HARDWARE WARRANTY

Mektronix Technology, Inc. ("Mektronix"), Mundelein, IL, warrants to the purchaser that all products manufactured by Mektronix will be free from defects in material and workmanship for a period of one year from the date of shipment from the factory or any authorized distributor of Mektronix. Mektronix's obligation under this warranty shall be strictly and exclusively limited to the repair or replacement, at the factory or a service center of Mektronix, of any such equipment or parts thereof which an authorized representative of Mektronix finds to be defective in material or workmanship under normal use. This warranty does not apply to any equipment which has been tampered with or altered in any way, which has been improperly installed or which has been subject to misuse, neglect or accident. There is no expressed or implied warranty of merchantability or fitness for a particular purpose. Mektronix shall have no liability whatsoever in any event for payment of any incidental or consequential damages, including, without limitation, damages for injury to any person or property.

NOTICE

The information furnished in this manual by Mektronix Technology, Inc. is believed to be accurate and reliable. However, no responsibility is assumed by Mektronix Technology, Inc. for its use; nor any infringements of patents or other rights of third parties which may result from its use. Mektronix Technology, Inc. reserves the right to make engineering refinements to all products and to make changes in this manual without prior written notification. No part of this manual may be reproduced in any form or by any means without prior written permission from Mektronix Technology, Inc.

Printed 10/18/99

LIMITED USE SOFTWARE LICENSE AGREEMENT

The terms and conditions of the Agreement will apply to the Software supplied herewith and derivatives obtained therefrom, including any copy. The term Software includes programs and related documentation supplied herewith. If you have executed a separate Software Agreement covering the Software supplied herewith, such Software Agreement will govern.

1. TITLE AND LICENSE GRANT

   The SOFTWARE is copyrighted and/or contains proprietary information protected by law. All SOFTWARE, and all copies thereof, are and will remain the sole property of Mektronix Technology, Inc. ("Mektronix"). Mektronix hereby grants you a personal, non-transferable and non-exclusive right to use all Software, in whatever form recorded, which is furnished to you under this Agreement. This grant is limited to use with Mektronix MCP-04 boards and may not be repackaged and resold without prior approval from Mektronix. Any other use of this Software shall automatically terminate this license.

   You agree to use your best efforts to see that any user of the Software licensed hereunder complies with the terms and conditions of this License Agreement and refrains from taking any steps, such as reverse assembly or reverse compilation, to derive a source code equivalent of the Software.

2. SOFTWARE USE

   A. You are permitted to make a single archive copy, provided the Software shall not be otherwise reproduced, copied, or disclosed to others in whole or in part.
   B. The Software together with any archived copy thereof, shall be either returned to Mektronix or destroyed when no longer used in accordance with this license Agreement.

3. LIMITED WARRANTY

   A. Mektronix warrants that the Software will be in good working order and will replace, without charge, any Software which is not in good working order if returned to the location where you obtained it within (90) days of delivery to you.
   B. Mektronix does ***not*** warrant that the functions of the Software will meet your requirements or that Software operation will be error-free or uninterrupted.

C. Mektronix has used reasonable efforts to minimize defects or errors in the SOFTWARE. HOWEVER, YOU ASSUME THE RISK OF ANY AND ALL DAMAGE OR LOSS FROM ITS USE, OR INABILITY TO USE THE SOFTWARE.

D. YOU UNDERSTAND THAT, EXCEPT FOR THE 90 DAY LIMITED WARRANTY RECITED ABOVE MEKTRONIX AND ITS AGENTS MAKE NO WARRANTIES, EXPRESSED OR IMPLIED, AND SPECIFICALLY DISCLAIM ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

4. EXCLUSIVE REMEDIES AND LIMITATIONS OF LIABILITIES

A. YOU AGREE THAT YOUR SOLE REMEDY AGAINST MEKTRONIX AND ITS AGENTS FOR LOSS OR DAMAGE CAUSED BY ANY DEFECT OR FAILURE IN THE SOFTWARE REGARDLESS OF THE FORM OF ACTION, WHETHER IN CONTRACT, TORT, INCLUDING NEGLIGENCE, STRICT LIABILITY OR OTHERWISE, SHALL BE THE REPLACEMENT OF MEKTRONIX FURNISHED SOFTWARE, PROVIDED SUCH SOFTWARE IS RETURNED TO MEKTRONIX WITH A COPY OF YOUR SALES RECEIPT.

B. Regardless of any other provisions of this Agreement, neither Mektronix nor its agents shall be liable for any indirect, incidental, or consequential damages (including lost profits) sustained or incurred in connection with the use, operation, or inability to use the SOFTWARE or for damages due to causes beyond the reasonable control of Mektronix or its agents attributable to any service, products or action of any other person.

You acknowledge that you have read this Agreement and understand it, and that by using the Software you agree to be bound by its terms and conditions. You further agree that, except for separate written agreements between Mektronix and you, this Agreement is the complete and exclusive statement of the rights and liabilities of the parties. This agreement supersedes all prior oral agreements, proposals or understandings, and any other communications between us relating to the subject matter of this Agreement.

TABLE OF CONTENTS

Section 1

DESCRIPTION

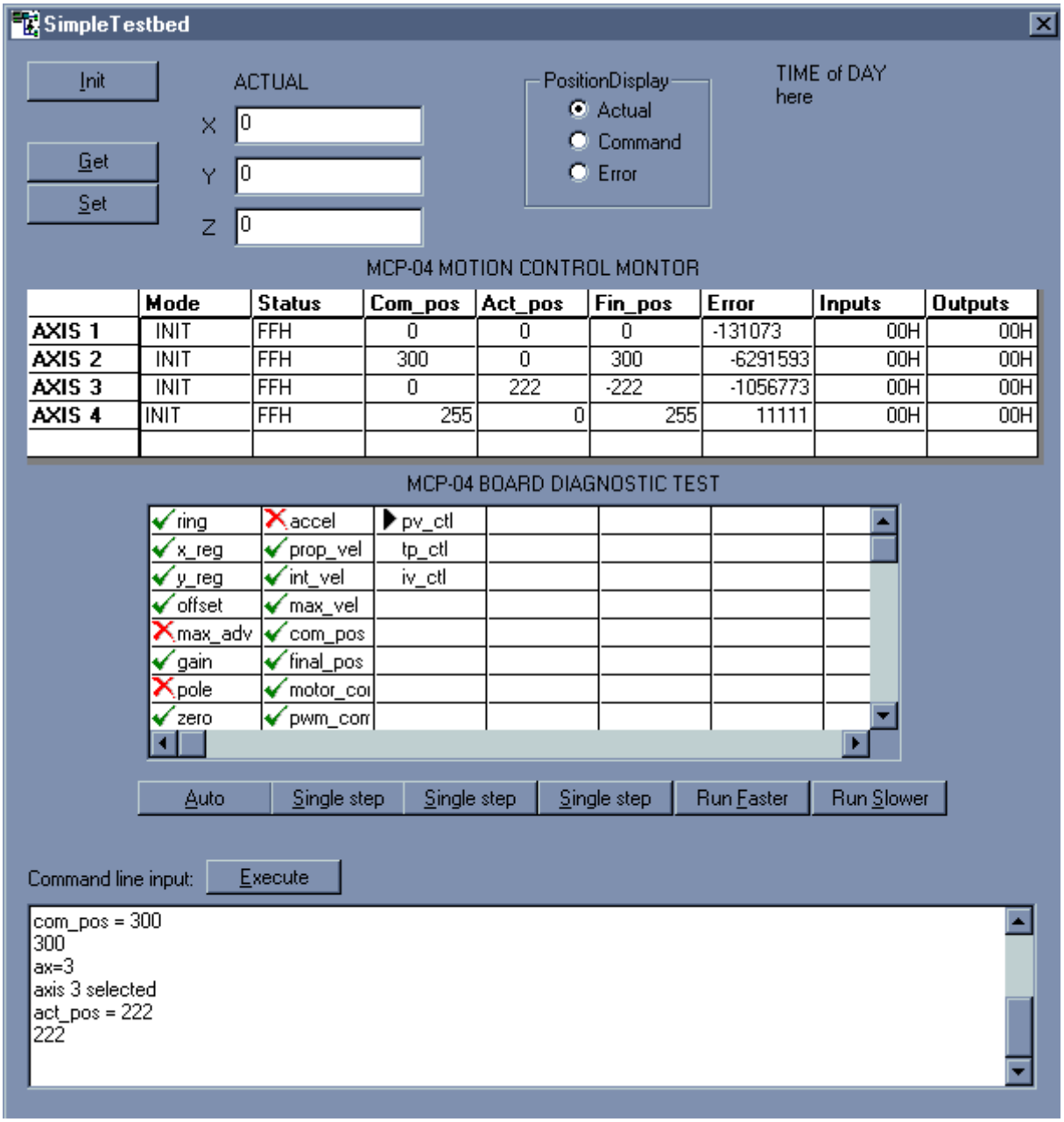
This manual describes the Mektronix MCP-04, 4-Axis Motion Controller and supporting software that runs on a PC compatible computer. The MCP-04 board is a general purpose motion controller that plugs into an ISA expansion slot of the personal computer. Multiple MCP-04 boards may be used in a system as long as each has a unique I/O address. The MCP-04 is a half length card that provides up to four axes of servo control, hosts three I/O ports, four spare incremental encoder inputs and an analog output port. Each axis controller provides all the necessary functions for closed-loop control of brush or brushless DC motors and stepper motors with encoder feedback signals.

The MCP-04 Exerciser program and Programming Interface Library operate on any computer running DOS version 3.3 or later. The MCP-04 Windows Testbed program and the Dynamic Link Library (DLL) is for use with the WIN/95, WIN/98 and WIN/NT operating systems. The Testbed program combines the functionality of the DOS Check and Exercise programs. The **on-screen monitor** displays the installed axis information and I/O port status of MCP-04 boards. The **automated diagnostic test** allows all of the board's registers to be tested and verified. Tests can be run one register at a time (single-step mode) or all of the registers can be run automatically. Tests may be also be run multiple times in a 'burn-in' mode.

The Exerciser allows interactive communication with the MCP-04 using the computer's keyboard. The Exerciser is very helpful for debugging a motion control system during the design phase of an application project. The on-screen monitor displays the installed axis information and I/O port status of MCP-04 boards. The Programming Interface Library routines provide source code interfacing to application programs. The application program may be written in Visual C++, Visual BASIC™, or other programming languages.

™ Visual C++ and Visual BASIC are registered trademarks of Microsoft Corporation.

Description



Closed-loop control provides better response characteristics and higher fault tolerance than open-loop systems. The advantages obtained by using closed-loop control are now affordable with the Mektronix MCP-04 motion controller boards. Special features of the on-board digital commutator can be used to your advantage in lowering the cost of the motor drivers and increasing the performance of brushless motors. The Mektronix MCP-3A6 four axis motor driver provides a direct digital hook-up to the MCP-04 board and is priced at less than $100 per axis.

Mektronix, in its commitment to providing only the finest motion control products, has used the highest quality components and construction techniques, ensuring long-term trouble-free operation. Surface mount components provide a small board layout and improved reliability for embedded PC applications. All connector pins are gold plated for corrosion resistance and special circuitry is used for reliable operation in noisy environments.

1.1. MANUAL ORGANIZATION

The MCP-04 Manual is divided into the following six sections:

- **Description**
- **Installation**
- **System Checkout**
- **MCP-04 Board Operation**
- **System Modeling and Tuning**
- **MCP-04 Software**

Description gives a general overview of how the MCP-04 Manual is organized and describes the features of the system. The DC and AC electrical specifications for the MCP-04 are given at the end of this section.

Installation deals with configuring the MCP-04 boards and installation of hardware and software in the PC. Interfacing the MCP-04 with external equipment is described with application examples.

System Checkout gives instructions for verifying the operation of the hardware and describes a method for establishing closed-loop control. An experimental method for tuning the digital compensator is given using a provided utility program.

MCP-04 Board Operation describes in detail the many capabilities of the MCP-04 motion controller. The built in operating features are described with examples of how to use these features in application programs.

System Modeling and Tuning describes an analytical method for tuning the digital compensator. Modeling of the system components is described so that the filter values can be esti-

mated using graphical methods for optimal control. This section may be skipped if the experimental method of tuning the controller provides acceptable performance.

MCP-04 Software is the operations guide for using the MCP-04 Exerciser program and how to interface with the supplied library routines. Reference manual pages for each command with descriptions and syntax are given in this section. The Programming Interface Library and Dynamic Link Library (DLL) allows application programmers to use high level functions without concern about register programming details.

1.2. OVERVIEW OF MCP-04

The MCP-04 motion controller boards provide an expandable general purpose motion control system that utilizes a PC compatible computer as its host. The half size MCP-04 board plugs into an ISA slot in the PC and controls four servo motors, spindle motors and external devices using the MCP-R16 I/O module. Three interconnects are provided for axis motor drives, incremental encoder inputs, and the I/O module. Sixteen 5 Amp relays and 28 user inputs are available on the small DIN rail mounted module.

The MCP-04 uses a digital compensator to improve system response and increase the stability of motion control systems. The provided control modes are Position Control, Proportional Velocity Control, Trapezoidal Profile Control, and Integral Velocity Control. During each of these modes, the digital compensator parameters may be adjusted as well as its sampling frequency.

Position Control performs point-to-point moves with no velocity profiling. Position Control may be used to perform unique profiling of each axis by sending new position data at the servo update rate. **Proportional Velocity Control** tracks the command velocity continuously until a new command is given. The controller will return to the command velocity if the motor has stalled and will not try to "catch-up". **Trapezoidal Profile Control** performs point to point moves by profiling the velocity trajectory to a trapezoid or triangle. The controller generates the necessary profile to conform to the acceleration, maximum velocity, and final position commands. If the maximum velocity is reached before the halfway point, the profile will be a trapezoidal, otherwise the profile will be triangular. **Integral Velocity Control** performs continuous velocity profiling as specified by the command velocity and

acceleration. After a change in command velocity, the controller will accelerate at the specified acceleration until the new command velocity has been reached.

Applications include PC robotic workstations, X-Y-Z stages, machining operations and automated testing equipment. An optional P-CNC software application package is available for executing NC programs conforming to EIA standard RS-274D.

1.2.1. Motion Controller Features

The MCP-04 controllers provide digital closed-loop control of brush or brushless DC motors and stepper motors, performing all functions required for closed-loop control and eliminating the need for an analog compensator or velocity feedback. All that is needed for a complete servo-system is a PC with a MCP-04 installed, motor drivers, and motors with an incremental shaft encoder. The following lists summarize the main features of the MCP-04 motion control system development package.

Hardware Features:

- High performance closed-loop control of brush or brushless DC and stepper motors
- Adjustable digital stabilizing filter which can achieve sampling rates up to 7.8 kHz
- Four selectable operational control modes that perform position and velocity profiling
- ±10 V analog output and 20 kHz PWM with sign bit command formats
- Commutation signals with phase advance for directly driving brushless motors
- Quadrature incremental encoder format compatible with differential line driver or TTL outputs
- Execution stop and limit for each axis
- Fits into half-size ISA expansion slot
- Expandable to as many axes required in the application system

Additional features of MCP-04

- 28 filtered inputs, 16 high-current outputs
- Additional 16-bit decoder for reading up to four spare external encoder inputs
- Spare analog output port that may be used for controlling external equipment

Software Features:

- Exerciser program provides interactive interface between user and MCP-04

Description

- Programming Interface Library and 32-bit DLL for developing Win32 software applications
- Sample utility program written in C++

- Windows™ 32-bit DLL for Windows 95/98/NT™ and NT device driver
- Optional P-CNC program that runs NC machine code maintaining compatibility with EIA standard RS-274D

1.2.2. Motion Controller Specifications

The MCP-04 specifications are given for completeness and should be useful for system developers.

™ Windows is a registered trademark of Microsoft Corporation.

™ Win95, Win98 and Windows NT are registered trademarks of Microsoft Corporation.

PARAMETER	MIN	TYP	MAX	UNIT	COMMENTS
Supply Inputs:					
+5V Supply current, I_{cc}		400	600	mA	
+12V Supply current, I_{cc}		12	50	mA	
-12V Supply current, I_{cc}		10	50	mA	
Board Inputs:					
Port voltage, V_{IN}	4.5		5.5	V	Filtered, 4.7k pull-ups
Limits voltage, V_{IN}	4.4		24	V	Filtered schmitt trigger
Encoder CMR, V_{IC}			±7	V	RS-422 compatible
Encoder hysteresis, V_{IH}		120		mV	
Board Outputs:					
Analog drive current, I_{OS}		20		mA	Motor commands and
Analog offset , V_{OO}			20	mV	DAC outputs
PWM sink current, I_{OL}			-24	mA	
PWM drive current, I_{OH}			15	mA	
PWM modulation freq		20.0		kHz	
Port Clamp Voltage, V_{DS}			45	V	Outputs A,B,C
Port drive current, I_N		250		mA	Continuous current

Table 1-1. MCP-04 DC Characteristics

T_A= 0°C to 70°C; Units = nSec

		2 MHz Clock	
PARAMETER	**SYMBOL**	**MIN**	**MAX**
Input pulse width; /Stop, /Limit	t_{IP}	600	
Input pulse width; /Index	t_{IX}	1600	
Input pulse width; CHA, CHB	t_{IAB}	1600	
Input pulse width; /Sync	t_{IS}	9000	
Delay CHA to CHB transition	t_{AB}	600	
Input rise/fall time CHA, CHB, Index	t_{IABR}		450
Input rise/fall time; /Stop, /Limit	t_{IR}		50
Delay time, /IOR fall to /IOR fall	t_{RC}	1950	
Delay time, /IOW fall to /IOW fall	t_{WC}	1830	
Output pulse width, PROF, INIT, Pulse, Sign, PHA-PHD	t_{OP}	500	

Table 1-3. MCP-04 AC Characteristics

Section 2

INSTALLATION

This section gives details on installing and configuring the MCP-04 hardware and supporting software. The hardware installation should be checked along the way using the Testbed program's diagnostic capabilities to verify correct operation of external components. Section 3 details procedures for performing a system checkout once the installation is complete.

2.1. SYSTEM REQUIREMENTS

A PC compatible computer serves as the host of the system. The minimum computer configuration required for the MCP-04 system is listed below.

- 1MB of memory
- one ISA expansion slot for each MCP-04 board
- a VGA compatible monitor
- DOS 3.3 or later versions
- Windows 95/98 or NT

Additional recommended options (not required):

- a 80486 µP based computer or higher
- an industrialized computer chassis
- flash disk for reliability

The following associated equipment are required to operate a motion control application.

- Motors with quadrature encoder position feedback (1 per axis)
- Motor drivers (1 per axis)
- Power supply for the motor drivers (1)
- Limit switch interface (1 per axis)
- External I/O interface (MCP-R16)

The motors may be either the brush or brushless type and must be sized for the particular application. The motor drivers must be suitable for driving the selected motors and must accept one of the motor command output formats. Typically this is the ±10V analog range command but the PWM and Commutator outputs may also be applied by knowledgeable users. The ±10 Volt analog command signal will interface directly to the motor driver without external components. Most driver manufacturers provide a differential command (velocity) input. A differential input is required so that the grounds on the MCP-04 are kept isolated.

The MCP-04 board needs a 270ns I/O read/write pulse-width to operate reliably. Figure 2.1 shows the timing specifications for read and write operations. The delay between consecutive I/O read/write cycles must be at least 1.5 µs. Additional timing information is given in Table 1-3. The software supplied by Mektronix Technology, Inc. guarantees these timing requirements are met by inserting the proper delay.

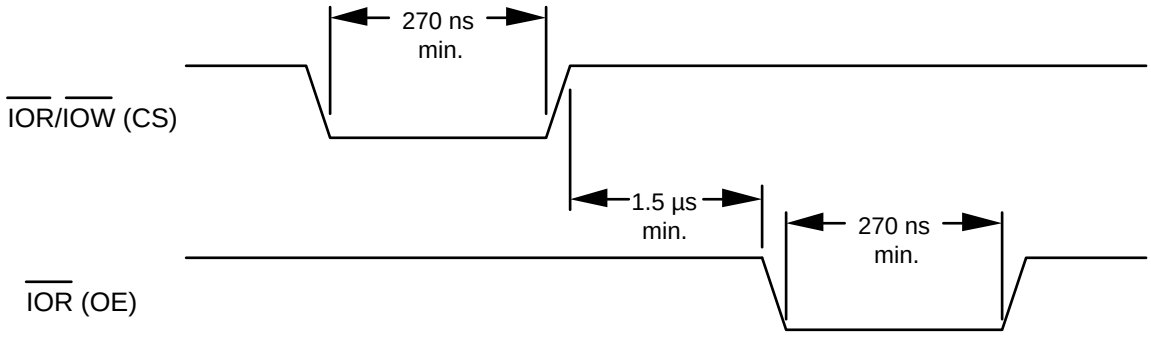


Figure 2.1 I/O Timing Specifications

2.2. SOFTWARE INSTALLATION

2.3.1. The distributed software must be first transferred to a hard disk or the flash disk system of the host computer. The host computer's environment must be set to work with the number and type of MCP-04 boards in your system. These procedures are explained in the following sections.

2.2.1. Distribution Software

The following directories and files are provided on the distribution disk:

UTILS DOS	DOS Utilities Directory
ex.exe	Motion controller Exerciser
mcp.hlp	Data file for on-line help and manual printing
check.exe	Check utility for MCP-04
mca.tst	Check data file
mcb.tst	Check data file
readme	Additional information
DLLWIN32	DLL Directory
SOURCE	Source Code Directory
Mcntrl32.dll	MCP-04 DLL
NT DEVICE DRIVER	NT Device Driver
Installmc.bat	Installs NT driver
Instdrv.exe	Driver installation utility
Mekmcpnt.sys	NT device driver
Samples WIN32	Sample Programs
SOURCE	Source code Directory
Mcntrl32.dll	MCP-04 DLL
Simple Testbed.exe	Sample Program

2.2.2. Installing Exerciser and Library Software

The exerciser and check utilities can be copied from the a: drive to the MCP-04 directory as follows:

```
XCOPY a:\utils\*.* c:\mcp-04\utils
```

Installation

The DLL32 subdirectory includes installation and source code files for Windows 98 and NT Workstation. The MCNTRL32 DLL should be copied to the SYSTEM32 subdirectory of your Windows directory (example, C:\WINDOWS\SYSTEM32).

A batch file (INSTALLMC.bat) is included that copies the MCNTRL32 DLL and the NT device driver to the C:\WINNT40\SYSTEM32 subdirectory, and installs the device driver for the MCP-04 motion control board. Do not run this batch file for Windows 95/98. This batch file is for the NT operating systems only. Type the following from the command line:

INSTALLMC <NT_root_dir>

for example,

INSTALLMC C:\WINNT40

To ensure that this device driver is loaded on every boot, go to the Control Panel and select Devices and change the Start Up option for the MEKMCNT from manual to automatic.

This batch file uses a utility program called INSTDRV, which can be used to remove the device driver from your system using the "remove" command line option. For example,

INSTDRV MEKMCNT remove

2.3. HARDWARE INSTALLATION

Before the MCP-04 motion controller board is installed in the expansion slots of your IBM-PC/AT or compatible computer, it must be configured as described in the following paragraphs.

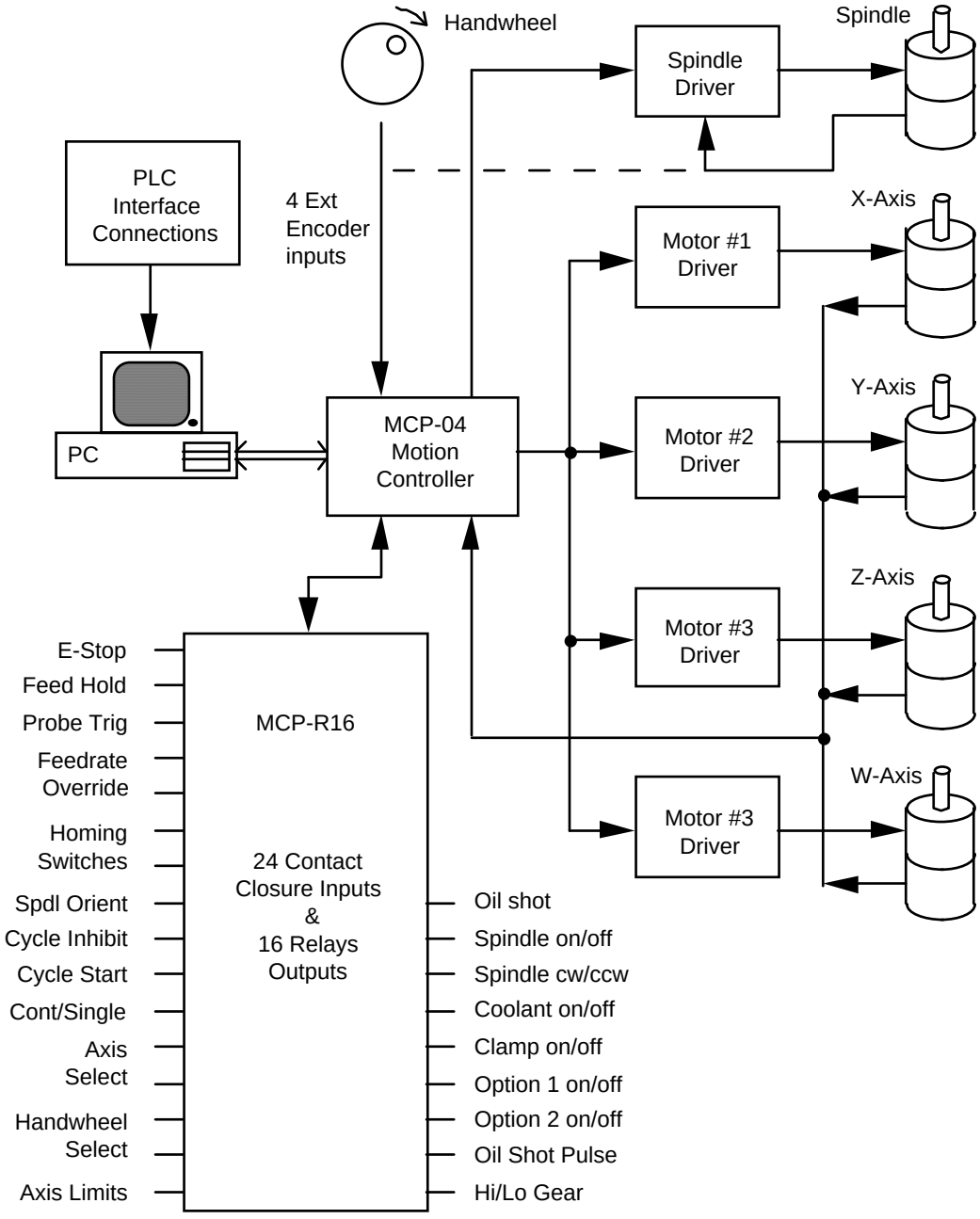


Figure 2-2. MCP-04 System Configuration for P-CNC

2.3.1. Configuring the MCP-04 Board

The MCP-04 Four Axis Motion Control board uses the PC's I/O addresses from 3E0 to 3EB hexadecimal. A standard PC configuration will not conflict with this address space. If addi-

tional I/O boards are installed, verify that they do not conflict with addresses of the MCP-04 board.

The only jumper configuration is for determining the limit switch polarity. When using the MCP-R16 I/O module, the limits must be configured with normally closed switches. In this way, a broken connection will always be detected. The MCP-04 board layout is given in Figure 2-3 showing jumpers JP1, JP2, JP3, and JP4. Each jumper has a '+' sign indicating a connection to +5V. Place jumpers on the '+' side when limit switches are not installed. Jumper to the opposite side (closest to the to edge) for normal configurations using the MCP-R16 and normally closed limit switches.

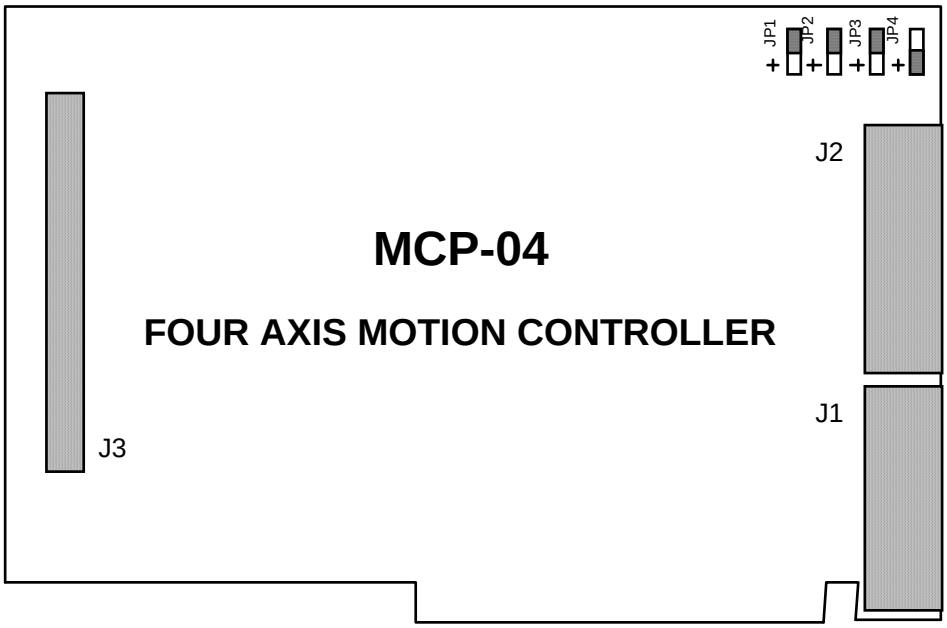


Figure 2-3. MCP-04 Board Layout

Connector J1 interfaces to the motor driver modules, J2 connects to the MCP-R16 I/O module, and J3 provides convenient break-out interconnections for axis encoders and up to four external encoders (used for other functions such as a handwheel and spindle speed measurement).

2.3.2. MCP-04 Connector Pin-outs

The layout of the interconnect wiring should be well thought out for modularity and noise immunity. Provided the interface connections are made in accordance with industry standards, your system will work reliably in industrial environments. The MCP-04 connector pin-outs given in Table 2-1 are laid out for easy cable break-out to multiple connectors.

J1 26-Pin High Density D-Sub Connector

J1-1	Command (Axis 1)	J1-10	Common (Axis 1)	J1-19	PWM Pulse (Axis 1)
J1-2	PWM Sign (Axis 1)	J1-11	Command (Axis 2)	J1-20	Common (Axis 2)
J1-3	PWM Pulse (Axis 2)	J1-12	PWM Sign (Axis 2)	J1-21	Command (Axis 3)
J1-4	Common (Axis 3)	J1-13	PWM Pulse (Axis 3)	J1-22	PWM Sign (Axis 3)
J1-5	Command (Axis 4)	J1-14	Common (Axis 4)	J1-23	PWM Pulse (Axis 4)
J1-6	PWM Sign (Axis 4)	J1-15	DAC Output	J1-24	GND
J1-7	Enable Out (Axis 3)	J1-16	Enable Out (Axis 2)	J1-25	GND
J1-8	+5V Output	J1-17	Enable Out (Axis 4)	J1-26	Enable Out (Axis 1)
J1-9	GND	J1-18	+5V Output		

J2 44-Pin High Density D-Sub Connector

J2-1	Port A Output DB0	J2-16	Port A Output DB1	J2-31	Port A Output DB2
J2-2	Port A Output DB3	J2-17	Port A Output DB4	J2-32	Port A Output DB5
J2-3	Port A Output DB6	J2-18	Port A Output DB7	J2-33	Port B Output DB0
J2-4	Port B Output DB1	J2-19	Port B Output DB2	J2-34	Port B Output DB3
J2-5	Port B Output DB4	J2-20	Port B Output DB5	J2-35	Port B Output DB6
J2-6	Port B Output DB7	J2-21	Axis 1 Limit Input	J2-36	Axis 2 Limit Input
J2-7	Axis 3 Limit Input	J2-22	Axis 4 Limit Input	J2-37	Port A Input DB0
J2-8	Port A Input DB1	J2-23	Port A Input DB2	J2-38	Port A Input DB3
J2-9	Port A Input DB4	J2-24	Port A Input DB5	J2-39	Port A Input DB6
J2-10	Port A Input DB7	J2-25	Port B Input DB0	J2-40	Port B Input DB1
J2-11	Port B Input DB2	J2-26	Port B Input DB3	J2-41	Port B Input DB4
J2-12	Port B Input DB5	J2-27	Port B Input DB6	J2-42	Port B Input DB7
J2-13	Port C Input DB0	J2-28	Port C Input DB1	J2-43	Port C Input DB2
J2-14	Port C Input DB3	J2-29	Port C Input DB4	J2-44	Port C Input DB5
J2-15	Port C Input DB6	J2-30	Port C Input DB7		

J3 50-Pin DIN Ribbon Connector

J3-1	+5V Output	J3-2	Ch A (Axis 1)
J3-3	/Ch A (Axis 1)	J3-4	Ch B (Axis 1)
J3-5	/Ch B (Axis 1)	J3-6	/Ch I (Axis 1)
J3-7	Ch I (Axis 1)	J3-8	2.6 Vref Output
J3-9	GND	J3-10	+5V Output
J3-11	Ch A (Axis 2)	J3-12	/Ch A (Axis 2)
J3-13	Ch B (Axis 2)	J3-14	/Ch B (Axis 2)
J3-15	/Ch I (Axis 2)	J3-16	Ch I (Axis 2)
J3-17	2.6 Vref Output	J3-18	GND
J3-19	+5V Output	J3-20	Ch A (Axis 3)
J3-21	/Ch A (Axis 3)	J3-22	Ch B (Axis 3)
J3-23	/Ch B (Axis 3)	J3-24	/Ch I (Axis 3)
J3-25	Ch I (Axis 3)	J3-26	2.6 Vref Output
J3-27	GND	J3-28	+5V Output
J3-29	Ch A (Axis 4)	J3-30	/Ch A (Axis 4)
J3-31	Ch B (Axis 4)	J3-32	/Ch B (Axis 4)
J3-33	/Ch I (Axis 4)	J3-34	Ch I (Axis 4)
J3-35	2.6 Vref Output	J3-36	GND
J3-37	+5V Output	J3-38	Ch B (Ext. Encoder 1)
J3-39	Ch A (Ext. Encoder 1)	J3-40	GND
J3-41	+5V Output	J3-42	Ch B (Ext. Encoder 2)
J3-43	Ch A (Ext. Encoder 2)	J3-44	GND
J3-45	Ch B (Ext. Encoder 3)	J3-46	Ch A (Ext. Encoder 3)
J3-47	+5V Output	J3-48	Ch B (Ext. Encoder 4)
J3-49	Ch A (Ext. Encoder 4)	J3-50	GND

Table 2-1. P-CNC Connector Pin-Outs, MCP-04 Board

2.3.3. Interface to I/O Module

The connector J2, on the MCP-04 board interconnects to the MCP-R16 I/O module, shown in Figure 2-4 below, using the 44 pin high density D-Sub cable. The cable is wired pin-to-pin from the male plug to the female receptacle. Screw terminal connections are provided for all inputs and outputs on the I/O module. Connect 24 Vdc to the terminals marked +, -, the negative terminal being referenced to the same ground as the PC's power supply. The 24 Vdc supply and the 110 Vac PC power should be switched on and off at the same time. Axis limit inputs are connected to the XYZW and ++++ terminals. The positive terminals are +24 Vdc outputs that connect to normally closed limit switches. The other side of the limit switch connects to the appropriate X, Y, Z, or W terminal. When limit switches are installed, JP1, JP2, JP3 and JP4 on the MCP-04 board must be configured to the ground side (opposite the '+' designator, i.e. closest to edge of PCB).

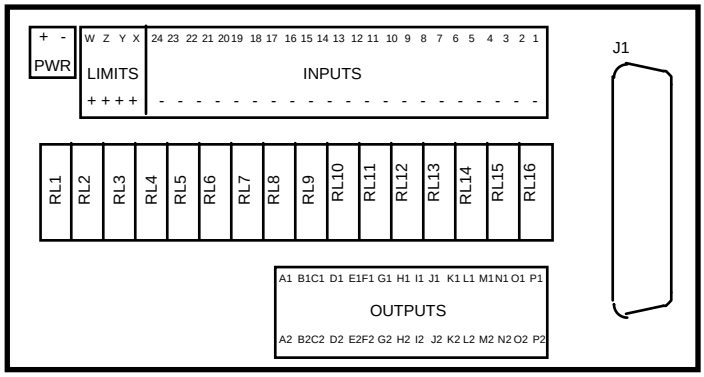


Figure 2-4 MCP-R16 I/O Module Layout

The contact closure inputs are connected to terminals 1 through 24. The other side of a switch is connected to the '-' negative terminal (GND). The assignment of the inputs is listed in Table 2-1. The relay outputs are routed to the terminals labeled A1/A24 to P1/P24. The outputs are rated to 5 Amp for 24 Vdc 110 Vac resistive loads. DC loads are recommended for added reliability.

The two cables coming from J1 and J2 use the color codes described in Table 2-2 below. J1 is a 26-pin connector that goes to the motor drivers and spindle controller. J2 is a 44-pin connector that mates to the MCP-R16 I/O module. The color codes allow easier pin number identification during installation and check-out.

Pin No.	Color Code
1	Black
2	White
3	Red
4	Green
5	Orange
6	Blue
7	White/Black
8	Red/Black
9	Green/Black
10	Orange/Black
11	Blue/Black
12	Black/White
13	Red/White
14	Green/White
15	Blue/White
16	Black/Red
17	White/Red
18	Orange/Red
19	Blue/Red
20	Red/Green
21	Orange/Green
22	Black/White/Red
23	White/Black/Red
24	Red/Black/White
25	Green/Black/White
26	Orange/Black/White
27	Blue/Black/White
28	Black/Red/Green
29	White/Red/Green
30	Red/Black/Green
31	Green/Black/Orange
32	Orange/Black/Green
33	Blue/White/Orange
34	Black/White/Orange
35	White/Red/Orange
36	Orange/White/Blue
37	White/Red/Blue
38	Black/White/Green
39	White/Black/Green
40	Red/White/Green
41	Green/White/Blue
42	Orange/Red/Green
43	Blue/Red/Green
44	Black/White/Blue

Table 2-2. Cable Color Code Chart

2.3.4. Interface to External Devices

Connections to the MCP-R16 I/O module are shown in Figure 2-5 on the following page. All switch inputs provide contact closures to the MCP-R16. **No voltages should be supplied to the inputs**; they are pulled up to 5 volts internally. 24 volts must be supplied to the +/- terminals on the MCP-R16 to operate the relay outputs. The outputs are reed relays that can be used with AC or DC loads with 24 Vdc recommended. The load current must be held to less than 5 amps at 120 VAC or 24 Vdc.

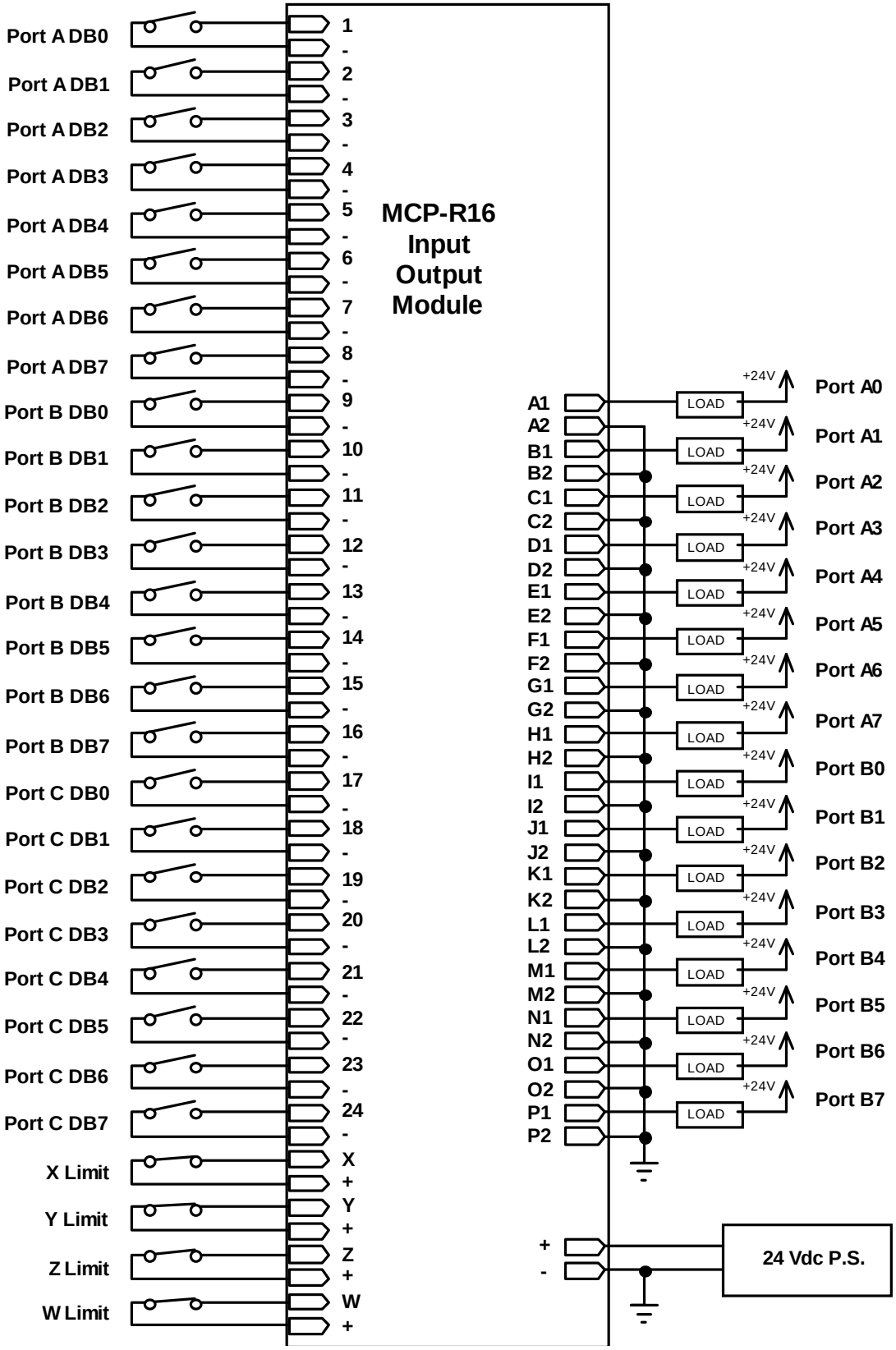


Figure 2-5 MCP-R16 I/O Module Layout

Quadrature Encoder Interface

Quadrature encoder signals are handled by the 75ALS195 line receiver, which provides 120 mV internal hysteresis and a maximum common-mode voltage of ±7 volts. The MCP-04 can be configured for TTL or differential line inputs. It is recommended that you use differential line inputs, since they provide better noise immunity than TTL. Refer to Figure 2-7 and configure the MCP-04 board as follows:

For TTL compatible inputs, connect jumpers in the cable connectors from the Vref line to the complementary inputs. This will provide a good reference voltage to the complementary inputs of the 75ALS195 now being used as a comparator. Route the Channel A and Channel B encoder signals only to the non-complementary inputs at each DB9 connector. The Index inputs are used for a stop signal during homing operation.

Position Encoders

The axis position encoders and the handwheel encoder must output two phase channels in quadrature. They may have either TTL or differential line driver outputs. (The handwheel encoder must use single line TTL encoders.) Differential transmission has the advantage of greater immunity to noise interference than single-ended transmission. For single-ended TTL signals jumper the 2.6 Vref output to the complementary encoder inputs inside the connector shell.

DB-9 Pin No.	Encoder Function
1	+5 Volts Out
2	/Ch A Input
3	/Ch B Input
4	Ch I Input
5	Ground
6	Ch A Input
7	Ch B Input
8	/Ch I Input
9	Vref Output

Table 2-6. DB-9 Axis Encoder Pin-outs

DB-15 Pin No.	Encoder Function
1	+5 Volts Out
2	Ch A, Hdw #1
3	+5 Volts Out
4	Ch A, Hdw #2
5	Ch B, Hdw #3
6	+5 Volts Out
7	Ch A, Spindle
8	Not Used
9	Ch B, Hdw #1
10	Ground
11	Ch B, Hdw #2
12	Ground
13	Ch A, Hdw #3
14	Ch B, Spindle
15	Ground

Table 2-7. DB-15 External Encoder Connections

The shaft encoder signal is typically produced by an optical or Hall-effect encoder in quadrature format. In this format, Channel A and Channel B square waves are offset by 90 degrees, making it possible to determine direction and to increase the resolution by a factor of four. For example, a 1000-line encoder produces 4000 counts per revolution. The required resolution and accuracy of the encoder will depend on the accuracy of your machine. The MCP-04 board can position to ±1 count of the encoder. Check the mechanical configuration and the accuracy specifications of the shaft encoder. Select an encoder resolution that exceeds the desired accuracy of your machine. A good rule of thumb is to select an encoder resolution about 10 times the mechanical accuracy. Another way to select the motor encoder is to set the maximum rapid speed to give 127,000 quadrature encoder counts per second at the selected servo update rate. This will give the best resolution possible for the position feedback.

The handwheel encoder is typically 100 pulses or 400 quadrature counts per revolution. A detent handwheel will also provide precise positioning. The handwheel is connected to one of the spare encoder inputs on the DB-9 connector.

Motors and Drivers

The following motors and associated equipment are required to drive the axes:

- Motors with incremental position encoders
- Motor drivers with power supply
- Power supply for the motor drivers
- Optional spindle motor with velocity feedback amplifier.

The motor drivers must be suitable for driving the selected motors. A current mode (torque) or velocity mode motor driver may be used. It is also possible to use a velocity mode drive by connecting the velocity feedback (tachometer) directly to the motor driver. The driver may use either of the command output formats from the MCP-04 boards; ±10V analog or pulse-width-modulation (PWM). The ±10V command signal will interface directly to the motor driver without external components. Most driver manufacturers provide a differential command input. Connect the differential HI input to the analog ±10V command line at connector J1 and the differential LO input to the common at the same connector. The ±10V command output can source up to 50 mA, allowing the use of cable lengths up to about 25 feet (using 24 AWG stranded wire) provided the input impedance to the motor driver is greater than 10 kΩ and the cable capacitance is less than 300 pF.

The power supply must meet the specification requirements of the motor drivers. An isolation transformer is used to provide a 110 Vac single phase to the power supply rectifier bridge. The resulting bus voltage is around 155 Vdc and must never exceed 180 Vdc when using the Mektronix DR-10A20 and 15A30 motor drivers. Three signal lines are used to connect to the Mektronix motor drivers; the ±10V command and common and the drive enable signal. The drive enable signal is a TTL output that is low (ground) when disabling the drives. This signal keeps the drives in their correct state during power on and off conditions. Otherwise, it is likely that the axes will move under these conditions. Figure 2-8 below shows the typical connections for the drivers supplied by Mektronix.

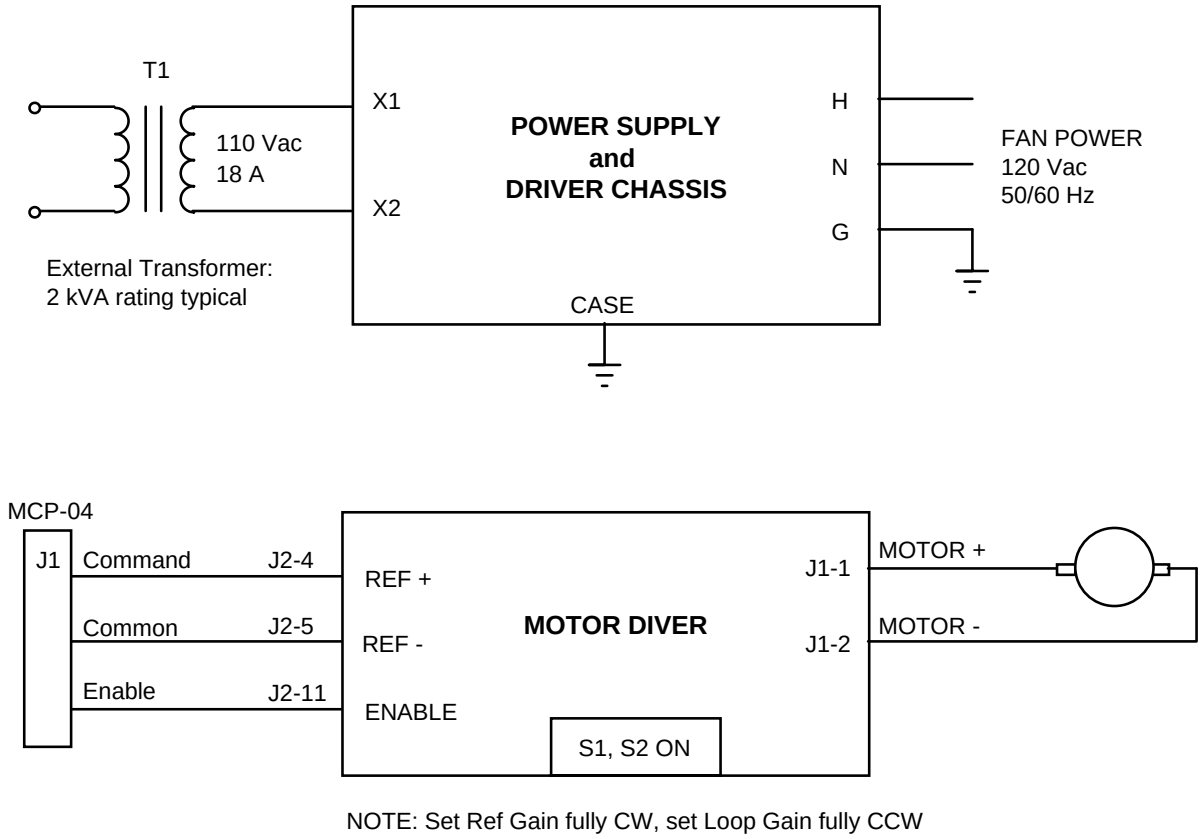


Figure 2-8. Mektronix Motor Driver Configuration

Digital to Analog Converter

An 8-bit digital to analog converter is provided that outputs either the factory preset 0 -10 V range or alternatively a ±10 V range when configured correctly. The analog output can be used with a voltage controlled device such as a spindle drive. In this case, the analog output voltage is reference to a spindle speed.

Axis Limit Inputs

When a Limit is triggered, the controller automatically goes into Initialize mode and outputs zero command voltage to the motor drivers. In order to clear the emergency flag, the limit condition must be removed. The MCP-04 has the capability to override the limit condition.

There is one limit input provided for each axis which should be triggered at both ends of travel for safety using a N.C. switch. The MCP-04 Limit inputs must be asserted for at least 1 μsec in order to trigger.

Cable Preparation

There are three input/output receptacles on the MCP-04 board, designated J1, J2, and J3. The functions of the receptacles are:

J1: provides outputs to motor drivers and spindle drives

J2: connects to the MCP-R16 I/O module

J3: flat ribbon cables expands to four DB9 connectors for each position encoder input and a DB15 for the spare encoder inputs.

The cable end of J1 that connects to the motor drivers must be assembled. The shield of the cable is tied to the connector shell at the DB26 connector. Leave the other end of the shield unconnected. Cables for J2 and J3 are supplied complete. The DB9 encoder connectors are often assembled onto blank PC card brackets for easy access at the rear of the computer chassis.

External Inputs/Outputs

The output ports are 5 volt compatible and can supply up to 250 mA of current and should be able to drive long cable lengths. The outputs may be interfaced directly to the MCP-R16 I/O module which contains 5 Amp dry contact relays. The relays are suitable for both AC and DC currents. The input ports are all contact closure to ground and feature filtered, schmitt trigger inputs. There are 24 contact closure inputs and 16 high-current outputs available from connector J2 on the MCP-04 board. Four additional inputs are reserved from Limit inputs. The I/O module uses 24 Vdc relays and require that voltage to be supplied to the power terminals on the MCP-R16 I/O module.

Homing Inputs

Homing inputs for each axis can be assigned to any port that is configured as an input. The homing switches provide the control direction while performing a homing sequence. The homing sequence consists of the following two stages:

1) the axis moves in the direction specified by the control bit (0 = positive and 1 = negative) with given specified velocity and acceleration. The first stage ends as soon as the control bit changes state.
2) the axis then moves in the opposite direction at one fifth the previous speed and stops when the control bit changes state again.
3) The axis then looks for the encoder index pulse and stops immediately. At the end of stage three the actual position is set to zero.

The homing operation is performed using the Homing command in the Exerciser or interface library. If power is lost and then reapplied, the current position will be defined as zero. It is recommended that the zero (homing position) be near the center of the work area. When mounting the homing switch in the center area it is important to remember to use the switch to inform the controller of the correct direction for homing. This may be accomplished by using a hall-effect sensor and a metal ridge that spans to the homing location. Figure 2-9 illustrates this concept.

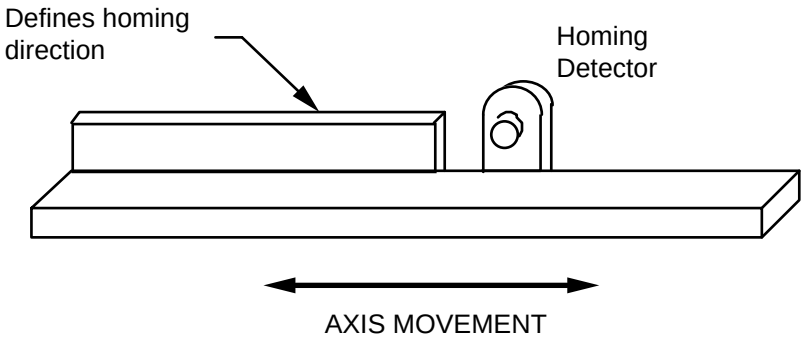


Figure 2-9. Homing Implementation

Limit Inputs

The Limit disables the output motor command and initializes the axis. To trigger the Limit you must open the contact closure from ground for at least 600 ns. The Limit should be used at both ends of travel in the system for safety. There is only one Limit input for each axis; the user must connect the left and right normally closed limit switches in series to logically OR the two Limits.

The controller continues to keep tract of the actual position during a Limit condition. This fact may be used to distinguish which trigger occurred. To get out of a limit condition, the control provides a "Limit Override" that must be used to allow jogging the effected axis off of a Limit.

Section 3

SYSTEM CHECKOUT

This section gives guidelines for establishing a control loop and manually exercising the MCP-04 boards to verify correct operation. The user will become familiar with the command structure and features of the MCP-04 Motion Controllers by following these instructions. Once the basic operation is well understood, the user can proceed with the advanced concepts discussed in the remaining sections of this manual.

3.1. VERIFY COMMUNICATION

3.1.1 Check Board Utility

The Check program is used to verify that the MCP-04 board is working properly and that the communication to and from the PC is reliable. IMPORTANT: Disable the motor drives before operating this utility, otherwise, the axis motors will move during the test. Follow the following procedure to verify that the MCP-04 board is working correctly. Four axis boards will need the MCINIT environment set to "4:3E0;" either in the AUTOEXE.BAT or from the command line. Type in the following command:

set MCINIT=4:3E0;

Change to the directory where 'check.exe' is installed. Run 'check.exe' from this directory. The program first comes up with a screen that detects the assignment and the number of axes detected. The number of axes defaults to four but can be configured for more using a DOS variable called MCINIT. Type 'y' to accept and continue with the program.

From the main menu of Check, type '1' to enter the Diagnostic tests. The first axis to test is axis one. Type 'a' to automatically test all the board registers for the axis. All the register names should change from lower case to upper case. If any register blinks then an error was detected. The "status" register will give an error if a limit or stop condition is detected. Type

'n' to go on to the next axis. Repeat the same procedure for all the installed axes and return to the main menu. Type '0' to return to the DOS prompt.

If you feel like performing a more stringent test, select '2' to run the Burn-in test. The test will run the same but 100 times longer. You should notice when the test reaches the "status" the count will decrement down from 100 at a periodic rate. If the count does not decrement to zero an error was detected. Return to the main menu and type '0' to get back to the DOS prompt.

3.1.2 Command Exerciser Utility

After installation of the hardware and software is complete, the system communication using the Exerciser should be established. A complete discussion of using the Exerciser is given in Section 6 of this manual. The user is encouraged to reference Section 6 as required to obtain more detailed operational instructions than what is given in this section.

Turn on the computer that has an MCP-04 board installed and change to the directory which contains the Exerciser and MCINIT.BAT file. The user input is given in bold lettering.

```
C:> cd\mcp\utils
C:\MCP\utils>
```

Next, before running the Exerciser program it is necessary to run MCINIT unless only one MCP-04 is being used and its starting address is set to the factory preset address of 3E0H. Type "ex" after establishing the proper DOS environment:

```
C:\MCP\utils>
MCP Series Board Exerciser (Version 2.01)
Copyright (c) 1987-1998 by Mektronix Technology, Inc.
.
```

At this point the Exerciser is waiting to receive commands issued from the keyboard. To establish communication with the board an easy test is to read and write from a register of an axis. The 'gain' register will be used for this purpose in the example given below.

```
. ?gain                                    check gain axis 1
```

16.00

. **gain = 12** set axis 1 gain to 12

12.00

. **?gain** read new value of gain

12.00

. **axis = 2** select axis 2 communication

axis 2 selected

. **?gain** check gain axis 2

16.00

. **gain = 12** set axis 2 gain to 12

12.00

. **?gain** read new value of gain

12.00

A series of commands that randomly check reading and writing to the controller can be stored in a text file and executed with the Exerciser program by typing

. **execute** filename

It is also possible for the user to automate this procedure by writing a simple program that issues a new value to a register and then reads it back to verify the register was set to the correct value. Refer to Section 6 on how to use the library commands with a high level programming language such as 'C'. If communication can not be established, check that MCINIT has been called and properly set to the correct starting address. From the DOS prompt type ‘SET’ to see the current environment settings.

3.2. CLOSED-LOOP CONTROL

This section describes how to close the control loop around a servo motor and how to adjust the digital compensator using an experimental method. A general procedure for establishing the proper feedback polarity is also given.

3.2.1. Establishing the Control Loop

The control loop may be closed after assembling all the necessary components in the control system. Make cables as per instructions given in Section 2 to interface the motor drivers and the incremental quadrature encoders to the MCP-04 boards. Connect these cables to the appropriate places when power is off. Disconnect the motor shafts from any mechanical transmission so that they are free to turn continuously. The control loop is closed when the position feedback is present from the quadrature encoders. Follow the procedure given below to verify that the control loop is functional.

Step 1. During installation make sure power is off to all equipment and then turn the power on to only the personal computer. Turn power on the motor drivers and adjust out any amplifier input offset so that the motors remain stationary.

Step 2. Run the Exerciser and turn the monitor on by typing:

. ex
. monitor on

The monitor screen will display information on all axes configured in the DOS environment variable. At this point all axes should be in "INIT" mode and the status should display E0H.

Step 3. Turn the motors/encoders with your hand and verify that all the position encoders are working by watching the "act_pos" change on the display.

Step 4. To establish the correct polarity for closed-loop control, output a small positive voltage command to the motor drivers by issuing either

. **motor_com = 83**
83 (or)
. **pwm_com = 3**
3

depending on which command output format is being used. If the motor does not turn you should increase the output command slightly. The motor should turn in

the positive direction. Verify this by observing the "act_pos" display increasing in value. If the actual position is decreasing in value, then the encoder CHA and CHB inputs need to be reversed.

Step 5. The final step is to begin servoing the motor by entering position control mode. There are four parameters that can be varied for each digital compensator; the Gain, Pole, Zero, and Sample Frequency. Next, before closing the control loop, set the filter Gain to a minimal value and the Zero to a large value for each axis.

. **gain = 2**
2.0
. **zero = .95**
0.94921875

Now the position loop may be closed typing the following from the Exerciser.

. **cm**
control mode entered: (axis 1)

CAUTION

There is the possibility that the position feedback is positive which could cause the motor to run away when closing the control loop. Make sure that if this condition does exist that it will not cause any damage.

At this point there are the two possibilities:

I. The motor runs away after any small disturbance and continues to turn at full speed. This indicates that the position feedback is positive and should be reversed. This condition is easily corrected by switching the Phase A and the Phase B encoder outputs.

II. The motor remains in the same position even after a disturbance. If you turn the motor shaft and let go, the motor will return to the same position. This indicates that the control loop has negative position feedback, as required.

At this point the control loop gain is very low and the motor will have little restoring torque. Gradually increase the gain by repeatedly commanding the Gain register to a higher value. Send the Exerciser command:

. **gain = <N>**

where N is the decimal value of gain.

If the motor begins to vibrate, reduce the gain slightly. The default value for gain is 16 decimal when the controller is first powered or issued a software reset.

3.2.2. Adjusting the Digital Compensator

An experimental method of tuning the digital compensator provides an alternative to the analytical approach given in Section 5 and may provide a benchmark for comparison. Special circuitry on the MCP-04 is provided to assist in tuning the digital compensation filter. The method is based on repeatedly stepping the motor and load in a back and forth motion using the Exerciser utility 'tune_filter'. Reference the Exerciser reference page in Section 6.4 for instructions on using this command. The step distance should be small so the command output signal remains in the linear range.

While the motor is stepping back and forth, the filter parameters Gain, Pole, and Zero can be changed for the best response. The pole term (POLE) does not significantly effect the system response unless the system has a high frequency resonance. Initially leave POLE to the default value. Start with a low gain, (GAIN < 10), and a low frequency zero, (ZERO = .95). Increase the gain until the motor begins to oscillate and then reduce GAIN to a safe margin. Observe the step response on the oscilloscope. Adjust the value of ZERO, and GAIN if necessary, by trail and error, until the desired step response has been achieved. The POLE term may now be adjusted to see if a larger value has any beneficial effects. Finally, the SAMPLE_FREQ may be adjusted to change the frequency response of the control system. The default sample frequency (1923 Hz) should provide more than enough bandwidth for most motion control systems. The sample frequency should be kept to within the 7.8 kHz rate because this is the highest frequency that can be used during profile modes.

A typical criteria for the step response is to achieve about a 30 percent overshoot of the step commanded position. The system accuracy will be effected by the system gain or stiffness.

For high accuracy systems it is desirable to have high loop gains so that any friction in the system will be overcome by the controller. The position error display of the Exerciser monitor will give an indication of the system tracking and positioning deadband.

3.3. EXERCISING THE MCP-04 BOARDS

Simple operating procedures can be issued from the Exerciser that demonstrate the operating modes of the MCP-04 board. The user is referred to Section 6 for a more detailed description on using the Exerciser. Start the Exerciser and follow the procedures given below.

3.3.1. Control Modes

The four control modes; Position Control, Proportional Velocity Control, Integral Velocity Control, and Trapezoidal Profile Control, provide the operating features of the MCP-04 Motion Controller boards. The following tutorial instructions show how these four control modes operate. The instructions are not repeated for each axis since the procedure is identical.

The default mode upon applying power to the MCP-04 board is Initialize (Init) mode. While in this mode the user should program all the necessary registers before executing a control mode. Motor command outputs can be issued while in the Initialize mode to check for correct open-loop operation. The following procedure tests the analog and PWM output pins and should be performed while the motor drivers are turned off, or the motors are free to turn continuously.

. **init**	put in Initialize mode
initialize mode: (axis 1)	
. **motor_com = ff**	outputs 10 volts to the command pin
FFH	
. **motor_com = 0**	outputs -10 volts to the command pin
00H	
. **motor_com = 80**	outputs 0 volts to the command pin
80H	
. **pwm_com = 50**	50% duty cycle to PWM pulse pin and
50	a low level to PWM sign pin

. **pwm_com = -50**	50% duty cycle to PWM pulse pin and
-50	a high level to PWM sign pin

The user should set all compensation parameters prior to entering a control mode. All control modes use some part of the digital compensator which must be set to the values determined in Section 3.2.

. **gain = N**
. **zero = N**
. **pole = N**
. **sample_freq = N**

Where N is user selectable. Once the digital compensator parameters are set, Position Control mode may be entered. Put each axis in Initialize mode and turn power on to the motor drivers.

Position Control Mode

Position Control mode servos on the current commanded position so that the controller tries to maintain zero error between the 'com_pos' and 'act_pos' registers. When entering from Initialize mode the command position is set to the actual position. The following gives an example of how to command a step position change to the controller.

. **init**	enter Initialize mode
initialize mode: (axis 1)	
. **act_pos = 0**	specify current position as zero
0	
. **enter_ctl_mode**	enter Position Control
control mode entered: (axis 1)	
. **?act_pos**	find current position
0	
. **com_pos=200**	move 200 counts as fast as possible
200	
. **?act_pos**	check new position
200	

All the other control modes are executed from Position Control mode. The provided software does not allow any of these modes to be executed directly from Initialize mode.

Proportional Velocity Control Mode

Proportional Velocity Control mode uses only the Gain factor of the digital compensation filter. The velocity is programmed using the Exerciser command 'prop_vel'. To execute Proportional Velocity Control mode try the following:

. **enter_ctl_mode**	enter control mode
control mode entered: (axis 1)	
. **prop_vel = 10**	set velocity command
10.0000	
. **go_pv_ctl**	start Proportional Velocity Control
proportional velocity control: (axis 1)	
. **?act_vel**	get current velocity
10	
. **prop_vel=0**	stop motor
0.0000	

Integral Velocity Control Mode

Integral Velocity Control mode uses all the parameters of the digital compensator filter when computing the motor command. The command acceleration and velocity must be set in this mode so that the motor profiles in velocity and accelerates to the new commanded velocity. The example given below demonstrates the basic features of this control mode.

. **enter_ctl_mode**	enter control mode
control mode entered: (axis 1)	
. **accel = .01**	set a small acceleration
.01171875	
. **int_vel = -10**	set command velocity in (-) direction
-10	
. **go_iv_ctl**	start Integral Velocity Control
integral velocity control: (axis 1)	

. **int_vel = 10**	decelerate to (+) velocity direction
10	
. **int_vel = 0**	decelerate to stop
0	

Trapezoidal Profile Control Mode

Trapezoidal Profile Control is used to profile to a final position at a specified velocity. The motion profile uses the acceleration register to ramp up to the commanded maximum velocity and to ramp down to the final position. Use the example below to initiate a velocity profile that looks like a trapezoid.

. **init**	enter Initialize mode
initialize mode: (axis 1)	
. **act_pos = 0**	specify current position as zero
0	
. **enter_ctl_mode**	positions control mode entered
control mode entered: (axis 1)	
. **accel = .02**	specify a small acceleration
0.01953125	
. **max_vel = 10**	command maximum velocity
10	
. **final_pos = 200000**	final position set to +200,000 counts
200000	
. **go_tp_ctl**	execute Trapezoidal Profile Control
trapezoidal profile control: (axis 1)	
. **?status**	check status (profile flag set)
D0H	repeat until motion complete
. **?status**	trajectory complete (position control)
C0H	
. **?act_pos**	get actual position
200000	

3.3.2. External Inputs/Outputs Ports

There are three input ports A, B, and C and two output ports A and B on the MCP-04 board. Ports can be read or written to as shown below.

. **port_c = 5**	output 5H to Port C
05H	
. **?port b**	read in 8-bit value from Port B
00H	
. **?port a**	read in 8-bit value from Port A
00H	

3.3.3. External Encoder

It is also possible to read in an additional quadrature encoder besides the Axes 1 through 4 encoders on the MCP-04 board using the Ext A and Ext B inputs at connector J3.

. **?e1**	read external encoder number 1
200445	
. **e1 = 0**	clear external encoder number 1 count
0	
. **?e2**	read external encoder number 2
0	

3.3.4. Digital to Analog Converter

The External DAC outputs a voltage to connector J1 on the MCP-04 board. The DAC output is normally configured for ±10 volt operation where 80H corresponds to 0 volts. To send a 8-bit hexadecimal number to the DAC use the following commands.

. **ext_dac = 80**	set external DAC to 0 volts
80H	
. **ext_dac = FF**	set external DAC to +10 volts
FFH	
. **ext_dac = 0**	set external DAC to -10 volts

00H

3.3.5. Limit

The Limit inputs can be tested by connecting external switches as previously described. The following procedure can be used to verify proper operation.

. **enter_ctl_mode**	enter position control mode
. control mode entered: (axis 1)	
. **echo long**	set echo format to *long*
. **?status**	get current status

status = E0H: no limit
no stop
not in initialize mode
not in trapezoidal profile
commutator count quadrature
commutator 3-phase
pwm sign reversal off

. **accel = .01**	set acceleration
command acceleration (axis 1): 0.01171875	
. **int_vel = 20**	set integral velocity
command integral velocity (axis 1): 20	
. **go_iv_ctl**	begin velocity profiling
integral velocity control: (axis 1)	

The stop is triggered by the index pulse of the position encoders. For this to happen, the signals must be enabled using Port C configuration output port. Manually set the Port C output so that bit 2 is low. The motor should stop at the next index pulse.

. **?status**

status = E0H: no limit
stop triggered
not in initialize mode
not in trapezoidal profile
commutator count quadrature
commutator 3-phase
pwm sign reversal off

. **clr_emergency** clear emergency flags
emergency flags of axis 1 cleared
. **int_vel = -20** set integral velocity
command integral velocity (axis 1): -20
. **go_iv_ctl** begin velocity profiling
integral velocity control: (axis 1)

While the axis is profiling in Integral Velocity Control mode, trigger the corresponding Limit input. The axis should immediately be disabled while the axis enters Initialize mode. Release the Limit input and continue with the procedure.

. **?status** get current status

status = E0H: limit triggered
no stop
in initialize mode
not in trapezoidal profile
commutator count quadrature
commutator 3-phase
pwm sign reversal off

. **clr_emergency** clear emergency flags
emergency flags of axis 1 cleared
. **enter_ctl_mode** enter position control mode
. control mode entered: (axis 1)
. **echo short** set echo format to *short*

Section 4

BOARD OPERATION

This section describes how the MCP-04 motion control board operates. The details of the operation is transparent to users of the MCP-04 Language Interface Libraries and Exerciser. However, some users may wish to add special functions and will need to understand how to interface to the MCP-04 motion controller boards.

4.1. ADDRESS COMMUNICATIONS

The communication to and from the MCP-04 uses the 8-bit parallel I/O bus provided on the personal computer. The I/O base address is programmable so that multiple boards are allowed in the same system. Once the programming environment has been set for the the number and types of motion controller boards installed, the Exerciser and Programming Interface Library provide a transparent interface for communication to the MCP-04 motion controller boards.

4.1.1. Address Decoding

The PC communicates with the MCP-04 using I/O addressable registers. These registers contain command and configuration information necessary to properly operate the board. The board uses I/O address space from 3E0 to 3EB.

A3	A2	A1	A0	Functional Description
0	0	0	0	Select Axis 1
0	0	0	1	Output Enable Axis 1
0	0	1	0	Select Axis 2
0	0	1	1	Output Enable Axis 2
0	1	0	0	Select Axis 3
0	1	0	1	Output Enable Axis 3
0	1	1	0	Select Axis 4
0	1	1	1	Output Enable Axis 4
1	0	0	0	Port A Rd/Wr
1	0	0	1	Port B Rd/Wr
1	0	1	0	Port C Rd/Wr
1	0	1	1	Port D Rd/Wr

Table 4-1. MCP-04 Board Select Addressing

When sending a command to an axis you write to address Select Axis. However, to read from a motor controller you must issue two I/O read operations. First to the Select Axis and then to the Output Enable Axis to receive valid data. The AC timing specifications given in Table 1-3 must be met. Port C outputs are used for internal configuration as shown below in Table 4-2.

Port C Bit	Functional Description
0	Sync
1	Limit Override
2	Home Enable (Index to Stop)
3	External Encoder Select
4	External Encoder Clear
5	External Encoder DB0
6	External Encoder DB1
7	Not Used

Table 4-2. Port C Internal Decoding

4.1.2. Register Programming

A15 through A10 make up the Register Base Address for each motion controller register. A9 through A0 make up the Starting Board Address and is mapped into the PC's I/O address space. For example, say you want to write 00H to the Program Mode register. The MCP-04 factory preset starting address is 3E0H and the Register Base Address is 1400H for the Program Mode register. In order to find the correct address for writing to Axis 2, you would add 1400H + 3E0H + 02H = 17E2H. Writing 00H to the Program Mode register performs a software reset in the motor controller. Table 4-5 shows all the registers and their control functions.

Register Base Addr	Function	Mode Used	Data Type	User Access
0 0 0 0 H	Flag Register	All	---------	rd/wr
1 4 0 0 H	Program Mode Register	All	scalar	wr
1 C 0 0 H	Status Register	All	---------	rd/wr [1]
2 0 0 0 H	8-bit Motor Command Port	All	2's compl.+0H	rd/wr
2 4 0 0 H	PWM Motor Command Port	All	2's compl.	rd/wr
3 0 0 0 H	Command Position (MSB)	Position Control	2's compl.	rd/wr [2]
3 4 0 0 H	Command Position	Position Control	2's compl.	rd/wr [2]
3 8 0 0 H	Command Position (LSB)	Position Control	2's compl.	rd/wr [2]
3 C 0 0 H	Sample Timer	All	scalar	wr
4 8 0 0 H	Actual Position (MSB)	All	2's compl.	rd [3]
4 C 0 0 H	Actual Position	All	2's compl.	rd [3]/wr[4]
5 0 0 0 H	Actual Position (LSB)	All	2's compl.	rd [3]
5 4 0 0 H	Actual Position [MSB]	All	2's compl.	wr [2]
5 8 0 0 H	Actual Position	All	2's compl.	wr [2]
5 C 0 0 H	Actual Position [LSB]	All	2's compl.	wr [2]
6 0 0 0 H	Commutator Ring	All	scalar [5]	rd/wr [6]
6 4 0 0 H	Commutator Velocity Timer	All	scalar	wr
6 8 0 0 H	X	All	scalar [5]	rd/wr
6 C 0 0 H	Y Phase Overlap	All	scalar [5]	rd/wr
7 0 0 0 H	Offset	All	2's compl.	rd/wr [6]
7 C 0 0 H	Maximum Phase Advance	All	scalar [5]	rd/wr [6]
8 0 0 0 H	Filter Zero, A	All except Prop Vel	scalar	rd/wr
8 4 0 0 H	Filter Pole, B	All except Prop Vel	scalar	rd/wr
8 8 0 0 H	Gain, K	All	scalar	rd/wr
8 C 0 0 H	Command Velocity (LSB)	Proportional Velocity	2's compl.	rd/wr
9 0 0 0 H	Command Velocity (MSB)	Proportional Velocity	2's compl.	rd/wr
9 8 0 0 H	Acceleration (LSB)	Int Vel and Trap Profile	scalar	rd/wr
9 C 0 0 H	Acceleration (MSB)	Int Vel and Trap Profile	scalar [5]	rd/wr
A 0 0 0 H	Maximum Velocity	Trapezoidal Profile	scalar [5]	rd/wr
A 4 0 0 H	Final Position (LSB)	Trapezoidal Profile	2's compl.	rd/wr
A 8 0 0 H	Final Position	Trapezoidal Profile	2's compl.	rd/wr
A C 0 0 H	Final Position (MSB)	Trapezoidal Profile	2's compl.	rd/wr
D 0 0 0 H	Actual Velocity (LSB)	Proportional Velocity	2's compl.	rd
D 4 0 0 H	Actual Velocity (MSB)	Proportional Velocity	2's compl.	rd
F 0 0 0 H	Command Velocity	Integral Velocity	2's compl.	rd/wr

NOTES:

1. Upper 4 bits are read only.
2. Writing to (LSB) latches all 24 bits.
3. Reading (LSB) latches data into all 24 bits.
4. Writing to middle 8-bits clears Actual Position Counter to zero.
5. The scalar limited to positive numbers (00H to 7FH).
6. The commutator registers have further limits which are discussed in the Commutator section

Table 4-5. Register Reference Table.

4.2. MOTION CONTROL SETTINGS

The operation of each axis is based on the Hewlett Packard HCTL-1100 and is restricted to its capabilities. This section describes its operational characteristics and controllable parameters.

4.2.1. Digital Compensator

All control modes use some part of the programmable digital filter D(z) to compensate for closed-loop system stability. In terms of the Exerciser and Programming Interface Library, the compensation D(z) has the form:

$$D(z) = GAIN \frac{(z - ZERO)}{(z + POLE)} \qquad (4\text{-}1a)$$

In terms of the HCTL-1100's registers the compensation D(z) has the form:

$$D(z) = \frac{K}{4} \frac{(z - \frac{A}{256})}{(z + \frac{B}{256})} \qquad (4\text{-}1b)$$

where:

z = the digital domain operator
K = Gain
A = Zero
B = Pole

The compensator is a first order lead filter which in combination with the sample timer T (R3C00H) affects the dynamic response and stability of the control system. The sample timer T determines the rate at which the control algorithm is executed. All the filter parameters, K, A, B, and T, can be changed at any time.

The contents of the Sample Timer register sets the sampling period of the HCTL-1100. The sampling period can be calculated using the following equation:

$$T = 16\ (R3C00H + 1)\ (1/\text{board clock frequency}) \qquad (4\text{-}2)$$

The sample timer has a limit on the minimum allowable sample time depending on the control mode being executed. The limits are given below:

	Minimum Value R3C00H
Position Control	7
Proportional Velocity Control	7
Trapezoidal Profile Control	15
Integral Velocity Control	15

The maximum value of R3C00H is FFH (255D). So the sample time can be varied from 64 μsec to 2048 μsec using the default 2 Mhz board clock frequency. The fastest sampling rate possible for Trapezoidal Profile Control is 7.8 kHz. The default value (64D) in the sample timer register should provide enough bandwidth for most applications.

The sample timer effects the compensation by adding phase lag to the control system. So, the complete compensation can be thought of as a lead-lag compensator. More phase lag is added by increasing the value in the sample timer register. A fast sampling rate will provide less quantization errors and faster response but may also cause high frequency oscillations. It is typically recommended that the sampling rate be at least ten times the mechanical bandwidth of the system. When synchronizing multiple axes, the sampling rate should be kept below 2 kHz in order to have enough processing time between samples.

4.2.2. Flag and Program Mode Registers

Flag Register

The Flag register contains flags F0 through F5. Each flag is set or cleared by writing to R0000H. The upper four bits are ignored while the lower three bits specify the flag address and the fourth bit specifies whether to set (bit=1) or clear (bit=0) the specified flag.

Bit Number:	7-4	3	2	1	0
Function:	x	set/clear	D2	D1	D0

F0 - Trapezoidal Profile Flag: set by the user to execute trapezoidal profile control. The flag is reset by the controller when the final position move is completed. If the motor can not keep up with the position commands the motor may still be moving after the F0 flag has reset. The status of F0 can be visually monitored by watching the green Profile LED. It may also be monitored at J1 and in bit 4 of the Status register.

F1 - Initialize Flag: set or cleared to indicate controller is in Initialize mode. The status of F1 can be visually monitored by watching the yellow Init LED. It may also be monitored at J1 and in bit 5 of the Status register. The user should never attempt to set or clear F1.

F2 - Unipolar Flag: set/cleared by the user to specify bipolar (clear) or unipolar (set) mode for the Motor Command Port. When commutating brushless motors, the direction of the motor rotation is governed by the order of firing of the motor phases which is under commutator phase control. In this case, the Unipolar mode can also be used to restrict the DAC output from 0 to 10 volts only.

F3 - Proportional Velocity Control Flag: set by the user to specify proportional velocity control.

F4 - Hold Commutator Flag: set/cleared by the user or automatically by the align mode. When set, this flag inhibits the internal commutator counters to allow open-loop stepping of a motor by using the commutator. Note that this feature is not intended to be used for controlling an open-loop stepper motor.

F5 - Integral Velocity Control: set by user to specify integral velocity control.

Reading the Flag register returns the status of the flags. Bit 0 to 5 contain the respective data of the flag status. For example, if Bit 0 is set (logic 1), then flag F0 is set. If bits 0 and 5 are set, then both flags F0 and F5 are set.

Program Mode Register

The Program Mode register, which is a write only register, executes the pre-programmed functions of the HCTL-1100. The Program Mode register is used along with the control

flags F0, F3, and F5 in the Flag register to change control modes. The user can write any of the following four commands to the Program Mode register.

00H - Software Reset
01H - Initialize Mode
02H - Align Mode
03H - Control Mode: flags F0, F3, and F5 in the Flag register specify which control mode will be executed.

The commands written to the Program Mode register are discussed later in more detail in Section 4.2.3.

Control flags F0, F3, and F5 in the Flag register determine which of the four control modes is executed. Only one control flag can be set at a time. After one of these control flags is set, the control modes are entered either automatically from Align or from the Initialize mode by writing 03H to the Program Mode register.

4.2.3. Emergency Flags and Status

The Stop and Limit inputs trigger hardware flags that signify the occurrence of an emergency condition and cause immediate change in the status of the effected axis. The Stop flag affects the axis only while in Integral Velocity Control mode. When the Stop flag is set, the axis will decelerate to a stop and remain in this mode with a command velocity of zero until the Stop flag is cleared and a new command integral velocity is specified. The Limit flag, when set in any control mode, causes the axis to go into Initialize mode, clearing the Motor Command and causing an immediate motor shutdown. When the Limit flag is set, none of the three control mode flags (F0, F3, or F5) are cleared.

The Stop and Limit flags are cleared when the inputs are disabled, signifying that the emergency condition has been corrected AND a write to the Status register (R1C00H) is executed. Any byte written to the Status register will try to clear the Stop and Limit flags, but, the lower 4 bits of that byte will also reconfigure the Status register. The interface library command 'clr_emergency' first reads the current configuration of the Status register before clearing the emergency flags so that the configuration does not change. Also, the interface library command 'enter_ctl_mode' clears the control mode flags before reestablishing Position Control mode.

The upper four bits of the Status register may be decoded to determine if an axis is profiling in Trapezoidal Profile control, in Initialize mode or whether the emergency conditions of Stop and Limit have occurred. The lower four bits of the Status register may be configured by writing the desired bit pattern as described in Table 4-6 (the upper four bits are ignored). Bit 0 controls the Sign Reversal Inhibit of the PWM command signal, while bits 1 and 2 are used to configure the Commutator described in Section 4.3. Bit 3 should always be set to 0.

Status Bit	Function
0	PWM Sign Reversal Inhibit 0 = off 1 = on
1	Commutator Phase Configuration 0 = off 1 = on
2	Commutator Count Configuration 0 = quadrature 1 = full
3	Always set to 0
4	Trapezoidal Profile Flag 1 = profiling
5	Initialize Flag 1 = in Initialize mode
6	Stop Flag 0 = Stop triggered
7	Limit Flag 0 = Limit triggered

Table 4-6. Status Register

4.2.4. Control Mode Operation

The HCTL-1100 has three set up routines and four control modes that may be executed. The three set up routines are:

- **Reset**
- **Initialize**
- **Align**

The four control modes available are:

- **Position Control**
- **Proportional Velocity Control**
- **Trapezoidal Profile Control**
- **Integral Velocity Control**

The HCTL-1100 switches from one mode to another as a result of one of the following three mechanisms:

1. By writing to the Program Mode register.
2. Setting/clearing flags F0, F3, or F5 by writing to the Flag register.
3. The controller switches automatically when certain conditions are met.

Figure 4-1 shows the flowchart for the set up routines and control modes, and shows the commands required to switch from one mode to another.

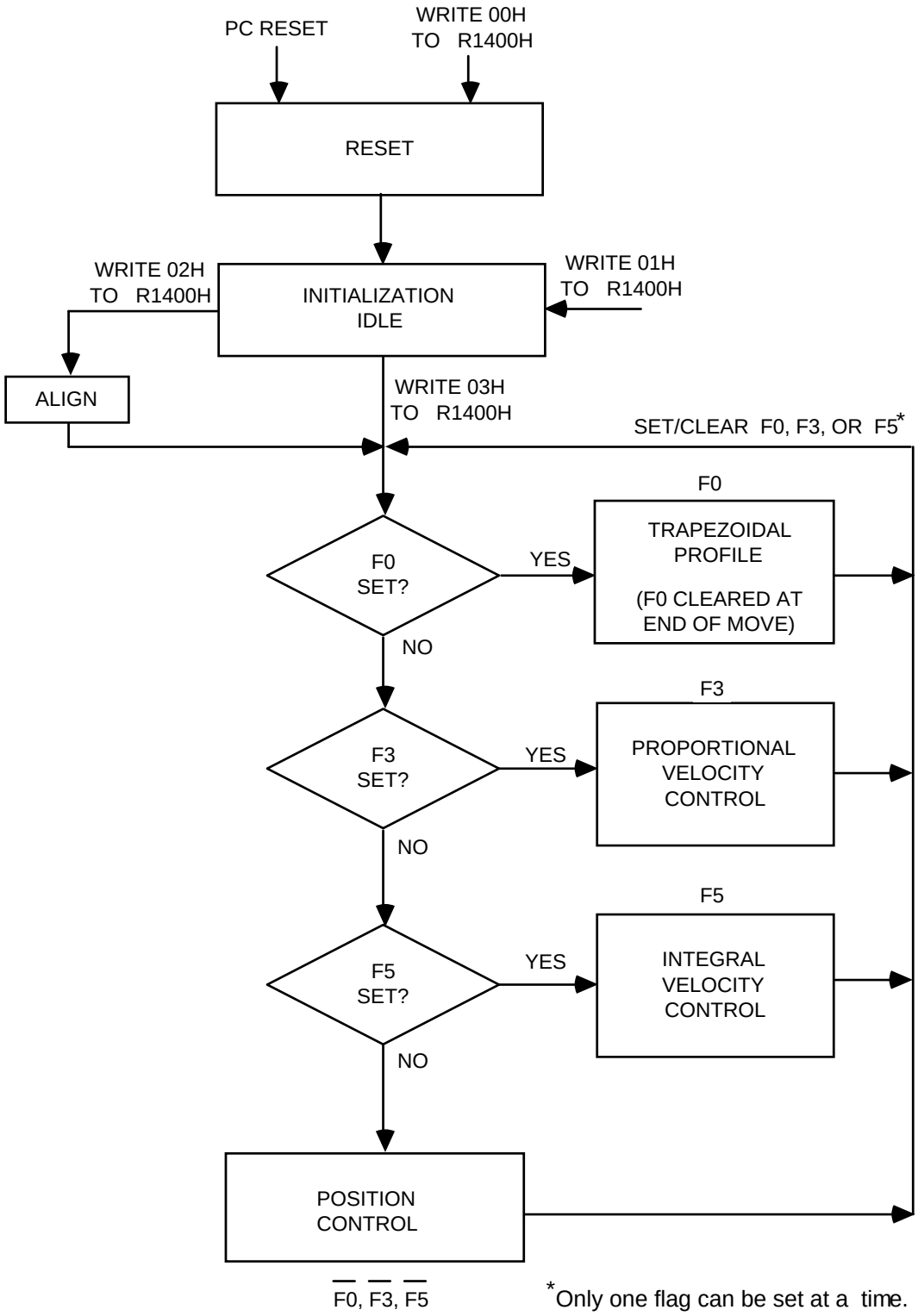


Figure 4-1. Operating Mode Flowchart

Reset and Initialization

This section describes the function of each set up routine and the default parameters that are preset.

Reset

The Reset mode is entered under all conditions by either a hard reset from the PC or a soft reset (write 00H to the Program Mode register). When you first turn on the PC, a hard reset occurs with the following results:

- All outputs are held low, except Sign and motor command.
- All flags (F0 through F5) are cleared.
- The PWM port is cleared to 0.
- The Motor Command port is preset to 80H (0V).
- The Commutator logic is cleared.
- The I/O control logic is cleared.
- A soft reset is executed.

When a soft reset is executed, the following conditions occur:

- The digital filter parameters are preset to:
  A = E5H (229D)
  B = K = 40H (64D)
- The sample timer is preset to 40H.
- The status register is cleared.
- The position counters are cleared to 0.

From Reset mode, the HCTL-1100 goes automatically to Initialize mode.

Initialize

The Initialize mode is entered either automatically from Reset or by writing 01H to the Program Mode register at any time. In the Initialize mode, the following conditions occur:

- The Initialize Flag (F1) is set.
- The PWM Motor Command Port is set to 00H.
- The Motor Command Port is set to 80H.

- Previously sampled data stored in the digital filter is cleared.

At this point you should pre-program all the necessary registers needed to execute the desired control mode. After setting up the control parameters, commands can be given to execute the desired motion.

Align Mode

The Align mode is executed only when the Commutator needs to be aligned to a multiphase motor. The Align mode can only be entered from Initialize mode by writing 02H to the Program Mode register (R1400H). Before attempting to enter the Align mode, the user should clear all control mode flags and set both the Command Position and Actual Position to zero.

The Align mode assumes: the encoder index pulse has been physically aligned to the last motor phase during encoder/motor assembly; the Commutator parameters have been correctly preprogrammed (reference Section 4.3 on the Commutator for more details); and a hard reset (PC's reset) had been executed while the motor is stationary.

The Align mode first disables the Commutator, and with open-loop control enables the first phase (PHA) and then the last phase (PHC or PHD) to orient the motor on the last phase torque detent. Each phase is energized for 2048 system sampling periods (1/f). For proper operation, the motor must come to a complete stop during the last phase enable. At this point the Commutator is enabled and commutation is closed-loop. After Align mode has been executed, the axis switches automatically to Position Control mode.

Position Control

Position Control performs point to point position moves with no velocity profiling. The 24 bits of the Command Registers specifies the desired position move. The position error is calculated from the desired command position and the 24 bits in the Actual Position Registers. The full digital lead compensator is applied and the calculated motor command is output. The controller will remain position locked at the command position until a new position command is given.

The actual and command position data is 24-bit 2's complement data stored in six 8-bit registers. Position is measured in encoder quadrature counts.

The command position resides in R3000H (MSB), R3400H, and R3800H (LSB). Writing to R3800H latches all 24-bits at once for the control algorithm. Therefore, the command position is written in sequence R3000H, R3400H, and R3800H. The command registers may be read in any desired order.

The actual position resides in R4800H (MSB), R4C00H, and R5000H (LSB). Reading R5000H latches the upper two bytes into an internal buffer. Therefore, Actual Position registers are read in the order of R5000H, R4C00H, and R4800H for correct instantaneous position data. The Actual Position registers can all be cleared to 0 by a write to R4C00H. The Actual Position may be set by writing to registers R5400H, R5800H, and R5C00H while in Initialize mode. Writing to register R5C00H latches data into all 24-bits.

Proportional Velocity Control

Proportional Velocity Control uses only the gain factor K from the digital compensator filter D(z). The algorithm compares the 16-bit Command Velocity registers with the 16-bit Actual Velocity registers and computes the velocity error. The velocity error is multiplied by K/4 and output as the motor command.

The Command Velocity and Actual Velocity are 16-bit 2's complement words. The units of velocity are encoder quadrature counts per sample time. In addition, the Command Velocity is internally divided by 16 to produce fractional resolution. The 16-bit velocity command is interpreted as 12-bits of integer and 4-bits of fraction. The Command Velocity resides in unlatched registers R9000H (MSB) and R8C00H (LSB). The registers can be read or written to in any order.

R9000H	R8C00H
IIII IIII	IIII.FFFF

Command Velocity Format

The actual velocity is computed only when in this mode and is stored in registers RD400H (MSB) and RD000H (LSB). There is no fractional component in the Actual Velocity registers and they can be read in any order.

The controller tracks the command velocity continuously until a new mode command is given. The system behavior after a new velocity command is governed only by the system dynamics until a steady state velocity is reached.

The Velocity Control mode is used for applications which require as fast a change in velocity as possible. This mode provides a "step response" velocity change without any filter compensation. The controller will return to the command velocity if the motor has stalled and will not try to "catch-up".

Integral Velocity Control

Integral Velocity Control performs continuous velocity profiling as specified by a command velocity and acceleration. Figure 4-2 shows the capability of this control mode. You can change velocity and acceleration any time to continuously profile velocity in time. Once the specified velocity is reached the controller maintains that velocity until a new velocity command is given. Any command velocity change will occur at the currently specified linear acceleration.

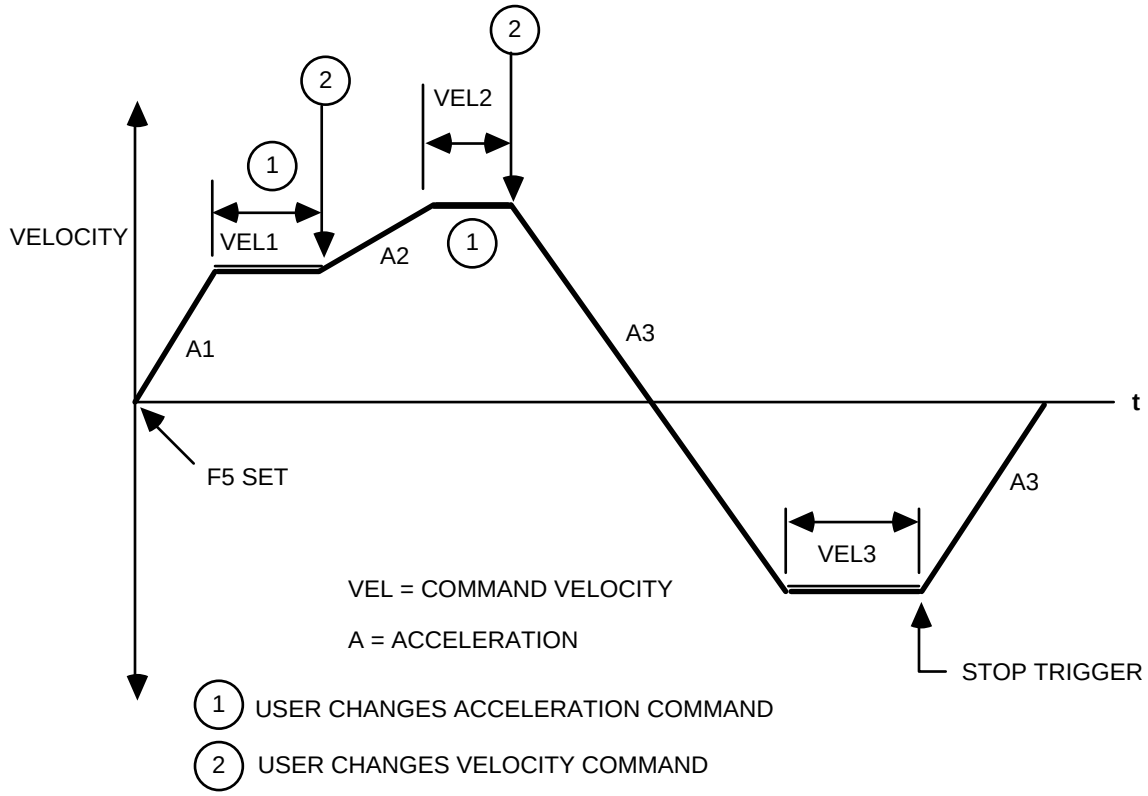


Figure 4-2. Integral Velocity Mode

The Command Velocity is an 8-bit 2's complement word stored in register RF000H. The units of velocity are quadrature counts per sample time. While the overall range of the velocity command is 8-bit 2's complement, the difference between any two sequential commands must be less than 7-bits magnitude (i.e.,127 decimal). For example, when executing a command velocity of 64H (+100D), the next velocity command must fall in the range of 7FH (+127D) (the maximum command range) to E5H (-27D).

The command acceleration is a 16-bit scalar word stored in registers R9C00H (MSB) and R9800H (LSB). The upper byte is the integer part and the lower byte is the fractional part. The integer part has a range of 00H to 7FH. The fractional part is internally divided by 256 to produce fractional resolution. The units of acceleration are quadrature counts per sample time squared.

R9C00H		R9800H
0III IIII	.	FFFF FFFF

Command Acceleration Format

Internally, the HCTL-1100 performs velocity profiling through position control. The controller generates position profiles based on the specified command velocity and acceleration. The advantage that this mode has over Proportional Velocity mode is that the system has zero steady state velocity error due to an added integral term in the profile generation. In the Integral Velocity mode, the system is actually a position control system and, therefore, the complete dynamic compensation D(z) is utilized.

If the external Stop line or Status bit 6 is asserted during this mode the controller automatically decelerates to zero velocity at the presently specified acceleration factor and stays in this condition until the flag is cleared. New velocity command data can then be given to restart Integral Velocity Control. The other control modes ignore the Stop flag and therefore are not effected by its assertion.

Trapezoidal Profile Control

Trapezoidal Profile Control performs point to point position moves and profiles the velocity trajectory to a trapezoid or triangle. The controller generates the necessary profile to conform to the acceleration, maximum velocity, and final position commands. If the maximum velocity is reached before the halfway distance point, the profile will be trapezoidal, otherwise the profile will be triangular. Figure 4-3 shows the two possible velocity profiles while in Trapezoidal Profile Control.

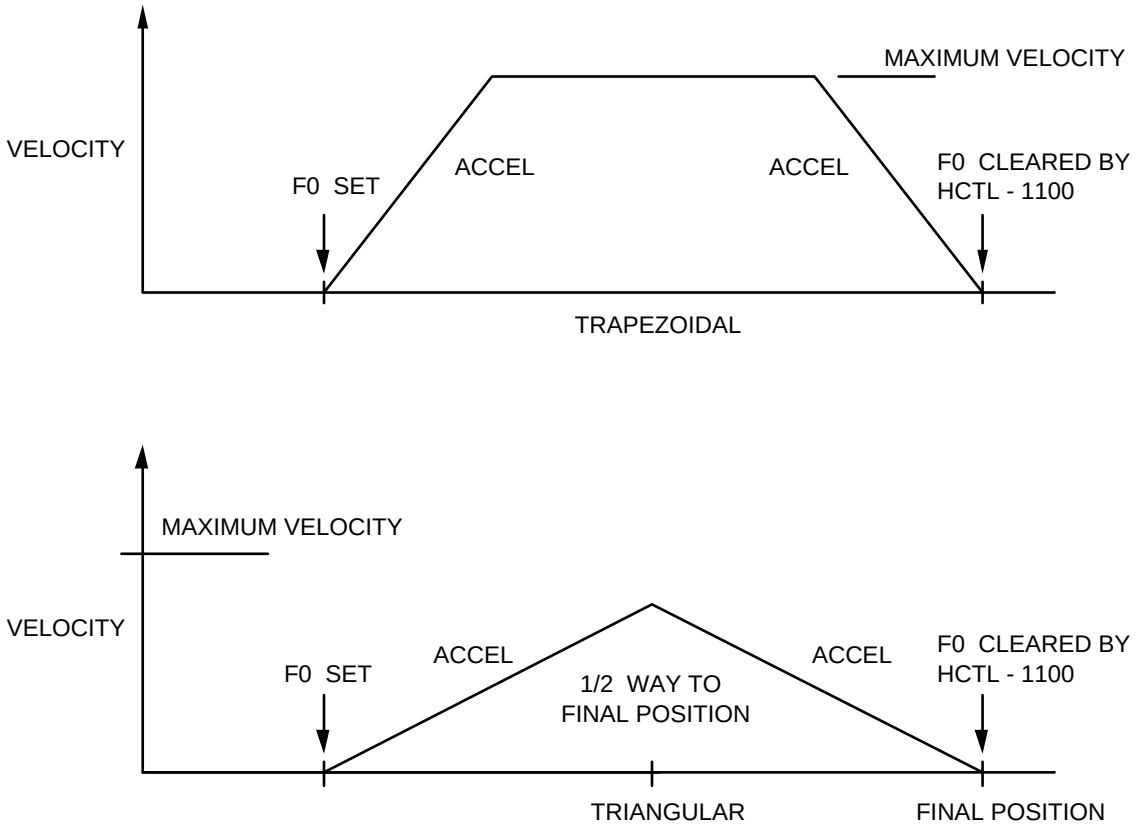


Figure 4-3. Trapezoidal Profile Control

The command data for this control mode is a 24-bit 2's complement final position written to RAC00H (MSB), RA800H, and RA400H (LSB). The acceleration resides in registers R9C00H (MSB) and R9800H (LSB). It has the same integer and fraction format as discussed under Integral Velocity Control. The maximum velocity is a 7-bit scalar range from 00H to 7FH written to register RA000H with units of quadrature counts per sample. The command data registers can be written or read in any order.

Once you have entered the desired data, flag F0 is set in the Flag register to commence motion. When the Trapezoidal Profile commands have been completed, the controller clears the F0 flag and locks on to the final position. The status of the Profile flag can be monitored in the Status register and at J1 Profile pin, or by the GRN Profile LED on the board. During a Trapezoidal Profile move, no new command data should be sent to the controller.

The internal profile generator in the controller produces a position profile using the Command Position (R3000H - R3800H) as the starting point and the Final Position (RAC00H -

RA400H) as the end point. The controller actually performs position control while the profile generator loads profile data into the Command Position registers. The full digital filter, D(z), is applied for compensation.

4.3. COMMUTATOR

The trend in new servo-systems is clearly towards the DC brushless motor. This type of motor generally has better reliability due to the lack of brushes and also has operational advantages. The brushless motor generally has a higher operating speed capability as its electronic commutation has no fundamental high frequency limit (like brush bounce of conventional brush motors). In addition, the brushless motor offers a higher peak torque, as current is only limited by voltage and winding resistance, and not by the maximum acceptable current density at the brush/commutator interface (assuming demagnetization limits do not apply). Finally, as heat is generated in the outer stator (I^2R losses) instead of in the rotor, better cooling is possible and therefore a higher continuous torque rating is possible.

The Commutator is used to output the proper phase sequences for electrical commutation of multi-phase motors. Variable reluctance, brushless DC, and stepper motors require electronic commutation and may be controlled with the Commutator's outputs. However, most brushless DC motor drivers will run directly from the ±10V analog command output without use of the provided commutator. The Commutator is useful in lowering component cost due to driving the motor coils directly with H-bridge type amplifiers and achieving higher speeds due to electronic phase advance.

4.3.1. Configuration Registers

The commutator is designed to work with 3 phase and 4 phase motors of various winding configurations and with various shaft encoder counts. Two phase motors may also be controlled by selecting the 4 phase configuration and ANDing the phase outputs appropriately.

Besides the correct phase enable sequence, the Commutator provides programmable phase overlap and phase advance. Phase overlap means that more than one phase output will be enabled at the transition between phases. Phase overlap is used for better torque ripple control and may also increase the average torque output of the motor. Phase advance causes the phase output to be ahead of the normal point in the rotation of the motor shaft, and allows

you to compensate for the frequency characteristics of the motor/driver combination. The amount of phase advance can be programmed to linearly increase with an increase in the velocity of the motor's shaft. The phase advance feature allows more time for current to build up in each coil of the motor which results in higher torques at high speeds. The Commutator can also be used to generate unique sequences by further decoding the phase outputs externally to drive more complex motors and drivers.

The outputs of the Commutator are located at receptacle J2 and are labeled PH-A, PH-B, PH-C, and PH-D. The Commutator uses both channels and the index pulse of an incremental encoder. The index pulse of the encoder must be physically aligned to the motor's torque cycle location so that this point may be used as the reference point with respect to the Commutator phase outputs. The index pulse should be permanently aligned during motor encoder assembly to the last motor phase. This is done by energizing the last phase of the motor during assembly and permanently attaching the encoder code wheel to the motor shaft so that the index pulse is active. Fine tuning of alignment for commutation purposes is done electronically using the Offset register once the complete control system is set up.

The Commutator is programmed by the data in the following registers:

- **Status Register**
- **Commutator Ring Register**
- **X Register**
- **Y Phase Overlap Register**
- **Offset Register**
- **Velocity Timer Register**
- **Maximum Phase Advance Register**

Figure 4-4 shows an example of the relationship between all the parameters. The following headings describe each of these registers with the Exerciser command given in parentheses. Refer to the MCP-04 Software section for more details.

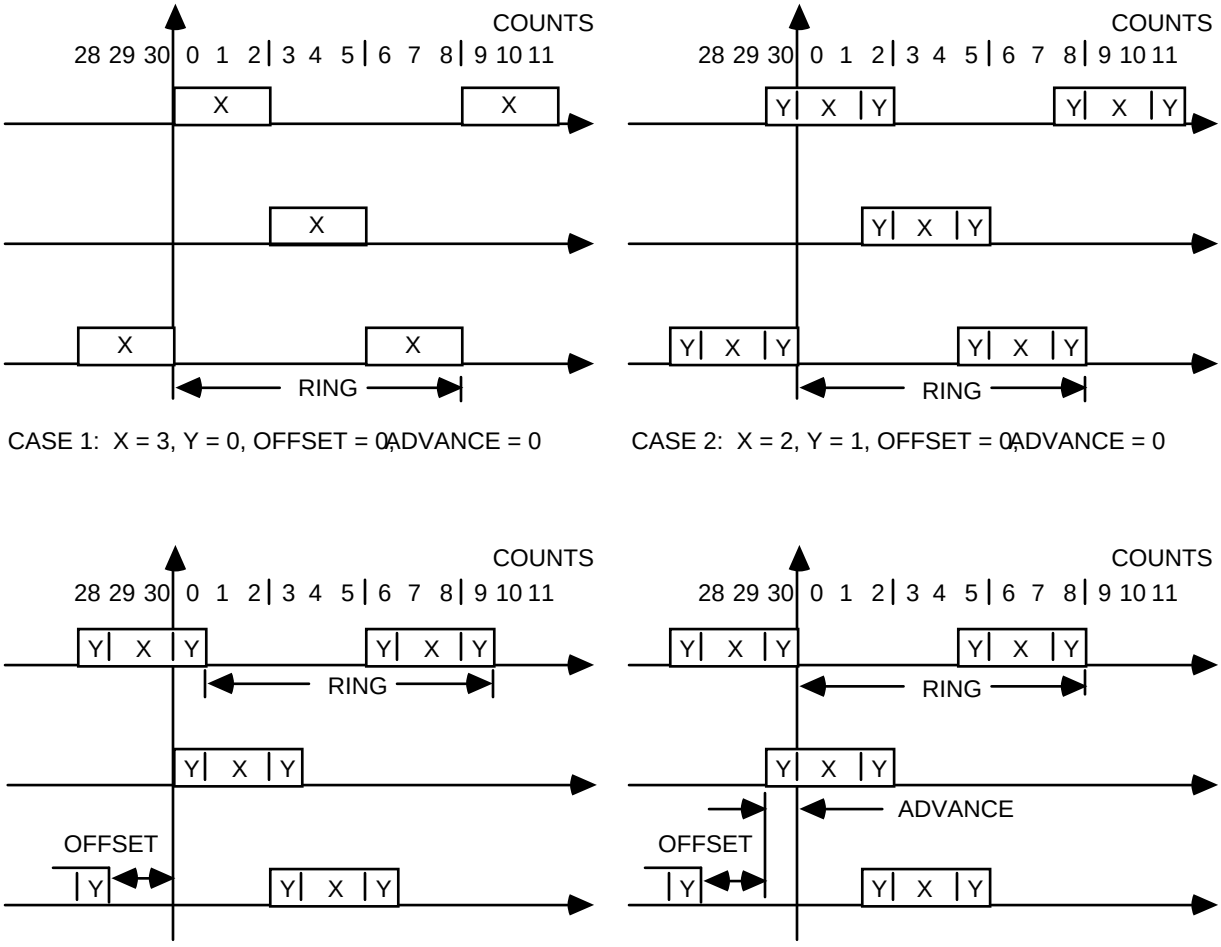


Figure 4-4 Commutator Configuration

Status Register (STATUS)

Bit #1 0 = 3 phase configuration.
1 = 4 phase configuration.
Bit #2 0 = position measured in quadrature counts.
1 = position measured in full counts.

Commutator Ring Register (RING)

The Ring register is a scalar and determines the length of the electrical cycle measured in full or quadrature counts as set by bit #2 in the Status register. The magnitude of the Ring is limited to 7FH.

X Register (X_REG)

The X register sets the interval during which a phase is active without overlap. The data must be a scalar 00H to 7FH.

Y Phase Overlap Register (Y_REG)

The Y Phase Overlap register sets the interval during which two sequential phases are both active. The data must be a scalar 00H to 7FH. X and Y must satisfy Equation 4-3.

$$X + Y = \text{Ring} / (\#\ \text{of phases}) \qquad (4\text{-}3)$$

The Ring, X, and Y registers define the basic electrical commutation cycle.

Offset Register (OFFSET)

The Offset register contains the 2's complement data which determines the relative start of the electrical cycle with respect to the index pulse. Since the index pulse must be physically referenced to the rotor, the offset performs fine alignment between the electrical and mechanical torque cycles.

Velocity Timer Register (VEL_TIMER)

The phase advance feature performs the function of linearly increasing the phase advance according to measured speed up to a set maximum. The Velocity Timer register contains scalar data (0H to FFH) which determines the amount of phase advance at a given velocity. The phase advance is interpreted in the units set for the ring counter by bit #2 in the Status register. The velocity is measured in revolutions per second.

$$\text{Advance} = Nv(\Delta t) \qquad (4\text{-}4)$$

where:

$\Delta t = 8 \times 10^{-6}$ (VEL_TIMER + 1)*
N = encoder counts/revolution

*Assuming a 2 MHz board clock frequency setting.

v = velocity (rev/sec.)

If the phase advance feature is not used, set the Command Velocity Timer register to zero.

Maximum Phase Advance Register (MAX_ADV)

The scalar data in the Maximum Phase Advance register sets the upper limit for phase advance regardless of rotor speed. Figure 4-5 shows the relationship between the Velocity Timer and Maximum Phase Advance registers. If the phase advance feature is not used, set the Maximum Phase Advance register to zero.

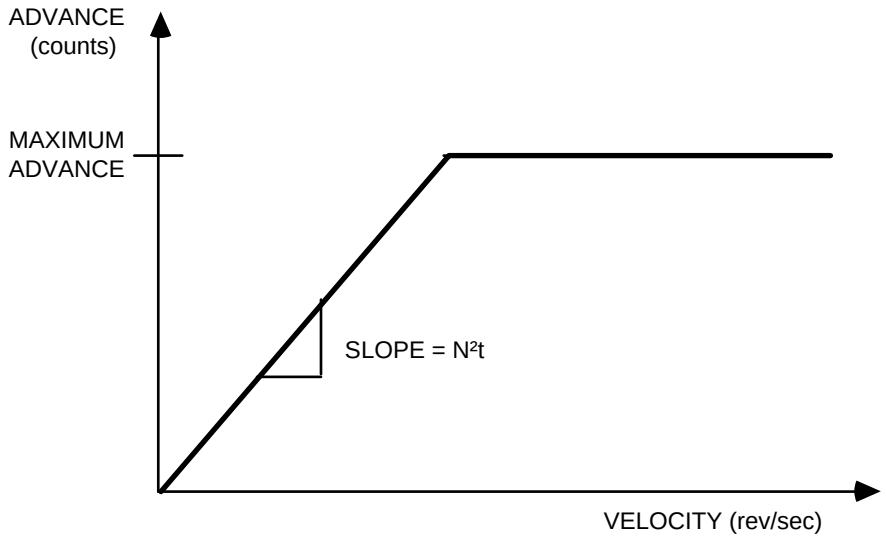


Figure 4-5. Phase Advance vs. Motor Velocity

4.3.2. Commutator Constraints

There are several numerical constraints to be aware of when using the HCTL-1100 Commutator. The parameters of Ring, X, Y, and Maximum Phase Advance must be positive numbers (00H to 7FH). The Offset register is an 8-bit 2's complement number. And, the equation below must be satisfied.

$$80H \leq 1.5\ \text{Ring} + \text{Offset} \pm \text{Max Advance} \leq 7FH \qquad (4\text{-}5)$$

The Commutator works on a circular ring counter principle whose range is defined by the Commutator Ring register. This means that for a ring of 96 counts and a needed offset of 10D, the Offset register can be programmed as 0AH (10D) or AAH (-86D), the latter satisfying Equation 4-5. Due to the restrictions imposed on the value of Ring, the resolution of the shaft encoder is limited for a given type of motor commutation. A motor with a large number of commutation cycles can have a larger resolution encoder.

Example: Determine the highest resolution shaft encoder that may be used to commutate a 4-phase, 4-pole brushless DC motor.

1). Select the 4-phase and full count mode for the Commutator by writing a 6 to the Status register. The Commutator full count mode should be used to obtain higher position resolution due to Equation 4-5.

2). A 4 phase, 4-pole motor will provide two torque cycles of four phases and 90 degree electrical torque cycles. So that:

Ring register = (encoder counts) / 2 torque cycles

3). By measuring the motor torque curve in both directions, it is determined that an offset of 3 degrees, and a phase overlap of 2 degrees is desired.

$$\text{Offset Register} = 3° \frac{\text{encoder counts}}{360 \text{ degrees}}$$

Since the maximum value for the Ring register is 127 decimal, the maximum encoder counts is 2 x 127 **= 254 counts**. An offset is needed of about 2 counts to give the required 3 degree offset. This is equivalent to 02H (2D) or 83H (-125D), the latter satisfying Equation 4-5. An encoder with 254 counts in quadrature provides less than 1/3rd of a degree resolution which is perfect for applications requiring high speeds. Applications requiring higher accuracy should use a motor with more commutation cycles or a higher transmission ratio. For example, a 4 phase 7.5 degree step angle stepper motor provides 12 commutation cycles per revolution. This type of motor could use a shaft encoder with 1524 quadrature counts for 1/17th of a degree resolution.

The recommended interface to DC brushless motors is given in Figure 4-6. The PWM Motor Command Port and the Commutator outputs are combined using AND gates to drive power

drivers. The diagram shows all four phases of the Commutator being used, but only three phases would be required for a three phase DC brushless motor.

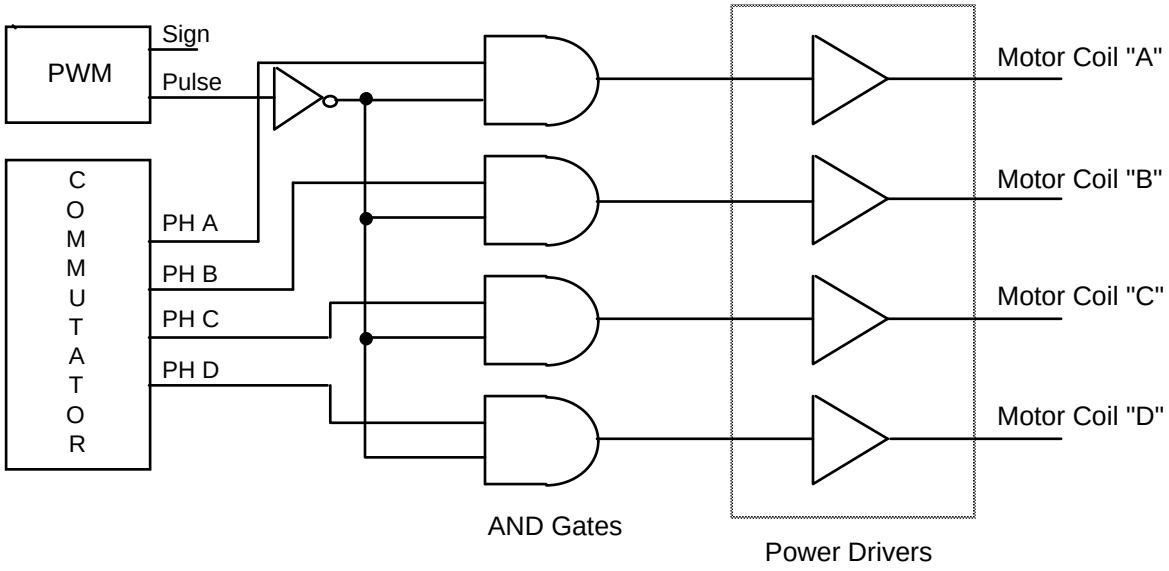


Figure 4-6. Interface to Brushless DC Motors

Section 5

SYSTEM MODELING AND TUNING

The design of your closed-loop motion control system will require the tuning of the HCTL-1100's digital compensator. This section describes an analytical method for tuning the digital compensator. An experimental procedure was given in Section 3.2.2 of this manual. The main difficulty of the analytical design method is that it requires knowledge of all the system parameters. The models are linearized and have proven reliable for most applications. The parameter that is most often difficult to estimate is the load inertia in complex structures or multi-link configurations. Many designers use a "worst case" inertia and tune the controller for that value. Other designers break down the inertial changes into four or five ranges and change the tuning accordingly. Other approaches may be implemented including software that may adjust the compensator parameters in real-time. One advantage of the digital compensator over an analog approach is that it allows much more flexibility in the design approach.

The material in this section assumes a general working knowledge of analog control design methods. Specifically, you should be familiar with Laplace transforms, the s-domain, pole/zero concepts, and Bode plots. The analytical method described computes the open-loop transfer function in the s-domain and uses the Bode plot to find the gain and phase margins. The digital compensator is used to increase the system bandwidth and is computed directly in the z-domain by referencing normalized frequency plots for the pole and zero of the digital compensator in the HCTL-1100. The z operator can be modeled as e^{sT} in the s-domain.

The steps recommended for designing a motion control system are as follows:

Step 1. Choose a motor and transmission for the required load. A sufficient torque margin at both continuous and duty cycle operation should be specified so that the maximum motor current will never be exceeded. Another common criteria for choosing a motor is based on the ratio of the peak torque (T_P) developed by the motor to the moment of inertia of the motor's armature and load ($J_M + J_L$). The objective is to select the motor with the maximum ratio of

$$\frac{T_P}{J_M + J_L}$$

Step 2. Choose a quadrature incremental encoder to monitor the motor's shaft position based on the encoder resolution and accuracy required for the application. A differential line driver output is recommended for added noise immunity.

Step 3. Choose an amplifier to drive the motor. The amplifier must be capable of supplying the current and voltage required by the motor for the load conditions. A pulse-width-modulated amplifier is recommended over a linear amplifier due to power efficiency and cost benefits.

Step 4. Model the open-loop transfer function of the system using s-plane transfer functions. A Bode plot showing phase margin and gain margin of the open-loop system can then be drawn from the open-loop transfer function.

Step 5. Choose the desired phase margin and gain-crossover frequency for the compensated system. The closed-loop response (step response and bandwidth) will be directly affected by these two Bode plot measurements.

Step 6. Find the HCTL-1100's digital compensation filter parameters based on the desired phase margin and gain-crossover frequency for the compensated system. This is done by using the normalized frequency plots given later in this section.

5.1. MODELING THE SYSTEM COMPONENTS

In order to understand the operation of the control system we need a mathematical model for all the system components. The functional elements of the control system shown in Figure 5-1 include a controller (the HCTL-1100's digital filter and the zero-order-hold), an amplifier, a motor (includes the load), and an incremental encoder.

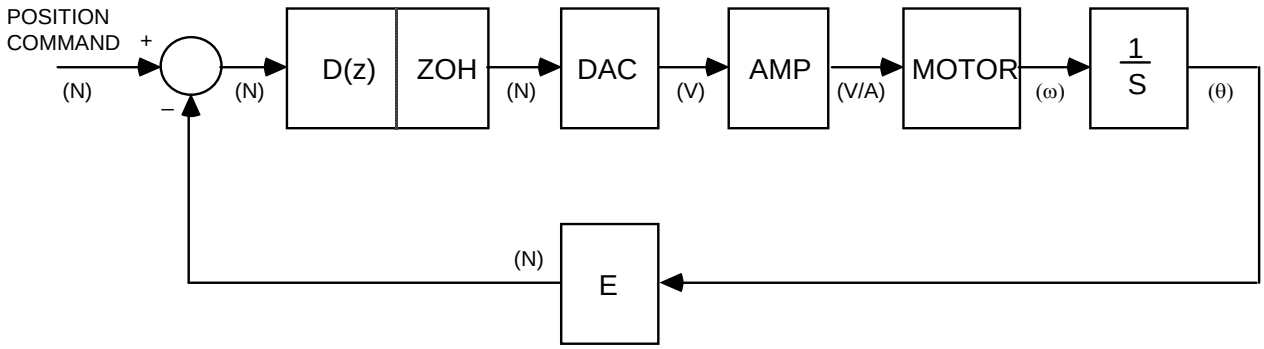


Figure 5-1. Functional Elements of the Control System

The open-loop transfer function, M(s), for this system is determined by multiplying all of the individual component transfer functions together.

$$M(s) = [ZOH][DAC][Amp][Motor][Encoder] \quad (5\text{-}1)$$

5.1.1. Zero Order Hold Transfer Function

The zero order hold (ZOH) models the delay of the discrete sampling time of the HCTL-1100. The delay is estimated to be half of the sampling period or $z^{-1/2}$. The continuous time transfer function of the ZOH is:

$$Z(s) = e^{-sT/2} \quad (5\text{-}2)$$

where:

T = the sampling time of the HCTL-1100 in seconds.

The magnitude contribution of the ZOH is unity for all frequencies. The phase contribution of the ZOH is:

$$P_{ZOH} = \omega \frac{T}{2} \text{ (radians)} \quad (5\text{-}3)$$

where:

ω = the frequency of interest (rad/sec)

T = the sampling time of the HCTL-1100 (sec)

Phase lag is added to the system by increasing the sampling time of the digital compensator. It is usually desirable therefore to choose the fastest sampling time possible so as to induce the least amount of phase lag into the system. Increasing the sampling time also allows for a higher possible system bandwidth. Generally, the sampling frequency should exceed the system bandwidth (in hertz) at least tenfold.

5.1.2. DAC Transfer Function

The MCP-04 uses an 8-bit DAC to output a ±10 volt range command signal. The DAC's transfer function is simply its gain (K_D). The DAC does not contribute a phase shift to the open-loop transfer function. The gain that should be used is:

$$K_D = 10/256 = .039 \text{ (volts/count)} \quad (5\text{-}4)$$

5.1.3. Amplifier Transfer Function

The amplifier transfer function is its gain (K_A). The amplifier does not contribute a phase shift to the open-loop transfer function when its electrical time constant is neglected. In general, if the desired bandwidth of the system is 10 times smaller than the amplifier's bandwidth, then the amplifier's electrical time constant can be neglected.

If the PWM port is used, the amplifier will be regulated by a range of 64H to 9CH (-100 to 100 decimal). The units of the gain (K_A) is a combination of the PWM port and the amplifier. A simplified formula for computing the gain of a PWM amplifier is:

$$K_A = \frac{\text{[max output] - [min output]}}{\text{[max duty cycle] - [min duty cycle]}} \quad (5\text{-}5)$$

The choice of using either a current or voltage amplifier *affects the motor transfer function*.

5.1.4. DC Motor Transfer Function

The motor transfer function includes the inertial loads connected to the motor shaft. The inertia of the load as seen by the motor must be calculated. A reference book on mechanical design should be consulted if required to determine the correct moment of inertia. The total moment of inertia for the system (J) includes the load inertia (J_L), the motor's armature inertia (J_M), and the encoder codewheel inertia (J_C). Therefore:

$$J = J_L + J_M + J_C \tag{5-6}$$

The DC motor parameters of interest that describe the electro-mechanical characteristics are the torque constant (K_T)[Nm/amp], the moment of inertia of the armature (J_M)[kg-m^2], the terminal resistance (R)[ohms], the voltage constant (K_E)[V-sec/rad], and armature inductance (L)[henries].

Different transfer functions are used for a motor driven by a voltage source amplifier than for a motor driven by a current source amplifier.

DC Motors Driven by Voltage Source Amplifiers

The DC motor transfer function for a motor driven by a voltage source amplifier is:

$$G(s) = \frac{\text{position input}}{\text{voltage input}} = \frac{\theta(s)}{V(s)} = \frac{1/K_E}{s(sT_M + 1)(sT_M + 1)} \tag{5-7}$$

for $T_M > 10\ T_E$.

where

$$T_M = \frac{RJ}{[K_E][K_T]}\ \text{(sec)} \tag{5-8}$$

is the *mechanical time constant*, and

$$T_E = \frac{L}{R}\ \text{(sec)} \tag{5-9}$$

is the *electrical time constant.*

Note that the s in the denominator indicates integration due to the fact that position is the output. The term J is the total system inertial load that the motor is driving, K_E is the voltage constant of the motor, and K_T is the torque constant of the motor.

The phase contribution of a motor ($P_M(\omega)$), driven at frequency ω by a voltage source is:

$$P_M(\omega) = -\arctan(\omega T_M) - \arctan(\omega T_E) - \pi/2 \text{ (radians)} \quad (5\text{-}10)$$

The magnitude contribution of a motor ($M_M(\omega)$), driven at frequency ω by a voltage source is:

$$M_M(\omega) = \frac{\frac{1}{K_E}}{\omega\sqrt{1 + (\omega T_M)^2}\sqrt{1 + (\omega T_E)^2}} \quad (5\text{-}11)$$

where

ω = the frequency of interest in rad/sec.

DC Motors Driven by Current Source Amplifiers

The DC motor transfer function for a motor driven by a current source is

$$G(s) = \frac{\text{position output}}{\text{current input}} = \frac{\theta(s)}{I(s)} = \frac{K_T}{Js^2} \quad (5\text{-}12)$$

The phase contribution of a motor ($P_M(\omega)$), driven at frequency ω by a current source is

$$P_M(\omega) = -\pi \text{ (radians)} \quad (5\text{-}13)$$

The magnitude contribution of a motor ($M_M(\omega)$), driven at frequency ω by a current source amplifier is:

$$M_M(\omega) = \frac{K_T}{J\omega^2} \quad (5\text{-}14)$$

where

ω = the frequency of interest in rad/sec.

5.1.5. Encoder Transfer Function

The incremental encoder's transfer function (E) is

$$E = \frac{C}{2\pi} = \frac{4N}{2\pi} \text{ (counts/rad)} \qquad (5\text{-}15)$$

where

C = quadrature counts per revolution.
N = the number of slits in the codewheel per revolution

The encoder's codewheel count does not contribute to the phase of the open-loop transfer function. The magnitude contribution of the encoder's codewheel count (E) to the open-loop transfer function is a constant.

$$E = \frac{C}{2\pi} \text{ (counts/rad)} \qquad (5\text{-}16)$$

The phase and magnitude contribution of the codewheel's inertia (J_C) has already been included in the total system load on the motor.

5.2. TUNING THE DIGITAL COMPENSATION FILTER

Now that all of the individual transfer functions have been determined for each component, the open-loop transfer function M(s) may be calculated by multiplying each individual transfer function together. The Bode plot can then be plotted for the open-loop transfer function. The purpose of the Bode plot is to show the frequency response of the uncompensated system. The following previously defined variables will be used to calculate the phase and magnitude equations.

E = gain of the incremental encoder (counts/rad)

K_A = gain of the amplifier (volts/volt or amps/volt for linear amplifiers and volts/count or amps/count for PWM amplifiers)
K_D = gain of the DAC (volts/count)
K_E = voltage constant of the motor (volt-sec/rad)
K_T = torque constant of the motor (N-m/amp)
J = total system moment of inertia (kg-m^2)
T_E = electrical time constant of the motor (sec)
T_M = mechanical time constant of the motor (sec)
$M_M(\omega)$ = magnitude of motor transfer function
$P_M(\omega)$ = phase of motor transfer function (radians)
$P_{ZOH}(\omega)$ = phase of zero order hold transfer function (radians)
T = sampling time of the HCTL-1100 (sec)
ω = frequency of interest (rad/sec)

The phase of the uncompensated open-loop transfer function ($P_U(\omega)$) is determined by the equation

$$P_U(\omega) = P_M(\omega) + P_{ZOH}(\omega) \text{ (radians)} \quad (5\text{-}17)$$

For a system employing a voltage source amplifier, the following equation is used to calculate the phase of the open-loop transfer function:

$$P_U(\omega) = -\arctan(\omega T_M) - \arctan(\omega T_E) - \pi/2 + \omega T/2 \text{ (radians)} \quad (5\text{-}18)$$

For systems with a current source amplifier, the equation becomes:

$$P_U(\omega) = -\pi - \omega T/2 \text{ (radians)} \quad (5\text{-}19)$$

Note: multiply $P_U(\omega)$ by $180°/\pi$ to obtain the phase in degrees.

The magnitude of the uncompensated open-loop transfer function ($M_U(\omega)$) is

$$M_U(\omega) = [M_M(\omega)][K_D][K_A][E] \quad (5\text{-}20)$$

The magnitude of the uncompensated open-loop transfer function using a voltage source is

$$M_U(\omega) = \frac{\frac{1}{K_E}[K_D][K_A]\frac{C}{2\pi}}{\omega\sqrt{1 + (\omega T_M)^2}\sqrt{1 + (\omega T_E)^2}} \tag{5-21}$$

The uncompensated magnitude for the open-loop transfer function of systems with a current amplifier is

$$M_U(\omega) = \frac{K_T K_D K_A C}{2\pi J\omega^2} \tag{5-22}$$

Note: to express the magnitude in db, use

$$db = 20\log(M_U(\omega)) \tag{5-23}$$

5.2.1. Determination of the Gain and Phase Margin

The next step is to make a Bode plot of the resulting equations for magnitude and phase and determine the gain and phase margins. Gain margin is the amount of gain in decibels that can be allowed to increase in the loop before the closed-loop system reaches instability. The phase margin is a measure of the relative stability of the closed-loop system expressed in degrees. As an illustrative example, consider that the open-loop transfer function of a system is given by

$$G(s) = \frac{10}{s(1 + 0.2s)(1+ 0.02s)}$$

The Bode plot of G(s) is shown in Figure 5-2. The frequency at which the open-loop gain is 1 (0 dB) is defined as the *gain-crossover frequency* (ω_c). The frequency when the phase angle is -180° is called the *phase-crossover frequency* (ω_p). The *gain margin* (GM) is defined as the magnitude of the open-loop transfer function evaluated at the phase-crossover frequency and referenced to the 0 dB axis. The *phase margin* (PM) is defined as the sum of the phase angle of the open-loop transfer function evaluated at the gain-crossover frequency plus 180 degrees. If both the phase and gain margins are positive the closed-loop system will be stable.

The gain margin (GM) equation is

$$GM = -20 \log(M_U(\omega_p)) \text{ (db)} \quad (5\text{-}24)$$

The phase margin (PM) equation is

$$PM = 180° + P_U(\omega_c)(180°/\pi) \text{ (degrees)} \quad (5\text{-}25)$$

Figure 5-2. Bode Plot of G(s) = 10/[s(1 +0.2s)(1 + 0.02s)]

5.2.2. Modification of the Open Loop Transfer Function

The open-loop transfer function can be modified with the MCP-04 digital compensator to improve the bandwidth and stability of the closed-loop system. The stability and the band-

width of the closed-loop system may be increased by increasing the phase margin and the gain-crossover frequency of the system. The digital compensation filter can contribute a maximum phase lead of approximately 80 degrees to the uncompensated phase margin. The amount of bandwidth that can be added by the digital filter is dependent on the system parameters. The gain of the filter and the amplifier can be used to change the gain margin of the system.

A realistic gain-crossover frequency should be selected to provide the desired system bandwidth. A system bandwidth above 60 Hz should be obtainable for most motion control systems. A well damped system should have a phase margin between 30° and 45°. The desired gain-crossover frequency and phase margin will be required for the design method that follows.

The uncompensated phase margin at the desired gain-crossover frequency (ω_C') must be calculated for the open-loop transfer function using equation (5-17). The phase lead (P_L) that the digital compensator filter must provide is found from the following equation.

$$P_L = PM_C - PM_U \text{ (degrees)} \tag{5-26}$$

where

PM_C = the desired compensated phase margin at ω_C'
PM_U = the uncompensated phase margin at ω_C'

The gain (K_F) required from the digital filter to make the gain equal to one at ω_C' is

$$K_F = \frac{1}{M_U(\omega_C')} \tag{5-27}$$

where

$M_U(\omega_C')$ = the magnitude of the uncompensated open-loop system at the desired gain-crossover frequency (ω_C')

The digital compensation filter is of the form

$$D(z) = \frac{[K][z-A]}{[4][z+B]} = \left[\frac{K}{4}\right]\left[\frac{z-A}{z}\right]\left[\frac{z}{z+B}\right] \tag{5-28}$$

Both the pole term (B) and the zero term (A) add phase lead to the system. The K term may be used to adjust the gain and compensate for the gain reduction associated with the digital filter's pole and zero term. The combination method uses graphs of the digital compensator pole and zero to determine values for the parameters A,B and K. Normalized frequencies are used in the graphs to allow a wide range of bandwidths and sample times. The normalized frequency ($\omega_N(\omega)$) is calculated as the frequency (ω) multiplied by the sampling time (T) of the system.

$$\omega_N(\omega) = [\omega][T] \text{ (radians)} \tag{5-29}$$

The normalized frequency plots for the phase and magnitude of the pole and zero terms are the result of the direct mapping of the digital compensation filter into the continuous time domain.

The graphs for the pole term are derived by substituting $z = e^{j\omega t}$ into the $z/(z + B)$ portion of Equation 5-28. The phase of the pole is found by taking the argument to get

$$P_P(\omega_N) = \arg\left[\frac{e^{j\omega T}}{e^{j\omega T} + B}\right] = -\arctan\left[\frac{B\sin\omega T}{B\cos\omega T + 1}\right] \tag{5-30}$$

The magnitude of the pole term is

$$M_p(\omega_N) = \left|\frac{e^{j\omega T}}{e^{j\omega T} + B}\right| = \sqrt{[B\cos\omega T + 1]^2 + [B\sin\omega T]^2} \tag{5-31}$$

The graphs of the zero term are derived from substituting $z = e^{j\omega T}$ into $(z - A)/z$ portion of Equation 5-28. The phase and magnitude of the zero term are derived in a similar manner and are given below.

$$P_Z(\omega_N) = \arg\left[\frac{e^{j\omega T} - A}{e^{j\omega T}}\right] = -\arctan\left[\frac{A\sin\omega T}{-A\cos\omega T + 1}\right] \tag{5-32}$$

and

$$M_Z(\omega_N) = \left|\frac{e^{j\omega T} - A}{e^{j\omega T}}\right| = \sqrt{[-A\cos\omega T + 1]^2 + [A\sin\omega T]^2} \tag{5-33}$$

Figure 5-3, 5-4, 5-5, and 5-6 show the results of plotting Equations 5-30 thru 5-33. Note that for all four graphs the value of the pole term (B) and the zero term (A) is always less than or equal to 1.0 for proper system design. Follow the guide lines outlined below to design the digital compensation filter by the combination method.

Step 1. Using Figure 5-3, choose a large value for the pole term (B) to contribute significant phase lead at the normalized desired gain-crossover frequency of the system.

Step 2. On Figure 5-4, read the value of the magnitude for the chosen value of B at the normalized desired gain-crossover frequency.

Step 3. Determine the remaining phase lead which must be provided from the zero term (A) of the digital compensator using the equation:

$$P_Z(\omega_{NC}') = P_L - P_P(\omega_{NC}') \text{ (degrees)} \qquad (5\text{-}33)$$

where

P_L = the required phase lead contributed from the digital compensator
$P_P(\omega_{NC}')$ = the phase lead contributed by the pole term at the desired normalized gain-crossover frequency

Step 4. On Figure 5-5, find the intersection of the normalized desired gain-crossover frequency and the required phase lead required from the zero term. Estimate the value of the zero term A at this point.

Step 5. From Figure 5-6, find the value of the zero term's magnitude by plotting the estimated zero term (A) at the desired normalized gain-crossover frequency.

Step 6. Use the following equation to find the value of the gain (K):

$$K = \frac{K_F}{M_Z(\omega_{NC}')\, M_P(\omega_{NC}')} \qquad (5\text{-}34)$$

where

K_F = the gain required by the digital filter to provide the desired gain-crossover frequency
$M_Z(\omega_{NC}')$ = the magnitude of the zero term
$M_P(\omega_{NC}')$ = the magnitude of the pole term

Step 7. Program the the digital filter parameters with the following:

Zero = A
Pole = B
Gain = K

Figure 5-3. Phase Lead Contribution of the Pole Term

Figure 5-4. Magnitude Contribution of the Pole Term

Figure 5-5. Phase Lead Contribution of the Zero Term

Figure 5-6. Magnitude Contribution of the Zero Term

Section 6

MCP-04 SOFTWARE

This section describes the Mektronix supporting software for the MCP-04 boards. The following is a list of the software programs provided with the distribution disk.

- Exerciser
- Check Utility
- C Interface Library
- Windows 98 DLL
- Windows NT Driver

The MCP-04 Exerciser allows user commands to be issued from the PC's keyboard or from a text file. It serves as a motion control development tool as well as a way to become familiar with the MCP-04 programming language interface libraries. The collection of Exerciser commands is not intended to be used as a programming language. User application programs should be implemented using high level languages such as C++.

We include the source code on the distribution disk for the high level programming language interface libraries so that you can examine how we have implemented the routines. You can compile these routines with your own compiler and then link them with your application programs.

6.1. SOFTWARE OPERATION

This sub-section describes the general principles used in implementing the software provided with the MCP-04 board. Important information about limiting and rounding conventions, board numbering and global axis numbering are discussed. We also include some tips for trouble shooting.

6.1.1. Limiting and Rounding Conventions

Whenever a register is set with a value out of its defined range, it will be limited with the corresponding boundary values. The only exceptions are the filter parameters zero and pole, where special rules are used. If a register is to be set with a real value that exceeds the register resolution, the value is *rounded* to the nearest possible setting. The actual setting in effect may be obtained as an echo from the Exerciser or as the return value of a library routine. Consult the specific manual pages in Section 6.4 for more information.

6.1.2. Board Numbers and Global Axis Numbers

MCP-04 software assigns a *global axis number* to each axis in your system, which may consists of a number of MCP-04 boards. The board numbers and global axis numbers are described in the Installation section.

6.1.3. Initialization

The MCP-04 boards must be initialized each time the host personal computer is booted. The initialization process consists of two procedures:

1) setting the DOS environment variable MCINIT by running the MCINIT.BAT file, and

2) calling the MCINIT library function from within an application program.

The only exception to the above is when the system consists of one MCP-04 whose starting address is set to 3E0H, in which case the MCINIT.BAT is not required. You may check the current DOS environment by typing

x> **set**

If environment variable MCINIT does not show up, locate and then run MCINIT.BAT. Make sure that a valid MCINIT environment shows up when command 'set' is issued from DOS before trying to run the Exerciser again.

If your application program was developed using the provided interface libraries and does not seem to work, you should first check the DOS environment variable MCINIT as described above. Then make sure subroutine 'mc_init' ('mc.init' in BASIC) is called before any subroutine or function in the MCP-04 Programming Interface Library.

6.2. MCP-04 EXERCISER

The MCP-04 Motion Controller Exerciser is designed to work with a motion control system employing any number of MCP-04 boards. It is useful as a motion control debugging tool and to become familiar with the commands. By executing commands from the keyboard or from an ASCII file, the Exerciser provides an interactive way to issue commands to the motion controller boards of your system.

6.2.1. Invocation

If EXERCISE.EXE is on a hard disk, say in directory C:\MCP, enter the directory and invoke the exerciser by typing

```
C> cd \mcp
C> ex
```

The Exerciser will prompt you with a period "**.**" and then wait for your command.

6.2.2. Usage

The Exerciser is an interactive tool that allows commands to be issued to the MCP-04 motion controller boards. The AXIS command selects which global axis to which commands are sent, since only one axis can be interrogated at a time. The ECHO command determines the output display, in response to a user command, in either *long*, *short* or *off* format. Commands can easily be repeated by pressing the [F3] key and then the [CR] (carriage return). Any DOS executable may also be invoked without leaving the Exerciser via a temporary DOS escape.

Commands

Commands to the Exerciser can be issued directly from the keyboard under the Exerciser prompt ('.') or from a ASCII text file. Cases (upper or lower) are irrelevant. Namely, a command may be in all upper cases, all lower cases, or a combination of upper and lower cases.

A Exerciser command uses one of the following three formats:

(1) . ? <variable>
(2) . <variable> = <value>
(3) . <command> < parameter 1> <parameter 2> ...

The registers and I/O ports on MCP-04 boards and certain system parameters are referred by *variables* in the Exerciser. A variable represents a piece of storage whose value can be set and/or retrieved. For example, the digital filter parameter *gain* is a variable. Thus, it can be referenced by the first two command formats. Try the following example:

. **gain = 21.25**	set GAIN to 21.25
21.25	echo from Exerciser
. **?gain**	what is GAIN ?
21.25	echo from Exerciser
. **gain = 56.50**	set GAIN to 56.50
56.50	echo from Exerciser
. **?gain**	what is GAIN
56.50	echo from Exerciser

While commands in general are for setting or getting HCTL-1100 registers, other commands perform auxiliary functions and have unique formats. For example:

. **reset**	reset current axis (axis 1)
reset: (axis 1)	echo from Exerciser
. **echo long**	set echo format to long

Consult reference pages in Section 6.4 for more information on Exerciser commands.

Current Axis

An important notion in the Exerciser is the *current axis*, which is set to global axis 1 when the Exerciser gets started. The current axis can be determined at any time by command:

. **?axis**

To select another axis, use:

. **axis =** <global axis number>

Most operations in the Exerciser apply to the current axis, or the board that contains the current axis.

On-line Help

Details on each command (syntax and semantics) are explained in the reference pages as well as in an on-line help facility. To start the help facility inside the Exerciser, type either

. **help**

or

. **help** <topic>

The former is menu driven in which topics are grouped according to their functionality. The latter simply prints user selected <topic> to the screen.

Echo Mode

Three echo modes are provided: *off*, *short* and *long*. The default mode is *short*. *Off* turns off all echoes to the screen. *Short* echoes the command briefly (usually the value of a variable involved in the command). *Long* echoes detailed information about each command executed. The following is an example.

. **? gain**	echo is *short*
16.00	
. **echo off**	set echo to *off*
. **? gain**	
. **echo long**	set echo to *long*
. **? gain**	
gain (axis 2): 16.00	

The echo to a 'set' command (command of the form <variable> = <value>) is the actual setting in effect. For example:

. **gain = 23.341**	the resolution is 0.25
23.50	nearest possible setting
? gain	
23.50	

Recall that the specified value is always *rounded* to the nearest possible setting.

Command File

Command files, stored as an ASCII text containing a series of Exerciser commands may be *executed* inside the Exerciser by issuing the EXECUTE command.

. **execute** <command file>

An ASCII text file may also be executed repeatedly by optionally specifying the number of iterations after the file name.

. **execute** <command file> [<number of iterations>]

The effect of executing a command file is the same as if the commands in the file were typed manually from the keyboard. Caution must be taken so that each command move will have enough time to complete before the next command is issued. The WAIT command in the Exerciser can be employed for this specific purpose.

An Exerciser command file may be also be executed directly from the DOS shell by typing

x> **exercise** <command file>

The commands in <command file> will be interpreted by the Exerciser. The control returns to DOS once the end of the command file has been reached.

Run (DOS Escape)

DOS commands or any DOS executable program (.BAT, .COM and .EXE files), even the Exerciser itself, may be invoked inside the Exerciser by employing the RUN command. For example:

. **run dir a:**

lists the root directory of drive A.

'!' is a short hand for 'run'. Therefore,

. **! ex**

invokes the Exerciser within the Exerciser. Up to 10 parameters may be passed to the DOS command or the executable program.

Repeat (!!)

The last exerciser command issued can be repeated by typing *!!* at the Exerciser dot prompt. For example:

. **? pole**
-0.25000000
. **!!**
. **? pole**
-0.25000000

Note that the function key F3 has the same effect as *!!*.

Short Hand

Each Exerciser command has an equivalent 2-letter short hand version. This is very convenient if you issue a lot of commands through the Exerciser. For example, *ap* is used for ACT_POS and *cm* is used for command ENTER_CTL_MODE. See Section 6.2.3 for a complete listing of Exerciser commands and their corresponding short hands.

Monitor

The MCP-04 Exerciser contains a monitor, which when activated, displays on the screen critical parameters and activities of the system in real time. Experience has shown that it is an extremely useful debugging tool.

The monitor can be activated by typing at the Exerciser prompt

. monitor on

The display shown in Figure 6-1 will then appear at the top portion of your PC's screen.

```
                MCP-04 Motion Controller Monitor

Axis  Mode   Status  Com_pos  Act_pos  Error  Final_pos  Inputs Outputs
 1  CTL_MODE  C0H    334455   334453      2      900       04     00
 2    INIT    E0H       77  1111234 ++++++       0       00     00
 3   IV_CTL   C0H     1122     1119      3      600       50     00
```

```
. ? gain
gain (axis 2) = 15.25
.
```

Figure 6-1. Exerciser Monitor Screen Display

Each axis is shown in one row with the current axis in reverse video. For each MCP-04 board, input and output Ports A, B and C are shown along with in-board axes No. 1, No. 2 and No. 3, respectively.

For each axis, the following system information is displayed:

Axis	Global axis number
Status	Status register
Mode	Control mode
Com_pos	Command position register
Act_pos	Actual position register
Error	difference between command and actual positions
Final_pos	Final position register (for trapezoidal profile control)

If the actual error is greater than +32,767, a number of +'s will be shown on the Error column. If the actual error is less than -32,768, a number of -'s will be shown.

Exerciser commands can be issued as usual from the lower portion of the screen. The monitor can be deactivated (turned off) by command

. monitor off

6.2.3. List of Exerciser Commands

Control Mode Commands

? com_pos com_pos = (n)	[cp]	get/set *command position* -8,388,608 ≤ n ≤ 8,388,607
? act_pos act_pos = (n)	[ap]	get/set *actual position* returns n (counts)
? gain gain = (n)	[gn]	get/set *gain* of filter 0 ≤ n ≤ 63.75
? zero zero = (n)	[zr]	get/set *zero* of filter 0 ≤ n < 1
? pole pole = (n)	[pl]	get/set *pole* of filter -1 < n ≤ 0

sample_freq = (n)	[sf]	set *sample frequency* of filter in Hz
? accel accel = (n)	[ac]	get/set *command acceleration* $0 \leq n < 128$ (counts/sample time^2)
? max_vel max_vel = (n)	[mv]	get/set *maximum velocity* $0 \leq n \leq 127$ (counts/sample time)
? final_pos final_pos = (n)	[fp]	get/set *final position* $-8{,}388{,}608 \leq n \leq 8{,}388{,}607$ (counts)
? prop_vel prop_vel = (n)	[pv]	get/set *proportional velocity* $-2{,}048 \leq n < 2{,}048$ (counts/samp. time)
? int_vel int_vel = (n)	[iv]	get/set *integral velocity* $-128 \leq n \leq 127$ (counts/sample time)
? act_vel	[av]	get *actual velocity* (counts/samp. time)
?ctl_mode	[md]	get/set control mode
go_tp_ctl	[gt]	go *trapezoidal profile* control
go_pv_ctl	[gv]	go *proportional velocity* control
go_iv_ctl	[gi]	go *integral velocity* control
enter_ctl_mode	[cm]	enter default *position control*
reset	[rs]	performs *software reset*
init	[in]	put controller in *Initialize* mode

Utility Commands

? axis axis = (n)	[ax]	get current *axis* number select *axis* number
monitor (on/off)	[mo]	turn monitor on/off for board
echo (mode)	[ec]	set *echo* mode
execute (file) (n)	[ex]	*execute* commands in file n times
help	[hp]	on-line *help*
run (command)	[!]	temporary escape to *DOS*
wait (n)	[wt]	*wait* n x 55 msec
repeat	[!!]	*repeat* last command (also F3)
quit	[qt]	*return* to DOS

Configuration Commands

? status	[st]	display current *status*
config	[cf]	prompts for desired *configuration*
bipolar	[bp]	± command voltage format
unipolar	[up]	+ command voltage format
open_comm_loop	[ol]	commutator *open-loop* control
close_comm_loop	[cl]	commutator *closed-loop* control
check_ring	[cr]	verify *comm. ring* parameters
check_comm	[cc]	verify *commutator* constraints
? ring	[rg]	get/set commutator *ring*
ring = (n)		$0 \leq n \leq 127$
? x_reg	[xr]	get/set commutator *x register*
x_reg = (n)		$0 \leq n \leq 127$
? y_reg	[yr]	get/set commutator *y register*
y_reg = (n)		$0 \leq n \leq 127$
? offset	[of]	get/set commutator *offset*
offset = (n)		$-128 \leq n \leq 127$
? max_adv	[ma]	get/set *max. commutator*
max_adv = (n)		*phase advance* $0 \leq n \leq 127$
vel_timer = (n)	[vt]	set commutator *velocity timer* $0 \leq n \leq 255$

Miscellaneous Commands

? motor_com	[mc]	get/set 8-bit *motor command*
motor_com = (hex)		
? pwm_com	[pw]	get/set *PWM command* port duty cycle
pwm_com = (n)		$-100 \leq n \leq 100$
align	[al]	perform commutator *alignment*
clr_emergency	[ce]	clear *emergency flags*
home	[hm]	perform *homing* sequence
tune_filter	[tf]	tune filter experimentally
? port_a	[pa]	read from *Port A*
port_a = (hex)		write to *Port A* (8-bits)
? port_b	[pb]	read from *Port B*
port_b = (hex)		write to *Port B* (8-bits)

? port_c	[pc]	read from *Port C*
port_c = (hex)		write to *Port C* (4-bits)
? ext_encoder1	[e1]	read 16-bit *external encoder1*
ext_encoder1 = 0		clear *external encoder* count
? ext_encoder2	[e2]	read 16-bit *external encoder2*
ext_encoder2 = 0		clear *external encoder* count
? ext_encoder3	[e3]	read 16-bit *external encoder3*
ext_encoder3 = 0		clear *external encoder* count
? ext_encoder4	[e4]	read 16-bit *external encoder4*
ext_encoder4 = 0		clear *external encoder* count
ext_dac = (hex)	[da]	write to *external dac*
sync	[sy]	*synchronize* HCTL-1100s

6.3. MCP-04 PROGRAMMING INTERFACE LIBRARIES

6.3.1. Programming in C

The source code for MCP-04 C Interface Library is in MC.C on the distribution disk. It allows the user to develop his/her own application programs in C without worrying about the details of the MCP-04 boards.

MCP.C was written for the Microsoft Visual C++ compiler (Release 1.52). The code was developed and tested using the large memory model supported by this compiler. Early releases of Microsoft C compiler should also work. This compiler was selected since it is representative of commercially available C compilers. The C Library is used to construct the Exerciser and Check utility programs.

Programming Hints

Data Type Conventions:

velocity, acceleration	float (32 bits)
filter parameters	float (32 bits)
position	long (32 bits)
ports, status	unsigned char (8 bits)
offset, pwm_com	int (16 bits)
others	unsigned int (16 bits)

Static Variables and Functions Private to MC.C:

There are several static variables and functions in MC.C that are private to the module. Care must been taken to decompose or to extract functions from MC.C.

MCP.H:

File MCP.H on the distribution disk contains all necessary type declarations for functions defined in MCP.C. Therefore, it is recommended to include MCP.H in every module that calls functions in this library.

Return Values of 'Set' functions:

The return value of a 'set' function reflects its actual setting in effect, which may be different from the input value provided by the user as it may be out of valid range and/or subject to the rounding rules.

List of Functions

Control Mode functions

value = get_com_pos(axis) set_com_pos(axis,value)	get/set *command position* -8,388,608 ≤ value ≤ 8,388,607
value = get_act_pos(axis) set_act_pos(axis value) clr_act_pos(axis)	get/set/clear *actual position* -8,388,608 ≤ value ≤ 8,388,607
value = get_gain(axis) set_gain(axis, value)	get/set *gain* of filter 0 ≤ value ≤ 63.75
value = get_zero(axis) set_zero(axis, value)	get/set *zero* of filter 0 ≤ value < 1
value = get_pole(axis) set_pole(axis, value)	get /set *pole* of filter -1 < value ≤ 0
set_sample_freq(axis, value, md) value = get_sample_timer(axis)	set *sample frequency* of filter get *sample timer* value
value = get_accel(axis) set_accel(axis, value)	get/set *command acceleration* 0 ≤ value < 128 (cnts/sample time^2)
value = get_max_vel(axis) set_max_vel(axis, value)	get/set *maximum velocity* 0 ≤ value ≤ 127 (cnts/sample time)
value = get_final_pos(axis) set_final_pos(axis, value)	get/set *final position* -8388608 ≤ value ≤ 8388607 (counts)
value = get_prop_vel(axis) set_prop_vel(axis, value)	get/set *proportional velocity* -2048 ≤ value < 2048 (cnts/samp time)
value = get_int_vel(axis) set_int_vel(axis, value)	get/set *integral velocity* -128 ≤ value ≤ 127 (cnts/sample time)

value = get_act_vel(axis)	get *actual velocity*
value = get_ctl_mode(axis)	get *control mode*
go_tp_ctl(axis)	go *trapezoidal profile* control
go_pv_ctl(axis)	go *proportional velocity* control
go_iv_ctl(axis)	go *integral velocity* control
enter_ctl_mode(axis)	enter default *position control*
reset(axis)	software *reset*
init(axis)	put controller in *Initialize* mode

Configuration Functions

value = get_status(axis)	get current *status*
set_config(axis, value)	set configuration *value*
set_bipolar(axis)	± command voltage format
set_unipolar(axis)	+ command voltage format
open_comm_loop(axis)	commutator *open-loop* control
close_comm_loop(axis)	commutator *closed-loop* control
check_ring(axis, ring , x, y, np)	verify *comm. ring* parameters
check_comm(axis, ring, off, ma)	verify *commutator* constraints
value = get_ring(axis)	get/set commutator *ring*
set_ring(axis, value)	$0 \leq$ value ≤ 127
value = get_x_reg(axis)	get/set commutator *x register*
set_x_reg(axis, value)	$0 \leq$ value ≤ 127
value = get_y_reg(axis)	get/set commutator *y register*
set_y_reg(axis, value)	$0 \leq$ value ≤ 127
value = get_offset(axis)	get/set commutator *offset*
set_offset(axis, value)	$-128 \leq$ value ≤ 127
value = get_max_adv(axis)	get/set *max. commutator phase*
set_max_adv(axis, value)	*advance* $0 \leq$ value ≤ 127
set_vel_timer(axis, value)	set commutator *velocity timer* $0 \leq$ value ≤ 255
value = naxis()	*number of axis* in the system
value = nboard()	*number of boards* in the system
value = board_type(axis)	*type of board* containing axis
value = board_num(axis)	*board number* containing axis
value = axis_num(axis)	in-board axis number

Miscellaneous Functions

value = get_motor_com(axis) set_motor_com(axis, value)	get/set *motor command* 0 ≤ value ≤ FFH
value = get_pwm_com(axis) set_pwm_com(axis, value)	PWM command port duty cycle -100 ≤ value ≤ 100
align(axis)	perform commutator alignment
clr_emergency(axis)	*clear emergency* flags
home(axis, board, port, bit, iv, acc)	perform *homing* sequence
value = read_port_a(board) write_port_a(board, value)	read from *Port A* write to *Port A* (8-bits)
value = read_port_b(board) write_port_b(board, value)	read from *Port B* write to *Port B* (8-bits)
value = read_port_c(board) write_port_c(board, value)	read from *Port C* write to *Port C* (4-bits)
read_encoder(board, sel) clr_encoder (board)	read *sel external encoder* (16 bit integer) clear *external encoder* counts
sync(board)	*synchronize* HCTL-1100's on board
mc_init()	establish system communication

6.3.2. Using the Mektronix Win32 DLL in Windows 95, 98 or NT Programs

Included in this distribution is the MCP-04 motion control dynamic link library that has been compiled for Win32 operating systems; Windows 95/98/NT. The installation program, by default, installs this DLL into the Windows system directory. The user can install the DLL into any directory, but please make sure that any application using the DLL has access to it (by having the DLL in the same directory as the application, or in one of the directories in the PATH).

The Windows NT operating system requires an additional device driver to be installed. Included in this distribution is the device driver, and a batch file to install it. When the batch file is run, it will confirm that if the driver was installed correctly. Note, repeated installa-

tions will generate a warning that the device driver has already been installed. The user can confirm the installation in the Control Panel by running the Devices application.

6.4. EXERCISER AND PROGRAMMING INTERFACE LIBRARY REFERENCE

This section contains the manual pages for the MCP-04 Exerciser commands and high level language interface library routines. Entries are listed in alphabetical order.

For each entry, the description is given along with proper syntactic information on its invocation for the Exerciser, C and QuickBASIC. Note that some are Exerciser only commands and others can be invoked only through the language interface libraries. Note, the obsolete C syntax has been updated.

ACCEL

Purpose command acceleration

Description Get/set command acceleration of an axis for both Integral Velocity and Trapezoidal Profile control modes. ACCEL must be set before entering one of these control modes. It ranges from 0.0 to 127.99609375 with resolution of 1/256 quadrature counts per sample time squared. When set, the given value is rounded to the nearest possible setting.

Exerciser

? accel		(short hand)	? ac
accel = <value>			ac = <value>

C/C++

float get_accel(*axis*) unsigned int *axis*;
float set_accel(*axis*, *value*) unsigned int *axis*; float *value*;

C/C++

float get_accel(unsigned int *axis*);
float set_accel(unsigned int *axis*, float *value*);

Return the actual command acceleration in effect.

See Also GO_IV_CTL, GO_TP_CTL

ACT_POS

Purpose actual position in all control modes

Description Get/set/clear the 24-bit actual position register for an axis . The ACT_POS register may be read while in any control mode. However, setting/clearing the register should only be done while in Initialize mode. The range of actual position for an axis is from -8,388,608 to 8,388,607 (quadrature counts). Limit settings will be used if the given value is out of this range.

Exerciser

? act_pos	(short hand)	? ap
act_pos = <value>		ap = <value>

C

long get_act_pos(*axis*) unsigned int *axis*;
long set_act_pos(*axis*, *value*) unsigned int *axis*; long *value*;
long clr_act_pos(*axis*) unsigned int *axis*;

Return actual position.

Caution Clearing ACT_POS will cause the motor to increment to the current command position while in Position Control mode. It is recommended to enter Initialize mode before clearing the ACT_POS register.

ACT_VEL

Purpose actual velocity in Proportional Velocity Control mode

Description Get actual velocity of an axis while in Proportional Velocity Control mode. The range for actual velocity is from -32768 to 32767 quadrature counts per sample time. The returned value does not contain a fractional part.

Exerciser ? act_vel (short hand) ? av

C float get_act_vel(*axis*) unsigned int *axis*;

Return the actual velocity of *axis*.

Caution The reading of act_vel is only valid while the axis is in Proportional Velocity Control mode.

See Also PROP_VEL, GO_PV_CTL

ALIGN

Purpose enter Align mode

Description The Align command puts the axis in Initialize mode, clears the actual position and sets the command position to zero before entering Align mode. This mode automatically aligns multi-phase brushless motors to the commutator automatically. Consult Section 4.3 for more information regarding Align mode. After completion, the selected axis enters Position Control mode.

Exerciser align (short hand) al

C align(*axis*) unsigned int *axis*;

Caution Align must be executed while *axis* is in Initialize mode.

See Also INIT, ACT_POS, COM_POS, CONFIG

AXIS

Purpose current axis

Description Display or select the current global axis number, which is determined by the order in which the board information is entered during system setup. Each MCP-04 occupies three or four axis numbers. The example given below shows how two boards may be configured.

Board No.	Board Type	Global Axis No.
1	MCP-03	1,2,3
2	MCP-04	4,5,6,7

Exerciser

? axis		(short hand)	? ax
axis = <value>			ax = <value>

Caution The diagnostics given for selecting an out of range axis number is solely based on the information in the DOS environment variable MCINIT, which must be changed by hand.

See Also NBOARD, NAXIS, AXIS_NUM, BOARD_NUM

AXIS_NUM

Purpose determine in-board axis number

Description Returns the in-board axis number for the given global axis number. For an axis on MCP-04, the return can be 1, 2, 3 or 4.

C unsigned int axis_num(*axis*) unsigned int *axis*;

Return in-board axis number

Caution the return depends on the DOS environment variable MCINIT. If it is not set right, the return is not predictable.

See Also BOARD_TYPE, NBOARD

BIPOLAR

Purpose set motor command output to bipolar mode

Description Set motor command output of the given axis to bipolar mode. This mode provides four quadrant motor control. This is the default mode when first powering up the MCP-04 boards.

Exerciser bipolar (short hand) bp

C set_bipolar(*axis*) unsigned int *axis*;

See Also MOTOR_COM, UNIPOLAR

BOARD_NUM

Purpose determine board number

Description Returns the board number containing the specified axis. The axis is given in global axis number.

C unsigned int board_num(*axis*) unsigned int *axis*;

Return board number

Caution the return depends on the DOS environment variable MCINIT. If it is not set right, the return is not predictable.

See Also BOARD_TYPE, NBOARD

BOARD_TYPE

Purpose determine board type

Description Returns 3 if the given axis is on a MCP-03 board, or 4 if the given axis is on a MCP-04 board.

C unsigned int board_type(*axis*) unsigned int *axis*;

Return board type.

Caution the return depends on the DOS environment variable MCINIT. If it is not set right, the return is not predictable.

See Also BOARD_NUM, NBOARD

CHECK_COMM

Purpose check commutator configuration

Description Read RING, OFFSET, and MAX_ADV from the board of the given axis and check whether the following equation holds:

$$-128 \leq 1.5 \cdot RING + OFFSET \pm MAX\_ADV \leq 127$$

In interface libraries, RING, OFFSET, and MAX_ADV are 'out' parameters; the corresponding values are retrieved.

Exerciser check_comm (short hand) cc

C int check_comm(*axis, ring, offset, max_adv*)
unsigned int *axis, *ring, *max_adv;* int **offset*;

Return true (non-zero) if checked OK, false (zero) otherwise.

See Also RING, OFFSET, MAX_ADV

CHECK_RING

Purpose check commutator ring configuration

Description Read current settings of RING, X_REG, Y_REG, and NPHASE (number of phases) of an axis. Check whether the following equation holds:

RING = (X_REG + Y_REG) · NPHASE

In interface libraries, RING, X_REG, Y_REG and NPHASE are 'out' parameters; the corresponding values are retrieved.

Exerciser check_ring (short hand) cr

C int check_ring(*axis, ring, x_reg, y_reg, nphase)*
unsigned int *axis*, **ring*, **x_reg*, **y_reg*, **nphase*;

Return TRUE (non zero) if checked OK, FALSE (zero) otherwise.

See Also RING, X_REG, Y_REG, STATUS, CONFIG

CLOSE_COMM_LOOP

Purpose commutator closed-loop control

Description Enable the internal commutator counters of the given axis to allow closed-loop control using the commutator. This is the normal setting for commutating brushless motors.

Exercise close_comm_loop (short hand) cl

C close_comm_loop(*axis*) unsigned int *axis*;

See Also OPEN_COMM_LOOP

CLR_EMERGENCY

Purpose clear STOP and/or LIMIT flags

Description Clear STOP and/or LIMIT flags set by external emergency opto-coupler trigger inputs for the given axis.

Exerciser clr_emergency (short hand) ce

C clr_emergency(*axis*) unsigned int *axis*;

See Also STATUS, CONFIG

COM_POS

Purpose command position in Position Control mode

Description Get/set command position in Position Control mode for the given axis. The 24-bit COM_POS register specifies the desired position. While in Position Control, changing the COM_POS will cause the motor to immediately move to the new commanded position without profiling. The range of command position for an axis is from -8,388,607 to 8,388,607 (quadrature counts). Limit settings will be used if the given value is out of this range.

Exerciser

? com_pos	(short hand)	? cm
com_pos = <value>		cm = <value>

C

long get_com_pos(*axis*) unsigned int *axis*;
long set_com_pos(*axis, value*)
unsigned int *axis*; long *value*;

Return the real command position setting in effect.

See Also ACT_POS, ENTER_CTL_MODE

CONFIG

Purpose configure PWM sign reversal, commutator phase and commutator count format.

Description In Exerciser, user is prompted to answer a number of questions on PWM sign reversal, number of commutator phases and commutator count format. The interface library requires an encoded integer representing the same information. The encoding is specified as follows:

bit 0:	0 = PWM sign reversal off 1 = PWM sign reversal on
bit 1:	0 = commutator 3 phase 1 = commutator 4 phase
bit 2:	0 = commutator count quad. 1 = commutator count full
bit 3:	always 0

The settings can be read back, along with other information, with command STATUS. The consistency of commutator settings may be checked by calling CHECK_RING.

Exerciser config (short hand) cf

C unsigned char set_config(*axis, value*)
unsigned int *axis*; unsigned char *value*;

Return encoding of the actual configuration in effect.

See Also STATUS, RING, X_REG, Y_REG, CHECK_RING

CTL_MODE

Purpose get current control mode

Description Returns the current control mode for the specified axis. The following encoding is used:

0:	INVALID	invalid control mode
1:	INIT	initialization/idle mode
2:	CTL_MODE	position control
3:	PV_CTL	proportional velocity control
4:	IV_CTL	integral velocity control
5:	TP_CTL	trapezoidal profile control

The control mode for an axis is derived from the reading from the FLAG register of HCTL-1100.

Exerciser ? ctl_mode short hand: ? md

If ECHO is set to *short*, the encoded control mode will be displayed. If ECHO is set to *long*, English description will be given.

C unsigned int get_ctl_mode(*axis*) unsigned int *axis*;

Return encoding of control mode.

See Also INIT, RESET, GO_IV_CTL, GO_TP_CTL, GO_PV_CTL, ENTER_CTL_MODE

ECHO

Purpose set echo mode for the Exerciser

Description The echo mode to the screen while in the Exerciser may be either *off*, *short* or *long*. *Off* completely suppresses any echo whereas *short* and *long* allow command responses to be displayed on the screen. *Short* echoes the command briefly (usually the value of a variable involved in the command). *Long* echoes detailed information about each command executed. The default echo mode is *short*.

Exerciser

echo off	(short hand)	ec off
echo short		ec short
echo long		ec long

Caution Echo mode is a global concept in the Exerciser. Altering it in a command file may affect the echo format upon returning to its caller.

Example

. ? gain	echo is *short*
16.00	
. echo off	set echo to *off*
. ? gain	
. echo long	set echo to *long*
. ? gain	
gain (axis 1): 16.00	

ENTER_CTL_MODE

Purpose enter Position Control mode

Description Enter Position Control mode for the given axis. COM_POS is also set to the current actual position which will cause the axis to hold the current position until a new command position is given or another mode is entered. All other control modes must be entered from Position Control mode.

Exerciser enter_ctl_mode (short hand) cm

C enter_ctl_mode(*axis*) unsigned int *axis*;

Caution GO-TP_CTL, GO_PV_CTL and GO_IV_CTL are not effective unless the axis is currently in Position Control mode.

See Also COM_POS, ACT_POS, CTL_MODE

EXECUTE

Purpose execute a command file while in the Exerciser

Description Exerciser commands stored in a text file may be executed as if they were typed from the keyboard in the Exerciser. The execution of command files may be nested and/or recursive. The optional <iteration> specifies the number of iterations the file will be executed. If omitted, the commands in <command file> are executed only once.

Exerciser execute <command file> [<iteration>]
(short hand) ex <command file> [<iteration>]

EXT_DAC

Purpose set external digital to analog converter

Description Set external 8-bit digital to analog converter (DAC) to a hexadecimal value on a MCP-04 board. Depending on how the converter is configured, (either 0-10 V or ±10 V operation) the value sent to the converter represents its output range. For example, a value of 80H will output 5 volts for 0-10 V operation or 0 volts for ±10 V operation.

Exerciser ext_dac = <value> (short hand) da = <value>

where <value> is given in hex (00 to FF)

C
```
unsigned char write_ext_dac(board, value)
unsigned int board; unsigned char value;
```

Return the setting in effect.

ENCODER

Purpose read/clear external position encoder

Description Read or clear one of four selected external encoders connected to a MCP-04 board. The four extenal encoders are multiplexed to one 16-bit incremental encoder decoder circuit. Writing to an external encoder is equivalent to using the clr_encoder() function.

Exerciser

? ext_encoder1	(short hand)	? e1, e2, e3, e4
encoder1 = <value>		e1 = 0

C

int read_encoder1(*board, sel*) int *board, int sel*;
int clr_encoder(*board*) int *board*;

Return 16 bit signed external encoder value.

FINAL_POS

Purpose final position for Trapezoidal Profile Control mode

Description Get/set final position for Trapezoidal Profile Control mode for the given axis. FINAL_POS must be set before issuing the GO_TP_CTL command. The range of its value is from -8,388,608 to 8,388,607 (quadrature counts). If out of range, one of the limit settings will apply.

Exerciser

? final_pos	(short hand)	? fp
final_pos = <value>		fp = <value>

C

long get_final_pos(*axis*) unsigned int *axis*;
long set_final_pos(*axis, value*)
unsigned int *axis*; long *value*;

Return the actual FINAL_POS setting in effect.

See Also GO_TP_CTL, ACCEL, MAX_VEL

GAIN

Purpose filter parameter gain

Description Get/set filter parameter gain for *axis* which may range from 0.0 to 63.75 with resolution 0.25. GAIN is related to the gain register, K, of the HCTL-1100 by the equation: GAIN = K/4. *Value* is rounded to the nearest possible setting when it is set. The filter parameter gain is utilized in all control modes.

Exercise

? gain		(short hand)	? gn
gain = <value>			gn = <value>

C

float get_gain(*axis*) unsigned int *axis*;
float set_gain(*axis, value*) unsigned int *axis*; float *value*;

Return the actual gain value in effect.

See also POLE, ZERO, SAMPLE_FREQ

GO_IV_CTL

Purpose enter Integral Velocity Control mode

Description Enter Integral Velocity Control mode for the axis. In Integral Velocity Control mode, the system is actually a position control system and therefore the complete dynamic compensation filter is used. The axis starts to move according to pre-commanded INT_VEL and ACCEL. The external STOP signal (indicated by Bit 6 of the STATUS register) is effective only in this mode and causes the motor to perform a controlled stop by decelerating at the specified command acceleration. This flag may be cleared by using CLR_EMERGENCY once the Stop trigger is removed.

Exerciser go_iv_ctl (short hand) gi

C go_iv_ctl(*axis*) unsigned int *axis*;

See Also INT_VEL, ACCEL, STATUS, CLR_EMERGENCY

GO_PV_CTL

Purpose enter Proportional Velocity Control mode

Description Enter Proportional Velocity Control mode for the given axis. In Proportional Velocity Control, only the GAIN is used for compensation (POLE and ZERO are not used). The axis moves according to current PROP_VEL setting. The actual velocity may be read back by command ACT_VEL.

Exerciser go_pv_ctl (short hand) gv

C go_pv_ctl(*axis*) unsigned int *axis*;

See Also PROP_VEL, ACT_VEL

GO_TP_CTL

Purpose enter Trapezoidal Profile Control mode

Description Enter Trapezoidal Profile Control mode for the specified axis. The internal profile generator produces a position profile using the present Command Position as the starting point and the Final Position as the end point according to the preset ACCEL and MAX_VEL. While an axis is profiling, no register of that controller may be set. However, the registers may still be read during Trapezoidal Profile Control.

Exerciser go_tp_ctl (short hand) gt

C go_tp_ctl(*axis*) unsigned int *axis*;

Caution The execution of certain commands may not be effective if they are issued while the axis is profiling in Trapezoidal Profile Control mode.

See also FINAL_POS, MAX_VEL, ACCEL, STATUS

HELP

Purpose Exerciser on-line help facility

Description This is the Exerciser on-line help facility. If a topic name is provided, it shows the manual page of that topic. Otherwise, a series of carefully designed menus lead to the manual pages one wants to see.

Exerciser

help		(short hand)	hp
help <topic>			hp <topic>

Caution The file MCP.HLP must be present in the current directory in order to get on-line help while using the Exerciser.

HOME

Purpose perform homing sequence

Description The function uses the specified *port* and *bit* on the given *board* to direct the homing sequence of the given axis. The homing sequence consists of two stages. In the first stage, the axis moves in the direction specified by the control bit (0 = positive and 1 = negative) with given integral velocity *ivel* and acceleration *accel*. The first stage ends as soon as the control bit changes state. In the second stage, the axis moves in the opposite direction with one fifth of the given *ivel* and stops when the control bits changes again. At the end of this homing sequence, the ACT_POS register is cleared and the axis is in Position Control mode. The following explains each parameter in detail:

axis: the current axis is used for the Exerciser. Otherwise, it is the global axis number.

board: board number that contains the port used in homing.

port: port identification on board specified. In the Exerciser, it can be either a, b or c. In the language interface libraries, use 1 or Port A, 2 for Port B and 3 for Port C.

bit: The bit number of the port to be employed in the homing sequence.

ivel: initial integral velocity used in the first stage of the homing process. If omitted in the Exerciser, the default is 5 cnts/sample time.

accel: acceleration used in the homing process. If omitted in the Exerciser, the default is 0.01 cnts/sample time^2.

Exerciser home <board> <port> <bit> [<ivel> <accel>]
(short hand) hm <board> <port> <bit> [<ivel> <accel>]

C home(*axis, board, port, bit, ivel, accel*)
unsigned int *axis*, *board*, *port*, *bit*; float *ivel, accel;*

INIT

Purpose enter Initialize mode

Description Enter Initialize mode for the axis. The following conditions occur:

- bit 5 of STATUS register is set to 1.
- PWM_COM is set to 0 (zero duty cycle).
- MOTOR_COM is set to 80H (0 volts).
- previously sampled data in the digital filter is cleared.

Exerciser init (short hand) in

C init (*axis*) unsigned int *axis*;

See Also RESET, STATUS

INT_VEL

Purpose Command velocity in Integral Velocity Mode.

Description Get/set command velocity in Integral Velocity Control mode for the axis. INT_VEL must be set before issuing the GO_IV_CTL command. The range of *value* is from -128 to 127 with a resolution of one quadrature count per sample time. When set, *value* is rounded to the nearest possible setting.

Exerciser

? int_vel	(short hand)	? iv
int_vel = <value>		iv = <value>

C float get_int_vel(*axis*) unsigned int *axis*;
float set_int_vel(*axis, value*) unsigned int *axis*; float *value*;

Return the actual command velocity in effect.

Caution The maximum step change in *value* must not exceed ±127 decimal.

See Also GO_IV_CTL, ACCEL

MAX_ADV

Purpose commutator maximum phase advance

Description Get/set commutator maximum phase advance of the given axis, which ranges from 0 to 127. A value greater than 127 will be limited. Consult Section 4.3 for other constraints regarding the commutator maximum phase advance register.

Exerciser

? max_adv	(short hand)	? ma
max_adv = <value>		ma = <value>

C

unsigned int get_max_adv(*axis*) unsigned int *axis*;
unsigned int set_max_adv(*axis, value*)
unsigned int *axis*, *value*;

Return the effective maximum velocity.

See Also CHECK_COMM, VEL_TIMER, RING

MAX_VEL

Purpose maximum velocity in Trapezoidal Profile Control mode

Description Get/set maximum velocity in Trapezoidal Profile Control mode for the given axis. MAX_VEL must be set before issuing the GO_TP_CTL command. The range of *value* is from 0 to 127 with a resolution of one quadrature count per sample time. When set, *value* is rounded to the nearest possible setting.

Exerciser

? max_vel		(short hand)	? mv
max_vel = <value>			mv = <value>

C

float get_max_vel(*axis*) unsigned int *axis*;
float set_max_vel(*axis, value*) unsigned int *axis*; float *value*;

Return the actual maximum velocity setting in effect.

See Also GO_TP_CTL, ACCEL, FINAL_POS

MC_INIT

Purpose establish communication between PC and MCP-04 boards

Description **MCINIT must be called before any other functions in the interface libraries.** It looks for DOS environment variable MCINIT and uses it to build a table that translates global axis numbers to their base addresses. This table will be used by the successive calls that refer axes by their global axis numbers.

The required DOS environment variable MCINIT may be established by running MCINIT.BAT which is generated by the Setup program (SETUP.EXE) on the distribution disk. If the environment variable can not be found, MC_INIT assumes that there is only one MCP-04 board whose starting address is 3E0H.

C int mc_init()

Return In C, it returns the number of axes used in the system, or -1 if an error occurred.

Caution The result of calling a routine in any interface library is unpredictable if MC_INIT is omitted.

MONITOR

Purpose monitor system status on screen while using Exerciser

Description Monitor displays critical system status and parameters on the screen while still allowing the user to issue Exerciser commands. It displays control mode, status, command and actual positions, final position, and tracking error for each axis in the system. It also displays I/O port activities for each MCP-04 board if the ports are configured. See Section 6.2.2.9. for a detailed description.

Exerciser	monitor on	(short hand)	mo on
	monitor off		mo off

MOTOR_COM

Purpose motor command register

Description Get/set motor command register for the given axis, which ranges from 00H to FFH. A *Value* greater than FFH is limited to FFH. Setting MOTOR_COM is only effective while in Initialize mode. The 8-bit motor command is output to a digital to analog converter whose range is from -10 V (00H) to +10 V (FFH). The default setting is 80H (0V) when entering Initialize mode.

Exerciser

? motor_com	(short hand)	? mc
motor_com = <value>		mc = <value>

C

unsigned int get_motor_com(*axis*)
unsigned int *axis*;

unsigned int set_motor_com(*axis, value*)
unsigned int *axis*, *value*;

Return the motor command register value in effect.

Caution No warning is given for a setting while the current axis is not in initialize mode, even though the setting has no effect.

See Also ENTER_CTL_MODE, INIT

NAXIS

Purpose	total number of axes
Description	returns the total number of axes in the system
C	unsigned int naxis()
Return	the total number of axes in the system.
Caution	the return depends on the DOS environment variable MCINIT. If it is not set right, the return is not predictable.
See Also	NBOARD, BOARD_TYPE, BOARD_NUM.

NBOARD

Purpose	total number of boards in the system
Description	returns the total number of MCP-04 boards in the system
C	unsigned int nboard()
Return	the total number of MCP-04 boards.
Caution	the return depends on the DOS environment variable MCINIT. If it is not set right, the return is not predictable.
See Also	NAXIS, BOARD_TYPE, BOARD_NUM.

OFFSET

Purpose commutator offset register

Description Get/set commutator offset of the given axis, which ranges from -128 to 127. A *value* out of this range will be limited. Consult Section 4.3 for other constraints regarding the commutator offset register.

Exerciser

? offset		(short hand)	? os
offset = <value>			os = <value>

C

int get_offset(*axis*) unsigned int axis;
int set_offset(*axis, value*) unsigned int *axis*; int *value*;

See Also CHECK_COMM, RING, MAX_ADV, VEL_TIMER

OPEN_COMM_LOOP

Purpose commutator open-loop control

Description Inhibit the internal commutator counters of the given axis to allow open-loop stepping of a motor using the commutator. This feature assists in setting up the alignment of the position encoder to the motor phases. This configuration is not to be used for running stepper motors in open-loop control.

Exerciser open_comm_loop (short hand) ol

C open_comm_loop(*axis*) unsigned int *axis*;

See Also CLOSE_COMM_LOOP, ALIGN

POLE

Purpose filter parameter Pole

Description Get/set filter parameter POLE for the given axis. It ranges from -0.9906935 to 0.0 with resolution of 1/256. In control terms, this places the POLE within the unit circle of the z-plane. When set, -|*value*| is rounded to the nearest possible setting. The filter parameter pole is utilized in all control modes except Proportional Velocity Control mode. POLE is related to the pole register B by the equation: POLE = B/256.

Exerciser

? pole	(short hand)	? pl
pole = < value>		pl = <value>

C

float get_pole(*axis*) unsigned int *axis*;
float set_pole(*axis, value*) unsigned int *axis*; float *value*;

Return the actual pole value in effect.

Caution If 0 < *value* < 1, then -*value* will be used.

See also GAIN, ZERO, SAMPLE_FREQ

PORT_A

Purpose 8-bit user configurable I/O Port A

Description Read from/write into Port A of the given board. Reading/writing values may range from 00H to FFH. When Port A is connected to an external encoder, it must be configured as an input port. In this case, the position can be read from external encoder by calling EXT_ENCODER. Port A is accessible at connector J-3 on a MCP-04 board after configuring as either on input or output port.

Exerciser

? port_a	(short hand)	? pa
port_a = <value>		pa = <value>

C

unsigned char read_port_a(*board*)
unsigned int board;
unsigned char write_port_a(*board, value*)
unsigned int board; unsigned char value;

Return the value retrieved from the port.

Caution Read/write values areeffective only if Port A is configured currently as input or output port, respectively, using the CONFIG_PORTS command. PORT_A is not applicable to MC-01 boards.

See Also EXT_ENCODER

PORT_B

Purpose 8-bit user configurable I/O Port B

Description Read from/write into Port B of *board* . Read/write values may range from 00H to FFH. Port B is accessible at connector J-3 on a MCP-04 board after configuring as either on input or output port.

Exerciser

? port_b		(short hand)	? pb
port_b = <value>			pb = <value>

C

unsigned char read_port_b(*board*)
unsigned int *board*;
unsigned char write_port_b(*board, value*)
unsigned int *board*; unsigned char *value*;

Return the value retrieved from the port.

PORT_C

Purpose user configurable lower 4 bits of Port C

Description Read from/write into 4-bit Port C on the given board. Read/write values may range from 0H to FH. Port C is accessible at connector J-3 on a MCP-04 board after configuring as either on input or output port.

Exerciser

? port_c	(short hand)	? pc
port_c = <value>		pc = <value>

C

unsigned char read_port_c(*board*)
unsigned int *board*;
unsigned char write_port_c(*board, value*)
unsigned int *board*; unsigned char *value*;

Return the effective value of the lower 4 bits of Port C.

PROP_VEL

Purpose command velocity in Proportional Velocity Control mode

Description Get/set command velocity in Proportional Velocity Control mode for the given axis. The PROP_VEL should be set before issuing the GO_PV_CTL command. The range of *value* is from -2048.0 to 2047.9375 with a resolution of 1/16 quadrature counts per sample time. When set, the given value is rounded to the nearest possible setting.

Exerciser

? prop_vel	(short hand)	? pv
prop_vel = <value>		pv = <value>

C

float get_prop_vel(*axis*) unsigned int *axis*;
float set_prop_vel(*axis, value*)
unsigned int *axis*; float *value*;

Return the actual command velocity in effect.

See Also GO_PV_CTL, ACT_VEL

PWM_COM

Purpose PWM duty cycle command register

Description Get/set PWM duty cycle command register of an axis, which effects the PWM Pulse and Direction outputs while in Initialize mode. The range is limited to ±100 with resolution of 1. For example, a command of 50 corresponds to a 50% drive signal in the positive direction. The modulation frequency of the PWM Pulse signal is fixed at either 20 kHz or 10 kHz with a board clock setting of 2 MHz or 1 MHz respectively.

Exerciser

? pwm_com	(short hand)	? pw
pwm_com = <value>		pw = <value>

C

int get_pwm_com(*axis*) unsigned int *axis*;
int set_pwm_com(*axis, value*)
unsigned int *axis*; int *value*;

Return the value of PWM command register in effect.

Caution No warning is given for a setting while the current axis is not in initialize mode, even though the setting has no effect.

See Also INIT

QUIT

Purpose quit Exerciser to DOS

Description QUIT stops execution of the command file being interpreted or exits to DOS if at the Exerciser program level. All setting on the MCP-04 boards will remain in effect unless power is turned off.

Exerciser quit (short hand) qt

REPEAT

Purpose repeats the last executed Exerciser command

Description !! repeats the execution of the last command issued in the Exerciser. The same effect may be achieved by using the function key [F3]. The last command is local to the command file being interpreted. For example,

```
. execute  move.cmd
.
( execution of commands in move.cmd)
.
. !!
execute  move.cmd
```

Note that !! did not repeat the last command in move.cmd.

Exerciser repeat (short hand) !!
[F3]

RESET

Purpose software reset

Description Execute a software reset which performs the following on the given axis:

- digital filter parameters are preset to GAIN = 16.00, ZERO = 0.89453125, POLE = -0.25.
- Sample frequency is set to 1,923.0 Hz if CLOCK_FREQ is 2 MHz or 961.5 Hz if CLOCK_FREQ is 1 MHz.
- status register is cleared.
- actual position is cleared to zero.
- enter Initialize mode.

Exerciser reset (short hand) rs

C reset (*axis*) unsigned int *axis*;

See Also INIT

RING

Purpose commutator ring

Description Get/set commutator ring register for an axis, which ranges from 0 to 127. A value greater than 127 will be limited. Consult Section 4.3 for other constraints regarding the commutator Ring register.

Exerciser

? ring		(short hand)	? rg
ring = <value>			rg = <value>

C

unsigned int get_ring(*axis*) unsigned int *axis*;
unsigned int set_ring(*axis, value*)
unsigned int *axis*, *value*;

See Also CHECK_RING, CHECK_COMM, X_REG, Y_REG

RUN

Purpose issuing DOS command in Exerciser

Description Temporarily escape to DOS and issue a DOS command in Exerciser. The given DOS command will be executed as if it were invoked directly under the DOS shell. Up to 10 parameters can be passed to the program through the command line.

Exerciser run <DOS command> (short hand) ! <DOS command>

Caution Be careful with the DOS command you chose to issue.

SAMPLE_FREQ

Purpose sampling frequency

Description Set sampling frequency for the given axis. User specified sample frequency should be given according to the following table, where mode should be 1 if integral velocity control or trapezoidal profile control is used for the axis or 0 otherwise.

clock freq	mode	min	max
1 MHz	0	244.125	7,812.50
1 MHz	1	244.125	3,906.25
2 MHz	0	488.250	7,812.50
2 MHz	1	488.250	15,625.00

Given sample frequency (in Hz) is rounded to the nearest possible setting. Sample frequency (SF) is related to the sample timer register (T) by the following equation:

SF = 62500 * CF / (T+1)

where CF is 1 for 1 MHZ clock or 2 for 2 MHZ clock.

Exerciser sample_freq = <value> (short hand) sf = <value>

C float set_sample_freq(*axis*, *value, mode*)
unsigned int *axis*, *mode*; float *value*;

Return the actual sampling frequency in effect.

Caution No warning is given for *value* higher than 7812.5 Hz in Trapezoidal Profile Control or Integral Velocity Control modes.

See Also GAIN, ZERO, POLE, SAMPLE_TIMER

SAMPLE_TIMER

Purpose sample timer register

Description Read the current value of the sample timer register, which is a downward counter. The counter's maximal value is determined by setting sample frequency (see SAMPLE_FREQ). The reading of the sample timer register may fall anywhere between this maximal value and 0. Application routines written in C and BASIC may use this command to access this register to perform certain operations, such as setting command position or synchronization between axes during the same sample period. The maximal value (T), for the sample timer register is related to sample frequency (SF) by

$$T = 62500 \left(\frac{CF}{SF} - 0.5 \right)$$

where CF is 1 for 1 MHZ clock; 2 for 2 MHZ clock.

C float get_sample_timer(*axis*)
unsigned int *axis*,

Return current value of the sample timer register

See Also SAMPLE_FREQ

STATUS

Purpose status register

Description Get current value of the status register of an axis. The lower four bits may be set by command CONFIG.

Exercise ? status (short hand) ? st

C unsigned char get_status(*axis*) unsigned int *axis*;

Return The return value is given in the following format:

bit 0:	0 = PWM sign reversal off 1 = PWM sign reversal on
bit 1:	0 = commutator 3 phase 1 = commutator 4 phase
bit 2:	0 = commutator count quad. 1 = commutator count full
bit 3:	always 0
bit 4:	0 = not in tp control 1 = in tp control
bit 5:	0 = not in initialize mode 1 = in initialize mode
bit 6:	0 = stop triggered 1 = stop not triggered
bit 7:	0 = limit triggered 1 = limit not triggered

See Also CONFIG, CLR_EMERGENCY

SYNC

Purpose provides a pulse for synchronizing sample timers on multiple axes

Description Generate a negative strobe on the sync pin of HCTL-1100s on a MCP-04 board. The synchronization among axes is useful only if involved axes are set to the same sample frequency.

SYNC is effective only if the axes are in Initialization mode.

Exerciser sync (short hand) sy

C sync(*board*)

Caution The user must first configure the I/O ports by using the CONFIG_PORTS command.

See Also CONFIG_PORTS

TUNE_FILTER

Purpose tune filter parameters experimentally

Description This Exerciser command tunes filter parameters (GAIN, POLE, ZERO and SAMPLE_FREQ) of the current axis experimentally. The command uses position control to move the axis forward and backward between -<incr> and +<incr>, where <incr> may be optionally specified or be default to 64, until [F0] is hit. The filter parameters may be changed "on the fly" using function keys [F1]-[F8].

If the current axis is on a MCP-04 board, the actual position is converted via external dac to its Monitor port so that the step response can be observed with an oscilloscope at TP4 If the current axis is on a MC-01 board and a MCP-04 is also present at your system, the EXT_DAC of the first MCP-04 board will be used for this purpose.

Exerciser tune_filter [<incr>] (short hand) tf [<inc>]

See Also CONFIG_PORTS, GAIN, POLE, ZERO, SAMPLE_FREQ

UNIPOLAR

Purpose set motor command output to unipolar mode

Description Set motor command output of an axis into unipolar mode. This mode provides only positive voltage command signals (two quadrant motor control).

Exerciser unipolar (short hand) up

C set_unipolar(*axis*) unsigned int *axis*;

Caution Putting an axis in unipolar mode may cause the motor to "run away" if your system is expecting a bipolar voltage command output.

See Also MOTOR_COM, BIPOLAR

VEL_TIMER

Purpose commutator velocity timer

Description Set commutator velocity timer for an axis to a value in the range from 0 to 255. A given value greater than 255 is limited. The data specifies the amount of phase advance at a given velocity. Consult Section 4.3 for more information regarding the VEL_TIMER register.

Exerciser vel_timer = <value> (short hand) vt = <value>

C unsigned int set_vel_timer(*axis,value*)
unsigned int *axis*, *value*;

See Also MAX_ADV, OFFSET, Y_REG, X_REG, RING

WAIT

Purpose wait for the specified time in Exerciser

Description Wait for the specified time in PC's clock 'ticks'. If no argument is provided, suspension is indefinite until any key is hit. Otherwise, the Exerciser prompt comes back in specified 'ticks'; each tick is about 55 ms.

Exerciser wait [<ticks>] (short hand) wt [<ticks>]

X_REG

Purpose commutator X register

Description Get/set commutator X register for an axis to a value in the range from 0 to 127. A given value greater than 127 will be limited. Consult Section 4.3 for other constraints regarding the X register.

Exerciser

? x_reg	(short hand)	? xr
x_reg = <value>		xr = <value>

C

unsigned char get_x_reg(*axis*) unsigned int *axis*;
unsigned int set_x_reg(*axis, value*)
unsigned int *axis*, *value*;

Return the effective X register value.

See Also CHECK_RING, Y_REG, CONFIG

Y_REG

Purpose commutator Y-phase register

Description Get/set commutator Y-phase register for an axis to a value in the range from 0 to 127. A given value greater than 127 is limited. Consult Section 4.3 for other constraints regarding the commutator Y-phase register.

Exerciser

? y_reg	(short hand)	? yr
y_reg = <value>		yr = <value>

C

unsigned int get_y_reg(*axis*) unsigned int *axis*;
unsigned int set_y_reg(*axis, value*)
unsigned int *axis*, *value*;

Return effective Y-phase register value.

See Also CHECK_RING, X_REG, CONFIG

ZERO

Purpose filter parameter zero

Description Get/set filter parameter Zero for an axis. It ranges from 0.0 to 0.99069375 with resolution of 1/256. In control terms, this places the filter zero within the unit circle of the z-plane. When set, *value* is rounded to the nearest possible setting. The filter parameter ZERO is utilized in all control modes except Proportional Velocity Control mode.

Exerciser

? zero	(short hand)	? zr
zero = <*value*>		zr = <value>

C

float get_zero(*axis*) unsigned int *axis*;
float set_zero(*axis, value*) unsigned int *axis*; float *value*;

Return the actual zero value in effect.

Caution If -1 < *value* < 0, then -*value* will be used.

See Also GAIN, POLE, SAMPLE_FREQ