

leetro

MPC08D Motion Controller

P
r
o
g
r
a
m
m

in g M a n u al

(version 0.4)

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

copyright statement

All rights reserved by
Legrand Automation
Technology Co.

(hereinafter referred to as Legrand Automation) reserves the right to modify the products and product specifications and other documents in this manual without prior notice.

Legrand Automation shall not be liable for any direct, indirect, incidental or consequential damages or liabilities arising out of the improper use of this manual or this product.

Legrand Automation Inc. owns the patents, copyrights and other intellectual property rights to this product and its software. Unauthorized direct or indirect reproduction, manufacture, processing, or use of this product and its related parts is prohibited.



preamble

Thank you for purchasing the MPC08D Motion Controller! The MPC08D is a cost-effective general-purpose controller developed by our company. This programmer's manual describes the use of MPC08D motion commands. Please understand the functions of the MPC08D before use.



safety warning

Observe the following warnings to avoid injury to the operator and others and to prevent damage to the machine.

- ◆ The "DANGER" and "WARNING" symbols below are labeled according

 Danger ous	Indicates a potentially dangerous situation that, if not avoided, will result in death or serious injury.
 Warnin gs	Indicates a potentially hazardous situation that, if not avoided, will result in minor or moderate injury, or material damage.

- ◆ The following symbols indicate what is prohibited or what must be observed.

	This symbol indicates that the operation is prohibited.
	This symbol indicates an operation that requires attention.

Summary of general security

Please review the following safety precautions to avoid injury and prevent damage to this product or any product connected to it. To avoid potential hazards, follow the detailed instructions for using this product.

Use the correct power cord. Use a power cord that meets national standards.

Connect and disconnect correctly. Connect the control card output to the adapter board first, then connect the motor and driver to the adapter board, and finally turn on the power. When disconnecting, turn off the external power supply first, then disconnect the motor and driver from the adapter board, and finally disconnect the control card from the adapter board.

Do not operate when there is a suspected malfunction. If you suspect damage to this product, have it inspected by qualified service personnel.

Do not operate in wet/humid environments. Do not operate in explosive atmospheres. Keep product surfaces clean and dry.

Prevent electrostatic damage. Electrostatic discharge (ESD) can cause damage to components in the motion controller and its accessories. To prevent ESD, handle the controller components carefully and do not touch the components on the controller. Do not place the controller on surfaces that may generate static electricity. Transport and store the controller in bags or containers that are protected from static electricity.

About guarantees

Warranty Time

The warranty period for products purchased at the specified location is 1 year.

Warranty Coverage

If a malfunction occurs due to our responsibility during the above warranty period, we provide repairs without charge.

The following coverage is not covered under warranty:

- For malfunctions caused by inappropriate environments or inappropriate use as documented in the instruction manual and other manuals.

- Accidental malfunction of this product caused by the user's device, control software, etc.
- Malfunctions caused by customer modifications to this product.
- Malfunctions caused by external major causes such as fire, earthquake and other natural disasters.

Product Applications

This product is designed and manufactured for general industrial applications. Uses that are beyond what is expected and have a significant impact on human life or property are not within the scope of service of the product.

Contact Information

Address: Block B, Block 8, Danyi Incubation Park, No.1, South Second Road, Fengjiawan Science Park, Hi-Tech Zone, Chengdu, Sichuan, P.R.China.

Seat (610041)

Chengdu Lechuang Automation

Technology Co., Ltd. company website:

<http://www.lectro.com>

Technical Support:

✧ Tel: (028) 85149977

✧ FAX: (028) 85187774

table of contents

table of contents

1 Use of libraries.....	1
1.1 Developing a Motion Control System for Windows.....	1
1.1.1 Developing Visual Basic Control Programs	2
1.1.2 Developing Visual C++ Control Programs.....	4
1.2 MPC08D Software Upgrade	6
2 Motion Controller Initialization.....	7
2.1 Controller initialization.....	7
2.1.1 Command List.....	7
2.1.2 Function Description	7
2.1.3 give an example	8
2.2 Control axis initialization.....	8
2.2.1 Command List.....	8
2.2.2 Function Description	9
2.3 Dedicated input signal setting	9
2.3.1 Command List.....	9
2.3.2 Function Description	10
3 Motion Functions.....	14
3.1 Speed Setting Function	14
3.1.1 Command List.....	14
3.1.2 Function Description	14
3.2 Point Motion Functions.....	16
3.2.1 Normal Motion Mode.....	16
3.2.2 Trapezoidal Variable Speed Motion Mode.....	17
3.2.3 S-Curve Variable Speed Motion Pattern.....	19
3.3 Continuous Motion Functions	20
3.3.1 Command List.....	20
3.3.2 Functional Description	20
3.3.3 Example Programs	21
3.4 Return-to-origin kinematics.....	21
3.4.1 Command List.....	21
3.4.2 Function Description	22
3.4.3 Example	23
3.5 Linear Interpolation Motion Functions.....	23
3.5.1 Command List.....	23

3.5.2 Function Description	24
4 Brake Function	25
4.1 Brake Function	25
4.2 Function Description	25
5 Position Setting and Reading Functions	27
5.1 Position Setting Functions	27
5.1.1 Command List	27
5.1.2 Function Description	27
5.2 Position Reading Functions	27
5.2.1 Command List	27
5.2.2 Function Description	28
6 State Handling Functions	29
6.1 Motion Status Query Functions	29
6.1.1 Command List	29
6.1.2 Function Description	29
6.2 Dedicated Input Detection Functions	30
6.2.1 Command List	30
6.2.2 Function Description	31
7 Generic IO Manipulation Functions	33
7.1 Digital IO Port Operation Functions	33
7.1.1 Command List	33
7.1.2 Function Description	33
8 Other Functions	37
8.1 Reverse Gap Processing	37
8.1.1 Command List	37
8.1.2 Function Description	37
8.1.3 Programs	38
8.2 Dynamically changing the target position	38
8.2.1 Command List	38
8.2.2 Function Description	39
8.2.3 Example Programs	39
8.3 Power-down protectable data area read/write function	39
8.3.1 Command List	39
8.3.2 Functional Description	40
8.1.3 Programs	41

8.4 Board number and version reading	42
8.4.1 Command List	42
8.4.2 Functional Description	42
8.5 Software Limit Handling	43
8.5.1 Command List	43
8.5.2 Function Description	43
8.5.3 routines	44
9 Error Codes and Handling Functions	45
9.1 Error Code Handling Functions	45
9.1.1 Command List	45
9.1.2 Functional Description	45
10 Function Description	47
10.1 Controller Initialization Functions	47
10.2 Property Setting Functions	48
10.3 Motion Parameter Setting Functions	55
10.4 Motion Command	60
10.4.1 Independent Motion Functions	61
10.4.2 Interpolated Motion Functions	64
10.5 Brake Function	66
10.6 Digital I/O Operation Functions	68
10.7 Special Function Functions	72
10.7.1 Reverse Gap Compensation	73
10.7.2 Dynamically changing the target position	74
10.7.3 Power-down protectable data area read/write function	74
10.8 Position and Status Setting Functions	78
10.9 Error Code Handling Functions	85
10.10 Controller Version Get Function	87
11 Function Index	89
12 appendix	Attachment
93	
12.1 P62-05 Adapter Board Pin Definitions	93
12.2 P37-05 Adapter Board Pin Definitions	94

1 Use of libraries

1.1 Development of a motion control system for Windows

Using the MPC08D Dynamic Link Library (DLL) developers can quickly develop motion control systems under the Windows platform. the MPC08D DLL is a standard Windows 32-bit DLL, and the selected development tool should support Windows standard 32-bit DLL calls.

After running the installation program from the product's companion CD-ROM, it will be installed in the installation directory (the default installation directory is
\\The folder "MPC08D" is automatically created under (Program Files) and its directory tree is shown below:

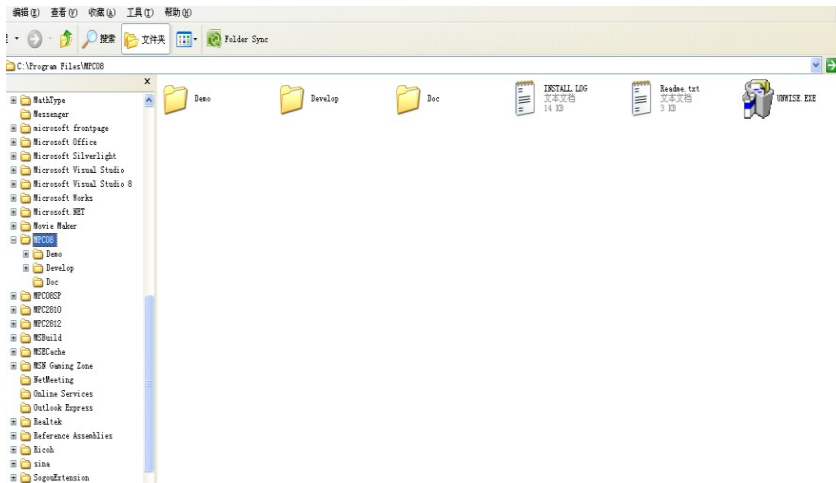


Figure 1-1 MPC08D Catalog Structure

- Y The "Demo" directory contains sample programs, where:
 - The "VBDemo" directory contains "Demo1" and "Demo2" are two VB examples, the source code is provided;
 - The "VCDemo" directory contains 4 sample programs, of which "Demo1" and "Demo2" provide the source code, "Demo1" is a VC static

load dynamic link
library example,
"Demo2" is a VC
dynamic load dynamic
link library example.

Demo1 is an example of VC static loading dynamic link library,
"Demo2" is an example of VC dynamic loading dynamic link
library. "Demo3" does not provide source code, with
function testing. "Demo4" does not provide source code, with
function libraries, drivers and firmware version reading
function of each board.

- ✂ The "Develop" directory contains the drivers and libraries for the MPC08D:
- The "Common" folder contains the drivers, libraries, etc. for the MPC08D;

- The "VB" folder contains the module files that need to be added when developing VB applications;
 - In the "VC" folder are the files needed to load the dynamic link library dynamically: "LoadDll.cpp" and "LoadDll.h", and the files needed to load the dynamic link library statically: "MPC08D.h" and "MPCC08D.lib". dynamic link library, you need to use the file "MPC08D.h" and "MPCC08D.lib".
- ✂ The "Doc" directory contains the user manual and programming manual for the MPC08D. A typical user-written motion process processing flow is shown in Figure 1-2.

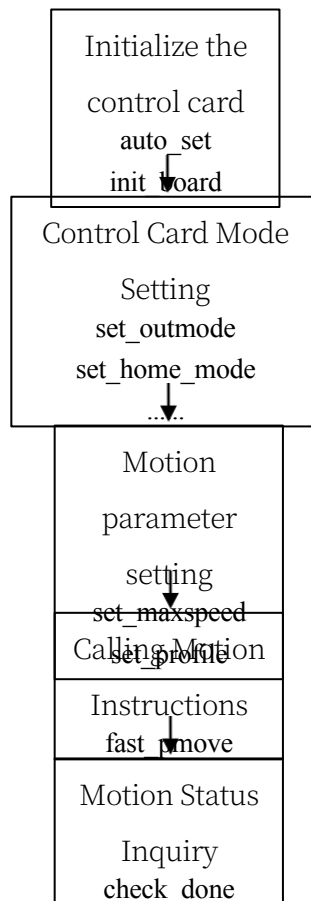


Figure 1-2 Typical Flowchart of Motion Instructions

The following describes how to utilize two commonly used development tools, Microsoft Visual Basic and Microsoft Visual C++ develops motion control programs for Windows-based platforms.

1.1.1 Developing Visual Basic Control Programs

(i) General

To develop Windows-based motion control programs, users can use VB5.0 or higher.

Version. A simple control program can be quickly developed by following the steps below.

1. Install the **MPC08D** driver and library;
2. Start **Visual Basic** and create a new project;
3. Copy the installed dynamic link library "**MPC08D.dll**" and function sound file "**MPC08D.bas**" to the project file;
4. Select the "**Add Module**" menu item under the "**Project**" menu to add the "**MPC08D.bas**" file to the project;
5. Call the motion function from within the application.

(ii) Dynamic link library function call methods

Calling a function in a dynamic link library (DLL) in VB should be a two-part job:

- (1) function declaration

The declaration of each function in a dynamic link library (DLL) in VB is already included in the

The **MPC08D.bas** file can be found on the **MPC08D** Motion Controller software installation disk.

\Develop\VB folder to find, the user just need to add the file into the VB project.

- (2) function call

If the return value of the calling function is null or no return value is required, it is called as follows:

```
con_pmove 1, 2000
```

maybe

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

To get the return value of a function, call it as follows:

```
Dim rtn As Long
```

```
rtn=con_pmove(1,2000)
```

(iii) Demonstration of the use of sample programs

There are two demo programs of motion control system developed under VB6.0 in the folder \Demo\VB Demo of MPC08D Motion

Controller software installation disk. Users can follow the steps below to compile and run the examples, and after familiarizing themselves with the corresponding programming methods, they can develop their own motion control systems according to their needs.

- (1) Follow the **MPC08D** software installation procedure for proper installation.
- (2) After the installation is complete, you can find the installation disk \Demo\VBDemo\ folder under the Demo1 or Demo2 folder.

- (3) Start the **VB6.0** Integrated Environment and open the project.
- (4) Make sure the board is properly set up and inserted into the computer.
- (5) Compile the project to generate an EXE file.
- (6) Run the generated EXE file.

1.1.2 Developing Visual C++ control programs

(i) Overview

Users can use **VC5.0** or higher to develop motion control systems on the **Windows** platform. there are two ways to call dynamic link libraries in **Visual C++**, using slightly different files.

(ii) Dynamic link library function call methods

(1) implicit call

The steps for an implicit call are as follows:

- (a) Install the **MPC08D** driver and library;
- (b) Start **Visual C++** and create a new project;
- (c) Copy the installed dynamic link libraries "**MPC08D.dll**" and "**MPC08D.lib**" and the function declaration file "**MPC08D.h**" into the project file. file;
- (d) Select the "**Settings**" menu item under the "**Project**" menu;
- (e) Switch to the "**Link**" tab and enter the file name "**MPC08D.lib**" in the "**Object/library modules**" field;
- (f) Add the declaration file "**MPC08D.h**" of the function library header file to the application program;
- (g) Call the motion function from within the application.

For details, see the demo example: **\Demo\VC Demo\Demo1**.

(2) explicit call

The explicit invocation method requires calling **Windows API** functions to load and release the dynamic link library. The method is as follows:

- (a) Call the **Windows API** function **LoadLibrary()** to load the DLL dynamically;

- (b) Call the Windows API function `GetProcAddress()` to get a pointer to the function in the **DLL** that will be called;
- (c) Use function pointers to call functions in the **DLL** to accomplish the corresponding functions;

(e) At the end of the program or when the functions in the DLL are no longer used, call the **Windows API** function `FreeLibrary()` to release the dynamic link library.

The **MPC08D** software has encapsulated the commonly used **DLL** functions into the class **CLoadDll**, and provides the source code for this class. This class contains member functions with the same names and parameters as the Motion Instruction Library functions. The source code can be found in the "**Develop\VC**" folder in the **MPC08D** installation directory, and the file names are **LoadDll.cpp** and **LoadDll.h**. The developer can add it to the project, add the object of this class in the appropriate place of the program, and call the functions in the **DLL** through the corresponding member functions. functions in the **DLL** through the corresponding member functions. For details, please refer to the demo example: **Demo\VC Demo\Demo2**.

The calling steps are as follows:

- (a) Install the **MPC08D** driver and library;
- (b) Start **Visual C++** and create a new project;
- (c) Copy the installed dynamic link libraries "**MPC08D.dll**" and "**MPC08D.lib**" and the function declaration file "**LoadDll.h**" into the project, and copy "**LoadDll.cpp**" to the project file;
- (d) Select the "**Files**" item in the "**Add To Project**" submenu under the "**Project**" menu;
- (e) Add "**LoadDll.h**" and "**LoadDll.cpp**" to the project;
- (f) Generate an object of **CLoadDll** in the application that calls the motion functions.

The above two methods are standard methods for calling dynamic link library functions in **VC**, if you want to get more specific calling methods and help, please refer to **Microsoft Visual Studio** Development Documentation **MSDN** or related **VC** reference books in the corresponding part of the content. If you need to share **MPC08D** with **mpc2810** or **mpc08B** controller, you should use explicit call.

(iii) Demonstration of the use of sample programs

The MPC08D Motion Controller software installation disk

\Demo\VCDemo\ folder contains four sample programs, of which "Demo1" and "Demo2" provides the source code, "Demo1" is VC static loading dynamic link library example, "Demo2" is VC dynamic loading dynamic link library example. Demo1" is an example of VC static loading dynamic link library, "Demo2" is an example of VC dynamic loading dynamic link library. Demo3" does not provide source code, with function testing. Demo4" does not provide the source code, with function libraries, drivers and firmware version reading function of each board. Users can compile and run the examples according to the following steps. After familiarizing with the corresponding programming methods, users can develop their own motion control system according to their needs. The steps are as follows:

- (1) Follow the MPC08D software installation procedure for proper installation.

- (2) After the installation is completed, you can find the corresponding folder under the '"Demo\VC Demo\'' folder in the installation directory.
- (3) Start the VC6.0 integrated environment and open the project files (demo1.dsw, demo2.dsw,).
(etc.).
- (4) Make sure the board is properly set up and inserted into the computer.
- (5) Compile and link the project to generate an EXE file.
- (6) Run the generated EXE file.

1.2 MPC08D Software Upgrade

Please visit our website (<http://www.leetro.com>) frequently to download the latest version of the driver and function library. The new version of the function library will keep the compatibility with the existing functions of the old version of the function library and add new functions as needed. Please consult your dealer or technical support department before upgrading. If you get a set of the latest installation program, you can follow the following method to upgrade your old library.

Row upgrades:

- (1) Close all running programs associated with the MPC08D;
- (2) Uninstall the original installer;
- (3) Run the new installer;
- (4) If Visual Basic 6.0 is used for development, copy the installed dynamic link library "MPC08D.dll" and the function declaration file "MPC08D.bas" into the project file, and recompile to generate the .EXE file.
- (5) If Visual C++6.0 is used for development, the installed dynamic link libraries "MPC08D.dll" and "MPC08D.lib" and the function declaration file "MPC08D.h" are copied into the project file to generate the .EXE file. "MPC08D.h" are copied into the project file

and recompiled to generate the .EXE file. For explicit call, copy the installed dynamic link libraries "MPC08D.dll", "MPC08D.lib" and function declaration files "LoadDll.h", "LoadDll.h", "LoadDll.h", "LoadDll.h", "LoadDll.h" and "LoadDll.h" into the project file. "MPC08D.dll", "MPC08D.lib" and function declaration files "LoadDll.h", "LoadDll.h", "LoadDll.h", "LoadDll.h", "LoadDll.h" and "LoadDll.h" are copied into the project file and recompiled to generate the .EXE file.

2 Motion Controller Initialization

2.1 Controller initialization

2.1.1 Instruction List

There are two controller initialization functions, if you want to use the MPC08D functions, you must first call the function `auto_set`, `init_board`, complete the initialization of the controller before calling other functions. The initialization functions are shown in Table 2-1.

Table 2-1 Controller Initialization Functions

function (math.)	return value	clarification
<code>auto_set</code>	≥ 0 : number of axes -1: Error	Auto-detect and auto-set controllers
<code>init_board</code>	≥ 0 : number of cards -1: Error	Initialization of controller hardware and software

2.1.2 Functional Description

(1) *auto_set* Description

The `auto_set` function must first be called to auto-detect the controller and perform a correct return of the number of axes on the card. Detect whether the user set the card number correctly.

(2) *init_board* Description

The `init_board` function must be called to detect the function library, driver, and controller firmware version number, and to initialize all registers of the controller to their default state. `init_board` should be called after `auto_set`. Execute the function correctly to return the number of controller cards installed in the computer. The initial

state of each control axis after initialization is:

- Pulse output mode: pulse/direction;
- Constant speed: 2000pps;
- Trapezoidal speed: initial speed 2000pps, high speed 8000pps, acceleration and deceleration 80000ppss;
- Vector constant velocity: 2000pps;
- Vector trapezoidal speed: 2000pps initial speed, 8000pps high speed, 80,000ppss acceleration and deceleration;
- Axis return to home position motion mode: only detect the home position proximity switch signal, and stop the motion immediately when there is a home position signal;
- Positive limit, negative limit, home position and alarm enable of each axis are active and high.

The board alarm signal enable is active high.

2.1.3 give an example

```
InitMPC08D()
{
    int Rtn.

    Rtn = auto_set();           //auto_set
    if(Rtn <= 0 )
    {
        Return error code
    }
    else
    {
        Auto setup successful, get axis number
    }

    Rtn = init_board();        //initialize
    board if(Rtn <= 0 )
    {
        Return error code
    }
    else
    {
        Initialization successful, get card count
    }
    .....
}
```

2.2 Control Axis Initialization

2.2.1 Instruction List

There are two control axis initialization functions, as shown in
Table 2-2.

Table 2-2 Control Axis Initialization Functions

function (math.)	return value	clarification
------------------	--------------	---------------

MPC08D Motion Controller

set_outmode	0: Correct	Setting the pulse output mode of an axis
-------------	------------	--

	-1: Error	
set_home_mode	0: Correct -1: Error	Setting the home mode of an axis

2.2.2 Functional Description

(1) *set_outmode* Description

The motion controller provides two pulse output modes: "pulse+direction" and "positive/negative pulse", and the default pulse output mode is "pulse+direction". The default pulse output mode is "pulse+direction". Call the `set_outmode` instruction to set the pulse output mode to "positive/negative pulse".

This function is called after the `init_board` function.

(2) *set_home_mode* Description

The motion controller provides four home return modes:

0: Detection of the home position proximity switch signal The axis stops moving immediately;

1: Stop motion as soon as an encoder Z-phase pulse signal is detected;

2: In trapezoidal speed mode, when the home position proximity switch signal is detected to be valid, the control axis gradually decelerates and stops at the acceleration set in the rapid motion mode;

3: In the trapezoidal speed mode, when the home signal is valid, the control axis gradually decelerates to a low speed according to the acceleration set in the rapid motion mode until the Z pulse is valid and stops the motion immediately.

Note that after returning to the home position, the position reset function "`reset_pos`" should be called to set the current position of the control axis to the home coordinate.

This function is called after the `init_board` function.

2.3 Dedicated input signal setting

2.3.1 Instruction List

There are eight functions related to the setting of dedicated input signal parameters, as shown in the table below. These functions are generally

The `init_board` function is called after the `init_board` function to initialize the control system dedicated input function.

Table 2-3 Dedicated Input Signal Parameter Setting Functions

function (math.)	return value	clarification
<code>enable_alm</code>	0: Correct -1: Error	Enable/disable external alarm signals for axes (default is enable) (able to)
<code>enable_el</code>	0: Correct -1: Error	Enable/disable external limit signals for axes (default is enable) (able to)

<code>enable_org</code>	0: Correct -1: Error	Enable/disable the external home signal of the axis (default is enable) (able to)
<code>enable_card_alm</code>	0: Correct -1: Error	Enable/disable the external alarm signal of the board (default is enable) (able to)
<code>set_alm_logic</code>	0: Correct -1: Error	Sets the effective level of the external alarm signal for the axis (default is (active high))
<code>set_el_logic</code>	0: Correct -1: Error	Sets the effective level of the external limit signal for the axis (default is (active high))
<code>set_org_logic</code>	0: Correct -1: Error	Sets the effective level of the external home signal for the axis (default is (active high))
<code>set_card_alm_logic</code>	0: Correct -1: Error	Set the effective level of the external alarm signal of the board (default is (active high))

2.3.2 Functional Description

(1) *enable_alm* Description

The motion controller provides an alarm signal input interface for each control axis, which is valid only for the motion of that axis. By default, the axis alarm switches are in the enable state, active high. At this time, each axis alarm switch should be connected to the corresponding axis alarm switch input pin and ~~on~~ corresponding to the "ALM" pin of the DB9 terminal of the adapter board) on the adapter board. Please refer to the section "Connection Method of Dedicated Inputs" in the "MPC08D User.doc" for the wiring method. When the alarm signal of an axis is triggered, the motion controller will automatically stop the motion of that axis.

If the control system does not use the axis alarm signal, call the "enable_alm" instruction to set the corresponding parameter to 0, so that this pin can be set as a general-purpose input port. The system saves its status in a special register. In this case, you can get the status of each bit of the register by the instruction "check_sfr_bit".

Table 2-4 Axis Alarm Signal Locations in Dedicated Input Signal Registers

MPC08D Motion Controller

status bit	Dedicated signals	clarification
18	ALM4	ALM 4 is used as a general-purpose input when invalid.
13	ALM3	ALM 3 is used as a general-purpose input when it is invalid.
8	ALM2	ALM 2 is used as a general-purpose input when it is invalid.
3	ALM1	General-purpose input when ALM 1 is invalid

(2) *enable_el* Description

The motion controller provides positive and negative limit signal input interfaces for each control axis. By default, the limit switches are in the enable state, active high. At this time, the limit switches of each axis should be connected to the corresponding limit signal input pins and OGND (corresponding to the EL+/EL- pins of the adapter board) on the adapter board. Wiring

Please refer to the section "Connection Method for Dedicated Inputs" in the MPC08D User.doc. When the limit switch of an axis is triggered, the motion controller will stop the motion in that direction immediately to protect the system safety.

If the control system does not use the limit signal, call the `"enable_el"` instruction to set the corresponding parameter to 0, so that this pin can be set as a general-purpose input port. The system saves its status in a special register. In this case, you can get the status of each bit of the register by the instruction `"check_sfr_bit"`. The following table shows the correspondence of the limit signals in the dedicated input register.

Table 2-5 Limit Signal Positions in Dedicated Input Signal Registers

status bit	Dedicated signals	clarification
16	EL4-	EL4 - Invalid as a general-purpose input port
15	EL4+	Used as a general-purpose input when EL4+ is invalid.
11	EL3-	EL3 - Invalid as a general-purpose input port
10	EL3+	Used as a general-purpose input when EL3+ is invalid.
6	EL2-	EL2 - Invalid as general-purpose input port
5	EL2+	Used as a general-purpose input when EL2+ is invalid.
1	EL1-	EL1 - Invalid as general-purpose input port
0	EL1+	Used as a general-purpose input when EL1+ is invalid.

(3) *enable_org* Description

The motion controller provides a home signal input interface for each control axis. By default, the home switch is in the enable state, active high. In this case, each axis home switch should be connected to the corresponding home signal input pin and OGND (corresponding to the **ORG** pin of the adapter board) on the adapter board. For wiring,

MPC08D User.doc, section "Connection Method of Dedicated Inputs". If the home switch is triggered when an axis is moving back to the home position, the motion controller will automatically stop the motion.

If the control system does not use the home signal, call the "enable_org" instruction to set the corresponding parameter to 0, so that this pin can be set as a general-purpose input port. The system saves its status in a special register. In this case, you can get the status of each bit of the register by the instruction "check_sfr_bit". The following table shows the correspondence of the origin signal in the dedicated input register.

Table 2-6 Home Signal Locations in Dedicated Input Signal Registers

status bit	Dedicated signals	clarification
17	ORG4	ORG4 is used as a general-purpose input port when it is invalid.
12	ORG3	ORG3 is used as a general-purpose input port when it is invalid.
7	ORG2	When ORG2 is invalid, it is used as a general-purpose input port.
2	ORG1	When ORG1 is invalid, it is used as a general-purpose input port.

(4) *enable_card_alm* Description

The motion controller provides a board alarm signal input interface. By default, the alarm switch is in the enable state, active high. In this case, the alarm switch should be connected to the corresponding alarm signal input pin and **GN**(corresponding to the **ALM** pin of the adapter board) on the adapter board. Please refer to "MPC08D User.doc" for the connection method of the dedicated inputs. If the board alarm switch is triggered when the Motion Controller is in motion, the Motion Controller will automatically stop the motion of all axes.

If the control system does not use the alarm signal of the card, call the "**enable_card_alm**" instruction to set the corresponding parameter to 0, so that this pin can be set as a general-purpose input port. The system saves its status in a special register. In this case, you can get the status of each bit of the register by the instruction "**check_sfr_bit**". The following table shows the relationship between the alarm signals in the dedicated input register.

Table 2-7 Card Alarm Signal Locations in Dedicated Input Signal Registers

status bit	Dedicated signals	clarification
20	ALM	Used as a general-purpose input when card ALM is invalid.

(5) **set_alm_logic** Description

The default axis alarm switch of the motion controller is a normally closed switch, i.e., when each axis is in the normal working state, its axis alarm switch input is low; when the axis alarm switch input is high, the alarm state of the corresponding axis will be triggered.

The trigger level of the axis alarm switch can be set by the parameter "**set_alm_logic**". Setting the corresponding parameter to 1 means that the alarm switch of the corresponding axis is triggered at a high level; setting the parameter to 0 means that the alarm switch of the corresponding axis is triggered at a low level.

(6) **set_el_logic** Description

The default limit switches of the motion controller are normally closed switches, i.e., when each axis is in the normal working state, its limit switch input is low; when the limit switch input is high, it will

trigger the limit state of the corresponding axis.

The trigger level of the limit switches can be set by the parameter **"set_el_logic"**. Setting the corresponding parameter to 1 means that the limit switches of the corresponding axes are triggered at a high level; setting the parameter to 0 means that the limit switches of the corresponding axes are triggered at a low level.

(7) *set_org_logic* Description

The default home switch of the motion controller is a normally closed switch, i.e., when each axis is in the normal working state, its home switch input is low; when the home switch input is high, the home state of the corresponding axis will be triggered.

The trigger level of the home switch can be set by the parameter **"set_org_logic"**. Setting the corresponding parameter to 1 means that the home switch of the corresponding axis is triggered at a high level; setting the parameter to 0 means that the home switch of the corresponding axis is triggered at a low level.

(8) *set_card_alm_logic* Description

The default board alarm switch of the motion controller is a normally closed switch, i.e., when the board is in normal operation, its alarm switch input is low; when the board alarm switch input is high, the alarm state of the corresponding board will be triggered.

The parameter of `"set_card_alm_logic"` can be used to set the trigger level of the alarm switch of the card. Setting the corresponding parameter to 1 means that the alarm switch of the corresponding card is triggered at a high level; setting the parameter to 0 means that the alarm switch of the corresponding card is triggered at a low level.

3 function of motion (math.)

3.1 Speed Setting Function

3.1.1 Instruction List

There are five velocity setting functions, as shown in Table 3-1. The velocity, acceleration and deceleration parameters in these functions are programmed in pulses.

Table 3-1 Speed Setting Functions

function (math.)	return value	clarification
set_maxspeed	0: Correct -1: Error	Setting the maximum speed of an axis
set_conspeed	0: Correct -1: Error	Setting the speed parameter of the axis in the normal speed motion mode
set_profile	0: Correct -1: Error	Setting the speed parameter of the axis in the rapid traverse mode
set_vector_conspeed	0: Correct -1: Error	Setting the vector constant velocity parameter for constant velocity motion mode
set_vector_profile	0: Correct -1: Error	Setting the vector trapezoidal speed parameter in rapid motion mode

3.1.2 Functional Description

(1) *set_maxspeed* Description

The user sets the maximum output pulse frequency of an axis, which is mainly used to determine the pulse output multiplier of the Motion Controller. The output pulse frequency of the MPC08D Motion Controller is controlled by two variables: the pulse resolution and the multiplier, and the product of the two is the output pulse

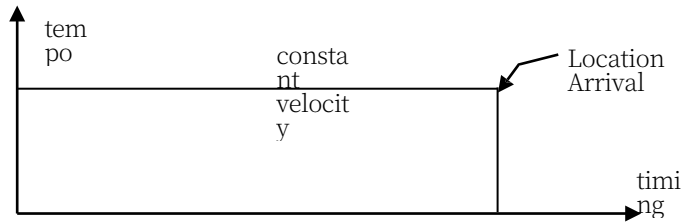
frequency. Due to the motion controller internal multiplier register length is limited to 13 bits (maximum value of 8191) if you want to achieve the maximum output pulse frequency of 2400KHz, the pulse resolution of $(2400000/8191) = 293\text{Hz}$, if the actual use of the pulse frequency of 100Hz, it is clear that the MPC08D device can only output a resolution of the pulse frequency (ie) 293Hz) To solve this problem, call `set_maxspeed` to set the maximum output pulse frequency to be achieved, such as `set_maxspeed (1, 100)` After setting the pulse resolution will be reset to $(100/8191) = 0.012\text{Hz}$, which will improve the speed resolution. However, at this time, the maximum output pulse frequency of the motion controller can only reach 100 Hz. Note: The maximum resolution of the MPC08D can reach 0.01, and at this time, the maximum output pulse frequency of the controller can only reach 82 Hz.

By default, the `init_board` function sets all axes to their maximum speed, which is 2000000

(2MHz) When using, you can set it according to the actual output speed to get better speed accuracy. Maximum output pulse frequency range: 81.91~2,000,000 Hz.

(2) *set_conspeed* Description

The *set_conspeed* function sets the speed of an axis in the normal speed mode. The speed curve for constant speed motion is shown in



the following figure:

Figure 3-1 Velocity Curve for Normal Velocity Mode of Motion

This speed may not match the final pulse output speed of the motion controller. This is because the output pulse frequency is controlled by two variables: pulse resolution and multiplier, and the larger the multiplier, the larger the error. Setting the maximum speed as v_{max} , the output multiplier as *multi*, the actual pulse output speed as v_{act} , and the set constant speed as v_{con} , the formula is as follows:

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

If this function is called multiple times, the last value set is valid and remains valid until the next change. The maximum pulse frequency can be set to 2000000 Hz and the minimum pulse frequency can be Set to 0.2 Hz.

(3) *set_profile* Description

The *set_profile* function sets the low speed (starting speed) high speed (target speed) and acceleration/deceleration values of an axis in the rapid motion mode (in the trapezoidal speed mode, the deceleration value can be not equal to the acceleration value) The velocity profile for the rapid motion mode is shown in the figure.

This speed may not correspond to the final pulse output speed of the motion controller, due to the

The same as `set_conspped`.

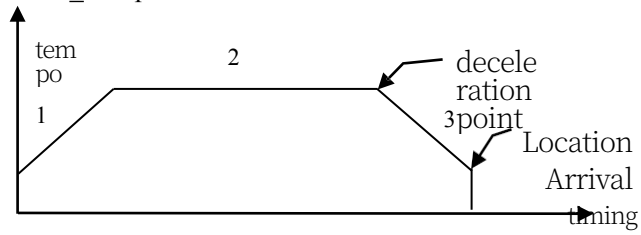


Figure 3-2 Velocity Profile for Rapid Motion Mode

1: Acceleration segment, the starting speed is accelerated to the target speed at the set acceleration;

- 2: High speed segment, maintain target speed until movement reaches deceleration point;
- 3: Deceleration section: the target speed is decelerated to the starting speed by the set acceleration;

In this fast motion mode, the pulse frequency is 10Hz minimum for low speed values, 2000000Hz maximum for high speed values, and 20 minimum for acceleration values.

(4) *set_vector_conspeed* Description

The *set_vector_conspeed* function sets the vector speed in the constant speed mode, which will be used in the constant speed motion of linear interpolation and circular interpolation in two or more axes. The vector speed is not directly related to the maximum speed set by *set_maxspeed*, i.e. the speed multiplier determined by *set_maxspeed* function does not affect the interpolation motion. The maximum pulse frequency can be set to 2000000 Hz and the minimum pulse frequency can be set to 1Hz.

(5) *set_vector_profile* Description

The *set_vector_profile* function sets the vector low speed to (start speed) vector high speed, and vector acceleration/deceleration in the fast motion mode (in the trapezoidal speed mode, the vector deceleration value can be not equal to the vector acceleration value) These vector values will be used in two and more axes of fast linear interpolation motion. The vector speed is not directly related to the maximum speed set by *set_maxspeed*, i.e. the speed multiplier determined by the *set_maxspeed* function does not affect the interpolation motion. The minimum pulse frequency can be set to 10Hz for low speed values, the maximum pulse frequency can be set to 2000000Hz for high speed values, and the minimum acceleration value can be set to 20.

3.2 Point Motion Functions

3.2.1 Normal Motion Mode

3.2.1.1 Instruction List

There are 6 normal speed mode point motion speed settings and motion functions, as shown in Table 3-2.

Table 3-2 Motion Functions for Constant Velocity Points

function (math.)	return value	clarification
set_maxspeed	0: Correct -1: Error	Setting the maximum speed of an axis
set_conspeed	0: Correct -1: Error	Setting the speed parameter of the axis in the normal speed mode. criticize (i.e. enumerate shortcomings)
con_pmove	0: Correct -1: Error	1 axis of relative positional point movement at constant velocity
con_pmove2	0: Correct -1: Error	2 axes move relative to the position points at constant velocity
con_pmove3	0: Correct	3 axes for relative positional point movement at normal speeds

	-1: Error	
con_pmove4	0: Correct -1: Error	4 axes for relative positional point movement at normal speeds

3.2.1.2 Functional Description

The MPC08D motion controller provides 4 constant speed point motion functions. Point motion means that each axis moves at its own set speed, acceleration, and travel until it reaches a set position or a stop command is called to stop the axis. The travel parameters in the point motion functions are programmed in pulses.

(1) *con_pmove*, *con_pmove2*, *con_pmove3*, *con_pmove4* Explanation

It is used to start one axis, two axes, three axes or four axes, and each axis independently makes point movement with normal speed, and the position parameters in the instruction all refer to relative displacement. Before starting the motion, call "*set_maxspeed*" to set the maximum speed needed for the control axis (which determines the pulse output multiplication rate and resolution of the motion controller) and then call "*set_conspeed*" to set the motion speed. Note: Multiple axes may not arrive at the same time when starting.

3.2.1.3 routine

```
void main( )
{
    auto_set();           // Detecting the controller
    init_board();         // initialize the controller
    set_maxspeed(1,2000); //set the max speed of 1
                           axis to 2000
    set_conspeed(1,2000); //Set 1-axis constant speed to 2000.
    con_pmove(1,5000);    //start 1-axis motion at normal speed, stroke
                           5000
}
```

3.2.2 Trapezoidal Variable Speed Motion Mode

3.2.2.1 Instruction List

There are seven trapezoidal variable speed point motion speed settings and motion functions, as shown in Table 3-3.

Table 3-3 Trapezoidal Velocity Mode Point Motion Functions

function (math.)	return value	clarification
set_s_curve	0: Correct -1: Error	Setting the Control Axis Rapid Motion Mode
set_maxspeed	0: Correct -1: Error	Setting the maximum speed of an axis

set_profile	0: Correct -1: Error	Setting low speed, high speed and acceleration for fast motion
fast_pmove	0: Correct -1: Error	1 axis for fast relative positional point movement
fast_pmove2	0: Correct -1: Error	2 axes for fast relative positional point movement
fast_pmove3	0: Correct -1: Error	3 axes for fast relative positional point movement
fast_pmove4	0: Correct -1: Error	4 axes for fast relative positional point movement

3.2.2.2 Functional Description

The MPC08D motion controller provides seven ladder mode point motion handler functions. Point motion refers to the movement of each axis according to its own set speed, acceleration, and travel until it reaches a set position or a stop command is called to stop the axis. The travel parameters in the point motion functions are programmed in pulses.

(1) *set_s_curve* Description

The motion controller provides two fast motion modes for the point motion axes: trapezoidal curve and S-curve. The trapezoidal acceleration/deceleration mode can be set to asymmetric trapezoidal acceleration/deceleration, i.e., the acceleration and deceleration can be set to different values. Use the "*set_s_curve*" function to set which speed mode is used when an axis is running fast. The system default is trapezoidal speed mode.

(2) *fast_pmove, fast_pmove2, fast_pmove3, fast_pmove4* Description

It is used to start one, two, three or four axes, each of which moves independently and in a rapid manner in a point position. The position parameters in the commands refer to the relative displacements. The trapezoidal speed is set by the command "*set_profile*". Before starting the motion, call "*set_maxspeed*" to set the maximum speed required for the

controlled axes (determining the pulse output multiplication rate and resolution of the motion controller) Multiple axes do not necessarily arrive at the same time when starting.

3.2.2.3 routine

```
void main( )
{
    auto_set( );           // Detecting the controller
    init_board( );         //initialize
    controller set_s_curve( 1,0); //set ladder
    speed mode
    set_maxspeed(1,5000); //Set the maximum speed of 1-axis to 2000
```

```

set_profile(1,100,5000,3000);//set 1-axis initial speed, high speed and
acceleration

fast_pmove (1,5000);//start 1-axis trapezoidal fast movement,
movement distance 5000

}

```

3.2.3 S-curve variable speed motion mode

3.2.3.1 Instruction List

The S-curve variable speed independent motion settings and motion functions are shown in Table 3-4. Table 3-4 S-curve variable speed independent motion function

function (math.)	return value	clarification
set_s_curve	0: Correct -1: Error	Setting the Control Axis Rapid Motion Mode
set_s_section	0: Correct -1: Error	Setting the speed parameters related to the S-shaped elevation speed
set_profile	0: Correct -1: Error	Setting low speed, high speed and acceleration for fast motion
fast_pmove	0: Correct -1: Error	1 axis for rapid point movement

3.2.3.2 Functional Description

The MPC08D motion controller provides one S-type fast point motion function. Point motion means that each axis moves at its own set speed, acceleration, and travel until it reaches a set position or a stop command is called to stop the axis.

(1) *set_s_section* Description

"set_profile" sets the initial speed, high speed and maximum acceleration of the S-section. "set_s_section" is called

after "set_profile" to set the speed change amount of S-section, and the set change amount of S-up and S-down speed can not be more than 1/2 of the high speed set before.

3.2.3.3 routine

```
void main( )
{
    auto_set( );           // Detecting the controller
    init_board( );         //initialize the
    controllerset_s_curve( 1,1); //set S-
    shaped speed mode
    set_profile(1,100,5000,1000); //set 1-axis initial speed, high speed and
    maximum acceleration
}
```



```

set_s_section(1,800,800); //Set the S-section of the elevation speed,
the change cannot be greater than high
                            1/2 of the speed.

fast_pmove (1,5000); //start 1-axis S-shape fast motion with a stroke
of 5000
}

```

3.3 (math.) continuous kinematic function

3.3.1 Instruction List

The continuous motion parameter settings and motion functions are shown in Table 3-5.

Table 3-5 Continuous Motion Functions

function (math.)	return value	clarification
set_maxspeed	0: Correct -1: Error	Setting the maximum speed of axis movement
set_profile	0: Correct -1: Error	Setting parameters for rapid axis movement
set_conspped	0: Correct -1: Error	Setting the parameters of the axis constant velocity motion
con_vmove	0: Correct -1: Error	1 axis in continuous motion at constant speed
con_vmove2	0: Correct -1: Error	2 axes in continuous motion at constant speed
con_vmove3	0: Correct -1: Error	3 axes in continuous motion at constant speed
con_vmove4	0: Correct -1: Error	4 axes in continuous motion at constant speed
fast_vmove	0: Correct -1: Error	1 axis for fast continuous motion

MPC08D Motion Controller		
fast_vmove2	0: Correct -1: Error	2 axes for fast continuous motion
fast_vmove3	0: Correct -1: Error	3 axes for fast continuous motion
fast_vmove4	0: Correct -1: Error	4 axes for fast continuous motion

3.3.2 Functional Description

The MPC08D motion controller provides 8 continuous motion functions. Continuous motion means that the axes are moving according to the

Each set speed, acceleration movement, until the corresponding direction of the limit, alarm signal, or call to stop the command to stop the movement.

(1) *con_vmove*, *con_vmove2*, *con_vmove3*, *con_vmove4* Explanation

It is used to start one axis, two axes, three axes or four axes at the same time, and each axis independently performs continuous motion with constant speed, which is set by the instruction "set_conspeed".

(2) *fast_vmove*, *fast_vmove2*, *fast_vmove3*, *fast_vmove4* Description

The trapezoidal speed is set by the command "set_profile". It is used to start one, two, three or four axes at the same time, and each axis independently makes continuous movement in a fast way.

3.3.3 routine

```
void main( )
{
    auto_set( );           // Detecting the controller
    init_board( );        // initialize controller
    set_maxspeed(1,10000); // set max speed
    set_profile(1,100,10000,8000); // set initial speed, high
    speed, acceleration fast_vmove(1,1); // start 1-axis
    forward continuous motion
    .....
}
```

3.4 Return-to-origin kinematics

3.4.1 Instruction List

The return-to-home motion parameter settings and motion functions are shown in Table 3-6.

Table 3-6 Return-to-origin motion function

function (math.)	return value	clarification
set_maxspeed	0: Correct -1: Error	Setting the maximum speed of axis movement

MPC08D Motion Controller

set_profile	0: Correct -1: Error	Setting parameters for rapid axis movement
set_conspped	0: Correct -1: Error	Setting the parameters of the axis constant velocity motion
set_home_mode	0: Correct -1: Error	Setting the mode of axis return to zero motion

con_hmove	0: Correct -1: Error	1 axis moving back to zero at constant speed
con_hmove2	0: Correct -1: Error	2 axes moving back to zero at constant velocity
con_hmove3	0: Correct -1: Error	3 axes return to zero motion at constant velocity
con_hmove4	0: Correct -1: Error	4 axes moving back to zero at constant velocity
fast_hmove	0: Correct -1: Error	1 axis for fast return-to-zero motion
fast_hmove2	0: Correct -1: Error	2 axes for fast return-to-zero motion
fast_hmove3	0: Correct -1: Error	3 axes for fast return-to-zero motion
fast_hmove4	0: Correct -1: Error	4 axes for fast return-to-zero motion

3.4.2 Functional Description

The MPC08D Motion Controller provides 8 home return motion functions. Return-to-home motion means that each axis moves at its own set speed and acceleration, and does not stop moving until there is an external home signal, limit signal, or alarm signal, or a stop command is called. If a valid limit signal is detected first during the home return process, the control axis will automatically reverse to find the home position.

The Motion Controller provides 4 home modes, which are set by the interface function "set_home_mode":

Mode 0: Origin Immediate stop mode. The control axis moves back to the home position at the set speed and stops immediately when the home signal is valid.

Mode 1: Stopping immediately when the Z pulse is valid. The control axis moves back to the home position at the set speed and stops immediately when the Z pulse signal is valid.

Mode 2: Slow stop with valid ORG signal. When the home position proximity switch signal is detected as valid in the trapezoidal speed mode, the control axis gradually decelerates and stops at the acceleration set by the rapid motion.

Mode 3: In the trapezoidal speed mode, when the home signal is valid, the control axis decelerates gradually to a low speed at the acceleration set in the rapid motion mode until the Z pulse is valid and stops the motion immediately.

NOTE: Modes 2 and 3 are only valid for fast return-to-zero motions.

(1) *con_hmove, con_hmove2, con_hmove3, con_hmove4* Description

It is used to start one axis, two axes, three axes or four axes at the same time, and each axis moves back to the home position independently with constant speed, which is set by the instruction "set_conspeed".

(2) *fast_hmove, fast_hmove2, fast_hmove3, fast_hmove4* Description

The trapezoidal speed is set by the command "**set_profile**". It is used to start one, two, three or four axes at the same time, and each axis moves back to the home position independently in a fast way.

3.4.3 routine

```
void main( )
{
    auto_set( );           // Detecting the controller
    init_board( );         //initialize controller
    set_home_mode(1,0);    //set 1-axis home mode 0
    set_maxspeed(1,1000);  //set max speed
    set_conspped(1,1000);  //initialize controller.
    //Set the zero return speed
    con_hmove(1,1);        //Start the 1-axis positive
    zero return motion
    .....
}
```

3.5 Linear Interpolation Motion Functions

3.5.1 Instruction List

There are six linear interpolation motion settings and motion functions, as shown in Table 3-7.

Table 3-7 Linear Interpolation Motion Functions

function (math.)	return value	clarification
set_vector_conspped	0: Correct -1: Error	Setting the speed of constant velocity motion
set_vector_profile	0: Correct -1: Error	Setting the speed parameter for rapid motion
con_line2	0: Correct -1: Error	Two axes in constant speed linear interpolating motion

MPC08D Motion Controller

con_line3	0: Correct -1: Error	Three axes in constant speed linear interpolation motion
con_line4	0: Correct -1: Error	Four axes in constant speed linear interpolating motion
fast_line2	0: Correct -1: Error	Rapid linear interpolation of two axes
fast_line3	0: Correct -1: Error	Rapid linear interpolation of three axes

fast_line4	0: Correct -1: Error	Rapid linear interpolation in four axes
------------	-------------------------	---

3.5.2 Functional Description

Interpolating motion refers to the linkage of two axes and multiple axes according to a certain algorithm, where the controlled axes are started at the same time and arrive at the target position at the same time. Interpolated motion runs at vector speeds, which are categorized into constant vector speeds and trapezoidal vector speeds. The travel parameter in the interpolating motion function is programmed in pulses.

The MPC08D motion controller provides six linear interpolated motion functions.

(1) *con_line2, con_line3, con_line4* Description

Two axes, three axes and four axes are activated respectively to make linear linkage with constant vector speed, and the motion speed of each controlled axis is the component speed of constant vector speed on the axis, and each controlled axis is activated at the same time and reaches the target position at the same time. The constant vector speed is set by the command "set_vector_conspeed".

(2) *fast_line2, fast_line3, fast_line4* Description

Two axes, three axes and four axes are activated respectively to make linear linkage with trapezoidal vector speed, and the motion speed and acceleration of each controlled axis is the component of trapezoidal vector speed and vector acceleration on the axis, and each controlled axis is activated at the same time, and reaches the target position at the same time. The trapezoidal vector speed is set by the command "set_vector_profile".

4 braking function

4.1 braking function

The brake correlation function is shown in Table 4-1.

Table 4-1 Braking Functions

function (math.)	return value	clarification
sudden_stop	0: Correct -1: Error	Immediate braking of one axis of motion
sudden_stop2	0: Correct -1: Error	Immediate braking of both axes of motion
sudden_stop3	0: Correct -1: Error	Immediate braking of the three axes of motion
sudden_stop4	0: Correct -1: Error	Immediate braking of all four axes of motion
decel_stop	0: Correct -1: Error	Smooth braking of one axis of motion
decel_stop2	0: Correct -1: Error	Smooth braking of both axes of motion
decel_stop3	0: Correct -1: Error	Smooth braking of the three axes of motion
decel_stop4	0: Correct -1: Error	Smooth braking of the four axes of motion
move_pause	0: Correct -1: Error	Pause an axis of motion
move_resume	0: Correct -1: Error	Restore motion in one axis

4.2 Functional Description

(1) *sudden_stop, sudden_stop2, sudden_stop3, sudden_stop4* Description

Immediate braking functions in immediate motion mode. They cause the controlled axis to immediately abort its motion. Immediately after this function is executed, the controller stops sending pulses to the motor driver to stop the motion.

(2) ***decel_stop, decel_stop2, decel_stop3, decel_stop4*** **Description**

Smooth braking function in immediate motion mode. Only for fast motion commands (trapezoidal speed, S-speed).

The "fast" commands (e.g. `fast_hmove`, `fast_vmove`, `fast_pmove2`, etc.) are valid, i.e. only commands that start with "fast" can perform smooth braking. They can make the speed of the controlled axis decrease from high speed to low speed (set by `set_profile`) and then stop the motion. The smooth braking function effectively prevents overshooting of the system during fast motion.

(3) *move_pause*, *move_resume* Description

These two commands are used to pause the current motion of an axis in Immediate Motion mode and to resume axis motion.

- For fast linear interpolation, fast point motion, fast continuous motion, fast return to original
When the "`move_pause`" function is called, the control axis decelerates at the set trapezoidal speed until it stops; and when the "`move_resume`" function is called, the control axis accelerates to a high speed at the trapezoidal speed.
- Corresponding to the normal speed linear interpolation motion, normal speed point motion, normal speed continuous motion, normal speed return to home position motion, call "`move_pause`" function, the control axis stops the motion immediately; call "`move_resume`", the control axis moves at normal speed immediately. After calling "`move_resume`", the control axis will move at normal speed immediately.

5 Position Setting and Reading Functions

5.1 Position Setting Functions

5.1.1 Instruction List

There are 2 position setting functions as shown in Table 5-1.

Table 5-1 Position Setting Functions

function (math.)	return value	clarification
set_abs_pos	0: Correct -1: Error	Setting the absolute position value of an axis
reset_pos	0: Correct -1: Error	Reset the current position value of an axis to zero

6.1.2 Functional Description

(1) *set_abs_pos* Description

Calling this function changes the current absolute position of the control axis to the set value, but no actual axis movement occurs between the previous position and this position. Calling this function and setting the second parameter to 0 realizes the function "reset_pos". The position parameters in the function are programmed in pulses.

This command will not work when the control axis is in motion.

(2) *reset_pos* Description

Function This function resets the absolute and relative position of the control axis to 0. It is usually called when the origin of the axis is found, and after calling this function, the current position value becomes 0. After this, all absolute position values are relative to this point.

This command will not work when the

control axis is in motion. The function also automatically zeroes the auxiliary encoder count.

5.2 Position Reading Functions

5.2.1 Instruction List

The position reading functions are shown in Table 5-2.

Table 5-2 Position Reading Functions

function (math.)	return value	clarification
<code>get_abs_pos</code>	0: Correct -1: Error	Read the absolute position value of an axis

5.2.2 Functional Description

(1) *get_abs_pos* Description

Call this function to read the current absolute position of the control axis. If the home position movement has been executed (the `"reset_pos"` instruction should be called after the home position is returned) then the absolute position is relative to the home position; if the home position movement has not been executed, then the absolute position is relative to the position at the time of power-on. The position parameter in the function is programmed in pulses.

The absolute position of the control axis acquired by this instruction is determined by the number of pulses output from the controller and does not reflect the actual position value in the event of a lost step or overshoot, for example.

6 state handler

6.1 Motion status query function

6.1.1 Instruction List

There are four motion status query functions, as shown in Table 6-1.

Table 6-1 Motion Status Query Functions

function (math.)	return value	clarification
check_status	Other: Status values -1: Error	Read the status of the specified axis
get_cur_dir	0: Stop state -1: Negative 1: Positive -2: Error	Reads the current direction of motion of the specified axis
check_done	0: Stop state 1: State of Motion -1: Error	Checks whether the motion of the specified axis has been completed
get_rate	Speed of motion	Reads the speed of the current movement

6.1.2 Functional Description

(1) *check_status* Description

Each axis of MPC08D control controller has a 32-bit status register. During the motion process, the user can check the working status of the axis by calling the instruction "*check_status*". The meaning of each bit in the status register is shown in the table below.

Table 6-2 Control Axis Status Register Definitions

data bit	state of affairs	clarification
D30	0: not valid, 1: valid	Home signal state
D29	0: not valid, 1: valid	Positive Limit Signal Status
D28	0: not valid, 1: valid	Negative Limit Signal Status

MPC08D Motion Controller

D26	0: not valid, 1: valid	Alarm signal status
(sth. or sb) else	internal state	
D8	0: not valid, 1: valid	Z Pulse signal status
D0	0: movement, 1: stop	motion state

(2) *check_done* Description

Used to detect the motion status of the specified axis. In the immediate motion mode, this function must be used to determine the motion status of the control axis, and the next motion command can be issued to the axis only after the axis stops moving. If the control axis is still moving, the system will discard the following motion instruction.

(3) *get_cur_dir* Description

Reads the current motion direction of the specified axis.

(4) *get_rate* Description

Reads the current motion speed of the control axis. It is possible that the speed read differs from the speed set by the user. This is mainly due to differences caused by the controller speed resolution. Because the output pulse frequency is controlled by two variables: pulse resolution and multiplier, the product of the two is the actual output pulse frequency. The function *set_maxspeed* sets the maximum output pulse frequency, that is, it modifies the pulse resolution, and the maximum speed can be set in accordance with the actual output speed in order to obtain a better speed accuracy.

6.2 Dedicated Input Detection Functions

6.2.1 Instruction List

There are seven dedicated input detection functions, as shown in Table 6-3.

Table 6-3 Dedicated Input State Detection Functions

function (math.)	return value	clarification
<i>check_limit</i>	0: Invalid 1: Positive limit signal valid -1: Negative limit signal is valid 2: Both positive and negative limit signals are valid -3: Error	Check that the limit signal for the specified axis is valid
<i>check_home</i>	0: Invalid 1: Effective -3: Error	Check whether the home

MPC08D Motion Controller

Programming Manual

		Signal of the specified axis is valid
check_alarm	0: Invalid 1: Effective -3: Error	Check the validity of the alarm signal for the specified axis
check_card_alarm	0: Invalid 1: Effective -3: Error	Check if the alarm signal of the board is valid
check_sfr	Other: IO Status -1: Error	Read all the switching status of the dedicated input port. attitude
check_sfr_bit	0: low level	Read the switching status of a bit of the dedicated input port.

	1: High level -1: Error	attitude
--	----------------------------	----------

6.2.2 Functional Description

(1) *check_limit, check_home, check_alarm, check_card_alarm*

Description

They are used to check the status of limit signal, home signal, alarm signal and board alarm signal of the specified axis. Calling the `"check_status"` function reads out the entire status register, while the above functions obtain the status of some of the signals. Note: Only if these dedicated inputs are enabled, i.e. `"enable_org"`, `"enable_limit"`, `"enable_alm"`, `"enable_org"`, `"enable_limit"`, `"enable_alm"`, `"enable_caed_alm"`, etc., the above functions can return the correct status of the dedicated inputs.

(2) *check_sfr, check_sfr_bit* Description

The motion controller saves the home, limit, alarm and deceleration signals of each control axis in a special register. If you do not use these special input signals, you can use commands such as `"enable_org"`, `"enable_limit"`, `"enable_alm"`, `"enable_caed_alm"` to invalidate the corresponding special signals, `"enable_org"`, `"enable_limit"`, `"enable_alm"`, `"enable_caed_alm"` and other commands can be used to invalidate the corresponding dedicated signals, at this time, these ports can be used as general-purpose inputs.

The `"check_sfr"` instruction reads the status of all dedicated input switches from this dedicated register.

The `"check_sfr_bit"` instruction can be used to read the status of a specific input switching bit from this dedicated register.

For the wiring diagram of the general-purpose input, refer to the "Connection Methods for General-Purpose Inputs and Outputs" in the "MPC08D User.doc".

The Dedicated Input Register bits are defined as shown in Table 6-4.

Table 6-4 Dedicated Input Register Definitions

status bit	input signal	clarification
20	ALM	General-purpose input when ALM is invalid
19	Z4	Cannot be enabled as a general-

MPC08D Motion Controller

		Programming Manual
		purpose input signal
18	ALM4	ALM4 is used as a general-purpose input when invalid.
17	ORG4	ORG4 is used as a general-purpose input port when it is invalid.
16	EL4-	EL4 - Invalid as a general-purpose input port
15	EL4+	Used as a general-purpose input when EL4+ is invalid.
14	Z3	Cannot be enabled as a general-purpose input signal
13	ALM3	General-purpose input when ALM3 is invalid
12	ORG3	ORG3 is used as a general-purpose input port when it is invalid.

11	EL3-	EL3 - Invalid as a general-purpose input port
10	EL3+	Used as a general-purpose input when EL3+ is invalid.
9	Z2	Cannot be enabled as a general-purpose input signal
8	ALM2	When ALM2 is invalid, it is used as a general-purpose input port.
7	ORG2	When ORG2 is invalid, it is used as a general-purpose input port.
6	EL2-	EL2 - Invalid as general-purpose input port
5	EL2+	Used as a general-purpose input when EL2+ is invalid.
4	Z1	Cannot be enabled as a general-purpose input signal
3	ALM1	When ALM1 is invalid, it is used as a general-purpose input port.
2	ORG1	When ORG1 is invalid, it is used as a general-purpose input port.
1	EL1-	EL1 - Invalid as general-purpose input port
0	EL1+	Used as a general-purpose input when EL1+ is invalid.

7 Generic IO Manipulation Functions

7.1 Digital IO Port Manipulation Functions

7.1.1 Instruction List

The Motion Controller MPC08D together with the IO expansion board EA1616 can provide 24 general-purpose inputs and 26 general-purpose outputs with opto-isolation. There are four general-purpose digital IO operation functions, as shown in Table 7-1.

Table 7-1 Digital IO Operation Functions

function (math.)	return value	clarification
checkin_byte	Other: IO Status -1: Error	Read all the switching status of the expansion inputs
checkin_bit	0: low level 1: High level -1: Error	Read the switching status of a bit of the extended input port
outport_byte	0: Correct -1: Error	Setting the switching status of each of the general-purpose outputs
outport_bit	0: Correct -1: Error	Setting the switching status of a bit on a general-purpose output port

7.1.2 Functional Description

(1) *checkin_byte*, *checkin_bit* Explanation

Users use this command to read the status of 24 general-purpose inputs of the motion controller, of which 0-8 are located in the motion controller MPC08D, and 9-24 are realized through the

"**checkin_byte**" Reads the status of all input switches from the 24-bit general-purpose inputs of the motion controller.

"**checkin_bit**" Reads the status of an input switch from the 24-bit general-purpose input of the motion controller.

For the wiring diagram of the general-purpose input port, please refer to the "Connection Method of General-Purpose Inputs and Outputs" section in the "MPC08D User.doc".

Table 7-2 Medium and High 16-bit (9-24) Universal Inputs Defined in the P37-05 Adapter Board

P37-05 Turn Board Pinout	37-core cable pinout	name (of a thing)	Explanation
P19	19	IN9	Universal Input 9
P37	37	IN10	Universal Input 10
P18	18	IN11	General purpose input 11
P36	36	IN12	Universal Input 12
P17	17	IN13	General purpose inputs 13
P35	35	IN14	General purpose inputs 14
P16	16	IN15	Universal Input 15
P34	34	IN16	Universal Input 16
P15	15	IN17	General Purpose Inputs 17
P33	33	IN18	Universal Input 18
P14	14	IN19	Universal Input 19
P32	32	IN20	Universal input 20
P13	13	IN21	Universal Input 21
P31	31	IN22	Universal Input 22
P12	12	IN23	Universal input 23
P30	30	IN24	Universal Input 24

The low 8-bit (1-8-way) general-purpose inputs are defined in the P62-05 adapter board as follows. Table 7-3 Low 8-bit (1-8-way) General Purpose Inputs Defined on the P62-05 Adapter Board

MPC08D Motion Controller

P62-05 Turn Board Pinout	name (of a thing)	Explanation
IN1	IN1	Universal Input 1
IN2	IN2	Universal Input 2
IN3	IN3	Universal Input 3
IN4	IN4	Universal Input 4
IN5	IN5	Universal Input 5
IN6	IN6	Universal Input 6
IN7	IN7	Universal Input 7
IN8	IN8	Universal Input 8

(3) *outport_byte, outport_bit* Description

This instruction is used to set the status of the 26-bit general-purpose outputs of the Motion Controller. The low 10 bits of the 26 bits are routed through the DB62 of the MPC08D main board at the P62-05 adapter board end, and the rest of the high 16 bits are routed through the DB37 at the P37-05 adapter board end.

"**outport_byte**" Sets the switching status of 26 bits of the general-purpose output port at the same time.

"**outport_bit**" Sets the switching status of a bit of the general-purpose output port.

For the wiring diagram of the 26-bit general-purpose output port, please refer to the "Connection Methods of General-Purpose Inputs and Outputs" section in the "MPC08D User.doc".

Table 7-4 Low 10-Bit General Purpose Output Port Definitions in the P62-05 Adapter Board

P62-05 pinout	name (of a thing)	clarification
OUT1	OUT1	Universal Output 1
OUT2	OUT2	Universal Output 2
OUT3	OUT3	Universal Output 3
OUT4	OUT4	Universal Output 4
OUT5	OUT5	Universal output 5
OUT6	OUT6	Universal output 6
OUT7	OUT7	Universal Output 7
OUT8	OUT8	Universal Output 8
OUT9	OUT9	Universal output 9
OUT10	OUT10	Universal Output 10

Table 7-5 High 16-Bit General Purpose Output Port Definitions in the P37-05 Adapter Board

P37-05 pinout	37-core cable pinout	name (of a thing)	clarification
P11	11	OUT11	Universal

MPC08D Motion Controller

		Programming Manual	output 11
P29	29	OUT12	Universal Output 12
P10	10	OUT13	General purpose outputs 13
P28	28	OUT14	General- purpose outputs 14
P9	9	OUT15	Universal Output 15
P27	27	OUT16	Universal Output 16
P8	8	OUT17	Universal Output 17
P26	26	OUT18	Universal Output 18
P6	6	OUT19	Universal Output 19
P24	24	OUT20	Universal output 20
P5	5	OUT21	Universal Output 21
P23	23	OUT22	Universal Output 22
P4	4	OUT23	General purpose output 23
P22	22	OUT24	Universal Output 24

P3	3	OUT25	Universal Output 25
P21	21	OUT26	General- purpose outputs 26

8 Other Functions

8.1 Reverse Gap Processing

8.1.1 Instruction List

There are three reverse gap processing functions, as shown in Table 8-1.

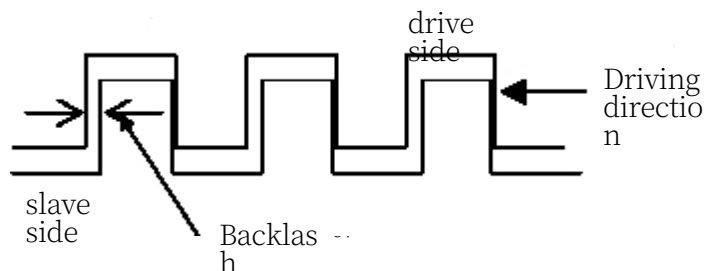
Table 8-1 Speed Setting Functions

function (math.)	return value	clarification
set_backlash	0: Correct -1: Error	Setting the compensation value for the gap created by the mechanism commutation
start_backlash	0: Correct -1: Error	Begins to compensate for position due to mechanism commutation clearance inaccuracies
end_backlash	0: Correct -1: Error	Stop compensating for position due to mechanism commutation gap inaccuracies

8.1.2 Functional Description

(1) *set_backlash*, *start_backlash*, *end_backlash* Description

As a drive, if a gear mechanism or other transmission device is used, there is more or less a backlash between two teeth, as shown in the figure below. When the driving direction of the drive side is changed (right to left or left to right as shown below) the effect of mechanical backlash is reduced by increasing the number of motion pulses on the drive side through the setting of the backlash compensation value.



When backlash exists in the control system, the three functions provided by **MPC08D** can effectively eliminate the backlash and ensure the system position accuracy. The steps are as follows:

- The function "**set_backlash**" is called to set the compensation of an axis according to the amount of backlash.

Value.

- After setting the backlash compensation value, call the function `"start_backlash"` to start the backlash compensation. The motion controller will automatically increase the backlash compensation when the control axis is reversed in the future.
- When it is necessary to end the backlash compensation, the function `"end_backlash"` is called to cancel the backlash compensation. In the future, when the control axis is reversed, the motion controller will no longer perform backlash compensation.

8.1.3 routine

```
void main( )
{
    auto_set();           // Detecting the controller
    init_board();         //initialize
    controller set_maxspeed(1,1000); //set
    max speed set_conspeed(1,1000);
    //set constant speed
    set_backlash(1,100); //set backlash for axis 1 to 100 (pulse)
    start_backlash(1);    //start backlash compensation
    con_pmove(1,10000); //start axis 1 forward movement
    10000
    .....// Wait for axis 1 to stop
    con_pmove(1,-10000); //start 1-axis reverse move 10000, after that
    the system automatically
                        100 more travel to compensate for backlash
    .....
    end_backlash(1);     //end backlash compensation
    .....
}
```


8.2 Dynamic change of target position

8.2.1 Instruction List

The motion controller provides the function of dynamically changing the target position during the motion, and there is one operation function, as shown in Table

8-2 shown.

Table 8-2 Dynamically Changing Target Position Processing Functions

function (math.)	return value	clarification
<code>change_pos</code>	0: Correct -1: Error	Realization of the function of dynamically changing the target position in motion

8.2.2 Functional Description

(1) *change_pos* Description

In relative position mode, use the function `change_pos` to change the target position dynamically. During single-axis point movement, in normal speed mode or trapezoidal speed mode, if the user finds that the end position of the motion command needs to be changed, the user can call "`change_pos`" to dynamically change the end position before or after the end of the point movement. The system will automatically move according to the new end position. The end position is the starting point of the last user issued motion instruction (`fast_pmove`, `con_pmove`). This function can be called multiple times to change the target position.

8.2.3 routine

```
void main( )
{
    auto_set( );           // Detecting the controller
    init_board( );         // initialize controller
    set_maxspeed(1,1000); // set max speed
    set_conspped(1,1000).  // set constant speed
    con_pmove(1,-1000); // initialize 1-axis negative
                          move 1000
    .....                // 1 During negative axial motion
    Change_pos(1,10000); // No need to wait for 1-axis motion to stop to
    con_pmove the
```

The start point is the starting point and moves in the reverse direction to the

```
.....  
}
```

8.3 Power-down protection for data area read/write function

8.3.1 Instruction List

The motion controller provides a power-down protected data area read/write function, which allows the user to write data information to the specified data storage area to realize functions such as software encryption, device parameter storage, and so on. Specific operation

The function is shown in Table 8-3.

Table 8-3 Power-Down Protectable Data Area Read/Write Functions

function (math.)	return value	clarification
write_password_flash	0: Write Success; -1: Write failure due to parameter error; -2: Write failure due to incorrect checksum password.	Data area write function with password protection
read_password_flash	0: Read data successfully; -1: Read failure due to parameter error; -2: Read failure due to incorrect checksum password.	Data area read function with password protection
clear_password_flash	0: Erase data successfully; -1: Erase failure due to parameter error; -2: Erasure loss due to incorrect check digits Defeat.	Erase data area with password protection
write_flash	0: Write Success; -1: Write failure due to parameter error;	Write function for the general data area
read_flash	0: Read data successfully; -1: Read failure due to parameter error;	General Data Area Read Functions
clear_flash	0: Erase data successfully; -1: Erase failure due to parameter error;	Erase general data area data

8.3.2 Functional Description

Motion controller MPC08D provides 1M storage space for power-down protected data area, which is divided into 16 pieces of area, each piece of area is 64K bytes in size. 16 pieces of area corresponding to the address allocation is shown in Table 8-4 below. Area 0 is password protected, i.e., after the user sets the read/write password for this area, the user needs to verify the password to erase, read or write the address of this area in the future, and areas 1-15 are general data areas, which do not need to verify the password to operate in this type of area.

The data type written in the power-down protected data area is

long type, and the number of data that can be written in each area is $16384 = (64 \times 1024 / 4)$ and the address number is 0--16383. The data written to address 16383 in area 0 is the read/write password, and the value of the read/write password for the first time is 0xffffffff.

Table 8-4 Relationship between Area Number and First Address

Area code	first address
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*

Description

Used to perform write, read and erase operations on data areas with password protection. Before writing to the data area, the area must be erased, otherwise the data to be written cannot be written. **Erase operation is valid for the whole chip, you can erase the area of the specified chip, so that all the bits in the 64K byte space of the area are "1".** Read and write operations are valid for 4-byte contiguous addresses, so please be careful when performing erase operations!

② *write_flash, read_flash, clear_flash* Description

Used to perform write, read and erase operations on general data areas. Erase operation must be performed on the area of the slice before writing operation to the data area, otherwise the data to be written cannot be written. **Erase operation is valid for the whole slice, and you can erase the area of the specified slice, making all bits of the 64K byte space of the area "1".** Read and write operations are valid for 4-byte contiguous addresses, so please be careful when performing erase operations!

8.1.3 routine

```
void main( )
```

{

Long data.

auto_set(); // Detecting the controller

init_board(); //Initialize the controller

```

clear_password_flash(1, 0xffffffff); //Erase area 0
write_password_flash(1,16383,0xccccaaaa,0xffffffff); // Repair the
read/write passwords to the following

                                Change to
0xccccaaaa write_password_flash(1,1, 0x01, 0xccccaaaa);//address
number 1 to area 0

                                Write data 0x01
read_password_flash(1,1,&data, 0xccccaaaa); //read area 0, ground
1

                                The value stored at the address
}

```

8.4 Board number and version reading

8.4.1 Instruction List

The motion controller provides **one** board number reading function and **three** version reading functions, as shown in Table 8-4.

Table 8-4 Board Number and Version Reading Functions

function (math.)	return value	clarification
check_IC	Others: Card Number -1: Error	Query the local ID number of the control card set by the user
get_lib_ver	0	Query the version of the library
get_sys_ver	0: Correct -1: Error	Query the driver version
get_card_ver	0: Correct -1: Error	Query Motion Controller Firmware Version

8.4.2 Functional Description

(1) *check_IC* Description

The Motion Controller is designed with a rotary switch to set the local ID number of each board when it is shared by multiple boards. The maximum setting value of the rotary switch is 0xFH, but at present, it

can only be set to 0x0F0x7F, and it can only support 6 boards sharing. Use "check_IC" function to read out the local ID number of the board. When "check_IC" is used, only one control card is installed in the computer.

When the Motion Controller is shipped from the factory, the initialized local ID numbers are all 0. If multiple cards are to be shared, the user needs to change this setting. For example, if 6 cards are shared, you need to set each card to 0, 1, 2 in turn,

If the local ID number is duplicated or has other values, the control system will indicate initialization failure after calling the board initialization functions "auto_set" and "init_board". Call "get_err", "get_last_err" and other functions to read the error information.

(2) *get_lib_ver, get_sys_ver, get_card_ver* Description

Used to read the function library version number, driver version number, and board version number that came with the motion controller, respectively.

After the function returns, the version number is stored in the function's argument pointer variable.

8.5 Software Limit Handling

8.5.1 Instruction List

The motion controller provides software limit function, when the absolute position of the control axis reaches the software limit point, the controller automatically stops the motion sharply or slowly according to the soft limit requirement.

Table 8-5 Soft Limit Operation Functions

function (math.)	return value	clarification
enable_softlimit	0: Correct -1: Error	Setting the soft limit function of the enabled axis
set_softlimit	0: Correct -1: Error	Setting the stopping method of the soft limit
set_softlimit_data	0: Correct -1: Error	Setting the positive and negative limit values for soft limits
check_softlimit	0: Correct -1: Error	Detection of soft limit

8.5.2 Functional Description

(1) *enable_softlimit* Description

The motion controller provides software limit function, use

"enable_softlimit" function to enable or disable the software limit function. When the absolute position of the control axis reaches the software limit point, the controller automatically stops the motion sharply or slowly according to the soft limit requirement.

(2) ***set_softlimit* Description**

Function "set_softlimit", set the comparison source of soft limit, MPC08D is fixed as internal pulse; set the way to stop the control axis after triggering the software limit: smooth braking, stop immediately.

(3) ***set_softlimit_data* Description**

The function "set_softlimit_data" sets the maximum coordinate limit of the software limit of the control axis in the forward, reverse and negative directions, and the value is the absolute coordinate. When the absolute position of the control axis reaches the coordinate limit of the software limit, the controller automatically stops the movement sharply or slowly according to the requirement of the soft limit.

(4) *check_softlimit* Description

The function "check_softlimit" queries whether a soft limit has occurred in the control axis.

8.5.3 routine

```
void main( )  
  
{  
  
int nflag;  
  
long nenc.  
  
.....  
  
enable_softlimit (1, 1); //enable axis 1 software limit function  
  
set_softlimit (1,0,1); //After a software limit occurs, the control axis  
is smoothly braked.  
  
set_softlimit_data (1,0,50000); //set the software negative limit  
point to 0, positive limit to 50000  
  
  
  
fast_pmove(1,10000); //send a motion command when the  
absolute coordinate of the control axis exceeds that of the  
coverage will  
  
//Stop motion, call check_softlimit to check
```

//Brake

.....
}

9 Error Codes and Handling Functions

9.1 error code handler

9.1.1 Instruction List

The Motion Controller can return the last 10 error states so that the user is aware of the recent status of the system.

There are three error code manipulation functions, as shown in Table 9-1.

Table 9-1 Error Code Operation Functions

function (math.)	return value	clarification
get_err	0: Correct -1: Error	Get the last 10 error codes
get_last_err	0: Correct -1: Error	Get the last error code
reset_err	0	Reset and clear all (10) error codes.

9.1.2 Functional Description

(1) *get_err* Description

Reads the error message from the last 10. The motion controller provides the following table of error messages.

Table 9-2 Meaning of Error Codes

error code (hexadecimal)	hidden meaning
0x00000000	The controller is working properly
0x00000004	Auto_set is not called to set the controller
0x00000005	Invalid device handle
0x00000006	Communication failure with the driver
0x00000007	Motion control card not detected

MPC08D Motion Controller

Programming Manual

0x00000008	Axis not detected on motion control card
0x00000009	The motion control card is not initialized or the initialization is not successful
0x0000000a	If too many motion control cards are installed in the computer, the MPC08D allows a maximum of four (4) MPC08Ds. card sharing
0x0000000b	Number of axes detected The number of axes in the card design does not match the number of axes in the MPC08D per card design. Number of axes 4

0x0000000c	Card local ID number is less than 0
0x0000000d	Card local ID number greater than 4
0x0000000e	Using a control card whose local ID number is not set to 0
0x0000000f	When sharing multiple cards, the local ID number of the first card is not set to 0.
0x00000010	When multiple cards are shared, the local ID number of each card is not incremented from 0.
0x00000012	Mismatch between driver version and library version
0x00000013	Mismatch between board firmware version and library version
0x00000014	Board initialization error
0x02000001	Wrong setting of the axis number parameter in the instruction
0x02000002	Wrong setting of speed parameter in the instruction
0x02000005	Incorrect setting of the card number parameter in the command
0x02000007	The two axes of circular motion are not on the same card
0x0200000a	Read/write control bit parameter exceeds maximum allowable range in digital IO operation
0x0200000d	The set watchdog timer value is not within the range of 1 to 60,000 milliseconds.
0x02000016	Set ellipse scale less than or equal to 0
0x02000017	Setting the axis number to the same value in multi-axis motion commands
0x02000018	Backlash less than 0
0x02000019	Parameter setting error
0x0200001b	Conflicts between functions when multiple functions are activated on the control axis
0x0200001c	Conflicts between functions when the control card activates multiple functions
0x0200001e	Failure of internal status detection before initiating a motion command
0x02000020	The batch setup commands <code>open_list</code> , <code>close_list</code> , <code>add_list</code> , etc. are not paired. expense or outlay
0x02000021	lookahead <code>start_lookahead</code> , <code>end_lookahead</code> not called in

	pairs Programming Manual
0x02000022	The look-ahead command is called in the batch process
0x02000023	Wrong instruction called in forward-looking processing

(2) ***get_last_err*** Description

Returns the most recent error code.

(3) ***reset_err*** Description

Zeroes the 10 error code variables.

10 function description



The motion controller functions are organized by function, and the instruction prototypes are described in the C language.

If you are learning to use the Motion Controller, or if the application is in the debugging stage, do not connect the mechanical system to avoid damage to the equipment by misuse.

10.1 Controller Initialization Functions

This class of functions is mainly used to initialize the MPC08D card. If you want to use the functions of MPC08D, you must first call the functions `auto_set` and `init_board` in order to complete the initialization of the controller before calling other functions.

Table 10-1 Controller and Axis Setting Functions

function prototype	clarification
<code>int auto_set(void)</code>	Auto-detect and auto-set controllers
<code>int init_board(void)</code>	Initialization of controller hardware and software

Function name: `auto_set`

Purpose Purpose: To automatically detect the number of motion controllers, the number of axes on each card, and to automatically set each motion controller using the `auto_set` function.

Syntax: `int auto_set (void)`

Description Description : you can call `auto_set` to complete the auto-detection of the number of motion controllers, the number of axes, and set these parameters automatically.

Return Value: If the call is successful, the `auto_set` function returns the total number of axes; if the card is not detected, the return value is

0; call failure returns -1.

System: WINDOWS 2000, WINDOWS XP
WINDOWS XP

System: **WINDOWS 2000,**

Note : Each program must first call **auto_set** to complete the auto-detection and auto-setting of the card, otherwise the motion controller will not work. This function can only be called once in the program.

cf. See:
invocation
example:

Function name: init_board

Purpose: Initialize the motion controller with the `init_board` function.

Language Method: `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（2000pps），normal speed（2000pps），trapezoidal speed（2000pps initial speed, 8000pps high speed, 80000ppss acceleration and deceleration）vector normal speed（2000pps）vector trapezoidal speed（2000pps initial speed, 8000pps high speed, 80000ppss acceleration and deceleration），etc. The function can only be called in the program. This function can only be called once in the program. **init_board function must be called after auto_set. If the init_board function is not called to initialize, the controller will not work properly. If you need to change the pulse output mode, speed and other initialization data, you can call other functions to modify.**

Return Value: If the call is successful, the `init_board` function returns the number of boards inserted; if the card is not detected, it returns 0; -1 indicates an error.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000, WINDOWS XP

See See:

`auto_set` call

example:

10.2 Property Setting Functions

This class function is mainly used to set the attributes of the MPC08D card's control axes to be used and the related attributes of

the board.

Table 10-2 Control Axis Attribute Setting Functions

function prototype	clarification
<code>int set_outmode(int ch, int mode. int outlogic)</code>	Setting the pulse output mode for each axis
<code>int set_home_mode(int ch, int home_mode)</code>	Setting Home Mode
<code>int set_dir(int ch, int dir)</code>	Setting the direction of motion of an axis
<code>int enable_alm(int ch, int flag)</code>	Setting the validity of the external alarm signal for an axis

<code>int enable_el(int ch, int flag)</code>	Setting the validity of the external limit signal for an axis
<code>int enable_org(int ch, int flag)</code>	Setting whether the external home signal of an axis is valid or not
<code>int enable_card_alm(int cardno,int flag)</code>	Set whether the alarm signal of the controller is valid or not
<code>int set_alm_logic(int ch,int flag)</code>	Setting an axis alarm signal to be active high also is active low
<code>int set_el_logic(int ch,int flag)</code>	Setting an axis limit signal to be active high also is active low
<code>int set_org_logic(int ch,int flag)</code>	Setting the home signal of an axis to be active high also is active low
<code>int set_card_alm_logic (int ch, int flag)</code>	Sets whether the board's alarm signal is active high or Active Low

Function name: set_outmode

P u r p o s e : To set the pulse output mode for each axis. The initialization of the board is set to pulse/direction mode, and if the driver requires a dual pulse (forward pulse/reverse pulse) control signal interface, then this function should be called after the `init_board` function.

Language Method: `int set_outmode (int ch, int mode, int outlogic)` **ch:** the control axis whose output mode needs to be set.
mode: Pulse output mode setting (1 is pulse/direction mode, 0 is double pulse mode)
outlogic: This parameter is not valid in MPC08D.

Description Description: By default, the `init_board` function sets all axes to pulse/direction mode. If the drive requires a dual pulse (forward and reverse pulse) mode input, then `set_outmode` should be called after the `init_board` function to

reset the required mode. Note: The output mode of the controller should match the input signal mode of the connected drive, otherwise the motor will not work properly.

Return value: if the output mode is set successfully, **set_outmode** returns 0, otherwise it returns

-1.

System: WINDOWS 2000, WINDOWS XP
WINDOWS XP

System: WINDOWS 2000,

See also: **init_board**

Call example: `set_outmode (2, 0, 1)/* Set the pulse output mode of the 2nd axis to double pulse mode. */`

Function name: set_home_mode

Purpose Purpose: Used to set the way to detect the home signal when each axis returns to home motion.

Syntax: `int set_home_mode (int ch, int home_mode)`

`ch`: is the axis set;

`home_mode`: the mode of detecting the home signal when returning to the home position movement
`0`: the axis stops the movement immediately when the home proximity switch signal is detected;

`1`: Stop motion as soon as an encoder Z-phase pulse signal is detected.

`2`: In trapezoidal speed mode, when the home position proximity switch signal is detected to be valid, the control axis gradually decelerates and stops at the acceleration set in the rapid motion mode;

`3`: In the trapezoidal speed mode, when the home signal is valid, the control axis gradually decelerates to a low speed according to the acceleration set in the rapid motion mode until the Z pulse is valid and stops the motion immediately.

Description: When the controlled equipment (e.g. CNC machine tools, etc.) moves back to the home position, the MPC08D motion controller will automatically detect the home signal and stop the motion automatically when it reaches the home position. The home position signal is usually sent by a proximity switch. In some occasions that require high positioning accuracy when returning to the home position, the home signal, in addition to the proximity switch signal, should also detect the Z-phase pulse of the optical encoder on the actuator motor, i.e., only when the proximity switch signal and the Z-phase pulse signal appear at the same time, it indicates that the home position has been reached. The function `set_home_mode` is used to set the way of detecting

the home signal of each axis during the home movement.

Note: Only if the actuator motor is equipped with a photo encoder, the Z-phase pulse signal can be used as the detection signal for the home motion, otherwise the home motion will not be completed correctly.

Return Value: 0 if the setting is successful, otherwise -1.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000,
WINDOWS XP

See:

Call example: set_home_mode (1, 1)

Function name: set_dir

Purpose: To set the direction of motion of the axis.

Syntax: int set_dir (int ch, int dir); ch: the
axis to be set.

dir: indicates the motion direction of the controlled axis, +1 indicates positive direction; -1 indicates negative direction.

Description **Description :** Call this function to set the motion direction of an axis before a new motion instruction is issued. However, the final motion direction is still determined by the motion instruction. This instruction is mainly used in some drives Situations requiring that the direction of motion must precede the pulse.

Return Value: If the function call succeeds, the return value is 0; otherwise, it returns -1. **System:** WINDOWS 2000, WINDOWS XP

See also:

Call example: `set_dir(1, -1);` /* Set the motion direction of the 1st axis to negative direction */

Function name: enable_alm

Purpose **Purpose:** To set whether the external alarm signal of the axis is valid or not.

Syntax: `int enable_alm (int ch, int flag);`

ch: the number of the control axis.

flag: the flag of whether the external alarm signal is valid or not, 1 means enable the external alarm signal;

0 Indicates that the external alarm signal is disabled.

Description : Call this function to set whether the external alarm signal of an axis is valid or not. If the external alarm signal of an axis is set to invalid, the corresponding alarm signal input port (ALM) can be used as a general-purpose input port, and the status of the corresponding port can be read with the function `check_sfr` or `check_sfr_bit`.

Return Value: If the function call succeeds, the return value is 0; otherwise, it returns -1. **System:** WINDOWS 2000, WINDOWS XP

See also: `check_sfr`, `check_sfr_bit`

Call example: `enable_alm (1, 0); /* Disable external alarm signal for 1st axis
*/`

Function name: `enable_el`

Purpose Purpose: To set whether the
external limit signal of the axis is valid or not.

Syntax: `int enable_el (int ch, int flag);`

`ch`: the number of the control axis.

`flag`: flag of whether the external limit signal is valid or not, 1
means enable the external limit signal;

0 Indicates that the external limit signal is disabled.

Example of call: `enable_el (1, 0);` /* Set the external limit signal of the 1st axis to invalid */
Description : Call this function to set whether the external limit signal of an axis is valid or not. If the external limit signal of an axis is set to

If the limit signal is set to invalid, the corresponding limit signal input port (EL+EL-) can be used as a general-purpose input port, and the status of the corresponding port can be read with the function `check_sfr` or `check_sfr_bit`.

Return Value: If the function call succeeds, the return value is 0; otherwise, it returns -1. System: WINDOWS 2000, WINDOWS XP

See also: `check_sfr`, `check_sfr_bit`

Function name: enable_org

Purpose : Purpose: To set whether the external origin signal of the axis is valid or not.

Syntax: `int enable_org (int ch, int flag);`

`ch`: the number of the control axis.

`flag`: the flag of whether the external home signal is valid or not, 1 means enable the external home signal;

0 Indicates that the external home signal is disabled.

Description : Call this function to set whether the external home signal of an axis is valid or not. If the external home signal of an axis is set to invalid, the corresponding home signal input port (ORG) can be used as a general-purpose input port, and the status of the corresponding port can be read with the function `check_sfr` or `check_sfr_bit`.

Return Value: If the function call succeeds, the return value is 0; otherwise, it returns -1. System: WINDOWS 2000, WINDOWS XP

See also: `check_sfr`, `check_sfr_bit`

Call example: `enable_org (1, 0);` /* set the external origin signal of the 1st

axis to invalid */

Function name: enable_card_alm

Purpose Purpose: To set whether the board alarm signal is valid or not.

Syntax: int enable_card_alm (int cardno, int flag)

cardno: the number of the card.

flag: the flag of whether the external alarm signal of the board is valid or not, 1 means enable the external alarm signal; 0 means disable the external alarm signal.

Description **Description :** Call this function to set whether the external alarm signal is valid or not. If the external alarm signal of the board is set to invalid, the corresponding alarm signal input port (ALM) can be used as a general-purpose input port, and the corresponding port status can be read by using the function `check_sfr` or `check_sfr_bit`.

Return Value: If the function call succeeds, the return value is 0; otherwise, it returns -1. **System:** WINDOWS 2000, WINDOWS XP

See also: `check_sfr`, `check_sfr_bit`

Call example: `enable_card_alm (1, 0); /* set card 1 alarm signal to invalid */`

Function name: set_alm_logic

Purpose **Purpose:** To set whether an axis alarm signal is active high or low. **Syntax** : `int set_alm_logic (int ch, int flag);`

ch: the axis to be set;

flag: external alarm signal valid level flag, 1 means external alarm switch triggered at high level; 0 means external alarm switch triggered at low level.

Description : Call this function to set the effective level of the control axis alarm signal trigger for external normally open or normally closed proximity switches. If the external alarm signal of the control axis is set to high level valid, the axis stops moving when the corresponding alarm signal input port is high. The system defaults to high level active during initialization.

Return Value: If the function call succeeds, the return value is 0; otherwise, it returns -1. **System:** WINDOWS 2000, WINDOWS XP

See also:

Call example: `set_alm_logic (1, 0); /* Set the alarm signal of the 1st axis to be active low */`

Function name: `set_el_logic`

Purpose Purpose: To set whether an axis limit signal is active high or active low. **Syntax** : `int set_el_logic (int ch, int flag);`
 `ch`: the number of the control axis.

flag: valid level flag of external limit signal, 1 means the external limit switch triggers the controller at high level; 0 means the external limit switch triggers the controller at low level.

Description : Call this function to set the effective level of the limit signal trigger for the control axis to meet the external normally open or normally closed proximity switch. If the external limit signal of the control axis is set to valid high level, the motion of the axis in a certain direction will be stopped automatically when the limit signal input port of that direction is high. The system defaults to active high during initialization.

Return Value: If the function call succeeds, the return value is 0; otherwise, it returns -1. **System:** WINDOWS 2000, WINDOWS XP

See also:

Call example: `set_el_logic (1, 0); /* set the limit signal of the 1st axis to be active low */`

Function name: set_org_logic

Purpose Purpose: To set whether an axis origin signal is active high or active low. **Syntax** : `int set_org_logic (int ch, int flag);`

ch: the number of the control axis.

flag: external home signal valid level flag, 1 means external home switch high level triggers the controller; 0 means external home switch low level triggers the controller.

Description : Call this function to set the effective level of the external home signal of the control axis for an external normally open or normally closed proximity switch. If the external home signal of the control axis is set to valid high level, the corresponding home signal input port will

indicate that the axis returns to the home position when it is high. The system defaults to high level active during initialization.

Return Value: If the function call succeeds, the return value is 0; otherwise, it returns -1. System: WINDOWS 2000, WINDOWS XP

See also:

Call example: `set_org_logic (1, 0) /* Set the home signal of the 1st axis to active low */`

Function name: set_card_alm_logic

Purpose Purpose: Used to set whether the card alarm signal is active high or active low. Syntax : int
`set_card_alm_logic (int cardno, int flag);`

cardno: the number of the board.

flag: external alarm signal valid level flag, 1 means the external alarm switch triggers the controller at high level; 0 means the external alarm switch triggers the controller at low level.

Description: The MPC08D Motion Controller has a board alarm signal that is valid for all axes on the board. When the alarm signal is valid, all controlled axes stop running, as opposed to an alarm signal for each axis that is valid only for that axis. Call this function to set the effective level of the board's alarm signal to meet the needs of external normally open or normally closed proximity switches. If the external alarm signal is set to high level, all axes will stop automatically when the corresponding alarm signal input port is high. The system defaults to high level active during initialization.

Return Value: If the function call succeeds, the return value is 0; otherwise, it returns -1. **System**: WINDOWS 2000, WINDOWS XP

See:

Call example: `set_card_alm_logic (1, 0) /* set card 1 alarm signal to active low */`

10.3 Motion parameter setting function

This category of functions is mainly used to set the MPC08D card motion speed, acceleration, position and other parameters. Table 10-3 Control Axis Attribute Setting Functions

function prototype	clarification
<code>int set_maxspeed(int ch , double speed)</code>	Setting the maximum speed of the control axis

MPC08D Motion Controller

int set_conspeed(int ch, double conspeed)	Setting the constant speed of each axis
set_profile(int ch , double vl , double vh , double ad, double dc)	Setting the trapezoidal speed for fast motion
int set_vector_conspeed(double con_speed)	Setting the constant vector speed
set_vector_profile(double vec_vl set_vector_profile(double	Setting the trapezoidal vector speed

double vec_vh, double vec_ad, double vec_dc);	
int set_s_curve(int ch, int mode)	Setting the rapid motion mode of an axis
int set_s_section(int ch, double accel_sec, double decel_sec)	Setting the S lift speed range for S-curve motion
int set_abs_pos (int ch, double pos)	Setting the absolute position value of an axis
int reset_pos (int ch)	Current position value reset to zero

Function name: set_maxspeed

Purpose Purpose: To set the maximum speed of the control axis.

Syntax: int set_maxspeed(int ch, double speed)

Method: int set_maxspeed (int ch,
double speed); ch: the set control axis.

speed: set the maximum speed value, the unit is pulse / second (pps)

Description : By default, the initialization of the board sets the maximum speed of all axes to 2MHz, at this time, the speed resolution is poor, **if you want to get a higher speed accuracy, you can set it according to the actual maximum speed.** The maximum pulse frequency can be set to 2,000,000 Hz. If the setting value exceeds the maximum value allowed, the controller will set it at 2MHz. The minimum pulse frequency can be set to 81.91 Hz, if the setting is less than the minimum value allowed, the controller is set at 81.91.

Return Value: if the output mode is set successfully, set_maxspeed returns 0, otherwise -1.

System: WINDOWS 2000, WINDOWS XP

System: WINDOWS 2000,

WINDOWS XP

See also: set_conspeed

Example: set_maxspeed(210000)* Set the maximum speed of the 2nd axis to 10000pps */

Function name: set_conspeed

Purpose: The `set_conspeed` function is used to set the speed of an axis in constant speed motion. Syntax : `int set_conspeed (int ch, double conspeed)`

`ch`: the number of the control axis.

`conspeed`: the set constant speed value in pulses per second (pps)

Description: The function `set_conspeed` sets the speed in the constant speed mode of motion. If this function is called several times, the last set value is valid and remains valid until the next change. The maximum pulse frequency can be set to 2000000 Hz, if the set value is exceeded the maximum pulse frequency can be set to 2000000 Hz.

If the setting value exceeds the maximum value allowed, the controller will be set at 2MHz. The minimum pulse frequency can be set to 0.2 Hz. If the set value is less than the allowed minimum value, the controller will set it at 0.2. The constant speed value is generally set low to avoid causing the control motor (especially open-loop stepper motors) to lose steps or overshoot. If high-speed motion is required, it is best to use the trapezoidal speed method.

Return Value: `set_conspeed` returns 0 if the constant speed value is set successfully, and -1 if there is an error. system: WINDOWS 2000, WINDOWS XP

See also: `set_profile`, `set_vector_conspeed`

Call example: `set_conspeed (2, 400)`

Function name: `set_profile`

Purpose Purpose: Use `set_profile` function to set the parameter values of trapezoidal speed in fast motion (including `fast_hmove`, `fast_vmove`, `fast_pmove`, etc.);

Syntax: `set_profile(int ch , double vl , double vh , double ad, double dc);` ch: number of the control axis.

`vl`: set the value of low speed (starting speed); unit is `pps` (pulse/second) `vh`: set the value of high speed (constant speed section); unit is `pps` (pulse per second); `ad`: set the magnitude of acceleration; unit is `ppss` (pulse/second/second).

`dc`: Sets the deceleration rate in `ppss` (pulses per second/second).

Description: the `set_profile` function sets the low speed (start speed) high speed (target speed) and acceleration/deceleration value of an axis in the fast motion mode (the deceleration value can be not equal to the acceleration value in the trapezoidal speed mode) The default values for these parameters are 2000, 8000, 80000, 80000. The minimum pulse frequency can be set to 10 Hz for the low speed value and the maximum pulse frequency can be set to 2000000 Hz for the high speed value. if the setting value exceeds the maximum value allowed, the controller will set it according to the 2 MHz. The minimum pulse frequency can be set to 10 Hz. If the setting

value is less than the allowed minimum value, the controller will set it at 10. The minimum acceleration value can be set to 20, below which the controller will be set at 20.

Return Value: If the parameter value is set successfully, `set_profile` returns 0, and -1 if there is an error. System:

WINDOWS 2000, WINDOWS XP

See also: `set_conspeed`, `set_vector_conspeed` See: `set_conspeed`, `set_vector_conspeed`, `set_vector_profile` call example:

`set_profile(3,600,6000,10000,10000)`

Function name: `set_vector_conspeed`

Purpose Purpose: Use the `set_vector_conspeed` function to set the vector speed in constant speed mode, which will be used in two-axis or multi-axis linear interpolation motion;

Syntax: int set_vector_conspeed (double vec_conspeed)

Method: int set_vector_conspeed (double
vec_conspeed)vec_conspeed: vector speed at
constant speed interpolation.

DescriptionThe set_vector_conspeed function sets vector speeds for the following 2-axis or multi-axis interpolating motion functions: con_line2, con_line3, con_line4, etc. It cannot set speeds for fast interpolating motions such as fast_lin2, fast_line3, etc. (their speeds depend on the set_vector_profile). It cannot set the speed for fast interpolating motions such as fast_line2, fast_line3, etc. (their speeds depend on the set_vector_profile) The maximum pulse frequency can be set to 2 MHz, and the controller will set it to 2 MHz if the setting value exceeds the maximum value allowed. The minimum pulse frequency can be set to 1 Hz, if the setting value is less than the allowed minimum value, the controller will set it to 1.

Return Value: set_vector_conspeed returns 0 if the parameter value is set successfully, and -1 if the parameter value is set wrongly.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000,
WINDOWS XP

Note Note: The constant vector speed should be set relatively small to avoid losing steps during the motion. For fast interpolation motion, such as fast_line2, fast_line3, etc., set_vector_profile can be used to set the motion speed.

See also: set_vector_profile , set_conspeed,
set_profile call example: set_vector_conspeed(1000)

Function name: set_vector_profile

Objective: To set vector trapezoidal speed parameters with set_vector_profile.

Purpose: To set the vector trapezoidal velocity parameter with set_vector_profile;

Syntax : set_vector_profile(double vec_vl , double vec_vh ,double
vec_ad,double vec_dc); vec_fl:

velocity value of vector low speed; **vec_fh**: velocity value of vector high speed; **vec_ad**: acceleration value of vector high speed; **vec_dc**: vector deceleration value

Description: The function **set_vector_profile** sets the vector trapezoidal velocity for fast interpolating functions such as **fast_line2** and **fast_line3**. This function does not set the motion speed for constant speed interpolation functions such as **con_line2** and **con_line3**. The maximum pulse frequency can be set to 4 MHz. If the setting value exceeds the maximum value allowed, the controller will set it to 4 MHz. The minimum pulse frequency can be set to 10 Hz, if the setting value is less than the allowed minimum value, the controller will set it according to 10, and the minimum acceleration value can be set to 20, and it will be processed according to 20 if it is lower than this value.

Return Value: The **set_vector_profile** function returns 0 if the call was successful, or -1 in the case of an error.

System: WINDOWS 2000, WINDOWS XP **System:** WINDOWS 2000, WINDOWS XP

See also: `set_vector_conspeed`, `fast_line2`, `fast_line3`

Call example: `set_vector_profile (1000, 16000, 10000, 10000)`

Function name: `set_s_curve`

Purpose: The `set_s_curve` function is used to set the fast motion mode of an axis. Syntax : `int set_s_curve(int ch,int mode)`

`ch`: Axis number

`mode`: rapid motion

mode 0 -

trapezoidal

acceleration/decele

ration mode

1-S Acceleration and deceleration modes

Description : Set the mode of the rapid motion of the axis by the `set_s_curve` function, in the default case it is 0, i.e. trapezoidal acceleration/deceleration.

Return Value: If the call is successful, the function returns 0, in case of error, it returns -1. System: WINDOWS 2000, WINDOWS XP

See :

Call example: `set_s_curve(1,1);`

Function name: `set_s_section`

Purpose: The `set_s_section` function is used to set the S-section of an axis with S-shaped elevation and deceleration. Syntax : `int set_s_section(int ch,double accel_sec,double decel_sec)`

`ch`: Axis number

`accel_sec`: S-segment acceleration value for S-shaped

acceleration, cannot be greater than 1/2 of the set high

speed. `decel_sec`: S-segment deceleration value for S-shaped

deceleration, cannot be greater than 1/2 of the set high speed. The `decel_sec`: S-shaped deceleration S-segment deceleration value, cannot be greater than 1/2 of the set high speed.

Description : For fast motion modes, an S-shaped speed profile can be used in order to make the lift speed process smoother. The `set_s_section` function is used to set

the value of the axes acceleration and deceleration. The up-speed is S-shape from initial speed to (initial speed + `accel_sec`), and S-shape from (high speed - `accel_sec`) to high speed; the down-speed is S-shape from high speed to (high speed - `decel_sec`), and S-shape from (initial speed + `decel_sec`) to initial speed. The initial speed and high speed are set by `set_profile`.

Return Value: If the call is successful, the function returns 0, in case of error, it returns -1. **S y s t e m** : WINDOWS 2000, WINDOWS XP

S e e :

Call example: `set_s_section (1,100,200);`

Function name: set_abs_pos

Purpose Purpose: Used to set the absolute position of the start of axis movement. Syntax: int set_abs_pos (int ch, double pos) Method: int set_abs_pos (int ch, double pos)

ch: the axis to be set;

pos: the starting absolute position of the axis to be set;

Description : This function is called to set the current absolute position to a certain value, but no actual movement of the axis will occur between the previous position and that position. Calling this function with the second parameter set to 0 accomplishes the function of the **reset_pos()** function. This function must be called with the assurance that the axis motion has stopped, otherwise it will cause confusion about the absolute position value.

Return Value: If the function call succeeds, the return value is 0; otherwise, it returns -1. System: WINDOWS 2000, WINDOWS XP

See :

Call example: set_abs_pos (1, 1000) /* set the current position of the 1st axis to 1000*/

Function name: reset_pos

Syntax: int reset_pos (int ch)

ch: Shaft number of the reset shaft;

Description: The **reset_pos** function ~~resets the~~ absolute and relative position of the specified axis to 0. It is usually called when the origin of the axis is found, and after this function is called, the current position value becomes 0. After this, all absolute position values are relative to this point. When calling this function, you must make sure that the axis motion has stopped, otherwise it will cause confusion in the absolute position value. Generally, this function should be called after a successful zero return motion.

Return Value: If the call is successful, `reset_pos` returns 0, otherwise it returns -1. System: WINDOWS 2000, WINDOWS XP

Note Explanation:

See also: `get_abs_pos`

Call example: `int reset_pos (1)`

10.4 movement instruction

Classified by the type of motion, there are three main types: point motion, continuous motion and back to the origin motion; according to the mode of motion can be divided into two kinds of independent motion and interpolated motion; according to the speed of motion can be divided into two kinds of normal speed motion and fast motion. For the convenience of description, the following motion instructions are categorized into independent motion and interpolation motion.

section to illustrate.

10.4.1 independent motion function (math.)

The so-called independent motion means that there is no linkage between the motions of the axes, which can be single-axis or to be multiple axes moving simultaneously at their respective speeds. Point motion, continuous motion, and home return motion are all independent motion. The function name format of the independent motion instruction is: **X_YmoveZ**, where:

X: Replaced by **con** and **fast**, with **con** denoting normal motion and **fast** denoting fast motion;

Y: is replaced by **p**, **v** and **h**. **p** denotes pointwise motion, **v** denotes continuous motion and **h** denotes back-to-home motion;

move: is the body of the instruction, indicating that the instruction is a movement instruction;

Z: no for single-axis motion, 2 for two-axis independent motion, 3 for three-axis independent motion.

vertical motion, and a **value** of 4 indicates four-axis independent motion.

For example, **con_vmove** is a single-axis constant velocity continuous motion function; **con_pmove2** is a two-axis

Normal speed point motion function; **fast_hmove3** is a fast home motion instruction for three axes.

For normal speed motion instruction, the motion speed is set by **set_conspeed**; for fast motion instruction, the motion speed is set by **set_profile**.

(1) Point Motion Functions

Point motion means that the controlled axes move separately at their respective speeds for a specified distance and stop automatically when they reach the target position. Note: In a 2-axis or 3-axis point motion function, the axes start moving at the same time, but do not necessarily reach the target position at the same time. The MPC08D library provides a total of 8 point motion functions:

Table 10-4 Point Motion Functions

MPC08D Motion Controller

function prototype	clarification
int con_pmove(int ch ,double step)	One axis does pointwise motion at constant speed
int fast_pmove(int ch , double step)	One axis for rapid point movement
int con_pmove2(int ch1, double step1, int ch2, double step2)	Two axes in pointwise motion at constant speed
int fast_pmove2(int ch1, double step1, int ch2, double step2)	Two axes for rapid point movement
int con_pmove3(int ch1, double step1, int ch2, double step2, int ch3,	Three axes in pointwise motion at constant speed

double step3)	
fast_pmove3(int fast_pmove3(int ch1, double step1,int ch2, double step2,int ch3, double step3)	Three axes for rapid point movement
int con_pmove4(int ch1, double,step1,int ch2,double step2,int ch3, double step3,int ch4,double step4)	Four axes do pointwise motion at constant speed
fast_pmove4(int fast_pmove4(int ch1, double,step1,int ch2,double step2,int ch3, double step3,int ch4,double step4)	Four axes for rapid point movement

Among

in: ch, ch1, ch2, ch3, ch4: axis number of the controlled axis;
STEP, STEP1, STEP2, STEP3, STEP4: It indicates the distance that the controlled axis moves from the current position, the positive number indicates the positive direction, the negative number indicates the negative direction, and its unit is the number of pulses.

Calling example:

```
con_pmove (1, -2000) /* the first axis is moved 2000 pulses in
the negative direction at its constant speed */
fast_pmove2 (2, 5000, 3, -1000) /* The second axis moves a
distance of 5000 pulses in a fast positive direction; the third
axis moves a distance of 1000 pulses in a fast negative
direction. */
```

Return Value: These functions return 0 if the call succeeds, or -1 in the case of an error.

(2) (math.) continuous kinematic function

Continuous motion means that the controlled axes move in a given direction at their respective speeds until they hit a limit switch or a braking function is called to stop them. A total of eight continuous motion instruction functions are provided in the MPC08D library:

Table 10-5 Continuous Motion Functions

MPC08D Motion Controller

function prototype	clarification
int con_vmove(int ch ,int dir1)	One axis in continuous motion at constant speed
int fast_vmove(int ch , int dir1)	Continuous motion of one axis at high speed
int con_vmove2(int ch1, int dir1,int ch2, int dir2)	Two axes in continuous motion at constant speed

<code>int fast_vmove2(int ch1, int dir1,int ch2, int dir2)</code>	Continuous motion of both axes at fast speeds
<code>int con_vmove3(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3)</code>	Three axes in continuous motion at constant speed
<code>int fast_vmove3(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3)</code>	Three axes for rapid continuous motion
<code>int con_vmove4(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3,int ch4, int dir4)</code>	Four axes in continuous motion at constant speed
<code>int fast_vmove4(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3,int ch4, int dir4)</code>	Four axes for rapid continuous motion

Among

in: **ch, ch1, ch2, ch3, ch4:** axis number of the controlled axis;
dir, dir1, dir2, dir3, dir4: indicates the motion direction of the controlled axis, +1 indicates positive direction; -1 indicates negative direction.

Calling example:

```
con_vmove (1, -1) /* the first axis moves continuously in the
negative direction at its constant speed */ fast_vmove2 (2, 1, 3, -
1) /* the second axis moves quickly and continuously in the
positive direction; the third axis moves quickly and
continuously in the negative direction. */
```

Return Value: These functions return 0 if the call succeeds, or -1 in the case of an error.

(3) (math.) the return-to-origin function

Return-to-home motion means that the controlled axes move at their respective speeds in a given direction until they hit a home signal, a limit switch, or a call to the brake function to stop. The MPC08D library provides a total of eight return-to-home command functions:

Table 10-6 Home Motion Functions

function prototype	clarification
<code>int con_hmove(int ch ,int dir1)</code>	One axis moves back to the origin at constant velocity

MPC08D Motion Controller

int fast_hmove(int ch , int dir)	One axis moves back to the origin in a fast motion
int con_hmove2(int ch1, int dir1,int ch2, int dir2)	Both axes move back to the origin at constant velocity

<code>int fast_hmove2(int ch1, int dir1,int ch2, int dir2)</code>	Both axes move back to the origin in a fast motion
<code>int con_hmove3(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3)</code>	The three axes move back to the origin at constant speed.
<code>int fast_hmove3(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3)</code>	Three axes in a rapid return-to-home motion
<code>int con_hmove4(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3,int ch4, int dir4)</code>	The four axes move back to the origin at constant speed.
<code>int fast_hmove4(int ch1, int dir1,int ch2, int dir2,int ch3, int dir3,int ch4, int dir4)</code>	Four axes in a rapid return-to-home motion

Among

in: `ch, ch1, ch2, ch3, ch4`: axis number of the controlled axis;
`dir, dir1, dir2, dir3, dir4`: indicates the motion direction of the controlled axis, +1 indicates positive direction; -1 indicates negative direction.

Calling example:

```
con_hmove (1, -1) /* the first axis moves back to the origin in
the negative direction at its constant speed */ fast_hmove2 (2, 1,
3, -1)/* the second axis moves back to the origin quickly in the
positive direction; the third axis moves back to the origin
quickly in the negative direction. */
```

Return Value: These functions return 0 if the call succeeds, or -1 in the case of an error.

N O T E : For a successful home return motion, the motion axis should be equipped with a normally open or normally closed home switch (proximity switch or sensor) The control axis will automatically reverse to find the home position if a valid limit signal is detected first during the home return.

10.4.2 Interpolated Motion Functions

Interpolating motion is a linkage of two or three axes according to a certain algorithm, where the controlled axes are started at the same time and arrive at the target position at the same time. The interpolating motion runs at vector speeds, which are categorized into constant vector speeds and trapezoidal vector speeds. The functions related to interpolating motion are:

(1) Linear interpolation function

Linear interpolating motion is a motion in which two or more axes are moving at a vector velocity (constant vector velocity or trapezoidal

The MPC08D library provides six linear interpolation functions:

Table 10-7 Linear Interpolation Functions

function prototype	clarification
<code>int con_line2(int ch1,double pos1,int ch2, double pos2)</code>	Two axes in constant velocity linear motion
<code>int fast_line2(int ch1, double pos1,int ch2 double pos2)</code>	Rapid linear motion of both axes
<code>int con_line3(int ch1, double pos1,int ch2, double pos2,int ch3, double pos3)</code>	Three axes in constant velocity linear motion
<code>int fast_line3(int ch1, double pos1,int ch2, double pos2,int ch3, double pos3)</code>	Rapid linear motion in three axes
<code>int con_line4(int ch1, double pos1,int ch2, double pos2,int ch3, double pos3,int ch4, double pos4)</code>	Four axes in constant velocity linear motion
<code>int fast_line4(int ch1, double pos1,int ch2, double pos2,int ch3, double pos3,int ch4, double pos4)</code>	Rapid linear motion in four axes

Among in:

`ch1, ch2, ch3, ch4`: axis number of the controlled axis.

`pos1, pos2, pos3, pos4`: indicates the distance the controlled axis moves from the current position, positive number indicates positive direction; negative number indicates negative direction, the unit is the number of pulses.

Return Value: These functions return 0 if the call succeeds, or -1 in the case of an error. call example:

```
con_line2 (1, -2000, 3, 1000)
```

```
/* Linear interpolating motion of the first and third axes at
constant vector speed, with the first axis moving in the
negative direction for a distance of 2000 pulses while the
third axis moves in the positive direction for a distance of
1000 pulses*/
```

```
fast_line3 (2, 5000, 3, -1000, 5, 3000)
```

```
/* The second, third, and fifth axes are linearly interpolated at
trapezoidal vector speeds, with the second axis moving in the
```

positive direction for a distance of 5000 pulses and the third axis moving in the negative direction for 1000 pulses.

The distance of the fifth axis is 3000 pulses in the positive direction. */

10.5 braking function

During the motion, if you need to pause or abort the motion of an axis or several axes, you can call the brake function to accomplish this.

Table 10-9 Braking Functions

function prototype	clarification
int sudden_stop(int ch)	Immediate braking of one axis of motion in immediate motion mode
int sudden_stop2(int ch1,int ch2)	Immediate braking of two axes of motion in immediate motion mode
int sudden_stop3(int ch1,int ch2,int ch3)	Immediate braking of the three axes of motion in the immediate motion mode
int sudden_stop4(int ch1,int ch2,int ch3,int ch4)	Immediate braking of the four axes of motion in immediate motion mode
int decel_stop(int ch)	Smooth braking of one axis of motion in immediate motion mode
int decel_stop2(int ch1,int ch2)	Smooth braking of the two axes of motion in immediate motion mode
int decel_stop3(int ch1,int ch2,int ch3)	Smooth braking of the three axes of motion in immediate motion mode
int decel_stop4(int ch1,int ch2,int ch3,int ch4)	Smooth braking of the four axes of motion in immediate motion mode
int move_pause(int ch)	Pause one axis of motion in immediate motion mode
int move_resume(int ch)	Resumption of motion in one axis of motion in the immediate motion mode

Function name: sudden_stop, sudden_stop2, sudden_stop3,

sudden_stop4 Purpose Purpose: To brake the axes immediately in the immediate motion mode.

Syntax: `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: axis number of the
controlled axis;

Description Description: The immediate braking function (or emergency stop function) is useful for all types of transportation in immediate motion mode.

The **sudden_stop** type braking function causes the controlled axis to immediately abort the motion. After this function is executed, the controller immediately stops sending pulses to the motor driver to stop the motion. This function is usually called during an emergency stop. For normal speed motion

(**con_YmoveZ**) can only call the emergency stop function for braking. Return Value: 0 for successful call, otherwise -1.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000, WINDOWS XP

Call example: **sudden_stop2** (1, 4) /* Immediate braking of the first and fourth axes */

Function name: decel_stop, decel_stop2, decel_stop3,

decel_stop4 Purpose Purpose: To smooth the motion of the brake axis in immediate motion mode.

Syntax: **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: axis number of the controlled axis;

Description Description: The **decel_stop** type of braking function is generally used for trapezoidal velocity motion.

(**fast_YmoveZ**) it can make the controlled axis speed from high speed to low speed first (set by **set_profile**) and then stop the motion. Generally, the **decel** type braking function should be called when the motion process needs to stop, so that the fast motion can be aborted smoothly (e.g. **fast_hmove**, **fast_vmove**, **fast_pmove2**, etc.) to avoid overshoot phenomenon. For normal speed movement (**con_YmoveZ**) this type of braking function is invalid.

Return Value: 0 for successful call, otherwise -1. System: WINDOWS

2000, WINDOWS XP

Example of call: **decel_stop** (2) /* smooth braking second axis */

Function name: move_pause

Purpose Purpose: To pause the
motion process in immediate motion
mode. Syntax: `int move_pause (int ch)`

`ch`: the axis number of the axis to be suspended;

Description Description : If you need to pause an axis during
immediate motion, you can call this function, and then call
`move_resume` afterwards to resume the continued operation.
For fast linear interpolation movement, fast point movement,
fast continuous movement, fast return to origin movement, etc.,
call "`move_pause`" function, the control axis will decelerate
with the set trapezoidal speed until it stops; call
"`move_resume` After calling "`move_resume`"
function, the control axis decelerates with the set
trapezoidal speed until it stops.

Speed accelerates to high speed.

Corresponding to the normal speed linear interpolation motion, normal speed point motion, normal speed continuous motion, normal speed return to home position motion, call **"move_pause"** function, the control axis stops the motion immediately; call **"move_resume"**, the control axis will move at normal speed immediately. After calling **"move_resume"**, the control axis will move at normal speed immediately.

Return value: if the call succeeds, **move_pause** returns 0, otherwise -1.

System: WINDOWS 2000, WINDOWS XP System: **WINDOWS 2000, WINDOWS XP**

Call example: **move_pause (1)**

Function name: move_resume

Purpose Purpose: To resume a suspended motion process in immediate motion mode. Syntax

: int move_resume (int ch)

ch: The axis number of the motion axis to be restored;

Description Description: If you want to pause in the middle of an immediate motion, you can call the **move_pause** function, and then call **move_resume** afterwards to resume and continue running.

For fast linear interpolation movement, fast point movement, fast continuous movement, fast return to origin movement, etc., call **"move_pause"** function, the control axis will decelerate with the set trapezoidal speed until it stops; call **"move_resume"** After calling **"move_resume"**, the control axis will accelerate to high speed with trapezoidal speed.

Corresponding to the normal speed linear interpolation motion, normal speed point motion, normal speed continuous motion, normal speed return to home position motion, call **"move_pause"** function, the control axis stops the motion immediately; call **"move_resume"**, the control axis will move at normal speed immediately. After calling **"move_resume"**, the control axis will move at normal speed immediately.

Return value: if the call succeeds, **move_resume** returns 0, otherwise -1.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000,
WINDOWS XP

Call example: `move_resume (1)`

10.6 Digital I/O Operation Functions

Table 10-10 Digital IO Functions

function prototype	clarification
<code>int checkin_byte(int cardno)</code>	Read the status of all general-purpose inputs of the board
<code>int checkin_bit(int cardno, int bitno)</code>	Read the status of one of the card's general-purpose inputs.

<code>int outport_byte(int cardno,int bytedata)</code>	Write all common outputs of the board
<code>int outport_bit(int cardno,int bitno,int status)</code>	Write to one of the board's general-purpose outputs
<code>int check_sfr(int cardno)</code>	Reads so external limits, home, alarms, and Z-pulses signal state
<code>int check_sfr_bit(int cardno,int bitno)</code>	Reads external limit, home, alarm, and Z-pulse signals The state of a signal in the

Function name: checkin_byte

Purpose Purpose: To read the status of all common inputs of the card.

Syntax: `int checkin_byte(int cardno);`

`cardno`: card number, i.e., the local ID number of the card set by the user, the value ranges from 1 to 1.
to the card's maximum number.

Description: This function allows you to read the status of all 24 general purpose inputs. The wiring is shown in **MPC08D User.pdf**, section "Connection Methods for General Purpose Inputs and Outputs".

Return Value: Return the status of the input port, bits D0 to D23 of the return value correspond to 24 input ports, a 1 means the input port is ON, a 0 means the input port is OFF; if there is an error, -1 is returned.

System: WINDOWS 2000, WINDOWS XP

System: WINDOWS 2000,

See also: `checkin_bit`

Function name: checkin_bit

Purpose Purpose: To read the status of a common input bit of the card extension.

Syntax: `int checkin_bit(int cardno,int bitno);`

cardno: card number, i.e., the local ID number of the card set by the user, the value ranges from 1 to 1.
to the card's maximum number;

bitno: Indicates the first bit, value range is 1~26.

Description This function can read the status of an input port. This method is in the section "Connection Methods of Common Inputs and Outputs" in "MPC08D User.pdf".

Return Value: Returns the state of the input port, a return value of 1 means that the input port is in the ON state, and 0 means that the input port is in the OFF state.

Indicates that the input port is OFF; returns -1 if an error occurs System: WINDOWS 2000, WINDOWS XP

See also: `checkin_byte`

Function name: `outport_byte`

Purpose: To set the status of the card's general-purpose output port.

Syntax: `int outport_byte(int cardno,int bytedata);`

`cardno`: card number, i.e., the local ID number of the card set by the user, the value ranges from 1 to 1.
to the card's maximum number;

`bytedata`: status byte, each corresponds to each output port;

D e s c r i p t i o n : The MPC08D card provides up to 26 general-purpose opto-isolated outputs for user use through expansion. The general-purpose outputs 1 to 10 are output from the 62-core cable of the MPC08D card, and the general-purpose outputs 11 to 26 are output from the expansion cable. All general-purpose outputs have 100mA driving capability. Through this function, you can set the status of these 26 outputs. Wiring see "MPC08D User.pdf" in the "general-purpose inputs and outputs of the connection method" section.

Return Value: 0 for correct setting; -1 if error occurs System: WINDOWS 2000, WINDOWS XP

See: `outport_bit`

Function name: `outport_bit`

Purpose: To set the switching status of one of the general-purpose outputs of the board.

Syntax: `int outport_bit(int cardno,int bitno,int status);`

`cardno`: card number, i.e., the local ID number of the card set by the user, the value ranges from 1 to 1.
to the card's maximum number.

`bitno`: Indicates the first output port, the

value range is 1 to 26. Status: The set status (1: ON; 0: OFF)

Description: The MPC08D card provides up to 26 general-purpose opto-isolated outputs for user use through expansion. The general-purpose outputs 1 to 10 are output from the 62-core cable of the MPC08D card, and the general-purpose outputs 11 to 26 are output from the expansion cable. All general-purpose outputs have 100mA driving capability. Through this function, you can set the status of these 26 outputs. Wiring see "MPC08D User.pdf" in the "general-purpose inputs and outputs of the connection method" section.

Return Value: 0 for correct setting; -1 if there
is an error System: WINDOWS 2000,
WINDOWS XP

See: `outport_byte`

Function name: `check_sfr`

P u r p o s e : To read information from board inputs,
including external axis alarms, limits, home position and
board alarm signals.

Syntax: `int check_sfr(int cardno);`

cardno: card number, i.e., the local ID number of the card set
by the user, the value ranges from 1 to 1.
to the card's maximum number.

Call Example: `check_sfr(1) /* Read external alarm, limit and home
signal of card No.1 */` **Description:** The MPC08D Motion

Controller has a register to save the 21 dedicated inputs of the board.

The `check_sfr` function reads the contents of this register
to find out the status of each dedicated input (e.g., limit,
home, alarm, Z-pulse, etc.) each axis. The `check_sfr` function
reads the contents of this register to understand the status
of each dedicated input port. If the external alarm, limit,
and home signal of an axis is set to invalid, the
corresponding dedicated signal input port can be used as a
general-purpose input port. The definition of each bit of the
register is shown in Table 10-11 below, and the function
`check_sfr` is used to read the status of all dedicated input ports
of the board.

Return Value : Return to the state of external switching signal, the
corresponding bit is 1, which means the input port is in
high level state, 0, which means the input port is in low
level state; if there is an error, then -1 will be returned.

System: WINDOWS 2000, WINDOWS XP **System:** WINDOWS 2000,
WINDOWS XP

See: Table 10-11 Special Register Definitions

status bit	Dedicated signals	clarification
D20	ALM	General-purpose input when

		ALM4 is invalid
D19	Z4	Z4 cannot be used as a general purpose input
D18	ALM4	ALM4 is used as a general-purpose input when invalid.
D17	ORG4	ORG4 is used as a general-purpose input port when it is invalid.
D16	EL4-	EL4 - Invalid as a general-purpose input port
D15	EL4+	Used as a general-purpose input when EL4+ is invalid.
D14	Z3	Z3 cannot be used as a general purpose input
D13	ALM3	General-purpose input when ALM3 is invalid
D12	ORG3	ORG3 is used as a general-purpose input port when it is invalid.
D11	EL3-	EL3 - Invalid as a general-purpose input port
D10	EL3+	Used as a general-purpose input when EL3+ is invalid.
D9	Z2	Z2 cannot be used as a general purpose input

D8	ALM2	When ALM2 is invalid, it is used as a general-purpose input port.
D7	ORG2	When ORG2 is invalid, it is used as a general-purpose input port.
D6	EL2-	EL2 - Invalid as general-purpose input port
D5	EL2+	Used as a general-purpose input when EL2+ is invalid.
D4	Z1	Z1 cannot be used as a general-purpose input
D3	ALM1	When ALM1 is invalid, it is used as a general-purpose input port.
D2	ORG1	When ORG1 is invalid, it is used as a general-purpose input port.
D1	EL1-	EL1 - Invalid as general-purpose input port
D0	EL1+	Used as a general-purpose input when EL1+ is invalid.

Function name: `check_sfr_bit`

P u r p o s e : To read the card inputs, including the dedicated inputs for each axis and the card alarm inputs.

Syntax: `int check_sfr_bit(int cardno,int bitno);`

cardno: card number, i.e., the local ID number of the card set by the user, the value ranges from 1 to 1.
to the card's maximum number.

bitno: Indicates the first input port, the value range is 1~21.

Call example: `check_sfr_bit(1,1) /* Read the external limit signal of card No.1 */`

Description: The MPC08D Motion Controller has a register that holds the status of 21 dedicated inputs on the board, including those dedicated to each axis (e.g., limit, home, alarm, Z-pulse, etc.) The `check_sfr` function can read the content of this register to know the status of each dedicated input. If the external alarm, limit and home signal of an axis is set to invalid, the corresponding dedicated signal input port can be used as a general-purpose

input port. The definition of each bit of the register is shown in Table 10-11 below, and the `check_sfr_bit` function can be used to read the status of a certain port.

Return Value: return to the state of external switching signal, the return value of 1 indicates that the input port is in a high level state, 0 indicates that the input port is in a low level state; if there is an error, then return to -1.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000, WINDOWS XP

See also: `enable_sd`, `enable_el`, `enable_org`

10.7 Special Function Functions

The MPC08D offers backlash compensation, dynamically changing target position, and a power-down protected data area.

Read and write functions. The interface for each function is as follows.

10.7.1 Reverse Gap Compensation

Table 10-12 Reverse Gap Compensation Functions

function prototype	clarification
int set_backlash (int ch, double (backlash))	Setting the compensation value for the gap created by the mechanism commutation
int start_backlash (int ch)	Start gap compensation
int end_backlash (int ch)	Termination gap compensation

Function Name: **set_backlash, start_backlash, end_backlash**

Purpose: To set the compensation value for
compensating the backlash caused by the
mechanism commutation with **set_backlash**. Start
backlash compensation with **start_backlash**.
Stop backlash compensation with **end_backlash**.

Syntax: int set_backlash (int ch, double backlash)

Method: int set_backlash (int ch, double
backlash) int start_backlash (int ch)
int end_backlash (int ch)

ch: control axis number.

backlash: the value of the gap due to the commutation of
the mechanism, in number of pulses, which must be
greater than or equal to zero.

Description: The function **set_backlash** sets a compensation
value in order to eliminate positional errors due to
commutation of the mechanism. The **start_backlash**
function starts the backlash compensation of the control
axes. The backlash compensation of the control axes is
terminated by the call to **end_backlash**. The **set_backlash**
function only sets the compensation value, the real
compensation value does not come into effect until the
start_backlash function is called. The **set_backlash** function
should be called before **start_backlash**, otherwise the
system uses the default compensation value (default 20

Return values: `set_backlash`, `set_backlash`, and `end_backlash` are returned if setup is successful

0, otherwise returns -1.

System: WINDOWS 2000, WINDOWS XP

System: **WINDOWS 2000,**

WINDOWS XP

See also:

Call example: `set_backlash (1, 12)`

`start_backlash (1)`

`end_backlash (1)`

10.7.2 Dynamic change of target position

Table 10-13 Dynamically Changing Target Position Function

function prototype	clarification
int change_pos(int ch, double pos)	Dynamically set target position during motion

Function name: change_pos

Purpose: Use the **change_pos** function to dynamically change the target position in relative position mode. During single-axis point movement, in normal speed mode or trapezoidal speed mode, if the user finds that the end position of the motion command needs to be changed, the user can call "**change_pos**" to dynamically change the end position before or after the end of the point movement. The system will automatically move according to the new end position. The end position of the previous user issued motion instruction is the same as the end position of the previous user issued motion instruction.

(**fast_pmove**, **con_pmove**) as the starting point. This function can be called multiple times to change the target position.

Syntax: **int change_pos(int ch, double pos);**

Parameter Description:

ch : Axis number;

pos: new relative target position

Description Description: Dynamically set the target position during motion.

Return Value: Returns 0 if the call is correct, -1 if it is incorrect. System:

WINDOWS 2000, WINDOWS XP

See also:

Call example: change_pos (1, 1000);

10.7.3 Power-down protection for data area

read/write function

Table 10-14 Power-Down Protectable Data Area Read/Write Functions

function prototype	clarification
<code>int write_password_flash(int cardno, int no, long data,long password)</code>	Data area write function with password protection
<code>read_password_flash(int cardno, int no, long *data,long password)</code>	Data area read function with password protection
<code>clear_password_flash(int cardno,</code>	Erase data area with password protection

long password)	
write_flash(int cardno,int piece, int no, long data)	Write function for the general data area
read_flash(int cardno,int piece,int no,long *data)	General Data Area Read Functions
clear_flash(int cardno, int piece)	Erase general data area

Function name: write_password_flash

Purpose: Used to write to the data area with password protection.

Before writing to the data area, the area must be erased, otherwise the data to be written cannot be written. The erase operation is valid for the whole chip, and the specified area of the chip is erased to make all the bits in the 64K byte space of the area as "1". The write operation operates on a 4-byte contiguous address.

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

Parameter Description:

cardno : Card number;

no: the number of the write address, in the range 0-16383, where 16383 is fixed as the read/write password storage area;

data: data to be written;

password: read/write check password; i.e., the data value stored in the address area of 16383, the password is also checked when the write address number is 16383;

Description Description: Used to perform write operations on data areas with password protection.

Return Value: 0 --write success return

-1 -- Write failure due to parameter error, return

-2 -- Write failure due to incorrect

password checking, return system: WINDOWS

2000, WINDOWS XP

See also: clear_password_flash

Call example: `write_password_flash(1, 1000, 0xffff0, 0xfffff);`

Function name: `read_password_flash`

Purpose: Read operation of the data area with password protection.

The read operation operates on a 4-byte contiguous address.

The read/write password needs to be verified and is responsible for not returning the correct value.

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

Parameter Description:

cardno : Card number;

no: the number of the read address, the range is 0-16382, of which address 16383 is fixed as the read/write password storage area, and the external interface is not allowed to read the operation;

data: store the read data;

password: read/write checksum password; i.e., the data value stored in address area No. 16383. Description : Used for read operation of the data area with password protection.

Return Value: 0 --Read successfully returned

-1 --Read failure due to parameter error, return

-2 -- Read failure due to incorrect

verification password, return system:

WINDOWS 2000, WINDOWS XP

See also:

Call example: `read_password_flash (1, 1000, &data, 0xffffffff);`

Function name: clear_password_flash

Purpose: Used to erase the data area with password protection. **Before writing to the data area, the area must be erased, otherwise the data to be written cannot be written. The erase** operation is valid for the whole chip, erase the area of the specified chip, so that all the bits of the 64K byte space of the area will be "1".

Syntax: **clear_password_flash(int cardno, long password);**

Parameter Description:

cardno : Card number;

password: read/write check password; i.e., the data value stored in the address area of 16383, the password is also checked when the write address number is 16383;

Description Description: Used to perform erase operation on the data area with password protection.

Return Value: 0 --Erase successfully returned

-1 -- Erase failed due to parameter error, return

-2 -- Erase failure due to incorrect

verification password, return system:

WINDOWS 2000, WINDOWS XP

See also: write_password_flash

Call example: `clear_password_flash (1,0xfffff);`

Function name: write_flash

Purpose: Used to write to the general data area. **Before writing to the data area, the area must be erased, otherwise the data to be written cannot be written.** The erase operation is valid for the whole chip, and the specified area of the chip is erased so that all bits in the 64K byte space of the area are "1". The write operation operates on a 4-byte contiguous address.

Syntax: **`int write_flash(int cardno,int piece, int no, long data);`**

Parameter Description:

cardno : Card number;

piece -- Piece number, range 1-15, size of each piece is 64K;

no -- The number of the write address, in the range 0-16383;

data - data to be written;

Description Description : Used for write operation to general data area.

Return Value: **0** -- Write success return

-1 - Write failure due to

parameter error, return system

System: WINDOWS 2000, WINDOWS XP

See also: `clear_flash`

Call example: `write_flash (1, 2, 1000, 0xfffe0);`

Function name: read_flash

Purpose: Used to perform read operations on the general data area.
The read operation operates on a 4-byte contiguous address.

Syntax: **`int read_flash(int cardno,int piece,int no,long *data);`**

Parameter Description:

cardno : Card number;

piece : **piece** number, range 1-15, the size of each piece is 64K;

no: the number of the read address,

the range is 0-16383; **data**: store the

Description Description: Used to perform read operations on the general data area.

Return Value: 0 --Read successfully returned

-1 --Read failure due to
parameter error, return system

System: WINDOWS 2000, WINDOWS XP

See also:

Call example: `read_flash(1, 2, 1000, &data);`

Function name: `clear_flash`

Purpose: Used for general data area erase operation. **Before writing to the data area, the area must be erased, otherwise the data to be written cannot be written. The erase operation is valid for the whole chip, and the area with the specified first address is erased, so that all the bits in the 64K byte space of the area will be '1'.**

Syntax: `int clear_flash(int cardno, int piece);`

Parameter Description:

cardno : Card number;

piece : Area number, range 1-15;

Description Description : Used for general data area for erase operation.

Return Value: 0--- Erase successful
return

-1 -- Erase failure due to
parameter error, return system:
WINDOWS 2000, WINDOWS XP

See also: `write_flash`

Call example: `clear_flash(1, 0xfffff);`

10.8 Position and state setting functions

Table 10-15 Query Functions

function prototype	clarification
<code>int get_max_axe(void)</code>	Read the total number of axes of the motion controller
<code>int get_board_num(void)</code>	Read the number of boards
<code>int get_axe(int board_no)</code>	Read the number of axes on the board
<code>int check_IC(int cardno)</code>	Queries the local ID number of the controller set by the user

MPC08D Motion Controller

Programming Manual

int get_abs_pos (int ch, double *pos)	Returns the absolute position of an axis
double get_conspeed(int ch)	Read the constant speed of each axis
int get_profile(int ch,double	Read trapezoidal speed for each axis

<code>*vl,double *vh,double *ad,double *dc)</code>	
<code>double get_vector_conspped(void)</code>	Read vector constant velocity
<code>int get_vector_profile(double *vec_vl,double *vec_vh,double *vec_ad,double *vec_dc)</code>	Read vector trapezoidal velocity
<code>double get_rate(int ch);</code>	Get the current speed of the axis
<code>int get_cur_dir (int ch)</code>	Returns the current direction of motion of an axis
<code>int check_status (int ch)</code>	Checking the status value of an axis
<code>int check_done (int ch)</code>	Detecting the completion of an axis movement
<code>int check_limit (int ch)</code>	Detecting the closure of an axis-specified limit switch
<code>int check_home(int ch)</code>	Checking that an axis has reached the home switch position
<code>int check_alarm(int ch)</code>	Checking an axis for external alarm signals
<code>int check_card_alarm(int cardno)</code>	Check the external alarm signal of the board
<code>int get_done_source(int ch,long *src)</code>	Detecting the cause of an axis stop

Function name: get_max_ave

Purpose: `get_max_ave` is used to read the total number of control axes. Syntax :

`int get_max_ave (void)`

Return Value: `get_max_ave` Returns the total number of control axes.

System: WINDOWS 2000, WINDOWS

XP

cf. See:

invocation

example:

```
int max_ave_num;
```

max_ace_num=get_max_ace()

Function name: get_board_num

Purpose: get_board_num is used to read the number of motion controllers installed in the computer. Syntax : int
get_board_num (void)

Return Value: get_board_num Returns the total number of boards.

System: WINDOWS 2000, WINDOWS XP

System: WINDOWS 2000,

WINDOWS XP

cf. See:

invocation

example:

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

Function name: get_ave

Purpose: `get_ave` is used to read the number of axes on a board. Syntax

```
: int get_ave (int board_no)
```

Return Value: `get_ave` Returns the number of axes on the board.

System: WINDOWS 2000, WINDOWS XP

System: WINDOWS 2000,

WINDOWS XP

cf. See:

invocation

example:

```
int ave_num.
ave_num=get_ave(1) //read the number of axes of the first card
```

Function name: check_IC

Purpose: To check the local ID number of a card. Syntax

```
int check_IC (int cardno)
```

cardno: card number, this parameter is not the same as the local ID number of the card in the previous interface function, it only indicates the sequential number of the card in the computer, the value range from 1 to the maximum number of cards. This function is generally used in the computer installed only one control card, read its local ID number.

Description : This function can query the local ID number of the motion controller. Return Value: Returns the ID number of the current motion controller.

System: WINDOWS 2000, WINDOWS XP Programming Manual System: WINDOWS 2000, WINDOWS XP

See :

Call example: `int ic=check_IC(1)`

Function name: get_abs_pos

Purpose Use `get_abs_pos` to read an absolute position relative to an initial or origin position. Syntax : `int get_abs_pos (int ch, double *abs_pos)`

`ch`: axis number of the read position;

`abs_pos`: a double-precision pointer to an absolute position;

Description : the function `get_abs_pos` gets the current absolute position of the specified axis, if it has executed the return-to-home motion and called the `reset_pos` function, then this absolute position is relative to the home position; if it has not executed the `reset_pos`, then this absolute position is relative to the position of the power-on time.

Return Value: `get_abs_pos` returns 0 if the call is successful, or -1 in the case of an error.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000, WINDOWS XP

Call example: `temp=get_abs_pos(1, &abs_pos)`

Function name: `get_conspeed`

Purpose: Use the `get_conspeed` function to get the constant speed set for an axis. Syntax : double
`get_conspeed (int ch)`

`ch`: the number of the control axis.

Description : Use the `get_conspeed` function to get the constant speed set by the specified axis.

Return value: function `get_conspeed` returns the constant speed value of the specified axis, and returns -1 in case of error.

system: WINDOWS 2000, WINDOWS XP

Annotation:

See also:

Calling example: `double speed;`

`speed=get_conspeed(2)`

Function name: `get_profile`

Purpose: To read the values of the parameters of the ladder speed with `get_profile`. Purpose: Use `get_profile` to read the value of each parameter of trapezoidal speed.

S y n t a x : `int get_profile(int ch,double *vl,double *vh,double *ad,double *dc) double *ls: pointer to start low; double *hs:: pointer to target high speed; double *accel: pointer to acceleration value; double *dc:`

pointer to deceleration value,

Description **Description:** The **get_profile** function returns the low, high, plus and minus speed values of the trapezoidal speed of an axis via a pointer.

Return value: if the call succeeds, **get_profile** returns 0, otherwise it returns -1. **System:** WINDOWS 2000, WINDOWS XP

See also:

Call example: **get_profile**(3, &ls, &hs, &accel, &dec)

Function name: get_vector_conspped

Purpose: The `get_vector_conspped` function to read the vector speed in constant speed mode. Syntax : `double get_vector_conspped (void)`

Description: The `get_vector_conspped` function reads the vector speeds of the following 2-axis or multi-axis interpolated kinematic functions: `con_line2`, `con_line3`, `con_line4` and so on.

Return Value: If the call is successful, returns the vector speed read off. System:

WINDOWS 2000, WINDOWS XP

Annotation:

See also: `set_conspped`, `set_profile`

Call example: `vec_conspped= get_vector_conspped()`

Function name: get_vector_profile

Purpose Purpose: To get the vector trapezoidal velocity parameter value using `get_vector_profile`;

Language. Method : `int get_vector_profile(double *vec_vl,double *vec_vh,double`

`*vec_ad,double *vec_dc);`

`*vec_fl`: pointer to vector low speed;

`*vec_fh`: pointer to vector high speed;

`*vec_ad`: pointer to vector acceleration;

`*vec_dc`: pointer to vector deceleration.

Description: Function `get_vector_profile` reads the vector trapezoidal velocity of `fast_line2`, `fast_line3` and other fast interpolation functions.

Return Value: The `get_vector_profile` function returns 0 if the call was successful, or -1 in the case of an error.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000, WINDOWS XP

See also:

Call example: `get_vector_profile (&vec_fl, &vec_fh, &vec_ad, &vec_dc)`

Function name: get_rate

Purpose: Use the `get_rate` function to get the actual speed of the current motion of an axis. Syntax : double `get_rate (int ch)`

`ch`: control axis number;

Description Description: The function `get_rate` reads the current actual running speed of the control axis.

Return Value: The function `get_rate` returns the current running speed of the specified axis, unit: pulses per second.

(pps) function call error returns -1.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000,
WINDOWS XP

See also See also:

`get_profile` call example:

`double speed;`

`speed=get_rate(2)`

Function name: `get_cur_dir`

Purpose Purpose: To get the
current direction of motion of an axis.

Syntax: `int get_cur_dir (int ch);`

`ch`: the axis to be queried;

Return value: if the function call fails, the return value is -2;
otherwise, it returns -1 to indicate that the current
movement direction is negative, 1 to indicate that the
current movement direction is positive, and 0 to indicate
that the movement stops.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000,
WINDOWS XP

See also:

Call example: `get_cur_dir(1); /* Get the current direction of motion of the
1st axis */`

Function name: `check_status`

Purpose Purpose: To read
the current status of an axis.

Syntax: `int check_status (int ch)`

`ch`: axis number of the state being read.

Description: The `check_status` function reads the status of the
specified axis. each axis of the MPC08D controller has a
32-bit (double-word) status value, which is used to query
the operating status of the axis. Axis Status

See Table 6-2 for the meanings of each of
the state words. The correct dedicated

input port status can be returned only if the commands "enable_org", "enable_limit", "enable_alm", etc. are called to enable the corresponding dedicated signals "enable_org", "enable_limit", "enable_alm", "enable_card_alm", etc. to enable the corresponding dedicated signals to return to the correct state of the dedicated input port.

Return Value: **check_status** returns the status of the specified axis if the call succeeds, or -1 in the case of an error.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000, WINDOWS XP

See also:

Call example: **ch_status=check_status (2)**

Function name: check_done

Purpose Purpose: The **check_done** function is used to check whether the motion of the specified axis has been completed.

Syntax: `int check_done (int ch)`

ch: Axis number inspected.

Description Description: The **check_done** function checks whether the specified axis is in motion or at rest.

Return Value: **check_done** returns 1 if the specified axis is in motion, 0 if the specified axis is at rest, and -1 if the function fails.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000, WINDOWS XP

See also:

Function name: check_limit

Purpose: The **check_limit** function is used to check if an axis limit signal is valid. Syntax : `int check_limit (int ch)`

ch: Axis number inspected;

Description : MPC08D Motion Controller is configured with two limit switch inputs for each axis, which are positive limit signal input and negative limit signal input. The function **check_limit** is used to detect the limit switch status of the specified axis and return whether the limit signal of the specified axis is valid or not. Only when the "enable_limit" instruction is called to enable the corresponding dedicated signal, can the correct dedicated input port status be returned.

Return value: if **check_limit** returns 1, it means the positive limit signal is valid, -1 means the negative limit signal is valid, 0 means both positive and negative limits are invalid, 2 means both positive and negative limits are valid, if there is an error in the call, then -3 is returned.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000, WINDOWS XP

Example call: `status=check_limit (1)`

Function name: check_home

Purpose: The **check_home** function is used to check if an axis home signal is valid. Syntax : `int`

ch: Axis number inspected.

Description : MPC08D Motion Controller is configured with one home switch input port for each axis. The function **check_home** is used to check whether the home signal of the specified axis is valid or not, and the correct state of the dedicated input port can be returned only if the corresponding dedicated signal is enabled by calling the "enable_org" instruction.

Return value: if **check_home** returns 1, it means the home signal is valid, if it returns 0, it means the home switch is invalid, and if there is an error in the call, it returns -3.

System: WINDOWS 2000, WINDOWS XP

System: WINDOWS 2000,

WINDOWS XP

Example call: `status=check_home (1)`

Function name: check_alarm

Purpose: Use the `check_alarm` function to check whether the external alarm signal is valid. Syntax: `int`

`check_alarm (int ch)`

`ch`: axis number.

Description: The MPC08D motion controller has one alarm switch input for each axis. The `check_alarm` function is used to check the status of the alarm switches of the specified axes and return whether there is a valid alarm signal input to the board. Only when the "`enable_alm`" instruction is called to enable the corresponding dedicated signal, can the correct dedicated input port status be returned.

Return Value: If `check_alarm` returns 1, it means the alarm signal is valid, if it returns 0, it means the alarm signal is invalid, and if there is an error in the call, it returns -3.

System: WINDOWS 2000, WINDOWS XP

System: WINDOWS 2000,

WINDOWS XP

See also:

Function name: check_card_alarm

Purpose: The `check_card_alarm` function is used to check whether the card alarm signal of the board is valid. Syntax :

`int check_card_alarm (int cardno)`

`cardno`: the number of the card being checked.

Description: The MPC08D motion controller board is configured with an alarm signal input port. The `check_card_alarm` function is used to check the alarm signal status of the card. Only when the "`enable_card_alm`" instruction is called to enable the corresponding dedicated signal, can the correct dedicated input port status be returned.

Return Value: If `check_card_alarm` returns 1, it means the alarm

signal of the card is valid, if it returns 0, it means the alarm signal of the card is invalid, if the call is wrong, it returns -3.

System: WINDOWS 2000, WINDOWS XP
WINDOWS XP

System: WINDOWS 2000,

See also:

10.9 error code handler

The motion controller can return the last 10 error states so that the user can understand the status of the system in detail.

The meanings of the error codes are shown in Table 9-2. There are three error code operation functions, as shown in Table 10-16.

Table 10-16 Error Code Handling Functions

function prototype	clarification
<code>int get_err(int index,int* data)</code>	Get Error Code
<code>int get_last_err()</code>	Get last error code
<code>int reset_err()</code>	Clear all errors

Function name: get_err

Purpose: The query is used to obtain the error codes of the last ten errors generated during the execution of previous commands.

Syntax: `int get_err(int index,int* data);`

index: store the index number of the error code, the index number of the most recent error code is 1, and so on backwards to 10.

data: save the returned error code.

Description: The `get_err` function queries the error code of the last ten errors generated during the execution of a previous instruction. With the return value and the error code table 11-2, users can quickly find the cause of program errors.

Return value: the error code of the most recent **index** error, -1 means that the **Index** parameter has exceeded the limit of 1 to 1.

10 range or there is no error in the system.

System: WINDOWS 2000, WINDOWS XP **System:** WINDOWS 2000, WINDOWS XP

See: Table 9-2 Error

Code Meanings Calling

Example: `int err;`

```
get_err(2,&err); /* Get the most recent 2nd generated error code
*/;
```

Function name: get_last_err

Purpose **Purpose:** To get the error code of the last error generated during the execution of a previous instruction. **Syntax :** `int get_last_err();`

Return value: the error code of the latest error, 0 means no error. **Call example:** `int err;`

System: WINDOWS 2000, WINDOWS XP

System: WINDOWS 2000,

WINDOWS XP

Table 9-2 Meaning of Error Codes

Function name: reset_err

Purpose Purpose: Used to clear the last ten error codes. After calling this function, call

The `get_last_err()` or `get_err()` functions will both return 0. Syntax : `int reset_err();`

Return Value: 0 for success, -1 for failure

System: WINDOWS 2000, WINDOWS XP **System:** WINDOWS 2000, WINDOWS XP

See also See: `get_last_err()`,
`get_err()` call example: `int err;`
`err=reset_err();`

10.10 Controller version fetch function

Table 10-17 Version Reading Functions

function prototype	clarification
<code>int get_lib_ver(long* major,long *minor1,long *minor2)</code>	Query the version of the library
<code>int get_sys_ver(long* major,long *minor1,long *minor2)</code>	Query the driver version
<code>int get_card_ver(long cardno,long *type,long* major,long *minor1,long *minor2)</code>	Query the version of the board

Function name: `get_lib_ver`

Purpose Purpose: To query the version of the library.

Syntax: `int get_lib_ver(long* major,long *minor1,long *minor2);` long
* major: pointer to the main version number of the library.

long * minor1: Pointer to function library minor version number 1.

long * minor2: pointer to library minor version number 2. Call example: `get_lib_ver(&major, &minor1, &minor2);`

Description Description : This function can query the version number of the function library of the motion controller, the function library version must match the driver version and

the firmware version of the board.

Return value: 0.

System: WINDOWS 2000, WINDOWS XP

System: **WINDOWS 2000,**

WINDOWS XP

See also:

Function name: get_sys_ver

Purpose Purpose: To query the driver version.

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

* major: pointer to the driver major version number.

long * minor1: Pointer to driver minor version number 1.

long * minor2: pointer to driver minor version

number 2. Call example: get_sys_ver(&major, &minor1, &minor2);

Description Description : This function can query the version number of the motion controller driver program, the driver version must match the function library version and the board firmware version.

Return Value: 0 for successful call, otherwise -1.

System: WINDOWS 2000, WINDOWS XP System: WINDOWS 2000, WINDOWS XP

See also:

Function name: get_card_ver

Purpose Purpose: To query the version of the board.

Syntax. Method: int get_card_ver(long cardno,long * type,long * major , long

*minor1 ,long *minor2);

long cardno: card number, i.e., the local ID number of the card set by the user, the value ranges from 1 to the maximum number of the card;

long * type: card type number, 2 for MPC08D. long * major: major version number returned.

long * minor1: the returned minor version number 1. long * minor2: the returned minor version number 2.

Call example: get_card_ver(1, &type, &major, &minor1, &minor2);

Description Description : This function can query the type and version number of the motion controller, the board

firmware version must match the function library version
and driver version.

Return Value: 0 for successful call,
otherwise -1. System: WINDOWS
2000, WINDOWS XP

See also:

11 function index

auto_set	46
check_alarm	84
check_card_alarm	84
check_done	83
check_home	83
check_IC	80
check_limit	83
check_sfr	70
check_sfr_bit	71
check_status	82
checkin_bit	68
checkin_byte	68
change_pos	73
clear_flash	75
clear_password_flash	75
con_hmove	62
con_hmove2	62
con_hmove3	64
con_hmove4	64
con_line2	65
con_line3	65
con_line4	65
con_pmove	60
con_pmove2	60
con_pmove3	60
con_pmove4	61
con_vmove	61
con_vmove2	61
con_vmove3	62

con_vmove4	-----	62
decel_stop	-----	65
decel_stop2	-----	65
decel_stop3	-----	65
decel_stop4	-----	65
enable_alm	-----	50
enable_card_alm	-----	51
enable_el	-----	50
enable_org	-----	51
end_backlash	-----	72
fast_hmove	-----	63
fast_hmove2	-----	63
fast_hmove3	-----	64
fast_hmove4	-----	64
fast_line2	-----	65
fast_line3	-----	65
fast_line4	-----	65
fast_pmove	-----	60
fast_pmove2	-----	60
fast_pmove3	-----	61
fast_pmove4	-----	61
fast_vmove	-----	61
fast_vmove2	-----	62
fast_vmove3	-----	62
fast_vmove4	-----	62
get_abs_pos	-----	80
get_axe	-----	79
get_board_num	-----	78
get_card_ver	-----	87
get_conspeed	-----	80
get_cur_dir	-----	82
get_err	-----	85
get_last_err	-----	85

get_lib_ver	86
get_max_axe	78
get_profile	80
get_rate	81
get_sys_ver	87
get_vector_conspeed	81
get_vector_profile	81
init_board	46
move_pause	66
move_resume	66
outport_bit	69
outport_byte	69
read_flash	76
read_password_flash	75
reset_err	86
reset_pos	59
set_abs_pos	59
set_alm_logic	52
set_backlash	72
set_card_alm_logic	53
set_conspeed	55
set_dir	49
set_el_logic	52
set_home_mode	49
set_maxspeed	55
set_org_logic	53
set_outmode	48
set_profile	56
set_s_curve	58
set_s_section	58
set_vector_conspeed	56
set_vector_profile	57
start_backlash	72

sudden_stop	-----	65
sudden_stop2	-----	65
sudden_stop3	-----	65
sudden_stop4	-----	65
write_flash	-----	76
write_password_flash	-----	74

12 appendix diary

12.1 P62-05 Adapter Board Pin Definitions

The P62-05 adapter board integrates the external pins for all dedicated and some general-purpose input and output signals of the MPC08D. The components are shown below in mm, including the mounting aperture: $\phi 3\text{mm}$.

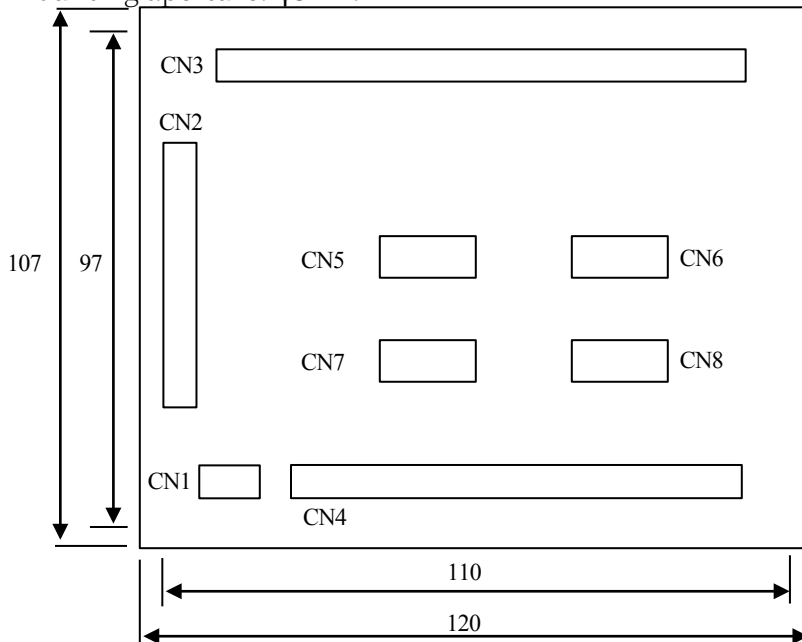


Figure 12-1 P62-05 Schematic

Table 12-1 P62-05 Interface Definitions

interface connector	functionality
CN1	24V switching power supply interface
CN2	DB62 interface to the motion controller
CN3	Limit, home, board alarm and 8 universal inputs (note: terminal +24V) and GND are board output 24 power supplies that can power sensors with dedicated input signals.)
CN4	10 Universal Outputs
CN5	Axis 1 Pulse, Direction and Axis Alarm, Z Pulse Signal Interface
CN6	Axis 2 Pulse, Direction and Axis Alarm, Z Pulse Signal Interface
CN7	Axis 3 Pulse, Direction and Axis Alarm, Z Pulse Signal Interface
CN8	Axis 4 Pulse, Direction and Axis Alarm, Z Pulse Signal Interface

12.2 P37-05 Adapter Board Pin Definitions

When using the MPC08D Motion Controller with the P62-05 adapter board, if more general purpose IO interfaces are required, use the P37-05 to connect the signal adapter cable C40 IO extension cable. The P37-05 pinout is defined below.

Table 12-2P37-05 Pin Definitions

P37-05 Turn Board Pinout	37 Coretro nic cable pin	name (of a thing)	Explanation
P19	19	IN9	Universal Input 9
P37	37	IN10	Universal Input 10
P18	18	IN11	General purpose input 11
P36	36	IN12	Universal Input 12
P17	17	IN13	General purpose inputs 13
P35	35	IN14	General purpose inputs 14
P16	16	IN15	Universal Input 15
P34	34	IN16	Universal Input 16
P15	15	IN17	General Purpose Inputs 17
P33	33	IN18	Universal Input 18
P14	14	IN19	Universal Input 19
P32	32	IN20	Universal input 20
P13	13	IN21	Universal Input 21
P31	31	IN22	Universal Input 22
P12	12	IN23	Universal input 23
P30	30	IN24	Universal Input 24
P11	11	OUT11	Universal output 11
P29	29	OUT12	Universal Output 12
P10	10	OUT13	General purpose outputs 13
P28	28	OUT14	General-purpose outputs 14
P9	9	OUT15	Universal Output 15
P27	27	OUT16	Universal Output 16

MPC08D Motion Controller

P8	8	OUT17	Universal Output 17
P26	26	OUT18	Universal Output 18

P7	7	DCV24	Input +24V (7, 25, 2, and 20 can be connected to any one of them.)
P25	25	DCV24	Input +24V
P6	6	OUT19	Universal Output 19
P24	24	OUT20	Universal output 20
P5	5	OUT21	Universal Output 21
P23	23	OUT22	Universal Output 22
P4	4	OUT23	General purpose output 23
P22	22	OUT24	Universal Output 24
P3	3	OUT25	Universal Output 25
P21	21	OUT26	General-purpose outputs 26
P2	2	DCV24	Input +24V
P20	20	DCV24	Input +24V
P1	1	OGND	24V Power Ground