leetro

MPC08D 运动控制器

编

程

手

册

（0.3 版）

乐创自动化技术有限公司
LEETRO AUTOMATION CO.,LTD.

版权申明

前言

感谢购买 MPC08D 运动控制器！MPC08D 是本公司研制的一款高性价比通用控制器。本编程手册描述 MPC08D 运动指令的使用。使用前请充分理解 MPC08D 的使用功能。

安全警告

注意以下警告，以免伤害操作人员及其他人员，防止机器损坏。

◆ 下面的“危险”和“警告”符号是按照其事故危险的程度来标出的。

 危 险	指示一个潜在的危险情况，如果不避免，将导致死亡或严重伤害。

 警 告	指示一个潜在的危险情况，如果不避免，将导致轻度或中度伤害，或物质损坏。

◆ 下列符号指示哪些是禁止的，或哪些是必须遵守的。

	这个符号表示禁止操作。

	这个符号表示须注意的操作。

常规安全概要

请查看下列安全防范措施以避免受伤害并防止对本产品或任何与其相连接的产品造成损伤。为避免潜在的危险，请按详细说明来使用本产品。

使用正确的电源线。请使用满足国家标准的电源线。

正确地连接和断开。先将控制卡输出连接至转接板，再将电机、驱动器连接到转接板，最后开启电源。断开时先关闭外部电源，再断开电机、驱动器与转接板的连接，最后断开控制卡与转接板的连接。

当有可疑的故障时不要进行操作。如果您怀疑本产品有损伤，请让有资格的服务人员进行检查。

不要在湿的/潮湿环境下操作。

不要在爆炸性的空气中操作。

保持产品表面清洁和干燥。

防止静电损伤。静电释放（ESD）可能会对运动控制器及其附件中的元件造成损伤。为了防止 ESD，请小心处理控制器元件，不要触摸控制器上元器件。不要将控制器放置在可能产生静电的表面。在防护静电的袋子或容器内运输和储存控制器。

关于保证

保修时间

在指定的地点购买的产品的保修期为 1 年。

保修范围

如果在上述质保期内由于本公司责任发生了故障，本公司提供无偿修理。

以下范围不在保修范围内：

- 对于说明书及其它手册记录的不适当环境或不适当使用引起的故障。

- 用户的装置、控制软件等引起本产品意外故障。
- 由客户对本产品的改造引起的故障。
- 火灾、地震及其它自然灾害等外部主要原因引起的故障。

产品的应用范围

本产品设计制造用于普通工业应用，超出预料的用途并对人的生命或财产造成重大的影响不在产品服务范围。

联系信息

通信地址：四川省成都市高新区冯家湾科园南二路 1 号大一孵化园 8 幢 B 座（610041）

成都乐创自动化技术股份有限公司

公司网站：htttp://www.leetro.com

技术支持：

- ✧ Tel：　（028）85149977
- ✧ FAX：　（028）85187774

目　　录

1 函数库的使用

1.1 开发 Windows 系统下的运动控制系统

利用 MPC08D 的动态链接库（DLL），开发者可以很快开发出 Windows 平台下的运动控制系统。MPC08D 动态链接库是标准的 Windows 32 位动态链接库，选用的开发工具应支持 Windows 标准的 32 位 DLL 调用。

运行产品配套光盘中的安装程序后，将在安装目录（默认安装目录为\Program Files）下自动生成“MPC08D”文件夹，其目录树如下图所示：

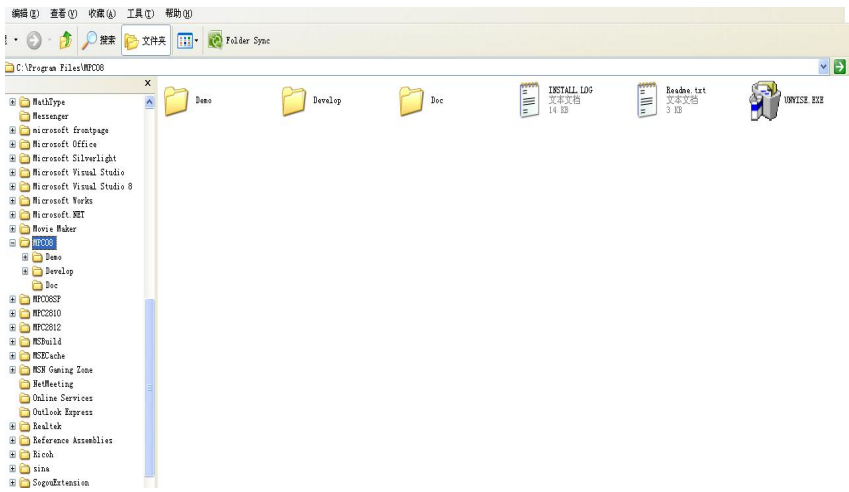

图 1-1 MPC08D 目录结构

（1）“Demo”目录中是示例程序，其中：

- “VBDemo”目录下包含“Demo1”和“Demo2”是两个 VB 示例，提供了源代码；
- “VCDemo”目录下包含 4 个示例程序，其中“Demo1”和“Demo2”提供了源代码，“Demo1”是 VC 静态加载动态链接库示例，“Demo2”是 VC 动态加载动态链接库示例。“Demo3”未提供源代码，具有函数测试功能。“Demo4”未提供源代码，具有函数库、驱动程序和各张板卡的固件版本读取功能。

（2）“Develop”目录中包含 MPC08D 的驱动程序和函数库，其中：

- “Common”文件夹中是 MPC08D 的驱动程序、函数库等；

- “VB”文件夹中是开发 VB 应用程序时需要加入的模块文件；
- “VC”文件夹中是动态加载动态链接库需要使用的文件：“LoadDll.cpp”和“LoadDll.h”，以及静态加载动态链接库时需要使用的文件“MPC08D.h”和“MPCC08D.lib”。

（3）“Doc”目录中包含 MPC08D 的用户手册和编程手册。

用户编写的运动过程处理典型流程如图 1-2 所示。

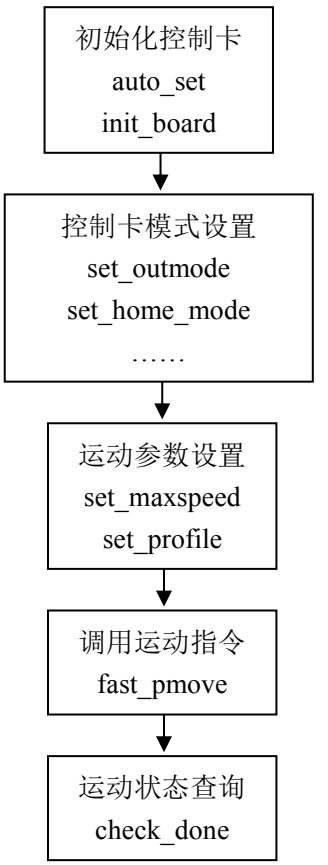

图 1-2 运动指令典型流程图

以下介绍如何利用两种常用的开发工具 Microsoft Visual Basic 和 Microsoft Visual C++开发基于 Windows 平台的运动控制程序。

1.1.1 开发 Visual Basic 控制程序

（一） 概述

为了开发基于 Windows 的运动控制程序，用户可以使用 VB5.0 或更高

版本。按照如下步骤可以快速开发一个简单的控制程序。

1．安装 MPC08D 驱动程序及函数库；

2．启动 Visual Basic，新建一个工程；

3. 将安装好的动态链接库“MPC08D.dll”和函数声明文件“MPC08D.bas”复制到工程文件中；

4．选择“Project”菜单下的“Add Module”菜单项，将“MPC08D.bas”文件添加到工程中；

5．在应用程序中调用运动函数。

（二） 动态链接库函数调用方法

在 VB 中调用动态链接库（DLL）中函数应包括两部分工作：

（1）函数声明

每一个动态链接库（DLL）中的函数在 VB 中的声明已经包含在 MPC08D.bas 文件中了，该文件可在 MPC08D 运动控制器软件安装盘 \Develop\VB 文件夹下找到，用户只需要将该文件添加进 VB 工程中即可。

（2）函数调用

若调用函数的返回值为空或不需要返回值，则按如下方法调用：

```
con_pmove 1，2000
```

或

```
call con_pmove (1,2000)
```

若要得到函数的返回值，则按如下方法调用：

```
Dim rtn As Long
rtn=con_pmove(1,2000)
```

（三）演示示例程序的使用

MPC08D 运动控制器软件安装盘\Demo\VBDemo 文件夹下有两个在 VB6.0 下开发的运动控制系统演示示例程序。用户可按照如下步骤编译并运行该示例，在熟悉了相应编程方法后，用户可根据需要开发自己的运动控制系统。

（1） 按照 MPC08D 软件的安装步骤进行正确安装。

（2） 安装完成后，可在安装盘\Demo\VBDemo\文件夹中下找到 Demo1 或 Demo2 文件夹。

（3） 启动 VB6.0 集成环境，并打开工程。

（4） 确保板卡已经正确设置并插入到计算机中。

（5） 编译该工程生成 EXE 文件。

（6） 运行生成的 EXE 文件。

1.1.2 开发 Visual C++控制程序

（一）概述

用户可以使用 VC5.0 或更高版本，来进行 Windows 平台下运动控制系统开发。Visual C++有两种调用动态链接库的方法，使用的文件略有不同。

（二）动态链接库函数调用方法

（1）隐式调用

隐式调用步骤如下：

（a）安装 MPC08D 驱动程序及函数库；

（b）启动 Visual C++，新建一个工程；

（c）将安装好的动态链接库“MPC08D.dll”、“MPC08D.lib”和函数声明文件“MPC08D.h”复制到工程文件中；

（d）选择“Project”菜单下的“Settings”菜单项；

（e）切换到“Link”标签页，在“Object/library modules”栏中输入“MPC08D.lib”文件名；

（f）在应用程序中加入函数库头文件的声明文件“MPC08D.h”；

（g）在应用程序中调用运动函数。

具体可参见演示示例：\Demo\VCDemo\Demo1。

（2）显式调用

显式调用方法需要调用 Windows API 函数加载和释放动态链接库。方法如下：

（a）调用 Windows API 函数 LoadLibrary()动态加载 DLL；

（b）调用 Windows API 函数 GetProcAddress()取得将要调用的 DLL 中函数的指针；

（c）用函数指针调用 DLL 中函数完成相应功能；

（e）在程序结束时或不再使用 DLL 中函数时，调用 Windows API 函数 FreeLibrary()释放动态链接库。

该方法比较繁琐。MPC08D 软件中已经将常用的 DLL 函数封装成类 CLoadDll，并提供该类的源代码。该类含有与运动指令库函数名及参数相同的成员函数。源代码可在 MPC08D 安装目录“\Develop\VC”文件夹下找到，文件名为 LoadDll.cpp 和 LoadDll.h。开发人员可将其添加进工程，在程序适当地方添加该类的对象，通过对应成员函数来调用 DLL 中的函数。具体可参见演示示例：\Demo\VCDemo\Demo2。

调用步骤如下：

（a）安装 MPC08D 驱动程序及函数库；

（b）启动 Visual C++，新建一个工程；

（c）将安装好的动态链接库“MPC08D.dll”、“MPC08D.lib”和函数声明文件“LoadDll.h”、“LoadDll.cpp”复制到工程文件中；

（d）选择“Project”菜单下的“Add To Project”的子菜单“Files”项；

（e）将“LoadDll.h”、“LoadDll.cpp”加入到工程中；

（f）在应用程序中生成 CLoadDll 的一个对象，调用运动函数。

以上在两种方法均为 VC 中调用动态链接库函数的标准方法，若要获得更具体的调用方法和帮助，请参考微软 Visual Stutio 开发文档 MSDN 或相关 VC 参考书籍中相应部分内容。如果需要将 MPC08D 与 mpc2810 或 mpc08B 控制器共用的时候，一般选用显式调用。

（三）演示示例程序的使用

MPC08D 运动控制器软件安装盘\Demo\VCDemo\文件夹下包含四个示例程序，其中“Demo1”和“Demo2” 提供了源代码，“Demo1”是 VC 静态加载动态链接库示例，“Demo2”是 VC 动态加载动态链接库示例。“Demo3”未提供源代码，具有函数测试功能。“Demo4”未提供源代码，具有函数库、驱动程序和各张板卡的固件版本读取功能。用户可按照如下步骤编译并运行示例，在熟悉了相应编程方法后，用户可根据需要开发自己的运动控制系统。使用步骤如下：

（1）按照 MPC08D 软件的安装步骤进行正确安装。

（2）安装完成后，可在安装目录“\Demo\VCDemo\”文件夹中下找到相应文件夹。

（3）启动 VC6.0 集成环境，并打开工程文件（demo1.dsw、demo2.dsw、.....等）。

（4）确保板卡已经正确设置并插入到计算机中。

（5）编译连接该工程生成 EXE 文件。

（6）运行生成的 EXE 文件。

1.2 MPC08D 软件升级

请您经常访问本公司的网站（http://www.leetro.com）以下载获取最新版本的驱动程序及函数库，新版本函数库将会保持与旧版函数库已有函数的兼容，并根据需要增加新的函数。升级前请先咨询公司经销商或技术支持部。

若您获得一套最新的安装程序，您可以按照以下方法对您的旧函数库进行升级：

（1）关闭与 MPC08D 相关的正在运行的所有程序；

（2）卸载原来的安装程序；

（3）运行新的安装程序；

（4）若使用 Visual Basic6.0 开发，将安装好的动态链接库“MPC08D.dll”和函数声明文件“MPC08D.bas”复制到工程文件中，重新编译生成.EXE 文件。

（5）若使用 Visual C++6.0 开发，隐式调用时，将安装好的动态链接库“MPC08D.dll”、“MPC08D.lib”和函数声明文件“MPC08D.h”复制到工程文件中，重新编译生成.EXE 文件。显式调用时，将安装好的动态链接库“MPC08D.dll”、“MPC08D.lib”和函数声明文件“LoadDll.h”、“LoadDll.cpp”复制到工程文件中重新编译生成.EXE 文件。

2 运动控制器初始化

2.1 控制器初始化

2.1.1 指令列表

控制器初始化函数有 2 个，若要使用 MPC08D 各项功能，必须首先依次调用函数 auto_set、init_board，完成控制器初始化后才能调用其它函数。初始化函数如表 2-1 所示。

表 2-1 控制器初始化函数

函数	返回值	说明
auto_set	≥0：轴数 -1：错误	自动检测和自动设置控制器
init_board	≥0：卡数 -1：错误	对控制器硬件和软件初始化

2.1.2 功能说明

（1）auto_set 说明

首先必须调用 auto_set 函数自动检测控制器，执行正确返回卡上轴数。检测用户设置的板卡号是否正确。

（2）init_board 说明

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。执行正确返回计算机内安装的控制器卡数。初始化后各控制轴初始状态为：

- 脉冲输出模式：脉冲/方向；
- 常速度：2000pps；
- 梯形速度：初速 2000pps、高速 8000pps、加减速 80000ppss；
- 矢量常速度：2000pps；
- 矢量梯形速度：初速 2000pps，高速 8000pps，加减速 80000ppss；
- 轴回原点运动模式：仅检测原点接近开关信号，有原点信号立即停止运动；
- 各轴正限位、负限位、原点及报警使能为有效状态，高电平有效。

板卡报警信号使能为有效状态，高电平有效。

2.1.3 举例

```
InitMPC08D()
{
    int Rtn;
    Rtn = auto_set();              //自动设置
    if(Rtn <= 0 )
    {
        返回错误代码
    }
    else
    {
        自动设置成功，获得轴数
    }

    Rtn = init_board();            //初始化板卡
    if(Rtn <= 0 )
    {
        返回错误代码
    }
    else
    {
        初始化成功，获得卡数
    }
    ……
}
```

2.2 控制轴初始化

2.2.1 指令列表

控制轴初始化函数有 2 个，如表 2-2 所示。

表 2-2 控制轴初始化函数

函数	返回值	说明
set_outmode	0：正确	设置轴的脉冲输出模式

	-1：错误	
set_home_mode	0：正确 -1：错误	设置轴的回原点模式

2.2.2 功能说明

（1）set_outmode 说明

运动控制器提供了两种脉冲输出模式：“脉冲+方向”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。

该函数在在 init_board 函数后调用。

（2）set_home_mode 说明

运动控制器提供四种回原点模式：

0：检测到原点接近开关信号轴立即停止运动；

1：检测到出现编码器 Z 相脉冲信号时立即停止运动；

2：梯形速度模式下，检测到原点接近开关信号有效时控制轴按快速运动方式设置的加速度逐渐减速停止；

3：梯形速度模式下，原点信号有效时，控制轴按快速运动方式设置的加速度逐渐减速至低速，直到 Z 脉冲有效立即停止运动。

注意，在回原点后，应调用位置复位函数“reset_pos“将控制轴当前位置设置为原点坐标。

该函数在在 init_board 函数后调用。

2.3 专用输入信号设置

2.3.1 指令列表

专用输入信号参数设置相关函数有 8 个，如下表所示。这些函数一般在 init_board 函数后调用，初始化控制系统专用输入功能。

表 2-3 专用输入信号参数设置函数

函数	返回值	说明
enable_alm	0：正确 -1：错误	使能/禁止轴的外部报警信号（默认为使能）
enable_el	0：正确 -1：错误	使能/禁止轴的外部限位信号（默认为使能）

enable_org	0：正确 -1：错误	使能/禁止轴的外部原点信号（默认为使能）
enable_card_alm	0：正确 -1：错误	使能/禁止板卡外部报警信号（默认为使能）
set_alm_logic	0：正确 -1：错误	设置轴的外部报警信号有效电平（默认为高电平有效）
set_el_logic	0：正确 -1：错误	设置轴的外部限位信号有效电平（默认为高电平有效）
set_org_logic	0：正确 -1：错误	设置轴的外部原点信号有效电平（默认为高电平有效）
set_card_alm_logic	0：正确 -1：错误	设置板卡外部报警信号有效电平（默认为高电平有效）

2.3.2 功能说明

（1）enable_alm 说明

运动控制器为每个控制轴提供了报警信号输入接口，该信号只对该轴的运动有效。默认情况下，轴报警开关处于使能状态，高电平有效。这时各轴报警开关应与转接板上相应轴报警开关输入引脚和OGND（对应转接板DB9端子的“ALM”引脚）相连。其接线方式请参见《MPC08D User.doc》中的“专用输入的连接方法”一节。当某轴的报警信号被触发时，运动控制器将自动停止该轴的运动。

若控制系统不使用轴报警信号，调用“enable_alm”指令，将相应参数设置为0，可将该引脚设置作为通用输入口使用。系统将其状态保存在一个专用寄存器中。这时，通过指令“check_sfr_bit”获取寄存器各位状态。

表 2-4 专用输入信号寄存器中轴报警信号位置

状态位	专用信号	说明
18	ALM4	ALM 4 无效时作为通用输入口
13	ALM3	ALM 3 无效时作为通用输入口
8	ALM2	ALM 2 无效时作为通用输入口
3	ALM1	ALM 1 无效时作为通用输入口

（2）enable_el 说明

运动控制器为每个控制轴提供了正负限位信号输入接口。默认情况下，限位开关处于使能状态，高电平有效。这时各轴限位开关应与转接板上相应限位信号输入引脚和 OGND（对应转接板 EL+/EL-引脚）相连。其接线方式

请参见《MPC08D User.doc》中的“专用输入的连接方法”一节。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。

若控制系统不使用限位信号，调用“enable_el”指令，将相应参数设置为 0，可将该引脚设置作为通用输入口使用。系统将其状态保存在一个专用寄存器中。这时，通过指令“check_sfr_bit”获取寄存器各位状态。下表为限位信号在专用输入寄存器的对应关系。

表 2-5 专用输入信号寄存器中限位信号位置

状态位	专用信号	说明
16	EL4-	EL4-无效时作为通用输入口
15	EL4+	EL4+无效时作为通用输入口
11	EL3-	EL3-无效时作为通用输入口
10	EL3+	EL3+无效时作为通用输入口
6	EL2-	EL2-无效时作为通用输入口
5	EL2+	EL2+无效时作为通用输入口
1	EL1-	EL1-无效时作为通用输入口
0	EL1+	EL1+无效时作为通用输入口

（3）enable_org 说明

运动控制器为每个控制轴提供了原点信号输入接口。默认情况下，原点开关处于使能状态，高电平有效。这时各轴原点开关应与转接板上相应原点信号输入引脚和 OGND（对应转接板 ORG 引脚）相连。其接线方式请参见《MPC08D User.doc》中的“专用输入的连接方法”一节。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。

若控制系统不使用原点信号，调用“enable_org”指令，将相应参数设置为 0，可将该引脚设置作为通用输入口使用。系统将其状态保存在一个专用寄存器中。这时，通过指令“check_sfr_bit”获取寄存器各位状态。下表为原点信号在专用输入寄存器的对应关系。

表 2-6 专用输入信号寄存器中原点信号位置

状态位	专用信号	说明
17	ORG4	ORG4 无效时作为通用输入口
12	ORG3	ORG3 无效时作为通用输入口
7	ORG2	ORG2 无效时作为通用输入口
2	ORG1	ORG1 无效时作为通用输入口

（4）enable_card_alm 说明

运动控制器提供了一个板卡报警信号输入接口。默认情况下，报警开关处于使能状态，高电平有效。这时报警开关应与转接板上相应报警信号输入引脚和 OGND（对应转接板 ALM 引脚）相连。其接线方式请参见《MPC08D User.doc》中的“专用输入的连接方法”一节。当运动控制器处于运动状态时，若板卡报警开关触发，运动控制器将自动停止所有轴的运动。

若控制系统不使用板卡报警信号，调用“enable_card_alm”指令，将相应参数设置为 0，可将该引脚设置作为通用输入口使用。系统将其状态保存在一个专用寄存器中。这时，通过指令“check_sfr_bit”获取寄存器各位状态。下表为报警信号在专用输入寄存器的对应关系。

表 2-7 专用输入信号寄存器中板卡报警信号位置

状态位	专用信号	说明
20	ALM	板卡 ALM 无效时作为通用输入口

（5）set_alm_logic 说明

运动控制器默认的轴报警开关为常闭开关，即各轴处于正常工作状态时，其轴报警开关输入为低电平；当轴报警开关输入为高电平时，将触发对应轴的报警状态。

通过指令“***set_alm_logic***”的参数可设置轴报警开关的触发电平，将相应参数设置为 1，表示对应轴的报警开关设置为高电平触发；参数设置为 0 表示对应轴的报警开关为低电平触发。

（6）set_el_logic 说明

运动控制器默认的限位开关为常闭开关，即各轴处于正常工作状态时，其限位开关输入为低电平；当限位开关输入为高电平时，将触发对应轴的限位状态。

通过指令“set_el_logic”的参数可设置限位开关的触发电平，将相应参数设置为 1，表示对应轴的限位开关设置为高电平触发；参数设置为 0 表示对应轴的限位开关为低电平触发。

（7）set_org_logic 说明

运动控制器默认的原点开关为常闭开关，即各轴处于正常工作状态时，其原点开关输入为低电平；当原点开关输入为高电平时，将触发对应轴的原点状态。

通过指令“set_org_logic”的参数可设置原点开关的触发电平，将相应参数设置为 1，表示对应轴的原点开关设置为高电平触发；参数设置为 0 表示对应轴的原点开关为低电平触发。

（8）set_card_alm_logic 说明

运动控制器默认的板卡报警开关为常闭开关，即板卡处于正常工作状态时，其报警开关输入为低电平；当板卡报警开关输入为高电平时，将触发对应板卡的报警状态。

通过指令“set_card_alm_logic”的参数可设置板卡报警开关的触发电平，将相应参数设置为 1，表示对应板卡的报警开关设置为高电平触发；参数设置为 0 表示对应板卡的报警开关为低电平触发。

3 运动函数

3.1 速度设置函数

3.1.1 指令列表

速度设置函数有 5 个，如表 3-1 所示。这些函数中速度、加速度和减速度参数均以脉冲为编程单位。

表 3-1 速度设置函数

函数	返回值	说明
set_maxspeed	0：正确 -1：错误	设置轴的最大速度
set_conspeed	0：正确 -1：错误	设置轴在常速运动方式下的速度参数
set_profile	0：正确 -1：错误	设定轴在快速运动方式下的速度参数
set_vector_conspeed	0：正确 -1：错误	设置常速运动方式下的矢量常速度参数
set_vector_profile	0：正确 -1：错误	设置快速运动方式下的矢量梯形速度参数

3.1.2 功能说明

（1）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。MPC08D 运动控制器的输出脉冲频率由两个变量控制：脉冲分辨率和倍率，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 13 位（最大值为 8191），如果要达到 2400KHz 的最大输出脉冲频率，脉冲分辨率为（2400000/8191）=293Hz，如果实际使用的脉冲频率为 100Hz，显然 MPC08D 器只能输出一个分辨率的脉冲频率（即 293Hz）。为了解决这个问题，调用 set_maxspeed 设置需要达到的最大输出脉冲频率，如 set_maxspeed（1，100）。设置后脉冲分辨率将被重新设置，为（100/8191）=0.012Hz，这样就能提高速度分辨率。但是，此时运动控制器的最大输出脉冲频率只能达到 100Hz。注意：MPC08D 的最高分辨率为可以达到 0.01，此时控制器的最大输出脉冲频率只能达到 82Hz。

在缺省情况下，init_board 函数将所有轴设置其最大速度，即 2000000

（2MHz）。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精度。最大输出脉冲频率范围：81.91~2000000 Hz。

（2）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：

图 3-1 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。这时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：

$$multip = \frac{V_{\max}}{8191} \qquad v_{act} = \text{int}(\frac{v_{con}}{multip}) \times multip$$

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 2000000 Hz，最小脉冲频率可设置为 0.2 Hz。

（3）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（梯型速度模式下，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。

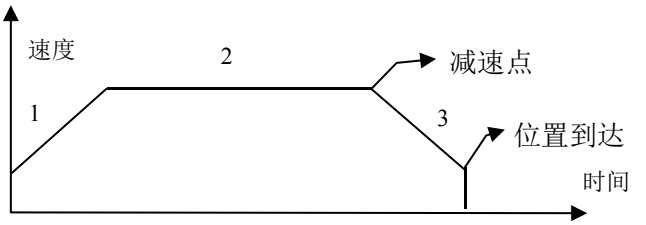

图 3-2 快速运动方式的速度曲线

1：加速段，起始速度按设定的加速度加速到目标速度；

2：高速段，保持目标速度，直至运动到减速点；

3：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。

（4）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 2000000 Hz，最小脉冲频率可设置为 1Hz。

（5）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（梯型速度模式下，矢量减速度值可以不等于矢量加速度值）。这些矢量值在两轴及以上快速直线插补运动中将会用到。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 2000000Hz，加速值最小可设置为 20。

3.2 点位运动函数

3.2.1 常速运动模式

3.2.1.1 指令列表

常速模式点位运动速度设置及运动函数有 6 个，如表 3-2 所示。

表 3-2 常速点位运动函数

函数	返回值	说明
set_maxspeed	0：正确 -1：错误	设置轴的最大速度
set_conspeed	0：正确 -1：错误	设置轴在常速运动方式下的速度参数
con_pmove	0：正确 -1：错误	1 个轴以常速做相对位置点位运动
con_pmove2	0：正确 -1：错误	2 个轴以常速做相对位置点位运动
con_pmove3	0：正确	3 个轴以常速做相对位置点位运动

	-1：错误	
con_pmove4	0：正确 -1：错误	4 个轴以常速做相对位置点位运动

3.2.1.2 功能说明

MPC08D 运动控制器提供了 4 个常速点位运动函数。点位运动是指各轴按各自设定的速度、加速度和行程运动，直到到达设定位置或调用停止指令使轴停止运动。点位运动函数中行程参数均以脉冲数为编程单位。

（1）con_pmove、con_pmove2、con_pmove3、con_pmove4 说明

分别用于启动一个轴、两个轴、三个轴或四个轴，各轴独立以常速方式作点位运动，指令中的位置参数均指相对位移量。启动运动前先调用“set_maxspeed”设置控制轴需要的最大速度（确定了运动控制器脉冲输出倍率及分辨率），然后调用指令“set_conspeed”设置运动速度。注意：多轴启动时不一定同时到达。

3.2.1.3 例程

```
void main( )
{
    auto_set( );                //检测控制器
    init_board( );              //初始化控制器
    set_maxspeed(1,2000);       //设置 1 轴最大速度为 2000
    set_conspeed(1,2000);       //设置 1 轴常速度为 2000
    con_pmove(1,5000);          //启动 1 轴常速运动，行程 5000
}
```

3.2.2 梯形变速运动模式

3.2.2.1 指令列表

梯形变速点位运动速度设置及运动函数有 7 个，如表 3-3 所示。

表 3-3 梯形速度模式点位运动函数

函数	返回值	说明
set_s_curve	0：正确 -1：错误	设置控制轴快速运动模式
set_maxspeed	0：正确 -1：错误	设置轴的最大速度

set_profile	0：正确 -1：错误	设置快速运动的低速、高速和加速度
fast_pmove	0：正确 -1：错误	1 个轴以快速做相对位置点位运动
fast_pmove2	0：正确 -1：错误	2 个轴以快速做相对位置点位运动
fast_pmove3	0：正确 -1：错误	3 个轴以快速做相对位置点位运动
fast_pmove4	0：正确 -1：错误	4 个轴以快速做相对位置点位运动

3.2.2.2 功能说明

MPC08D 运动控制器提供了 7 个梯形模式点位运动处理函数。点位运动是指各轴按各自设定的速度、加速度和行程运动，直到到达设定位置或调用停止指令使轴停止运动。点位运动函数中行程参数均以脉冲数为编程单位。

（1）set_s_curve 说明

运动控制器为点位运动轴提供了两种快速运动模式：梯形曲线和 S 形曲线，其中梯型加减速模式可以设置为非对称的梯型加减速，即加速度和减速度可以设置为不同的数值。使用“set_s_curve”函数设置某个轴在快速运行时采用哪一种升降速方式。系统默认为梯形速度模式。

（2）fast_pmove、fast_pmove2、fast_pmove3、fast_pmove4 说明

分别用于启动一个轴、两个轴、三个轴或四个轴，各轴独立以快速方式作点位运动，指令中的位置参数均指相对位移量。梯形速度由指令“set_profile”设置。启动运动前先调用“set_maxspeed”设置控制轴需要的最大速度（确定了运动控制器脉冲输出倍率及分辨率）。多轴启动时不一定同时到达。

3.2.2.3 例程

```
void main( )
{
    auto_set( );              //检测控制器
    init_board( );            //初始化控制器
    set_s_curve( 1,0);        //设置梯形速度模式
    set_maxspeed(1,5000);  //设置 1 轴最大速度为 2000
```

```
    set_profile(1,100,5000,3000);//设置 1 轴初速、高速和加速度
    fast_pmove (1,5000);    //启动 1 轴梯形快速运动，运动距离 5000
}
```

3.2.3 S 型曲线变速运动模式

3.2.3.1 指令列表

S 型曲线变速独立运动设置及运动函数如表 3-4 所示。

表 3-4 S 形曲线变速独立运动函数

函数	返回值	说明
set_s_curve	0：正确 -1：错误	设置控制轴快速运动模式
set_s_section	0：正确 -1：错误	设置 S 型升降速的相关速度参数
set_profile	0：正确 -1：错误	设置快速运动的低速、高速和加速度
fast_pmove	0：正确 -1：错误	1 个轴以快速做点位运动

3.2.3.2 功能说明

MPC08D 运动控制器提供了 1 个 S 型快速点位运动函数。点位运动是指各轴按各自设定的速度、加速度和行程运动，直到到达设定位置或调用停止指令使轴停止运动。

（1）set_s_section 说明

“set_profile”设置 S 形的初速、高速和最大加速度。“set_s_section”在“set_profile”之后调用，设置 S 形段的速度变化量，所设置的 S 升速降速变化量不能大于之前设置高速的 1/2。

3.2.3.3 例程

```
void main( )
{
    auto_set( );              //检测控制器
    init_board( );            //初始化控制器
    set_s_curve( 1,1);        //设置 S 形速度模式
    set_profile(1,100,5000,1000);//设置 1 轴初速、高速和最大加速度
```

```
    set_s_section(1,800,800); //设置升降速的 S 段，变化量不能大于高速的 1/2。
    fast_pmove (1,5000);    //启动 1 轴 S 形快速运动，行程为 5000
}
```

3.3 连续运动函数

3.3.1 指令列表

连续运动参数设置和运动函数有如表 3-5 所示。

表 3-5 连续运动函数

函数	返回值	说明
set_maxspeed	0：正确 -1：错误	设置轴运动的最大速度
set_profile	0：正确 -1：错误	设置轴快速运动的参数
set_conspeed	0：正确 -1：错误	设置轴常速运动的参数
con_vmove	0：正确 -1：错误	1 个轴以常速做连续运动
con_vmove2	0：正确 -1：错误	2 个轴以常速做连续运动
con_vmove3	0：正确 -1：错误	3 个轴以常速做连续运动
con_vmove4	0：正确 -1：错误	4 个轴以常速做连续运动
fast_vmove	0：正确 -1：错误	1 个轴以快速做连续运动
fast_vmove2	0：正确 -1：错误	2 个轴以快速做连续运动
fast_vmove3	0：正确 -1：错误	3 个轴以快速做连续运动
fast_vmove4	0：正确 -1：错误	4 个轴以快速做连续运动

3.3.2 功能说明

MPC08D 运动控制器提供了 8 个连续运动函数。连续运动是指各轴按

各自设定的速度、加速度运动，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运动。

（1）con_vmove、con_vmove2、con_vmove3、con_vmove4 说明

分别用于同时启动一个轴、两个轴、三个轴或四个轴，各轴独立以常速方式作连续运动，常速度由指令“set_conspeed”设置。

（2）fast_vmove、fast_vmove2、fast_vmove3、fast_vmove4 说明

分别用于同时启动一个轴、两个轴、三个轴或四个轴，各轴独立以快速方式作连续运动，梯形速度由指令“set_profile”设置。

3.3.3 例程

```
void main( )
{
    auto_set( );                    //检测控制器
    init_board( );                  //初始化控制器
    set_maxspeed(1,10000);          //设置最大速度
    set_profile(1,100,10000,8000);//设置初速、高速、加速度
    fast_vmove(1,1);                //启动 1 轴正向连续运动
    ……
}
```

3.4 回原点运动函数

3.4.1 指令列表

回原点运动参数设置和运动函数如表 3-6 所示。

表 3-6 回原点运动函数

函数	返回值	说明
set_maxspeed	0：正确 -1：错误	设置轴运动的最大速度
set_profile	0：正确 -1：错误	设置轴快速运动的参数
set_conspeed	0：正确 -1：错误	设置轴常速运动的参数
set_home_mode	0：正确 -1：错误	设置轴回零运动的模式

con_hmove	0：正确 -1：错误	1 个轴以常速做回零运动
con_hmove2	0：正确 -1：错误	2 个轴以常速做回零运动
con_hmove3	0：正确 -1：错误	3 个轴以常速做回零运动
con_hmove4	0：正确 -1：错误	4 个轴以常速做回零运动
fast_hmove	0：正确 -1：错误	1 个轴以快速做回零运动
fast_hmove2	0：正确 -1：错误	2 个轴以快速做回零运动
fast_hmove3	0：正确 -1：错误	3 个轴以快速做回零运动
fast_hmove4	0：正确 -1：错误	4 个轴以快速做回零运动

3.4.2 功能说明

MPC08D 运动控制器提供了 8 个回原点运动函数。回原点运动是指各轴按各自设定的速度、加速度运动，直到有外部原点信号、限位信号或报警信号，或者调用停止指令才停止运动。控制轴在回原点过程中，若先检测到有效的限位信号，控制轴将自动反向找原点。

运动控制器提供了 4 种回原点方式，通过接口函数“set_home_mode”设置工作方式：

方式 0：Origin 立即停止方式。控制轴以设定速度作回原点运动，原点信号有效时立即停止运动。

方式 1：Z 脉冲有效立即停止。控制轴以设定速度作回原点运动，Z 脉冲信号有效时立即停止运动。

方式 2：ORG 信号有效缓慢停止。梯形速度模式下，检测到原点接近开关信号有效时，控制轴按快速运动设置的加速度逐渐减速停止。

方式 3：梯形速度模式下，原点信号有效时，控制轴按快速运动方式设置的加速度逐渐减速至低速，直到 Z 脉冲有效立即停止运动。

注意：方式 2 和方式 3 只对快速回零运动有效。

（1）con_hmove、con_hmove2、con_hmove3、con_hmove4 说明

分别用于同时启动一个轴、两个轴、三个轴或四个轴，各轴独立以常速方式作回原点运动，常速度由指令“set_conspeed”设置。

（2）fast_hmove、fast_hmove2、fast_hmove3、fast_hmove4 说明

分别用于同时启动一个轴、两个轴、三个轴或四个轴，各轴独立以快速方式作回原点运动，梯形速度由指令“set_profile”设置。

3.4.3 例程

```
void main( )
{
    auto_set( );                //检测控制器
    init_board( );              //初始化控制器
    set_home_mode(1,0);         //设置 1 轴回原点模式 0
    set_maxspeed(1,1000);       //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
    ……
}
```

3.5 线性插补运动函数

3.5.1 指令列表

线性插补运动设置和运动函数有 6 个，如表 3-7 所示。

表 3-7 线性插补运动函数

函数	返回值	说明
set_vector_conspeed	0：正确 -1：错误	设置常速运动的速度
set_vector_profile	0：正确 -1：错误	设置快速运动的速度参数
con_line2	0：正确 -1：错误	两个轴作常速直线插补运动
con_line3	0：正确 -1：错误	三个轴作常速直线插补运动
con_line4	0：正确 -1：错误	四个轴作常速直线插补运动
fast_line2	0：正确 -1：错误	两个轴作快速直线插补运动
fast_line3	0：正确 -1：错误	三个轴作快速直线插补运动

fast_line4	0：正确 -1：错误	四个轴作快速直线插补运动

3.5.2 功能说明

插补运动是指两轴及多轴按照一定的算法进行联动，被控轴同时启动，并同时到达目标位置。插补运动以矢量速度运行，矢量速度分为常矢量速度和梯形矢量速度。插补运动函数中行程参数以脉冲数为编程单位。

MPC08D 运动控制器提供了 6 个线性插补运动函数。

（1）con_line2、con_line3、con_line4 说明

分别启动两个轴、三个轴、四个轴以常矢量速度作线性联动，每个被控轴的运动速度为常矢量速度在该轴上的分速度，各个被控轴同时启动，并同时到达目标位置。常矢量速度由指令“set_vector_conspeed”设置。

（2）fast_line2、fast_line3、fast_line4 说明

分别启动两个轴、三个轴、四个轴以梯形矢量速度作线性联动，每个被控轴的运动速度、加速度为梯形矢量速度、矢量加速度在该轴上的分量，各个被控轴同时启动，并同时到达目标位置。梯形矢量速度由指令“set_vector_profile”设置。

4 制动函数

4.1 制动函数

制动相关函数如表 4-1 所示。

表 4-1 制动函数

函数	返回值	说明
sudden_stop	0：正确 -1：错误	立即制动一个运动轴
sudden_stop2	0：正确 -1：错误	立即制动两个运动轴
sudden_stop3	0：正确 -1：错误	立即制动三个运动轴
sudden_stop4	0：正确 -1：错误	立即制动四个运动轴
decel_stop	0：正确 -1：错误	光滑制动一个运动轴
decel_stop2	0：正确 -1：错误	光滑制动两个运动轴
decel_stop3	0：正确 -1：错误	光滑制动三个运动轴
decel_stop4	0：正确 -1：错误	光滑制动四个运动轴
move_pause	0：正确 -1：错误	暂停一个运动轴
move_resume	0：正确 -1：错误	恢复一个轴的运动

4.2 功能说明

（1）sudden_stop、sudden_stop2、sudden_stop3、sudden_stop4 说明

立即运动方式下立即制动函数。它们使被控轴立即中止运动。这个函数执行后，控制器立即停止向电机驱动器发送脉冲，使之停止运动。

（2）decel_stop、decel_stop2、decel_stop3、decel_stop4 说明

立即运动方式下光滑制动函数。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。它们可以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。

（3）move_pause、move_resume 说明

这两个指令用于暂停立即运动模式下某轴的当前运动和恢复轴的运动。

- 对于快速直线插补运动、快速点位运动、快速连续运动、快速回原点运动等，调用“move_pause”函数，控制轴即以设置的梯形速度减速直到停止；调用“move_resume”后，控制轴又以梯形速度加速到高速。
- 对应常速直线插补运动、常速点位运动、常速连续运动、常速回原点运动，调用“move_pause”函数，控制轴立即停止运动；调用“move_resume”后，控制轴立即以常速度运动。

5 位置设置和读取函数

5.1 位置设置函数

5.1.1 指令列表

位置设置函数有 2 个，如表 5-1 所示。

表 5-1 位置设置函数

函数	返回值	说明
set_abs_pos	0：正确 -1：错误	设置一个轴的绝对位置值
reset_pos	0：正确 -1：错误	复位一个轴的当前位置值为零

6.1.2 功能说明

（1）set_abs_pos 说明

调用该函数可将控制轴当前绝对位置改为设定值，但从上一个位置到该位置之间不会产生轴的实际运动。调用该函数并将第二个参数设为 0 可实现“reset_pos”函数的功能。函数中位置参数以脉冲数为编程单位。

当控制轴处于运动状态时，该指令将不起作用。

（2）reset_pos 说明

函数该函数将控制轴的绝对位置和相对位置复位至 0，通常在轴的原点找到时调用，调用这个函数后，当前位置值变为 0，这以后，所有的绝对位置值均是相对于这一点的。

当控制轴处于运动状态时，该指令将不起作用。

该函数同时自动将辅助编码器计数值清零。

5.2 位置读取函数

5.2.1 指令列表

位置读取函数如表 5-2 所示。

表 5-2 位置读取函数

函数	返回值	说明
get_abs_pos	0：正确 -1：错误	读取一个轴的绝对位置值

5.2.2 功能说明

（1）get_abs_pos 说明

调用该函数可读取控制轴当前绝对位置值。如果执行过回原点运动（回原点后应调用“reset_pos”指令），那么这个绝对位置是相对于原点位置的；如果没有执行过回原点运动，那么这个绝对位置是相对于开机时的位置。函数中位置参数以脉冲数为编程单位。

该指令获取的控制轴绝对位置是由控制器输出脉冲的数量决定，在丢步或过冲等情况下，不能反映实际的位置值。

6 状态处理函数

6.1 运动状态查询函数

6.1.1 指令列表

运动状态查询函数有 4 个，如表 6-1 所示。

表 6-1 运动状态查询函数

函数	返回值	说明
check_status	其它：状态值 -1：错误	读取指定轴的状态
get_cur_dir	0：停止状态 -1：负向 1：正向 -2：错误	读取指定轴的当前运动方向
check_done	0：停止状态 1：运动状态 -1：错误	检查指定轴的运动是否已经完毕
get_rate	运动的速度	读取当前运动的速度

6.1.2 功能说明

（1）check_status 说明

MPC08D 控制控制器每个控制轴都有 1 个 32 位状态寄存器，在运动过程中，用户可以通过调用指令“check_status”查询轴的工作状态。该状态寄存器中每位（bit）的含义如下表所示。

表 6-2 控制轴状态寄存器定义

数据位	状态	说明
D30	0：无效 、1：有效	原点信号状态
D29	0：无效 、1：有效	正向限位信号状态
D28	0：无效 、1：有效	负向限位信号状态
D26	0：无效 、1：有效	报警信号状态
其他	内部状态	
D8	0：无效 、1：有效	Z 脉冲信号状态
D0	0：运动 、1：停止	运动状态

（2）check_done 说明

用于检测指定轴的运动状态。在立即运动方式下，必须使用该函数判断控制轴的运动状态，只有在该轴停止运动后，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。

（3）get_cur_dir 说明

读取指定轴的当前运动方向。

（4）get_rate 说明

读取控制轴的当前运动速度。有可能读取的速度与用户设置的速度有差异。这主要是由于控制器速度分辨率引起的差异。因为输出脉冲频率由两个变量控制：脉冲分辨率和倍率，两者的乘积为实际输出的脉冲频率。函数 set_maxspeed 设置最大输出脉冲频率即为修改脉冲分辨率，使用时可按照实际输出速度设置最大速度以获得比较好的速度精度。

6.2 专用输入检测函数

6.2.1 指令列表

专用输入检测函数有 7 个，如表 6-3 所示。

表 6-3 专用输入状态检测函数

函数	返回值	说明
check_limit	0：无效 1：正限位信号有效 -1：负限位信号有效 2：正负限位信号均有效 -3：错误	检查指定轴的限位信号是否有效
check_home	0：无效 1：有效 -3：错误	检查指定轴的原点信号是否有效
check_alarm	0：无效 1：有效 -3：错误	检查指定轴的报警信号是否有效
check_card_alarm	0：无效 1：有效 -3：错误	检查板卡的报警信号是否有效
check_sfr	其它：IO 状态 -1：错误	读取专用输入口所有的开关量状态
check_sfr_bit	0：低电平	读取专用输入口某位的开关量状

	1：高电平 -1：错误	态

6.2.2 功能说明

（1）check_limit、check_home、check_alarm、check_card_alarm 说明

分别用于检测指定轴的限位信号状态、原点信号状态、报警信号状态和板卡报警信号状态。调用“check_status”函数可读取出整个状态寄存器值，而上述这几个函数分别获取其中部分信号状态。注意：只有在这些专用输入使能的情况下，即“enable_org”、“enable_limit”、“enable_alm”、“enable_caed_alm”等指令使相应专用信号使能，上述函数才能返回正确的专用输入口状态。

（2）check_sfr、check_sfr_bit 说明

运动控制器将个控制轴的原点、限位、报警、减速信号保存在一个专用寄存器中，若不使用这些专用输入信号，可用“enable_org”、“enable_limit”、“enable_alm”、“enable_caed_alm”等指令使相应专用信号无效，此时，这些端口可用作通用输入口。

通过“check_sfr”指令可从该专用寄存器中读取所有专用输入开关量状态。

通过“check_sfr_bit”指令可从该专用寄存器中读取某位专用输入开关量状态。

作通用输入口时的接线图请参见《MPC08D User.doc》中的“通用输入、输出的连接方法”一栏。

专用输入寄存器各位定义如表 6-4 所示。

表 6-4 专用输入寄存器定义

状态位	输入信号	说明
20	ALM	ALM 无效时作为通用输入口
19	Z4	不能使能为通用输入信号
18	ALM4	ALM4 无效时作为通用输入口
17	ORG4	ORG4 无效时作为通用输入口
16	EL4-	EL4-无效时作为通用输入口
15	EL4+	EL4+无效时作为通用输入口
14	Z3	不能使能为通用输入信号
13	ALM3	ALM3 无效时作为通用输入口
12	ORG3	ORG3 无效时作为通用输入口

11	EL3-	EL3-无效时作为通用输入口
10	EL3+	EL3+无效时作为通用输入口
9	Z2	不能使能为通用输入信号
8	ALM2	ALM2 无效时作为通用输入口
7	ORG2	ORG2 无效时作为通用输入口
6	EL2-	EL2-无效时作为通用输入口
5	EL2+	EL2+无效时作为通用输入口
4	Z1	不能使能为通用输入信号
3	ALM1	ALM1 无效时作为通用输入口
2	ORG1	ORG1 无效时作为通用输入口
1	EL1-	EL1-无效时作为通用输入口
0	EL1+	EL1+无效时作为通用输入口

7 通用 IO 操作函数

7.1 数字 IO 口操作函数

7.1.1 指令列表

运动控制器 MPC08D 配合 IO 扩展板 EA1616 可以提供带光电隔离 24 路通用输入口和 26 路通用输出口。通用数字 IO 操作函数有 4 个，如表 7-1 所示。

表 7-1 数字 IO 操作函数

函数	返回值	说明
checkin_byte	其它：IO 状态 -1：错误	读取扩展输入口所有的开关量状态
checkin_bit	0：低电平 1：高电平 -1：错误	读取扩展输入口某位的开关量状态
outport_byte	0：正确 -1：错误	设置通用输出口各位的开关量状态
outport_bit	0：正确 -1：错误	设置通用输出口某位的开关量状态

7.1.2 功能说明

（1）checkin_byte、checkin_bit 说明

用户使用该指令读取运动控制器 24 路通用输入口状态，其中 0-8 路位于运动控制器 MPC08D，9-24 路通过 EA1616 扩展实现。

“checkin_byte” 从运动控制器的 24 位通用输入口读入所有输入开关量状态。

“checkin_bit” 从运动控制器的 24 位通用输入口读入某一位输入开关量状态。

通用输入口接线图请参见《MPC08D User.doc》中的“通用输入、输出的连接方法”一栏。

表 7-2 中高 16 位（9-24 路）通用输入口在 P37-05 转接板中定义

P37-05 转接板引脚	37 芯电缆引脚	名称	说 明
P19	19	IN9	通用输入 9
P37	37	IN10	通用输入 10
P18	18	IN11	通用输入 11
P36	36	IN12	通用输入 12
P17	17	IN13	通用输入 13
P35	35	IN14	通用输入 14
P16	16	IN15	通用输入 15
P34	34	IN16	通用输入 16
P15	15	IN17	通用输入 17
P33	33	IN18	通用输入 18
P14	14	IN19	通用输入 19
P32	32	IN20	通用输入 20
P13	13	IN21	通用输入 21
P31	31	IN22	通用输入 22
P12	12	IN23	通用输入 23
P30	30	IN24	通用输入 24

低 8 位（1-8 路）通用输入口在 P62-05 转接板中的定义如下。

表 7-3 低 8 位（1-8 路）通用输入口在 P62-05 转接板中定义

P62-05 转接板引脚	名称	说 明
IN1	IN1	通用输入 1
IN2	IN2	通用输入 2
IN3	IN3	通用输入 3
IN4	IN4	通用输入 4
IN5	IN5	通用输入 5
IN6	IN6	通用输入 6
IN7	IN7	通用输入 7
IN8	IN8	通用输入 8

（3）outport_byte、outport_bit 说明

用户使用该指令设置运动控制器 26 位通用输出口开关量状态。26 位中低 10 位通过 MPC08D 主板 DB62 引出在 P62-05 转接板端，其余高 16 位通过 DB37 引出在 P37-05 转接板端。

“outport_byte”同时设置 26 位通用输出口开关量状态。

“outport_bit” 设置通用输出口某位的开关量状态。

26 位通用输出口接线图请参见《MPC08D User.doc》中的“通用输入、输出的连接方法”一栏。

表 7-4 低 10 位通用输出口在 P62-05 转接板中定义

P62-05 引脚	名称	说明
OUT1	OUT1	通用输出 1
OUT2	OUT2	通用输出 2
OUT3	OUT3	通用输出 3
OUT4	OUT4	通用输出 4
OUT5	OUT5	通用输出 5
OUT6	OUT6	通用输出 6
OUT7	OUT7	通用输出 7
OUT8	OUT8	通用输出 8
OUT9	OUT9	通用输出 9
OUT10	OUT10	通用输出 10

表 7-5 高 16 位通用输出口在 P37-05 转接板中定义

P37-05 引脚	37 芯电缆引脚	名称	说明
P11	11	OUT11	通用输出 11
P29	29	OUT12	通用输出 12
P10	10	OUT13	通用输出 13
P28	28	OUT14	通用输出 14
P9	9	OUT15	通用输出 15
P27	27	OUT16	通用输出 16
P8	8	OUT17	通用输出 17
P26	26	OUT18	通用输出 18
P6	6	OUT19	通用输出 19
P24	24	OUT20	通用输出 20
P5	5	OUT21	通用输出 21
P23	23	OUT22	通用输出 22
P4	4	OUT23	通用输出 23
P22	22	OUT24	通用输出 24

P3	3	OUT25	通用输出 25
P21	21	OUT26	通用输出 26

8 其它功能

8.1 反向间隙处理

8.1.1 指令列表

反向间隙处理函数有 3 个，如表 8-1 所示。

表 8-1 速度设置函数

函数	返回值	说明
set_backlash	0：正确 -1：错误	设置由于机构换向形成间隙的补偿值
start_backlash	0：正确 -1：错误	开始补偿由于机构换向间隙而导致的位置误差
end_backlash	0：正确 -1：错误	停止补偿由于机构换向间隙而导致的位置误差

8.1.2 功能说明

（1）set_backlash、start_backlash、end_backlash 说明

作为驱动装置，如果使用齿轮机构或其它传动装置，两个齿之间或多或少存在间隙，如下图所示。当驱动侧的驱动方向发生改变时（如下图右到左或左到右），通过反向间隙补偿值的设定增加驱动侧的运动脉冲数，减小机械间隙的影响。

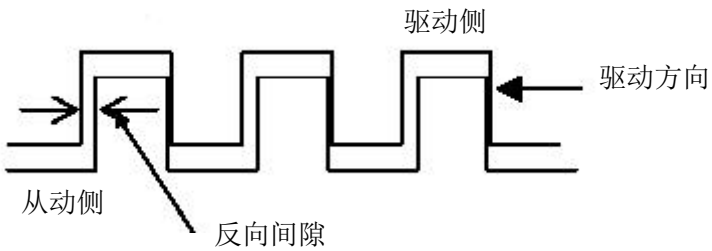

图 8-1 反向间隙示意图

当控制系统存在反向间隙时，使用 MPC08D 提供的三个函数可有效消除反向间隙，保证系统位置精度。使用步骤如下：

- 根据反向间隙大小，调用函数“set_backlash”设置一个轴的补偿

值。

- 设置好反向间隙补偿值后，调用函数“start_backlash”启动反向间隙补偿。以后控制轴反向时，运动控制器自动增加反向间隙补偿。
- 当需要结束反向间隙补偿，调用函数“end_backlash”取消反向间隙补偿。以后控制轴反向时，运动控制器不再进行反向间隙补偿。

8.1.3 例程

```
void main( )
{
    auto_set( );              //检测控制器
    init_board( );            //初始化控制器
    set_maxspeed(1,1000);     //设置最大速度
    set_conspeed(1,1000);     //设置常速度
    set_backlash(1,100);      //设置轴 1 的反向间隙为 100（脉冲）
    start_backlash(1);        //启动反向间隙补偿
    con_pmove(1,10000);       //启动 1 轴正向移动 10000
    ……//等待轴 1 停止
    con_pmove(1,-10000);      //启动 1 轴反向移动 10000，此后系统自动
                              多运动 100 的行程，以补偿反向间隙
    ……
    end_backlash(1);          //结束反向间隙补偿
    ……
}
```

8.2 动态改变目标位置

8.2.1 指令列表

运动控制器提供运动中动态改变目标位置功能，操作函数有 1 个，如表 8-2 所示。

表 8-2 动态改变目标位置处理函数

函数	返回值	说明
change_pos	0：正确 -1：错误	实现运动中动态改变目标位置功能

8.2.2 功能说明

（1）change_pos 说明

在相对位置模式下使用函数 change_pos 来动态改变目标位置。单轴点位运动过程中，在常速模式或梯形速度模式下，若用户发现发出的运动指令终点位置需要改变，可在该点位运动结束前或者结束后调用“change_pos”动态改变终点位置。系统将自动按新的终点位置运动。该终点位置以上一条用户发出的运动指令（fast_pmove、con_pmove）的起点为起点。可多次调用该函数来改变目标位置。

8.2.3 例程

```
void main( )
{
    auto_set( );                //检测控制器
    init_board( );              //初始化控制器
    set_maxspeed(1,1000);       //设置最大速度
    set_conspeed(1,1000);       //设置常速度
    con_pmove(1,-1000);         //启动 1 轴负向移动 1000
    ……                          //1 轴向负向运动过程中
    Change_pos(1,10000);        //不用等待 1 轴运动停止，以 con_pmove 的
                                起点为起点反向向 10000 的位置运动。
    ……
}
```

8.3 可掉电保护数据区读写功能

8.3.1 指令列表

运动控制器提供可掉电保护数据区读写功能，用户可以向指定的数据存储区写入数据信息，以实现诸如软件加密、设备参数存储等功能。具体的操

作函数如表 8-3 所示。

表 8-3 可掉电保护数据区读写函数

函数	返回值	说明
write_password_flash	0：写成功； -1：参数错误导致的写失败； -2：校验密码错误导致的写失败。	带密码保护功能的数据区写函数
read_password_flash	0：读取数据成功； -1：参数错误导致的读失败； -2：校验密码错误导致的读失败。	带密码保护功能的数据区读函数
clear_password_flash	0：擦除数据成功； -1：参数错误导致的擦除失败； -2：校验密码错误导致的擦除失败。	擦除带密码保护功能的数据区
write_flash	0：写成功； -1：参数错误导致的写失败；	一般数据区的写函数
read_flash	0：读取数据成功； -1：参数错误导致的读失败；	一般数据区的读函数
clear_flash	0：擦除数据成功； -1：参数错误导致的擦除失败；	擦除一般数据区数据

8.3.2 功能说明

运动控制器 MPC08D 提供 1M 存储空间的可掉电保护数据区，分为 16 片区域，每片区域大小为 64K 字节。16 片区域对应的地址分配如下表 8-4 所示。其中区域 0 具备密码保护功能，即当用户对该区域设置读写密码 password 后，以后对该片区域的地址进行擦除、读和写操作均需要校验 password。区域 1-15 为一般数据区，对该类区域进行操作不需要校验密码。

可掉电保护数据区内写入的数据类型为 long 型，每片区域可写入的数据个数为 16384 =（64*1024/4），地址编号为 0--16383。对区域 0 的 16383 号地址写入的数据即为读写密码 password，首次使用读写密码 password 值为 0xffffffff。

表 8-4 区域编号与首地址对于关系表

区域编号	首地址
0	0X000000
1	0X010000
2	0X020000

3	0X030000
4	0X040000
5	0X050000
6	0X060000
7	0X070000
8	0X080000
9	0X090000
10	0X0A0000
11	0X0B0000
12	0X0C0000
13	0X0D0000
14	0X0E0000
15	0X0F0000

（1）write_password_flash、read_password_flash、clear_password_flash 说明

用于对带密码保护功能的数据区进行写、读和擦除操作。对数据区进行写操作之前必须对该片区域进行擦除操作，否则无法写入待写数据。**擦除操作对整片有效，可以对指定片区的区域进行擦除，使得该区域的 64K 字节空间所有位为“1”。读写操作对 4 个字节的连续地址有效，所以进行擦除操作时请谨慎！**

（2）write_flash、read_flash、clear _flash 说明

用于对一般数据区进行写、读和擦除操作。对数据区进行写操作之前必须对该片区域进行擦除操作，否则无法写入待写数据。**擦除操作对整片有效，可以对指定片区的区域进行擦除，使得该区域的 64K 字节空间所有位为“1”。读写操作对 4 个字节的连续地址有效，所以进行擦除操作时请谨慎！**

8.1.3 例程

```
void main( )
{
    Long data;
    auto_set( );              //检测控制器
    init_board( );            //初始化控制器
```

```
    clear_password_flash(1, 0xffffffff);   //擦除区域 0
    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
}
```

8.4 板卡号和版本读取

8.4.1 指令列表

运动控制器提供 1 个板卡号读取函数和 3 个版本读取函数，如表 8-4 所示。

表 8-4 板卡号和版本读取函数

函数	返回值	说明
check_IC	其它：卡号 -1：错误	查询用户设置的控制卡的本地 ID 号
get_lib_ver	0	查询函数库的版本
get_sys_ver	0：正确 -1：错误	查询驱动程序的版本
get_card_ver	0：正确 -1：错误	查询运动控制器固件版本

8.4.2 功能说明

（1）check_IC 说明

运动控制器上设计了一个旋钮开关，可以设定多块板卡共用时各板卡的本地 ID 号。旋钮开关最大设定值为 0xFH，目前只能设定为 0x0H～0x7H，只能支持 6 块板卡共用。使用函数“check_IC”可读取出板卡的本地 ID 号。使用“check_IC”时，计算机中只安装一张控制卡。

运动控制器出厂时，初始化本地 ID 号均为 0，如果要多卡共用，用户需要改变该设置。例如，若 6 卡共用，则需要将各板卡依次设置为 0、1、2、

3、4、5。若设置的本地 ID 号有重复或为其它值，控制系统调用板卡初始化函数“auto_set”和“init_board”后，会提示初始化失败。调用“get_err”、“get_last_err”等函数可读取错误信息。

（2）get_lib_ver、get_sys_ver、get_card_ver 说明

分别用于读取运动控制器附带的函数库版本号、驱动程序版本号、板卡版本号。

函数返回后，版本号存放在函数的参数指针变量中。

9 错误代码及处理函数

9.1 错误代码处理函数

9.1.1 指令列表

运动控制器可返回最近 10 个错误状态，以便用户了解系统最近状况。错误代码操作函数有 3 个，如表 9-1 所示。

表 9-1 错误代码操作函数

函数	返回值	说明
get_err	0：正确 -1：错误	获得最近 10 个错误代码
get_last_err	0：正确 -1：错误	获得最近一个错误代码
reset_err	0	将所有（10 个）错误代码复位清零

9.1.2 功能说明

（1）get_err 说明

读取最近的 10 个中的错误信息。运动控制器提供了如下错误信息表。

表 9-2 错误代码含义

错误代码（16 进制）	含义
0x00000000	控制器工作正常
0x00000004	没有调用 auto_set 设置控制器
0x00000005	无效的设备句柄
0x00000006	与驱动程序通讯失败
0x00000007	检测不到运动控制卡
0x00000008	运动控制卡上检测不到轴
0x00000009	运动控制卡没有进行初始化或初始化没有成功
0x0000000a	计算机内安装了过多的的运动控制卡，MPC08D 最大允许 4 卡共用
0x0000000b	检测到的轴数板卡设计的轴数不符合，MPC08D 每张卡设计轴数为 4

0x0000000c	板卡本地 ID 号小于 0
0x0000000d	板卡本地 ID 号大于 4
0x0000000e	使用一张控制卡时，其本地 ID 号没有设置为 0
0x0000000f	多卡共用时，第一张板卡的本地 ID 号没有设置为 0
0x00000010	多卡共用时，各板卡的本地 ID 号不是从 0 开始依次增大
0x00000012	驱动程序版本与函数库版本不匹配
0x00000013	板卡固件版本与函数库版本不匹配
0x00000014	板卡初始化错误
0x02000001	指令中轴号参数设置错误
0x02000002	指令中速度参数设置错误
0x02000005	指令中卡号参数设置错误
0x02000007	圆弧运动的两个轴不在同一张卡上
0x0200000a	数字 IO 操作中读写控制位参数超过最大允许范围
0x0200000d	设置的看门狗定时器定时值不在 1～60000 毫秒范围内
0x02000016	设置的椭圆比例小于等于 0
0x02000017	多轴运动指令中轴号设置成相同值
0x02000018	反向间隙小于 0
0x02000019	参数设置错误
0x0200001b	控制轴启动多种功能时，功能间有冲突
0x0200001c	控制卡启动多种功能时，功能间有冲突
0x0200001e	启动运动指令前，内部状态检测不通过
0x02000020	批处理设置指令 open_list、close_list、add_list 等没有成对调用
0x02000021	前瞻 start_lookahead、end_lookahead 没有成对调用
0x02000022	在批处理中调用了前瞻指令
0x02000023	在前瞻处理中调用了错误的指令

（2）get_last_err 说明

返回最近一次错误代码。

（3）reset_err 说明

将 10 个错误代码变量清零。

10 函数描述

运动控制器函数按功能排列，指令原型以 C 语言描述。
如果正在学习使用运动控制器，或者应用程序正处于调试阶段，请不要连接机械系统，以免误操作损坏设备。

10.1 控制器初始化函数

该类函数主要用于初始化 MPC08D 卡。若要使用 MPC08D 各项功能，必须首先依次调用函数 auto_set、init_board，完成控制器初始化后才能调用其它函数。

表 10-1 控制器和轴设置函数

函数原型	说明
int auto_set(void)	自动检测和自动设置控制器
int init_board(void)	对控制器硬件和软件初始化

函 数 名：auto_set

目　　的：用 auto_set 函数自动检测运动控制器的数量、各卡上的轴数，并自动设置每块运动控制器。

语　　法：int auto_set（void）;

描　　述：可以调用 auto_set 完成运动控制器的数量、轴数的自动检测，并自动设置这些参数。

返 回 值：如果调用成功，auto_set 函数返回总轴数；若检测不到卡，返回 0；调用失败返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

注　　释：每个程序必须首先调用 auto_set 完成对卡的自动检测和自动设置，否则运动控制器不能工作。该函数在程序中只能被调用一次。

参　　见：

调用例子：

函 数 名：init_board

目　　的：用 init_board 函数初始化运动控制器。

语　　法：int init_board（void）；

描　　述：在用 auto_set 自动检测和设置之后，必须调用 init_board 函数来对控制器进行初始化。init_board 函数主要初始化控制器的各个寄存器、各轴的脉冲输出模式（脉冲/方向）、常速度（2000pps）、梯形速度（初速 2000pps，高速 8000pps，加减速 80000ppss）、矢量常速度（2000pps）、矢量梯形速度（初速 2000pps，高速 8000pps，加减速 80000ppss）等等。该函数在程序中只能调用一次。**init_board 函数必须在 auto_set 之后调用。如果不调用 init_board 函数初始化，控制器将不能正常工作。若需改变脉冲输出模式、速度等初始化数据，可调用其它函数来修改。**

返 回 值：如果调用成功，init_board 函数返回插入的板卡数；若检测不到卡，返回 0；-1 表示出错。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：auto_set

调用例子：

10.2 属性设置函数

该类函数主要用于设置 MPC08D 卡控制轴的使用属性和板卡的相关属性。

表 10-2 控制轴属性设置函数

函数原型	说明
int set_outmode(int ch, int mode, int outlogic)	设置各轴脉冲输出模式
int set_home_mode(int ch, int home_mode)	设置回原点模式
int set_dir(int ch，int dir)	设置一个轴的运动方向
int enable_alm(int ch，int flag)	设置一个轴的外部报警信号是否有效

int enable_el(int ch，int flag)	设置一个轴的外部限位信号是否有效
int enable_org(int ch，int flag)	设置一个轴的外部原点信号是否有效
int enable_card_alm(int cardno,int flag)	设置控制器的报警信号是否有效
int set_alm_logic(int ch,int flag)	设置一个轴的报警信号是高电平有效还是低电平有效
int set_el_logic(int ch,int flag)	设置一个轴的限位信号是高电平有效还是低电平有效
int set_org_logic(int ch,int flag)	设置一个轴的原点信号是高电平有效还是低电平有效
int set_card_alm_logic (int ch，int flag)	设置板卡的报警信号是高电平有效还是低电平有效

函 数 名：set_outmode

目　　的：用于设置每个轴的脉冲输出模式。板卡初始化设置为脉冲/方向模式，如果驱动器要求双脉冲（正向脉冲/反向脉冲）控制信号接口，那么应在 init_board 函数后调用该函数。

语　　法：int set_outmode（int ch，int mode，int outlogic）；

ch：需要设置输出方式的控制轴。

mode：脉冲输出模式设置（1 为脉冲 / 方向方式，0 为双脉冲方式）。

outlogic：该参数在 MPC08D 中无效。

描　　述：在缺省情况下，init_board 函数将所有轴设置为脉冲 / 方向模式。如果驱动器要求双脉冲（正向脉冲和反向脉冲）模式的输入，那么应在 init_board 函数后调用 set_outmode 重新设置所要求的模式。注意：控制器的输出模式应与所连接的驱动器的输入信号模式一致，否则电机将不能正常工作。

返 回 值：如果输出方式设置成功，则 set_outmode 返回值为 0，否则返回 -1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：init_board

调用例子：set_outmode（2，0，1）；/*将第 2 轴的脉冲输出模式设置为双脉冲模式。*/

函 数 名：set_home_mode

目　　的：用于设置各轴回原点运动时检测原点信号的方式。

语　　法：int set_home_mode（int ch，int home_mode）

ch：是所设置的轴；

home_mode：回原点运动时检测原点信号的方式

0：检测到原点接近开关信号轴立即停止运动；

1：检测到出现编码器 Z 相脉冲信号时立即停止运动。

2：梯形速度模式下，检测到原点接近开关信号有效时控制轴按快速运动方式设置的加速度逐渐减速停止；

3：梯形速度模式下，原点信号有效时，控制轴按快速运动方式设置的加速度逐渐减速至低速，直到 Z 脉冲有效立即停止运动。

描　　述：在被控设备（比如数控机床等）回原点运动时，MPC08D 运动控制器将自动检测原点信号，并在到达原点位置时自动停止运动。原点信号一般由接近开关发送。在一些回原点时定位精度要求较高的场合，原点信号除了接近开关信号之外，还要检测执行电机上光电编码器的 Z 相脉冲，即仅当接近开关信号和 Z 相脉冲信号同时出现时，才表明已到达原点。函数 set_home_mode 就是用于设置每个轴在回原点运动时检测原点信号的方式。注意：只有执行电机上装有光电编码器时，才能使用 Z 相脉冲信号作为回原点运动的检测信号，否则将无法正确完成回原点运动。

返 回 值：如果设置成功，返回 0，否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：set_home_mode（1，1）；

函 数 名：set_dir

目　　的：用于设置轴的运动方向。

语　　法：int set_dir（int ch，int dir）；

ch：所要设置的轴。

dir：表示被控轴的运动方向，+1 表示正方向；-1 表示负方向。

描　　述：调用该函数可在新的运动指令发出前设定某轴的运动方向。但最终运动方向还是由运动指令确定。该指令的主要用于某些驱动器要求运动方向必须比脉冲先发出的情形。

返 回 值：如果函数调用成功，则返回值为 0；否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：set_dir（1，-1）；/*将第 1 轴的运动方向设置成为负方向*/

函 数 名：enable_alm

目　　的：用于设置轴的外部报警信号是否有效。

语　　法：int enable_alm（int ch，int flag）；

ch：控制轴的编号。

flag：外部报警信号是否有效的标志，1 表示使能外部报警信号；0 表示禁止外部报警信号。

描　　述：调用该函数设置某轴的外部报警信号是否有效。如果将某轴的外部报警信号设置为无效，则对应的报警信号输入端口（ALM）可作为通用输入口使用，使用函数 check_sfr 或 check_sfr_bit 可读取相应端口状态。

返 回 值：如果函数调用成功，则返回值为 0；否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：check_sfr，check_sfr_bit

调用例子：enable_alm（1，0）；/*禁止第 1 轴的外部报警信号*/

函 数 名：enable_el

目　　的：用于设置轴的外部限位信号是否有效。

语　　法：int enable_el（int ch，int flag）；

ch：控制轴的编号。

flag：外部限位信号是否有效的标志，1 表示使能外部限位信号；0 表示禁止外部限位信号。

调用例子：enable_el（1，0）；/*将第 1 轴的外部限位信号设置为无效*/

描　　述：调用该函数设置某轴的外部限位信号是否有效。如果将某轴的外部限位信号设置为无效，则对应的限位信号输入端口（EL+、EL-）可作为通用输入口使用，使用函数 check_sfr 或 check_sfr_bit 可读取相应端口状态。

返 回 值：如果函数调用成功，则返回值为 0；否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：check_sfr，check_sfr_bit

函 数 名：enable_org

目　　的：用于设置轴的外部原点信号是否有效。

语　　法：int enable_org（int ch，int flag）；

ch：控制轴的编号。

flag：外部原点信号是否有效的标志，1 表示使能外部原点信号；0 表示禁止外部原点信号。

描　　述：调用该函数设置某轴的外部原点信号是否有效。如果将某轴的外部原点信号设置为无效，则对应的原点信号输入端口（ORG）可作为通用输入口使用，使用函数 check_sfr 或 check_sfr_bit 可读取相应端口状态。

返 回 值：如果函数调用成功，则返回值为 0；否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：check_sfr，check_sfr_bit

调用例子：enable_org（1，0）；/*将第 1 轴的外部原点信号设置为无效*/

函 数 名：enable_card_alm

目　　的：用于设置板卡报警信号是否有效。

语　　法：int enable_card_alm（int cardno, int flag）

cardno：板卡的编号。

flag：板卡的外部报警信号是否有效的标志，1 表示使能外部报警信号；0 表示禁止外部报警信号。

描　　述：调用该函数设置外部报警信号是否有效。如果将板卡的外部报警信号设置为无效，则对应的报警信号输入端口（ALM）可作为通用输入口使用，使用函数 check_sfr 或 check_sfr_bit 可读取相应端口状态。

返 回 值：如果函数调用成功，则返回值为 0；否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：check_sfr，check_sfr_bit

调用例子：enable_card_alm（1，0）；/*将 1 号卡报警信号设置为无效*/

函 数 名：set_alm_logic

目　　的：用于设置一个轴报警信号是高电平有效还是低电平有效。

语　　法：int set_alm_logic（int ch，int flag）；

ch：所要设置的轴；

flag：外部报警信号有效电平标志，1 表示外部报警开关高电平触发；0 表示外部报警开关低电平触发。

描　　述：调用该函数设置控制轴报警信号触发的有效电平，以满足外部常开或常闭接近开关的需要。如果将控制轴的外部报警信号设置为高电平有效，则对应的报警信号输入端口为高电平时该轴停止运动。初始化时系统默认高电平有效。

返 回 值：如果函数调用成功，则返回值为 0；否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：set_alm_logic（1，0）；/*将第 1 轴的报警信号设置成为低电平有效*/

函 数 名：set_el_logic

目　　的：用于设置一个轴限位信号是高电平有效还是低电平有效。

语　　法：int set_el_logic（int ch，int flag）；

ch：控制轴的编号。

flag：外部限位信号有效电平标志，1 表示外部限位开关高电平触发控制器；0 表示外部限位开关低电平触发控制器。

描　　述：调用该函数设置控制轴限位信号触发的有效电平，以满足外部常开或常闭接近开关的需要。如果将控制轴的外部限位信号设置为高电平有效，则某方向的限位信号输入端口为高电平时，轴在该方向的运动自动停止。初始化时系统默认高电平有效。

返 回 值：如果函数调用成功，则返回值为 0；否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：set_el_logic（1，0）；/*将第 1 轴的限位信号设置成为低电平有效*/

函 数 名：set_org_logic

目　　的：用于设置一个轴原点信号是高电平有效还是低电平有效。

语　　法：int set_org_logic（int ch，int flag）；

ch：控制轴的编号。

flag：外部原点信号有效电平标志，1 表示外部原点开关高电平触发控制器；0 表示外部原点开关低电平触发控制器。

描　　述：调用该函数设置控制轴的外部原点信号有效电平，以满足外部常开或常闭接近开关的需要。如果将控制轴的外部原点信号设置为高电平有效，则对应的原点信号输入端口为高电平时表示轴回到原点。初始化时系统默认高电平有效。

返 回 值：如果函数调用成功，则返回值为 0；否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：set_org_logic（1，0）；/*将第 1 轴的原点信号设置成为低电平有效*/

函 数 名：set_card_alm_logic

目　　的：用于设置板卡报警信号是高电平有效还是低电平有效。

语　　法：int set_card_alm_logic（int cardno，int flag）；

cardno：板卡的编号。

flag：外部报警信号有效电平标志，1 表示外部报警开关高电平触发控制器；0 表示外部报警开关低电平触发控制器。

描　　述：MPC08D 运动控制器有一个板卡报警信号，该报警信号对板卡的所有轴均有效，当该报警信号有效时所有的控制轴均停止运动，区别于每轴的报警信号仅对该轴有效。调用该函数设置板卡的报警信号有效电平，以满足外部常开或常闭接近开关的需要。如果将外部报警信号设置为高电平有效，则对应的报警信号输入端口为高电平时所有轴自动停止运动。初始化时系统默认高电平有效。

返 回 值：如果函数调用成功，则返回值为 0；否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：set_card_alm_logic（1，0）；/*将 1 号卡报警信号设置成为低电平有效*/

10.3 运动参数设置函数

该类函数主要用于设置 MPC08D 卡运动速度、加速度、位置等参数。

表 10-3 控制轴属性设置函数

函数原型	说明
int set_maxspeed(int ch , double speed)	设置控制轴最大速度
int set_conspeed(int ch, double conspeed)	设置各轴常速度
set_profile(int ch , double vl , double vh , double ad, double dc)	设置快速运动的梯形速度
int set_vector_conspeed(double con_speed)	设置常矢量速度
set_vector_profile(double vec_vl ,	设置梯形矢量速度

double vec_vh ,double vec_ad,double vec_dc);	
int set_s_curve(int ch,int mode)	设置轴的快速运动模式
int set_s_section(int ch,double accel_sec,double decel_sec)	设置 S 曲线运动的 S 升降速范围
int set_abs_pos (int ch，double pos)	设置一个轴的绝对位置值
int reset_pos（int ch）	当前位置值复位至零

函 数 名：set_maxspeed

目　　的：用于设置控制轴的最大速度。

语　　法：int set_maxspeed（int ch，double speed）；

ch：所设置的控制轴。

speed：设置的最大速度值，单位为脉冲 / 秒（pps）。

描　　述：在缺省情况下，板卡初始化设置所有轴的最大速度为 2MHz。此时速度分辨率较差，**若要获得较高速度精度，可按照实际最大速度进行设置。**最大脉冲频率可设置为 2000000 Hz，若设置值超过允许的最大值，控制器将按 2MHz 设置。最小脉冲频率可设置为 81.91 Hz，若设置值小于允许的最小值，控制器按 81.91 设置。

返 回 值：如果输出方式设置成功，则 set_ maxspeed 返回值为 0，否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：set_conspeed

调用例子：set_maxspeed（2，10000）；/*将第 2 轴的最大速度设置为 10000pps*/

函 数 名：set_conspeed

目　　的：用 set_conspeed 函数来设置一个轴在常速运动时的速度。

语　　法：int set_conspeed（int ch，double conspeed）；

ch：控制轴的编号。

conspeed：设定的常速度值，单位为脉冲 / 秒（pps）。

描　　述：函数 set_conspeed 可以设定在常速运动方式下的速度。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 2000000 Hz，若设置值超

过允许的最大值，控制器将按 2MHz 设置。最小脉冲频率可设置为 0.2 Hz，若设置值小于允许的最小值，控制器按 0.2 设置。常速度值一般设置较低，以免造成控制电机（尤其是开环的步进电机）丢步或过冲。如果需要高速运动，最好使用梯形速度方式。

返 回 值：如果常速度值设置成功，set_conspeed 返回 0 值，出错时返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：set_profile，set_vector_conspeed

调用例子：set_conspeed（2，400）；

函 数 名：set_profile

目　　的：用 set_profile 函数来设定在快速运动（包括 fast_hmove，fast_vmove，fast_pmove 等）方式下的梯形速度的各参数值；

语　　法：set_profile(int ch , double vl , double vh , double ad, double dc)；

ch：控制轴的编号;

vl：设定低速（起始速度）的值；单位为 pps（脉冲 / 秒）；

vh：设定高速（恒速段）的速度值；单位为 pps（脉冲秒）;

ad：设定加速度大小；单位为 ppss（脉冲 / 秒 / 秒）;

dc：设定减速度大小；单位为 ppss（脉冲 / 秒 / 秒）

描　　述：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（梯型速度模式时，减速度值可以不等于加速度值）。这几个参数的缺省值分别为 2000、8000、80000、80000。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 2000000Hz。若设置值超过允许的最大值，控制器将按 2MHz 设置。最小脉冲频率可设置为 10 Hz，若设置值小于允许的最小值，控制器按 10 设置。加速值最小可设置为 20，小于该值将按 20 设置。

返 回 值：如果设定参数值成功，set_profile 返回 0，出错返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：set_conspeed，set_vector_conspeed，set_vector_profile

调用例子：set_profile（3，600，6000，10000,10000）；

函 数 名：set_vector_conspeed

目　　的：用 set_vector_conspeed 函数来设置常速方式下的矢量速度，这个矢量速度在两轴或多轴直线插补运动中将会用到；

语　　法：int set_vector_conspeed（double vec_conspeed）；
vec_conspeed：在常速插补时的矢量速度。

描　　述：函数 set_vector_conspeed 为下列二轴或多轴插补运动函数设置矢量速度：con_line2、con_line3、con_line4 等。它不能为 fast_lin2、fast_line3 等快速插补运动设置运动速度（它们的速度依赖于 set_vector_profile）。最大脉冲频率可设置为 2MHz，若设置值超过允许的最大值，控制器将按 2MHz 设置。最小脉冲频率可设置为 1 Hz，若设置值小于允许的最小值，控制器按 1 设置。

返 回 值：如果设定参数值成功，set_vector_conspeed 返回 0，出错返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

注　　释：常矢量速度应设置为相对较小一些，以免在运动过程中丢步。对于快速插补运动，如：fast_line2、fast_line3 等来说，可用 set_vector_profile 来设置运动速度。

参　　见：set_vector_profile ，set_conspeed，set_profile

调用例子：set_vector_conspeed（1000）；

函 数 名：set_vector_profile

目　　的：用 set_vector_profile 来设置矢量梯形速度参数；

语　　法： set_vector_profile(double vec_vl , double vec_vh ,double vec_ad,double vec_dc);
vec_fl：矢量低速的速度值；
vec_fh：矢量高速的速度值；
vec_ad：矢量高速的加速度值；
vec_dc：矢量减速度值

描　　述：函数 set_vector_profile 为 fast_line2，fast_line3 等快速插补函数设置矢量梯形速度。这个函数不为 con_line2，con_line3 等常速插补函数设置运动速度。最大脉冲频率可设置为 4MHz，若设置值超过允许的最大值，控制器将按 4MHz 设置。最小脉冲频率可设置为 10 Hz，若设置值小于允许的最小值，控制器按 10 设置，加速值最小可设置为 20，低于该值按 20 处理。

返 回 值：如果调用成功，set_vector_profile 函数返回 0，在出错的情况下，返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：set_vector_conspeed，fast_line2，fast_line3

调用例子：set_vector_profile（1000，16000，10000，10000）；

函 数 名：set_s_curve

目　　的：用 set_s_curve 函数来设置轴的快速运动模式。

语　　法：int set_s_curve(int ch,int mode)

ch：轴号

mode：快速运动模式

0—梯形加减速模式

1—S 形加减速模式

描　　述：通过 set_s_curve 函数设置轴的快速运动的模式，在缺省情况下为 0，即梯形加减速。

返 回 值：如果调用成功，函数返回 0，在出错的情况下，返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：set_s_curve(1,1)；

函 数 名：set_s_section

目　　的：用 set_s_section 函数来设置轴的 S 形升降速的 S 段。

语　　法：int set_s_section(int ch,double accel_sec,double decel_sec)

ch：轴号

accel_sec：S 形升速的 S 段升速值, 不能大于设置的高速的 1/2。

decel_sec：S 形减速的 S 段减速值, 不能大于设置的高速的 1/2。

描　　述：对于快速运动方式，为了使升降速过程更为平稳，可以采用 S 形速度曲线。通过 set_s_section 函数设置轴的升降速值。升速时从初速到（初速+ accel_sec）之间为 S 形，从（高速- accel_sec）到高速之间为 S 形；降速时从高速到（高速- decel_sec）之间为 S 形，从（初速+ decel_sec）到初速之间为 S 形。其中的初速、高速由 set_profile 设置。

返 回 值：如果调用成功，函数返回 0，在出错的情况下，返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：set_s_section (1,100,200)；

函 数 名：set_abs_pos

目　　的：用于设置轴的运动起始绝对位置。

语　　法：int set_abs_pos（int ch，double pos）；

ch：所要设置的轴；

pos：所要设置的该轴的起始绝对位置；

描　　述：调用该函数可将当前绝对位置设置为某一个值，但从上一个位置到该位置之间不会产生轴的实际运动。调用该函数并将第二个参数设为 0 可实现 reset_pos()函数的功能。调用该函数时必须确保该轴运动已经停止，否则将引起绝对位置值的混乱。

返 回 值：如果函数调用成功，则返回值为 0；否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：set_abs_pos（1，1000）；/*将第 1 轴的当前位置设置为 1000*/

函 数 名：reset_pos

语　　法：int reset_pos（int ch）；

ch：被复位轴的轴号；

描　　述：函数 reset_pos 将指定轴的绝对位置和相对位置复位至 0，通常在轴的原点找到时调用，调用这个函数后，当前位置值变为 0，这以后，所有的绝对位置值均是相对于这一点的。调用该函数时必须确保该轴运动已经停止，否则将引起绝对位置值的混乱。一般来说，这个函数应在成功地执行回零运动后调用。

返 回 值：如果调用成功，reset_pos 返回 0，否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

注　　释：

参　　见：get_abs_pos

调用例子：int reset_pos（1）；

10.4 运动指令

按运动类型分类，主要有三种类型：点位运动、连续运动和回原点运动；按运动方式可分为独立运动和插补运动两种；按运动速度可分为常速运动和快速运动两种。为了描述方便，下面将运动指令分为独立运动和插补运动两

部分来说明。

10.4.1 独立运动函数

所谓独立运动指各轴的运动之间没有联动关系，可以是单轴运动，也可以是多轴同时按各自的速度运动。点位运动、连续运动和回原点运动都属于独立运动。独立运动指令的函数名格式为：X_YmoveZ，其中：

X：由 con 和 fast 替代，con 表示常速运动，fast 表示快速运动；

Y：由 p、v 和 h 替代，p 表示点位运动，v 表示连续运动，h 表示回原点运动；

move：为指令主体，表示该指令为运动指令；

Z：没有时为单轴运动，为 2 时表示两轴独立运动，为 3 时表示三轴独立运动，为 4 时表示四轴独立运动。

例如：con_vmove 为单轴的常速连续运动函数；con_pmove2 为两轴的常速点位运动函数；fast_hmove3 为三轴的快速回原点运动指令。

对于常速运动指令，运动速度由 set_conspeed 设定；对于快速运动指令，运动速度由 set_profile 设定。

（1）点位运动函数

点位运动是指被控轴以各自的速度分别移动指定的距离，在到达目标位置时自动停止。注意：在两轴或三轴的点位运动函数中，各轴同时开始运动，但不一定同时到达目标位置。在 MPC08D 函数库中共提供了 8 个点位运动指令函数：

表 10-4 点位运动函数

函数原型	说明
int con_pmove(int ch ,double step)	一个轴以常速做点位运动
int fast_pmove(int ch , double step)	一个轴以快速做点位运动
int con_pmove2(int ch1, double step1,int ch2, double step2)	两个轴以常速做点位运动
int fast_pmove2(int ch1, double step1,int ch2, double step2)	两个轴以快速做点位运动
int con_pmove3(int ch1, double step1,int ch2, double step2,int ch3,	三个轴以常速做点位运动

double step3)	
int fast_pmove3(int ch1, double step1,int ch2, double step2,int ch3, double step3)	三个轴以快速做点位运动
int con_pmove4(int ch1, double,step1,int ch2,double step2,int ch3, double step3,int ch4,double step4)	四个轴以常速做点位运动
int fast_pmove4(int ch1, double,step1,int ch2,double step2,int ch3, double step3,int ch4,double step4)	四个轴以快速做点位运动

其 中：

ch、ch1、ch2、ch3、ch4：被控轴的轴号；

step、step1、step2、step3、step4：表示被控轴从当前位置开始移动的距离，正数表示正方向，负数表示负方向，其单位为脉冲数。

调用例子：

con_pmove（1，-2000）；/*第一轴以其常速向负方向移动 2000 个脉冲的距离*/

fast_pmove2（2，5000，3，-1000）；/*第二轴以快速向正方向移动 5000 个脉冲的距离；第三轴以快速向负方向移动 1000 个脉冲的距离。*/

返 回 值：如果调用成功，这些函数返回 0，在出错情况下返回-1。

（2）连续运动函数

连续运动是指被控轴以各自的速度按给定的方向一直运动，直到碰到限位开关或调用制动函数才会停止。在 MPC08D 函数库中共提供了八个连续运动指令函数：

表 10-5 连续运动函数

函数原型	说明
int con_vmove(int ch ,int dir1)	一个轴以常速做连续运动
int fast_vmove(int ch , int dir1)	一个轴以快速做连续运动
int con_vmove2(int ch1, int dir1,int ch2, int dir2)	两个轴以常速做连续运动

int fast_vmove2(int ch1, int dir1,int ch2, int dir2)	两个轴以快速做连续运动
int con_vmove3(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3)	三个轴以常速做连续运动
int fast_vmove3(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3)	三个轴以快速做连续运动
int con_vmove4(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3,int ch4, int dir4)	四个轴以常速做连续运动
int fast_vmove4(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3,int ch4, int dir4)	四个轴以快速做连续运动

其　中：

ch、ch1、ch2、ch3、ch4：被控轴的轴号；

dir、dir1、dir2、dir3、dir4：表示被控轴的运动方向，+1 表示正方向；-1 表示负方向。

调用例子：

con_vmove（1，-1）；/*第一轴以其常速向负方向连续运动*/

fast_vmove2（2，1，3，-1）；/*第二轴快速向正方向连续运动；第三轴快速向负方向连续运动。*/

返 回 值：如果调用成功，这些函数返回 0，在出错情况下返回-1。

（3）回原点函数

回原点运动是指被控轴以各自的速度按给定的方向一直运动，直到碰到原点信号、限位开关或调用制动函数才会停止。在 MPC08D 函数库中共提供了八个回原点运动指令函数：

表 10-6 回原点运动函数

函数原型	说明
int con_hmove(int ch ,int dir1)	一个轴以常速做回原点运动
int fast_hmove(int ch , int dir1)	一个轴以快速做回原点运动
int con_hmove2(int ch1, int dir1,int ch2, int dir2)	两个轴以常速做回原点运动

int fast_hmove2(int ch1, int dir1,int ch2, int dir2)	两个轴以快速做回原点运动
int con_hmove3(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3)	三个轴以常速做回原点运动
int fast_hmove3(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3)	三个轴以快速做回原点运动
int con_hmove4(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3,int ch4, int dir4)	四个轴以常速做回原点运动
int fast_hmove4(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3,int ch4, int dir4)	四个轴以快速做回原点运动

其　中：

ch、ch1、ch2、ch3、ch4：被控轴的轴号；

dir、dir1、dir2、dir3、dir4：表示被控轴的运动方向，+1 表示正方向；-1 表示负方向。

调用例子：

con_hmove（1，-1）；/*第一轴以其常速向负方向作回原点运动*/

fast_hmove2（2，1，3，-1）；/*第二轴快速向正方向作回原点运动；第三轴快速向负方向作回原点运动。*/

返 回 值：如果调用成功，这些函数返回 0，在出错情况下返回-1。

注　　释：要成功地实现回原点运动，运动轴上应设有常开或常闭型原点开关（接近开关或传感器）。控制轴在回原点过程中，若先检测到有效的限位信号，控制轴将自动反向找原点。

10.4.2 插补运动函数

插补运动是指两轴或三轴按照一定的算法进行联动，被控轴同时启动，并同时到达目标位置。插补运动以矢量速度运行，矢量速度分为常矢量速度和梯形矢量速度。与插补运动有关的函数有：

（1）线性插补函数

线性插补运动是指两个轴或两个以上轴以矢量速度（常矢量速度或梯形

矢量速度）作线性联动，每个被控轴的运动速度为矢量速度在该轴上的分速度，各个被控轴同时启动，并同时到达目标位置。MPC08D 函数库中提供六个线性插补函数：

表 10-7 线性插补函数

函数原型	说明
int con_line2(int ch1,double pos1,int ch2, double pos2)	两个轴做常速直线运动
int fast_line2(int ch1, double pos1,int ch2 double pos2)	两个轴做快速直线运动
int con_line3(int ch1, double pos1,int ch2, double pos2,int ch3, double pos3)	三个轴做常速直线运动
int fast_line3(int ch1, double pos1,int ch2, double pos2,int ch3, double pos3)	三个轴做快速直线运动
int con_line4(int ch1, double pos1,int ch2, double pos2,int ch3, double pos3,int ch4, double pos4)	四个轴做常速直线运动
int fast_line4(int ch1, double pos1,int ch2, double pos2,int ch3, double pos3,int ch4, double pos4)	四个轴做快速直线运动

其 中：

ch1、ch2、ch3、ch4：被控轴的轴号。

pos1、pos2、pos3、pos4：表示被控轴从当前位置开始移动的距离，正数表示正方向；负数表示负方向，单位为脉冲数。

返回值：如果调用成功，这些函数返回 0，在出错情况下返回-1。

调用例子：

```
con_line2（1，-2000，3，1000）;
/*第一轴和第三轴以常矢量速度作线性插补运动,第一轴向负方向移动 2000 个脉冲的距离，同时第三轴向正向移动 1000 个脉冲的距离*/
fast_line3（2，5000，3，-1000，5，3000）;
/*第二轴、第三轴和第五轴以梯形矢量速度作线性插补运动，第二轴向正方向移动 5000 个脉冲的距离；第三轴向负方向移动 1000
```

个脉冲的距离；第五轴向正方向移动 3000 个脉冲的距离。*/

10.5 制动函数

在运动过程中，如果需要暂停或中止某个轴或某几个轴的运动，可以调用制动函数来完成。

表 10-9 制动函数

函数原型	说明
int sudden_stop(int ch)	立即运动方式下，立即制动一个运动轴
int sudden_stop2(int ch1,int ch2)	立即运动方式下，立即制动二个运动轴
int sudden_stop3(int ch1,int ch2,int ch3)	立即运动方式下，立即制动三个运动轴
int sudden_stop4(int ch1,int ch2,int ch3,int ch4)	立即运动方式下，立即制动四个运动轴
int decel_stop(int ch)	立即运动方式下，光滑制动一个运动轴
int decel_stop2(int ch1,int ch2)	立即运动方式下，光滑制动二个运动轴
int decel_stop3(int ch1,int ch2,int ch3)	立即运动方式下，光滑制动三个运动轴
int decel_stop4(int ch1,int ch2,int ch3,int ch4)	立即运动方式下，光滑制动四个运动轴
int move_pause(int ch)	立即运动方式下，暂停一个运动轴
int move_resume(int ch)	立即运动方式下，恢复一个运动轴的运动

函 数 名：sudden_stop，sudden_stop2，sudden_stop3，sudden_stop4

目　　的：立即运动模式下，立即制动轴的运动。

语　　法：int sudden_stop(int ch)

int sudden_stop2(int ch1,int ch2)

int sudden_stop3(int ch1,int ch2,int ch3)

int sudden_stop4(int ch1,int ch2,int ch3,int ch4)

ch、ch1、ch2、ch3、ch4：被控轴的轴号；

描　　述：立即制动函数（或称急停函数）对立即运动模式下所有类型的运

动都有效。sudden_stop 类型制动函数使被控轴立即中止运动，这个函数执行后，控制器立即停止向电机驱动器发送脉冲，使之停止运动。该函数通常在紧急停车时调用。对于常速运动方式（con_YmoveZ），只能调用急停函数进行制动。

返 回 值：调用成功返回 0，否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

调用例子：sudden _stop2（1，4）；/*立即制动第一、四轴*/

函 数 名：decel_stop，decel_stop2，decel_stop3，decel_stop4

目　　的：立即运动模式下，光滑制动轴的运动。

语　　法：int decel_stop (int ch)
int decel_stop2 (int ch1,int ch2)
int decel_stop3(int ch1,int ch2,int ch3)
int decel_stop4(int ch1,int ch2,int ch3,int ch4)
ch、ch1、ch2、ch3、ch4：被控轴的轴号；

描　　述：decel_stop 类型的制动函数一般用于梯形速度运动方式（fast_YmoveZ），它可以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。一般在运动过程需要停止时应调用 decel 类型制动函数，以便能够光滑地中止快速运动（如：fast_hmove、fast_vmove、fast_pmove2 等），以免发生过冲现象。对于常速运动方式（con_YmoveZ），这类制动函数无效。

返 回 值：调用成功返回 0，否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

调用例子：decel_stop（2）；/*光滑制动第二轴*/

函 数 名：move_pause

目　　的：立即运动模式下，暂停运动过程。

语　　法：int move_pause（int ch）；
ch：需要暂停轴的轴号；

描　　述：在立即运动过程中，若需要暂停一个轴的运动，可调用该函数，之后再调用 move_resume 恢复继续运行。
对于快速直线插补运动、快速点位运动、快速连续运动、快速回原点运动等，调用“move_pause”函数，控制轴即以设置的梯形速度减速直到停止；调用“move_resume”后，控制轴又以梯形

速度加速到高速。

对应常速直线插补运动、常速点位运动、常速连续运动、常速回原点运动，调用“move_pause”函数，控制轴立即停止运动；调用“move_resume”后，控制轴立即以常速度运动。

返 回 值：如果调用成功，则 move_pause 返回值为 0，否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

调用例子：move_pause（1）；

函 数 名：move_resume

目　　的：立即运动模式下，恢复被暂停的运动过程。

语　　法：int move_resume（int ch）；

ch：需要恢复运动轴的轴号；

描　　述：在立即运动过程中，若要在中途暂停，可调用 move_pause 函数，之后再调用 move_resume 恢复继续运行。

对于快速直线插补运动、快速点位运动、快速连续运动、快速回原点运动等，调用“move_pause”函数，控制轴即以设置的梯形速度减速直到停止；调用“move_resume”后，控制轴又以梯形速度加速到高速。

对应常速直线插补运动、常速点位运动、常速连续运动、常速回原点运动，调用“move_pause”函数，控制轴立即停止运动；调用“move_resume”后，控制轴立即以常速度运动。

返 回 值：如果调用成功，则 move_resume 返回值为 0，否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

调用例子：move_resume（1）；

10.6 数字 I/O 操作函数

表 10-10 数字 IO 函数

函数原型	说明
int checkin_byte(int cardno)	读取板卡所有通用输入口状态
int checkin_bit(int cardno,int bitno)	读取板卡某位通用输入口状态

int outport_byte(int cardno,int bytedata)	写板卡所有通用输出口
int outport_bit(int cardno,int bitno,int status)	写板卡某位通用输出口
int check_sfr(int cardno)	读取所以外部限位、原点、报警和 Z 脉冲信号状态
int check_sfr_bit(int cardno,int bitno)	读取外部限位、原点、报警和 Z 脉冲信号中某个信号的状态

函 数 名：checkin_byte

目　　的：读取板卡所有的通用输入状态。

语　　法：int checkin_byte(int cardno)；

cardno：卡编号，即用户设置的板卡本地 ID 号，取值范围从 1 到卡最大编号。

描　　述：通过该函数可以读入所有 24 个通用输入口的状态。接线见《MPC08D User.pdf》中“通用输入、输出的连接方法”一节内容。

返 回 值：返回输入口的状态，返回值的 D0 到 D23 位对应 24 个输入口，该位为 1 表示输入口处于 ON 状态，为 0 表示该输入口处于 OFF 状态；如果出错则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：checkin_bit

函 数 名：checkin_bit

目　　的：读取板卡扩展的某一位通用输入状态。

语　　法：int checkin_bit(int cardno,int bitno)；

cardno：卡编号，即用户设置的板卡本地 ID 号，取值范围从 1 到卡最大编号；

bitno：表示第几位，取值范围为 1~26。

描　　述：通过该函数可以读入某一个输入口的状态。接线方式见《MPC08D User.pdf》中“通用输入、输出的连接方法”一节内容。

返 回 值：返回输入口的状态，返回值为 1 表示输入口处于 ON 状态，为 0

表示该输入口处于 OFF 状态；如果出错则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：checkin_byte

函 数 名：outport_byte

目　　的：设置板卡通用输出口状态。

语　　法：int outport_byte(int cardno,int bytedata)；

cardno：卡编号，即用户设置的板卡本地 ID 号，取值范围从 1 到卡最大编号；

bytedata：状态字节，各位对应各输出口；

描　　述：MPC08D 卡通过扩展提供最多 26 个通用光电隔离输出口，供用户使用。其中通用输出 1～通用输出 10 从 MPC08D 板卡 62 芯电缆输出，通用输出 11～通用输出 26 从扩展线输出。所有通用输出都有 100mA 的驱动能力。通过该函数可以设置这 26 个输出口的状态。接线见《MPC08D User.pdf》中“通用输入、输出的连接方法”一节内容。

返 回 值：正确设置返回 0；如果出错则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：outport_bit

函 数 名：outport_bit

目　　的：设置板卡某个通用输出口开关量状态。

语　　法：int outport_bit(int cardno,int bitno,int status)；

cardno：卡编号，即用户设置的板卡本地 ID 号，取值范围从 1 到卡最大编号。

bitno：表示第几个输出口，取值范围为 1~26。

Status：设置的状态（1：ON；0：OFF）。

描　　述：MPC08D 卡通过扩展提供最多 26 个通用光电隔离输出口，供用户使用。其中通用输出 1～通用输出 10 从 MPC08D 板卡 62 芯电缆输出，通用输出 11～通用输出 26 从扩展线输出。所有通用输出都有 100mA 的驱动能力。通过该函数可以设置这 26 个输出口的状态。接线见《MPC08D User.pdf》中“通用输入、输出的连接方法”一节内容。

返 回 值：正确设置返回 0；如果出错则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：outport_byte

函 数 名：check_sfr

目　　的：读取板卡输入口信息，包括轴外部报警、限位、原点和板卡报警信号等。

语　　法：int check_sfr(int cardno);

cardno：卡编号，即用户设置的板卡本地 ID 号，取值范围从 1 到卡最大编号。

调用例子：check_sfr（1）；/*读取 1 号卡的外部报警、限位及原点信号*/

描　　述：MPC08D 运动控制器有一个寄存器保存板卡 21 个专用输入口状态，包括各轴专用输入口（如限位、原点、报警、Z 脉冲等）。check_sfr 函数可读取这个寄存器内容，了解各专用输入口状态。如果将某轴的外部报警、限位、原点信号设置为无效，则对应的专用信号输入端口可作为通用输入口使用，寄存器每一位的定义见下表 10-11，使用函数 check_sfr 读取板卡所有专用输入口状态。

返 回 值：返回外部开关量信号的状态，相应位为 1 表示输入口处于高电平状态，为 0 表示该输入口处于低电平状态；如果出错则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：表 10-11 专用寄存器定义

状态位	专用信号	说明
D20	ALM	ALM 无效时作为通用输入口
D19	Z4	Z4 不能做通用输入使用
D18	ALM4	ALM4 无效时作为通用输入口
D17	ORG4	ORG4 无效时作为通用输入口
D16	EL4-	EL4-无效时作为通用输入口
D15	EL4+	EL4+无效时作为通用输入口
D14	Z3	Z3 不能做通用输入使用
D13	ALM3	ALM3 无效时作为通用输入口
D12	ORG3	ORG3 无效时作为通用输入口
D11	EL3-	EL3-无效时作为通用输入口
D10	EL3+	EL3+无效时作为通用输入口
D9	Z2	Z2 不能做通用输入使用

D8	ALM2	ALM2 无效时作为通用输入口
D7	ORG2	ORG2 无效时作为通用输入口
D6	EL2-	EL2-无效时作为通用输入口
D5	EL2+	EL2+无效时作为通用输入口
D4	Z1	Z1 不能做通用输入使用
D3	ALM1	ALM1 无效时作为通用输入口
D2	ORG1	ORG1 无效时作为通用输入口
D1	EL1-	EL1-无效时作为通用输入口
D0	EL1+	EL1+无效时作为通用输入口

函 数 名：check_sfr_bit

目　　的：读取板卡输入口信息，包括包括各轴专用输入口和板卡报警输入口。

语　　法：int check_sfr_bit(int cardno,int bitno)；

cardno：卡编号，即用户设置的板卡本地 ID 号，取值范围从 1 到卡最大编号。

bitno：表示第几个输入口，取值范围为 1~21。

调用例子：check_sfr_bit（1,1）；/*读取 1 号卡的外部限位信号*/

描　　述：MPC08D 运动控制器有一个寄存器保存板卡 21 个专用输入口状态，包括各轴专用输入口（如限位、原点、报警、Z 脉冲等）。check_sfr 函数可读取这个寄存器内容，了解各专用输入口状态。如果将某轴的外部报警、限位、原点信号设置为无效，则对应的专用信号输入端口可作为通用输入口使用，寄存器每一位的定义见下表 10-11，使用函数 check_sfr_bit 可读取某个端口状态。

返 回 值：返回外部开关量信号的状态，返回值的为 1 表示该输入口处于高电平状态，为 0 表示该输入口处于低电平状态；如果出错则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：enable_sd，enable_el，enable_org

10.7 特殊功能函数

MPC08D 提供反向间隙补偿、动态改变目标位置和可掉电保护数据区

读写功能。各个功能的接口如下。

10.7.1 反向间隙补偿

表 10-12 反向间隙补偿函数

函数原型	说明
int set_backlash（int ch， double backlash）	设置由于机构换向形成间隙的补偿值
int start_backlash（int ch）	开始间隙补偿
int end_backlash（int ch）	终止间隙补偿

函 数 名：set_backlash，start_backlash，end_backlash

目　　的：用 set_backlash 设置补偿由于机构换向形成间隙的补偿值。
用 start_backlash 启动反向间隙补偿。
用 end_backlash 停止反向补偿补偿。

语　　法：int set_backlash（int ch，double backlash）；
int start_backlash（int ch）；
int end_backlash（int ch）
ch：控制轴编号。
backlash：由于机构换向形成的间隙值，单位为脉冲数，必须大于等于 0。

描　　述：函数 set_backlash 设置一个补偿值，以便消除由于机构换向形成的位置误差。调用函数 start_backlash 后，开始对控制轴进行反向间隙补偿。终止控制轴的反向间隙补偿调用 end_backlash。set_backlash 函数仅是设置补偿值，真正的补偿值到调用函数 start_backlash 才起作用。函数 set_backlash 应在调用 start_backlash 前调用，否则系统采用缺省补偿值（缺省值为 20 个脉冲）。

返 回 值：如果设置成功，set_backlash、set_backlash 和 end_backlash 返回 0，否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：set_backlash（1，12）；
start_backlash（1）；
end_backlash（1）

10.7.2 动态改变目标位置

表 10-13 动态改变目标位置函数

函数原型	说明
int change_pos(int ch, double pos)	运动中动态设置目标位置

函 数 名：change_pos

目　　的：在相对位置模式下使用函数 **change_pos** 来动态改变目标位置。单轴点位运动过程中，在常速模式或梯形速度模式下，若用户发现发出的运动指令终点位置需要改变，可在该点位运动结束前或者结束后调用“**change_pos**”动态改变终点位置。系统将自动按新的终点位置运动。该终点位置以上一条用户发出的运动指令（**fast_pmove**、**con_pmove**）的起点为起点。可多次调用该函数来改变目标位置。

语　　法：**int change_pos(int ch, double pos)**;

参数说明：

ch ：轴号；

pos：新的相对目标位置

描　　述：运动中动态设置目标位置。

返 回 值：调用正确返回 0，错误返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：change_pos (1，1000);

10.7.3 可掉电保护数据区读写功能

表 10-14 可掉电保护数据区读写函数

函数原型	说明
int write_password_flash(int cardno, int no, long data,long password)	带密码保护功能的数据区写函数
read_password_flash(int cardno, int no, long *data,long password)	带密码保护功能的数据区读函数
clear_password_flash(int cardno,	擦除带密码保护功能的数据区

long password)	
write_flash(int cardno,int piece, int no, long data)	一般数据区的写函数
read_flash(int cardno,int piece,int no,long *data)	一般数据区的读函数
clear_flash(int cardno, int piece)	擦除一般数据区

函 数 名：write_password_flash

目　　的：用于对带密码保护功能的数据区进行写操作。**对数据区进行写操作之前必须对该片区域进行擦除操作，否则无法写入待写数据。**擦除操作对整片有效，对指定片区的区域进行擦除，使得该区域的 64K 字节空间所有位为“1”。写操作对 4 个字节的连续地址进行操作。

语　　法：**int write_password_flash(int cardno, int no, long data,long password)**;

参数说明：

cardno ：卡号；

no：写地址的编号，范围为0-16383，其中16383固定为读写密码存放区；

data：待写数据；

password：读写校验密码；即16383号地址区域保存的数据值，当写地址编号为16383时也校验该密码；

描　　述：用于对带密码保护功能的数据区进行写操作。

返 回 值：0　--写成功返回

-1 --参数错误导致的写失败，返回

-2 --由于校验密码错误导致的写失败,返回

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：clear_password_flash

调用例子：write_password_flash (1，1000，0xfffe0，0xffffff);

函 数 名：read_password_flash

目　　的：用于对带密码保护功能的数据区进行读操作。读操作对 4 个字节的连续地址进行操作，对带密码保护功能的数据区进行读操作时

需要校验读写密码，负责无法返回正确的数值。

语　　法：**int read_password_flash (int cardno, int no, long *data,long password)**;

参数说明：

cardno ：卡号；

no：读地址的编号，范围为0-16382，其中16383号地址固定为读写密码存放区，不准许外部接口进行读操作；

data：存放读取数据；

password：读写校验密码；即16383号地址区域保存的数据值。

描　　述：用于对带密码保护功能的数据区进行读操作。

返 回 值：0　--读成功返回

-1 --参数错误导致的读失败，返回

-2 --由于校验密码错误导致的读失败,返回

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：read _password_flash (1，1000，&data，0xffffff);

函 数 名：clear_password_flash

目　　的：用于对带密码保护功能的数据区进行擦除操作。**对数据区进行写操作之前必须对该片区域进行擦除操作，否则无法写入待写数据。**擦除操作对整片有效，对指定片区的区域进行擦除，使得该区域的 64K 字节空间所有位为“1”。

语　　法：**clear_password_flash(int cardno, long password)**;

参数说明：

cardno ：卡号；

password：读写校验密码；即16383号地址区域保存的数据值，当写地址编号为16383时也校验该密码；

描　　述：用于对带密码保护功能的数据区进行擦除操作。

返 回 值：0　--擦除成功返回

-1 --参数错误导致的擦除失败，返回

-2 --由于校验密码错误导致的擦除失败,返回

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：write_password_flash

调用例子：clear_password_flash (1,0xffffff)；

函 数 名：write_flash

目　　的：用于对一般数据区进行写操作。**对数据区进行写操作之前必须对该片区域进行擦除操作，否则无法写入待写数据。**擦除操作对整片有效，对指定片区的区域进行擦除，使得该区域的 64K 字节空间所有位为“1”。写操作对 4 个字节的连续地址进行操作。

语　　法：**int write_flash(int cardno,int piece, int no, long data)**；

参数说明：

cardno ：卡号；

piece -- 片区号，范围1-15，每片区域的大小为64K；

no -- 写地址的编号，范围为0-16383；

data--待写数据；

描　　述：用于对一般数据区进行写操作。

返 回 值：0 --写成功返回

-1 --参数错误导致的写失败，返回

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：clear_flash

调用例子：write_flash (1，2，1000，0xfffe0)；

函 数 名：read_flash

目　　的：用于对一般数据区进行读操作。读操作对 4 个字节的连续地址进行操作。

语　　法：**int read_flash(int cardno,int piece,int no,long *data)**；

参数说明：

cardno ：卡号；

piece ：片区号，范围1-15，每片区域的大小为64K；

no：读地址的编号，范围为0-16383；

data：存放读取数据。

描　　述：用于对一般数据区进行读操作。

返 回 值：0　--读成功返回

-1 --参数错误导致的读失败，返回

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：read _flash (1，2，1000，&data)；

函 数 名：clear_flash

目　　的：用于一般数据区进行擦除操作。**对数据区进行写操作之前必须对该片区域进行擦除操作，否则无法写入待写数据。**擦除操作对整片有效，对指定首地址的区域进行擦除，使得该区域的 64K 字节空间所有位为“1”。

语　　法：**int clear_flash(int cardno, int piece)**；

参数说明：

cardno ：卡号；

piece ：片区号，范围1-15；

描　　述：用于一般数据区进行擦除操作。

返 回 值：0　--擦除成功返回

-1 --参数错误导致的擦除失败，返回

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：write_flash

调用例子：clear_flash (1,0xffffff)；

10.8 位置和状态设置函数

表 10-15 查询函数

函数原型	说明
int get_max_axe(void)	读取运动控制器总轴数
int get_board_num(void)	读取板卡数
int get_axe(int board_no)	读取板卡上轴数
int check_IC(int cardno)	查询用户设置的控制器的本地 ID 号
int get_abs_pos（int ch，double *pos）	返回一个轴的绝对位置值
double get_conspeed(int ch)	读取各轴常速度
int get_profile(int ch,double	读取各轴梯形速度

*vl,double *vh,double *ad,double *dc)	
double get_vector_conspeed(void)	读取矢量常速度
int get_vector_profile(double *vec_vl,double *vec_vh,double *vec_ad,double *vec_dc)	读取矢量梯形速度
double get_rate(int ch);	取得轴当前速度
int get_cur_dir (int ch)	返回一个轴的当前运动方向
int check_status（int ch）	检查一个轴的状态值
int check_done（int ch）	检测一个轴的运动是否完成
int check_limit（int ch）	检测一个轴指定的限位开关是否闭合
int check_home(int ch)	检查一个轴是否已经到达原点开关位置
int check_alarm(int ch)	检查一个轴的外部报警信号
int check_card_alarm(int cardno)	检查板卡的外部报警信号
int get_done_source(int ch,long *src)	检测一个轴的停止原因

函 数 名：get_max_axe

目　　的：get_max_axe 用于读取总的控制轴数。

语　　法： int get_max_axe（void）;

返 回 值：get_max_axe 返回总控制轴数。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：

```
int max_axe_num;
max_axe_num=get_max_axe（）;
```

函 数 名：get_board_num

目　　的：get_board_num 用于读取计算机内安装的运动控制器数。

语　　法： int get_board_num（void）;

返 回 值：get_board_num 返回总板卡数。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：

```
int card_num;
card_num=get_board_num（）;
```

函 数 名：get_axe

目　　的：get_axe 用于读取板卡上的轴数。

语　　法：int get_axe（int board_no）;

返 回 值：get_axe 返回板卡上的轴数。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：

```
int axe_num;
axe_num=get_axe（1）；//读取第一张卡的轴数
```

函 数 名：check_IC

目　　的：用于查询卡的本地 ID 号。

语　　法：int check_IC（int cardno）;

cardno：卡号，该参数与前面接口函数中板卡本地 ID 号不一样，仅表示计算机内板卡的顺序号，取值范围从 1 到最大卡数。该函数一般用于计算机中只安装有一张控制卡时，读取其本地 ID 号。

描　　述：该函数可以查询运动控制器的本地 ID 号。

返 回 值：返回当前运动控制器的 ID 号。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：int ic=check_IC（1）;

函 数 名：get_abs_pos

目　　的：用 get_abs_pos 读取一个相对于初始位置或原点位置的绝对位置。

语　　法：int get_abs_pos（int ch，double *abs_pos）;

ch：读取位置的轴号；

abs_pos：一个指向绝对位置的双精度指针；

描　　述：函数 get_abs_pos 获取指定轴的当前绝对位置，如果执行过回原点运动，并调用了 reset_pos 函数，那么这个绝对位置是相对于原点位置的；如果没有执行过 reset_pos，那么这个绝对位置是相对于开机时的位置。

返 回 值：如果调用成功，get_abs_pos 返回 0 值，在出错情况下返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

调用例子：temp=get_abs_pos（1，&abs_pos）；

函 数 名：get_conspeed

目　　的：用 get_conspeed 函数来获取某个轴所设置的常速度。

语　　法：double get_conspeed（int ch）；

ch：控制轴的编号。

描　　述：用 get_conspeed 函数来获取指定轴所设置的常速度。

返 回 值：函数 get_conspeed 返回指定轴的常速度值，出错时返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

注　　释：

参　　见：

调用例子：double speed;

speed=get_conspeed（2）；

函 数 名：get_profile

目　　的：用 get_profile 来读取梯形速度的各参数值。

语　　法：int get_profile(int ch,double *vl,double *vh,double *ad,double *dc)

double *ls：指向起始低度的指针；

double *hs:：指向目标高速的指针；

double *accel：指向加速度值的指针；

double *dc：指向减速度值的指针；

描　　述：函数 get_profile 通过指针返回一个轴的梯形速度的低速、高速和加、减速度值。

返 回 值：如果调用成功，get_profile 返回 0 值，否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：get_profile（3，&ls，&hs，&accel，&dec）；

函 数 名：get_vector_conspeed

目　　的：get_vector_conspeed 函数来读取常速方式下的矢量速度。

语　　法：double get_vector_conspeed（void）。

描　　述：函数 get_vector_conspeed 读取下列二轴或多轴插补运动函数的矢量速度：con_line2、con_line3、con_line4 等。

返 回 值：如果调用成功，返回读去的矢量速度。

系　　统：WINDOWS 2000、WINDOWS XP

注　　释：

参　　见：set_conspeed，set_profile

调用例子：vec_conspeed= get_vector_conspeed（）；

函 数 名：get_vector_profile

目　　的：用 get_vector_profile 来获取矢量梯形速度参数值；

语　　法：int get_vector_profile(double *vec_vl,double *vec_vh,double *vec_ad,double *vec_dc)；

*vec_fl：指向矢量低速的指针；

*vec_fh：指向矢量高速的指针；

*vec_ad：指向矢量加速度的指针；

*vec_dc：指向矢量减速度的指针。

描　　述：函数 get_vector_profile 读取 fast_line2，fast_line3 等快速插补函数的矢量梯形速度。

返 回 值：如果调用成功，get_vector_profile 函数返回 0，在出错的情况下，返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：get_vector_profile（&vec_fl，&vec_fh，&vec_ad，&vec_dc）；

函 数 名：get_rate

目　　的：用 get_rate 函数来获取当前某个轴的实际运动速度。

语　　法：double get_rate（int ch）；

ch：控制轴编号；

描　　述：函数 get_rate 读取控制轴当前的实际运行速度。

返 回 值：函数 get_rate 返回指定轴的当前运行速度，单位：每秒脉冲数（pps），函数调用出错返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：get_profile

调用例子：double speed;

speed=get_rate（2）;

函 数 名：get_cur_dir

目　　的：用于获取轴的当前运动方向。

语　　法：int get_cur_dir（int ch）；

ch：所要查询的轴；

返 回 值：如果函数调用失败，则返回值为-2；否则返回-1 表示当前运动方向负向，返回 1 则表示当前运动方向正向，0 表示运动停止。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：get_cur_dir（1）；/*获取第 1 轴的当前运动方向*/

函 数 名：check_status

目　　的：读取一个轴的当前状态。

语　　法：int check_status（int ch）;

ch：所读取状态的轴号。

描　　述：函数 check_status 读取指定轴的状态。MPC08D 控制器每个轴都有 1 个 32 位（双字）的状态值，用于查询轴的工作状态。轴状态字各位的含义见表 6-2。只有在调用“enable_org”、“enable_limit”、“enable_alm”、“enable_card_alm”等指令使相应专用信号使能，才能返回正确的专用输入口状态。

返 回 值：如果调用成功，check_status 返回指定轴的状态值，在出错时返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

调用例子：ch_status=check_status（2）;

函 数 名：check_done

目　　的：用 check_done 函数来检查指定轴的运动是否已经完毕。

语　　法：int check_done（int ch）；
　　ch：所检查的轴号。
描　　述：函数 check_done 检查指定轴是在运动中还是在静止状态。
返 回 值：如果指定轴正在运动状态，check_done 返回 1，如果指定轴正在静止状态，check_done 返回 0，函数调用失败返回-1。
系　　统：WINDOWS 2000、WINDOWS XP
参　　见：

函 数 名：check_limit
目　　的：用 check_limit 函数来检查一个轴限位信号是否有效。
语　　法：int check_limit（int ch）；
　　ch：所检查的轴号；
描　　述：MPC08D 运动控制器每轴配置有两个限位开关输入口，分别为正限位信号输入口和负限位信号输入口。函数 check_limit 用于检测指定轴的限位开关状态，返回指定轴的限位信号是否有效。只有在调用 “enable_limit”指令使相应专用信号使能，才能返回正确的专用输入口状态。
返 回 值：如果 check_limit 返回 1 表示正向限位信号有效，返回-1 表示负向限位信号有效，返回 0 则表示正负限位均无效，返回 2 表示正向限位和负向限位同时有效，若调用出错则返回-3。
系　　统：WINDOWS 2000、WINDOWS XP
调用例子：status=check_limit（1）；

函 数 名：check_home
目　　的：用 check_home 函数来检查一个轴原点信号是否有效。
语　　法：int check_home（int ch）；
　　ch：所检查的轴号。
描　　述：MPC08D 运动控制器每轴配置有一个原点开关输入口。函数 check_home 用于检测指定轴的原点信号是否有效，只有在调用“enable_org”指令使相应专用信号使能，才能返回正确的专用输入口状态。
返 回 值：如果 check_home 返回 1 表示原点信号有效，返回 0 则表示原点开关无效，若调用出错则返回-3。

系　　统：WINDOWS 2000、WINDOWS XP

调用例子：status=check_home（1）；

函 数 名：check_alarm

目　　的：用 check_alarm 函数来检查外部报警信号是否有效。

语　　法：int check_alarm（int ch）；

ch：轴号。

描　　述：MPC08D 运动控制器每个轴均有一个报警开关输入口。函数 check_alarm 用于检测板指定轴的报警开关状态，返回是否有有效的报警信号输入板卡。只有在调用“enable_alm”指令使相应专用信号使能，才能返回正确的专用输入口状态。

返 回 值：如果 check_alarm 返回 1 表示报警信号有效，返回 0 则表示报警信号无效，若调用出错则返回-3。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

函 数 名：check_card_alarm

目　　的：用 check_card_alarm 函数来检查板卡的卡报警信号是否有效。

语　　法：int check_card_alarm（int cardno）；

cardno：所检查的板卡号。

描　　述：MPC08D 运动控制器板卡配置一个报警信号输入口。函数 check_card_alarm 用于检测板卡的报警信号状态。只有在调用“enable_card_alm”指令使相应专用信号使能，才能返回正确的专用输入口状态。

返 回 值：如果 check_card_alarm 返回 1 表示板卡报警信号有效，返回 0 则表示板卡报警信号无效，若调用出错则返回-3。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

10.9 错误代码处理函数

运动控制器可返回最近 10 个错误状态，以便用户了解系统状况，详细的错误代码含义见表 9-2。错误代码操作函数有 3 个，如表 10-16 所示。

表 10-16 错误代码处理函数

函数原型	说明
int get_err(int index,int* data)	获取错误代码
int get_last_err()	获取最后一次错误代码
int reset_err()	清除所有错误

函 数 名：get_err

目　　的：用于查询获取之前指令执行过程中产生的最近十次错误的错误代码。

语　　法：int get_err(int index,int* data)；

index：存储错误代码的索引号，最近出现的错误代码索引号为 1，以此倒推到 10。

data：保存返回的错误代码。

描　　述：函数 get_err 查询获取之前指令执行过程中产生的最近十次错误的错误代码。通过返回值，再对照错误代码表 11-2，用户可以快速的找到程序错误原因。

返 回 值：最近第 index 次错误的错误代码，-1 表示 Index 参数超出了 1 到 10 的范围或系统没有错误。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：表 9-2 错误代码含义

调用例子：int err;

get_err(2,&err); /*获取最近第 2 次产生的错误代码*/；

函 数 名：get_last_err

目　　的：用于获取之前指令执行过程中产生的最近一次错误的错误代码。

语　　法：int get_last_err();

返 回 值：最近一次错误的错误代码，0 表示没有错误。

调用例子：int err;

err=get_last_err();

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：表 9-2 错误代码含义

函 数 名：reset_err

目　　的：用于清除最近十次错误代码。调用该函数之后，再调用 get_last_err()或 get_err()函数均将返回 0。

语　　法：int reset_err();

返 回 值：成功返回 0，失败返回-1

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：get_last_err()，get_err()

调用例子：int err;
err=reset_err();

10.10 控制器版本获取函数

表 10-17 版本读取函数

函数原型	说明
int get_lib_ver(long* major,long *minor1,long *minor2)	查询函数库的版本
int get_sys_ver(long* major,long *minor1,long *minor2)	查询驱动程序的版本
int get_card_ver(long cardno,long *type,long* major,long *minor1,long *minor2)	查询板卡的版本

函 数 名：get_lib_ver

目　　的：用于查询函数库的版本。

语　　法：int get_lib_ver(long* major,long *minor1,long *minor2)；
long * major：指向函数库主版本号的指针。
long * minor1：指向函数库次版本号 1 的指针。
long * minor2：指向函数库次版本号 2 的指针。

调用例子：get_lib_ver(&major, &minor1, &minor2)；

描　　述：该函数可以查询运动控制器函数库的版本号，函数库版本必须与驱动程序版本及板卡的固件版本匹配。

返 回 值：0。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

函 数 名：get_sys_ver

目　　的：用于查询驱动程序的版本。

语　　法：int get_sys_ver(long* major,long *minor1,long *minor2)；

long * major：指向驱动程序主版本号的指针。

long * minor1：指向驱动程序次版本号 1 的指针。

long * minor2：指向驱动程序次版本号 2 的指针。

调用例子：get_sys_ver(&major, &minor1, &minor2)；

描　　述：该函数可以查询运动控制器驱动程序的版本号，驱动程序版本必须与函数库版本及板卡固件版本匹配。

返 回 值：成功调用返回 0，否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

函 数 名：get_card_ver

目　　的：用于查询板卡的版本。

语　　法：int get_card_ver(long cardno,long * type,long * major , long *minor1 ,long *minor2)；

long cardno：卡编号，即用户设置的板卡本地 ID 号，取值范围从 1 到卡最大编号；

long * type：卡类型号，MPC08D 为 2。

long * major：返回的主版本号。

long * minor1：返回的次版本号 1。

long * minor2：返回的次版本号 2。

调用例子：get_card_ver(1，&type, &major, &minor1, &minor2)；

描　　述：该函数可以查询运动控制器的类型和版本号，板卡固件版本必须与函数库版本及驱动程序版本匹配。

返 回 值：成功调用返回 0，否则返回-1。

系　　统：WINDOWS 2000、WINDOWS XP

参　　见：

11 函数索引

12 附　录

12.1 P62-05 转接板引脚定义

P62-05 转接板集成了 MPC08D 所有专用和部分通用输入输出信号的外部引脚。其组成如下所示，单位：mm。其中安装孔径：ϕ3mm。

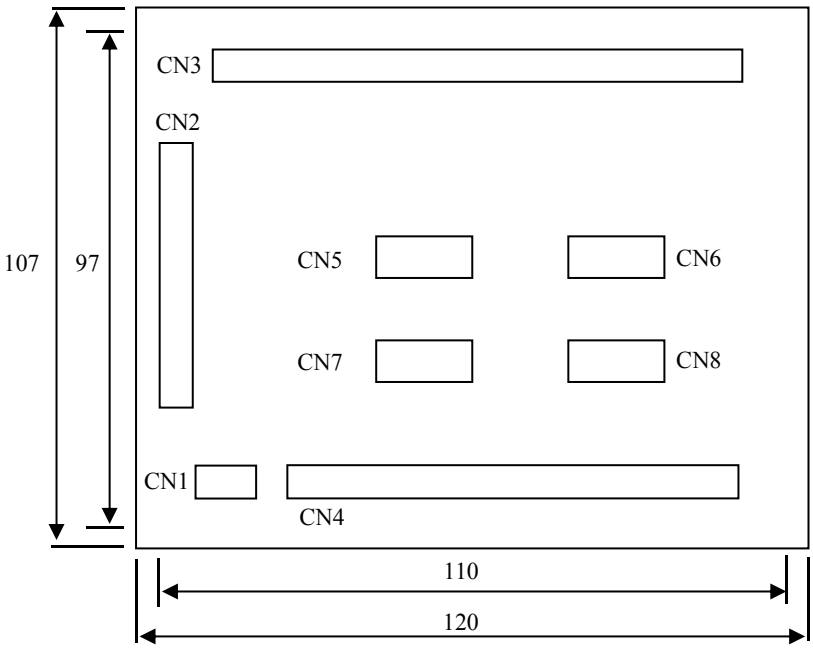

图 12-1 P62-05 示意图

表 12-1 P62-05 接口定义

接口端子	功能
CN1	24V 开关电源接口
CN2	连接运动控制器的 DB62 接口
CN3	限位、原点、板卡报警及 8 路通用输入接口（注：该处端子+24V 和 GND 为板卡输出 24 电源，可为专用输入信号的传感器供电。）
CN4	10 路通用输出接口
CN5	轴 1 脉冲、方向及轴报警、Z 脉冲信号接口
CN6	轴 2 脉冲、方向及轴报警、Z 脉冲信号接口
CN7	轴 3 脉冲、方向及轴报警、Z 脉冲信号接口
CN8	轴 4 脉冲、方向及轴报警、Z 脉冲信号接口

12.2 P37-05 转接板引脚定义

使用 MPC08D 运动控制器配合 P62-05 转接板时，若需要较多通用 IO 接口时，就要使用 P37-05 连接信号转接线 C40　IO 扩展线。P37-05 引脚定义如下。

表 12-2　P37-05 引脚定义

P37-05 转接板引脚	37 芯电缆引脚	名称	说 明
P19	19	IN9	通用输入 9
P37	37	IN10	通用输入 10
P18	18	IN11	通用输入 11
P36	36	IN12	通用输入 12
P17	17	IN13	通用输入 13
P35	35	IN14	通用输入 14
P16	16	IN15	通用输入 15
P34	34	IN16	通用输入 16
P15	15	IN17	通用输入 17
P33	33	IN18	通用输入 18
P14	14	IN19	通用输入 19
P32	32	IN20	通用输入 20
P13	13	IN21	通用输入 21
P31	31	IN22	通用输入 22
P12	12	IN23	通用输入 23
P30	30	IN24	通用输入 24
P11	11	OUT11	通用输出 11
P29	29	OUT12	通用输出 12
P10	10	OUT13	通用输出 13
P28	28	OUT14	通用输出 14
P9	9	OUT15	通用输出 15
P27	27	OUT16	通用输出 16
P8	8	OUT17	通用输出 17
P26	26	OUT18	通用输出 18

P7	7	DCV24	输入+24V（7、25、2 和 20 任接一路即可。）
P25	25	DCV24	输入+24V
P6	6	OUT19	通用输出 19
P24	24	OUT20	通用输出 20
P5	5	OUT21	通用输出 21
P23	23	OUT22	通用输出 22
P4	4	OUT23	通用输出 23
P22	22	OUT24	通用输出 24
P3	3	OUT25	通用输出 25
P21	21	OUT26	通用输出 26
P2	2	DCV24	输入+24V
P20	20	DCV24	输入+24V
P1	1	OGND	24V 电源地