_abs_pos(1, 0);//设置 1 轴当前位置为绝对位置 0
reset_pos(2);//设置 2 轴当前位置为绝对位置 0
set_s_curve(1, 0); //设置 1 轴加减速为梯形速度模式
set_s_curve(2, 0); //设置 2 轴加减速为梯形速度模式
set_maxspeed(1, 5000); //设置 1 轴最大速度为 5000
set_maxspeed(2, 5000); //设置 2 轴最大速度为 5000
set_profile(1, 100, 5000, 2000, 2000); //设置 1 轴初速、高速、加速度和减速度
set_profile(1, 80, 3000, 2000, 2000);  //设置 2 轴初速、高速、加速度和减速度
fast_pmove2(1, 2000, 2, 1000);//1 轴 2 轴以梯形快速运动方式运动到 2000，1000
//启动一段时间后
double dbPos = 0, dbRate = 0;
get_abs_pos(1, &dbPos);//获取 1 轴的绝对位置
dbRate=get_rate(1);//获取1轴的运动速度
```

## 7.3.3指令详解

### 7.3.3.1 set_conspeed

指令原型：int  set_conspeed(int Axis, double speed);

```
//-------------------------------------------------
// 输入参数
//      Axis - 轴号，1~32号轴;
//      speed - 常速速度（正值），范围10~8M pps
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：设置轴常速运行速度
// 注意事项：设置con_pmove、con_pmove_to指令的运动速度
//-------------------------------------------------
```

### 7.3.3.2 set_s_curve

指令原型：int  set_s_curve(int Axis, int mode);

```
//-------------------------------------------------
// 输入参数
//      Axis - 轴号，1~32号轴;
//      mode- 0：梯形加减速    1：S型加减速
```

```
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：设置快速运动曲线模式
// 注意事项：系统默认为梯形速度模式
//-------------------------------------------------
```

#### 7.3.3.3 set_profile

指令原型：int  set_profile(int Axis, double vl, double vh, double ad, double dc);

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// vl ---- 快速运动低速（正值），范围：10~8M pps
// vh ---- 快速运动高速（正值），范围 :10≤vl≤vh≤8M pps
// ad ---- 上升加速度（正值），范围 :10≤ad≤12.5M pps
// dc ---- 下降加速度（正值），范围 :10≤dc≤12.5M pps
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：设置轴快速运动参数
//-------------------------------------------------
```

#### 7.3.3.4 set_s_section

指令原型：int  set_s_section(int Axis, double accelSec, double decelSec);

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// accelSec ---- 加加速段和减加速段的加加速度值，范围10~375M ；
// decelSec ---- 加减速段和减减速段的加减速度值，和accel_sec值相同，否则调用会失败
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：设置轴 S 型曲线运动加加速度
// 注意事项：在“set_profile”之后调用，只能在快速点位运动中有效，插补运动不提供S曲线加减速模式。加加速段和减加速段的加加速度值，范围10~375M，且加减速段和减减速段的加减速度值必须相等，否则调用会失败。
//-------------------------------------------------
```

#### 7.3.3.5 con_pmove

指令原型：int con_pmove(int Axis, double Step);

```
//-------------------------------------------------
// 输入参数
// Axis---- 轴号
// Step ---- 相对当前位置的位移（正值为正向，负值为负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：单轴常速相对位置点位运动
// 注意事项：
//-------------------------------------------------
```

#### 7.3.3.6 con_pmove_to

指令原型：int con_pmove_to(int Axis, double Step);

```
//-------------------------------------------------
// 输入参数
// Axis---- 轴号
// Step ---- 绝对位置值
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：单轴常速绝对位置点位运动指令
```

```
// 注意事项：
//-------------------------------------------------
```

#### 7.3.3.7 con_pmove2

指令原型：int  con_pmove2(int Axis1, double Step1, int Axis2, double Step2);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Step1 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis2 ---- 轴号
// Step2 ---- 相对当前位置的位移（正值为正向，负值为负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：两轴常速绝对位置点位运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.3.3.8 con_pmove3

指令原型：int  con_pmove3(int Axis1, double Step1, int Axis2, double Step2, int Axis3, double Step3);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Step1 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis2 ---- 轴号
// Step2 ---- 相对当前位置的位移（正值为正向，负值为负向）
 // Axis3 ---- 轴号
// Step3 ---- 相对当前位置的位移（正值为正向，负值为负向）

// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：三轴常速绝对位置点位运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.3.3.9 con_pmove4

指令原型：int con_pmove4(int Axis1, double Step1, int Axis2, double Step2,int Axis3, double Step3, int Axis4,

double Step4);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Step1 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis2 ---- 轴号
// Step2 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis3 ---- 轴号
// Step3 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis4 ---- 轴号
// Step4 ---- 相对当前位置的位移（正值为正向，负值为负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：四轴常速绝对位置点位运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.3.3.10 fast_pmove

指令原型：int　fast_pmove(int Axis, double Step);

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// Step ---- 相对当前位置的位移（正值为正向，负值为负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：单轴快速相对位置点位运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.3.3.11 fast_pmove_to

指令原型：int fast_pmove_to(int Axis, double Step)

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// Step ---- 绝对位置值
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：单轴快速绝对位置点位运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.3.3.12 fast_pmove2

指令原型：int　fast_pmove2(int Axis1, double Step1,int Axis2, double Step2);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Step1 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis2 ---- 轴号
// Step2 ---- 相对当前位置的位移（正值为正向，负值为负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：两轴快速相对位置点位运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.3.3.13 fast_pmove3

指令原型：int　fast_pmove3(int Axis1, double Step1,int Axis2, double Step2,int Axis3, double Step3);

```
//-------------------------------------------------
// 输入参数
// Axis1---- 轴号
// Step1 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis2---- 轴号
// Step2 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis3---- 轴号
// Step3 ---- 相对当前位置的位移（正值为正向，负值为负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：三轴快速相对位置点位运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.3.3.14 fast_pmove4

指令原型：int　fast_pmove4(int Axis1, double Step1,int Axis2, double Step2,int Axis3, double Step3,int Axis4,

double Step4);

```
//-------------------------------------------------
// 输入参数
// Axis1---- 轴号
// Step1 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis2---- 轴号
// Step2 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis3---- 轴号
// Step3 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis4---- 轴号
// Step4 ---- 相对当前位置的位移（正值为正向，负值为负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：四轴快速相对位置点位运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.3.3.15 set_abs_pos

指令原型：int　set_abs_pos(int Axis, double Pos);

```
//-------------------------------------------------
// 输入参数
// Axis---- 轴号
// Pos ---- 绝对位置值（正值）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：设置轴当前绝对位置
// 注意事项：
//-------------------------------------------------
```

#### 7.3.3.16 reset_pos

指令原型：int reset_pos(int Axis);

```
//-------------------------------------------------
// 输入参数
// Axis---- 轴号
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：将轴的绝对位置和相对位置清零
// 注意事项：
//-------------------------------------------------
```

#### 7.3.3.17 get_abs_pos

指令原型：int　get_abs_pos(int Axis, double *pos);

```
//-------------------------------------------------
// 输入参数
// Axis---- 轴号
// pos ---- 绝对位置
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：获取轴当前绝对位置
// 注意事项：
//-------------------------------------------------
```

**7.3.3.18 get_rate**

指令原型：double get_rate(int Axis);

```
//--------------------------------------------------
// 输入参数
// Axis---- 轴号
// 输出参数：
// 返回值：非负数-成功，负数-失败
// 功能描述：获取轴的当前速度
// 注意事项：
//--------------------------------------------------
```

# 7.4. 直线插补运动的编程

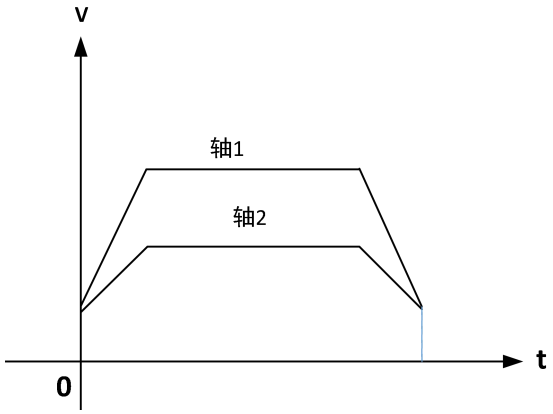
## 7.4.1应用说明

应用场景关联：直线插补运动的编程场景，主要是设备的某几个轴同时启动，各轴按照设定插补速度的分速度到达指定位置，其特性是运动同时启动同时达到。例如在异型插件机设备应用中，当有板进料后，XY 需运动带动机头上的相机到 Mark 点拍照，此时 XY 轴使用插补运动控制，同时启动，同时到达。

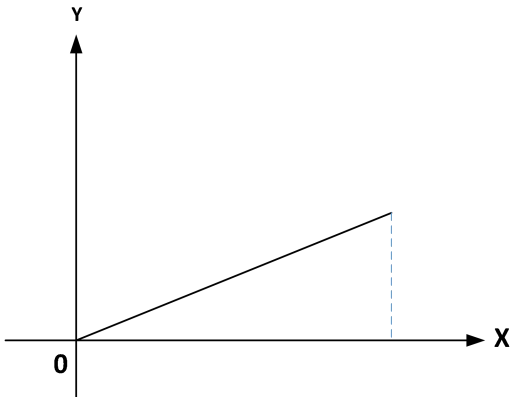
## 7.4.2调用示例

**两轴插补运动**

```
set_s_curve(1, 0); //设置 1 轴梯形速度模式
set_s_curve(2, 0); //设置 2 轴梯形速度模式
set_maxspeed(1, 5000); //设置 1 轴最大速度为 5000
set_maxspeed(2, 5000); //设置 2 轴最大速度为 5000
set_vector_profile(200, 5000, 3000, 3000);
fast_line2(1, 2000, 2, 1000);//1 轴2轴以梯形插补运动到2000, 1000
```




**两轴定位运动速度时间图**




**两轴插补路径图**

在插补运动中，2 轴同时启动，并同时到达终点。X、Y 轴的运动速度之比为 ΔX：ΔY，二轴合成的矢量速度为：

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

插补 3 轴运动中 3 轴的速度比为 ΔX：ΔY：ΔZ，三轴合成的矢量速度为：

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

插补 4 轴运动中 4 轴的速度比为 ΔX：ΔY：ΔZ：ΔU，四轴合成的矢量速度为：

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

## 7.4.3指令详解

### 7.4.3.1 set_vector_conspeed

指令原型：int  set_vector_conspeed(double conspeed);

```
//-------------------------------------------------
// 输入参数：
// conspeed---- 常速速度（正值），范围 10~8M pps
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：设置常速插补运动运行速度（矢量速度）
// 注意事项：
//-------------------------------------------------
```

### 7.4.3.2 set_vector_profile

指令原型：int set_vector_profile(double vec_vl, double vec_vh, double vec_ad, double vec_dc)

```
//-------------------------------------------------
// 输入参数：
// vec_vl ---- 快速运动低速（正值），范围 10~4M pps
// vec_vh ---- 快速运动高速（正值），范围 10~4M pps
// vec_ad ---- 上升加速度（正值），范围 10~12.5M pps
// vec_dc ---- 下降加速度（正值），范围 10~12.5M pps
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：设置快速插补运动参数（矢量速度）
// 注意事项：
//-------------------------------------------------
```

### 7.4.3.3 con_line2

指令原型：int con_line2(int Axis1, double Step1, int Axis2, double Step2)

```
//-------------------------------------------------
// 输入参数：
// Axis1 ---- 轴号
// Step1 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis2 ---- 轴号
// Step2 ---- 相对当前位置的位移（正值为正向，负值为负向）
// 输出参数：
// 返回值：0-成功，-1-失败
```

```
// 功能描述：两轴常速直线插补运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.4.3.4 con_line3

指令原型：int con_line3(int Axis1, double Step1, int Axis2, double Step2, int Axis3, double Step3)

```
//-------------------------------------------------
// 输入参数：
// Axis1 ---- 轴号
// Step1 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis2 ---- 轴号
// Step2 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis3 ---- 轴号
// Step3 ---- 相对当前位置的位移（正值为正向，负值为负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：三轴常速直线插补运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.4.3.5 con_line4

指令原型：int con_line4(int Axis1, double Step1, int Axis2, double Step2,int Axis3, double Step3, int Axis4, double Step4)

```
//-------------------------------------------------
// 输入参数：
// Axis1 ---- 轴号
// Step1 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis2 ---- 轴号
// Step2 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis3 ---- 轴号
// Step3 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis4 ---- 轴号
// Step4 ---- 相对当前位置的位移（正值为正向，负值为负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：四轴常速直线插补运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.4.3.6 con_line2_to

指令原型：int con_line2_to(int Axis1, double Step1, int Axis2, double Step2)

```
//-------------------------------------------------
// 输入参数：
// Axis1 ---- 轴号
// Step1 ---- 相对零点位置的位移（绝对位置）
// Axis2 ---- 轴号
// Step2 ---- 相对零点位置的位移（绝对位置）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：两轴常速直线插补运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.4.3.7 con_line3_to

指令原型：int con_line3_to(int Axis1, double Step1, int Axis2, double Step2,int Axis3, double Step3)

```
//-------------------------------------------------
```

```
// 输入参数：
// Axis1 ---- 轴号
// Step1 ---- 相对零点位置的位移（绝对位置）
// Axis2 ---- 轴号
// Step2 ---- 相对零点位置的位移（绝对位置）
// Axis3 ---- 轴号
// Step3 ---- 相对零点位置的位移（绝对位置）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：三轴常速直线插补运动指令
// 注意事项：
//------------------------------------------------
```

#### 7.4.3.8 con_line4_to

指令原型：int con_line4_to(int Axis1, double Step1, int Axis2, double Step2,int Axis3, double Step3, int Axis4, double Step4)

```
//------------------------------------------------
// 输入参数：
// Axis1 ---- 轴号
// Step1 ---- 相对零点位置的位移（绝对位置）
// Axis2 ---- 轴号
// Step2 ---- 相对零点位置的位移（绝对位置）
// Axis3 ---- 轴号
// Step3 ---- 相对零点位置的位移（绝对位置）
// Axis4 ---- 轴号
// Step4 ---- 相对零点位置的位移（绝对位置）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：四轴常速直线插补运动指令
// 注意事项：
//------------------------------------------------
```

#### 7.4.3.9 fast_line2

指令原型：int fast_line2(int Axis1, double Step1, int Axis2, double Step2)

```
//------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Step1 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis2 ---- 轴号
// Step2 ---- 相对当前位置的位移（正值为正向，负值为负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：两轴快速直线插补运动指令
// 注意事项：
//------------------------------------------------
```

#### 7.4.3.10 fast_line3

指令原型：int fast_line3(int Axis1, double Step1, int Axis2, double Step2, int Axis3, double Step3)

```
//------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
```

```
// Step1 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis2 ---- 轴号
// Step2 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis3 ---- 轴号
// Step3 ---- 相对当前位置的位移（正值为正向，负值为负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：三轴快速直线插补运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.4.3.11 fast_line4

指令原型：int fast_line4(int Axis1, double Step1, int Axis2, double Step2,int Axis3, double Step3, int Axis4, double Step4)

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Step1 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis2 ---- 轴号
// Step2 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis3 ---- 轴号
// Step3 ---- 相对当前位置的位移（正值为正向，负值为负向）
// Axis4 ---- 轴号
// Step4 ---- 相对当前位置的位移（正值为正向，负值为负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：四轴快速直线插补运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.4.3.12 fast_line2_to

指令原型：int fast_line2_to(int Axis1, double Step1, int Axis2, double Step2)

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Step1 ---- 相对零点位置的位移（绝对位置）
// Axis2 ---- 轴号
// Step2 ----相对零点位置的位移（绝对位置）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：两轴快速直线插补运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.4.3.13 fast_line3_to

指令原型：int fast_line3_to(int Axis1, double Step1, int Axis2, double Step2, int Axis3, double Step3)

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Step1 ---- 相对零点位置的位移（绝对位置）
// Axis2 ---- 轴号
```

```
// Step2 ----相对零点位置的位移（绝对位置）
// Axis3 ---- 轴号
// Step3 ----相对零点位置的位移（绝对位置）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：三轴快速直线插补运动指令
// 注意事项：
//-------------------------------------------------
```

#### 7.4.3.14 fast_line4_to

指令原型：int fast_line4_to(int Axis1, double Step1, int Axis2, double Step2, int Axis3, double Step3, int Axis4, double Step4)

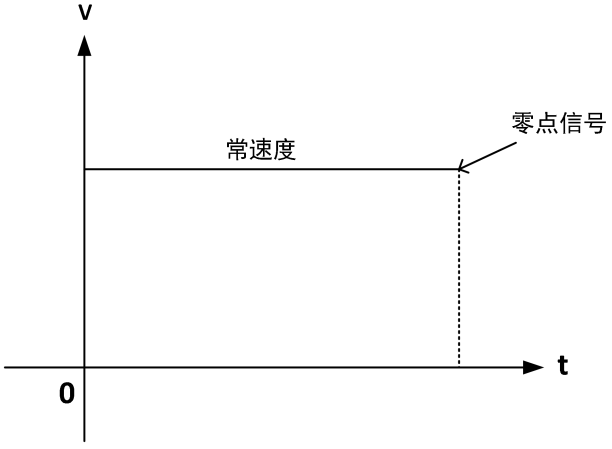
```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Step1 ---- 相对零点位置的位移（绝对位置）
// Axis2 ---- 轴号
// Step2 ----相对零点位置的位移（绝对位置）
// Axis3 ---- 轴号
// Step3 ---- 相对零点位置的位移（绝对位置）
// Axis4 ---- 轴号
// Step4 ----相对零点位置的位移（绝对位置）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：四轴快速直线插补运动指令
// 注意事项：
//-------------------------------------------------
```

## 7.5. 回零运动的编程

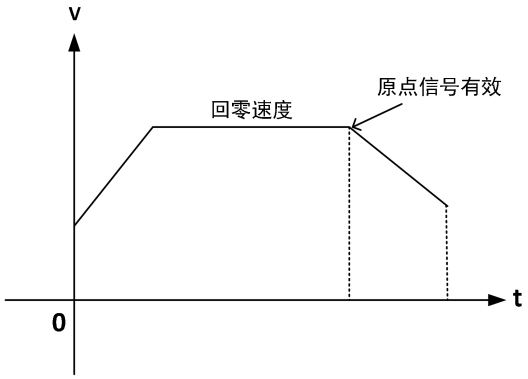
### 7.5.1应用说明

在设备上，一般各轴有固定不变的参考点，其位置由机械挡块或行程开关来确定，通常该参考点作为该轴坐标系的零点。设备每次开机后都会先让各坐标轴回到设备一个固定点上，重新建立各轴的坐标系，这一固定点就是坐标系的零点，使轴回到这一固定点的操作称为回零运动。

P1 提供 4 种回零方式。 通过接口函数“set_home_mode”设置工作方式，详见 4.3 接“轴配置” 。



**原点有效立即停止**




**原点有效减速停止**

## 7.5.2调用示例

**回零运动**

```
set_home_mode(1, 0); //设置 1 轴回原点模式 0 ，立即停止
set_maxspeed(1, 5000); //设置最大速度
set_conspeed(1, 1000); //设置回零速度
con_hmove(1, 1); //启动 1 轴正向回零运动
```

## 7.5.3指令详解

### 7.5.3.1 con_hmove

指令原型：int  con_hmove(int Axis, int Dir);

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// Dir ---- 方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：单轴常速回零运动指令
// 注意事项：常速度由指令“set_conspeed”设置
//-------------------------------------------------
```

### 7.5.3.2 con_hmove2

指令原型：int  con_hmove2(int Axis1, int Dir1,int Axis2, int Dir2);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Dir1 ---- 方向（ 1：正向 -1：负向）
// Axis2 ---- 轴号
// Dir2 ---- 方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：两轴常速回零运动指令
// 注意事项：常速度由指令“set_conspeed”设置
//-------------------------------------------------
```

#### 7.5.3.3 con_hmove3

指令原型：int　con_hmove3(int Axis1, int Dir1,int Axis2, int Dir2,int Axis3, int Dir3);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Dir1 ---- 方向（ 1：正向 -1：负向）
// Axis2 ---- 轴号
// Dir2 ---- 方向（ 1：正向 -1：负向）
// Axis3 ---- 轴号
// Dir3 ---- 方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：三轴常速回零运动指令
// 注意事项：常速度由指令“set_conspeed”设置
//-------------------------------------------------
```

#### 7.5.3.4 con_hmove4

指令原型：int　con_hmove4(int Axis1, int Dir1,int Axis2, int Dir2,int Axis3, int Dir3,int Axis4, int Dir4);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Dir1 ---- 方向（ 1：正向 -1：负向）
// Axis2 ---- 轴号
// Dir2 ---- 方向（ 1：正向 -1：负向）
// Axis3 ---- 轴号
// Dir3 ---- 方向（ 1：正向 -1：负向）
// Axis4 ---- 轴号
// Dir4 ---- 方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：四轴常速回零运动指令
// 注意事项：常速度由指令“set_conspeed”设置
//-------------------------------------------------
```

#### 7.5.3.5 fast_hmove

指令原型：int　fast_hmove(int Axis, int Dir);

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// Dir ---- 方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：单轴快速回零运动指令
// 注意事项：梯形速度由指令“set_profile”设置
//-------------------------------------------------
```

#### 7.5.3.6 fast_hmove2

指令原型：int　fast_hmove2(int Axis1, int Dir1,int Axis2, int Dir2);

```
//-------------------------------------------------
// 输入参数
```

```
// Axis1 ---- 轴号
// Dir1 ---- 方向（ 1：正向 -1：负向）
// Axis2 ---- 轴号
// Dir2 ---- 方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：两轴快速回零运动指令
// 注意事项：梯形速度由指令“set_profile”设置
//-------------------------------------------------
```

#### 7.5.3.7 fast_hmove3

指令原型：int  fast_hmove3(int Axis1, int Dir1,int Axis2, int Dir2,int Axis3, int Dir3);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Dir1 ---- 方向（ 1：正向 -1：负向）
// Axis2 ---- 轴号
// Dir2 ---- 方向（ 1：正向 -1：负向）
// Axis3 ---- 轴号
// Dir3 ---- 方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：三轴快速回零运动指令
// 注意事项：梯形速度由指令“set_profile”设置
//-------------------------------------------------
```

#### 7.5.3.8 fast_hmove4

指令原型：int  fast_hmove4(int Axis1, int Dir1,int Axis2, int Dir2,int Axis3, int Dir3,int Axis4, int Dir4);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Dir1 ---- 方向（ 1：正向 -1：负向）
// Axis2 ---- 轴号
// Dir2 ---- 方向（ 1：正向 -1：负向）
// Axis3 ---- 轴号
// Dir3 ---- 方向（ 1：正向 -1：负向）
// Axis4 ---- 轴号
// Dir4 ---- 方向（ 1：正向 -1：负向）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：四轴快速回零运动指令
// 注意事项：梯形速度由指令“set_profile”设置
//-------------------------------------------------
```

## 7.6. 运动轴的制动

### 7.6.1应用说明

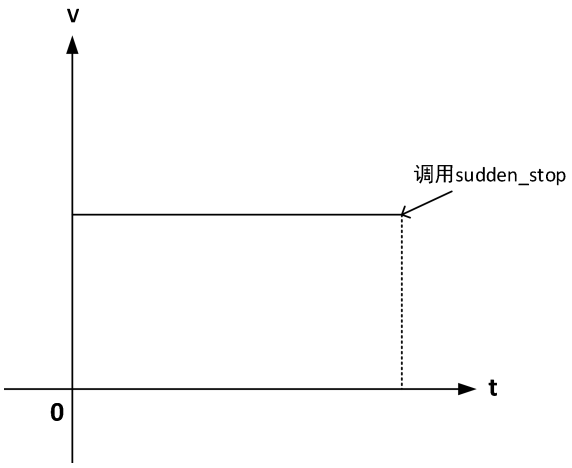
制动函数的作用是让运动的轴停下来。制动函数分为立即制动和光滑制动。

立即制动函数（sudden_stop）使控制器立即停止向电机驱动器发送脉冲，使之停止运动。适用于所有的单轴运动。

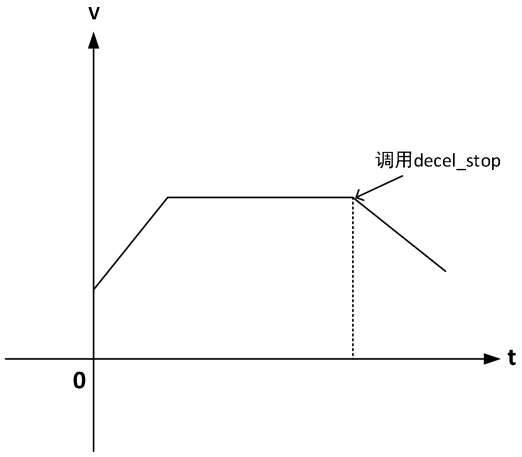
光滑制动函数（decel_stop）只对快速运动指令（梯形速度、S 形速度）有效，即只有“fast”开始

的运动指令（如：fast_hmove、fast_vmove、fast_pmove2 等）才能进行光滑制动。光滑制动函数使被控轴的速度先从高速降至低速（由 set_proflile 设定），然后停止运动。光滑制动函数可有效防止快速运动时系统的过冲现象。

暂停函数(move_pause)的作用是让运动的轴暂停下来,快速运动指令用的是设置的速度(set_profile)的方式光滑制动，常速运动指令用的是立即制动的方式。暂停恢复函数（move_resume）的作用时将暂停下来的轴恢复运动完成剩下的指令，快速运动指令用的是设置的速度（set_profile）的方式进行剩下的运动，常速运动指令立即恢复到原来的速度进行剩下的运动。两个函数需配合使用，且适用于所有单轴运动。




立即制动函数调用后轴立即停止运动




光滑制动函数调用后轴缓慢停止运动

## 7.6.2调用示例

**立即制动函数**

```
set_maxspeed(1, 5000); //设置最大速度
set_conspeed(1, 1000); //设置速度
con_vmove(1, 1); //启动 1 轴正向运动
//启动一段时间后
sudden_stop(1);//停止轴运动
```

**光滑制动函数**

```
set_maxspeed(1, 5000); //设置最大速度
set_profile(1, 100, 5000, 3000);//设置初速、高速、加速度
fast_vmove(1, 1); //启动 1 轴正向连续运动
//启动一段时间后
decel_stop(1);//停止轴运动
```

## 7.6.3指令详解

### 7.6.3.1 sudden_stop

指令原型：int　sudden_stop(int Axis);

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：单轴立即停止
// 注意事项：

//-------------------------------------------------
```

### 7.6.3.2 sudden_stop2

指令原型：int　sudden_stop2(int Axis1,int Axis2);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Axis2 ---- 轴号
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：两轴立即停止
// 注意事项：

//-------------------------------------------------
```

### 7.6.3.3 sudden_stop3

指令原型：int　sudden_stop3(int Axis1,int Axis2,int Axis3);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Axis2 ---- 轴号
// Axis3 ---- 轴号
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：三轴立即停止
// 注意事项：

//-------------------------------------------------
```

### 7.6.3.4 sudden_stop4

指令原型：int　sudden_stop4(int Axis1,int Axis2,int Axis3,int Axis4);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Axis2 ---- 轴号
// Axis3 ---- 轴号
// Axis4 ---- 轴号
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：四轴立即停止
// 注意事项：
```

```
//-------------------------------------------------
```

#### 7.6.3.5 decel_stop

指令原型：int  decel_stop(int Axis);

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：快速运动模式下单轴缓停
// 注意事项：
//-------------------------------------------------
```

#### 7.6.3.6 decel_stop2

指令原型：int  decel_stop2(int Axis1,int Axis2);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Axis2 ---- 轴号
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：快速运动模式下两轴缓停
// 注意事项：
//-------------------------------------------------
```

#### 7.6.3.7 decel_stop3

指令原型：int  decel_stop3(int Axis1,int Axis2,int Axis3);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Axis2 ---- 轴号
// Axis3 ---- 轴号
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：快速运动模式下三轴缓停
// 注意事项：
//-------------------------------------------------
```

#### 7.6.3.8 decel_stop4

指令原型：int  decel_stop4(int Axis1,int Axis2,int Axis3,int Axis4);

```
//-------------------------------------------------
// 输入参数
// Axis1 ---- 轴号
// Axis2 ---- 轴号
```

```
// Axis3 ---- 轴号
// Axis4 ---- 轴号
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：快速运动模式下四轴缓停
// 注意事项：

//-------------------------------------------------
```

#### 7.6.3.9 move_pause

指令原型：int　　move_pause(int Axis)

```
//-------------------------------------------------
// 功能：暂停一个运动轴
  参数：
  Axis ---- 轴号
//返回值：  0（正确）----- -1（错误）

//-------------------------------------------------
```

#### 7.6.3.10 move_resume

指令原型：int　　move_resume(int Axis)

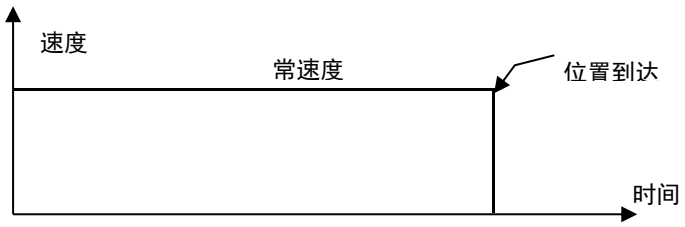
```
//-------------------------------------------------
//功能：恢复一个轴的运动
  参数：
  Axis ---- 轴号
    返回值：  0（正确）----- -1（错误）

//-------------------------------------------------
```

## 7.7. 轴运动速度设置的说明

**1）　set_conspeed 说明**

函数 set_conspeed 设定一个轴在常速运动方式下的速度。常速运动速度曲线如下图所示：




**常速运动方式的速度曲线**

该速度与运动控制器最终脉冲输出速度可能不一致。这是因为输出脉冲频率由两个变量控制：脉冲分辨率和倍率，倍率越大则误差越大。设设置的最大速度为 $V_{max}$，输出倍率为 multip，脉冲实际输出速度为 $V_{act}$，设置的常速度为 $V_{con}$，有如下计算公式：

$$multip = \frac{V_{\max}}{8191}$$

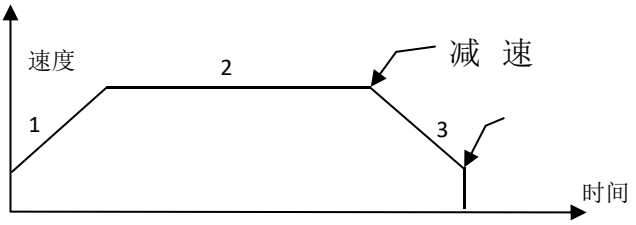
$$v_{act} = \text{int}(\frac{v_{con}}{multip}) \times multip$$

如果多次调用这个函数，最后一次设定的值有效，而且在下一次改变之前，一直保持有效。最大脉冲频率可设置为 8000000 Hz，最小脉冲频率可设置为 0.2 Hz。

**2) set_profile 说明**

函数 set_profile 设定一个轴在快速运动方式下的低速（起始速度）、高速（目标速度）、加 / 减速度值（梯形速度模式下，减速度值可以不等于加速度值）。快速运动方式速度曲线如图所示。

该速度与运动控制器最终脉冲输出速度可能不一致，原因与 set_conspeed 一样。




**快速运动方式的速度曲线**

- 加速段，起始速度按设定的加速度加速到目标速度；
- 高速段，保持目标速度，直至运动到减速点；
- 减速段：目标速度按设定的加速度减速到起始速度；

在该快速运动方式下，低速值脉冲频率最小为 10Hz，高速值脉冲频率最大为 2000000Hz，加速值最小为 20。

**3) set_vector_conspeed 说明**

函数 set_vector_conspeed 设置常速方式下的矢量速度，这个矢量速度在两轴及以上直线插补及圆弧插补常速运动中将会用到。矢量速度与 set_maxspeed 设置的最大速度无直接关系，即 set_maxspeed 函数确定的速度倍率不影响插补运动。最大脉冲频率可设置为 8000000 Hz，最小脉冲频率可设置为 1Hz。

**4) set_vector_profile 说明**

函数 set_vector_profile 设置快速运动方式下的矢量低速为（起始速度）、矢量高速、矢量加/减速度（矢量减速度等于矢量加速度）。这些矢量值在两轴及以上直线插补、圆弧插补快速运动中将会用到。矢量速度与 set_maxspeed 设置的最大速度无直接关系，即 set_maxspeed 函数确定的速度倍率不影响插补运动。低速值脉冲频率最小可设置为 10Hz，高速值脉冲频率最大可设置为 8000000Hz，加速值最小可设置为 20。

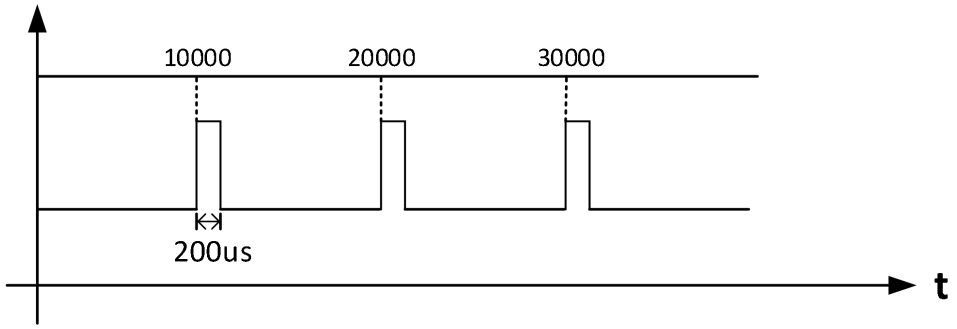
# 8. 位置比较输出

## 8.1. 相关指令列表

| 位置比较输出 | |
|---|---|
| set_compare_outport | 设置轴对应的位置比较输出口 |
| start_comparePulse | 在 HSIO 口直接输出一个脉冲或者电平信号 |
| start_compareData | 设置非等间距模式的位置比较输出 |
| start_compareLinear | 设置等间距位置比较输出输出 |
| get_compareStatus | 读取已输出的位置比较信号数量，即比较过的点数 |
| compare_stop | 取消位置比较输出 |

## 8.2. 应用说明

位置比较输出，就是运动过程中，通过设定比较的目标位置，当电机运动到指定目标位置后，输出设定的电平信号，末端执行器（相机，激光器等）收到信号后可快速执行相应的功能（拍照，出激光）。在视觉应用领域，位置比较输出的通俗叫法是飞拍，飞拍就是使用硬件比较输出或精准输出端口在极短时间内触发相机拍照，而被测物品在拍照过程中仍处于运动状态。飞拍的功能有利于提高触发的效率，实现运动中的数据捕获。下图为多点位置比较输出的示意图。




## 8.3. 调用示例

**位置比较输出函数**

```
    long nPos[3] = { 10000,20000,30000 };
    int nCnt[3] = { 0 };
    set_compare_outport(1, 1); //设置 1 轴位置比较输出口 1
    start_compareData(1, 0, 0, 1, 200, nPos,3);//1 轴脉冲位置作为比较源，在比较位置输出 3 个宽度为 200us 的脉冲
   //启动一段时间后
     get_compareStatus(1, nCnt);//获取比较的点数
```

## 8.4. 指令详解

**8.4.1.1 set_compare_outport**

指令原型：int  set_compare_outport(int Axis, int CmpOut)

```
//-----------------------------------------------------
// 输入参数
// Axis ---- 轴号
// CmpOut ----位置比较输出口
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：设置比较轴的位置比较输出口
// 注意事项：在使用位置比较输出前，先调用该函数配置轴的位置比较输出口
//-----------------------------------------------------
```

**8.4.1.2 start_comparePulse**

指令原型：int  start_comparePulse(int Axis,int PulseType,int Level,long Time)

```
//-----------------------------------------------------
// 输入参数
// Axis ---- 轴号
// PulseType ---- HSIO口输出信号类型：
                          0 --- 输出脉冲信号，脉宽由参数 time 确定
                          1 --- 电平信号
// Level ------ PulseType为0时，本参数设置HSIO是否输出脉冲
                                        0 -- 表示本通道不输出脉冲
                                        1 -- 表示本通道输出脉冲
```

```
                PulseType为1时，本参数设置HSIO的输出电平
                                        0 -- 表示输出低电平
                                        1 -- 表示输出高电平
// Time ---- 参数PulseType为0时，用来设定脉宽，范围[1，65535]，单位：us
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：在HSIO口直接输出一个脉冲或者电平信号
// 注意事项：

//-----------------------------------------------------
```

#### 8.4.1.3 start_compareData

指令原型：int start_compareData(int Axis,int Source,int PulseType,int StartLevel,int Time,long *pBuf,int Count)

```
//-----------------------------------------------------
// 输入参数
// Axis ---- 轴号
// Source ---- 位置比较数据来源：
                                0 ：指令脉冲位置
                                1 ：编码器反馈位置
// PulseType ---- HSIO口输出信号类型：
                                 0 ：输出脉冲信号，脉宽由参数 time 确定
                                 1 ：  电平信号
// StartLevel ---- 设置HSIO信号的初始电平：0 - 低电平  1 - 高电平
          建议设置为1（初始化后默认状态）
// Time ---- 参数pulsetype为0时，用来设定脉宽，  范围[1，65535]，单位：us
// pBuf ---- HSIO的比较位置缓冲区，位置值为绝对位置的距离，必须是单调上升的序列或单调下降的序列
// Count ---- pBuf数组的长度，最大值为4096
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：启动非等间距位置比较输出
// 注意事项：

//-----------------------------------------------------
```

#### 8.4.1.4 start_compareLinear

指令原型：int start_compareLinear(int Axis,int Source,long StartPos,long RepeatTimes, int Interval,int Time)

```
//-----------------------------------------------------
// 输入参数
// Axis ---- 轴号
// Source ---- 位置比较数据来源：
        0 ：指令脉冲位置
        1 ：编码器反馈位置
// StartPos ---- 位置比较起始位置，单位：pulse
// RepeatTimes ---- 位置比较重复次数
// Interval ----位置间隔，单位：pulse。大于0的正整数。
// Time ---- 参数pulsetype为0时，用来设定脉宽，范围[1，  65535]，单位：us
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：启动等间距位置比较输出
// 注意事项：

//-----------------------------------------------------
```

#### 8.4.1.5 get_compareStatus

```
指令原型：int  get_compareStatus(int Axis,int *pCount)

//-------------------------------------------------------

// 输入参数
// pCount ---- 存储位置比较已输出的上升沿个数。若以电平方式输出，则返回值为通道1或通道2已输出的上升沿个数，但与已完成的位置比较个数不等（以电平方式输出时，已完成的位置比较个数为已输出上升沿和已输出下降沿个数的总和）；若以脉冲方式输出，则返回值为通道1或通道2已输出的上升沿个数，与已完成的位置比较个数相等。
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：设置比较轴的位置比较输出口
// 注意事项：

//-------------------------------------------------------
```

#### 8.4.1.6 compare_stop

```
指令原型：int  compare_stop(int Axis)

//-------------------------------------------------------
//功能：取消位置比较输出
  参数：
  Axis ---- 轴号
//返回值：0（正确）-----  -1（出错）

//-------------------------------------------------------
```

# 9. 高速位置锁存

## 9.1. 相关指令列表

| 高速位置锁存 | |
|---|---|
| enable_lock_enc | 使能高速锁存功能 |
| get_locked_flag | 查询是否发生锁存 |
| get_locked_encoder | 查询到控制器已锁存编码器值后，读取锁存值 |

## 9.2. 应用说明

MPC3400 和 MPC3800 提供编码器位置锁存功能，即在外来锁存信号（轴 Z 脉冲接口）触发下，立即响应，控制器自动将当前的编码器值保存。通常用来快速锁定流水线上碰到传感器时的产品位置、快速锁定产品工件的位置、快速锁定胶阀针管位置等。系统默认情况下不启用编码器位置锁存。

锁定胶阀针管位置：

**图-对针机构示意图**

在进行该工艺流程时，需先让设备回零确定零点位置，然后调用相关函数使能编码器位置锁存功能。最后控制 X 直线运动轴到相应的位置，让针管在途中经过相应的激光传感器触发锁存信号。此时，记录 X 轴反馈的编码器位置确定针管在设备坐标系下的 X 轴坐标。

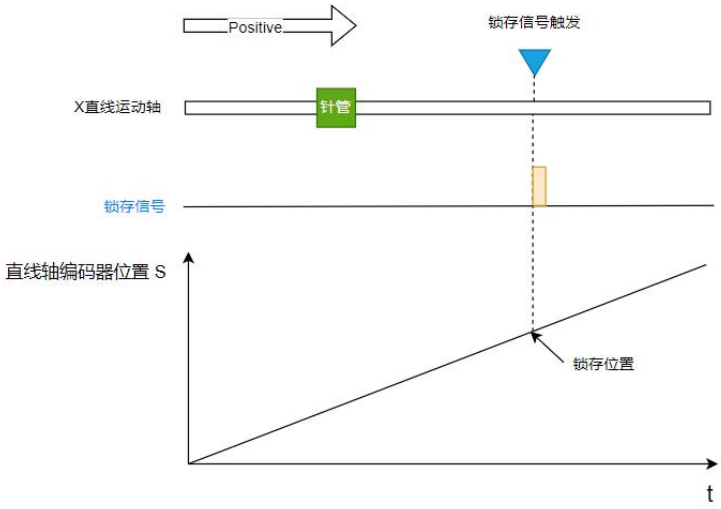



**图-对针工艺流程示意图**

## 9.3. 调用示例

1 号轴上升沿触发单次高速位置锁存

```
long pos = 0;
if (0 == enable_lock_enc(1, 2, 0))//设置 1 号轴为上升沿单次锁存的模式
{
    //设置 1 号轴为高速位置锁存功能成功
}
fast_pmove(1, 200000);//移动，经过该轴 Z 脉冲信号触发位置
if (0 == get_locked_flag(1))//检查该轴 Z 脉冲信号是否触发位置锁存
{
    if (0 == get_locked_encoder(1, 0, &pos))
    {
        //已获取 1 轴锁存位置，根据需求做处理
    }
}
```

## 9.4. 指令详解

9.4.1.1 enable_lock_enc

指令原型:int enable_lock_enc(int Axis, int Flag, int Mode);

//------------------------------------------------

```
// 功能描述：使能高速锁存功能。MPC3400和MPC3800提供编码器位置锁存功能，即在外来锁存信号触发下，控制器自动将当前的编码器值保存。系统默认情况下不启用编码器位置锁存。 使能编码器位置锁存功能后，控制器编码器Z相脉冲输入口定义为锁存信号输入引脚。
// 输入参数：
    Axis ---- 轴号
    Flag---- 编码器锁存使能标记。取值范围：0, 1, 2。
             0-禁止编码器位置锁存；
             1-下降沿触发锁存；
             2-上升沿触发锁存。
   Mode ---- 0- 单次锁存
             1-连续锁存
// 输出参数：无
// 返回值：=0 - 成功， =-1 - 失败
//-------------------------------------------------
```

#### 9.4.1.2 get_locked_flag

指令原型:int get_locked_flag(int Axis);

```
//-------------------------------------------------
// 功能描述： 用户通过该函数查询系统是否发生锁存。当有外部锁存信号触发控制器的锁存操作后，返回标志“1”，表示已锁存到一个编码器值。
// 输入参数：
    Axis ---- 轴号
// 输出参数：无
// 返回值：=0 - 成功， =-1 - 失败
//-------------------------------------------------
```

#### 9.4.1.3 get_locked_encoder

指令原型:  int get_locked_encoder(int Axis,int Num,long *Enc);

```
//-------------------------------------------------
// 功能描述：查询到控制器已锁存编码器值后，通过该函数读取锁存值。
// 输入参数：
    Axis ---- 轴号
    Num----保留，未使用
// 输出参数：Enc-----一个指向锁存值的常整型指针
// 返回值：=0 - 成功， =-1 - 失败
//-------------------------------------------------
```

# 10. 反向间隙补偿

## 10.1. 相关指令列表

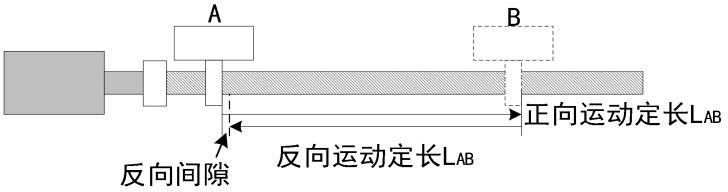
| 反向间隙补偿 | |
|---|---|
| start_backlash | 启动反向间歇补偿 |
| end_backlash | 结束反向间隙补偿 |
| set_backlash | 设置反向间隙补偿 |

## 10.2. 应用说明

当电机带动传动机构进行往复运动时，存在均没走到位且偏移量固定，那机构存在反向间隙。P1 系列控制卡提供的反向间隙补偿指令可以有效消除反向间隙，保证系统位置精度。

反向间隙应用场景：（丝杆传动方式）

物体从起点 A 点运动一定距离到 B 点，在反向从起点 B 运动相同距离到 A，在换向运动中传动机构产生了反向间隙。在应用软件编程中，先根据测量的反向间隙值调用 set_backlash 设置反向间隙补偿参数，然后在发送运动指令之前先调用 start_backlash 启动反向间隙补偿，当运动全部完成后调用 end_backlash 结束反向间隙补偿。



反向间隙生效后，以第一条运动指令的运动方向为正方向。假如第二条运动指令与第一条运动指令方向相反，则在第二条指令运动时会添加反向间歇补偿。

举例：

1）物体处于位置 A 为 0 时,设置使能反向间隙 x,然后物体从 A 往 B 处运动长度为 L 停止后。读取物　体绝对位置（get_abs_pos）的值，此时应为 L。

2）物体反向运动从 B 到 A，运动长度为-L 停止后，读取物体绝对位置（get_abs_pos）的值，此时应为-x。

3）物体反向运动从 A 到 B，运动长度为 L 停止后，读取物体绝对位置（get_abs_pos）的值，此时应为 L。

使能反向间隙补偿，不会对位置比较输出（使用指令位置）造成影响。

## 10.3. 调用示例

**反向间隙补偿**

```
set_s_curve(1, 0); //设置 1 轴梯形速度模式
set_maxspeed(1, 5000); //设置 1 轴最大速度为 5000
set_profile(1, 100, 5000, 2000,2000); //设置 1 轴初速、高速、加速度和减速度
set_backlash(1, 20);//设置 1 轴反向间隙为 20
start_backlash(1);//1 轴启动反向间隙补偿
fast_pmove(1, 10000);//1 轴以梯形快速运动方式从 A 运动到 B
//一段时间等待轴运动停止后
fast_pmove(1, -10000);//1 轴以梯形快速运动方式从 B 运动到 A
end_backlash(1);//1轴结束反向间隙补偿
```

## 10.4. 指令详解

**10.4.1.1 start_backlash**

指令原型：int　start_backlash(int Axis);

```
//-----------------------------------------------
// 功能描述：启动反向间歇补偿
// 输入参数：
//      Axis    - 轴号，取值：1~32
// 输出参数：无
// 返回值：0-成功，-1-失败
//-----------------------------------------------
```

**10.4.1.2 end_backlash**

指令原型：int　end_backlash(int Axis);

```
//-----------------------------------------------
// 功能描述：结束反向间隙补偿
// 输入参数：
//      Axis    - 轴号，取值：1~32
// 输出参数：无
// 返回值：0-成功，-1-失败
```

```
//-----------------------------------------------
```

#### 10.4.1.3 set_backlash

指令原型：int  set_backlash(int Axis, double Blash);

```
//-----------------------------------------------
// 功能描述：打开或关闭多个通用输出口
// 输入参数：
//      Axis  - 轴号，取值：1~32
//      Blash - 补偿值，取值：正数
// 输出参数：无
// 返回值：0-成功，-1-失败
//-----------------------------------------------
```

# 11. 螺距误差补偿

## 11.1. 相关指令列表

| 螺距误差补偿 | |
|---|---|
| set_leadscrew_comp | 设置螺距补偿参数 |
| enable_leadscrew_comp | 设置螺距补偿的使能与禁止 |

## 11.2. 应用说明

螺距误差产生的原因：

滚珠丝杠由于加工制造存在偏差，丝杠每一处螺距有可能不完全一致，螺距的累积误差造成了实际目标位置偏差。在设备装配过程中，丝杠轴线导轨平行度的误差也会产生目标位置偏差。

螺距误差补偿的作用

螺距误差补偿通过调整控制系统的参数增减指令值的脉冲数，实现实际移动距离与指令移动距离相接近，以提高设备的定位精度。

螺距误差测量方法：

轴回零完成之后，先让轴往正方向运动，每走完一个设定步长得到一个理论值与实际值偏差作为补偿值。

螺距误差的影响：

设置该功能会影响轴绝对位置（获取到的绝对位置会添加误差补偿值），该功能会影响位置比较输出（在指令位置比较模式下，指令比较位置会添加误差补偿值）。

## 11.3. 调用示例

螺距误差补偿

```
int Axis = 1;
double dbCompPos[10] = { 1,1,-1,2,-1,-1,1,1,-1,-1 };
double dbCompNeg[10] = { 1,1,-1,2,-1,-1,1,1,-1,-1 };
set_leadscrew_comp(Axis, 10, 0, 100, dbCompPos, dbCompNeg); //设置螺距补偿参数
enable_leadscrew_comp(Axis, 1);//使能螺距补偿
set_profile(Axis, 1000, 2000, 2000, 2000);
fast_pmove_to(Axis, 1000);
```

## 11.4. 指令详解

### 11.4.1.1 set_leadscrew_comp

指令原型：int　set_leadscrew_comp (int Axis,　int Num, double StartPos, double LenPos, double *compos, double *compNeg)

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// Num ---- 补偿点数,2~100
// StartPos------ 补偿起始位置
// LenPos ---- 补偿段的总长度
// Compos ---- 对应正向运动时，各点位置的补偿值
// ComNeg ---- 对应负向运动时，各点位置的补偿值
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：设置螺距补偿功参数
// 注意事项：
//-------------------------------------------------
```

### 11.4.1.2 enable_leadscrew_comp

指令原型：int　enable_leadscrew_comp (int Axis,　int Enable)

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// Enable ----螺距补偿使能状态， 0：禁止， 1：使能
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：使能或禁止螺距补偿功能
// 注意事项：当前版本只对正向螺距误差补偿，负向没有螺距误差补偿。
//-------------------------------------------------
```

# 12. 运动变速/变位

## 12.1. 相关指令列表

| 运动变速/变位 | |
|---|---|
| change_speed | 运动中变速度 |
| change_pos | 动态改变目标位置，目标位置为相对位置 |
| change_pos_to | 动态改变目标位置，目标位置为绝对位置 |

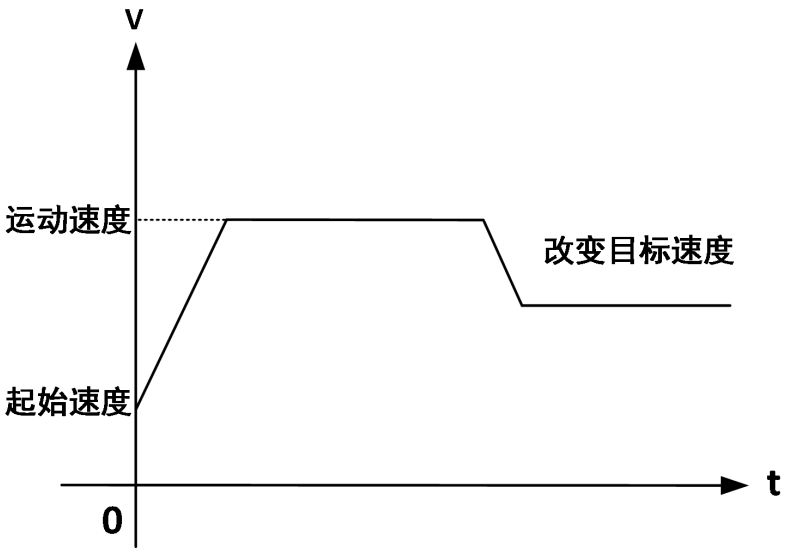
## 12.2. 应用说明

在电子装配设备在加工过程中，往往有取料-拍照纠偏-放料的过程，这个过程中，在 XY 轴带动机头在取料位置完成取料后，会以一定的速度到拍照位置给物料拍照，拍照完成再快速移动到放料位置。从拍照完成到计算出偏移角度需要一定的时间，为提高效率，编程时可以不等计算结果处来，XY 轴带动机头先向理论的拍照位置移动，在这个过程中，偏移角度计算出来了，再在运动中改变目标位置和速度。调用“ change_speed”立即改变控 制轴运动速度，但最大值不能超过“ set_maxspeed”设置的最大速度，最小值不能低于 0.2Hz。

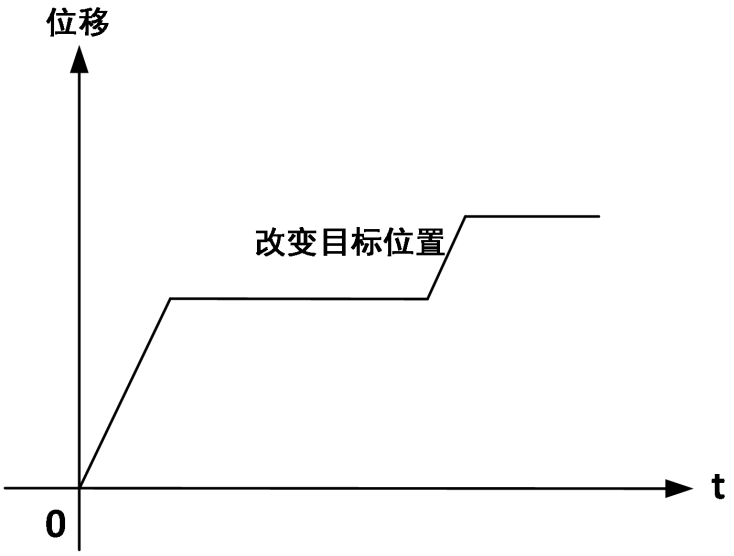
## 12.3. 调用示例

运动中变位置

```
int Axis = 1;
set_profile(Axis, 1000, 2000, 2000, 2000);
fast_pmove_to(Axis, 10000);
//运动一段时间后
 change_pos_to(Axis, 20000);//终点位置变为20000
```




运动中改变速度




运动中改变目标位置

## 12.4. 指令详解

#### 12.4.1.1 change_speed

指令原型：int change_speed (int Axis, double speed)

```
//--------------------------------------------------
// 输入参数
// Axis ---- 轴号
// Speed---目标速度
// 输出参数：
// 返回值：0-成功，-1-失败
```

```
// 功能描述：运动中改变速度
// 注意事项：

//-------------------------------------------------
```

#### 12.4.1.2 change_pos

指令原型：int  change_pos(int Axis, double Pos)

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// Pos ---- 目标位（相对于上条运动指令起始位置）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：运动中改变目标位置，目标位置为相对位置
// 注意事项：

//-------------------------------------------------
```

#### 12.4.1.3 change_pos_to

指令原型：int  change_pos_to(int Axis, double Pos)

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// Pos ---- 目标位（相对于零点位置）
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：运动中改变目标位置，目标位置为绝对位置
// 注意事项：

//-------------------------------------------------
```

# 13. 手轮功能的编程

## 13.1. 相关指令列表

| 手轮功能 | |
|---|---|
| enable_handwheel | 使能轴为手轮模式 |

## 13.2. 应用说明

在 3C 电子装配装备领域，厂家使用手轮的主要用途是辅助操作人员进行工艺点位置定位，模拟工艺路径。

P1 系列的 MPC3400 和 MPC3800 提供了手脉功能。调用该函数使能手脉功能，并设置跟随倍率后，控制卡的指定轴的辅助编码器设定为手脉输入接口，跟随倍率只能设置为 1，10，100。只能在要设置的轴没有运动时将其设置为手脉工作模式。手脉功能不能与编码器锁存功能同时启动。手脉功能启动后，该轴不能发运动指令。

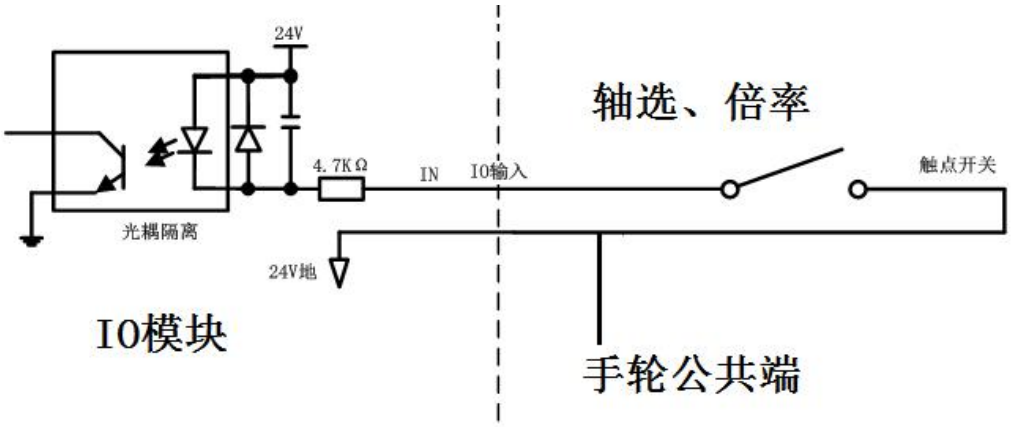
## 13.3. 调用示例

**制动函数**

```
int tAxis = 1;//选择轴 1，轴号根据轴选信号选择
int tMul = 10;//设备倍率为 10，倍率根据倍率信号选择
int tMode = 1;//选择手轮模式
enable_handwheel(tAxis, tMul, tMode);//启动该轴手轮功能
//操作手轮
int tMode = 0;//关闭手轮模式
enable_handwheel(tAxis, tMul, tMode);//关闭该轴手轮功能
```

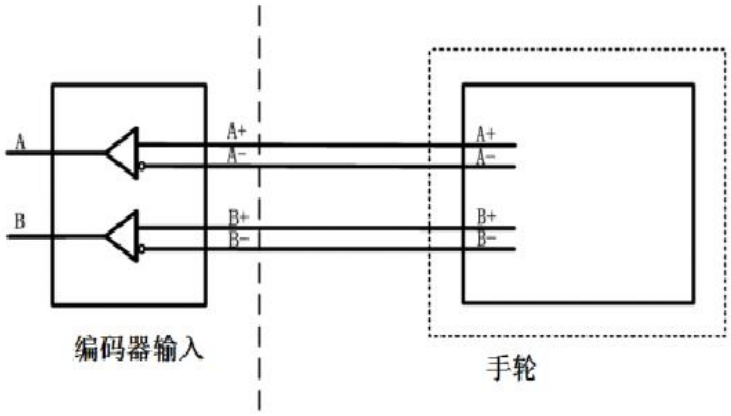
## 13.4. 硬件接口

1）轴选、倍率与 MPC3400、MPC3800 通过 IO 信号控制。




**电气原理图**

2）手轮脉冲信号与 MPC3400、MPC3800 编码器通过差分信号相连。




**差分信号电气原理图**

## 13.5. 指令详解

#### 13.5.1.1 enable_handwheel

指令原型：　int　enable_handwheel (int Axis,　int Mul, int Mode)

```
//-------------------------------------------------
// 输入参数
// Axis ---- 轴号
// Mul---- 手轮跟随倍率，规定设为1，10，100
// Mode------ 使能控制标记，1-使能手轮控制，0-取消手轮控制
// 输出参数：
// 返回值：0-成功，-1-失败
// 功能描述：设置是否使能轴为手轮模式
// 注意事项：

//-------------------------------------------------
```

# 14. 其他操作编程

## 14.1. 相关指令列表

| 系统信息查询 | |
|---|---|
| get_max_axe | 获取总轴数 |
| get_board_num | 获取总卡数 |
| get_axe | 获取控制卡的轴数 |
| get_lib_ver | 获取函数库文件版本号 |
| get_sys_ver | 获取第一张卡的驱动程序版本 |
| get_sys_ver_ex | 获取控制卡驱动程序版本号 |
| get_card_ver | 获取控制卡固件程序版本号 |
| get_sn | 获取控制卡的序列号 |
| **参数设置查询** | |
| get_profile | 获取设定的单轴变速参数 |
| get_vector_conspeed | 获取设定的矢量常速参数 |
| get_vector_profile | 获取设定的矢量变速参数 |
| check_IC | 查询卡号是否设置正确? |
| read_input_filter | 查询通用输入口软件滤波设置 |
| read_special_input_filter | 查询专用输入口软件滤波设置 |
| **错误信息查询** | |
| get_err | 获取控制卡最近 10 条错误的信息 |
| get_last_err | 获取控制卡最进 1 条错误的信息 |
| reset_err | 清除控制卡的错误信息 |
| **自定义存储区操作** | |
| write_password_flash | 设置控制卡加密用户通用存储区的密码 |
| read_password_flash | 获取控制卡加密用户通用存储区的密码 |
| clear_password_flash | 清除控制卡加密用户通用存储区设置的密码 |

| | |
|---|---|
| write_flash | 写控制卡非加密通用存储区 |
| read_flash | 读控制卡非加密通用存储区 |
| clear_flash | 将指定片区数据擦写 bit 位置 1 |
| **其他** | |
| set_dir | 设置轴信号方向的电平 |
| outport_byte_ex | 空操作（兼容以前版本） |
| check_exoutport_status | 空操作（兼容以前版本） |

## 14.2. 系统信息查询

### 14.2.1应用说明

在控制卡正常初始化以后，调用本章节的函数分别用于读取运动控制器附带的总卡数、总轴数、函数库版本号、驱动程序版本号、板卡版本号等运动控制卡参数进行读取。以便于定位版本问题，确认版本功能。

### 14.2.2调用示例

**获取控制卡信息**

```
int mTotalCards=0,mTotalAxis = 0;
mTotalCards = get_board_num();//获取所有控制卡总卡数
mTotalAxis = get_max_axe();//获取所有控制卡所拥有的轴总数
int mAxis = get_axe(1);//获取卡 1 的轴数
long major, long minor1, long minor2;
get_lib_ver(&major,&minor1,&minor2);//获取当前函数库版本
```

### 14.2.3指令详解

**14.2.3.1 get_max_axe**

指令原型：int get_max_axe();

```
//-------------------------------------------------
// 功能描述：获取所有控制卡所拥有的轴总数。
// 输入参数：无
// 输出参数：无
// 返回值：>1 - 成功，获取总轴数  =-1 - 失败
//-------------------------------------------------
```

**14.2.3.2 get_board_num**

指令原型：int get_board_num();

```
//-------------------------------------------------
// 功能描述：获取总卡数。
// 输入参数：无
// 输出参数：无
// 返回值：>0 - 成功，获取总卡数  =-1 - 失败
//-------------------------------------------------
```

#### 14.2.3.3 get_axe

指令原型：int get_axe(int cardno);

```
//-------------------------------------------------
// 功能描述：获取特定卡号的轴数。
// 输入参数：
    cardno:获取的卡号
// 输出参数：无
// 返回值：>1 - 成功，获取的轴数  =-1 - 失败
//-------------------------------------------------
```

#### 14.2.3.4 get_lib_ver

指令原型：int get_lib_ver(long* major, long* minor1, long* minor2);

```
//-------------------------------------------------
// 功能描述：获取当前函数库版本。
// 输入参数：无
// 输出参数：
    major ---- 主版本号
    minor1---- 一级版本号
    minor2---- 二级版本号
// 返回值：=0 - 成功，  =-1 - 失败
//-------------------------------------------------
```

#### 14.2.3.5 get_sys_ver

指令原型：int get_sys_ver(long* major, long* minor1, long* minor2);

```
//-------------------------------------------------
// 功能描述：获取第一张卡的驱动程序版本
// 输入参数：无
// 输出参数：
    major ---- 主版本号
    minor1---- 一级版本号
    minor2---- 二级版本号
// 返回值：=0 - 成功，  =-1 - 失败
//-------------------------------------------------
```

#### 14.2.3.6 get_sys_ver_ex

指令原型：int get_sys_ver_ex(int cardno,long* major, long* minor1, long* minor2);

```
//-------------------------------------------------
// 功能描述：获取指定卡的驱动程序版本。
// 输入参数：cardno ---- 卡号
// 输出参数：
    major ---- 主版本号
    minor1---- 一级版本号
    minor2---- 二级版本号
// 返回值：=0 - 成功，  =-1 - 失败
//-------------------------------------------------
```

#### 14.2.3.7 get_card_ver

指令原型：int get_card_ver(int cardno, long* type, long* major, long* minor1, long* minor2);

```
//-------------------------------------------------
// 功能描述：获取指定卡的驱动程序版本。
// 输入参数：cardno ----卡号
```

```
// 输出参数：
    major ---- 主版本号
    minor1---- 一级版本号
    minor2---- 二级版本号
// 返回值：=0 - 成功，  =-1 - 失败
//--------------------------------------------------
```

#### 14.2.3.8 get_sn

指令原型：int get_sn(int Card, char *Sn, int SnSize int *RtnNum)

```
//--------------------------------------------------
// 功能描述：获取指定板卡的SN号
// 输入参数：Card ----卡号
             SnSize ---- Sn 数组长度，SnSize>12
// 输出参数：
        Sn---返回的SN 号信息。使用时需要定义一个长度大于12的char 类型数组作为该参数;
        RtnNum---- 返回的SN 长度。
// 返回值：=0 - 成功，  =-1 - 失败
//--------------------------------------------------
```

## 14.3. 参数设置查询

### 14.3.1应用说明

在控制卡正常初始化以后，调用本章节的函数对设置过的单轴常速度，单轴变速度，矢量常速度，矢量变速度，专用输出口和通用输出的滤波参数进行读取。

### 14.3.2调用示例

**获取指定轴的设置速度**

```
    auto_set(); //检测控制器
    init_board(); //初始化控制器
    double vl = 0, vh = 0, ad = 0, dc = 0;
    get_profile(1, &vl, &vh, &ad, &dc);//获取1轴的各速度参数
```

### 14.3.3指令详解

#### 14.3.3.1 get_profile

指令原型：int get_profile(int ch, double* vl, double* vh, double* ad, double* dc);

```
//--------------------------------------------------
// 功能描述：  获取设定的单轴变速参数
// 输入参数：无
// 输出参数：
    Axis ---- 轴号
    Vl ---- 低速
    Vh ---- 高速
    Ad ---- 上升加速度
    Dc ---- 下降加速度
// 返回值：=0 - 成功，  =-1 - 失败
//--------------------------------------------------
```

#### 14.3.3.2 get_vector_conspeed

指令原型：double get_vector_conspeed();

```
//-------------------------------------------------
// 功能描述：  获取设定的单轴变速参数
// 输入参数：无
// 输出参数：无
// 返回值：>0 - 成功，返回设定的矢量常速  =-1 - 失败
//-------------------------------------------------
```

#### 14.3.3.3 get_vector_profile

指令原型：int get_vector_profile(double *VecVl, double *VecVh, double *VecAd, double *VecDc));

```
//-------------------------------------------------
// 功能描述：  获取设定的矢量常速参数
// 输入参数：无
// 输出参数：
     VecVl ---- 低速
     VecVh ---- 高速
     VecAd ---- 上升加速度
     VecDc ---- 下降加速度
// 返回值：=0 - 成功  =-1 - 失败
//-------------------------------------------------
```

#### 14.3.3.4 check_IC

指令原型：int check_IC(int Card);

```
//-------------------------------------------------
// 功能描述：  获取拨码号码。运动控制器上设计了一个拨码开关，可以设定多块板卡共用时各板卡的拨码号。
// 输入参数：
      Card ---- 卡号
// 输出参数：无
// 返回值：>0 - 成功，返回拨码号码  =-1 - 失败
//-------------------------------------------------
```

#### 14.3.3.5 read_input_filter

指令原型：int read_input_filter(int Card,int BitNo, int* Time);

```
//-------------------------------------------------
// 功能描述：  读取通用输入口的滤波时间。默认值为1000us。
// 输入参数：  无
// 输出参数：
     Card---- 卡号
     BitNo ----通用输入口位标记，范围 1 ~ 32
     Time ----读取的滤波时间，单位：us，范围0~65000
// 返回值：=0  -成功 ，  =-1 - 失败
//-------------------------------------------------
```

#### 14.3.3.6 read_special_input_filter

指令原型：int read_special_input_filter(int Card,int BitNo, int* Time);

```
//-------------------------------------------------
// 功能描述：  读取专用输入口的滤波时间。默认值为1000us。
// 输入参数：  无
// 输出参数：
     Card---- 卡号
     BitNo ----通用输入口位标记，范围 1 ~ 32
     Time ----读取的滤波时间，单位：us，范围0~65000
// 返回值：=0  -成功 ，  =-1 - 失败
```

```
//-------------------------------------------
```

# 14.4. 错误信息查询

## 14.4.1应用说明

P1 系列控制卡提供有错误信息查询指令，可以查询记录系统运行中报错的状况。软件工程师在程序编制调试过程中，可以通过调用错误信息查询指令，通过返回的错误代码，能准确了解软件编制的错误，方便软件调试修改。在系统软件运行中通过记录软件操作错误代码，可以方便设备调试人员和操作人员了解设备运行过程中出现的故障情况。

**获取错误代码**

```
int data;
int mRtn = get_err(5, &data);//获取最近第5次产生的错误代码
mRtn = reset_err();//清除控制卡的错误信息
```

## 14.4.2调用示例

## 14.4.3指令详解

**14.4.3.1 get_err**

指令原型：int  get_err(int index, int* data);

```
//-------------------------------------------------
// 功能描述：获取控制卡最近10条错误的信息
// 输入参数：
//      index  - 错误索引号，取值：1~10
// 输出参数：
//      data   - 存储的错误代码
// 返回值：0-成功，-1-index错误或系统没有错误
//-------------------------------------------------
```

get_err 错误代码：初始化错误说明

| 错误代码 | 对应 Error Code | 含义 |
|---|---|---|
| 0x00000004 | ERR_AUTO_SET_NOT | 初始化未调用 auto_set() |
| 0x00000006 | ERR_WDM_COMM | 与底层通信失败：未安装驱动或驱动未正确加载 |
| 0x00000007 | ERR_INIT_NO_BOARD | 检测不到板卡 |
| 0x00000008 | ERR_INIT_NO_AXES | 板卡上检测不到轴 |
| 0x00000009 | ERR_INIT_BOARD_NOT | 未进行初始化或初始化未成功 |
| 0x0000000a | ERR_INIT_TOO_BOARD | 检测到的板卡数目超过允许的卡数；P1 系列默认支持 5 卡共用 |
| 0x0000000b | ERR_INIT_TOO_AXES_CARD | 检测到板卡上的轴数超过板卡设计的轴数；MPC1400/3400 每卡 4 轴；MPC1800/3800 每卡 8 轴。 |
| 0x0000000c | ERR_INIT_NO_NAMECARD | 错误的板卡 ID 号 |
| 0x00000012 | ERR_INIT_SYS | 驱动程序版本错误 |
| 0x00000013 | ERR_INIT_CARD_TYPE | 板卡类型出错 |
| 0x00000019 | ERR_INIT_CONFIG | 加载配置出错 |

| get_err 错误代码：API 参数错误 | | |
|---|---|---|
| 错误代码 | 对应 Error Code | 含义 |
| 0x02000019 | ERR_PARAM_VALUE | 参数设置错误 |
| 0x0200001e | ERR_BEFOREMOTION | 指令冲突或不支持该功能 |

#### 14.4.3.2 get_last_err

指令原型：int  get_last_err();

```
//-------------------------------------------------
// 功能描述：获取控制卡最进1条错误的信息
// 输入参数：无
// 输出参数：无
// 返回值：0-没有错误，Error Code-成功
//-------------------------------------------------
```

#### 14.4.3.3 reset_err

指令原型：int  reset_err();

```
//-------------------------------------------------
// 功能描述：清除控制卡的错误信息
// 输入参数：无
// 输出参数：无
// 返回值：0-成功，-1-失败
//-------------------------------------------------
```

## 14.5. 自定义存储区操作

### 14.5.1应用说明

P1 系列控制卡提供 1M bytes 掉电数据保护存储空间；其存储空间包括两部分：可加密存储空间和非加密存储空间。

| 1M bytes | 逻辑片区 | 逻辑地址 |
|---|---|---|
| 数据存储空间（非加密） | 15（64K bytes） | 序号：0~16383（共 16K） |
| | 14（64K bytes） | 序号：0~16383（共 16K） |
| | 13（64K bytes） | 序号：0~16383（共 16K） |
| | 12（64K bytes） | 序号：0~16383（共 16K） |
| | 11（64K bytes） | 序号：0~16383（共 16K） |
| | 10（64K bytes） | 序号：0~16383（共 16K） |
| | 9（64K bytes） | 序号：0~16383（共 16K） |
| | 8（64K bytes） | 序号：0~16383（共 16K） |
| | 7（64K bytes） | 序号：0~16383（共 16K） |
| | 6（64K bytes） | 序号：0~16383（共 16K） |
| | 5（64K bytes） | 序号：0~16383（共 16K） |
| | 4（64K bytes） | 序号：0~16383（共 16K） |
| | 3（64K bytes） | 序号：0~16383（共 16K） |
| | 2（64K bytes） | 序号：0~16383（共 16K） |
| | 1（64K bytes） | 序号：0~16383（共 16K） |
| 加密存储空间 | 0（64K bytes） | 序号：0~16383（共 16K） |

应用场景：

1，设备应用软件中有些特殊的工艺数据参数没有对设备终端使用商开放，软件工程师在程序编制过程中，通过调用 **write_password_flash** 等指令，将特殊工艺数据存储在板卡加密区里，用来进行技术保护。

2，有些设备厂商，为了保证自身的利益。会将设备应用软件和设备控制卡进行绑定处理，软件工程师在程序编制过程中，通过调用指令 **write_password_flash**，将软件启动的密码存储在板卡加密区里，只有正确输入启动密码设备才能正常工作。

3，当某种类型设备有些常用固定的工艺数据参数，软件工程师在程序编制过程中，通过调用 **write_flash** 等指令，将设备固定工艺数据存储在数据区，当每次应用软件需要使用时可以直接使用，不用每次输入。

## 14.5.2调用示例

**加密存储区数据操作**

```
long data;
clear_password_flash(1, 0xffffffff);//清除加密存储区设置的密码
write_password_flash(1, 16383, 0xccccaaaa, 0xffffffff);//将读写密码修改为 0xccccaaaa
write_password_flash(1, 1, 0x01, 0xccccaaaa);  //对区域 0 的 1 号地址写入数据 0x01
read_password_flash(1, 1, &data, 0xccccaaaa);  //读取区域0的1号地址所保存的数值
```

注意：密码片区中**第 16383 逻辑地址用来存储密码**，出厂密码为 0xffffffff。对密码片区进行读写操作需要提供其第 **16383** 号地址存储的 long 型密码。

## 14.5.3指令详解

### 14.5.3.1 write_password_flash

指令原型：int  write_password_flash(int cardno, int no, long data, long password);

```
//-------------------------------------------------
// 功能描述：设置控制卡加密用户通用存储区的密码
// 输入参数：
//      cardno  - 卡号，取值：1~4
//      no      - 地址编号，取值：1~16383
//      data    - 写入的数据
//      password- 校验密码（16383号逻辑地址）
// 输出参数：无
// 返回值：0-正确，-1-写入错误参数，-2-校验密码出错
//-------------------------------------------------
```

### 14.5.3.2 read_password_flash

指令原型：int  read_password_flash(int cardno, int no, long* data, long password);

```
//-------------------------------------------------
// 功能描述：获取控制卡加密用户通用存储区的密码
//      cardno  - 卡号，取值：1~4
//      no      - 地址编号，取值：1~16383
//      password- 校验密码（16383号逻辑地址）
// 输出参数：
//      data    - 保存指定地址内的数据
// 返回值：0-正确，-1-参数错误，-2-校验密码出错
//-------------------------------------------------
```

#### 14.5.3.3 clear_password_flash

指令原型：int　clear_password_flash(int cardno, long password);

```
//-------------------------------------------------
// 功能描述：清除控制卡加密用户通用存储区设置的密码
// 输入参数：
//      cardno  - 卡号，取值：1~4
//      password- 校验密码
// 输出参数：无
// 返回值：0-正确，-1-参数错误，-2-校验密码出错
//-------------------------------------------------
```

#### 14.5.3.4 write_flash

指令原型：int write_flash(int cardno,int piece, int no, long data);

```
//-------------------------------------------------
// 功能描述：写控制卡非加密通用存储区
// 输入参数：
//      cardno  - 卡号，取值：1~4
//      piece   - 片区号，取值：1-15
//      no      - 地址编号，取值：1~16383
//      data    - 写入的数据
// 输出参数：无
// 返回值：0-正确，-1-参数错误
//-------------------------------------------------
```

#### 14.5.3.5 read_flash

指令原型：int read_flash(int cardno,int piece,int no,long *data );

```
//-------------------------------------------------
// 功能描述：读控制卡非加密通用存储区
// 输入参数：
//      cardno  - 卡号，取值：1~4
//      piece   - 片区号，取值：1-15
//      no      - 地址编号，取值：1~16383
// 输出参数：
//      data    - 存储读出的数据
// 返回值：0-正确，-1-参数错误
//-------------------------------------------------
```

#### 14.5.3.6 clear_flash

指令原型：int clear_flash(int cardno,int piece );

```
//-------------------------------------------------
// 功能描述：读控制卡非加密通用存储区
// 输入参数：
//      cardno  - 卡号，取值：1~4
//      piece   - 片区号，取值：1-15
// 输出参数：无
// 返回值：0-正确，-1-参数错误
//-------------------------------------------------
```

# 14.6. 其他

## 14.6.1应用说明

P1 系列控制卡指令接口兼容 MPC 系列控制卡，其中该章节指令在 P1 系列控制卡指令中进行保留，方便用户可有效继承由 MPC 系列控制卡编写的原代码工程。

## 14.6.2指令详解

### 14.6.2.1 set_dir

指令原型：int  set_dir(int Axis, int Dir);

```
//-------------------------------------------------
// 功能描述：设置轴脉冲信号方向
// 输入参数：
//      Axis  - 轴号，取值：1~32
//      Dir   - 方向，取值：1-正方向，-1-负方向
// 输出参数：无
// 返回值：0-成功，-1-失败
//-------------------------------------------------
```

### 14.6.2.2 outport_byte_ex

指令原型：int  outport_byte_ex(int Card, int Data);

```
//-------------------------------------------------
// 功能描述：设置扩展卡上的通用输出口状态
// 输入参数：
//      Card  - 卡号，取值：1~4
//      Data  - 扩展卡通用输出口状态，某位为0表示输出低电平，为1表示无输出
// 输出参数：
// 返回值：0-成功，-1-失败
//-------------------------------------------------
```

### 14.6.2.3 check_exoutport_status

指令原型：int  check_exoutport_status(int Card);

```
//-------------------------------------------------
// 功能描述：获取扩展卡通用输出口的状态
// 输入参数：
//      Card  - 卡号，取值：1~4
// 输出参数：无
// 返回值：非负（通用输出口状态）-成功，-1-失败
//-------------------------------------------------
```