DCX-PCI300

Modular Multi-Axis Motion Control System

Motion Controller User's Manual

Revision 1.2b

Precision MicroControl Corporation
2075-N Corte del Nogal
Carlsbad, CA 92009-1415 USA

Tel: (760) 930-0101
Fax: (760) 930-0222

www.pmccorp.com

Information: info@pmccorp.com
Technical Support: support@pmccorp.com

LIMITED WARRANTY

All products manufactured by PRECISION MICROCONTROL CORPORATION are guaranteed to be free from defects in material and workmanship, for a period of five years from the date of shipment. Liability is limited to FOB Factory repair, or replacement, of the product. Other products supplied as part of the system carry the warranty of the manufacturer.

PRECISION MICROCONTROL CORPORATION does not assume any liability for improper use or installation or consequential damage.

Information in this document is subject to change without notice.

Precision MicroControl
2075-N Corte del Nogal
Carlsbad, CA 92009-1415

Phone: (760)930-0101
Fax: (760)930-0222
World Wide Web: www.pmccorp.com
Email:
Information: info@pmccorp.com
Technical support: support@pmccorp.com
Sales: sales@pmccorp.com

Table of Contents

User manual revision history

Rev.	*Date*	*Description*
1.0 Pre	3/12/2001	Preliminary release
	4/26/2001	Added MCCL & Multi-Tasking descriptions
	5/3/2001	Miscellaneous edits
	5/14/2001	Added - Initializing and Restoring Controller Configuration description
	5/21/2001	BF320 pinouts
	5/21/2001	BF3XX-H pinouts and high density connectors module mapping
	5/21/2001	Flash Wizard 2.20 now supported
	5/24/2001	Updated for firmware revision 1.1a
	5/24/2001	IIR filter description
	5/24/2001	Integral gain option description
	6/7/2001	Noted that the DCX-MC360 does not currently support Capture & Compare
1.0	6/7/2001	Initial release
1.1	7/26/2001	Added support for Motion Integrator
	8/14/2001	Miscellaneous edits
1.2	1/15/2002	Added Dual Axis Motion Control Modules (DCX-MC302-H, DCX-MC362-H)
	1/15/2002	Added Windows XP as a supported operating system
	1/22/2002	Updated to match firmware revision 2.0a
	1/23/2002	Added MCAPI support to Position Capture / Position Compare description
	1/25/2002	Added MCAPI support for User Define digital filters
	2/7/2002	Added MCAPI support to closed loop stepper description
	2/7/2002	Changed Homing routines
	2/13/2002	Updated to match firmware revision 2.1a
	2/26/2002	Updated to match MCAPI 3.2
1.2b	8/16/2002	Updated ribbon cable connector manufacturer part number
	12/19/2003	Added J5 connector label to DCX-PCI300 motherboard drawing

Contact us at:

Precision MicroControl
2075-N Corte del Nogal
Carlsbad, CA 92009-1415

Phone: (760)930-0101
Fax: (760)930-0222
World Wide Web: www.pmccorp.com
Email:
Information: info@pmccorp.com
Technical support: support@pmccorp.com
Sales: sales@pmccorp.com

Prologue

The documentation set for the DCX-PCI300 is divided into four volumes. The titles of each of the individual volumes are:

DCX-PCI300 Introduction and Installation Guide
DCX-PCI300 User's Manual
Motion Control Application Programming Interface (MCAPI) Reference Manual
Motion Control Command Language (MCCL) Reference Manual

All four volumes of the documentation set are available on PMC's **MotionCD**. In addition to PDF versions of the DCX-PCI300 documentation set the **MotionCD** includes:

- Tutorials (PowerPoint presentations)
  - An Introduction to PMC Motion Control
  - Installing a PMC Motion Controller (Does not Address PCI bus controllers)
  - Introduction to Motion Control Programming with the Motion Control API
  - Servo Systems Primer
  - DCX Servo Tuning
- PMC AppNOTES – detailed descriptions of specific motion control applications
- PMC TechNOTES – one page technical support documents
- PMC Product catalogs and brochures

Chapter

1

Introduction

This document describes the use of the DCX-PCI300 Modular Multi-Axis Motion Control System. For controller and software installation information please refer to the **DCX-PCI300 Introduction and Installation Guide**.

The DCX-PCI300 is an Intel compatible PC computer based servo motor, stepper motor, and I/O controller.

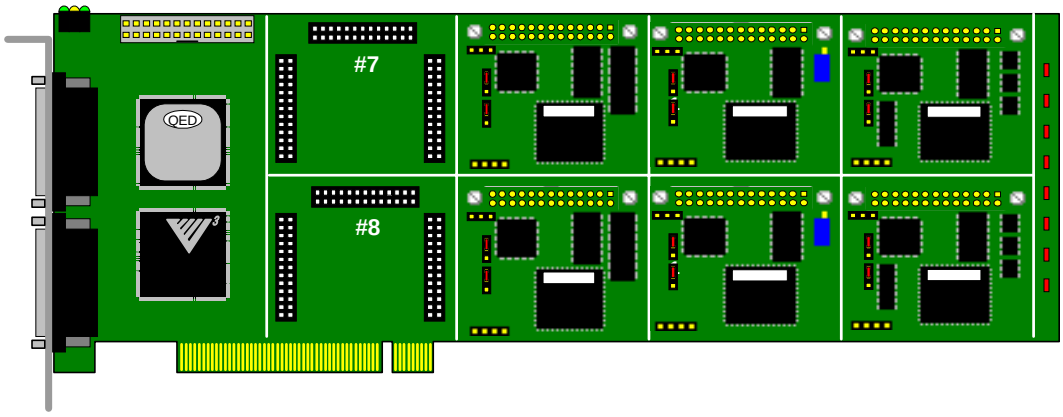


Figure 1: The high density connector version (DCX-PCI300-H) of the DCX-PCI300 Motion Controller

The DCX-PCI 300 is a true PCI 'plug and play' card. When the PC is turned on, the DCX-PCI300 is *dynamically addressed* into the memory map of the PC. The PC communicates to the motion controller via dual ported memory on the DCX-PCI300. By communicating via dual ported memory the PC is able to issue commands (move a motor, change the velocity, etc.) to the controller, and retrieve data from the controller (report to position of an axis, report the state of a digital input, etc.) without interrupting the basic operations of the controller

But a hardware based motion control card provides only one half of the overall motion control solution. State of the art motion control systems typically require sophisticated multi-threaded application programs and eye catching operator interfaces. PMC's **Motion Control Application Programming**

Interface (**MCAPI**) provides the machine designer with device drivers and a powerful function library for Windows XP/2000/NT/Me/98 based applications.

Figure 2: PMC's Windows Motion Control Panel

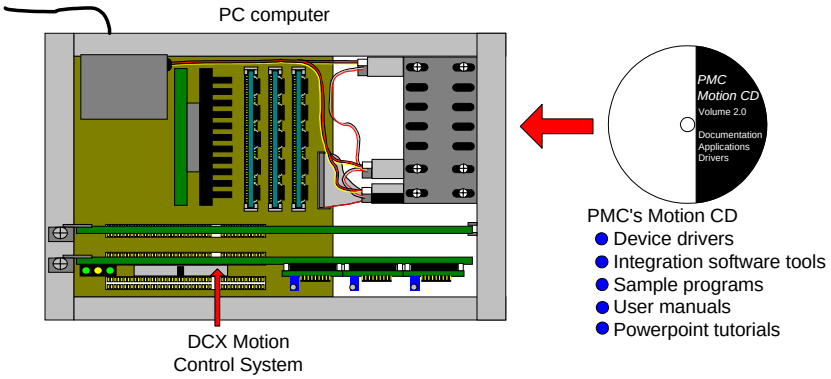
```
MCEnableAxis( HCTRLR hCtlr, Word xAxis, short int bState );
MCMoveRelative( HCTRLR hCtlr, Word xAxis, double Distance );
MCIsStopped( HCTRLR hCtlr, Word xAxis, double Timeout );
```

Figure 3: Function Library examples

The MCAPI supports today's popular programming environments including:

- C/C++
- Visual Basic
- Delphi
- LabVIEW

The DCX-PCI300 Motion Controller can be installed in most any Windows PC computer. It executes motion functions independent of the host, so other than the minimum requirements for the selected operating environment (XP/2000/NT/ME/98), the DCX-PCI300 **does not require or use any additional PC resources** (CPU speed, PC memory, hard disk space, etc...). All documentation, tutorials, and software (drivers, function library, diagnostics and utilities) are available on PMC's **MotionCD**.



The term DCX refers to a system consisting of from 1 to 9 circuit boards assembled together to form a motion control assembly. The platform for a main component of the DCX system is the DCX-PCI300 "motherboard".

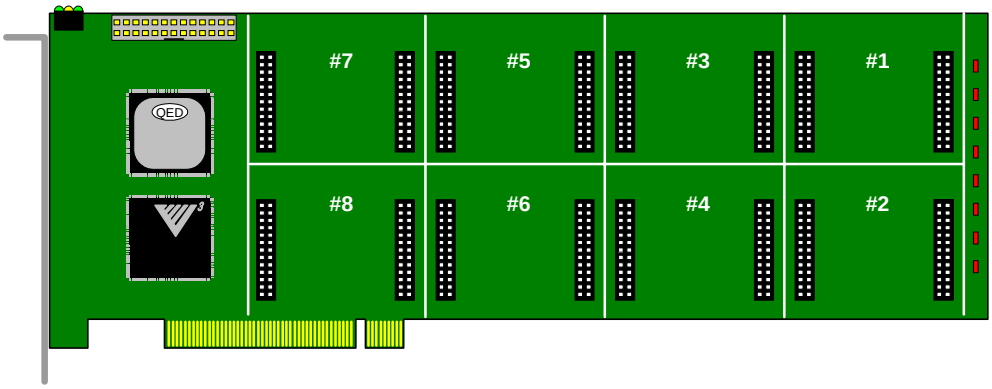
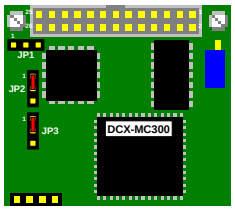


Figure 4: DCX-PCI300-H Motion Control Motherboard

On to this platform the user installs one or more DCX modules, which are two inch square daughter cards. These modules provide the low level motion control processing and control signals (+/- 10V command, Step/Direction, Limit +, Limit -, Amp/Drive Enable, etc.). For a detailed description of the capabilities and part numbers of the components that make up the DCX-PCI300 Modular Motion Control System please refer to **Chapters 1 and 5** of the **DCX-PCI300 Introduction and Installation Guide**.

DCX Motion Control Modules

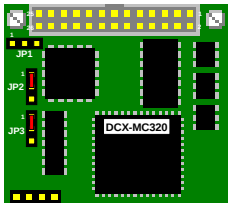


DCX-MC300 Servo Motor Control Module
- DCX-MC300-H (for high density cabling)
- DCX-MC300-R (for ribbon cable connections)

Supported motor type: DC Brushless, Brush, Hydraulic Servo Valves, Pneumatic Servo Valves

Command output: **+/- 10 volt, 16 bit analog for use with servo amplifier**

I/O
Inputs (opto isolated)- Encoder Coarse Home, Limit +, and Limit -, Amplifier Fault
Output (opto isolated) – Amplifier Enable
Feedback: Quadrature Incremental Encoder Interface, 10 MHz
- Primary - Quadrature Incremental Encoder, 10MHz, Single ended (A, B, Z) or Differential (A+, A-, B+, B-, Z+, Z-)
- Auxiliary - Quadrature Incremental Encoder, 10 MHz, Single ended (A, B, Z+, Z-)

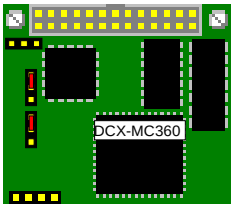


DCX-MC320 AC Brushless Servo Motor Control Module with on-board Sine Commutation
- DCX-MC320-H (for high density cabling)
- DCX-MC320-R (for ribbon cable connections)

Supported motor type: Brushless AC Servo, Linear Motor

Command output: dual **+/- 10 volt, 16 bit analog for use with servo amplifier**

I/O
Inputs (opto isolated) - Encoder Coarse Home, Limit +, and Limit -, Amplifier Fault
Output (opto isolated) – Amplifier Enable
Feedback:
- Primary - Quadrature Incremental Encoder, 10 MHz, Single ended (A, B, Z) or Differential (A+, A-, B+, B-, Z+, Z-)
- Auxiliary – Hall Effect Sensors (A, B, C)



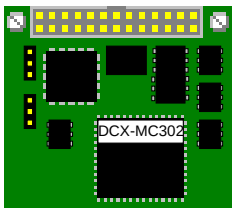
DCX-MC360 Stepper Motor Control Module
- DCX-MC300-H (for high density cabling)
- DCX-MC300-R (for ribbon cable connections)

Supported motor type: Open loop stepper, Closed loop stepper, Step/Dir controlled servo

Command output: **Step/Direction or CW/CCW (software programmable), open collector drivers (+5 to +30 volts @ 125 ma)**

I/O
Inputs (opto isolated)- Home, Limit +, and Limit -, Null
Outputs (open collector driver) – Drive Enable, Half/Full step, Full/Half current
Feedback (optional):
- Quadrature Incremental Encoder, 10MHz, Single ended (A, B, Z) or Differential (A+, A-, B+, B-, Z+, Z-)

DCX Motion Control Modules (continued)

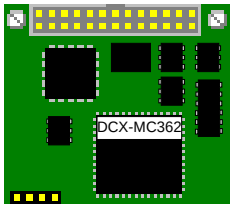


DCX-MC302-H Dual Servo Motor Control Module

Supported motor type: DC Brushless, Brush, Hydraulic Servo Valves, Pneumatic Servo Valves

Command output: Dual **+/- 10 volt, 16 bit analog for use with servo amplifier**

I/O
Inputs (opto isolated)- Dual Encoder Coarse Home, Dual Limit +, Dual Limit -, Dual Amp. Fault
Output (opto isolated) – Dual Amplifier Enable
Feedback: Dual Quadrature Incremental Encoder Interface, 10 MHz
Single ended (A, B, Z) or Differential (A+, A-, B+, B-, Z+, Z-)



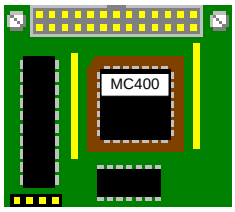
DCX-MC362-H Dual Stepper Motor Control Module

Supported motor type: Open loop stepper or Step/Dir controlled servo

Command output: Dual **Step/Direction or CW/CCW (software programmable), open collector drivers (+5 to +30 volts @ 125 ma)**

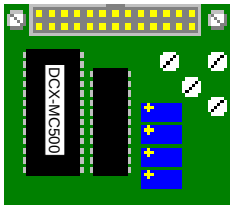
I/O
Inputs (opto isolated)- Dual Home, Dual Limit +, Dual Limit -, Dual Drive Fault
Outputs (open collector driver) – Dual Drive Enable, Dual Full/Half current

DCX General Purpose I/O Modules



DCX-MC400 - 16 Channel Digital I/O Expansion module
DCX-MC400-H (for high density cabling)
DCX-MC400-R (for ribbon cable connections)

Each channel is individually programmable as either an input or output
TTL level (0 – 5 volt, 2 ma sink/source)



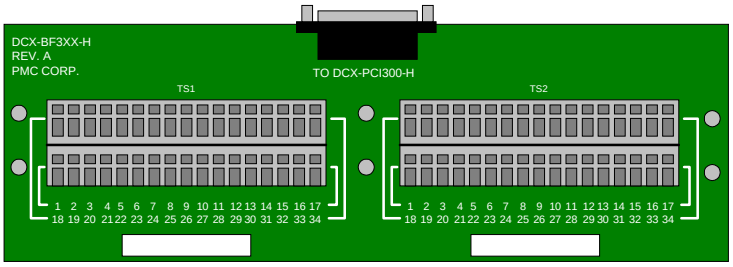
DCX-MC500 – 4 Channel Analog I/O Expansion module
MC500-H – 4 input channels & 4 output channels (high density cabling)
MC510-H – 4 input channels only (high density cabling)
MC520-H – 4 output channels only (high density cabling)

MC500-R – 4 input channels & 4 output channels (ribbon cable connections)
MC510-H – 4 input channels only (ribbon cable connections)
MC520-H – 4 output channels only (ribbon cable connections)

Inputs – 4 channels, 0 – 5 volts, 12 bit
Outputs – 4 channels, 0 – 5 volts and/or –10 - +10 volts, 12 bit

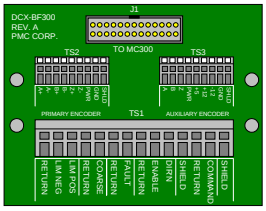
DCX Motion Control Breakout Assemblies

High Density Connection Breakouts

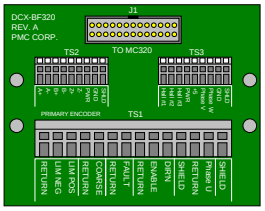


DCX-BF3XX-R – DIN Rail mounted breakout assembly for all -H DCX Modules. Each unit DCX-BF3XX-H breakouts out all signals for 2 DCX module locations.

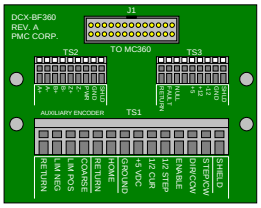
Ribbon Cable Connection Breakouts



DCX-BF300-R – DIN Rail mounted breakout assembly for DCX Servo Motor Control Module (DCX-MC300-R).



DCX-BF320-R – DIN Rail mounted breakout assembly for DCX AC Brushless Servo Motor Control Module (DCX-MC320-R).



DCX-BF360-R – DIN Rail mounted breakout assembly for DCX Stepper Motor Control Module (DCX-MC360-R).

Chapter Contents

- Typical motion control system
- The Command Set is the Heart of the Motion Controller
- The Modular Architecture of the DCX-PCI300
- Why does a servo need to be tuned?

Chapter 2

DCX Motion Control Primer

First things first, what is motion control?

Using a digital processor to coordinate the movement of mechanical systems

In years past the typical motion control system was comprised of :

- A PLC (Programmable Logic Controller) which served as the digital processor
- A user interface from which the user could program and monitor the actions of the PLC
- One or more motors, either servo or stepper
- An amplifier/driver for each motor – provides the drive current for the motor windings
- A feedback device is required to 'close the loop' if servo motors are being controlled
- End of travel (or Limit switches) sensors are used for linear motion axes
- The load – here a platform (or stage) is mounted on bearings. A lead screw is coupled to the motor shaft. When the motor rotates, the stage moves along the lead screw.

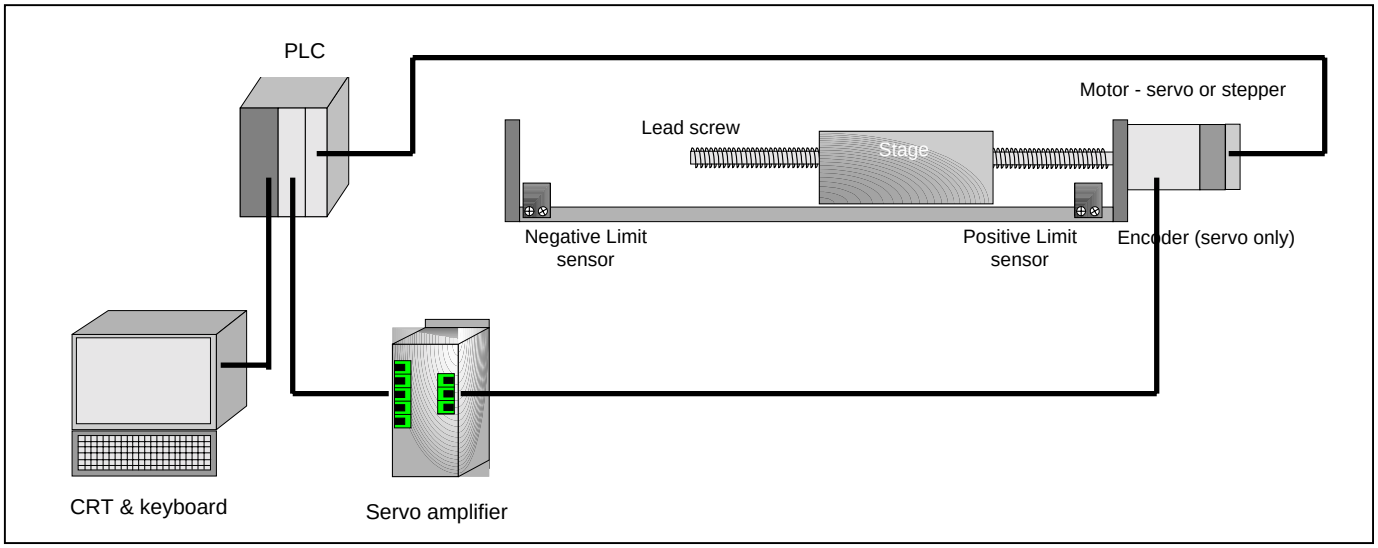


Figure 5: Traditional PLC motion control system

Today's state of the art motion control systems require sophisticated GUI's (Graphical User Interface) and sophisticated multi-threaded application programs to allow the machine operator to communicate with the machine. The GUI is typically implemented using high level programming languages (C/C++) designed to run on today's powerful Windows PC's.

The PLC motion control system (figure 5), which was programmed in cryptic and proprietary languages, is replaced by a PC computer and a motion control card with Windows device drivers. The machine designer is offered the freedom of multiple operating systems (Windows XP/2000, NT, 98 & 95) and programming environments (C/C++, Visual Basic, Delphi, & LabVIEW).

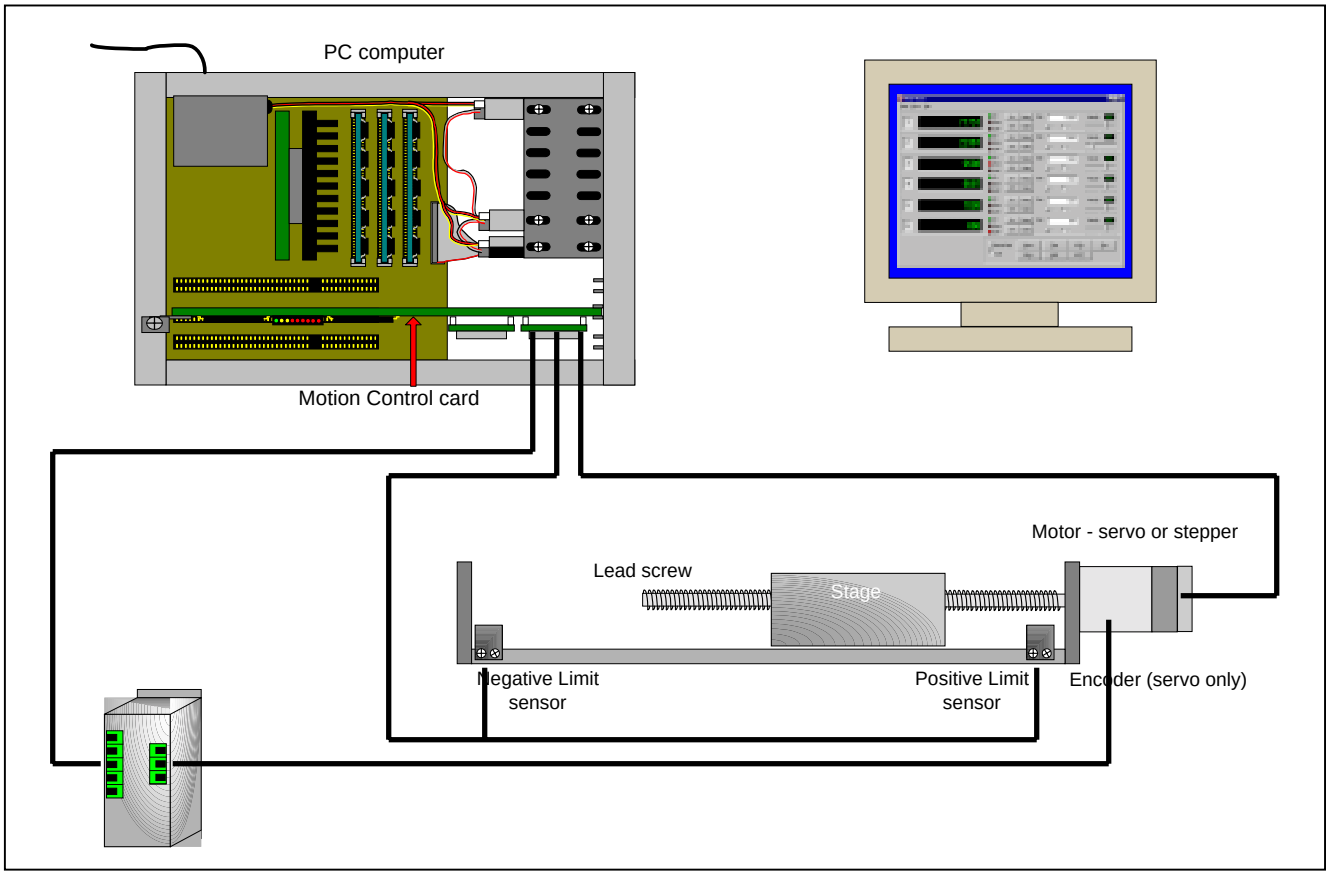


Figure 6: Typical PC based motion control system

The Command Set - the Heart of the Motion Controller

The motion controller is much more than an I/O card with DAC outputs and encoder inputs. The primary task of a PC based motion controller is to off load control and monitoring duties from the PC processor. While most of today's motion controllers have CPU's powerful enough to control the missile defense systems of a small nation, without a powerful and efficient low level command set the motion controller would be nothing more than a very expensive, very dumb I/O card. Everything that a motion control card does (and for that matter **everything that it does not do**) is dependent on the command set. The command set of a state of the art motion controller should include:

- Moving one, some, or all motors simultaneously
- Calculating the trajectories and executing synchronized motion (linear interpolation, circular contouring, helical motion)
- Setting trajectory parameters (maximum velocity, acceleration, deceleration)
- Setting PID filter parameters (proportional gain, derivative gain, derivative sampling period, integral gain, integral limit, allowable following error
- Indicating when a move is complete
- Reporting the status of an axis, current position of an axis, target of a move, current following error
- Electronic gearing of axes
- Homing an axis

The command set for the DCX-PCI300 is called MCCL (Motion Control Command Language) and it supports well over 200 operations. For a complete listing and description of the DCX-PCI300 command set please refer to the **DCX-PCI300 MCCL Reference Manual**.

The primary market for the DCX-PCI300 is multi-threaded Windows NT applications programmed in C/C++. In these types of environments the application program issues calls to PMC's motion control function library (MCAPI). The MCAPI converts the function call into the equivalent MCCL command/commands. The device driver then handles passing the MCCL command code to the motion control card.

For the non programmer, or when it is necessary to determine if unexpected machine behavior is the fault of hardware or software, The MCAPI includes a utility that allows the user to issue MCCL commands directly to the DCX-PCI300. From the keyboard MCCL commands can be entered one character at a time and executed when the user enters a carriage return. From the File Menu the user can download a MCCL text file to the controller.

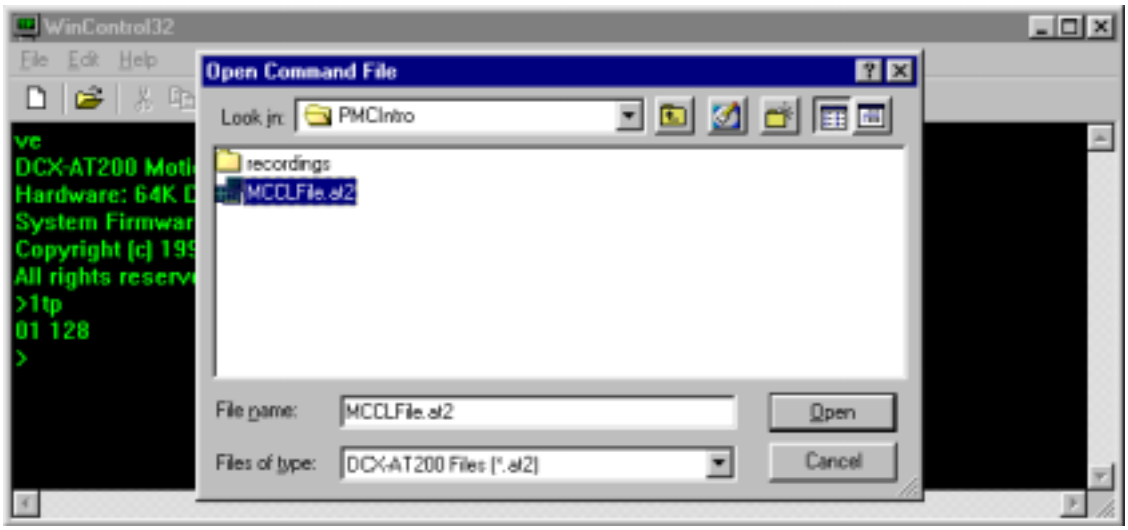


Figure 7: WinControl allows the user to issue MCCL commands directly to the DCX-PCI300

The Modular Architecture of the DCX-PCI300

The DCX-PCI300 is a modular multi-axis motion control card. The architecture of the DCX controller is based on the concept of ***Distributed Control***. Unlike control cards that use a single DSP for communication, motion control, and event sequencing, the DCX controller distributes the processing load, resulting in more deterministic behavior. Here is a diagram detailing the modular DCX-PCI300 and associated components.

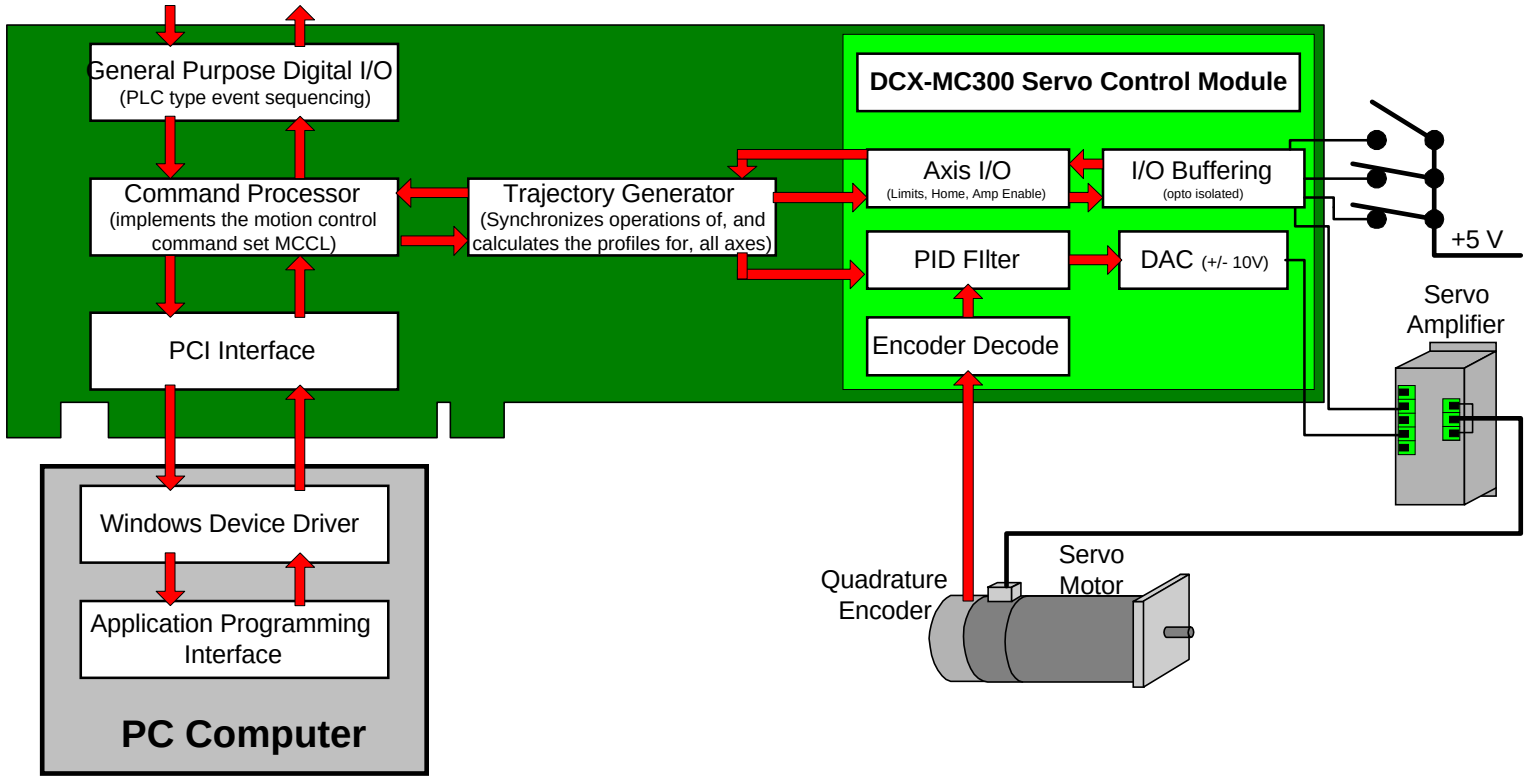


Figure 8: DCX-PCI block diagram

A 192MHz MIPS processor on the DCX motherboard handles PC bus communication and trajectory planning. Low level motion control (PID filter & encoder decode) and dedicated I/O for each axis are handled on the DCX motion control module by a 40 MHz DSP.

Why does a servo need to be tuned?

A servo is a closed loop system, which the dictionary describes as:

> *An automatic system in which the output is constantly compared with the input through some form of feedback. The error (or difference) between the two quantities can be used to bring about the desired amount of control.*

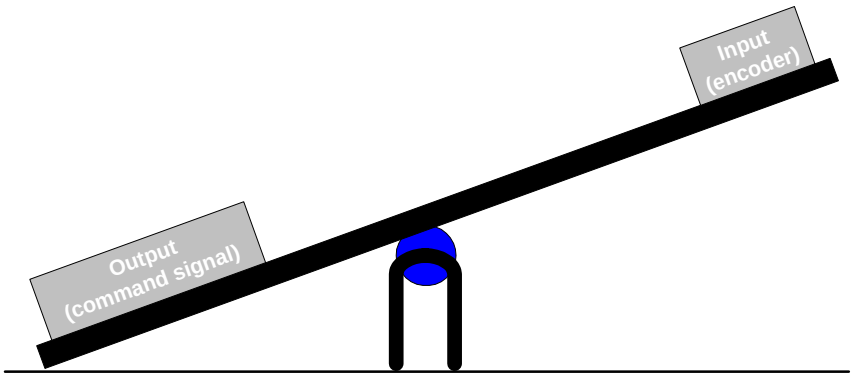
In typical servo systems:

- The output is a +/- 10 volt (torque or velocity) command that is applied as an input to a servo amplifier
- The input described in the dictionary definition comes from an encoder. An encoder is an opto electric device that generates two pulse trains that are phase shifted by 90 degrees
- In order for a servo system to perform properly, the difference (error) between the input and output is multiplied by a set of gain values which results in a new output, bringing about the desired amount of control

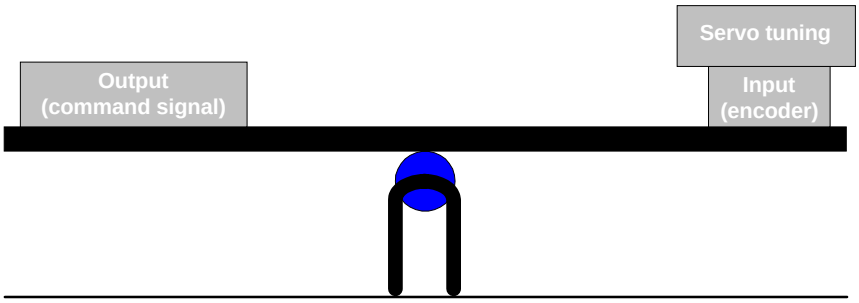
Servo tuning is the process in which the gain values are determined. From one servo axis to another the gain values will change depending on differences between the motion controller, motor, encoder, and load. When a user attempts to move an axis without first tuning the servo (determining the gain values) the motion controller will not be able to calculate the **appropriate** output command to apply to the servo amplifier. One of the two following undesirable results will probably be observed:

- The axis will not move at all
- The axis moves but does not stop at the target, oscillation will probably be present

Imagine a seesaw, with the +/- 10 volt torque/velocity command on one side and the response of the motor/load (feedback from an encoder) on the other side.



Until the servo is tuned, the system is effectively out of balance. Only after a servo has been tuned can the controller calculate the appropriate torque/velocity command output for a given user defined motion.



To tune a servo axis use the Servo Tuning program included with PMC's Motion Integrator software. For assistance with servo tuning refer to the **Motion Control** chapter of this manual or view the PowerPoint tutorials **Servo Systems Primer** and **DCX-AT300 Servo Tuning Tutorial** on **PMC's MotionCD**.

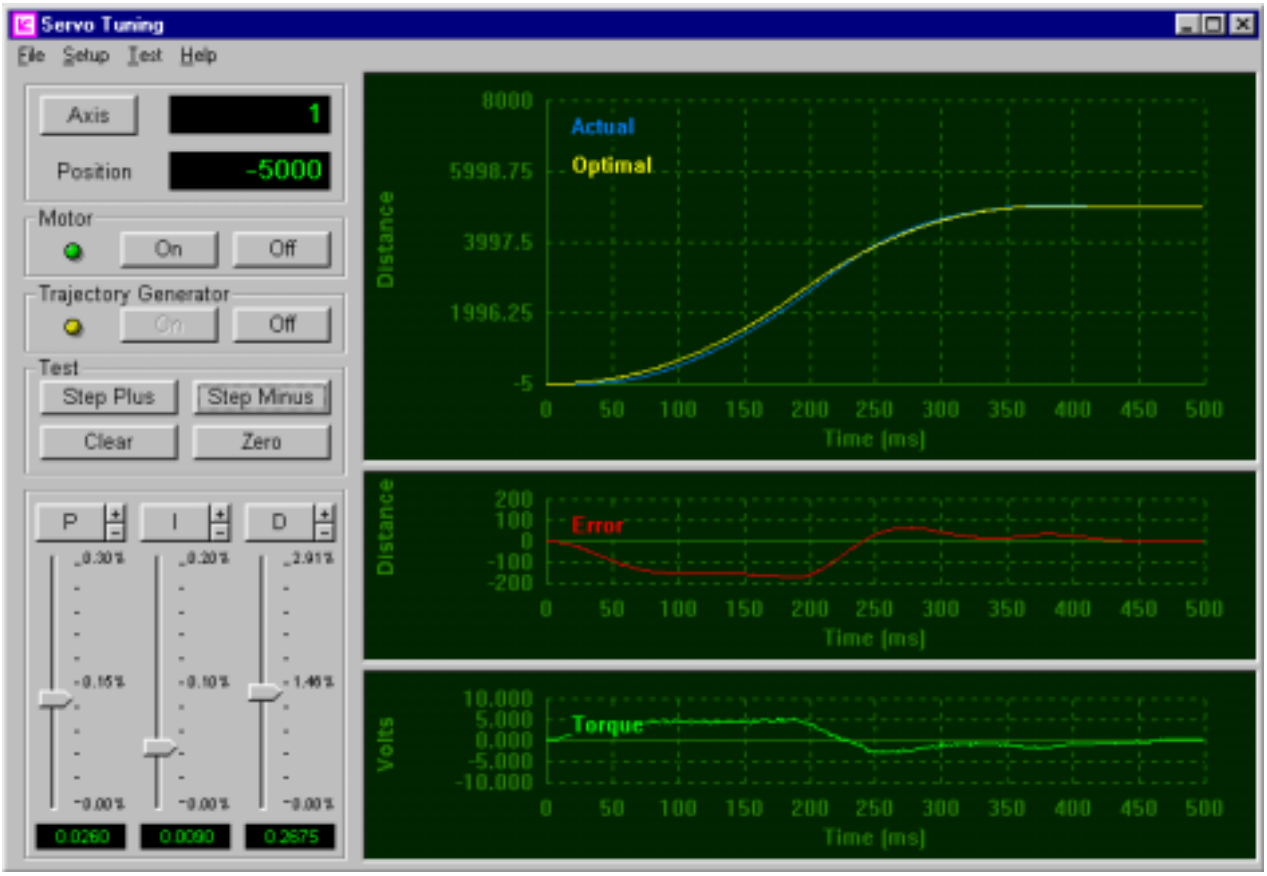


Figure 9: The Servo Tuning program is used to select PID gain values

Chapter Contents

Chapter 3

PC Communication Interfaces

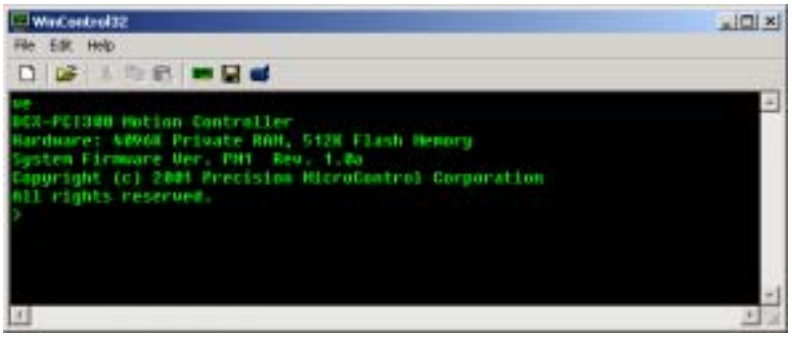
High Speed Binary interface

For PC based application programs the DCX controller provides a high speed binary interface for communicating with the PC via the PCI bus. This interface is implemented using dual ported memory and is mapped into the PC by the BIOS during 'Plug and Play' bus enumeration. PMC's MCAPI provides Windows device drivers and a high level function library for C++, Visual Basic, Delphi, and LabVIEW applications programming. For additional information about available software and integration tools please refer to the **Programming, Software, and Utilities** chapter of the **Introduction and Installation Guide**.

ASCII MCCL Interface

The DCX-PCI300 also provides a PCI ASCII communication interface. When using the WinControl utility the ASCII interface allows the user to communicate directly with the DCX in its native language, MCCL (Motion Control Command Language). The WinControl utility is installed as a component of the MCAPI (Motion Control Application Programming Interface), which is available from PMC's **Motion CD** or web site **www.pmccorp.com**

WinControl.exe



In addition to allowing the user to issue MCCL commands from the keyboard one character at a time, the WinControl utility supports downloading a MCCL text file to the controller. Simply store the command lines in a file using a text editor. Use WinControl's File menu option to open the file. Each command line will be executed as it is displayed. Documenting commands can be added to the MCCL program by preceding the comment by a semi colon.

Chapter Contents

Chapter 4

DCX Operation Basics

Introduction

At its lowest level the operation of the DCX is similar to a microprocessor, it has a predefined instruction set of operations that it can perform. This instruction set, known as MCCL (Motion Control Command Language), consists of over 200 operations that include motion, setup, conditional (If/Then), mathematical, and I/O operations.

However the typical PC based application will never use these low level commands. Instead the programmer will call high level functions (C++, Visual Basic, Delphi, or LabVIEW), which are passed to the DCX via the MCAPI device driver. A example MCAPI function description is:

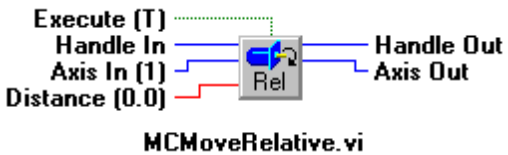
Move to relative position

This command generates a motion of relative distance of n in the specified direction. A motor number must be specified and that motor must be in the on state for any motion to occur. If the motor is in the off state, only its internal target position will be changed.

compatibility: MC300, MC320, M3260
see also: Move to absolute position

C++ Function: void MCMoveRelative(HCTRLR hCtlr, WORD wAxis, double Distance);
Delphi Function: procedure MCMoveRelative(hCtlr: HCTRLR; wAxis: Word; Distance: Double);
VB Function: Sub MCMoveRelative (ByVal hCtrlr As Integer, ByVal axis As Integer, ByVal distance As Double)
MCCL command: aMRn *a* = Axis number *n* = integer or real

LabVIEW VI:



Throughout this manual, when a DCX operation is referenced, the MCAPI command function will be identified by bold, italicized text. The following description differentiates between an absolute and relative move.

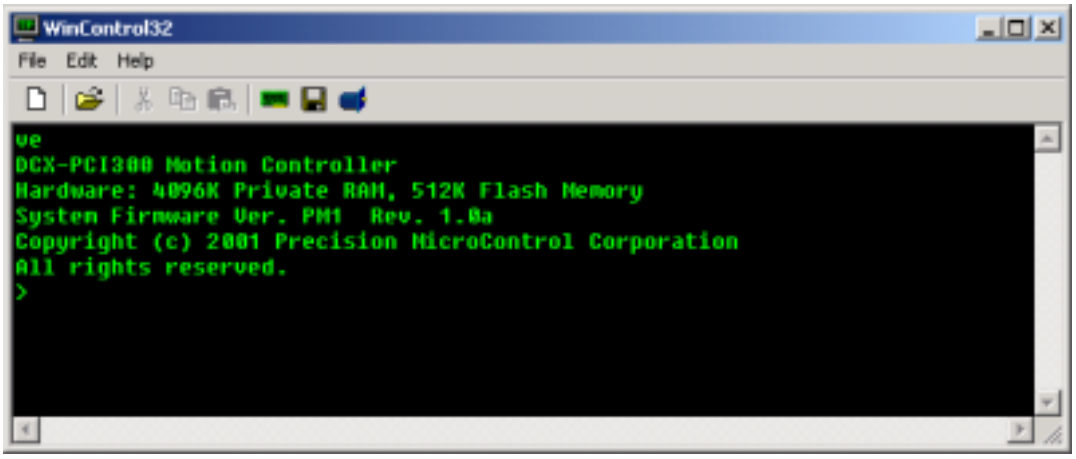
> Point to Point motion is commanded using one of two DCX functions. To move an axis to an absolute position use the function ***MCMoveAbsolute***. To move an axis a relative distance from the current position use the function ***MCMoveRelative***.

Low Level DCX Operations

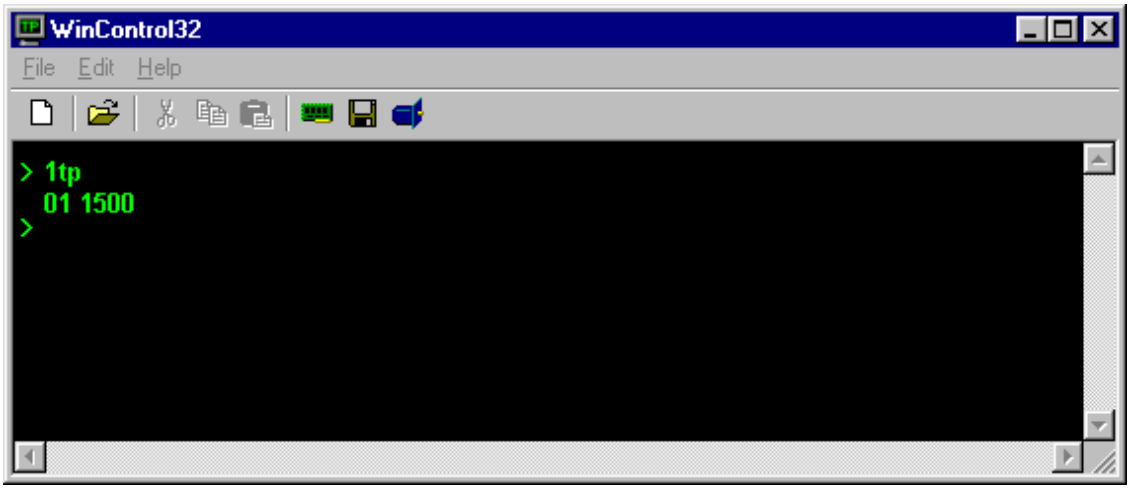
The WinControl utility allows the user to communicate with the DCX in the native language (MCCL) of the controller. This utility communicates with the controller via the PCI ASCII interface. All MCCL commands are described in detail in the **DCX-PCI300 MCCL Reference Manual**.

> **Note** – For information on installing the MCAPI and the DCX-PCI300 please refer to the **DCX-PCI300 Introduction and Installation Guide**.

MCCL commands are two character alphanumeric mnemonics built with two key characters from the description of the operation (eg. "MR" for ***Move Relative***). When the command is received by the DCX (followed by a carriage return) it will be executed. The following graphic shows the result of executing the VE command. This command causes the DCX to report firmware version and the amount of installed memory.



All axis related MCCL commands will be preceded by an axis specifier, identifying to which axis the operation is intended. The following graphic shows the result of issuing the Tell Position (aTP) command to axis number one.



Note that each character typed at the keyboard should be echoed to your display. If you enter an illegal character or an illegal series of valid characters, the DCX will echo a question mark character, followed by an error code. The **MCCL Error Code** listing can be found in the **Chapter 11** of this manual. On receiving this response, you should re-enter the entire command/command string. If you make a mistake in typing, the backspace can be used to correct it, the DCX will not begin to execute a command until a carriage return is received.

Once you are satisfied that the communication link is correctly conveying your commands and responses, you are ready to check the motor interface. When the DCX is powered up or reset, each motor control module is automatically set to the "motor off" state. In this state, there should be no drive current to the motors. For servos it is possible for a small offset voltage to be present. This is usually too small to cause any motion, but some systems have so little friction or such high amplifier gain, that a few millivolts can cause them to drift in an objectionable manner. If this is the case, the "null" voltage can be minimized by adjusting the offset adjustment potentiometer on the respective servo control module.

Before a motor can be successfully commanded to move certain parameters must be set by issuing commands to the DCX. These include; PID filter gains (servo only), trajectory parameters (maximum velocity, acceleration, and deceleration), allowable following error (servo only), configuring motion limits (hard and soft).

At this point the user should refer to the **Motion Control** chapter sections titled **Theory of Operation – Motion Control**, **Servo Operation** and **Stepper Operation**. There the user will find more specific information for each type of motor, including which parameters must be set before a motor should be turned on and how to check the status of the axis.

Assuming that all of the required motor parameters have been defined, the axis is enabled with the **M**otor o**N** (***a*MN**) command. Parameter ‘a’ of the Motor oN command allows the user to turn on a specific axes or all axes. To enable all, enter the Motor oN command with parameter ‘a’ = 0. To enable a single axis issue the Motor oN command where ‘a’ = the axis number to be enabled.

After turning a particular axis on, it should hold steady at one position without moving. The **T**ell **T**arget (***a*TT**) and **T**ell **P**osition (***a*TP**) commands should report the same number. There are several commands that are used to begin motion, including **M**ove **A**bsolute (***a*MA*n***) and **M**ove **R**elative (***a*MR*n***). To move axis 2 by 1000 encoder counts, enter 2MR1000 and a carriage return. If the axis is in the "Motor oN" state, it should move in the direction defined as positive for that axis. To move back

to the previous position enter 2MR-1000 and a carriage return.

With the DCX controller, it is possible to group together several commands. This is not only useful for defining a complex motion that can be repeated by a single keystroke, but is also useful for synchronizing multiple motions. To group commands together, simply place a comma between each command, pressing the return key only after the last command.

A repeat cycle can be set up with the following compound command:

```
2MR1000,WS0.5,MR-1000,WS0.5,RP6   <return>
```

This command string will cause axis 2 to move from position 1000 to position –1000 7 times. The **R**e**P**eat (**RP*n***) command at the end causes the previous command to be repeated 6 additional times. The **W**ait for **S**top (***a*WS*n***) commands are required so that the motion will be completed (trajectory complete) before the return motion is started. The number 0.5 following the WS command specifies the number of seconds to wait after the axis has ceased motion to allow some time for the mechanical components to come to rest and reduce the stresses on them that could occur if the motion were reversed instantaneously. Notice that the axis number need be specified only once on a given command line.

A more complex cycle could be set up involving multiple axes. In this case, the axis that a command acts on is assumed to be the last one specified in the command string. Whenever a new command string is entered, the axis is assumed to be 0 (all) until one is specified.

Entering the following command:

```
2MR1000,3MR-500,0WS0.3,2MR1000,3MR500,0WS0.3,RP4   <return>
```

will cause axis 2 to move in the positive direction and axis 3 to move in the negative direction. When both axes have stopped moving, the WS command will cause a 0.3 second delay after which the remainder of the command line will be executed.

After going through this complex motion 5 times, it can be repeated another 5 times by simply entering a return character. All command strings are retained by the controller until some character other than a return is entered. This comes in handy for observing the position display during a move. If you enter:

```
1MR1000   <return>
1TP    <return>
(return)
(return)
(return)
(return)
```

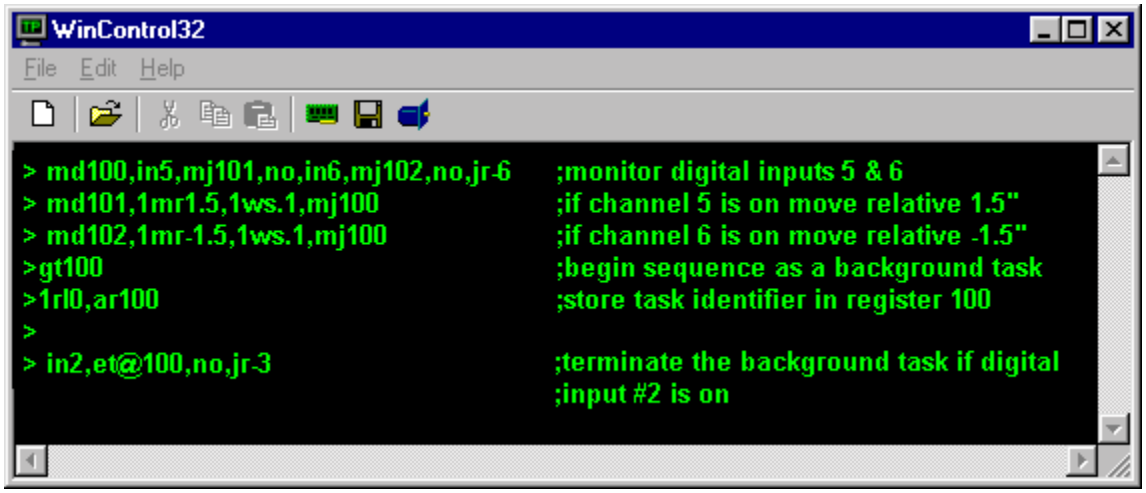
The DCX will respond with a succession of numbers indicating the position of the axis at that time. Many terminals have an "auto-repeat" feature that allows you to track the position of the axis by simply holding down the return key.

Another way to monitor the progress of a movement is to use the ***Repeat*** command without a value. If you enter:

```
1MR10000  <return>
1TP,RP    <return>
```

The position will be displayed continuously. These position reports will continue until stopped by the operator pressing the Escape key.

While the DCX is executing commands, it will ignore all alphanumeric keys that are pressed. The user can abort a currently executing command or string by pressing the escape key. If the user wishes only to pause the execution of commands, the user should press the space bar. In order to restart command execution press the space bar again. If after pausing command execution, the user decides to abort execution, this can be done by pressing the escape key.



Chapter Contents

- Theory of DCX Motion Control
- DCX Servo Basics
- Tuning the Servo
- DCX Stepper Basics
- Closed Loop Steppers
- Moving Motors with PMC demo’s
- Defining the Characteristics of a Move
- Velocity Profiles
- Point to Point Motion
- Constant Velocity Motion
- Contour Motion (arcs and lines)
- Electronic Gearing
- Jogging
- Defining Motion Limits
- Homing Axes
- Motion Complete Indicators
- On the Fly Changes
- Feed Forward (Velocity, Acceleration, Deceleration)
- Save and Restore Axis Configuration

Chapter

5

Motion Control

This chapter describes the basic building blocks of DCX motion control. In general, the modes of motion described in this chapter are common to both servo and stepper motors, with specific differences detailed in the text.

Theory of DCX Motion Control

The DCX motherboard (DCX-PCI300) uses a 192 MHz 32 bit MIPS processor that is programmed to perform motion control tasks. Specially designed servo or stepper motor control modules are installed on the motherboard to configure it for controlling from 1 to 8 servos or stepper motors. Each DCX motion control module (DCX-MC300, DCX-MC320, DCX-MC360) installed on the motherboard provides all the circuitry required to control one motor and its associated axis I/O (home, limits, amp/driver enable, fault, etc...).

The motherboard processor implements a trajectory generator (trapezoidal, S curve, and parabolic) that calculates the desired position and velocity of each servo or stepper motor at fixed time intervals. These values are sent to the respective servo (DCX-MC300 or DCX-MC320) or stepper module (DCX-MC360) installed on the DCX motherboard. Each servo or stepper module has a 40 MHz DSP which is programmed to provide the appropriate control of the servo or stepper motor interfaced to the module.

Servo Motor Control

The DCX servo modules use a velocity feed-forward and position feedback loop to control the servo. The **DCX-MC300** uses a 16 bit, +/-10 volt analog output signal to an external servo amplifier. The **DCX-MC320** uses two 16 bit, +/- 10 volt analog outputs to provide the phase A and B commutation commands allowing a Sine drive amplifier to control brushless servo's and linear motors.

Incremental encoder inputs to these modules provide feedback for closing the position loop. In operation, the servo module subtracts the actual position (feedback position) from the desired position (trajectory generator position), and the resulting position error is processed by the module's digital filter. The output of the digital filter and the velocity feed-forward are combined to set the module's

output level. The external amplifier uses the command signal to drive the motor to the desired position.

The DCX modules DSP monitors the motor's position via an incremental encoder. The two quadrature signals from the encoder are used to keep track of the absolute position of the motor. Each time a logic transition occurs at one of the quadrature inputs, the DCX position counter is incremented or decremented accordingly. This provides four times the resolution over the number of lines provided by the encoder. The encoder interface is buffered by a differential line receiver on the DCX module. Jumpers on the DCX module allow the user to configure the differential receiver for use with single ended or differential encoder.

A "Proportional Integral Derivative" (PID) digital filter on the module is used to compensate the servo feedback loop. The motor is held at the desired position by applying a restoring force to the motor that is proportional to the position error, plus the integral of the error, plus the derivative of the error. The following discrete-time equation illustrates the control performed by the servo controller:

u(n) = Kp*E(n) + Ki sum E(n) + Kd[E(n') - E(n' - 1)]

where u(n) is the module's output signal output at sample time n, E(n) is the position error at sample time n, n' indicates sampling at the derivative sampling rate, and kp, ki, and kd are the discrete-time filter parameters loaded by the users. The first term, the proportional term, provides a restoring force proportional to the position error. The second term, the integration term, provides a restoring force that grows with time. The third term, the derivative term, provides a force proportional to the rate of change of position error. It provides damping in the feedback loop. The sampling interval associated with the derivative term is user-selectable; this capability enables the servo controller to control a wider range of inertial loads.

Stepper Motor Control
The MC360 stepper module contains a pulse generator that is used to provide step and direction (or clockwise/counter clockwise) signals to an external stepper motor driver. In addition to auto calibration on power up, the module has an internal feedback loop which accurately maintains the output pulse frequency. The auxiliary encoder inputs of the module can be connected to an optional incremental encoder for motor position verification or closed loop stepper control.

DCX Servo Basics

The basic steps required to implement closed loop servo motion are:

- Proper encoder operation
- Setting the allowable following error
- Verify proper motor/encoder phasing
- Tuning the servo (PID)

Quadrature Incremental Encoder
All closed loop servo systems require position or velocity feedback. These feedback devices output signals that relay position and/or velocity with which motion controller ‘closes the loop’. The most common feedback device used with intelligent motion control systems is a quadrature incremental encoder.

A quadrature incremental encoder is an opto electric feedback device. A light source and photo sensor pickup are used to detect markings on a glass 'scale'. The more markings on the glass scale, the higher the resolution of the encoder. Circuitry connected to the photo sensor generates two wave forms (Phase A and Phase B), which have a phase difference of 90 degrees. This phase difference is used by the encoder input circuitry of the DCX to:

Determine the direction of rotation (positive or negative) of the encoder/motor
Enhance the resolution of the encoder by a factor of 4.

For example, a 500 line quadrature incremental encoder will have 2000 encoder counts per full rotation. The 90 degree phase difference is also used to determine the direction of motion of the encoder. If phase A comes before phase B, the DCX will determine that motion is in the positive or clockwise direction. If phase B comes before phase A, the DCX will determine that motion is in the negative or counter-clockwise direction.

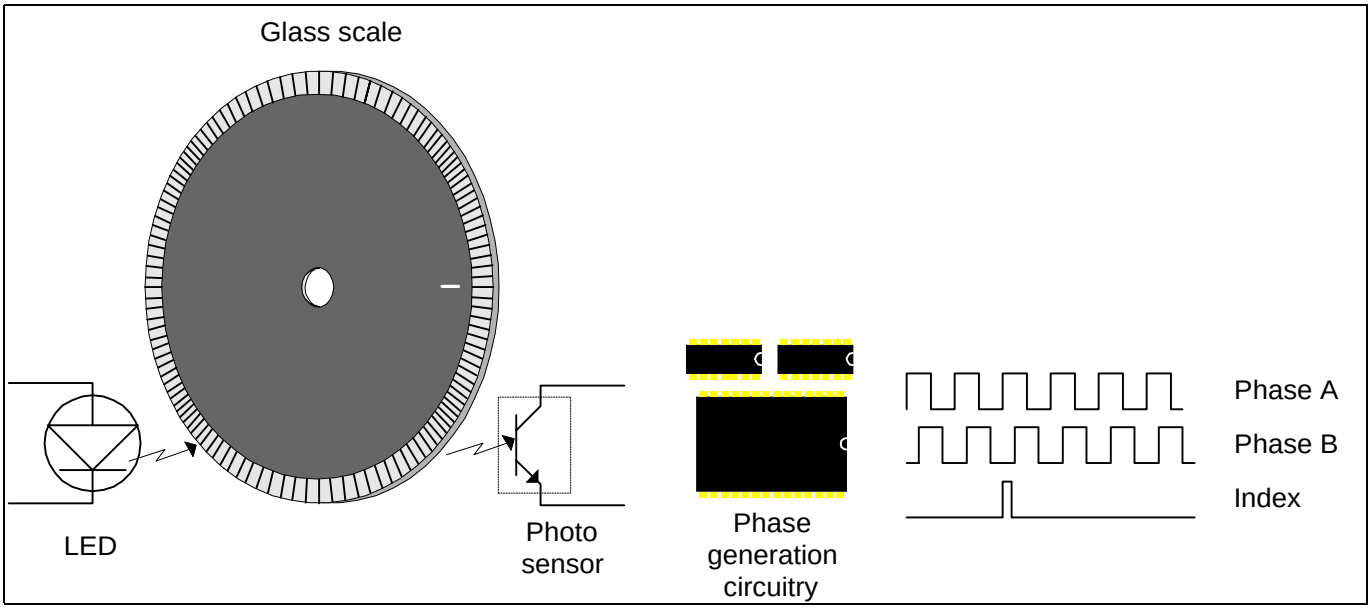
Some quadrature encoders include an additional 'mark' on the glass scale, which is used to generate an index pulse. This signal, which 'goes active' once per rotation, is used by the motion controller to accurately home (re-define the position of an axis) the axis. Please refer to the **Homing Axes** section of this chapter.

There are few options that are typically associated with quadrature encoders.

Output type: Differential or single ended
Differential outputs (A+, A-, B+, B-) are recommended for superior noise immunity but the DCX supports either output type

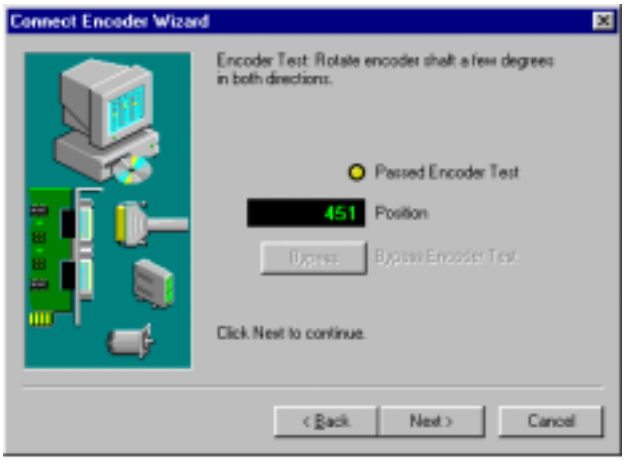
Index or no Index (used for homing the axis)
Differential Index (Z+, Z-) is recommended but the DCX supports single ended Z+ or Z-

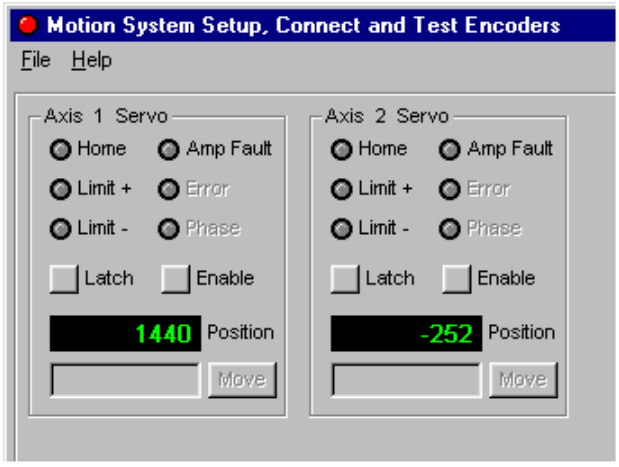
+5 volt supply required or +12 volt supply required.
A +5 volt encoder is recommended but the DCX also supports a +12V encoder



Encoder Checkout
The Motion Integrator program provides easy to use tools for testing the operation of an encoder.. The user has the option of using the Connect Encoder Wizard or the Motion System Setup Test Panel.

Note – Unlike the Connect Encoder Wizard, the Motion System Setup Test panel **does not** allow the user to verify the operation of the encoder Index.



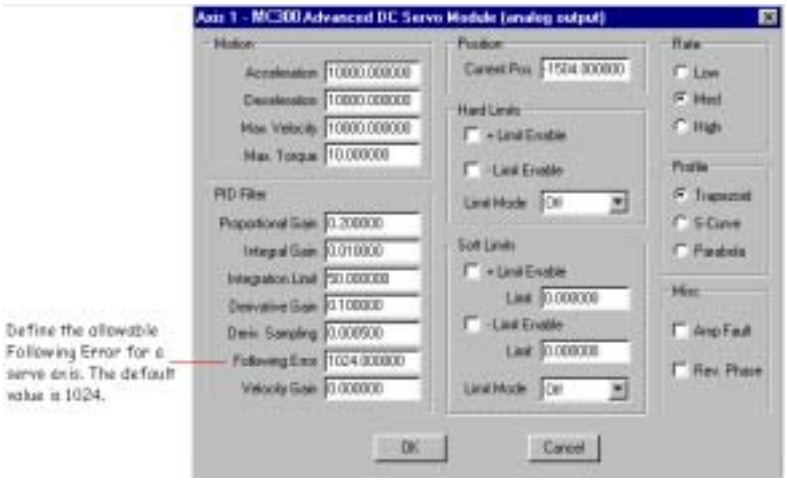


Manually rotate the motor/encoder in either direction, the position reported should increment or decrement accordingly. Refer to the Troubleshooting guide if the DCX does not report a change of position.

Setting the Allowable Following Error
Following error is the difference between where an axis **'is'** and where the controller has **'calculated it should be'**. All servo systems require 'some' position error to generate motion. When a servo axis is turned on, if a position error exists, the PID algorithm will cause a command voltage to be applied to the servo to correct the error.

While an axis is executing a move, the following error will typically be between 20 and 1000 encoder counts. Very high performance systems can be 'tightly tuned' to maintain a following error within 5 to 10 encoder counts. Systems with low resolution encoders and/or high inertial loads will typically maintain a following error between 150 and 5000 encoder counts during a move.

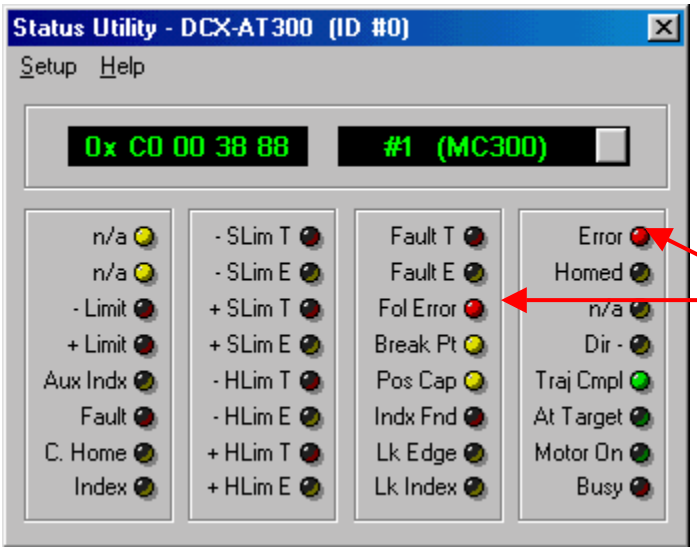
The DCX supports 'hard coded' following error checking. If at anytime the difference between the optimal position and the current position exceeds the user defined 'allowable following error', an error condition will be indicated. The axis will be disabled (Amplifier Enable output turned off, output command signal set to 0.0V) and the axis status word will indicate that an error has occurred. The ***MCEnableAxis( )*** function is used to clear a following error condition. To disable 'hard coded' following error checking set the allowable following error to zero.



The three conditions that will typically cause a following error are:

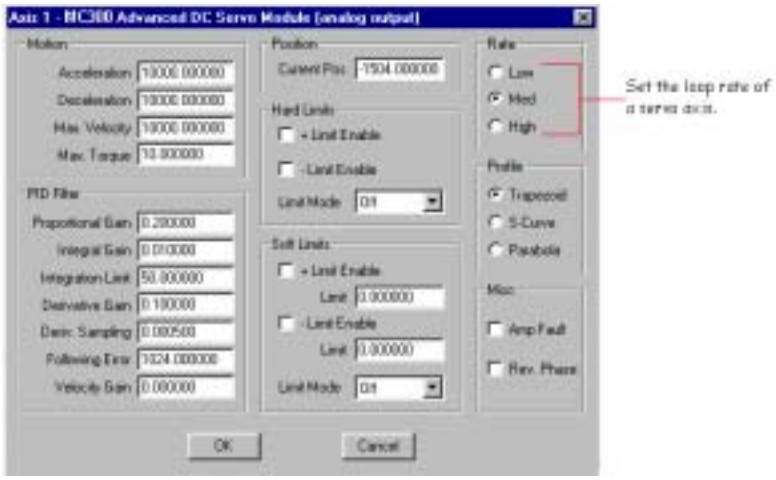
1) Improper servo tuning (Proportional gain **too low**)
2) Velocity profile that the system cannot execute (moving too fast)
3) The axis is reversed phased (positive command results in negative motion)

The Status Panel screen shot below shows the typical display when a following error has occurred.



When the red Error and Fol Error LED's are on it indicates that a following error has occurred

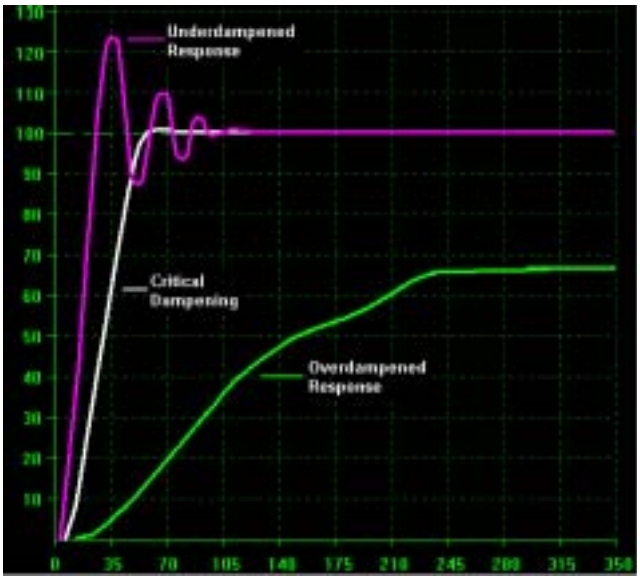
Selecting the Servo Loop Rate
The DCX supports three servo loop rates:



Servo Loop Rate Setting	*Description*
High	**8 KHz servo loop rate**
Medium	4 KHz servo loop rate (default)
Low	2 KHz servo loop rate

Tuning the Servo

A servo motor motion system is a closed loop system with negative feedback. Servo tuning is the process of adjusting the gains (proportional, derivative, and integral) of this axis controller to get the best possible performance from the system. A servo motor and its load both have inertia, which the servo amplifier must accelerate and decelerate while attempting to follow a change in the input (from the motion controller). The presence of inertia will tend to result in over-correction, with the system oscillating or "ringing" beyond either side of its target (under-damped response). This ringing must be damped, but too much damping will cause the response to be sluggish (over-damped response). Proper balancing will result in an ideal or critically-damped system.



Comprehensive PowerPoint tutorials covering servo system basics and a step by step procedure for tuning servos are available on PMC's MotionCD.

The servo system is tuned by applying a command output or 'step response', plotting the resulting motion, then adjusting parameters of the digital PID filter until an acceptable system response is achieved. A step response is an output command by the motion controller to a specific position. A typical step response distance used for tuning a servo is 100 encoder counts. If the system requires:

- Very short duration moves (less than 100 msec's)
- Very small following error value (less than 20 encoder counts)

Then a step response of 50 encoder counts is recommended. If the servo system is moving a high inertial load (minimal friction) then the step response should be increased to 200 – 1000 encoder counts. There is a 'loose' relationship between the step response and the following error of the system. The shorter the step response when tuning the servo, the lower the following error during

application motion.

> Note – Using a short step response (5 – 20 counts) may result in an unstable system that oscillates during and after a commanded move.

From the Windows Start menu open the Servo Tuning program (Programs\MotionControl\ MotionIntegrator\Servo Tuning). From the menu bar select **Setup** and then **Test Setup.** Configure the Test Setup dialog as shown (these settings will command a step response of 100 encoder counts plot window time period of 500 msec's):

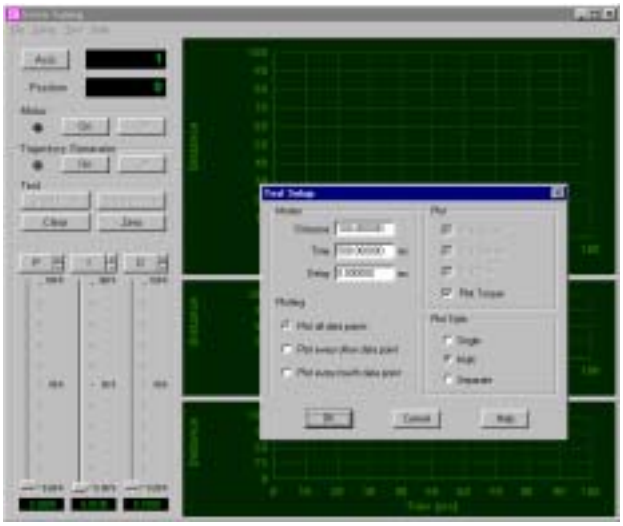


Figure 10:Setting the step response distance and plot window period

From the menu bar select **Setup** and then **Servo Setup.** Configure the Servo Setup dialog as shown:

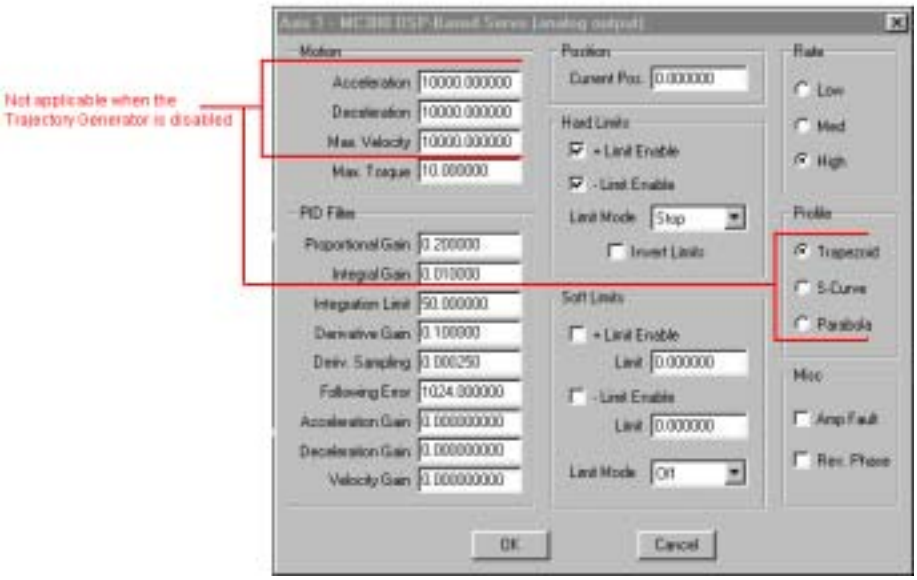
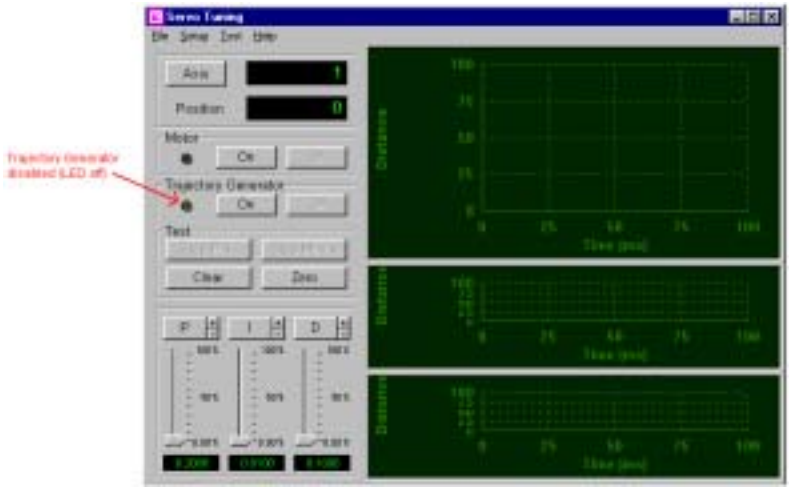


Figure 11:Configure the servo parameters

While setting proportional and derivative gain, the step response should occur with the **Trajectory Generator** disabled. This will result in the magnitude of the output signal being determined only by a PD filter, the controller will not apply a maximum velocity or ramping (acceleration/deceleration).



Setting Proportional Gain

Proportional gain controls the responsiveness of a servo system. Set the slide controls for 'I' (Integral gain) and 'D' (Derivative gain) to 0. Set the slide control for 'P' for .05. Turn the Motor On. Make sure the Trajectory Generator is **off**. Press the Step Plus button, the motor should move and a position versus time plot will be displayed.

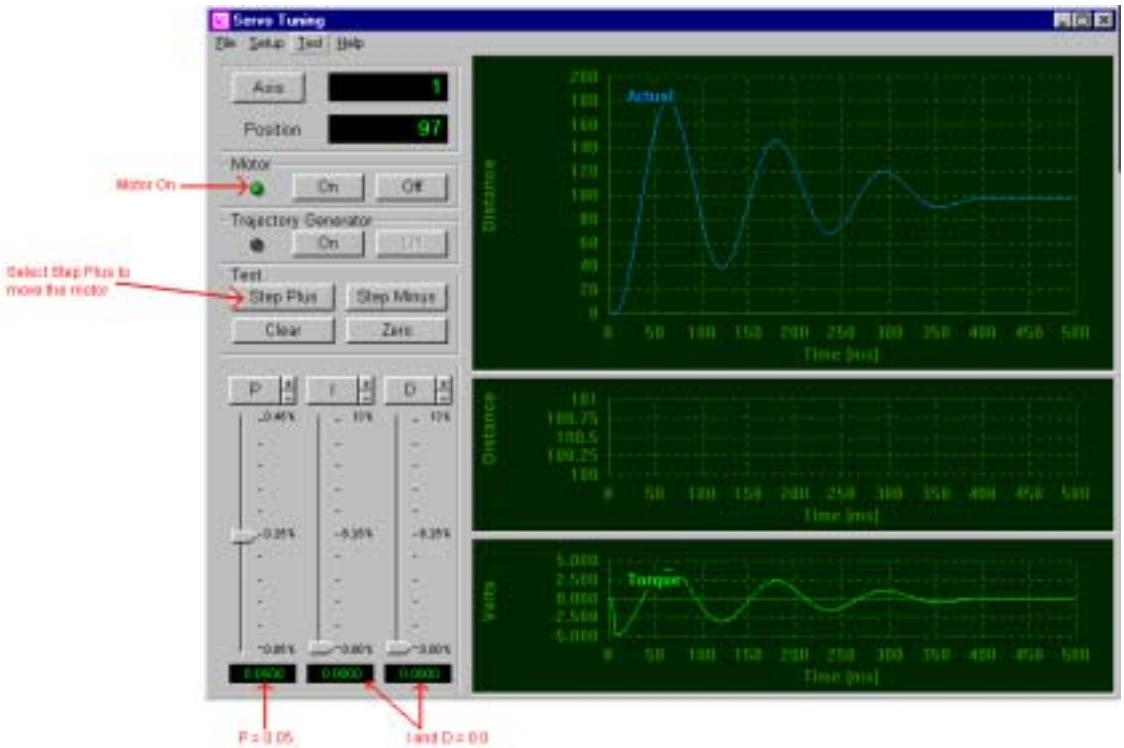


Figure 12: First move (using only proportional gain)

Adjust the proportional gain until the plotted path crosses the target three times (no more, no less) and stops within 5% of the target.

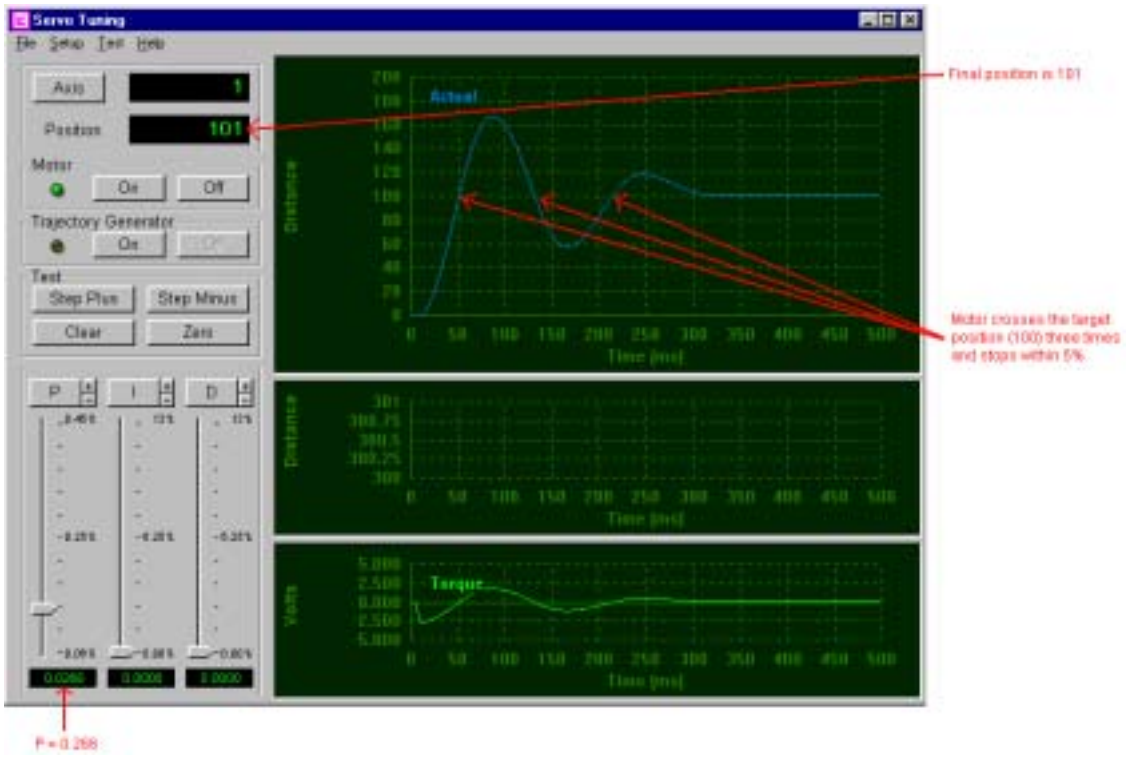


Figure 13: Adjust P (proportional gain) until the motor crosses the target 3 times , no more and no less

If no plotted position path is shown:

- If the ***Motor On LED is still on***, the proportional gain is to low. Increase 'P' by 100%.
- If the ***Motor On LED is off*** an error has occurred. The most likely cause is a following error has occurred which would indicate that the servo is reversed phased. Open the Servo Setup dialog box and select the Reverse Phase option or 'swap' the phase A and B connections from the encoder to the DCX servo module. Turn the motor back on and proceed with the tuning process. If a position path plot is still not displayed refer to the Troubleshooting chapter of this manual.

Setting Derivative Gain

Derivative gain acts as a dampening factor for the servo system. The DCX-PCI300 defaults to calculating the derivative term every time the PID filter is executed (every 125 usec's). For many servo systems this will cause the derivative term to be too large, which may cause a buzzing or grinding noise. For most servo systems it is recommended that the derivative term be calculated every 4 to 8 servo loops. Open the Servo Setup dialog and set the derivative sampling period to 0.00075 (every 6 servo loops). For high inertia (heavy load with minimal friction) applications, the derivative sampling period should be set to between .001 and .002.

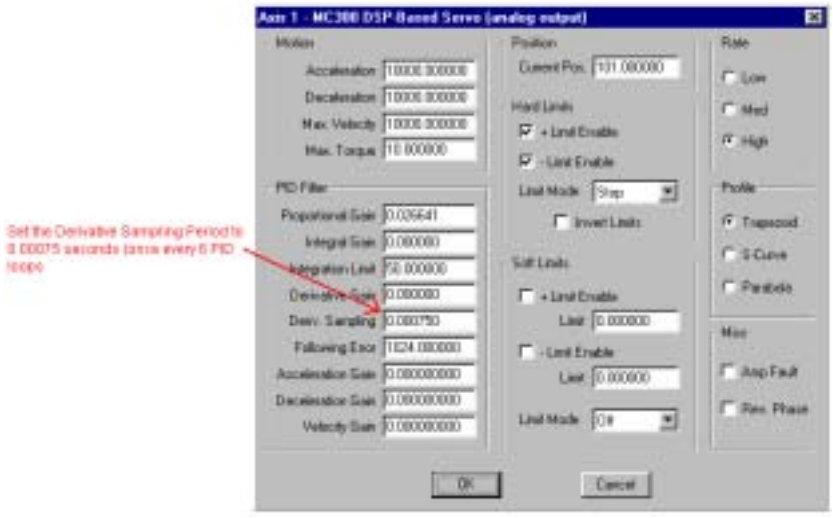


Figure 14:Set Derivative Sampling Period to 0.00075 seconds (every 6 PID loops)

Add a little Derivative gain and then move the motor. Repeat this process until the amount of overshoot (difference between the target and the most positive position) is between 20% to 25%. The goal is to identify the derivative gain setting that:

1) Limits overshoot to between 20% to 25%
2) The final position is as close to the target as possible

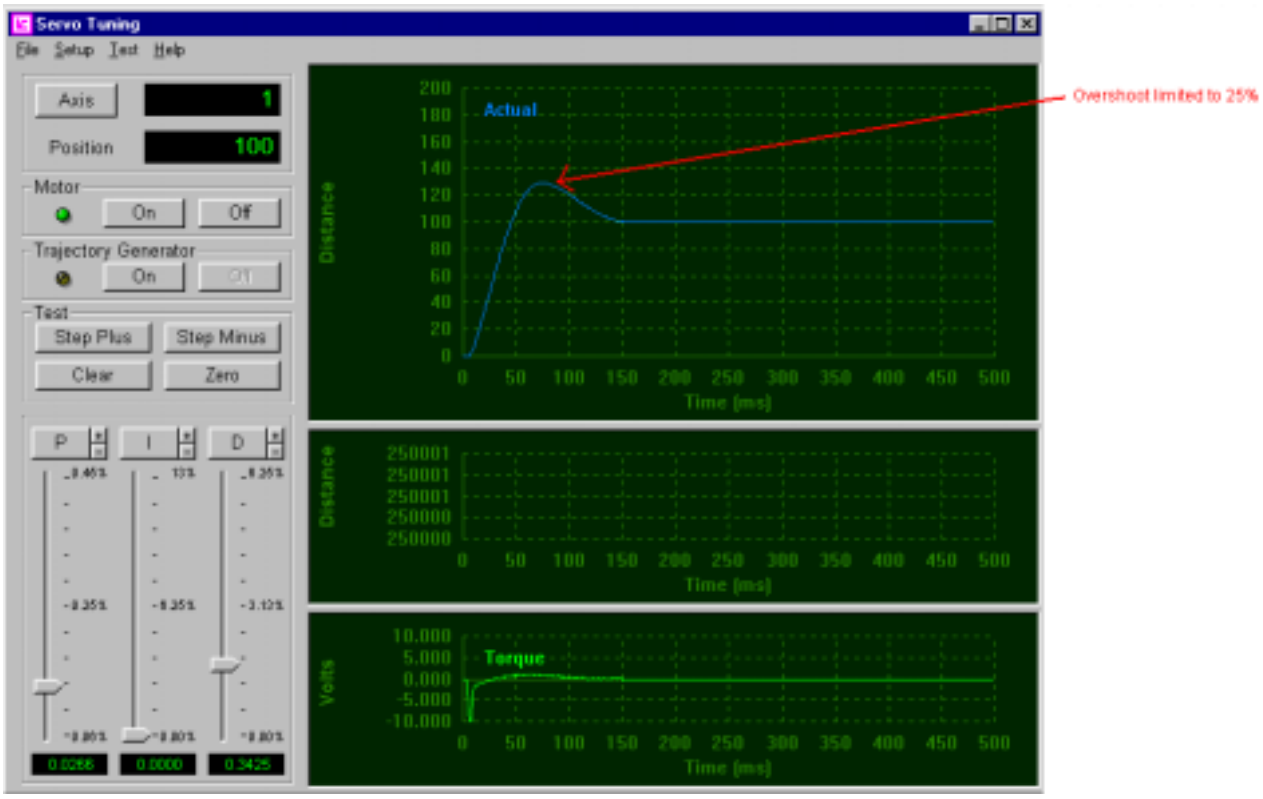


Figure 15:Use Derivative Gain (D) to limit overshoot to 25%

Setting the Integral Gain

Due to friction, 'sticktion', amplifier offset, etc... most servo systems are unable to settle at the target if using only proportional and derivative gain. Integral gain provides a restoring force that increases with time. It is used to correct a static position error of a servo system. If the servo is unable to repeatedly position within +/- one encoder count of the target Integral Gain will, in most cases, position the servo at the target.

To configure the Servo Tuning utility for setting the integral gain:

- Enable the trajectory generator.
- Define trajectory parameters (max. velocity, acceleration, and deceleration) in the Servo Setup dialog
- Define a typical application move distance and duration in the Test Setup dialog

For this example:

- Maximum velocity = 100,000 counts per second
- Acceleration and deceleration = 100,000 counts per second per second
- Move distance = 3,000 counts
- Plot window time = 700 msec's

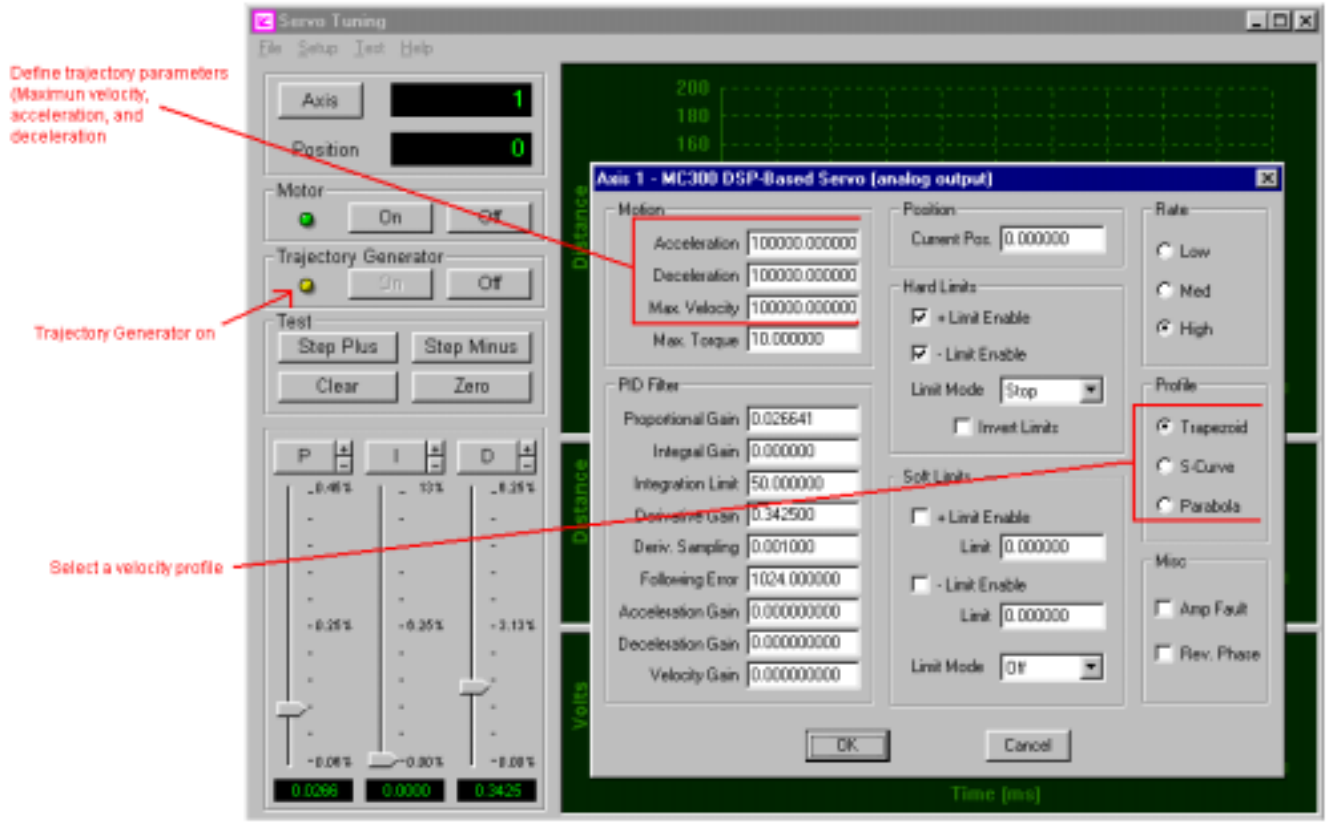


Figure 16:Turn on the Trajectory Generator, define trajectory parameters, select a velocity profile

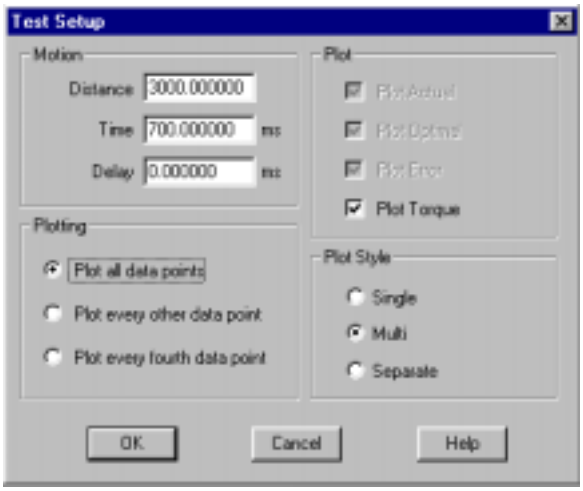


Figure 17:Define move distance and plot window period

With the trajectory generator enabled, a step response will cause two plot traces to be displayed in the upper window. The blue trace is a plot of the actual positions of the servo. The yellow trace is a plot of the calculated (or optimal) positions of the servo. The optimal positions are the result of calculations by the DCX based on the trajectory parameters (max. velocity, acceleration, and deceleration) defined in the Servo Setup dialog. With the Trajectory Generator enabled a plot of the following error (red trace in the middle window) is also displayed. Select the Step Plus button to move the axis.

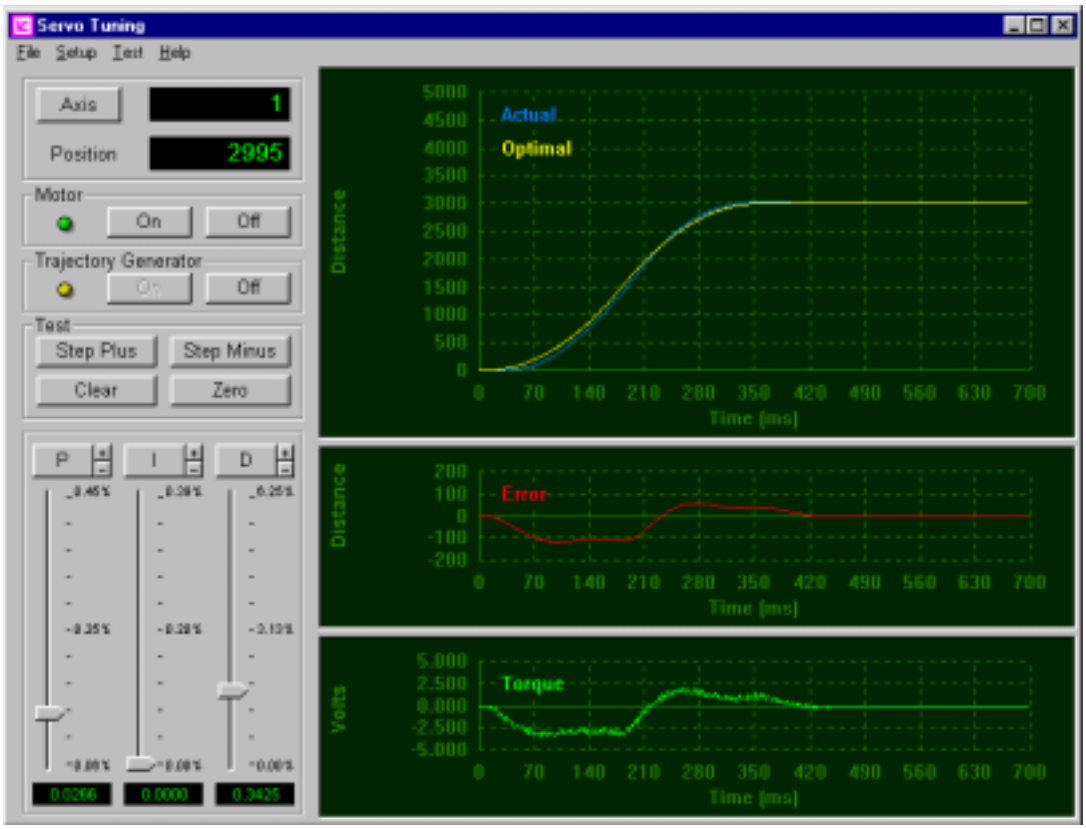


Figure 18:Results of a typical application move prior to setting the Integral Gain

Without executing another move, **slowly** increase the integral gain (I slide control) until the position readout indicates that the axis has reached the target position of the move.

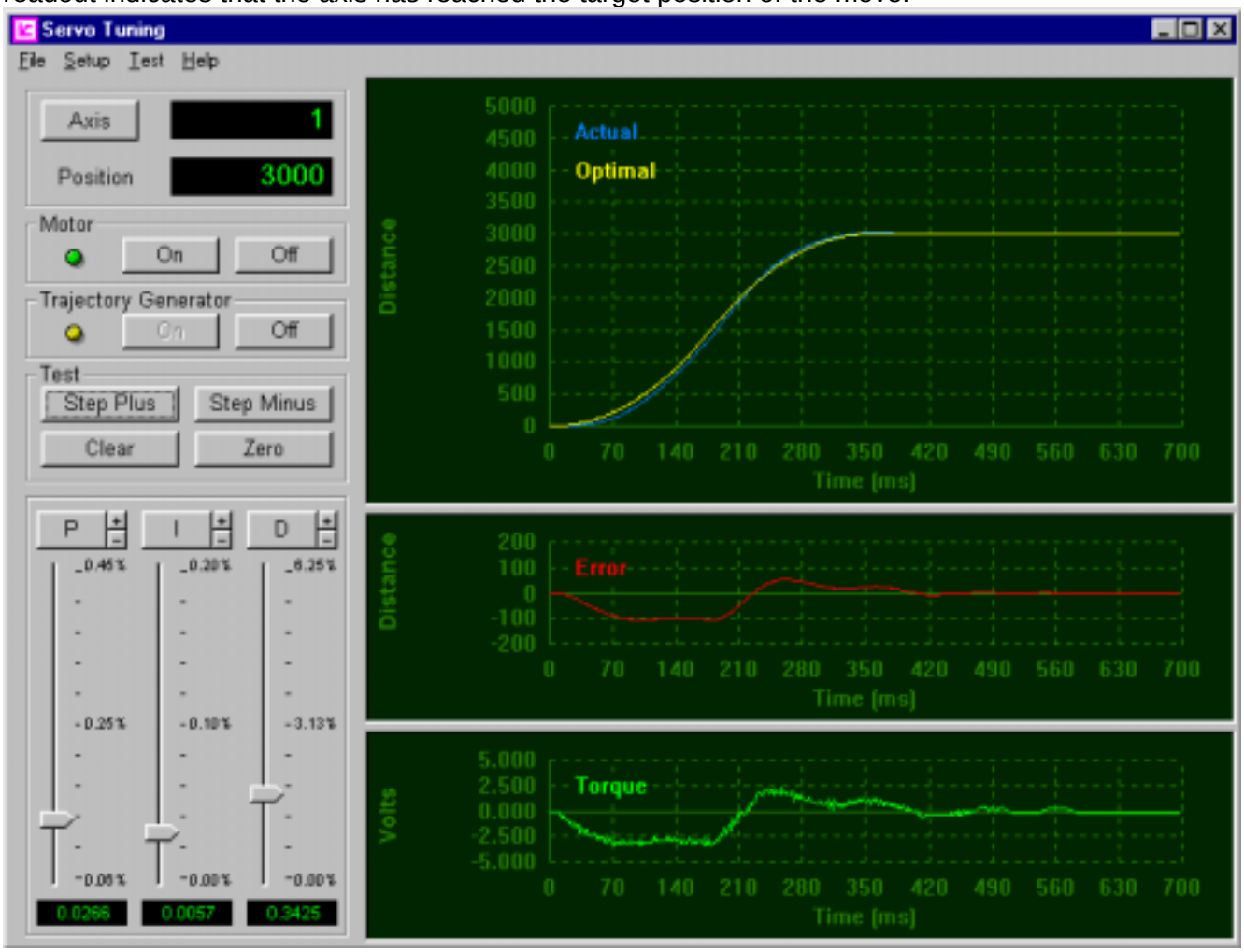


Figure 19:With an Integral Gain setting of 0.0057 the axis repeatedly positions to the target. The axis is now tuned.

If the 'I' control has reached 50% and the axis has not reached the target either:

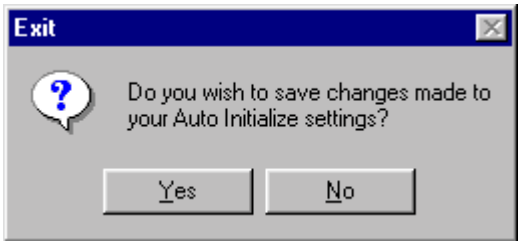
- The Integral Limit is too low, limiting the restoring force that the integral gain can apply. Double the value in the Servo Setup dialog
- The Integral gain slide control range needs to be increased. In the PID setup dialog double the value for the integral gain upper limit

Once the position readout indicates that the axis is at the target execute another move (Step Plus). If the axis stops and settles within one encoder count of the target the servo has been successfully

tuned. If the position readout indicates that the servo is unable to settle, reduce the setting of the integral gain (I term). Execute additional moves until the axis settles at the target. For additional information on integral gain please refer to the description of **Turning off integral gain during a move** in the **Application Solutions** chapter of this manual.

Saving the Tuning Parameters

Once an axis has been tuned you should save the PID and trajectory parameters. Select **Save All Axis Settings** from the **File** menu. Selecting this option will load all servo settings into the MCAPI.INI file (in the Widows folder). In addition when you elect to close the Servo Tuning program it will prompt the user about saving the settings.

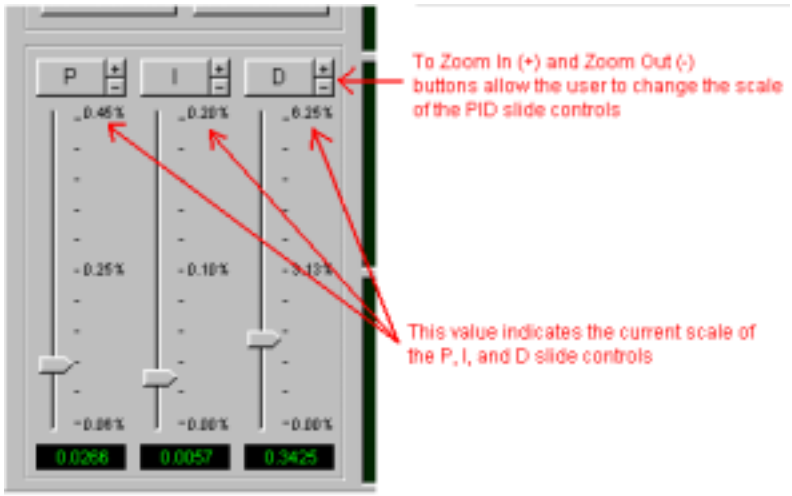


Electing to save the Auto Initialize settings causes the Servo Tuning utility to call the MCAPI Common Dialog function **MCDLG_SaveAxis**. All servo parameters (PID, Trajectory, Limits, etc...) will be saved in the dialog

To define these servo parameters from a user's application program, call the MCAPI Common Dialog function **MCDLG_RestoreAxis**.

Changing the Scale of the Slide Controls

At the top of each slide control is a value showing the current setting as a percentage of the current maximum setting. To change the range of one or more slide controls select the Zoom In (+) or Zoom Out (-) buttons.



For additional information on servo tuning please refer to the tutorials on the MotionCD.

Executing cycle operations from the Servo Tuning program

Beginning with revision 2.4 the servo tuning program allows the user to execute cycle operations. From the Test Setup dialog define the move distance, dwell between positive and negative moves, cycle repeat count, and dwell between cycles.

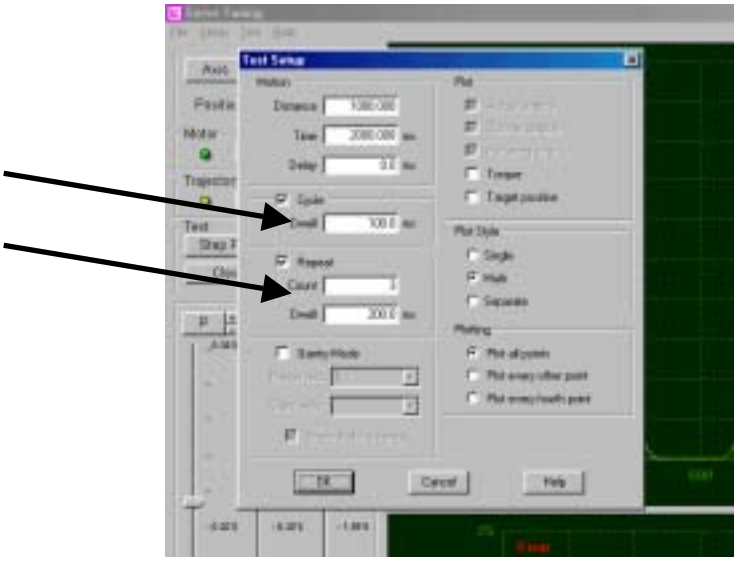


Figure 20: Use the Test Setup dialog to configure the distance, dwell, and repeat count of cycle operations

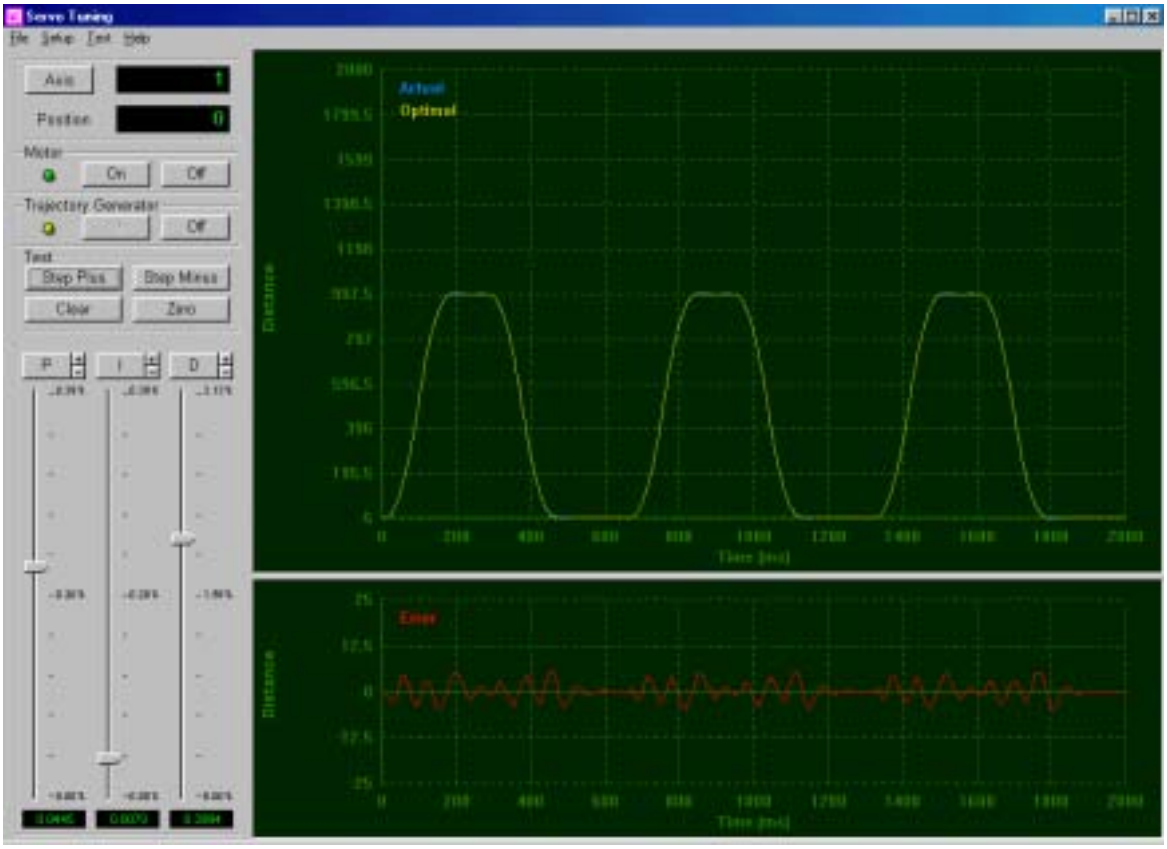


Figure 21: Plotting position and following error of three 1000 count move cycles

Tuning Velocity Mode Amplifier Servo Systems

A velocity mode amplifier incorporates an analog tachometer to provide the feedback for the velocity loop, which is closed within the amplifier. The velocity loop is considered the primary or 'inner' loop of this type of servo system. The DCX, which is a position controller, will close the secondary or 'outer' position loop of the servo system. Combining a velocity mode amplifier with a position loop controller results in what is known as a dual loop system. When this type of system is to be used, it is recommended that the encoder not be directly coupled to the motor. The encoder should be mounted on the external mechanics, as closely coupled as possible to the load or 'end effector'. Typically in a dual loop system, a linear scale (encoder) will be mounted on the slides of each axis.

The most important step of tuning a servo that uses a velocity mode amplifier is to follow the amplifier manufacturers setup instructions ***to the letter***. Since the amplifier provides the **primary** servo control, if it is not setup correctly there is no possibility of attaining acceptable servo system performance.

There are significant differences when tuning servo systems that close the velocity loop external to the DCX (position loop) controller. The digital PID filter of the DCX becomes a secondary component in the generation of the output signal that is applied to the velocity mode amplifier. The primary component that the DCX will use to generate the servo command signal is the Feed Forward term.

Feed Forward defines a voltage level output from the DCX, which in turn commands the velocity mode amplifier to rotate the motor at a specific velocity.

Prior to tuning the servo system the velocity feed forward term must be determined. The following example describes how to calculate and set velocity feed forward of a servo axis:

Setting the Velocity Feed Forward

The main component required to set the velocity feed forward of a DCX servo axis is to determine the output level of the tachometer at a specific motor velocity. For this example, a typical tachometer specification would state:

Output Range	0.0 to +10V
Tach Output @ 1K RPM	1.0 volt

The specification describes a tachometer with an output range of 0 – 10V. The tachometer output ratio is 1.0V per 1,000 RPM's. The resolution of the linear scale encoder is 2000 encoder counts per inch, and the maximum velocity of the axis is 50 inches per second. Note: the servo amplifier may require scaling adjustments for the *RPM/Tachometer voltage output* ratio. The velocity feed forward is calculated as follows:

DCX output = Velocity (encoder counts/sec) **X** Feed forward term (encoder counts/volt/sec.)

10 volts = 100,000 counts/sec. **X** Feed forward term (encoder counts * volt/sec.)

Feed forward = 10 volts / 100,000 counts per sec.

0.0001 = 10 volts / 100,000 counts per sec.

```
1VG0.0001                                ;set velocity gain (velocity feed
                                         ;forward ) with MCCL command

// set velocity gain (velocity feed forward) using MCAPI function
//
    MCGetFilterConfig( hCtrlr, iAxis, &Filter );
    Filter.VelocityGain     = ( hCtlr, 1, 0.0001 );
    MCSetFilterConfig( hCtrlr, iAxis, &Filter );
```

Tuning the Servo

After setting the velocity feed forward (velocity gain) as shown above, open the Servo Tuning Utility. Configure the utility as follows:

1) From the Setup menu, select Servo Setup and define the trajectory parameters (velocity, acceleration, and deceleration) to match the application requirements.

2) From the Test Setup menu define a typical application move distance and duration. For this example, the move distance is 2000 encoder counts. The move duration is set to 420 milliseconds.

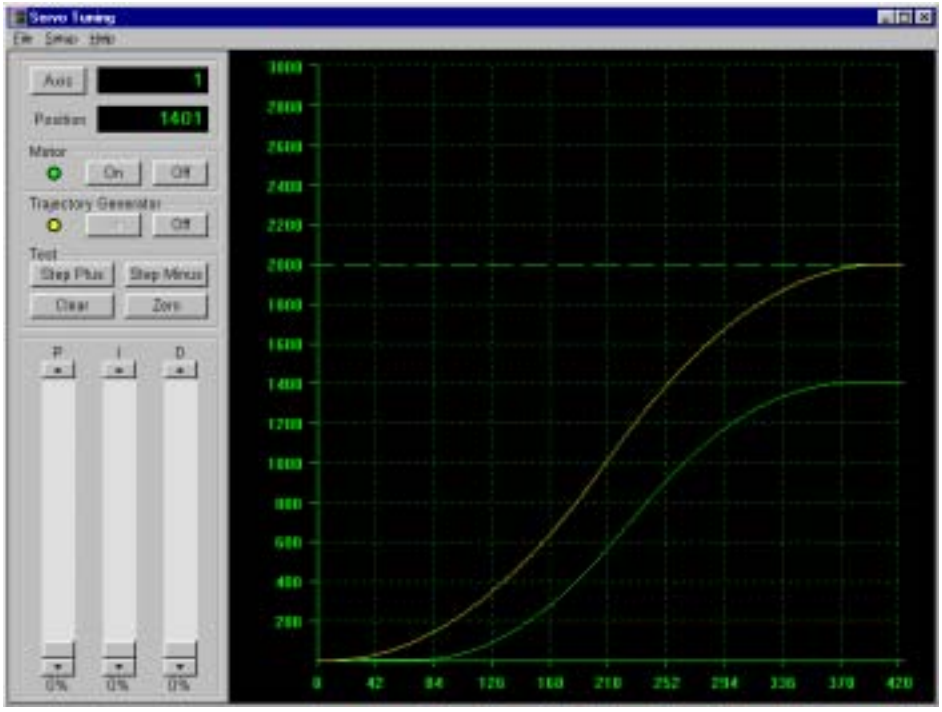
3) Set the Proportional (P), Integral (I), and Derivative (D) slide controls to 0%.

4) Turn on the Trajectory generator

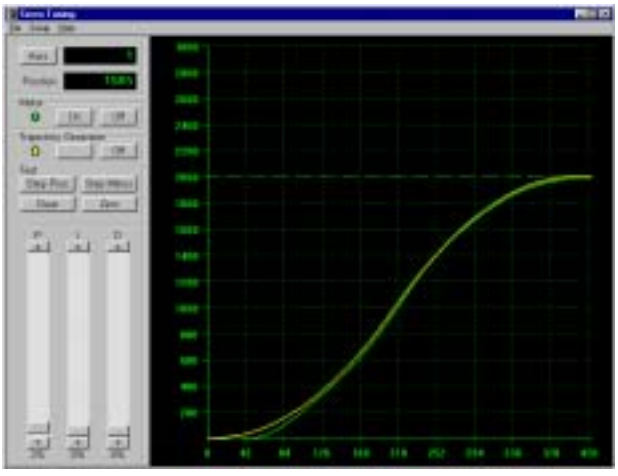
5) Turn the motor on

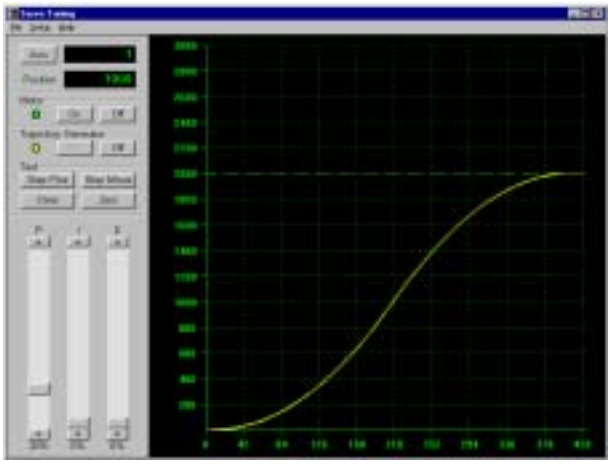
6) Press the Step Plus pushbutton

A response similar to the following graphic should be observed:



Increase the ‘P’ term 1-2 % at a time until the position display indicates that the axis is within +/- 2 counts of the target.

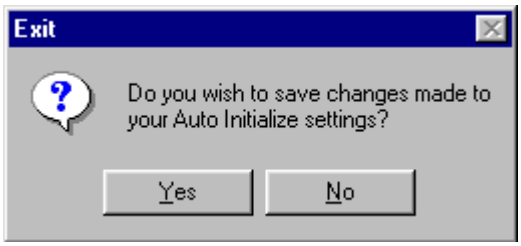




Increase the “I” term 1% at a time until the axis repeatedly positions to the target. If increasing the Integral setting causes the axis becomes unstable:

1) Reduce the Integral Limit setting (Setup – Servo Setup)
2) Reduce the scale of the 'I' term slide control (Setup – PID setup)

Saving the Tuning Parameters
When servo tuning is complete, closing the tuning utility will prompt this message about saving the Auto Initialize setting, selecting **Yes** will store all settings for all installed axes. Selecting **No** will cause all settings to be discarded.



Acceleration and Deceleration Feed Forward
For most applications velocity feed forward is sufficient for accurately positioning the axis. However for applications that require a very high rate of change, acceleration and deceleration gain must be used to reduce the following error at the beginning and end of a move.

Acceleration and deceleration feed forward values are calculated using a similar algorithm as used for velocity gain. The one difference is the velocity is expressed as encoder counts per second, while acceleration and deceleration are expressed as encoder counts per second per second.

DCX output = Accel./Decel. (encoder counts/sec/sec.) **X** Feed forward term (encoder counts * volt/sec./sec.)

Acceleration and deceleration feed forward values should be set prior to using the Servo Tuning Utility to set the proportional and integral gain.

Systems with Electrical or Mechanical Dead Band
Some servo systems may demonstrate significant dead band due to friction, sticktion, or insufficient amplifier drive power. This will typically be indicated when the output command to the servo is relatively high but the axis does not move.

Systems of this type can be very difficult to 'tune'. To overcome the limitations of the system and get the axis moving, the proportional gain would need to be set very high. This will tend to make the system become unstable, causing the axis to 'oscillate' at the end of a move. The ***Output Deadband*** (***aODn***) command is used to compensate for the electrical and or mechanical dead band in a system by modifying the calculated output signal, allowing the module to simulate a 'frictionless' system. The deadband value will be added to a positive output and subtracted from a negative output.

Programming an Output Offset

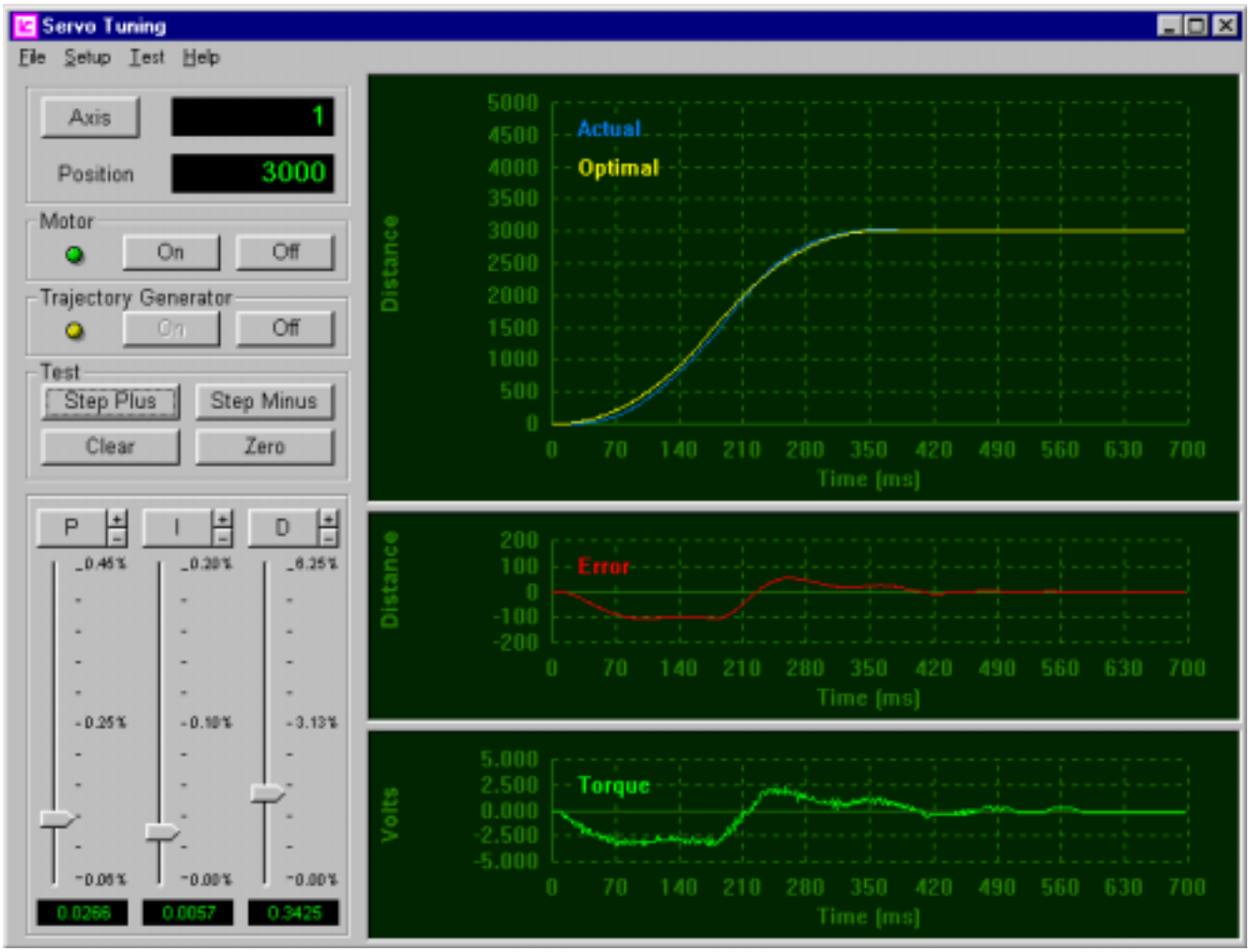
Both the MC300 and MC320 servo modules have output offset adjustment potentiometers for manually setting the zero point of the servo command output. The Output Offset command allows the user to enter a programmable output offset ranging from –10V to +10V.

Moving an Axis

Once the servo is tuned, the axis is ready to perform velocity profile moves. PMC's **Motor Mover** program allows the user to execute absolute, relative, and cycle move sequences, monitor position and status of the axis. By selecting the **Setup** button the user can; set velocity parameters (maximum velocity, acceleration, and deceleration), set velocity profile (Trapezoidal, S curve, or Parabolic), and enable motion limits.

Figure 22: PMC's Motor Mover can be used to move as many as 8 axes simultaneously

By turning on the Trajectory Generator while in the Servo Tuning Utility, its plotting capabilities can be used to display the performance of the axis during a velocity profile move. In this mode two sets of points are plotted. The yellow trace is the optimal position (as calculated by the DCX), the blue trace is the actual position of the axis. The difference between the two plots is the following error (red).


Servo Tuning
File Setup Test Help
Axis
1
Position
3000
Motor
On
Off
Trajectory Generator
On
Off
Test
Step Plus
Step Minus
Clear
Zero
P
I
D
0.0266
0.0057
0.3425
Actual
Optimal
Distance
Time (ms)
Error
Torque
Volts

DCX Stepper Basics

The DCX motion control system supports both open loop and closed loop stepper motion.

Open Loop Stepper Motion
Commanding motion of a stepper motor with no position or velocity feedback is known as 'Open Loop'. To successfully complete the commanded move, the DCX controller counts each step pulse issued to the stepper motor driver. When the position of an axis is queried (by issuing the function ***MCGetPosition ( )*** or MCCL **T**ell **P**osition (***a*TP**) command), the number of pulses issued to the stepper driver is reported. Since there is no position (or velocity) feedback there is no need to 'tune' the axis. However, the axis module must be configured (Trajectory parameters, Velocity Profile, Limits etc...). Please refer to the following stepper setup dialog:

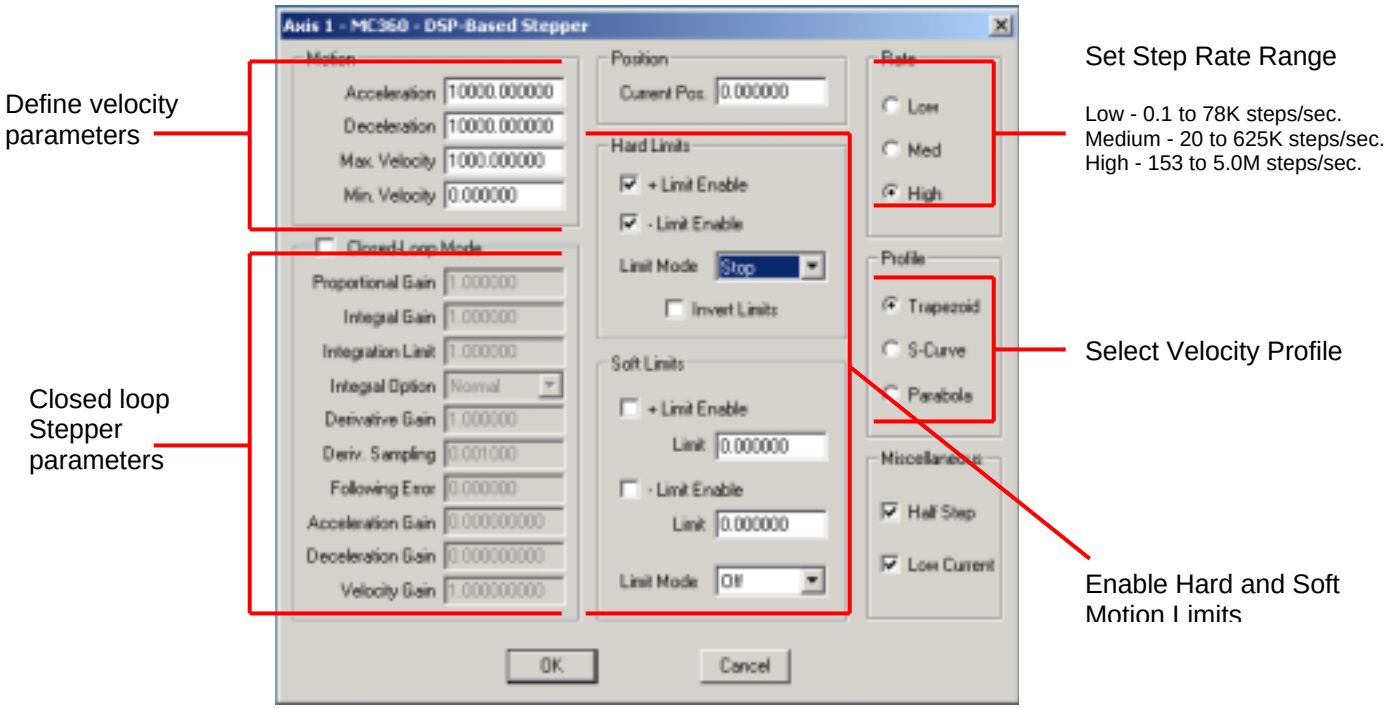


Figure 23: Stepper axis Setup dialog

The Minimum Velocity of a stepper axis must be set to a non zero value. The default value is 1,000 steps per second. The recommended setting of the minimum velocity is from 1% to 10% of the maximum velocity.

Stepper drivers typically use the Direction output from the stepper controller signals to determine the observed direction of motion. If the observed direction of motion is not correct (moving positive causes counter clockwise instead of clockwise rotation) set axis scaling to -1.0.

Closed Loop Steppers

Closed loop stepper control requires a DCX-MC360 stepper module (the DCX-MC362 dual stepper does not support closed loop mode) and MCAPI revision 3.2 or higher.

When configured as a closed loop stepper the DCX-MC360 does not support Position Capture or Position Compare.

The advancements in stepper motor/micro stepping driver technology have allowed many machine builders to maintain ‘servo like’ performance while reducing costs by moving to closed loop stepper systems. While closed loop steppers are still be susceptible to ‘stalling’, they are not plagued with the familiar open loop stepper system problem of loosing steps due to friction (mechanical binding) or system resonance.

For high accuracy stepper applications, the DCX supports closed loop control of stepper motors using quadrature incremental encoders for position feedback. The stepper axis will be controlled as if it is a closed loop servo, the quantity and frequency of step pulses applied to the stepper driver is based on the trajectory parameters of the move and the position error of the axis. Prior to attempting to operate a stepper motor in closed loop mode the basic system components (motor, driver, wiring, and controller) should be verified by moving in open loop mode. For information on operating an open loop stepper please refer to the **DCX Stepper Basics** and **Moving Motors with Motor Mover** sections in this chapter. If the stepper motor does not operate as expected please refer the **Troubleshooting** chapter.

While executing closed loop stepper motion, when the target position equals the current encoder position, the DCX-MC360 step pulse generator (PID filter) will be turned off within 1 micro second.

Unlike a closed loop servo, if the final position of the stepper encoder is beyond the target position of the move **the motor will not be commanded to move back to the target**.

Closed Loop Stepper Setup
There are four steps required to configure a stepper to operate in closed loop :

1) Connect and verify operation of the encoder
2) Define the Encoder / Steps ratio
3) Set the trajectory parameters
4) Tune the axis

Connect and verify the encoder

Connect the stepper motor's encoder to the DCX-MC360 stepper module as shown in the following diagram (for detailed wiring information please refer to the **Connectors, Jumpers, and Schematics** chapter in this manual).. If a single ended encoder is being used inputs A-, B- and Z- are not connected.

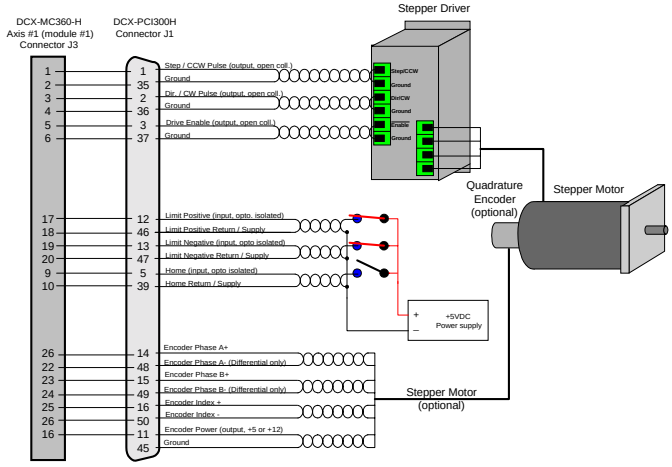


Figure 24: Typical closed loop stepper interconnections

To verify the operation of the encoder open the **Motor Mover** program (Start\Programs\Motion Control\Motion Integrator\Motor Mover). From the **Stepper Setup** dialog select **Closed Loop Mode** and **OK**.:

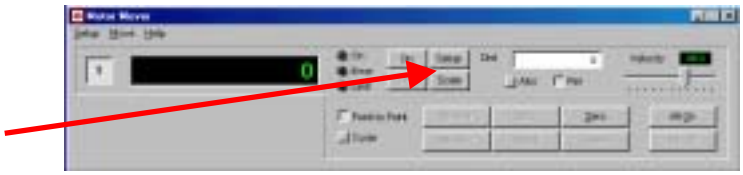


Figure 25:Select Setup to open the dialog

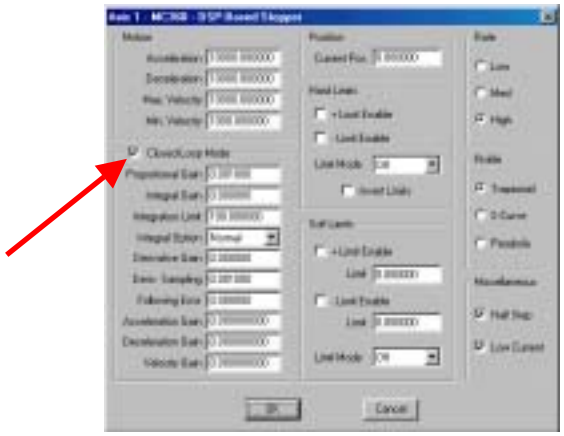


Figure 26:From the Stepper Setup dialog select the Closed Loop Mode check box

When closed loop mode is enabled the **Motor Mover** position readouts will display the position of the

encoder. Rotate the motor / encoder shaft back and forth and verify that the position display changes accordingly.

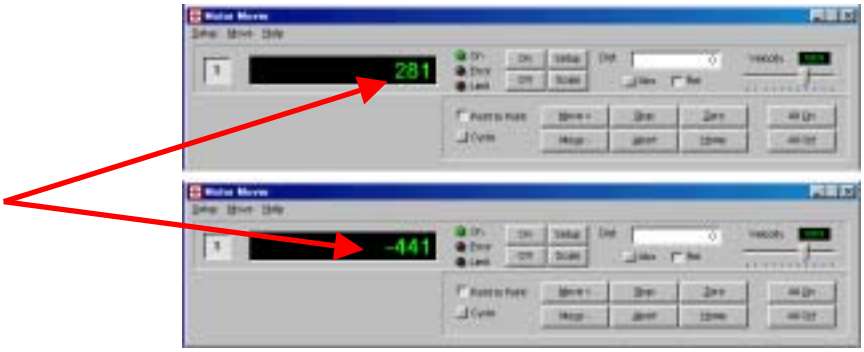


Figure 27: In closed loop mode the Motor Mover position readout displays encoder position

After switching a stepper axis into or out of closed loop mode, use the ***MCEnableAxis ( )*** function to disable and then enable the axis (reinitialize the position registers.

Define the motor steps per rotation / encoder counts per rotation ratio
When operating in closed loop mode, move commands are issued in units of encoder counts. The **EncoderScaling** member of the ***MCFILTEREX*** data structure is used to configure the controller for converting encoder units to step pulses. The value is calculated by dividing motor steps per rotation by encoder counts per rotation. For example, if there 2000 encoder counts per rotation (500 line encoder) and the stepper motor has 51,200 steps per rotation, the Encoder Scaling value would be

EncoderScaling = motor steps per rotation / encoder counts per rotation.
EncoderScaling = 51,200 / 2000
EncoderScaling = 25.6

The Encoder Scale can also be defined from the **Stepper Setup** dialog of the **Servo Tuning** or **Motor Mover** programs.

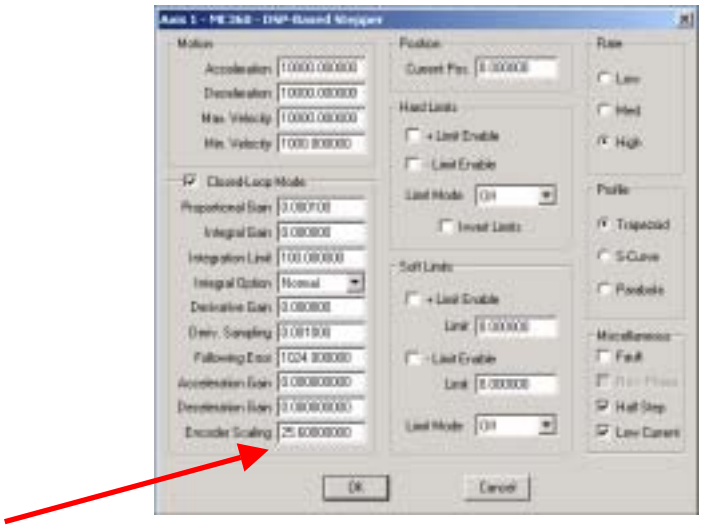


Figure 28:Enter the closed loop steps / encoder scale

Set the trajectory parameters
As with an open loop stepper, the trajectory parameters (maximum velocity, acceleration, deceleration, and minimum velocity) must be set prior to commanding motion. These values can be set using the ***MCMOTION*** data structure or can be entered from the **Stepper Setup** dialog of **Servo Tuning** or **Motor Mover**.

Closed loop stepper trajectory parameters (and move distances) are specified in **encoder units**, not motor step units.

Tune the axis
When a stepper axis is configured for closed loop operation the default proportional gain is set to 0.0001, which should be sufficient to move the axis **near** the specified target. Further adjustments of the proportional and integral gain allow the controller to:

Minimize the following error while moving
Eliminate slow speed slewing of the axis near the end of the move
Settle within 1 encoder count of the target

Use the PMC Servo Tuning program (\Start\Programs\Motion Control\Motion Integrator\Servo Tuning) to tune the closed loop stepper.

Step 1 - Enter a typical move distance (in encoder counts) and move duration (in milliseconds) using the **Test Setup** dialog (Setup\Test Setup).

Step 2 - Verify that the Trajectory Generator is on (yellow LED)

Step 3 - Set the Proportional gain Slide Control Scale 0.20% (**Press P+ zoom button**)

Step 4 - Verify that the Proportional gain is set to 0.0001, Integral and Derivative gain = 0. Generally Derivative gain and Integral gain are not required to tune a closed loop stepper.

Step 5 - From the **Servo Setup** dialog verify that **Closed Loop Mode** is enabled and that the **Encoder Scaling** has been set

Step 6 - Toggle the **Motor Off** and **Motor On** buttons to initialize the closed loop position registers

Step 7 - Start the move with the **Move +** or **Move -** buttons

Step 8 - Observe the plot of following error during the move

Step 9 - Increase the proportional gain and repeat the move until the point of diminishing returns is reached (the following error no longer decreases). Further increases of the proportional gain will tend to cause the motor to emit a grinding noise or stall during a commanded move.

Step 10 - If the axis moves slowly near the end of the move and/or stops a few counts short of the target the Minimum Velocity is probably set too low.

Step 11 - Save the closed loop stepper settings by selecting **Save All Axes Settings** from the Servo Tuning **File** menu. This operation will copy all settings into the MCAPI.INI file so that any windows application program can load axis settings upon opening.

For additional information on using the Servo Tuning program please refer to:

The **Tuning the Servo** section of the **Motion Control** chapter
The Servo Tuning program on-line help

To disable closed loop stepper operation, issue the ***MCSetInoutMode*** function with *Mode* = **MC_IM_OPEN_LOOP** or deselect the closed loop check box in the Servo Tuning Servo Setup dialog..

Reverse Phasing of a closed loop stepper
If the closed loop stepper is reverse phased, issuing a move command will cause the motor to 'take off' in the wrong direction at full torque / speed. Once the position error exceeds the value entered for the allowable following error (default = 1024) a motor error will occur and the axis will stop. To change the phasing either:

- Issuing the ***MCSetServoOutputPhase ( )*** function with *Phase* = **MC_PHASE_REVERSE**
- Selecting the **Reverse Phase** option in the Servo Tuning Servo Setup dialog
- Swap the encoder phase A and B connections to the MC360 module.

Closed loop stepper example
Axis number one is a 51,200 micro steps per rotation stepper motor. A 2,000 count (500 line) incremental encoder is coupled to the stepper motor shaft. The required maximum step rate for this application is 896,000 steps per second (1050 RPM), which requires the axis to be configured for High Speed step range. After verifying the operation of the closed loop stepper from within the Servo Tuning program, save the configuration with the File menu **Save All Axis Settings** option. From a users application program to load the closed loop configuration call the ***MCDLG_RestoreAxis*** function from the PMC Common Motion Dialog Library. To load the closed loop axis configuration from a PMC application program (Servo Tuning or Motor Mover) select **Auto Initialize** from the **File** menu.

Moving Motors with Motor Mover

After defining the step output mode and the step range the axis is ready to execute motion. The Motor Mover program allows the user to execute absolute, relative, and cycle move sequences, monitor position and status of the axis. By selecting the **Setup** button the user can; set velocity parameters (maximum velocity, acceleration, and deceleration), set velocity profile (Trapezoidal, S curve, or Parabolic), and enable motion limits.

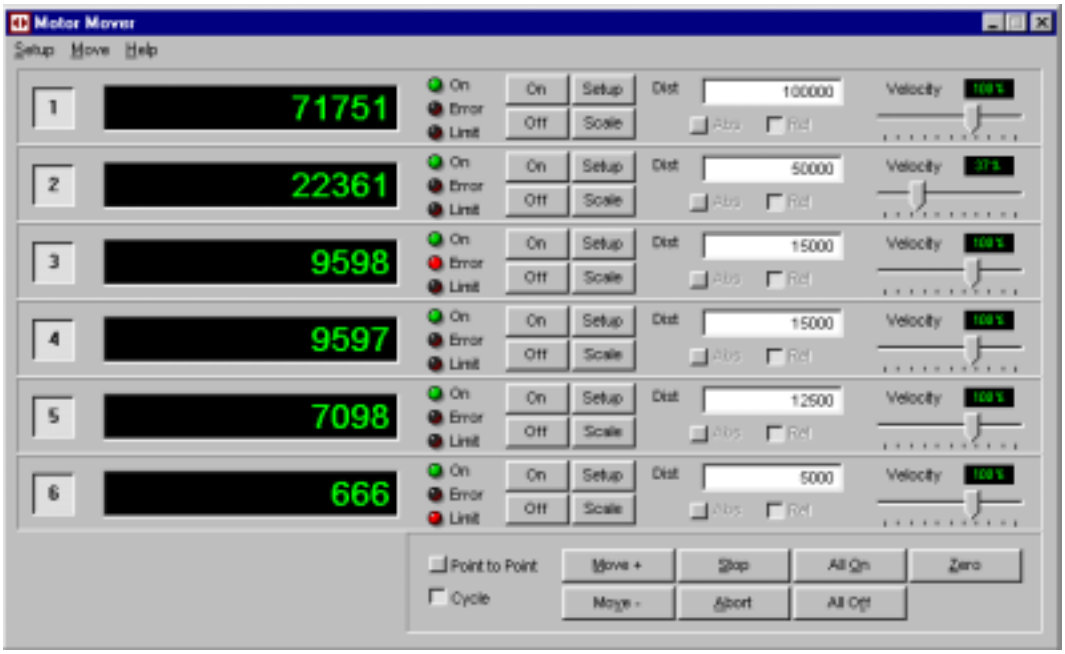


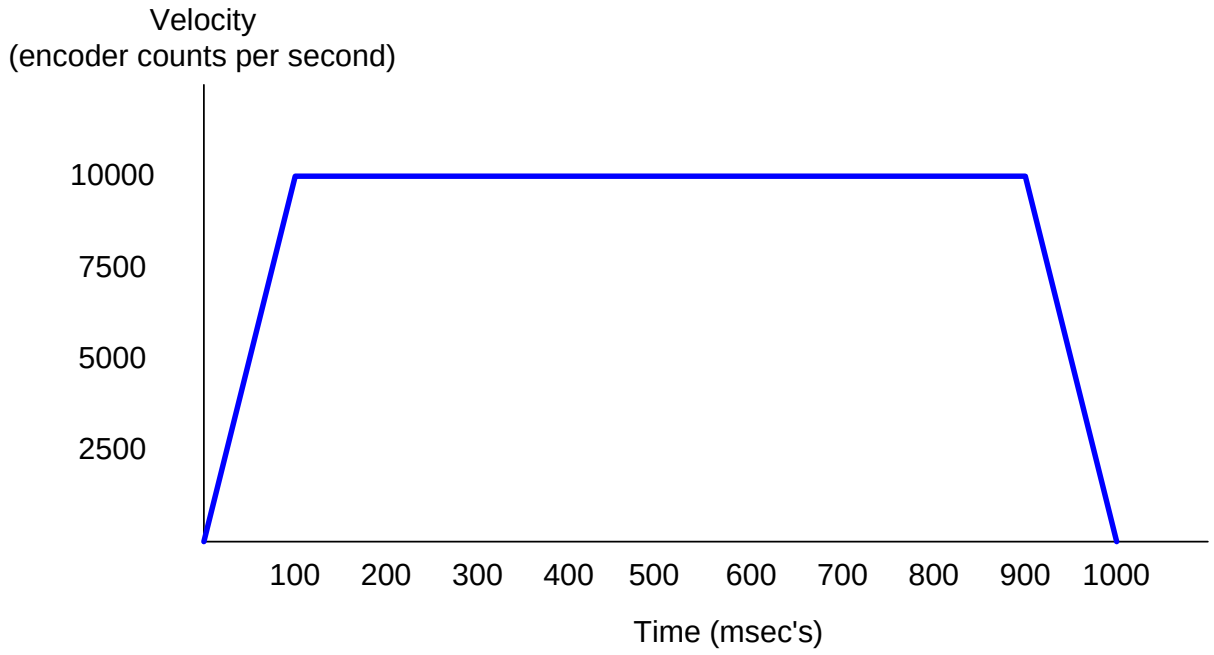
Figure 29: PMC's Motor Mover can be used to command motion for as many as 8 axes simualtaneously

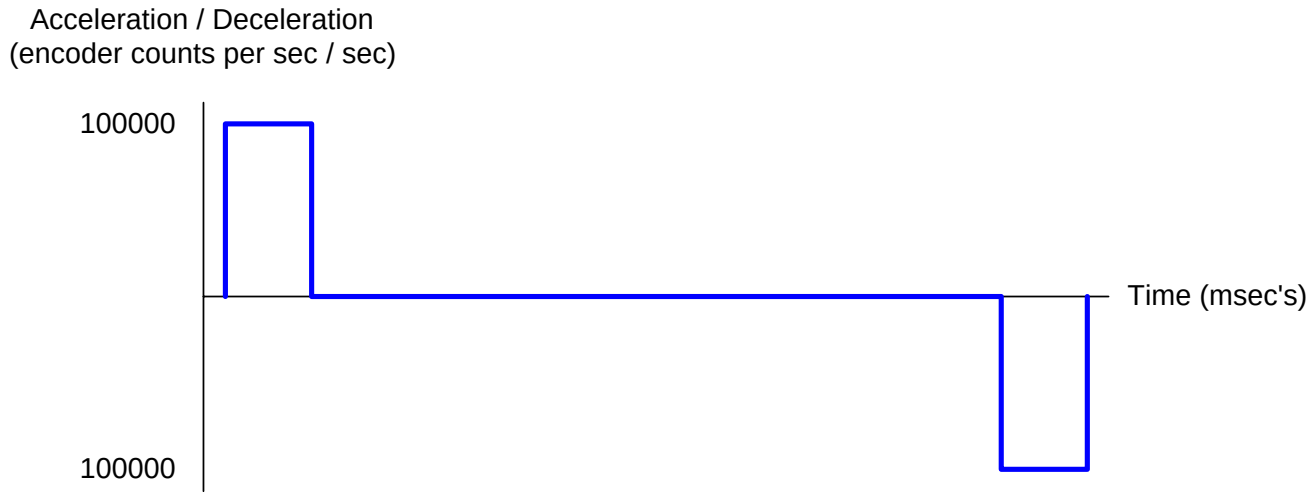
Defining the Characteristics of a Move

Prior to executing any move, the user should define the parameters of the move. The components that make up a move are:

```
// Set axis 1 maximum velocity
// Set axis 1 acceleration
// Set axis 1 deceleration
// Set profile as Trapezoidal
// Set Position mode
// Set target (10000), begin move

MCSetVelocity( hCtlr, 1, 10000.0 );
MCSetAcceleration( hCtlr, 1, 100000.0 );
MCSetVelocity( hCtlr, 1, 100000.0 );
MCSetProfile( hCtlr, 1, MC_PROF_TRAPEZOID );
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_POSITION );
MCMoveRelative( hCtlr, 1, 100000.0 );
```

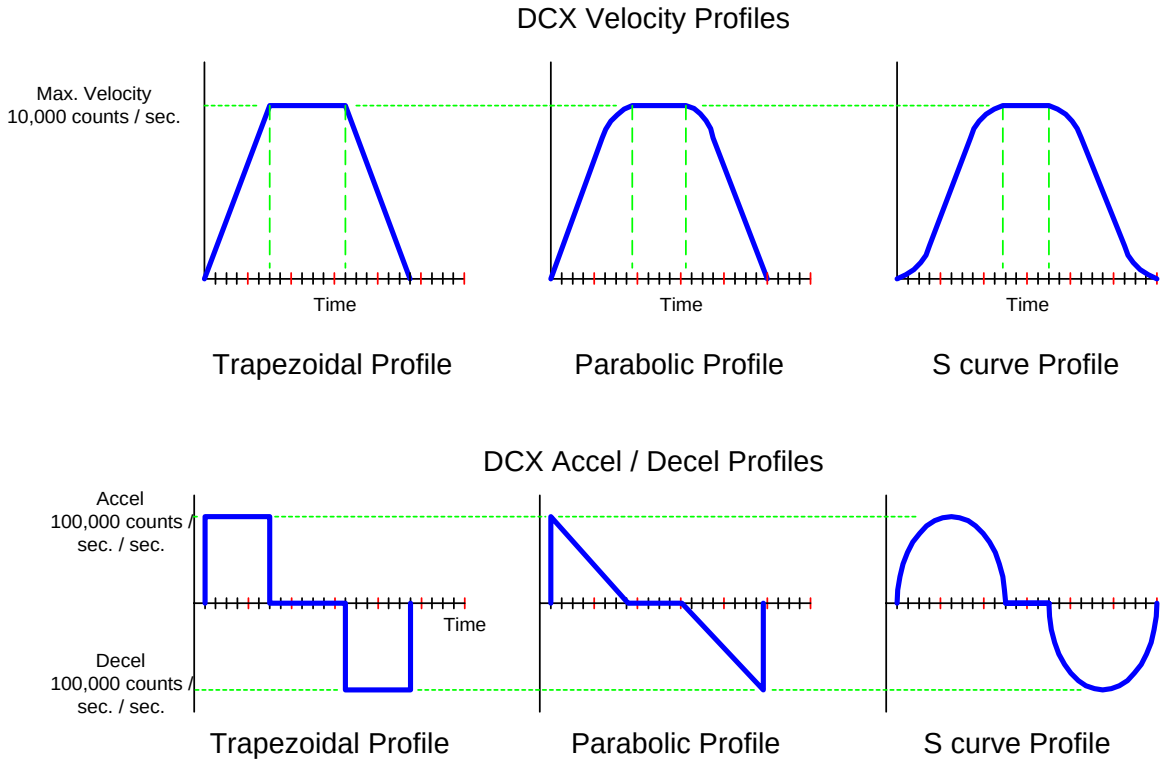
The parameters defined in the program example above specify a move to position 100,000. During the move the velocity will not exceed 10,000 encoder counts per second. A trapezoidal velocity profile will be calculated by the DCX. The rate of change (acceleration and deceleration) will be 100,000 encoder counts per second/per second, there by reaching the maximum velocity (10,000 counts per second) in 100 msec's. The resulting velocity and acceleration profiles follow:





Velocity Profiles

The user can select one of three different velocity profiles that the DCX will then use to calculate the trajectory of a move.



Trapezoidal Profile – (servo & steppers) `MCSetProfile( hCtlr, 1, MC_PROF_TRAPEZOID );`
Shortest time to target when using the same trajectory parameters
Profile most likely to result 'jerk' and/or oscillation
Supports 'on the fly' target changes

Parabolic Profile – (stepper only) `MCSetProfile( hCtlr, 1, MC_PROF_PARABOLIC );`
Slow 'roll off' minimizes lost steps at high velocity
Initial linear rate of change eliminates 'cogging'
On the fly changes will cause the axis to first decelerate to a stop

S curve Profile – (servo only) `MCSetProfile( hCtlr, 1, MC_PROF_SCURVE );`
'True sine' rate of change effectively eliminates 'jerk' and/or oscillation
Longest time to target when using the same trajectory parameters
On the fly changes will cause the axis to first decelerate to a stop

Point to Point Motion

To perform point to point motion of a servo or stepper motor, the following steps are required:

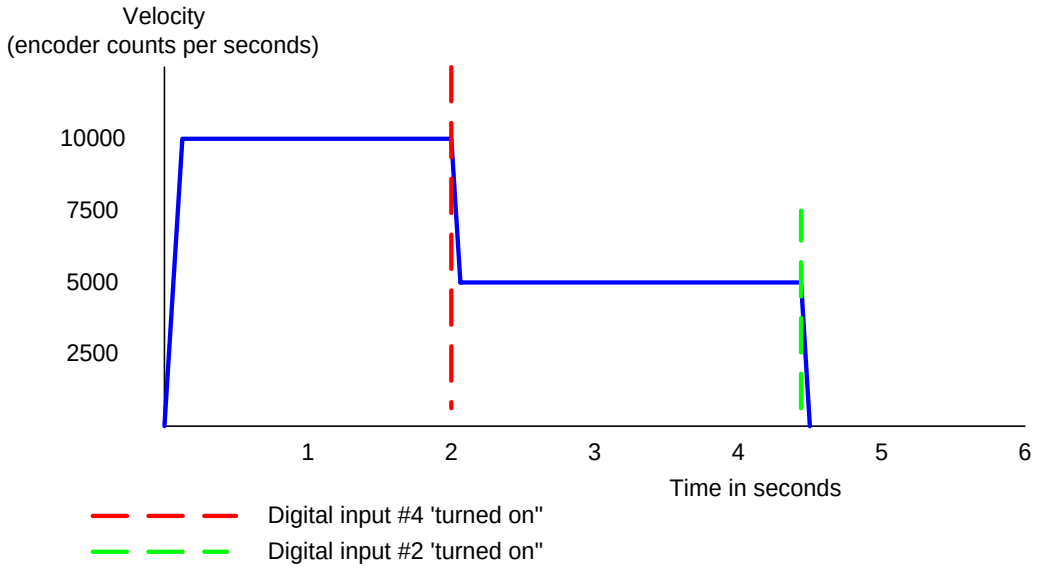
```
// Enable the axis
// Enable Position mode
// Define the velocity profile (trapezoidal, S curve, or parabolic)
// define maximum velocity
// define acceleration
// define deceleration
// execute the move

MCEnableAxis( hCtlr, 1, TRUE );
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_POSITION );
MCSetProfile( hCtlr, 1, MC_PROF_TRAPEZOIDAL );
MCSetVelocity( hCtlr, 1, 10000.0 );
MCSetAcceleration( hCtlr, 1, 25000.0 );
MCSetDeceleration( hCtlr, 1, 50000.0 );
MCMoveRelative( hCtlr, 1, 122.5 );
```

Constant Velocity Motion

To move a servo or stepper at a continuous velocity until commanded to stop:

```
// Enable the axis
// Enable Velocity mode
// Define the velocity profile (trapezoidal, S curve, or parabolic)
// define maximum velocity
// define acceleration
// define deceleration
// define the direction (positive or negative) of the move
// begin motion of axis 1
// wait for digital I/O #4 to be true
// reduce velocity
// wait for digital I/O #2 to be true
// stop the motion of axis 1


MCEnableAxis( hCtlr, 1, TRUE );
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_VELOCITY );
```

```
MCSetProfile( hCtlr, 1, MC_PROF_TRAPEZOIDAL );
MCSetVelocity( hCtlr, 1, 10000.0 );
MCSetAcceleration( hCtlr, 1, 100000.0 );
MCSetDeceleration( hCtlr, 1, 100000.0 );
MCSetDirection( hCtlr, 1, POSITIVE );
MCGo( hCtlr, 3 );
MCWait For DigitalIO( hCtlr, 4, TRUE );
MCSetVelocity( hCtlr, 1, 5000.0 );
MCWait For DigitalIO( hCtlr, 2, TRUE );
MCStop( hCtlr, 1 );
```



Contour Motion (arcs and lines)

The DCX supports Linear Interpolated motion with any combination of two to fifteen axes and Circular Contouring on as many as four groups of two axes. Executing a multi axis contour move requires:

Turn the axes on
Define the axes in the contour group and the controlling axis
Define the trajectory (Vector Velocity, Vector Acceleration and Vector Deceleration)
Define the type of contour move (Linear, Circular, user defined) and the move targets
Loading the Contour Buffer for Continuous Path Contouring

Defining the contour group

The ***MCSetOperatingMode( )*** command is used to define the axes in a contour group. Issue this command to each of the axes in the contour group. The parameter *wMaster* should be set to the lowest axis number of the servo or stepper motor that will be moving on the contour. This axis will then be defined as the 'controlling' axis for the contour group. The following example configures axis 1, 2, and 3 for contour motion with axis #1 defined as the controlling axis.

```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 3, 1, MC_MODE_CONTOUR );
```

Define the trajectory parameters

The ***MCGetContourConfig( )***, ***MCSetContourConfig( )***, and **MCContour** data structure are used to define the trajectory parameters of a contour motion. The default units of the vector velocity are encoder counts or steps per second. The default units of vector acceleration and vector deceleration are encoder counts or steps per second per second. The default units of velocity override is a percentage the setting for vector velocity.

```
//  Motion settings (GetDlgItemDouble() is a helper function defined
//  elsewhere)
//
case IDOK:
   MCGetContourConfig( hCtrlr, iAxis, &Motion );
   Contour.Vector.Accel     = GetDlgItemDouble( hDlg, IDC_TXT_ACCEL );
   Contour.VectorDecel     = GetDlgItemDouble( hDlg, IDC_TXT_DECEL );
   Contour.VectorVelocity     = GetDlgItemDouble( hDlg, IDC_TXT_VELOCITY );
   Contour.VelocityOverride     = GetDlgItemDouble( hDlg, IDC_TXT_MAX_TORQUE);
   MCSetContourConfig( hCtrlr, iAxis, &Motion );
```

Define the type of contour move

The *nMode* parameter of the ***MCBlockBegin( )*** function is used to define the type of contour move to be executed. The following types of contour motion are supported:

nMode parameter	*Contour move type*	*Description*
MC_BLOCK_CONTR_USER	User defined, 1 to 6 axes	Specifies that this block is a user defined contour path motion. *lNum* should be set to the controlling axis number.
MC_BLOCK_CONTR_LIN	Linear interpolated move, 1 to 6 axes	Specifies that this block is a linear contour path motion. *lNum* should be set to the controlling axis number.
MC_BLOCK_CONTR_CW	Clockwise arc, 2 axes	Specifies that this block is a clockwise arc contour path motion. *lNum* should be set to the controlling axis number.
MC_BLOCK_CONTR_CCW	Counter Clockwise arc, 2 axes	Specifies that this block is a counter-clockwise arc contour path motion. *lNum* should be set to the controlling axis number.

Examples of a linear move and a clockwise arc follow:

```
//  Linear move
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_LIN, 1 );
   MCMoveAbsolute( hCtlr, 1, 10000.0 );
   MCMoveAbsolute( hCtlr, 2, 20000.0 );
   MCMoveRelative( hCtlr, 3, -5000.0 );
MCBlockEnd( hCtlr, NULL );


//  Clockwise arc move
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CW, 1 );
   MCArcCenter( hCtlr, 1, MC_CENTER_ABS, 20000.0 );
   MCArcCenter( hCtlr, 2, MC_CENTER_ABS, 0.0 );
   MCMoveAbsolute( hCtlr, 1, 40000.0 );
   MCMoveAbsolute( hCtlr, 2, 0.0 );
MCBlockEnd( hCtlr, NULL );
```

Loading the Contour Buffer for Continuous Path Contouring

The DCX Contour Buffer is used to support Continuous Path Contouring. When a single contour move is executed, the axes will decelerate (at the specified vector velocity) and stop at the target. If multiple contour move commands are issued, the contour buffer allows moves to smoothly transition from one to the other. The vector motion will not decelerate and stop until the contour buffer is empty or an error condition (max following error exceeded, limit sensor ‘trip’, etc...) occurs.

When axis 1 is the controlling axis, up to 256 linear or 128 arc motions (an arc move takes up twice as much buffer space) can be queued up in the Contouring Buffer. If one of the other five axes is the controlling axis, only 16 motions can be queued up. The ***MCGetContouringCount( )*** command will report how many contour moves have been executed since the axes were last configured for contour motion with ***MCSetOperatingMode( )***. The contouring count is stored as a 32 bit value, which means that 2,147,483,647 contour moves can be executed before the contour count will ‘roll over’.

To delay starting contour motion until the contour buffer has been loaded use the ***MCEnableSynch()*** command. This command should be issued to the controlling axis **before** issuing any contour moves. Moves issued after the ***MCEnableSynch( )*** command will be queued into the contour buffer. To begin executing the moves in the buffer, issue the ***MCGoEx( )*** command to the controlling axis . To return to normal operation (immediate execution of contour move commands), issue ***MCEnableSynch( )*** to the controlling axis with the state = FALSE.

Multi Axis Linear Interpolated moves

An example of three linear interpolated moves is shown below. Once the first compound move command is issued, motion of the three axes will start immediately (at the specified vector velocity). The other two compound commands are queued into the contouring buffer. As long as additional contour moves reside in the contour buffer continuous path contour motion will occur. In this example, smooth vector motion will continue (without stopping) until all three linear moves have been completed (the contour buffer has been emptied). At this time the axes will simultaneously decelerate and stop.

```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 3, 1, MC_MODE_CONTOUR );


//  Motion settings (GetDlgItemDouble() is a helper function defined
//  elsewhere)
//
case IDOK:
   MCGetContourConfig( hCtrlr, iAxis, &Motion );
   Contour.Vector.Accel     = GetDlgItemDouble( hDlg, IDC_TXT_ACCEL );
   Contour.VectorDecel     = GetDlgItemDouble( hDlg, IDC_TXT_DECEL );
   Contour.VectorVelocity     = GetDlgItemDouble( hDlg, IDC_TXT_VELOCITY );
   Contour.VelocityOverride     = GetDlgItemDouble( hDlg, IDC_TXT_MAX_TORQUE);
   MCSetContourConfig( hCtrlr, iAxis, &Motion );


//  Linear move #1
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_LIN, 1 );
   MCMoveAbsolute( hCtlr, 1, 85000.0 );
   MCMoveRelative( hCtlr, 2, 12000.0 );
   MCMoveAbsolute( hCtlr, 3, -33000.0 );
MCBlockEnd( hCtlr, NULL );

//  Linear move #2
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_LIN, 1 );
   MCMoveAbsolute( hCtlr, 1, 0.0 );
   MCMoveAbsolute( hCtlr, 2, 0.0 );
   MCMoveAbsolute( hCtlr, 3, 0.0 );
MCBlockEnd( hCtlr, NULL );

//  Linear move #3
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_LIN, 1 );
   MCMoveAbsolute( hCtlr, 1, 5000.0 );
   MCMoveRelative( hCtlr, 2, 23000.0 );
   MCMoveAbsolute( hCtlr, 3, -16000.0 );
MCBlockEnd( hCtlr, NULL );
```

Arc Motion

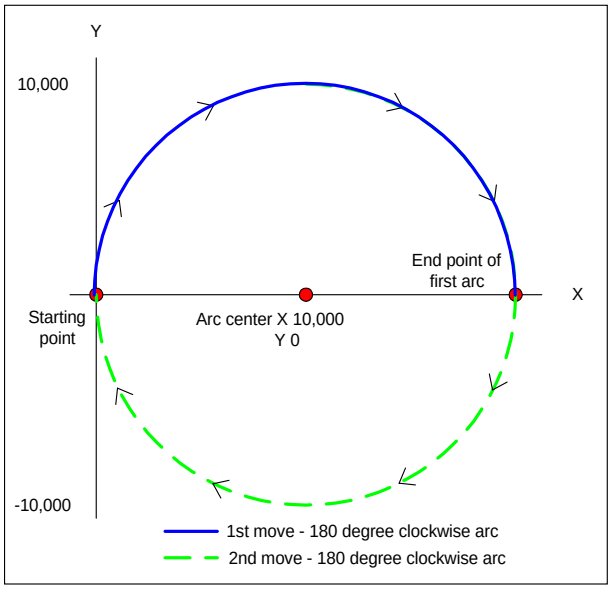
The DCX supports specifying an arc motion in two axes in any of three different ways:

Specify center and end point
Specify radius and end point (not supported by MCAPI)
Specify center and ending angle (not supported by MCAPI)

When the first arc motion is issued, motion of the two axes will start immediately (at the specified vector velocity). Additional contour motions will be queued into the contouring buffer. As long as additional contour moves reside in the contour buffer continuous path contour motion will occur. In this example, smooth vector motion will continue (without stopping) until all both arc motions have been completed (the contour buffer has been emptied). At this time the axes will simultaneously decelerate and stop.

Arc motions by specifying the center point and end point

The ***MCArcCnter( )*** command is used to specify the center position of the arc. This command also defines which two axes will perform the arc motion. The ***MCMoveAbsolute( )*** or ***MCMoveRelative( )*** commands are used to specify the end point of the arc. A spiral motion will be performed if the distance from the starting point to center point is different than the distance from the center point to end point. An example of two arc motions is shown below:



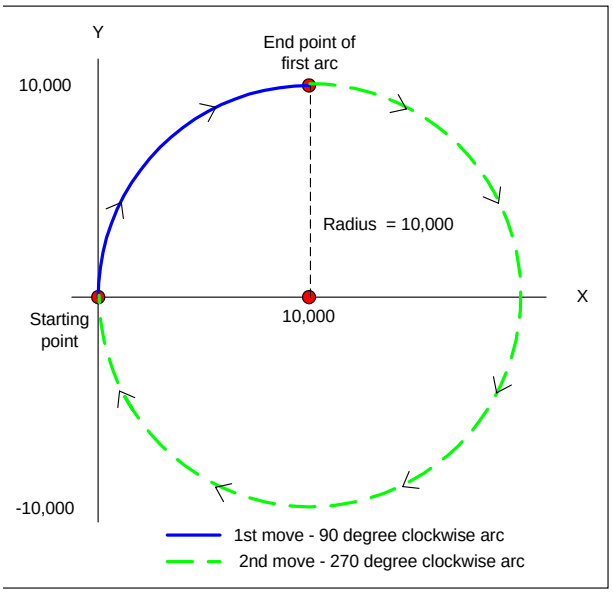

```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );


//  Motion settings (GetDlgItemDouble() is a helper function defined
//  elsewhere)
//
case IDOK:
   MCGetContourConfig( hCtrlr, iAxis, &Motion );
   Contour.Vector.Accel     = GetDlgItemDouble( hDlg, IDC_TXT_ACCEL );
   Contour.VectorDecel     = GetDlgItemDouble( hDlg, IDC_TXT_DECEL );
   Contour.VectorVelocity     = GetDlgItemDouble( hDlg, IDC_TXT_VELOCITY );
   Contour.VelocityOverride     = GetDlgItemDouble( hDlg, IDC_TXT_MAX_TORQUE);
   MCSetContourConfig( hCtrlr, iAxis, &Motion );


//  Clockwise arc move #1
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CW, 1 );
   MCArcCenter( hCtlr, 1, MC_CENTER_ABS, 10000.0 );
   MCArcCenter( hCtlr, 2, MC_CENTER_ABS, 0.0 );
   MCMoveAbsolute( hCtlr, 1, 20000.0 );
   MCMoveAbsolute( hCtlr, 2, 0.0 );
MCBlockEnd( hCtlr, NULL );
```

```
//  Clockwise arc move #2
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CCW, 1 );
   MCArcCenter( hCtlr, 1, MC_CENTER_REL, -10000.0 );
   MCArcCenter( hCtlr, 2, MC_CENTER_REL, 0.0 );
   MCMoveRelative( hCtlr, 1, -20000.0 );
   MCMoveRelative( hCtlr, 2, 0.0 );
MCBlockEnd( hCtlr, NULL );
```

Arc motions by specifying the radius and end point

The ***MCArcRadius( )*** function is used to execute an arc move by specifying the radius and end point of an arc. The *Axis* parameter should equal the controlling axis for the contour move. The parameter *Radius* should equal the radius of the arc. If the arc is greater than 180 degrees, the parameter *Radius* must be expressed as a negative number. The ***MCMoveAbsolute( )*** or ***MCMoveRelative( )*** commands are used to specify the end point of the arc. An example of two arc motions is shown below:

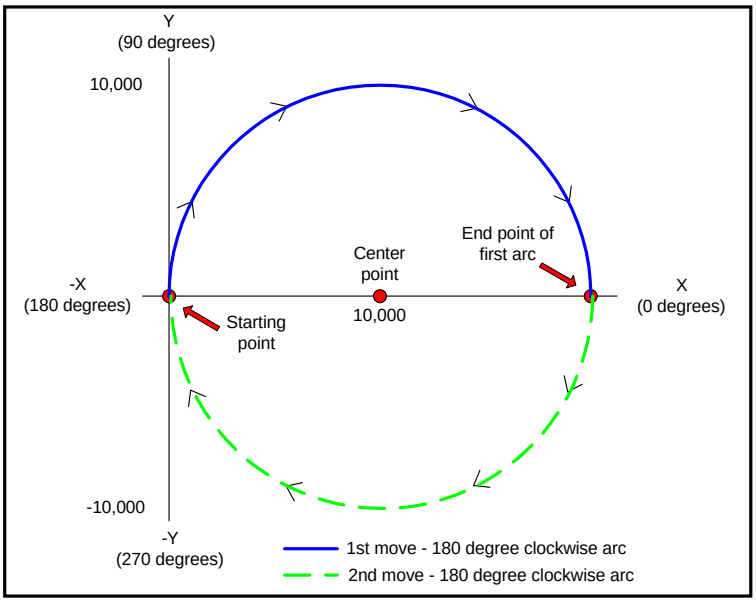
```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );


//  90 degree Clockwise arc move #1
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CW, 1 );
   MCArcRadius( hCtlr, 1, 10000.0 );
   MCMoveRelative( hCtlr, 1, 10000.0 );
   MCMoveRelative( hCtlr, 2, 10000.0 );
MCBlockEnd( hCtlr, NULL );


//  270 degree Clockwise arc move #2
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CW, 1 );
   MCArcRadius( hCtlr, 1, -10000.0 );
   MCMoveRelative( hCtlr, 1, -10000.0 );
   MCMoveRelative( hCtlr, 2, -10000.0 );
MCBlockEnd( hCtlr, NULL );
```



Arc motions by specifying the center point and ending angle

The ***MCArcEndingAngle( )*** function is used to execute an arc move by specifying the ending angle and center point of an arc. The *Axis* parameter should equal the controlling axis for the contour move. The parameter *Angle* should equal the ending angle (absolute or relative) of the arc. When using this method to specify an arc, the ***MCMoveAbsolute( )*** and ***MCMoveRelative( )*** functions are not used. The ***MCArcCenter( )*** function defines the radius of the arc. An example of two arc motions is shown below:

```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );


//  Clockwise arc move #1
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CW, 1 );
   MCArcCenter( hCtlr, 1, MC_CENTER_ABS, 10000.0 );
   MCArcCenter( hCtlr, 2, MC_CENTER_ABS, 0.0 );
   MCArcEndAngle( hCtlr, 1, MC_ABSOLUTE, 0.0 );
MCBlockEnd( hCtlr, NULL );


//  Clockwise arc move #2
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_CW, 1 );
   MCArcCenter( hCtlr, 1, MC_CENTER_REL, -10000.0 );
   MCArcCenter( hCtlr, 2, MC_CENTER_REL, 0.0 );
   MCArcEndAngle( hCtlr, 1, MC_RELATIVE, 1800.0 );
MCBlockEnd( hCtlr, NULL );
```



Changing the velocity 'on the fly'
'On the fly' velocity changes during contour mode motion are accomplished by using the **VelocityOverride** member of the **MCContour** data structure. Issue the command (to the controlling axis) to scale the vector velocity of a linear or arc motion. The rate of change is defined by the current settings for vector acceleration and vector deceleration.

Changing the velocity of a contour group using Velocity Override is not supported for S-curve and/or Parabolic velocity profiles.

Cubic Spline Interpolation of linear moves
To have the DCX perform 'curve fitting' (cubic spline interpolation) on a series of linear moves, issue the ***MCEnableSynch( )*** command to the controlling axis. Next issue linear contour path commands to points on the curve. After loading the desired number of moves into the contour buffer, issue a ***MCGOEx( )*** command with the value *Param* set to 1. Motion will continue from the first to the last point in the contour buffer. To return to normal operation, issue the ***MCEnableSynch( )*** command with parameter *pState* = FALSE.

Note that when performing cubic spline interpolation, only **128 motions** can be queued up if **axis 1 is the controlling axis**. If the controlling axis is not axis one**, only 16 motions** can be queued up in the controller.

User Defined Contour path
When executing contour motion the DCX assumes that the axes are arranged in an orthogonal geometry. The controller will calculate the distance and period of a move as follows:

Beginning position: X=0 Y=0 Z=0
Target position: X=10,000 Y=10,000 Z=1000

Calculated Contour Distance = $\sqrt{(X^2 + Y^2 + Z^2)}$
$= \sqrt{(10{,}000^2 + 10{,}000^2 + 1{,}000^2)}$
$= \sqrt{(100{,}000{,}000 + 100{,}000{,}000 + 1{,}000{,}000)}$
$= \sqrt{201{,}000{,}000}$
$= 14177.44$

The period, or elapsed time of the move, is a simple matter of applying the current settings for Vector Acceleration + Vector Velocity + Vector Deceleration to the Calculated Contour Distance.

For applications where orthogonal geometry is not applicable, the DCX allows the user to define a custom contour distance. This is accomplished by:

1) The command sequence must be preceded by the **C**ontour **P**ath (***a*CP*n***) command (***a*** = the controlling axis) with parameter ***n*** = 0.
2) **C**ontour **D**istance (***a*CD*n***) must be the last command in the compound command sequence, with parameter ***n*** = the Calculated Contour Distance of the move

The DCX will use the current settings for vector velocity, vector acceleration, and vector deceleration to calculate the period of the motion. When a User Defined Contour Path is selected (***MCBlockBegin*** with parameter *nMode* set to **MC_BLOCK_CONTR_USER**), the ***MCContourDistance( )*** function is used to enter the non-orthogonal contour distance.

```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 3, 1, MC_MODE_CONTOUR );


//  User defined move #1
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_USER, 1 );
   MCMoveAbsolute( hCtlr, 1, 1000.0 );
   MCMoveAbsolute( hCtlr, 2, 1000.0 );
   MCMoveAbsolute( hCtlr, 3, 1000.0 );
   MCContourDistance( hCtlr, 1, 10000.0 );
MCBlockEnd( hCtlr, NULL );


//  User defined move #2 - the Distance parameter is 10,000 + 10,000 = 20,000
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_USER, 1 );
   MCMoveAbsolute( hCtlr, 1, 0.0 );
   MCMoveAbsolute( hCtlr, 2, 0.0 );
   MCMoveAbsolute( hCtlr, 3, 0.0 );
   MCContourDistance( hCtlr, 1, 20000.0 );
MCBlockEnd( hCtlr, NULL );
```

For the ***MCContourDistance( )*** function, the parameter *Distance* is an absolute value, relative to the positions of the included axes when the ***MCSetOperatingMode( )*** function was last issued. Re-issuing the ***MCSetOperatingMode( )*** function will reset the current contour distance to zero.

Special case: setting the Maximum Velocity of an Axis
When executing simple point to point or velocity mode motions the maximum velocity of each axis is set individually. When executing multi axis contour moves, the maximum velocity is typically expressed as the velocity of the 'end effector' of the contour group. In a cutting application the 'end effector' would be the tool doing the cutting (torch, laser, knife, etc…). Setting the maximum velocity of an axis in the contoured group is not typically supported.

By combining a user define contour path (***MCBlockBegin*** with parameter *nMode* set to **MC_BLOCK_CONTR_USER**) with the ***MCContourDistance( )*** command with parameter *Distance* = 0, the user can execute multi axis contour moves and limit the maximum velocity of an individual axis. In this mode of operation the ***MCVectorVelocity( )*** command is **not** used to set the velocity of the contour group. The axis with the **longest move time** (calculated by distance, velocity, acceleration, and deceleration) will define the total time for the contour move. For moves of sufficient distance where the axis has enough time to fully accelerate, this one axis will move at its preset maximum velocity. All axes will move at or below their specified maximum velocities. All axes will start and stop at the same time. In the following example, axes 1 and 2 are commanded to move the same distance but the maximum velocity for axis two is 1/3 that of axis one. Since both axes are moving the same distance, they will both travel at a maximum velocity of 100 counts per second.

```
MCSetVelocity( hCtlr, 1, 300.0 );
MCSetAcceleration( hCtlr, 1, 1000.0 );
MCSetDeceleration( hCtlr, 1, 1000.0 );

MCSetVelocity( hCtlr, 2, 100.0 );
MCSetAcceleration( hCtlr, 2, 1000.0 );
MCSetDeceleration( hCtlr, 2, 1000.0 );


MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );

MCContourdistance( hCtlr, 1, 0.0 );


MCBlockBegin( hCtlr, MC_BLOCK_CONTR_USER, 1 );
   MCMoveRelative( hCtlr, 1, 1000.0 );
   MCMoveRelative( hCtlr, 2, 1000.0 );
MCBlockEnd( hCtlr, NULL );
```

If the commanded move distance of axis one was 2000 counts it would move at a higher velocity than axis two, but it would not reach its maximum velocity as set by the **MCSetVelocity( )** command.

Electronic Gearing

The DCX supports slaving any axis or axes to a master. Moving the master axis will cause the slave to move based on the specified slave ratio. The optimal position of the slave axis is calculated by multiplying the optimal position of the master by the gearing ratio of the slave. The slave's optimal position is maintained proportional to the master's position. This can be used in applications where multiple motors drive the same load. Gearing supports both servos and stepper axes, with the master axis operating in jogging, position, velocity, or contouring mode. If a following error or limit error occurs on any of the geared axes (master or slaves) all axes in the geared group will stop.

The MCAPI function ***MCEnableGearing( )*** configures and initiates gearing. The slave ratio can be set to any integer or real value. If the slave ratio is a positive value, a move in the positive direction of the master will cause a move in the positive direction of the slave. If the slave ratio is a negative value, a move in the positive direction of the master will cause a move in the negative direction of the slave. The following program example configures axes 2, 3, and 4 as slaves of axis 1.

```
// Enable gearing of axis 2, 3, and 4
// Move axis 1 (master), slaves (axes 2, 3, and 4) will move at define ratio
MCEnableGearing( hCtlr, 2, 1, 0.5, TRUE );
MCEnableGearing( hCtlr, 3, 1, 12.87, TRUE );
MCEnableGearing( hCtlr, 4, 1, -125, TRUE );
MCMoveRelative (hCtlr, 1, 215.0 );

// disable gearing
MCEnableGearing( hCtlr, 2, 1, 0.5, FALSE );
MCEnableGearing( hCtlr, 3, 1, 12.87, FALSE );
MCEnableGearing( hCtlr, 4, 1, -125, FALSE );
```

Note – if the slave axes are servo's, the **PID parameters** for each axis **must be defined prior** to beginning master/slave operation.

Note – Changing the slave ratio 'on the fly' may cause the mechanical system to 'jerk' or the DCX to 'error out' (following error).

Jogging

In some applications it may be necessary to have a means of manually positioning the motors. Since the DCX is able to control the motion of servos and steppers with precision at both low and high speeds, all that is required to support manual positioning is: .

- A PC with a game port
- A PC joystick
- PC based software that positions the axes in Velocity mode

Jogging without writing software
One of the tools provided with the MCAPI is the Joystick Demo. This tool allows the user to configure and then jog one or two axes.

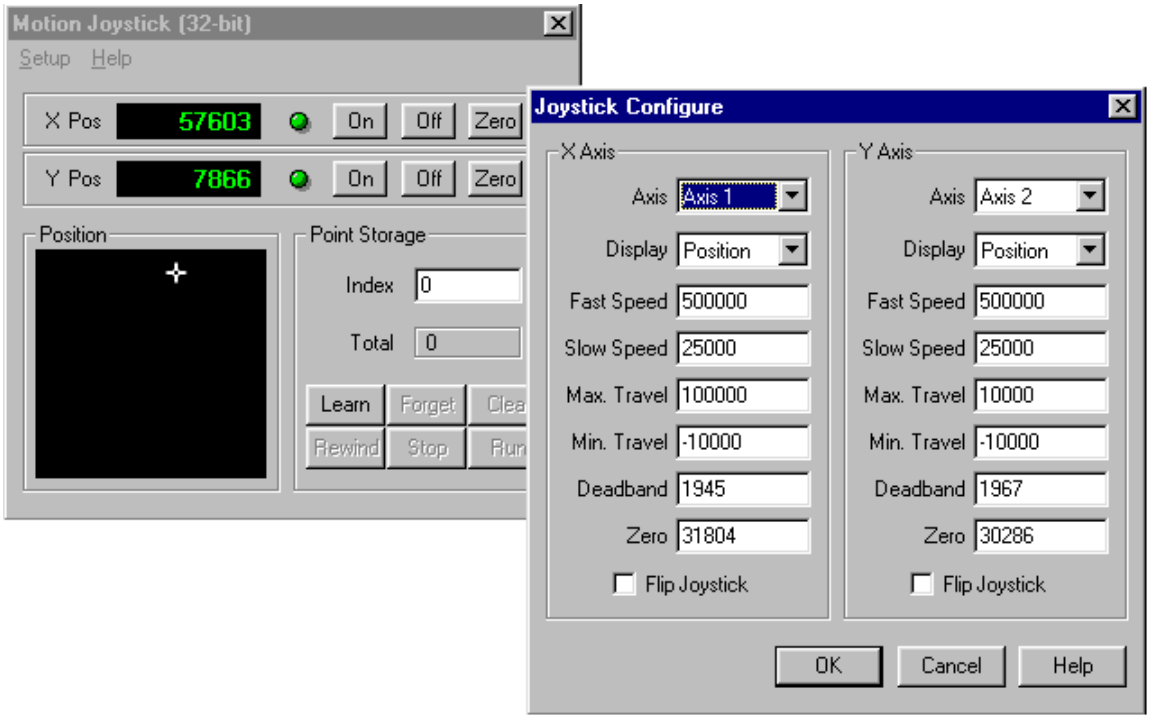
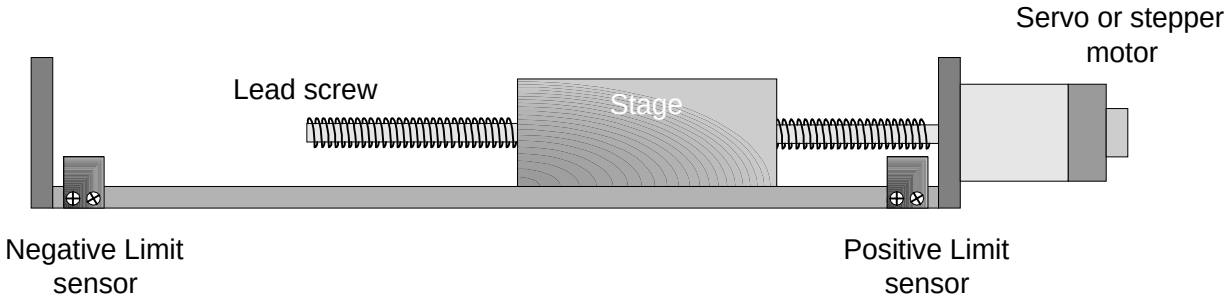


Figure 30: Joystick Demo program

Using the Joystick Demo in your application program
After the MCAPI has been installed the source files for the Joystick Demo are available in the Motion Control folder \Program Files\Motion Control\Motion Control API\Sources\Joy.

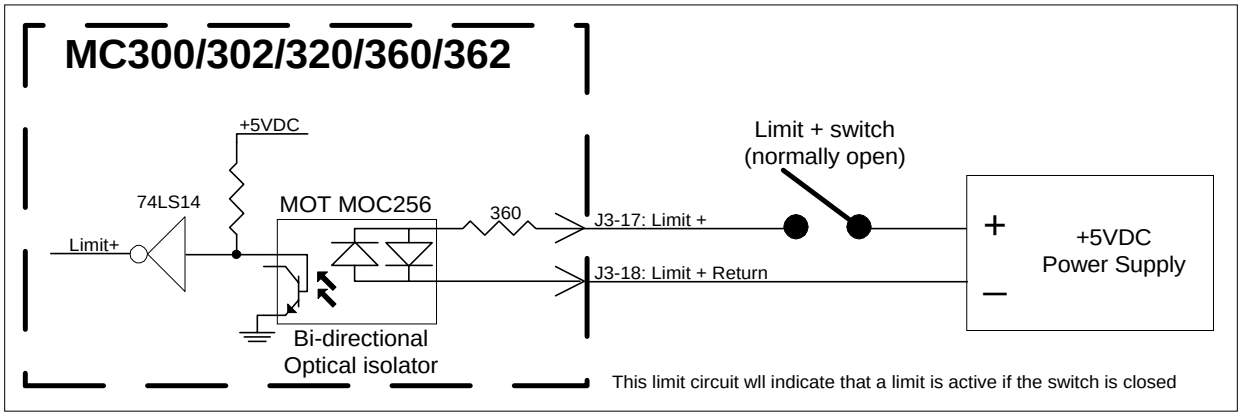
Defining Motion Limits

The DCX Motion Controller implements two types of motion limits error checking. End of travel or 'Hard' limit switch/sensor inputs and 'soft' user programmable position limits.



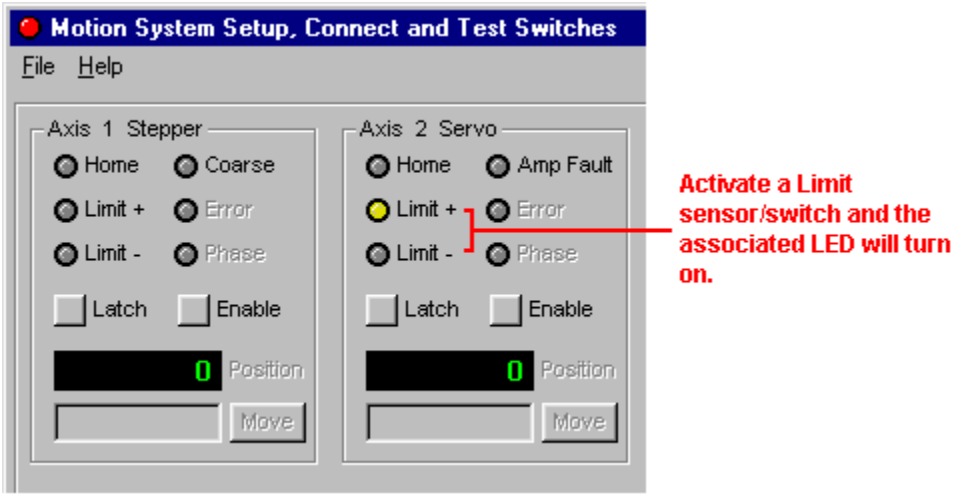
Hard Limits

The Limit + and Limit - inputs of MC3XX motion control modules use bi-directional optical isolators for interfacing to the external limit sensors. The following wiring example details the typical connections for a limit switch.



When limit error checking is enabled by the ***MCSetLimits( )*** function, the limit tripped flags (MC_STAT_PLIM_TRIP and MC_STAT_MLIM_TRIP) indicate an error condition. For a normally closed limit switch, the MC_LIMIT_INVERT parameter must be used to re-define the active level of the limit circuit.

Use the Motion Integrator Motion System Setup Test Panel to test the limit sensors, wiring, and MC3XX operation.



If a normally closed limit sensor circuit is used, the Motion Integrator Test Panel will indicate that the limit sensor is active when the optical isolator (MOC256) is conducting.

The limit LED's of the Motion Integrator Test Panel display the **current state** (MC_STAT_PLIM and MC_STAT_MLIM), not the 'tripped' flag (MC_STAT_PLIM_TRIP and MC_STAT_MLIM_TRIP) of the limit inputs.

The DCX supports two levels of limit switch handling:

Auto axis disable
Simple monitoring

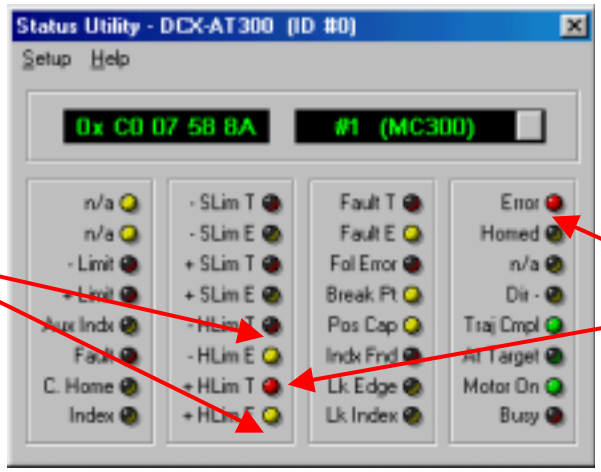
The MCAPI function ***MCSetLimits( )*** allows the user to enable the Auto Axis Disable capability of the DCX. This feature implements a hard coded operation that will stop motion of an axis when a limit switch is active. This background operation requires no additional DCX processor time, and once enabled, requires no intervention from the user's application program. However it is recommended that the user periodically check for a limit tripped error condition using the ***MCGetStatus( )***, ***MCDecodeStatus( )*** functions. The ***MCSetLimit( )*** function provides the following limit flags:

Flag	***Description***
MC_LIMIT_PLUS	Enables the Positive/High hard limit
MC_LIMIT_MINUS	Enables the Negative/Low hard limit
MC_LIMIT_BOTH	Enables the Positive and Negative hard limits
MC_LIMIT_OFF	Turn off the axis when the hard limit input 'goes' active
MC_LIMIT_ABRUPT	Stop the axis abruptly when the hard limit input goes active
MC_LIMIT_SMOOTH	Decelerate and stop the axis when the hard limit input goes active
MC_LIMIT_INVERT	Invert the active level of the hard limit input to high true. Typically used for normally closed limit sensors

When a limit event occurs, motion of that axis will stop and the error flags (MC_STAT_ERROR and MC_STAT_PLIM_TRIP or MC_STAT_MLIM_TRIP) will remain set until the motor is turned back on by ***MCEnable( )***. The axis must then be moved out of the limit region with a move command

(**MCMoveAbsolute( )**, ***MCMoveRelative( )***). The Status Panel screen shot below shows the typical display when a hard limit sensor is tripped during a move.

The yellow +Hlim E and -HLim E LED's indicated that hard coded limit error checking is enabled.



When the red Error and +HLim T or -HLim T LED's are on it indicates that a over travel error has occurred

```
// Set the both hard limits of axis 1 to stop smoothly when tripped, ignore
// soft limits:
//
MCSetLimits( hCtlr, 1, MC_LIMIT_BOTH | MC_LIMIT_SMOOTH, 0, 0.0, 0.0 );


// Set the positive hard limit of axis 2 to stop by turning the motor off.
// Because axis 2 uses normally closed limit switches we must also invert the
// polarity of the limit switch. Soft limits are ignored.
MCSetLimits( hCtlr, 2, MC_LIMIT_PLUS | MC_LIMIT_OFF | MC_LIMIT_INVERT, 0, 0.0,
0.0 );
```

If the user does not want to use the Auto Axis Disable feature, the current state of the limit inputs can be determined by polling the DCX using the ***MCGetStatus( )***, ***MCDecodeStatus( )*** functions. The flag for testing the state of the Limit + input is **MC_STAT_INP_PLIM**. The flag for testing the state of the Limit - input is **MC_STAT_INP_MLIM**.

Soft Limits

Soft motion limits allow the user to define an area of travel that will cause a DCX error condition. When enabled, if an axis is commanded to move to a position that is outside the range of motion defined by the ***MCSetLimit( )*** function, an error condition is indicated and the axis will stop. The ***MCSetLimit( )*** function provides the following limit flags:

Flag	***Description***
MC_LIMIT_PLUS	Enables the High/Positive soft limit
MC_LIMIT_MINUS	Enables the Low/Negative soft limit
MC_LIMIT_BOTH	Enables the High and Low soft limits
MC_LIMIT_OFF	Turn off the axis when the hard limit input 'goes' active
MC_LIMIT_ABRUPT	Stop the axis abruptly when the hard limit input goes active
MC_LIMIT_SMOOTH	Decelerate and stop the axis when the hard limit input goes active

When a soft limit error event occurs, the error flags (**MC_STAT_ERROR** and **MC_STAT_PSOFT_TRIP** or **MC_STAT_MSOFT_TRIP**) will remain set until the motor is turned back on by ***MCEnable( )***. The axis must then be moved back into the allowable motion region with a move command (**MCMoveAbsolute( )**, ***MCMoveRelative( )***).

```
// Assume axis 3 is a linear motion with 500 units of travel. Set the both
// hard limits of this axis to stop abruptly. Set up soft limits that will
// stop the motor smoothly 10 units from the end of travel (i.e. at 10
// and 490).

MCSetLimits( hCtlr, 3, MC_LIMIT_BOTH | MC_LIMIT_ABRUPT, MC_LIMIT_BOTH |
MC_LIMIT_SMOOTH, 10.0, 490.0 );
```

Homing Axes

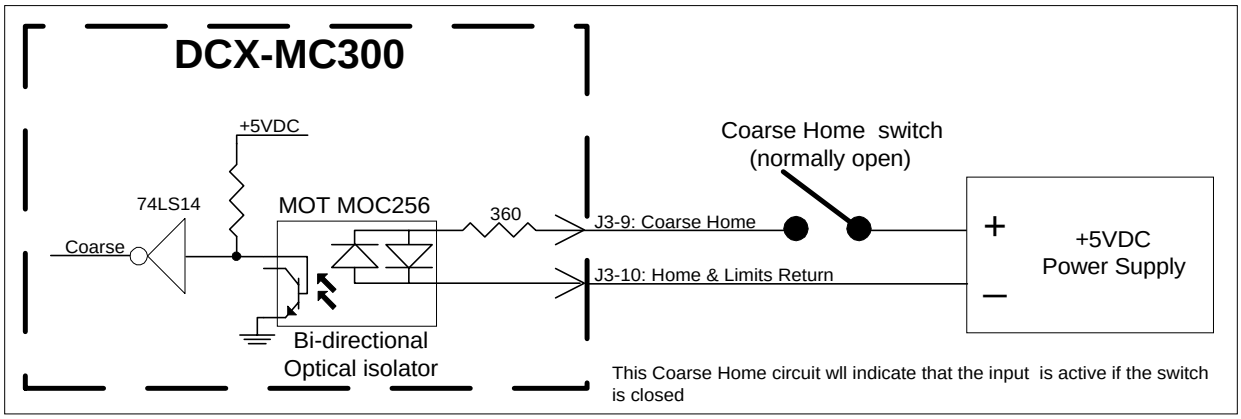
When power is applied or the DCX is reset, the current position of all servo and stepper axes are initialized to zero. If they are subsequently moved, the controller will report their positions relative to the position where they were last initialized. At any time the user can call the ***MCSetPosition*( )** function to re-define the position of an axis.

In most applications, there is some position/angle of the axis (or mechanical apparatus) that is considered 'home'. Typical automated systems utilize electro-mechanical devices (switches and sensors) to signal the controller when an axis has reached this position. The controller will then define the current position of the axis to a value specified by the user. This procedure is called a homing sequence. The DCX is not shipped from the factory programmed to perform a specific homing operation. Instead, it has been designed to allow the user to define a custom homing sequence that is specific to the system requirements. The DCX provides the user with two different options for homing axes:

1) **High level function calls using the MCAPI** - Easy to program homing sequences using MCAPI function calls.

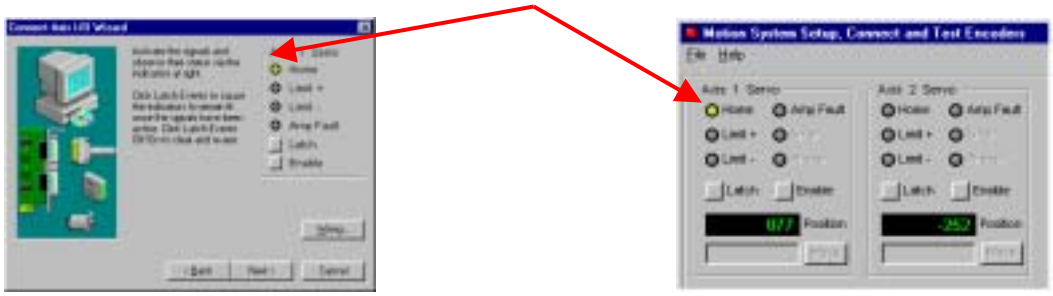
2) **MCCL Homing macro's stored in on-board memory** - When executed as background tasks, MCCL homing macro's allow the user to home multiple axes simultaneously. For additional information on macro's and background tasks please refer to the **DCX-PCI300 MCCL Command Reference manual**.

Connecting a Home Sensor
The Home inputs (Coarse Home - servo's & closed loop steppers, Home – open loop stepper) of MC3XX motion control modules use bi-directional optical isolators for interfacing to the external home sensor. The following wiring example details the typical connections for a Home sensor switch.



Verifying the operation of the Home Sensor
Most motion applications will use a home sensor as a part of the homing sequence. Use Motion Integrator's Connect Axis I/O Wizard or Motion System Setup Test Panel to verify the proper

operation of the encoder index.



Verifying the operation of the Index Mark of an Encoder
Most closed loop system applications will use the Index mark of the encoder to define the 'home' position of a servo. Use Motion Integrator's Connect Encoder Wizard to verify the proper operation of the encoder index.

Programming Homing Routines
The DCX-PCI300 provides sophisticated programming support for homing Closed Loop Servos, Closed Loop Steppers, and Open Loop Steppers. The following two tables summarize which commands are provided for homing operations.

MCAPI homing functions

Axis Type	Module Type	Functions	Input	Notes
Closed Loop Servo	MC300, MC302, MC320	MCIndexArm MCWaitForIndex MCIsIndexFound	Encoder Index	
Closed Loop Servo	MC300, MC302, MC320	MCFindIndex	Encoder Index	Use only from within background task
Closed Loop Stepper	MC360	MCIndexArm MCWaitForIndex MCIsIndexFound	Aux. Encoder Index	
Closed Loop Stepper	MC360	MCFindIndex	Aux. Encoder Index	Use only from within background task
Open Loop Stepper	MC360, MC362	MCIndexArm MCWaitForIndex MCIsIndexFound	Home	
Open Loop Stepper	MC360, MC362	MCFindEdge	Home	Use only from within background task

MCCL homing commands

Axis Type	Module Type	Command	Input	Notes
Closed Loop Servo	MC300, MC302, MC320	IA & WI	Encoder Index	
Closed Loop Servo	MC300, MC302, MC320	FI	Encoder Index	Use only from within background task
Closed Loop Stepper	MC360	IA & WI	Aux. Encoder Index	
Closed Loop Stepper	MC360	FI	Aux. Encoder Index	Use only from within background task
Open Loop Stepper	MC360, MC362	EL & WE	Home	
Open Loop Stepper	MC360, MC362	FE	Home	Use only from within background task

Homing a Rotary Stage (closed loop servo or closed loop stepper) with the Encoder Index
Many servo motor encoders generate an index pulse once per rotation. For a multi turn rotary stage, where one rotation of the encoder equals one rotation of the stage, an index mark alone is sufficient for homing the axis. When an axis need only be homed within 360 degrees no additional qualifying sensors (coarse home) are required.

The following C example uses the ***MCIndexArm( )***, ***MCIsIndexFound( )***, and ***MCWaitForIndex( )*** functions for homing a closed loop system. For complete C code homing samples that can be cut and pasted into an application program please refer to the MCAPI on-line help (MCAPI.HLP).

```
// Arm index and wait for index to be found
//
MCIndexArm( hCtlr, 1, 0.0 );
if (!MCIsIndexFound( hCtlr, 1, 10.0 )) {
    // Index not found within time limit (10 seconds),
    // error handling code goes here
}

//
// Process index and stop motor
//
MCWaitForIndex( hCtlr, 1 );       // controller 'processes' index data
MCStop( hCtlr, 1 );               // stop
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
Sleep( 100 );     // let motor settle 100 msec (WIN32 API function)
```

The following MCCL example uses the **I**ndex **A**rm (***aIAn***) and the Wait for **I**ndex (***a*WI**) commands to home a closed loop system. For complete MCCL homing samples that can be downloaded to the controller and executed please refer to the MotionCD (\PCI300\MCCL\Homing).

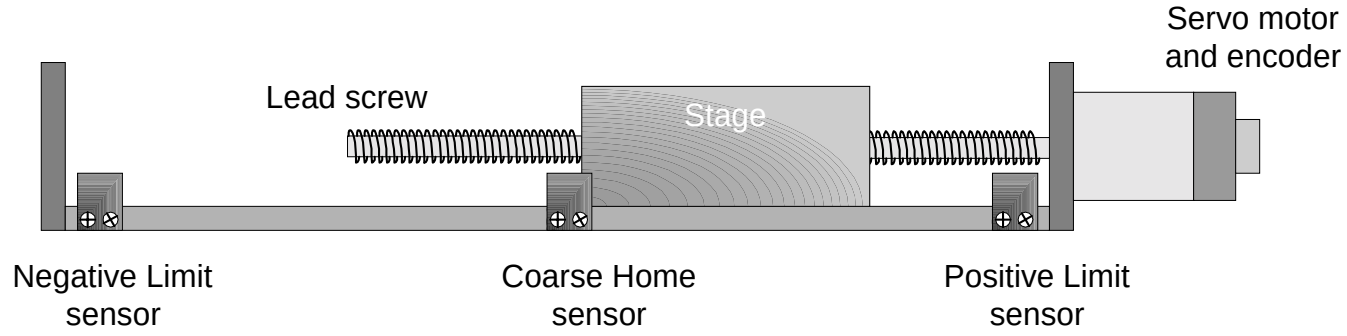
```
;MCCL rotary axis homing sequence index mark
MD10,1IA0,LU"STATUS",1RL@0,IC10,JR-3,NO,1WI,MJ11
                                          ;arm and capture index
MD11,1ST,1WS.01,1PM,1MN,1MA0,1WS.01       ;stop, initialize axis, move to index
                                          ;mark
```

Homing a Closed Loop Axis with Coarse Home and Encoder Index Inputs

A typical axis will incur multiple rotations of the motor/encoder over the full range of travel. This type of system will typically utilize a coarse home sensor to qualify which of the index pulses is to be used to home the axis. The Limit Switches (end of travel) provide a dual purpose:

1) Protect against damage of the mechanical components.
2) Provide a reference point during the initial move of the homing sequence

The following diagram depicts a typical linear stage.



When power is applied or the DCX is reset, the position of the stage is unknown. The axis is commanded to execute a velocity mode move, checking the status of both the Coarse Home sensor and the Limit + sensor. Once the axis is within the Coarse Home sensor the ***MCIndexArm( )***, ***MCIsIndexFound( )***, and ***MCWaitForIndex( )*** functions are used to reference the reported position of the axis to the index mark. The MCEnableAxis( ) function completes the homing operation by reinitialize all position registers. The following flow chart describes a typical homing procedure. If the positive limit sensor is activated the stage will change direction prior to homing the axis.

Homing a Closed Loop System - Encoder Index, Coarse Home Sensor, and Over Travel Limits

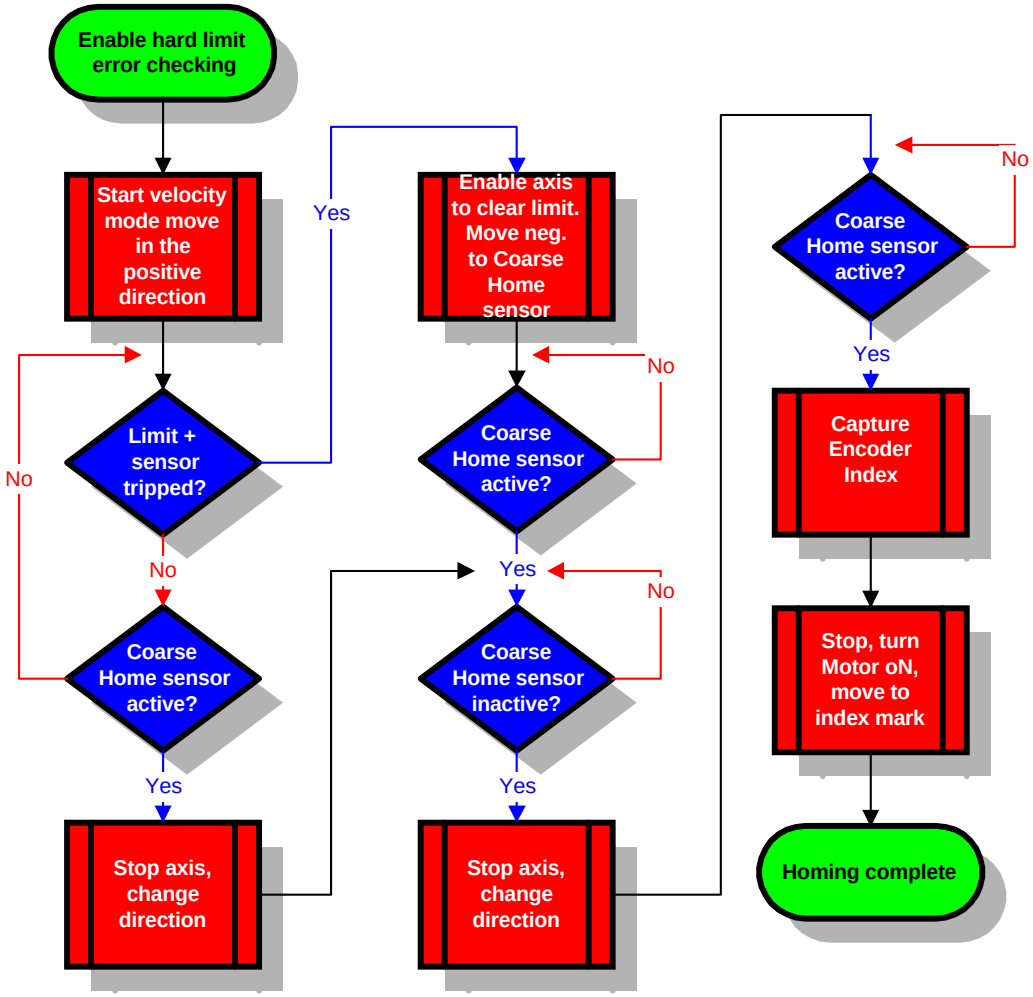


Figure 31: Typical homing routine for a servo

The following C example uses the ***MCIndexArm( )***, ***MCIsIndexFound( )***, and ***MCWaitForIndex( )*** functions for homing a closed loop system. For complete C code homing samples that can be cut and pasted into an application program please refer to the MCAPI on-line help (MCAPI.HLP).

```
// MCAPI linear stage homing sequence using the index mark
//
MCIndexArm( hCtlr, 1, 1000.0 );
if (!MCIsIndexFound( hCtlr, 1, 10.0 )) {
    // Index not found within time limit (10 seconds),
    // error handling code goes here
```

```
}
// Process index and stop motor
MCWaitForIndex( hCtlr, 1 );       // controller 'processes' index data
MCStop( hCtlr, 1 );               // stop
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
Sleep( 100 );     // let motor settle 100 msec (WIN32 API function)


// Move back to location of index mark
//
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_POSITION );
MCEnableAxis( hCtlr, 1, TRUE );
MCMoveAbsolute( hCtlr, 1, 0.0 );
MCIsStopped( hCtlr, 1, 2.0 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
Sleep( 100 );
```

The following MCCL example uses the **I**ndex **A**rm (***aIAn***) and the Wait for **I**ndex (***a*WI**) commands to home a closed loop system. For complete MCCL homing samples that can be downloaded to the controller and executed please refer to the MotionCD (\PCI300\MCCL\Homing).

```
;MCCL linear stage homing sequence using the index mark
MD10,1IA1000,MC20,1WI,1ST,1WS.01,MJ11      ;capture index (position = 1000) then stop
MD11,1PM,1MN,1MA1000,1WS.1                 ;initialize axis, move to index


;homing sub routines
MD20,LU"STATUS",1RL@0,IS10,BK,NO,JR-5          ;test for Index Found
```

Homing a Closed Loop Axis with a Limit sensor

An axis can be homed even if no index mark or coarse home sensor is available. This method of homing utilizes one of the limit (end of travel) sensors to also serve as a home reference.

This method **is not recommended** for applications that require **high repeatability and accuracy**. To achieve the highest possible accuracy when using this method, significantly reduce the velocity of the axis while polling for the active state of the limit input.

The following MCAPI and MCCL sequences will home an axis at the position where the positive limit sensor ‘goes active’:

The following C example uses the ***MCSetPosition( )*** function to redefine the encoder position a closed loop system. For complete C code homing samples that can be cut and pasted into an application program please refer to the MCAPI on-line help (MCAPI.HLP).

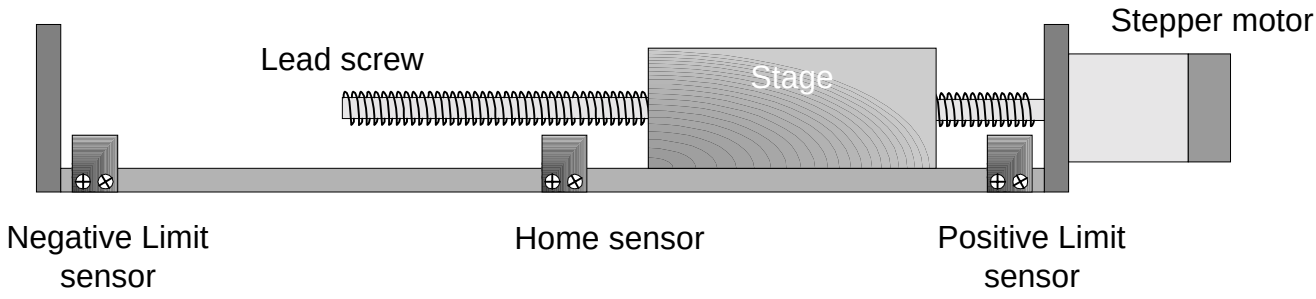
```
// MCAPI homing sequence (using positive limit sensor)
// the axis must have already been moved into (and tripped) the positive limit
// sensor


// Once the positive limit switch is active, move negative until switch is inactive
//
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
MCEnableAxis( hCtlr, 1, TRUE );
MCDirection( hCtlr, 1, MC_DIR_NEGATIVE );
MCSetVelocity( hCtlr, 1, 1000.0 );
MCGoEx( hCtlr, 1, 0.0 ) );
dwStatus = MCGetStatus( hCtlr, 1);
if (!MCDecodeStatus( hCtlr, dwStatus, MC_STAT_INP_PLIM)) {
   dwStatus = MCGetStatus( hCtlr, 1)
}

// Stop the axis and define the leading edge of the limit switch as position 0
//
MCAbort( hCtlr, 1 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
MCSetPosition( hCtlr, 1, 0.0 );
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_POSITION );
MCEnableAxis( hCtlr, 1, TRUE );
MCMoveAbsolute( hCtlr, 1, -100.0 );
```

The following MCCL example uses the **D**efine **H**ome (***aDHn***) command to redefine the encoder position of a closed loop system. For complete MCCL homing samples that can be downloaded to the controller and executed please refer to the MotionCD (\PCI300\MCCL\Homing).

```
; MCCL linear stage homing sequence using the positive limit sensor
MD1,1LM2,1LN3,MJ10                      ;call homing macro
MD10,1VM,1DI0,1GO,LU"STATUS",1RL@0,IS17,MJ11,NO,JR-5
                                        ;move and poll the Limit + sensor
MD11,1WS0.01,1MN,1DI1,1SV1000,1GO,LU"STATUS",1RL@0,IC28,MJ12,NO,JR-5
                                        ;move negative until limit + inactive
MD12,1AB,1WS.1,1DH0,1PM,1MN,1MA-100     ;stop immediately when limit + not active,
                                        ;define position as 0. Move to position –100.
```

Homing open loop steppers

Open loop steppers are typically homed based on the position of a home sensor. Unlike servos that use a precision reference index mark, steppers are more prone to homing inaccuracies due the lower repeatability of the single electro mechanical home sensor. To achieve the highest possible repeatability; reduce the velocity of the axis and always approach the home sensor from the same direction. Here is a typical linear axis controlled by an open loop stepper motor. A home sensor defines the home position of the axis. End of travel or Limit Switches are used to protect against damage of the mechanical components.



When power is applied or the DCX is reset, the position of the stage is unknown. The following command sequence will move the stage in the positive direction. If the positive limit sensor is activated before the Home sensor the stage will change direction, until home sensor is located. When the Home sensor is activated the ***MCEdgeArm ( )*** and ***MCIsEdgeFound ( )*** functions are used to capture the position of the Home sensor active edge.

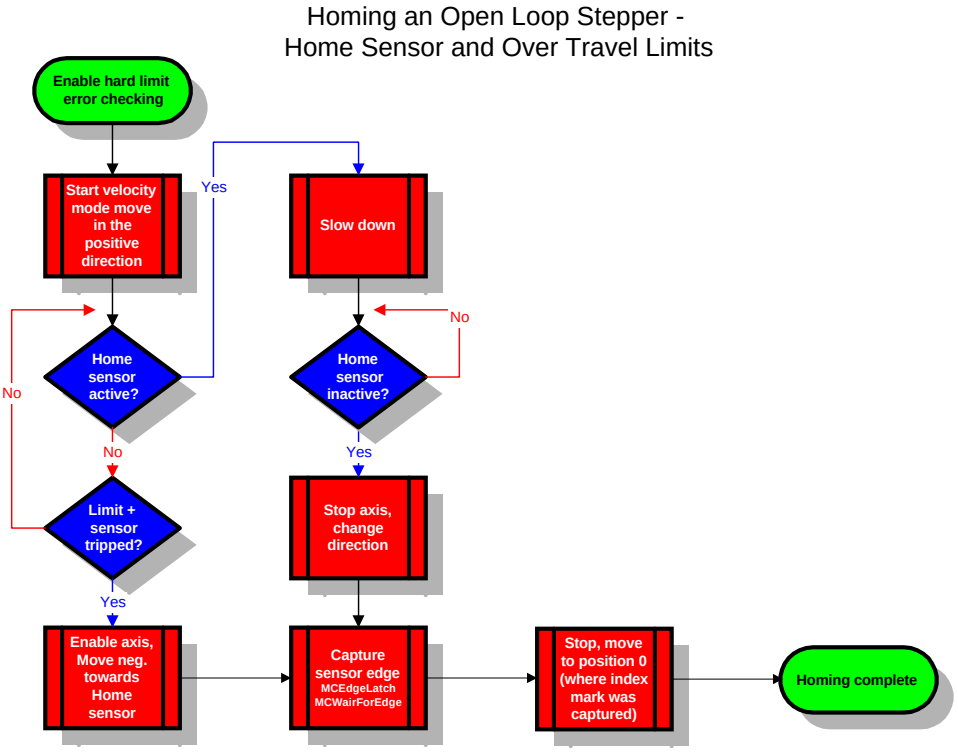


Figure 32: Typical homing routine for a stepper

The following C example uses the ***MCEdgeArm( )***, ***MCIsEdgeFound( )***, and ***MCWaitForEdge( )*** functions for homing a closed loop system. For complete C code homing samples that can be cut and pasted into an application program please refer to the MCAPI on-line help (MCAPI.HLP).

```
// MCAPI open loop stepper linear stage homing sequence using the home sensor
//
MCEdgeArm( hCtlr, 1, 1000.0 );
if (!MCIsEdgeFound( hCtlr, 1, 10.0 )) {
    // Edge not found within time limit (10 seconds),
    // error handling code goes here
}
// Process edge and stop motor
MCWaitForEdge( hCtlr, 1 );       // controller 'processes' edge data
MCStop( hCtlr, 1 );               // stop
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
Sleep( 100 );     // let motor settle 100 msec (WIN32 API function)


// Move back to location of home sensor edge
//
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_POSITION );
MCEnableAxis( hCtlr, 1, TRUE );
MCMoveAbsolute( hCtlr, 1, 0.0 );
MCIsStopped( hCtlr, 1, 2.0 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
Sleep( 100 );
// Enable / disable axis to set MC_STAT_INP_INDEX to monitor the current
// state (not capture & latch) of Home sensor
MCEnableAxis( hCtlr, 1, FALSE );
MCWait( hCtlr, 0.01 );
MCEnableAxis( hCtlr, 1, TRUE );
```

Prior to issuing ***MCEdgeArm ( )*** the status flag **MC_STAT_INP_INDEX** will indicate the current state of the Home Sensor (1 = active, 0 = inactive). After issuing ***MCEdgeArm ( )*** **MC_STAT_INP_INDEX** will be latched when the Home sensor edge has been captured. To clear latching of **MC_STAT_INP_INDEX** issue:

```
MCEnableAxis( hCtlr, 1, FALSE );
MCEnableAxis( hCtlr, 1, TRUE );
```

The following MCCL example uses the **E**dge **A**rm (***aEAn***) and the Wait for **E**dge (***a*WE**) commands to home a closed loop system. For complete MCCL homing samples that can be downloaded to the controller and executed please refer to the MotionCD (\PCI300\MCCL\Homing).

```
; MCCL Stepper linear stage homing sequence using Home & positive limit ;sensors
MD1,1LM2,1LN3,MJ10                          ;enable limits, call homing macro
MD10,1VM,1DI0,1SV10000,1GO,LU"STATUS",1RL@0,IS24,MJ11,NO,IS17,MJ13,NO,JR-8
                                            ;test for sensors (home and +limit)
MD11,LU"STATUS",1RL@0,IC24,MJ12,NO,JR-5     ;continue moving until home sensor off
MD12,1ST,1WS.1,1DI1,1SV5000,1GO,MJ14        ;move back to the home sensor
MD13,1WS0.01,1MN,1DI1,1SV5000,1GO,MJ14      ;move out of limit sensor range
                                            ;back toward the home sensor
MD14,1EL0,MC15,1WE,1ST,1WS.1,1MF,1MN,1PM,1MA-100
                                             ;capture the active edge of the
                                             ;home sensor. Stop axis and
                                             ;define a position 0, ;move to
                                             ;position -100
MD15,LU"STATUS",1RL@0,IS10,BK,NO,JR-5       ;loop status for Edge found bit set
```

Prior to issuing **E**dge **L**atch (***a*EL*n***) the status bit 24 Index / Home will indicate the current state of the Home Sensor (1 = active, 0 = inactive). After issuing **E**dge **L**atch (***a*EL*n***) status bit 24 will be latched when the Home sensor edge has been captured. To clear latching issue:

```
1MF,1MN
```

Homing a Open Loop Stepper with a Limit sensor

An axis can be homed even if no home sensor is available. This method of homing utilizes one of the limit (end of travel) sensors to also serve as a home reference. The following MCAPI and MCCL sequences will home an axis at the position where the positive limit sensor ‘goes active’:

The following C example uses the ***MCSetPosition( )*** function to redefine the encoder position a closed loop system. For complete C code homing samples that can be cut and pasted into an application program please refer to the MCAPI on-line help (MCAPI.HLP).

```
// MCAPI homing sequence (using positive limit sensor)
// the axis must have already been moved into (and tripped) the positive limit
// sensor


// Once the positive limit switch is active, move negative until switch is inactive
//
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
MCEnableAxis( hCtlr, 1, TRUE );
MCDirection( hCtlr, 1, MC_DIR_NEGATIVE );
MCSetVelocity( hCtlr, 1, 1000.0 );
```

```
MCGoEx( hCtlr, 1, 0.0 ) );
dwStatus = MCGetStatus( hCtlr, 1);
if (!MCDecodeStatus( hCtlr, dwStatus, MC_STAT_INP_PLIM)) {
   dwStatus = MCGetStatus( hCtlr, 1)
}

// Stop the axis and define the leading edge of the limit switch as position 0
//
MCAbort( hCtlr, 1 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
MCSetPosition( hCtlr, 1, 0.0 );
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_POSITION );
MCEnableAxis( hCtlr, 1, TRUE );
MCMoveAbsolute( hCtlr, 1, -100.0 );
```

The following MCCL example uses the **D**efine **H**ome (***aDHn***) command to redefine the encoder position of a closed loop system. For complete MCCL homing samples that can be downloaded to the controller and executed please refer to the MotionCD (\PCI300\MCCL\Homing).

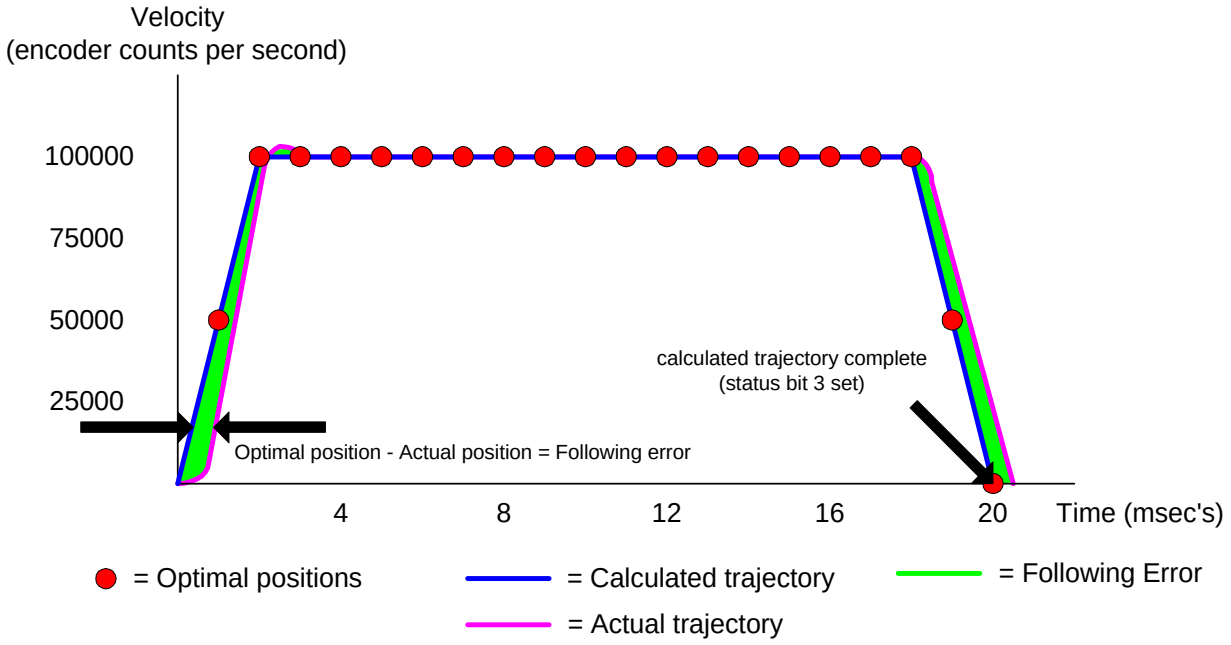
```
; MCCL linear stage homing sequence using the positive limit sensor
MD1,1LM2,1LN3,MJ10                      ;call homing macro
MD10,1VM,1DI0,1GO,LU"STATUS",1RL@0,IS17,MJ11,NO,JR-5
                                        ;move and poll the Limit + sensor
MD11,1WS0.01,1MN,1DI1,1SV1000,1GO,LU"STATUS",1RL@0,IC28,MJ12,NO,JR-5
                                        ;move negative until limit + inactive
MD12,1AB,1WS.1,1DH0,1PM,1MN,1MA-100     ;stop immediately when limit + not active,
                                        ;define position as 0. Move to position –100.
```

Motion Complete Indicators

When the DCX receives a move command, the Trajectory Generator calculates a velocity profile. This profile is based on:

The target position (absolute or relative)
The user defined trajectory parameters (velocity, acceleration, and deceleration)

The velocity profile, as calculated by the DCX trajectory generator, is made up by a series of calculated 'Optimal Positions' that are evenly spaced along the motion path in increments of 1 msec's. These 1 msec optimal positions are passed to the DCX servo modules, which then performs a linear interpolation at the selected servo loop rate.



For a closed loop servo, when the calculated optimal position of an axis is equal to the move target, the calculated 'digital trajectory' of the move has been completed and the **MC_STAT_TRAJ** status flag (MCCL status trajectory complete bit 3) will be set (as shown in the Status Panel graphic below). For a closed loop stepper axis when the encoder position is equal to the move target, the trajectory of the move has been completed and the **MC_STAT_TRAJ** status flag will be set. For an open loop stepper axis when the step count (pulses issued) is equal to the move target, the trajectory of the move has been completed and the **MC_STAT_TRAJ** status flag will be set.

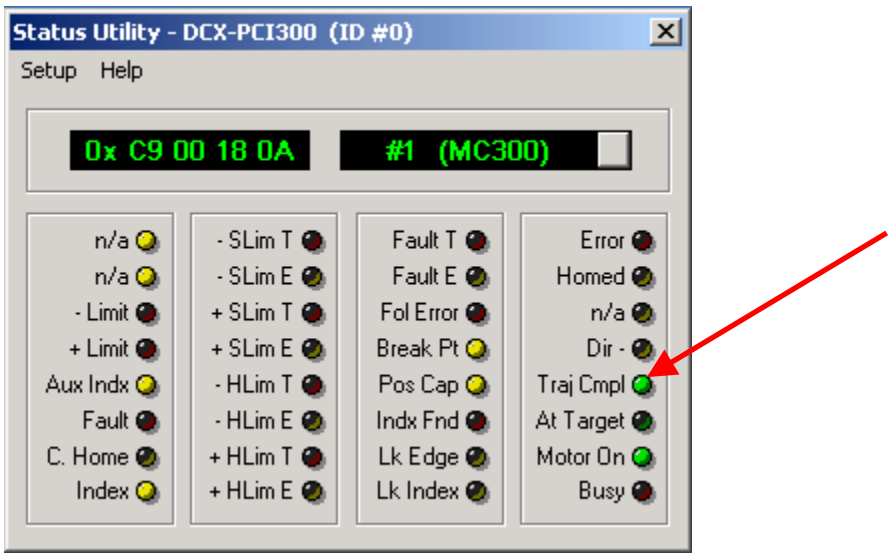


Figure 33: MCAPI Status Panel utility

The **MC_STAT_TRAJ** status flag is the conditional component of the ***MCIsStopped( )*** and ***MCWaitForStop( )*** functions. As shown by the trajectory graph above, the typical lag or following

error during a servo move can cause the **MC_STAT_TRAJ** flag to be set **before the axis has reached its target**. Issuing **MCIsStopped( )** with a timeout value specified or **MCWaitForStop( )** with a *Dwell* time specified allows the user to delay execution move has been completed (following error = 0). In the example below, the **MCIsStopped( )** function (with a 2 second timeout) is used to poll the axis for MC_STAT_TRAJ = true. The Windows SLEEP function is used to allow the axis to stop and settle for 100 milliseconds. command includes a Dwell of 5 msec's, allowing the axis to stop and settle.

```
MCMoveRelative( hCtlr, 2, 500.0 );      // move 500 counts
MCIsStopped( hCtlr, 1, 2.0 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
Sleep( 100 );
```

Another method of indicating the end of a move of a servo is to use ***MCIsAtTarget( )*** or ***MCWaitForTarget( )*** functions. To satisfy the conditions of ***MCIsAtTarget( )*** and ***MCWaitForTarget( )*** , the axis must be within the **Deadband** range (encoder counts +/- or stepper pulses +/-) for the time period specified by **DeadbandDelay**, both of which are defined within the **MCMotion** data structure. The **MC_STAT_AT_TARGET** flag will be set when the conditions for both Deadband and Deadbanddelay have been met.

```
MCMoveRelative( hCtlr, 1, 1250.0 );     // move 1250 counts
MCWaitForTarget( hCtlr, 1, 0.005 );     // wait till MC_STAT_TRAJ set plus
                                        // msec's

MCIsAtTarget( hCtlr, 1, 2.0 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to reach the target within time limit (2 seconds),
    // error handling code goes here
}
Sleep( 100 );
```

On the Fly changes

During a point to point or constant velocity move of one or more axes, the DCX supports 'on the fly' changes of:

- Target
- Maximum Velocity
- Acceleration
- Deceleration
- PID parameters

Changes made to any or all of these motion settings while an axis is moving will take affect within 1 msec.

Note – Changing the PID parameters (Proportional gain, Derivative gain, Integral gain) ‘on the fly’ may cause the axis to jump, oscillate, or ‘error out’.

S-curve or Parabolic velocity profiles:

1) Changing the target position on the fly will cause the axis to **decelerate to a stop before** proceeding to the new target
2) On the fly changes of trajectory parameters (max. velocity, accel, decel) will not be implemented until the current move has been completed

If an “on the fly” target position change requires a change of direction the axis will first decelerate to a stop. The axis will then move in the opposite direction to the new target. This will occur if:

1) The new target position is in the opposite direction of the current move
2) A **‘near target**’ is defined. A near target is a condition where the current deceleration rate will not allow the axis to stop at the new target position. In this case the axis will decelerate to a stop at the user define rate, which will result in an overshoot. The axis will then move in the opposite direction to the new target.

If an on the fly change requires the axis to change direction, the DCX command interpreter will stall, not accepting any additional commands, until the change of direction has occurred (deceleration complete).

Feed Forward (Velocity, Acceleration, Deceleration)

Feed forward is a method in which the controller increases the command output to a servo in order to reduce the following error of an axis. Traditionally feed forward is associated with servo systems that use velocity mode amplifiers, but simple current mode amplifiers used for high velocity and high rate of change applications can also benefit from the use of feed forward.

The basic concept of feed forward is to match the servo command voltage output of the controller to a specific velocity of axis. This essentially adds a user defined offset to the digital PID filter, resulting in more accurate motion by reducing the following error. For example:

The maximum velocity of an axis is 500,000 encoder counts per second. With a typical load applied, the user determines that a servo command voltage of 8.25V will cause the motor to rotate at 500,000 encoder counts per second. The feed forward algorithm used by the DCX to generate the servo command output is:

DCX output = Velocity (encoder counts/sec) **X** Feed forward term (encoder counts/volt/sec.)

with a velocity of 500,000 counts per second at a command input of 8.25V the algorithm will be:

8.25 volts = 500,000 counts/sec. **X** Feed forward term (encoder counts * volt/sec.)

Feed forward = 8.25V / 500,000 counts per sec.

0.0000165 = 10 volts / 100,000 counts per sec.

```
1VG0.0000165                          ;set velocity gain (velocity feed
                                      ;forward ) with MCCL command

// set velocity gain (velocity feed forward) using MCAPI function
//
   MCGetFilterConfig( hCtrlr, iAxis, &Filter );
   Filter.VelocityGain     = ( hCtlr, 1, 0.0000165 );
   MCSetFilterConfig( hCtrlr, iAxis, &Filter );
```

An axis that has been tuned without feed forward will need to be **re-tuned** when the feed forward has been changed to a non zero value.

See the description of Tuning a Velocity Mode amplifier in the **Tuning the Servo** section of the **Motion Control** chapter

When feed forward is incorporated into the digital PID filter it becomes the primary component in generating the servo command output voltage. Typically the setting of the other terms of the filter will be:

Proportional gain – reduced by 25% to 50%
Integral gain – reduced by 5% to 25%
Derivative gain – set to zero, if the axis is too responsive reduce the gain of the amplifier

Acceleration and Deceleration Feed Forward

For most applications, velocity feed forward is sufficient for accurately positioning the axis. However for applications that require a very high rate of change, acceleration and deceleration gain must be used to reduce the following error at the beginning and end of a move.

Acceleration and deceleration feed forward values are calculated using a similar algorithm as used for velocity gain. The one difference is the velocity is expressed as encoder counts per second, while acceleration and deceleration are expressed as encoder counts per second per second.

DCX output = Accel./Decel. (encoder counts/sec/sec.) * Feed forward term (encoder counts * volt/sec./sec.)

Acceleration and deceleration feed forward values should be set prior to using the Servo Tuning Utility to set the proportional and integral gain.

Save and Restore Axis Configuration

The MCAPI Motion Dialog library includes ***MCDLG_SaveAxis( )*** and ***MCDLG_RestoreAxis( )***. These high level dialogs allow the programmer to easily maintain and update the settings for servo and stepper axes.

MCDLG_SaveAxis( ) encodes the motion controller type and module type into a signature that is saved with the axis settings. ***MCDLG_RestoreAxis( )*** checks for a valid signature before restoring the axis settings. If you make changes to your hardware configuration (i.e. change module types or controller type) ***MCDLG_RestoreAxis( )*** will refuse to restore those settings.

You may specify the constant **MC_ALL_AXES** for the wAxis parameter in order to save the parameters for all axes installed on a motion controller with a single call to this function.

If a NULL pointer or a pointer to a zero length string is passed as the *PrivateIniFile* argument the default file (MCAPI.INI) will be used. Most applications should use the default file so that configuration data may be easily shared among applications. Acceptance of a pointer to a zero length string was included to support programming languages that have difficulty with NULL pointers (e.g. Visual Basic).

Chapter Contents

- Auxiliary Encoders
- Backlash Compensation
- Emergency Stop
- Encoder Rollover
- User Defined Filters (Notch, Low Pass, High Pass, and Band Pass)
- Flash Memory Firmware Upgrade
- Initializing and Restoring Controller Configuration
- Learning/Teaching Points
- Building MCCL Macro Sequences
- MCCL Multitasking
- Pause and Resume Motion
- Position Capture
- Position Compare
- Reassigning Axis Numbers
- Record and Motion Data
- Manually Resetting the DCX
- Single Stepping MCCL Programs
- Tangential Knife Control
- Threading Operations
- Torque Mode Output Control
- Turning off Integral gain during a move
- Upgrading from a DCX-AT200 motion control system
- Defining User Units
- DCX Watchdog

Chapter

6

Application Solutions

Auxiliary Encoders

Dual axis modules (DCX-MC302, DCX-MC362) do not support auxiliary encoders.

Servo systems typically use an encoder for position feedback. The encoder is usually mounted to the motor housing and the glass scale of the encoder is coupled directly to the shaft of the motor. This direct coupling provides the DCX with position feedback of the motor shaft, allowing the controller to position the shaft of the motor independent of external mechanical inaccuracies (slipping belts, gear backlash, lead screw runout).

However the 'task at hand' of most motion control applications is not to rotate the shaft of a motor, it is to automate a manual operation. To accomplish this, the shaft of the motor is connected to the external mechanics that will actually be doing the work. Take for example a pick and place machine with axes X, Y, and Z. Due to a myriad of gears, pulleys, belts, and lead screws there may be no more than a 'loose' association between the motor shaft of the X axis and the actual position of the X axis' 'end effector'. This is where an auxiliary encoder can be used to significantly improve the positioning accuracy of a servo or stepper system.

Servo Axes with Auxiliary Encoders
An auxiliary encoder is required when the user must reposition an axis to compensate for the discontinuity between the motor shaft and the mechanics that position the 'end effector'.

While similar in connections, the operation and configuration of a servo and auxiliary encoder is significantly different from a Dual Loop Servo. For a description, please refer to the **Dual Loop Servo** section of the **Motion Control** chapter.

Typically an auxiliary encoder is added to a closed loop servo to allow the user to retrieve the position of the 'end effector' **at the end of a move**. The position of the auxiliary encoder is not a component of the servo command output as calculated by the digital PID filter. The auxiliary encoder is used to determine whether or not the axis is properly positioned.

```
// After a move compare the target and auxiliary encoder position.
// If short of the target, execute a move = the difference of the target &
// encoder position

MCMoveAbsolute( hCtlr, 1, 1675.5 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
if (MCGetTargetEx( hCtlr, 2, &Target ) == MCERR_NOERROR )
if (MCGetAuxEncPosEx( hCtlr, 1, &Position ) == MCERR_NOERROR)
   if (Position < 1674.0)
      (Target – Position = AuxEncDiff)
      MCMoveRelative( hCtlr, 1, AuxEncDiff );
      if (!MCIsStopped( hCtlr, 1, 2.0 )) {
          // Motor failed to stop within time limit (2 seconds),
          // error handling code goes here
      }
 if (MCGetAuxEncPosEx( hCtlr, 1, &Position ) == MCERR_NOERROR)
 if (Position < 1675.0)
             . . .  // print error message
```

Open Loop Stepper Axes with Auxiliary Encoders

An auxiliary encoder may be used in conjunction with a stepper motor to provide verification of a move. The advantages of an open loop stepper over a closed loop axis are:

The output pulse train of an open loop system is much more stable
Easier to configure - open loop systems require no tuning

Typically an encoder is added to an open loop stepper to allow the user to retrieve the encoder position **at the end of a move**. The reported position of the auxiliary encoder is used to determine whether or not the axis is properly positioned.

```
// After a move compare the target and auxiliary encoder position.
// If short of the target, execute a move = the difference of the target &
// encoder position

MCMoveAbsolute( hCtlr, 1, 122.5 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
if (MCGetTargetEx( hCtlr, 2, &Target ) == MCERR_NOERROR )
if (MCGetAuxEncPosEx( hCtlr, 1, &Position ) == MCERR_NOERROR)
   if (Position < 122.0)
      (Target – Position = AuxEncDiff)
      MCMoveRelative( hCtlr, 1, AuxEncDiff );
      if (!MCIsStopped( hCtlr, 1, 2.0 )) {
         // Motor failed to stop within time limit (2 seconds),
         // error handling code goes here
      }
if (MCGetAuxEncPosEx( hCtlr, 1, &Position ) == MCERR_NOERROR)
   if (Position < 122.0)
```

```
                . . .  // print error message
```

For information about closed loop stepper motion, please refer to the **Closed Loop Steppers and Homing Axes** sections of the **Motion Control** chapter.

Homing the Auxiliary Encoder

The auxiliary encoder of a servo or stepper may be homed in one of two ways:

Home the encoder using the Auxiliary Encoder Index input
Re-define the position of the auxiliary encoder when the primary axis position is initialized

If the encoder includes an index mark output it is recommended that this signal be used to home both the reported position of the axis and the auxiliary encoder. The repeatability of a system homed using the index mark will be significantly better than that of a system that uses a mechanical switch/electromechanical sensor. The following programming example will reference both the reported position of an open loop stepper and the auxiliary encoder at the location of the Index mark:

The following C example uses the ***MCFindAuxEncIdx( )***, ***MCSetAuxEncPos ( ), and MCSetPosition ( )*** functions to redefine the step count and encoder position an open loop stepper with an auxiliary encoder. For complete C code homing samples that can be cut and pasted into an application program please refer to the MCAPI on-line help (MCAPI.HLP).

```
MCFindAuxEncIdx( hCtlr, 1, 0.0 );
dwStatus = MCGetStatus( hCtlr, 1 );
while (! MCDecodeStatus( hCtlr, dwStatus, MC_STAT_INP_AUX ))
       dwStatus = MCGetStatus( hCtlr, 1 );
MCStop( hCtlr, 1 );
if (!MCIsStopped( hCtlr, 1, 2.0 )) {
    // Motor failed to stop within time limit (2 seconds),
    // error handling code goes here
}
MCGetAuxEncIdxEx( hCtlr, 1, &CapturedAuxEncoderPosition);
MCGetAuxEncPosEx( hCtlr, 1, &AuxEncoderPosition );
EncoderPulsesFromIndex = AuxEncoderPosition - CapturedAuxEncoderPosition;
MCSetAuxEncPos( hCtlr, 1, EncoderPulsesFromIndex );
StepsFromIndex = EncoderPulsesFromIndex * EncoderPulsesToSteps;
MCSetPosition( hCtlr, 1, StepsFromIndex );
```

Unlike the ***MCFindIndex( )*** function, which re-defines the position reported by a servos' encoder, ***MCSetAuxEncPos( )*** does not re-define the position of the auxiliary encoder. ***MCSetAuxEncPos( )*** only arms the capture of the encoder index mark, which is then indicated by the status bit **MC_STAT_INP_AUX** being set.

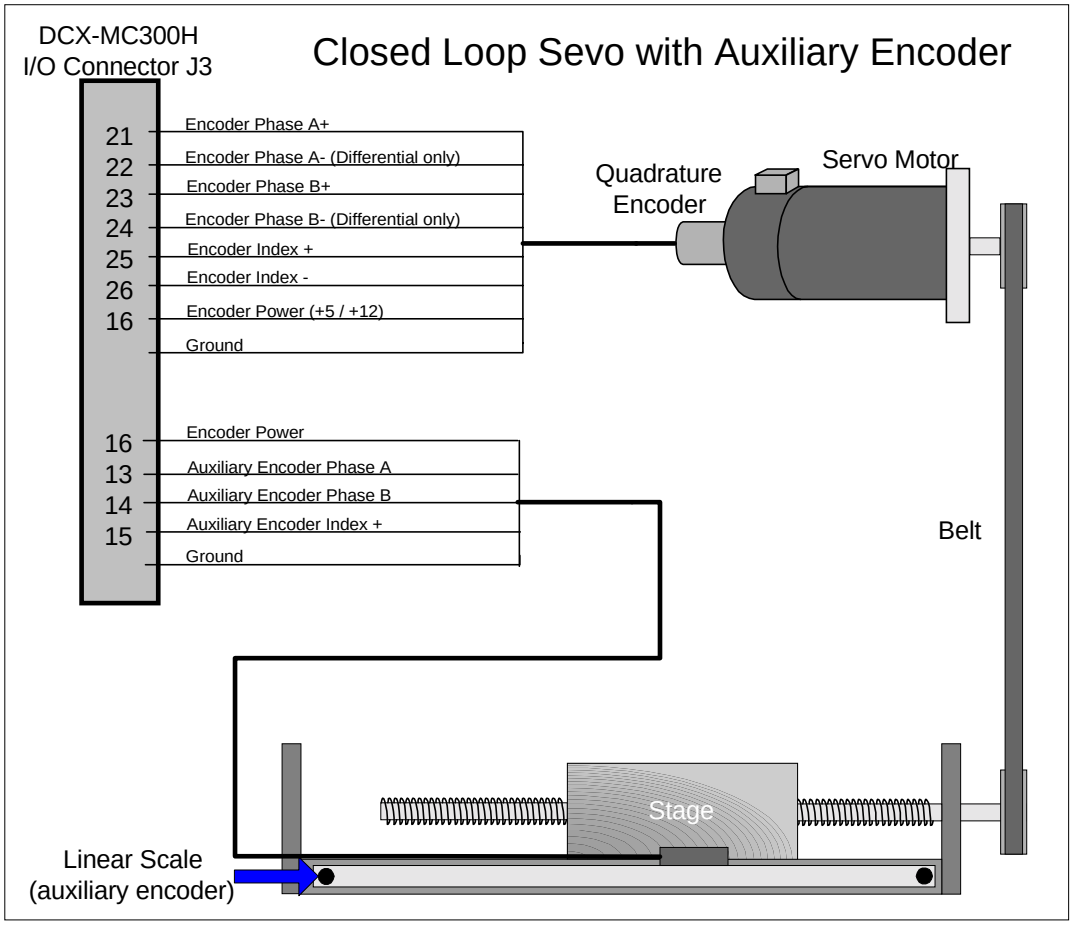
```
;MCCL example - define positions at auxiliary encoder index mark
1AF,LU"STATUS",1RL@0,IS27,BK,NO,JR-5       ;Enable aux. encoder index mark
                                           ;capture, loop until index captured
1AX,AR100                                  ;load accumulator with encoder
                                           ;position when index occurred, store
                                           ;the value in user register 100
```

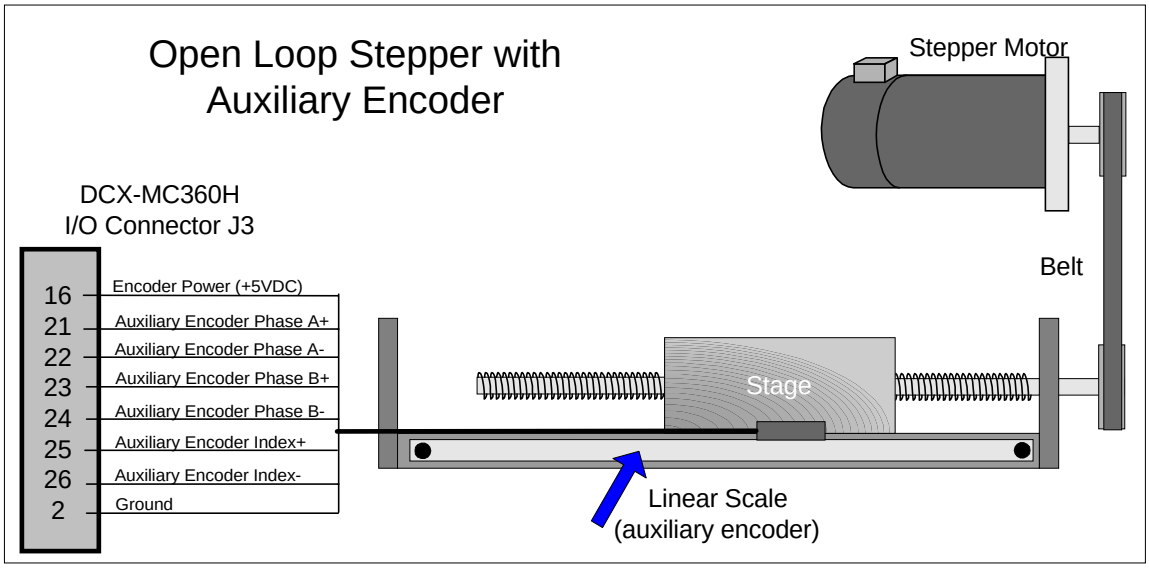
For MCCL homing samples that can be downloaded to the controller and executed please refer to the MotionCD (\PCI300\MCCL\Homing).

If no encoder index mark output is available, the position of the auxiliary encoder can be redefined at anytime using the MCAPI function **MCSetAuxEncPos( )**.

Auxiliary Encoder Connections

The two diagrams that follow illustrate the typical wiring connections required for interfacing DCX motion control modules to an auxiliary encoder. For additional information please refer to the **Connectors, Jumpers, and Schematics** chapter.





Verifying the Operation of the Auxiliary Encoder
Enabling Closed Loop Mode (from the Stepper Setup dialog) causes PMC's Motor Mover to display the position of auxiliary encoder.

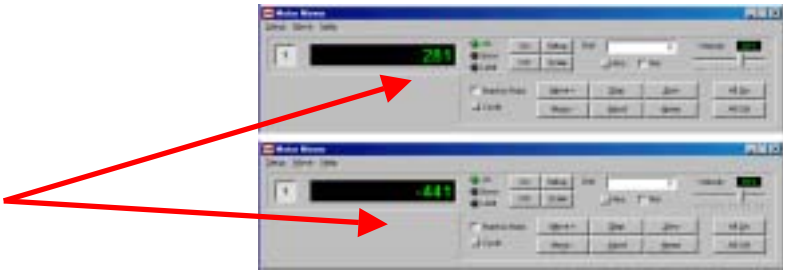
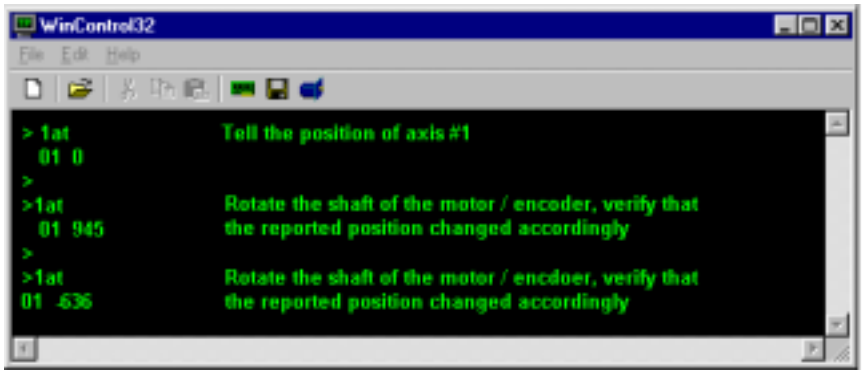


Figure 34: Rotate the motor / encoder shaft back and forth. Verify that the position is changing accordingly

From WinControl, issuing the **A**uxiliary encoder **T**ell position (***a*AT**) command will cause the current position of the auxiliary encoder to be reported.



Backlash Compensation

In applications where the mechanical system isn't directly connected to the motor, it may be required that the motor move an extra amount to compensate for system backlash. When backlash compensation is enabled, the DCX controller will offset the target position of a move by the user defined backlash distance. This feature is only available for servos (MC300 MC320) at this time.

The function ***MCEnableBacklash( )*** is used to initiate backlash compensation. The *Backlash* parameter of this function sets the amount of compensation and should be equal to one half of the amount the axis must move to take up the backlash when it changes direction. The units for this command parameter are encoder counts, or the units established by the ***MCSetScale( )*** command for this axis.

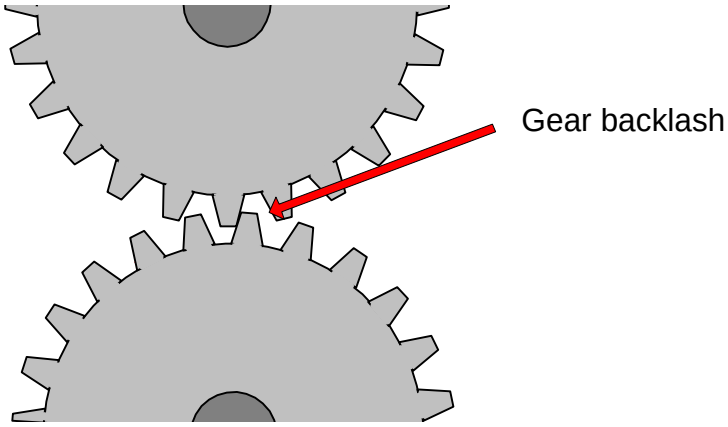
When this feature is enabled, the controller will add or subtract the backlash distance from the motor's commanded position during all subsequent moves. If the motor moves in a positive direction, the distance will be added; if the motor moves in a negative direction, it will be subtracted. When the motor finishes a move, it will remain in the compensated position until the next move.

Prior to enabling backlash compensation, the motor should be positioned halfway between the two positions where it makes contact with the mechanical gearing. This will allow the controller to take up the backlash when the first move in either direction is made, without "bumping" the mechanical position.

While backlash compensation is enabled, the response to the ***MCGetPosition( )***, ***MCTellTarget( )*** and ***MCTellOptimal( )*** commands will be adjusted to reflect the ideal positions as if no mechanical backlash was present.

For the example below assume that the system has 200 encoder counts of backlash. This example moves the system to the middle of the backlash range and enables compensation. Note that the compensation value (in encoder counts) used with ***MCEnableBacklash()*** is half of the total amount of backlash.

```
MCMoveRelative( hCtlr, 1, -100.0 );         // move to middle of backlash
MCWaitForStop( hCtlr, 1 );                  // let motion finish
MCEnableBacklash( hCtlr, 1, 100.0, TRUE );  // enable backlash compensation
```



Emergency Stop

Many applications that use motion control systems must accommodate regulatory requirements for immediate shut down due to emergency situations. Typically these requirements do not allow an emergency shut down to be controlled by a programmable computing device. The drawing below depicts an application where an emergency stop must be a completely 'hard wired' event.

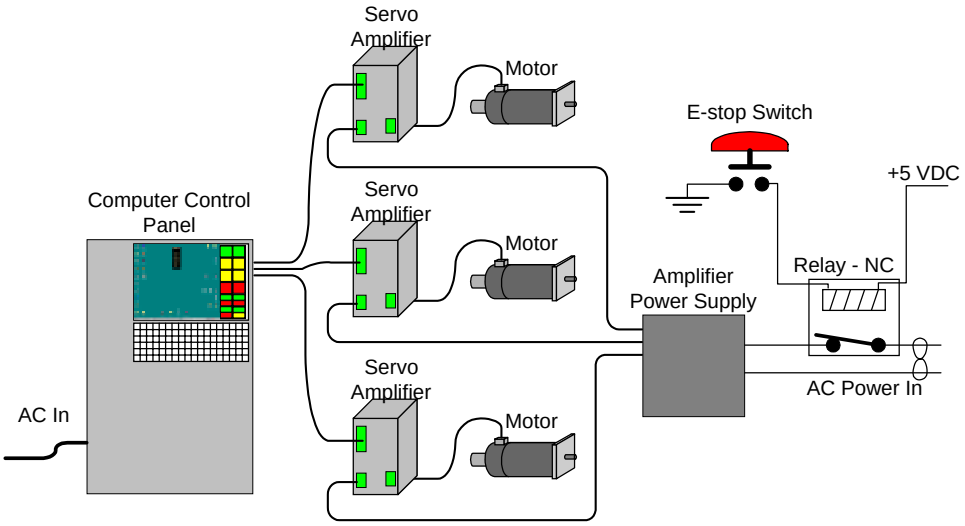


Figure 35:Typical 'hard wired' E-stop

This 'hard wired' E-stop circuit uses a relay to disconnect power from the servo amplifiers. The motors and amplifiers would certainly be disabled, but the motion controller and the application program will have no indication that an error condition exists.

Wiring the E-Stop switch to the DCX
There are two ways to wire the DCX so that it can monitor the E-stop switch:

1) Connect the E-stop switch to one of the general purpose digital I/O lines
2) Connect the Amplifier Fault (MC300 & MC320) and Drive Fault (MC360) inputs to the E-stop switch

E-stop switch connected to DCX General Purpose Digital Input
Wire the E-stop switch to a general purpose digital I/O (channel #1). Each DCX digital channel has a 4.7K resistor pulled up to +5 volts. A background task is used to monitor the state of the input. If the channel is configured for low 'low true' operation, the input will report its state as 'off' until the E-stop switch is activated. The ***WaitForDigitalIO( )*** function will stay active in background until the input 'goes true'. For additional information on macro's and background tasks please refer to the **DCX-PCI300 MCCL Command Reference manual**.

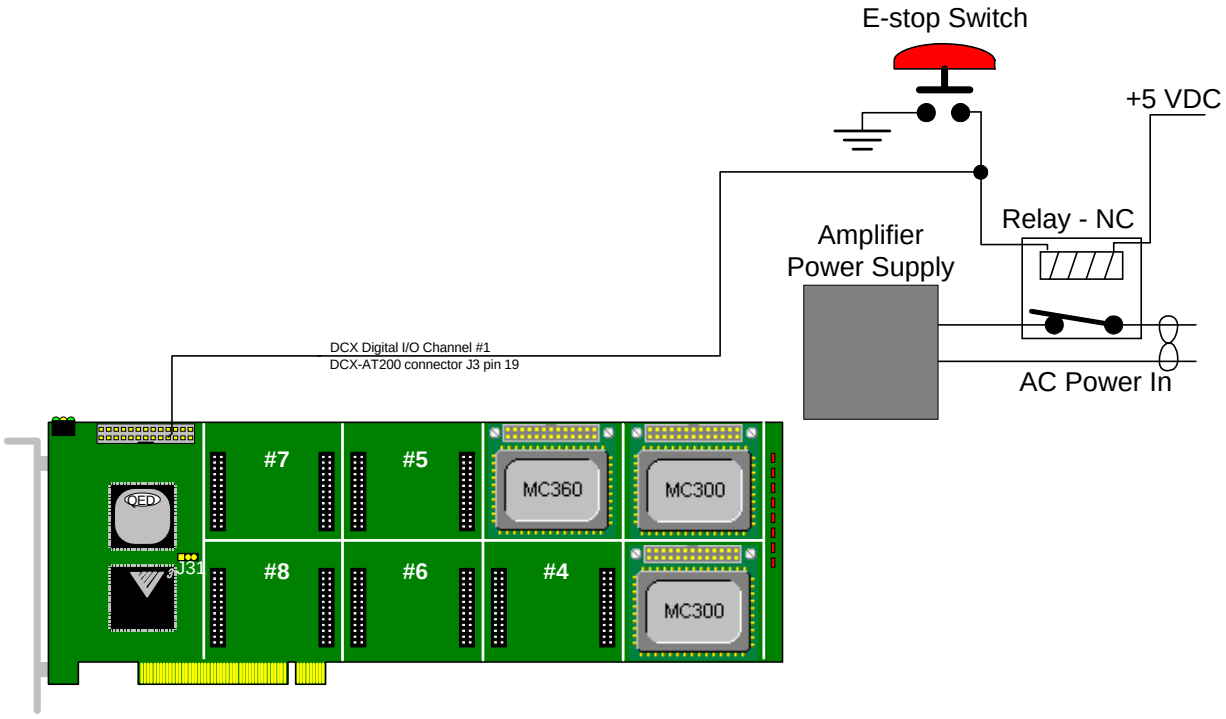


Figure 36:E-stop switch wired to DCX-PCI300 general purpose digital input

```
if (MCBlockBegin ( hCtlr,MC_BLOCK_TASK, 0) ==MCERR_NOERROR )    {
    MCSetRegister ( hCtlr, 100, 0, MC_TYPE_LONG);
    MCConfigureDigitalIO ( hCtlr, 1, MC_DIO_LOW );
    MCWaitForDigitalIO ( hCtlr, 1, TRUE );
    MCSetRegister hCtlr, 100, 1, MC_TYPE_LONG);
    MCEnableAxes( hCtlr, MC_ALL_AXES, FALSE );
    MCBlockEnd ( hCtlr, NULL);
}


// periodically poll the user register #100 for a value of 1. If true the user
// can jump to an E-stop handling routine.

MCGetUserRegister (hCtlr, 100, &Estop, MC_TYPE_LONG);
```

E-stop switch connected to Amplifier Fault servo module input

The Amplifier Fault inputs of MC300 and MC320 servo modules and/or the Drive Fault inputs of a MC360 stepper module can be used to disable motion with no user software action required. The E-stop switch is wired to the Amplifier/Drive Fault input (connector J3 pin 10 for servo modules or pin 7 for stepper modules) of **each module**. Auto shut down of motion upon activation of the E-stop switch is enabled by the **MCMotion** structure member **EnableAmpFault**. When the E-stop switch is activated:

1) The axis is disabled (PID loop terminated, Amplifier Enable output turned off)
2) The status flag **MC_STAT_AMP_FAULT** will be set for each axis
3) The status flag **MC_STAT_ERROR** will be set for each axis

When the E-stop condition has been cleared, motion can be resumed after issuing the ***MCEnableAxis*** function with the parameter *wAxis* set to **MC_ALL_AXES**.

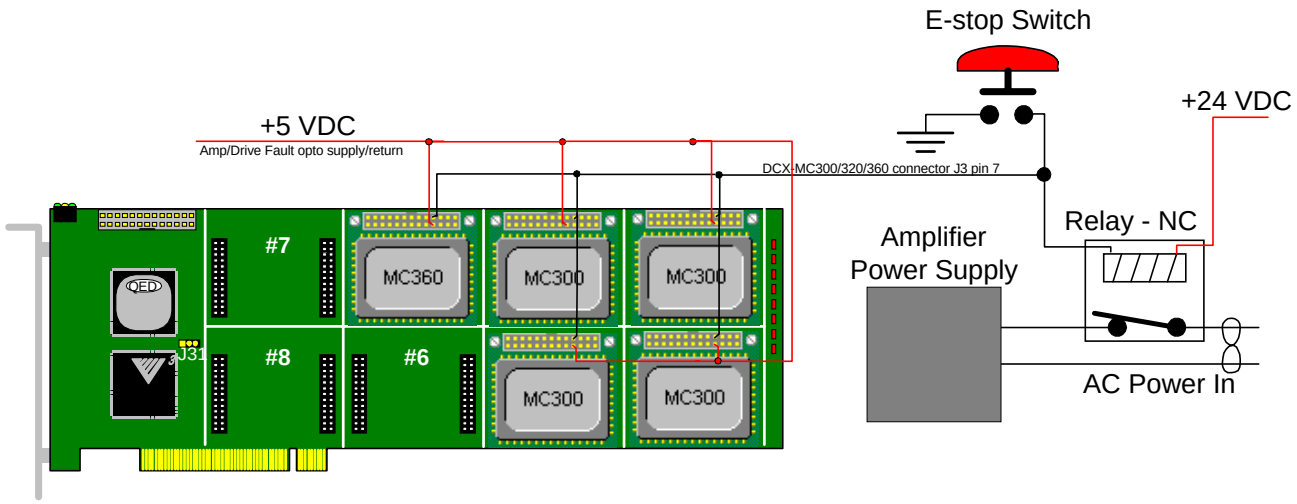


Figure 37:E-stop circuit wired to the fault input of DCX modules

Encoder Rollover

The DCX motion controller provides 32 bit position resolution, resulting in a position range of –2,147,483,647 to 2,147,483,647. For an application where the axis is moving at maximum velocity (1million encoder counts/steps per second), the encoder would rollover in approximately 3.58 minutes. When the encoder rolls over, the reported position of the axis will change from a positive to a negative value. For example, if the axis is at position 2,147,483,647 the next positive encoder count will cause the DCX to report the position as –2,147,483,647.

If a user scaling other than 1:1 has been defined the DCX controller will report the position in user units. The reported position at which the value will rollover is based on the user scaling. If user scaling is set to 10,000 encoder counts to one position unit, the reported position will rollover at position 214,748.3647. The next positive encoder count will cause the DCX to report the position as –214,748.3647.

Encoder rollover during Position Mode moves
The DCX does not support executing Position Mode moves when the encoder rolls over. No matter what the commanded position, the axis will stop at the rollover position (2,147,483,647 or –214,748.3647).

Encoder rollover during Velocity Mode moves
No disruption or unexpected motion will occur if a rollover occurs during a Velocity mode (***MCSetOperatingMode***, **MC_MODE_VELOCITY**) move.

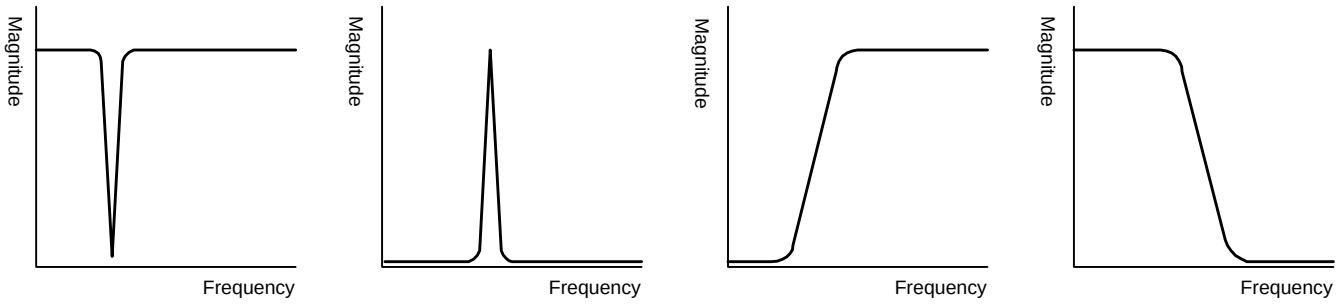
Prior to executing a velocity mode move in which the encoder position may rollover the axis **must** be homed (MCFindIndex or MCSetPosition) to position 0. Defining a offset to the home position will cause the axis to pause at the rollover point.

User Defined Filters (Notch, Low Pass, High Pass, and Band Pass)

The DCX-MC302 Dual Servo Control module does not support user defined filters.

The DCX-PCI300 supports user defined IIR (Infinite Impulse Response) filters for each axis of servo motion. User defined filters include:

- Notch filter, otherwise known as a Band Stop filter, allows the user to define a specific frequency to be attenuated.
- Low Pass filter - removes the high frequency response of a servo system.
- High Pass filter - removes the low frequency response from the system.
- Band Pass filter - blocks both low and high frequency.



Notch Filter **Band Pass** **High Pass** **Low Pass**

It is not uncommon for a servo system and its load to exhibit mechanical resonance's. One or more Notch filters can be cascaded to attenuate these resonance's.

The DCX-PCI300, DCX motion control modules, and associated software is not designed to detect, record, or display mechanical resonance's. The user is responsible for providing the necessary equipment for analyzing resonance's.

Each axis supports as many as six biquad stages, providing up to 12th order performance. The form of a biquad stage is shown in the following equation:

$$y_2 = a_0 * x_2 + a_1 * x_1 + a_2 * x_3 + b_1 * y_0 + b_2 * y_1$$

The DCX-PCI300 supports as many as two IIR filters per servo axis. Usually this would means that the user could define two Notch filters, the controller does support combining two different filter types (Notch & Low Pass, Notch and High Pass, etc...).

The setting of the PID filter loop rate (HS, MS, LS) determines how many biquad stages an axis can execute. The table below details the association:

Biquad Stages	High Speed	Medium Speed	Low Speed
1	**Yes**	**Yes**	**Yes**
2		**Yes**	**Yes**
3		**Yes**	**Yes**
4		**Yes**	**Yes**
5			**Yes**
6			**Yes**

While cascading filter stages will increase attenuation (the deeper notch), it will also tend to increase ripple. In other words, don't use any more stages than you need

Calculating the filter coefficients

A DOS utility program IIRFilter.exe available on the MotionCD (PCI300\iir filter\) allows the user to define the type, quantity, and frequency response for an axis. The utility will then generate a MCCL command file that can be downloaded to the controller via WinControl or the **MCDLG_DownloadFile** MCAPI common dialog.

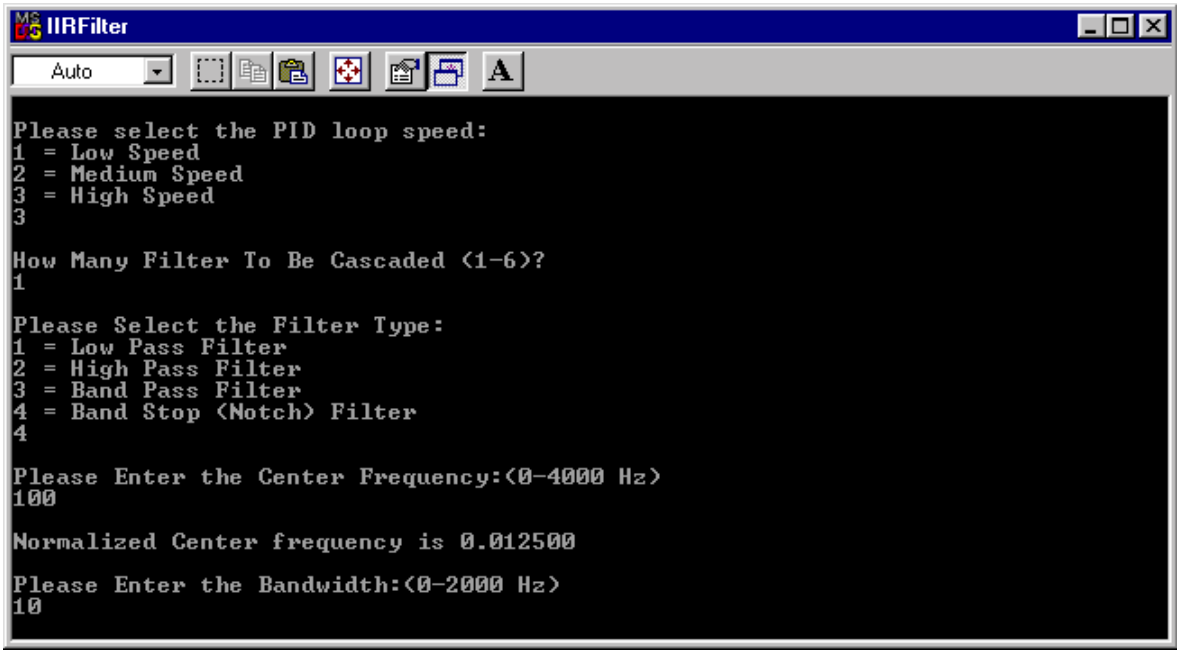


Figure 38: The DOS IIR filter utility (iirfilter.exe) calculates the filter coefficients, which can then be downloaded to the controller via WinControl

Example – Defining a Notch filter
A machine builder has detected that axis one of a four axis machine has a significant resonance at 100 Hz. The following steps will configure the DCX-PCI300 to implement a Notch filter at 100 Hz with a bandwidth of 10Hz.

Step #1 – Define the filter and calculate the coefficients
Open the IIR filter utility (IIRFilter.exe). Enter the following:

Select controller type: 1 = PCI300 <enter>
Select the filter type: 4 = Band Stop (Notch) filter <enter>
Select PID loop speed: 3 = High Speed <enter>
Enter the Center Frequency: 100 <enter>
Bandwidth: 10 <enter>

The utility will calculate the filter coefficients and store them in an MCCL command file named Flt_Coef.pci3.

```
1ZF                ;Zero filter to clear previous loaded filter coefficients

;1 Band Stop (Notch) Filters with center frequency at 100 Hz and bandwidth 10 Hz

1FL0.998531        ;Load filter 1 coefficient a0
1FL-1.990906       ;Load filter 1 coefficient a1
1FL0.998531        ;Load filter 1 coefficient a2
1FL1.986358        ;Load filter 1 coefficient b1
1FL-0.992514       ;Load filter 1 coefficient b2
```

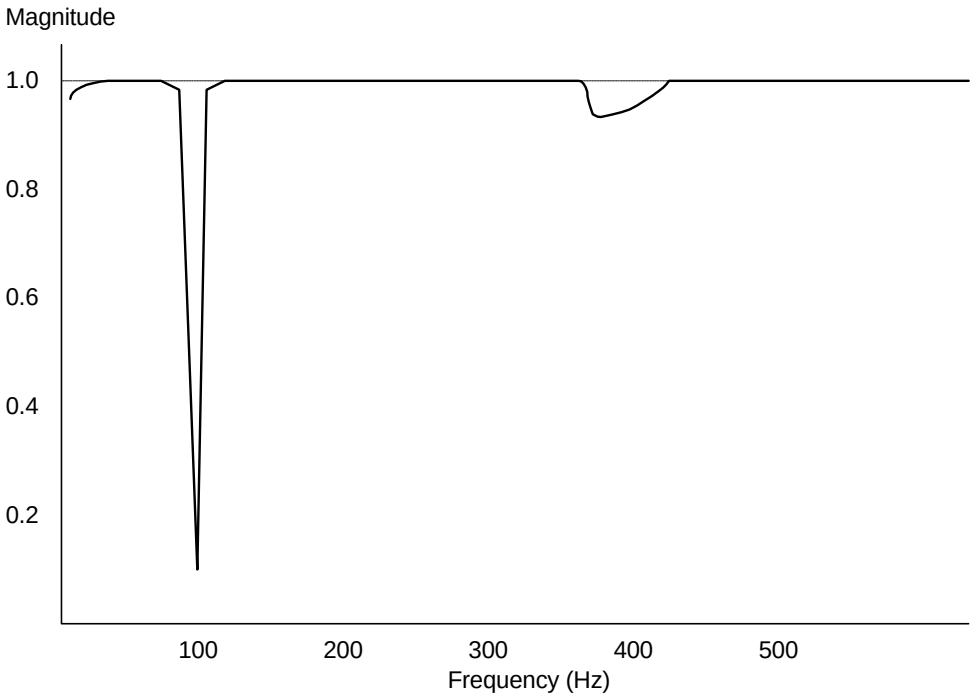


Figure 39: 100 Hz Notch filter frequency response

Download the filter coefficient file to the controller with WinControl or use the PMC Common Motion Dialog function ***MCDLG_DownloadFile***. To enable the digital filter issue the **Y**es IIR **F**ilter (***a*YF**) command from WinControl or call the ***MCEnableDigitalFilter ( )*** function from a high level program. Enable the axis with the ***MCEnableAxis function ( )***.

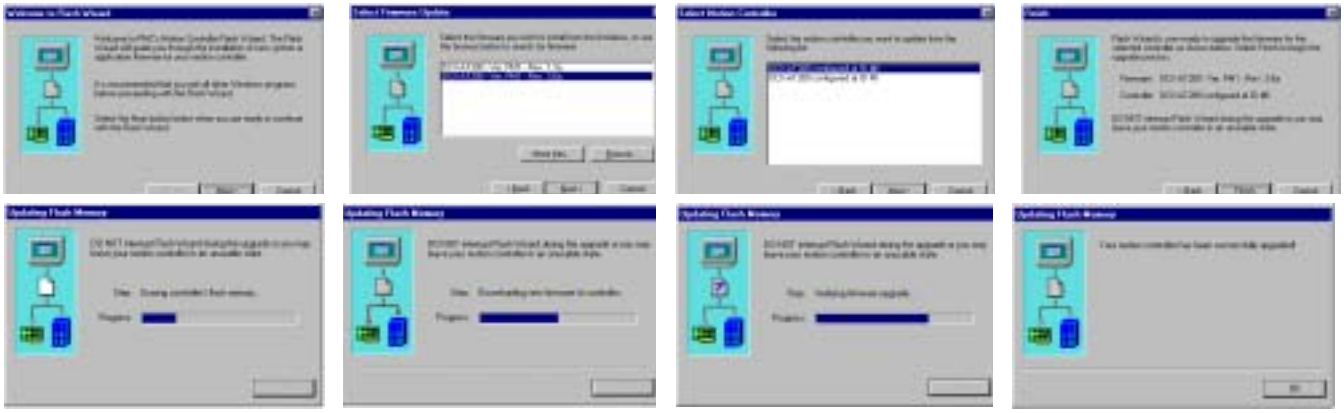
The servo will perform significantly different with the IIR filter enabled so you will need to re-tune the axis.

The Save and Restore functions of MCAPI 3.2.0000 do not support the IIR filter parameters. Each time the controller is initialized (reset or power cycle) you will need to download the coefficient file.

Flash Memory Firmware Update

Each time the PC is re-booted (reset or power cycle) the operating code (typically called firmware) for the DCX-PCI300 is loaded into on-board SDRAM (Static Dynamic Random Access Memory). The source files for the operating code is written to the PC's hard disk drive during the installation of the MCAPI.

PMC's **Flash Wizard** (the DCX-PCI300 requires Flash Wizard rev. 2.20) is a windows utility that allows the user to easily update the operational code. Code updates are available from the **MotionCD** or from PMC's web site **www.pmccorp.com**.



With Windows 98 and MCAPI 3.02.000 a verification error may occur during code download. To complete the firmware upgrade close Flash Wizard and restart the PC.

Initializing and Restoring Controller Configuration

When the controller is reset or the computer is turned on all motion (PID settings, Vel/Accel/Decel,Limits), I/O (High / Low true), and global controller (User Scaling) settings revert to default values (default setting are listed on page 258).

New Applications & First Time users
PMC's Motion Integrator was designed specifically for first time users and new applications. Not only does it proceed step by step through the testing and configuration of a motion control system, it automatically saves all settings by creating an initialization file. Upon completion of Motion Integrator (including Servo Tuning) , all other PMC application programs can be directed to load the setting by selecting **Auto Initialize** from the **File** menu.

Figure 40: Launch Motor Mover with user defined controller setting by selecting Auto Initialize

Saving user define settings
Upon recognition by the MCAPI that one or more DCX motion controllers are installed, an initialization file (mcapi.ini) is copied into the Windows folder. Initially this file contains only information about the controller type and interface settings. If at anytime the user selects **Save All Axes Settings** from the **File** menu of any of PMC's application programs, the current settings for all installed axes will be written into the mcapi.ini file.

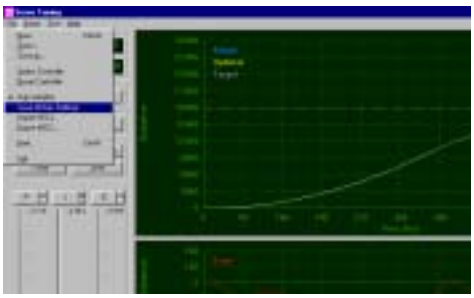


Figure 41: Saving axis settings to mcapi.ini from PMC's Servo Tuning program

To define controller settings from a user's application program, call the Motion Control Dialog function ***MCDLG_SaveAxis***.

Selecting **Save All Axes** from the File menu of a PMC application program will **over write all previously stored settings**.

Restoring controller settings
Other than opening a handle for the controller, the first step in all user application programs should be to restore all previously defined axis settings. This is accomplished by calling the Motion Dialog ***MCDLG_RestoreAxis***.

Learning/Teaching Points

As many as 256 points can be stored for ***each axis*** in the DCX's point memory by using the **MCLearnPoint( )** function. A stored point can be either the actual position of an axis (**MC_LRN_POSITION**) or the target position of an axis (**MC_LRN_TARGET**).

The value **MC_LRN_POINT** would typically be used in conjunction with jogging. The operator would jog the axes along the desired path, issuing the ***MCLearnPoint( )*** command at regular intervals. The ***MCMovePoint( )*** command would then be used to 'play back' the path traversed by the operator.

For applications where the target point data was previously recorded and stored in the PC, the value **MC_LRN_TARGET** would be used to load the target points into the DCX. For some applications, using MCLearnPoint( ) to load a series of moves may be 'easier' than issuing a series of contour mode linear moves, even though the results would be the same.

Once all points have been stored, the axes are commanded to move to the stored positions with ***MCMoveToPosition( )***. The parameter *wIndex* indicates to which stored point the axis should move.

```
// Move axis 1 and store position in consecutive point storage locations.

WORD wIndex;
MCEnableAxis( hCtlr, 1, TRUE );         // motor on
MCGoHome( hCtlr, 1 );                   // start from absolute zero
MCWaitForStop( hCtlr, 1, 0.100 );

for (wIndex = 0; wIndex < 5; wIndex++) {
    MCMoveRelative( hCtlr, 1, 1234.0 );  // move
    MCWaitForStop( hCtlr, 1, 0.100 );    // are we there yet?
    MCLearnPoint( hCtlr, 1, wIndex, MC_LRN_POSITION );
}


// Store several positions for axis 4 without actually moving the axis. Note // that
axis is disabled with MCEnableAxis( ) prior to storing positions

WORD wIndex;
MCEnableAxis( hCtlr, 4, FALSE );        // motor off
for (wIndex = 0; wIndex < 5; wIndex++) {
    MCMoveRelative( hCtlr, 4, 2468.0 );  // nothing actually moves
    MCLearnTarget( hCtlr, 4, wIndex, MC_LRN_TARGET );
}
```

```
// This example moves to the stored positions, dwelling for 0.2 seconds at
// each point.

WORD wIndex;
MCEnableAxis( hCtlr, 4 );               // enable axis
for (wIndex = 0; wIndex < 5; wIndex++) {
   MCMoveToPoint( hCtlr, 4, wIndex );   // move to next point
   MCWaitForStopped( hCtlr, 4, 0.2 );
}
```

To cause the DCX to perform linear interpolated moves between the taught points, place each of the axes in contour mode. Use the lowest axis number as the contour mode command parameters, this is the controlling axis. Set the vector velocity and accelerations of the controlling axis. Issue a single ***MCMoveToPoint( )*** command to the controlling axis with the point numbers as the command parameter. Note that when point memory is used with motors in contour mode, point 0 should not be used. This example executes linearly interpolated moves through three stored points of axes 1, 2, and 3.

```
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 2, 1, MC_MODE_CONTOUR );
MCSetOperatingMode( hCtlr, 3, 1, MC_MODE_CONTOUR );

//  Linear interpolated move sequence through stored points

for (wIndex = 1; wIndex < 4; wIndex++) {
   MCBlockBegin( hCtlr, MC_BLOCK_CONTR_LIN, 1 );
      MCMoveToPoint( hCtlr, 1, wIndex );
      MCMoveToPoint( hCtlr, 1, wIndex );
      MCMoveToPoint( hCtlr, 1, wIndex );
   MCBlockEnd( hCtlr, NULL );
}
```

Building MCCL Macro Sequences

A powerful feature of the DCX is the ability to define MCCL (Motion Control Command Language) command sequences as macros.

For additional information on macro's and MCCL (Motion Control Command Language) commands please refer to the **MCCL Reference Manual**.

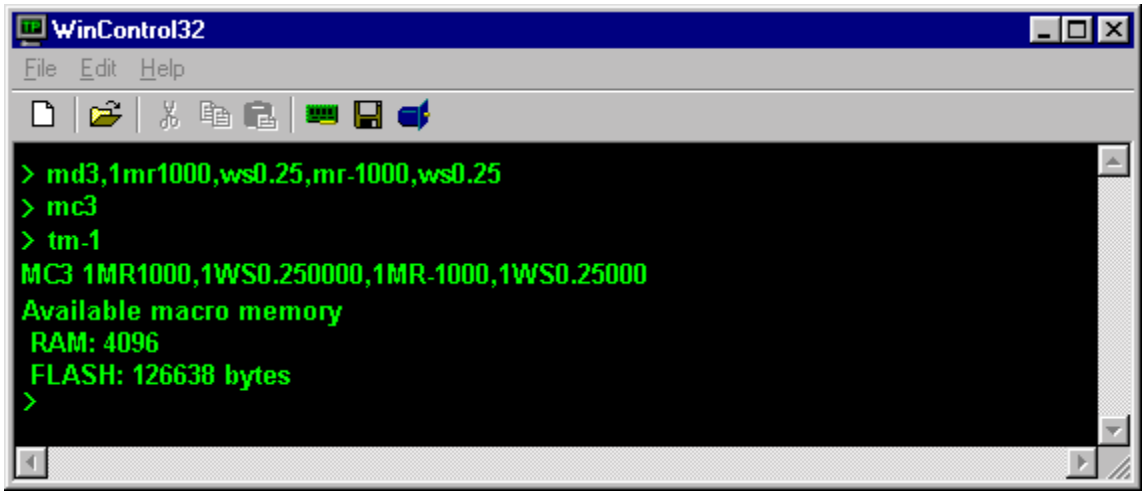
A DCX macro is a user define sequence of operations that is executed by issuing a single command. For example:

```
1MR1000,WS0.25,MR-1000,WS0.25
```

will cause the motor attached to axis 1 to move 1000 counts in the positive direction, wait one quarter second after it has reached the destination, then move back to the original position followed by a similar delay. If this sequence were to represent a frequently desired motion for the system, it could be defined as a macro command. This is done by inserting a Macro Define (**MD*n***) command as the first command in the command string. For example:

```
MD3,1MR1000,WS0.25,MR-1000,WS0.25
```

will define macro #3. Whenever it is desired to perform this motion sequence, issue the command Macro Call (MC3). To command the DCX to display the contents of a macro, issue the **T**ell **M**acro (**TM*n***) command with parameter 'n' = the number of the macro to be displayed. To display the contents of all stored macro's issue the Tell macro command with parameter 'n' = -1.



Once a macro operation has begun, the host will not be able to communicate with the DCX until the **macro has terminated**. For information on communicating with the controller while executing macro's please refer to the section titled **MCCL Multi-Tasking**.

The DCX can store up to 1000 user defined macros. Each macro can include as many as 255 bytes. Depending on the type of command and type of parameter, a command can range from 2 bytes (a command with no parameter) to 10 bytes (a command with a 64 bit floating point parameter).

All memory on the DCX-PCI300 is volatile, which means that the data in memory will be cleared when the controller is reset or power to the board is turned off. The **R**eset **M**acro (**RM*n***) command is used to erase macros.

Since the DCX provides no protection against overflowing the macro storage space, it is suggested that the user monitor the amount of memory available for macro storage. The **T**ell **M**acro (**TM*n***) command can be used to display the amount of RAM memory available for macros storage at any give time.

To terminate the execution of any macro that was started from WinControl press the escape key. To start a macro that runs indefinitely without 'locking up' communication with the host, start the macro's with the ***generate a Background task*** (GT) command instead of the ***Call macro command*** (MC). This will allow the operations called by macro 0 to execute as a background task. Please refer to the next section **Multi-Tasking**.

The DCX-PCI300 supports Single Stepping of any MCCL macro command executing as the foreground task. For additional information please refer to **Single Stepping MCCL Programs** later in this chapter.

MCCL Multi-Tasking

The DCX command interpreter is designed to accept commands from the user and execute them immediately. With the addition of sequencing commands, the user is able to create sophisticated command sequences that run continuously, performing repetitive monitoring and control tasks. The drawback of running a continuous command sequence is that the command interpreter is not able to accept other commands from the user.

Once a macro operation has begun, the host will not be able to communicate with the DCX until the **macro has terminated**.

The DCX supports Multi-tasking, which allows the controller to execute continuous monitoring or control sequences as background tasks while the foreground task communicates with the 'host'.

With the exception of ***reporting commands*** (Tell Position, Tell Status, etc...), any MCCL commands can be executed in a background task. Prior to executing a command sequence/macro as a background task, the ***user should always test the macro by first executing it as a foreground task***. When the user is satisfied with the operation of the macro, it can be run as a background task by issuing the **G**enerate **T**ask (**GT*n***) command, specifying the macro number as the command parameter. After the execution of the Generate Task command, the accumulator (register 0) will contain an identifier for the background task. Within a few milliseconds, the DCX will begin running the macro as a background task in parallel with the foreground command interpreter. The DCX will be free to accept new commands from the user.

```
;Multitasking example – while axis #1 is moving, monitor the state of digital
;input #4. When the input goes active, stop axis #1 and terminate the
;background task

AL0,AR10                          ;define user register 10 as input #4 active
                                  ;flag register
AL0,AR100                         ;define user register #100 as background task
                                  ;ID register

MD100,IN4,MJ101,NO,1JR-3          ;jump to macro 101 when digital input #4
                                  ;turns on
MD101,1ST,1WS.05,AL1,AR10,ET@100  ;stop axis #1. Terminate background task

GT100,AR@100,1VM,1DI0,1GO         ;spawn macro #10 as background task. Store
                                  ;task ID into register #100. Start axis #1
                                  ;moving in velocity mode,
```

Note: Immediately after 'spawning' the background task (with the GTn command), the value in the accumulator (task identifier) should be stored in a user register. This value will be required to terminate execution of the background task.

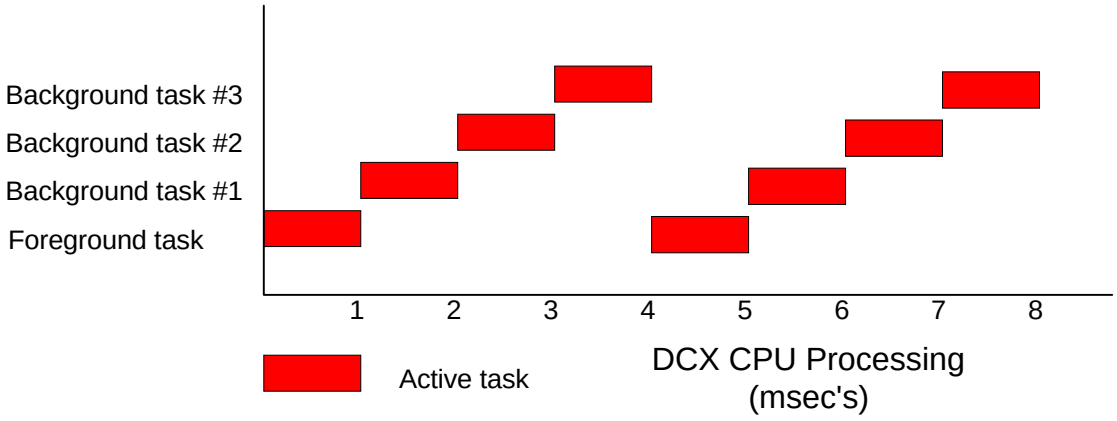
Another way to create a background task is to place the Generate Task command as the first command in a command line, using a parameter of 0. This instructs the command interpreter to take all the commands that follow the Generate Task command and cause them to run as a background task. The commands will run identically to commands placed in a macro and generated as a task.

```
;Multitasking example – while axis #1 is moving, monitor the state of the
;motor error status bit (bit 7). If error occurs set bit #1 of user
;register 200

GT0,AR@100,LU"STATUS",1RL@0,IC7,JR-3,NO,AL1,AR200,ET@100
                                  ;loop on axis #1 status bit 7, if set; set
                                  ;bit #1 of register 200, terminate task using
                                  ;Task ID (in register #100)
```

Within the background task, the commands can move motors, wait for events, or perform operations on the registers, totally independent of any commands issued in the foreground. However, the user must be careful that they do not conflict with each other. For example, if a background task issues a move command to cause a motor to move to absolute position +1000, and the user issues a command at the same time to move the motor to -1000, it is unpredictable whether the motor will go to plus or minus 1000.

In order to prevent conflicts over the registers, the background task has its own set of registers 0 through 9 (register 0 is the accumulator). These are private to the background task and are referred to as its 'local' registers. The balance of the registers, 10 through 255, are shared by the background task and foreground command interpreter, they are referred to as 'global' registers. If the user wishes to pass information to or from the background task, this can be done by placing values in the global register. Note that when a task is created, an identifier for the task is stored in register 0 of both the parent and child tasks.

The DCX is able to run multiple background tasks, each with their own set of registers, but can only have one foreground command interpreter. The maximum number of background tasks is 13. Each background task and the foreground command interpreter get an equal share of the DCX processor's time. When one or more background tasks are active the DCX Task Handler will begin issuing local DCX interrupts every 1 millisecond. Each time the task handler interrupt is asserted, the DCX will switch from executing one task to the next. For example if three background tasks are active, plus the foreground task (always active), each of the four tasks will receive 1 msec of processor time every 4 msec's.



While a background task executes a **Wait** command, that task no longer receives any processor time. For tasks that perform monitoring functions in an endless loop, the command throughput of the DCX can be improved by executing a **Wait** command at the end of the loop until the task needs to run again.

A common way for a background task to be terminated, is when the command sequence of the task finishes execution. This will occur at the end of the macro or if a **B**rea**K** (**BK**) command is executed. When a task is terminated, the resources it required are made available to run other background tasks.

```
;Multitasking example – this background task will terminate itself if the
;motor error status bit for axis #1 is set. This sequence is similar to the
;previous example except that the task is self terminating, so register #100
is not required.

GT0,LU”STATUS”,1RL@0,IC7,JR-3,NO,AL1,AR200
                                ;loop on axis #1 status bit 7, if set; set
                                ;bit #1 of register 200, task self terminates
                                ;(no commands left to execute)
```

Alternatively, the **E**scape **T**ask (**Te*n***) command can be used to force a background task to terminate. When a task is generated by the **GT** command, a value known as the **Task ID** is placed into the accumulator. This value should immediately be copied into a user register. The parameter to this command must be the value that was placed in accumulator (register 0) of the parent task, when the Generate Task command was issued.

```
;Multitasking example – Terminating a background task with the Escape Task
command.

GT100,AR@150                    ;call macro #100 as a background task, copy
                                ; task ID into user register 150

ET@150                          ;to terminate background task issue escape
                                ; task command with parameter n = Task ID
```

Pause and Resume Motion

The current release of the Motion Control API (3.2.0000) does not provide high level function calls for Pause and Resume. The following descriptions use MCCL commands to configure an axis for position compare. The MCAPI OEM low level function ***pmccmdex( )*** can be used to issue MCCL commands via the MCAPI.

Future releases of the MCAPI will resolve this lack of support.

The Save Configuration (***a*SC*n***) and Restore Configuration (***a*RC*n***) commands can be used with the Velocity Override command to pause and resume motion.

Each of these commands takes an axis specifier *a* and requires a file number as the command parameter n. These commands save and restore the entire motor table. This includes the public motor table in dual port memory and the private motor table in internal RAM.

These commands allow the motors to be stopped (aVO0) during a contour move, their configurations saved, switched to any other mode (except contouring), moved about and then returned to their original positions, their configurations restored, and then commanded to continue the contour move (aVO1.0).

Note: Prior to resuming motion it is very important that the axes be returned to the **exact** position at which the motor table was saved. If this is not done, the axis will either jump to the position at which motion was paused or it may error out.

Position Capture

The DCX-MC302, DCX-MC320, and DCX-MC362 do not support position capture.

The DCX supports capturing the position of the primary encoder (MC300) or the step count register (MC360) on the leading edge of the Position Capture input. As many as 512 captured positions can be stored in the recording memory of the DCX module. For servo modules the maximum frequency of position captures is based on the servo loop setting (High = 8KHz, Medium = 4 KHz, Low = 2 KHz). For stepper axes the maximum frequency is fixed at 1 KHz.

The MCAPI function ***MCEnableCapture ( )*** is used to initiate position capture. When this feature is enabled the current position will be recorded on the rising edge of the capture input. If parameter *count* equals 1 the module will capture only one position. If parameter *count* equals 2 the module will capture two positions, and so on. When the number of positions captured = *count* , the **MC_STAT_POS_CAPT** flag (bit 11 of the axis status) will be set. To report the number of positions captured issue the ***MCGetCount ( )*** function with the *type* = **MC_COUNT_CAPTURE**. To disable position capture issue ***MCEnableCapture ( )*** with parameter *count* equal to 0. Captured positions may be retrieved using the ***MCGetCapturedData( )*** function.

```
Long int count;
double    data{10};

MCEnableAxis( hCtlr, 1, 1 );
MCMoveRelative( hCtlr, 1, 10000.0 );


// Capture 10 positions
//
MCEnableCapture( hCtlr, 1, 10.0 );


//  Retrieve the 10 captured positions into local array
```

```
//

do {
   MCGetCount(( hCtlr, 1, MC_COUNT CAPTURE, &count);
}  while (count <10);

MCGetCaptureData( hCtlr, 1, MC_CAPTURE_ACTUAL, 0, 10, &data );
```

Position Compare

The DCX-MC302, DCX-MC320, and DCX-MC362 do not support position compare.

The DCX modules provide a high speed open collector output to indicate that a position compare event has occurred. The assertion of this output is based on the position of the primary encoder (MC300) or the step count register (MC360). As many as 512 compare positions can be stored in the recording memory of the DCX module.

Compare pre defined positions
To configure an axis for position compare first use the MCAPI function ***McConfigureCompare ( )*** to define the number of compare positions (as many as 512) and the compare output mode. Then issue the MCAPI function ***MCEnableCompare ( )*** with the *flag* = **MC_COMPARE_ENABLE**. This will terminate any current compare operation and initializes the compare index to 0. After starting a move, when the actual position is equal to the compare position the compare output will be turned on (pulled to ground) and the next compare position will be loaded into the compare register. When all position compare events have been completed the **MC_STAT_BREAKPOINT** flag of the axis status will be set.

Compare at incremental distances
For compare events at fixed distances of travel use the function ***MCEnableCompare ( )*** and:

1) Store the beginning point (first compare position) in the first location of *values*
2) Set the *num* parameter to 1
3) Set the *inc* parameter to the distance (counts or steps) between compare events

Maximum compare frequency
The position update frequency of a DCX servo module (MC300/320) module is based on the setting of the servo loop rate (High = 8KHz, Medium = 4 KHz, Low = 2 KHz). Therefore the distance between compare positions cannot be such that the time from one compare event to the next is less than the position update frequency of the module (High = 125usec. , Medium = 250 usec., Low = 500 usec.). For MC360 stepper modules the update frequency is always 1KHz. The time between compare events cannot be less than 1000 usec's.

Compare output signal configuration
When the compare output is activated as the result of a compare or breakpoint occurrence, the compare output signal will react according to the which mode has been selected with the *mode* parameter of the ***MCConfigureCompare ( )*** function.

mode	***Description***
MC_COMPARE_DISABLE	Disables the compare output
MC_COMPARE_INVERT	Inverts the active level of the compare output
MC_COMPARE_ONESHOT	Configures the compare output for one shot operation (one shot period is defined by the *period* parameter of ***McConfigureCompare*** ( ) function. The one shot pulse period range is from 1usec. to 1.0 second. For one shot periods less than 50 milliseconds the timer resolution is 1 micro second. For one shot periods greater than 50 milliseconds the timer resolution is 50 milliseconds.
MC_COMPARE_STATIC	Configures the compare output to turn on when a compare event occurs. The output will stay on until a new compare event is called
MC_COMPARE_TOGGLE	Configures the compare output to toggle between the active and inactive state each time a compare event occurs

For all of the output modes, the compare output will be activated within 1/2 microsecond of the encoder reaching the position. The optical isolator on the compare output signal takes an additional 2 to 3 microseconds to turn on depending on the load circuit. This optical isolator will take about 50 microseconds to turn off (depending on the load). When the compare output mode is set to Disabled, the output will be at its' in-active level. The controller sets the output mode to Disabled on power up or reset.

To report the number of compare events that have occurred issue the ***MCGetCount ( )*** function with the *type* = **MC_COUNT_COMPARE**. To disable position compare issue ***MCEnableCompare ( )*** with parameter *flag* value = **MC_COMPARE_DISABLE**.

```
//
//  Use positions spaced 5 units apart, beginning at 10.0 as compare
//  positions. Toggle the output pin on valid compares. Wait for 20
//  compares to complete.
//
data[0] = 10.0;     // starting point
MCConfigureCompare( hCtlr, 1, data, 1, 5.0, MC_COMPARE_TOGGLE, 0.0 );

MCEnableCompare( hCtlr, 1, MC_ENABLE_COMPARE );   // enable compare
MCMoveRelative( hCtlr, 1, 100.0 );

do {      // wait for 5 points
    MCGetCount( hCtlr, 1, MC_COUNT_COMPARE, &count );
} while (count < 20);
```

Reassigning Axis Numbers

The current release of the Motion Control API (3.2.0000) does not provide high level function calls for reassigning axis numbers. The following descriptions use MCCL commands to configure an axis for position compare. The MCAPI OEM low level function ***pmccmdex( )*** can be used to issue MCCL commands via the MCAPI.

Future releases of the MCAPI will resolve this lack of support.

The DCX defaults to assigning axis numbers logically, not based on a motor module's physical location. In the following graphic three modules are installed on a DCX-PCI300. When the computer is power up the MC320 in module location #1 will automatically be defined as axis one. The MC320 in module location #3 would be defined as axis two. The MC300 in module location #5 would be defined as axis three.

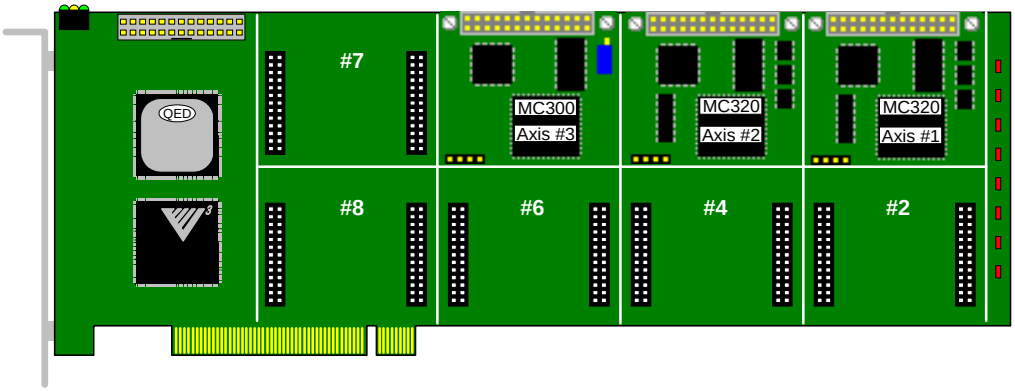


Figure 42:Assigning axis numbers to DCX motion control modules

Using the Use Physical (***aUPn***) command the user can redefine the axis number of DCX motion control module. Referencing the previous graphic, to redefine axes 2 and 3 as axes 3 and 5:

```
UP                              ;issue the UP command with no axis
                                ;specifier a or parameter n, this step
                                ;is required to clear the logical axis
                                ;number assignment performed by the
                                 DCX-PCI300 on power up.

3UP3                            ;Reassign the module in physical
                                ;location 3 (parameter n) as axis 3
                                ;(axis specifier a)
5UP5                            ;Reassign the module in physical
                                ;location 5 (parameter n) as axis 5
                                ;(axis specifier a)
```

Note – The reassignment of axes **must be done** before sending any commands (setup, move, etc…) to the controller.

Note – The first step to changing axis numbers is to clear all axis assignments by issuing the Use Physical assignment (***aUPn***) command with no axis specifier *a* and parameter *n*. Once this has been done all axes must be reassigned with the **UP** command, even the axes for which the automatically assigned axis number was correct.

Record Motion Data

The DCX supports capturing and retrieving motion data for servo axes (MC300, MC302, MC320) and closed loop stepper axes (MC360). Captured position data is typically used to analyze servo motor performance and PID loop tuning parameters. PMC's Servo Tuning utility uses this function to analyze servo performance. The MCAPI function ***MCCaptureData( )*** is used to acquire motion data for a servo axis. This function supports capturing:

- Actual Position versus time
- Optimal Position versus time
- Following error versus time
- DAC output versus time (DCX-MC300 and MC320)

The time base (8 KHz, 4 KHz, 2 KHz) for captured data is set by **Rate** member of the **MCMotion** data structure. The function ***MCGetCapturedData( )*** is used to retrieve the captured data. This example captures 1000 data points from axis 3, then reads the captured data into an array for further processing.

```
double Data[1000];

MCBlockBegin( hCtlr, MC_BLOCK_COMPOUND, 0 );
MCCaptureData( hCtlr, 3, 1000, 0.001, 0.0 );
MCMoveRelative( hCtlr, 3, 1000.0 );
MCWaitForStop( hCtlr, 3, 0.0 );
MCBlockEnd( hCtrlr, NULL );

//  Retrieve captured actual position data into local array
//
if (MCGetCaptureData( hCtlr, 3, MC_DATA_ACTUAL, 0, 1000, &Data ) {
   . . .        // process data
```

Resetting the DCX

The DCX supports software controlled reset. To reset the DCX-PCI300 motherboard and all installed axes issue the MCAPI function ***MCReset( )***. For additional information please refer to the **DCX-PCI300 MCAPI Reference Manual**.

Most PMC application programs (Motor Mover, Servo Tuning, WinControl) allow the user to reset the controller by selecting ***Reset Controller*** from the WinControl File menu.

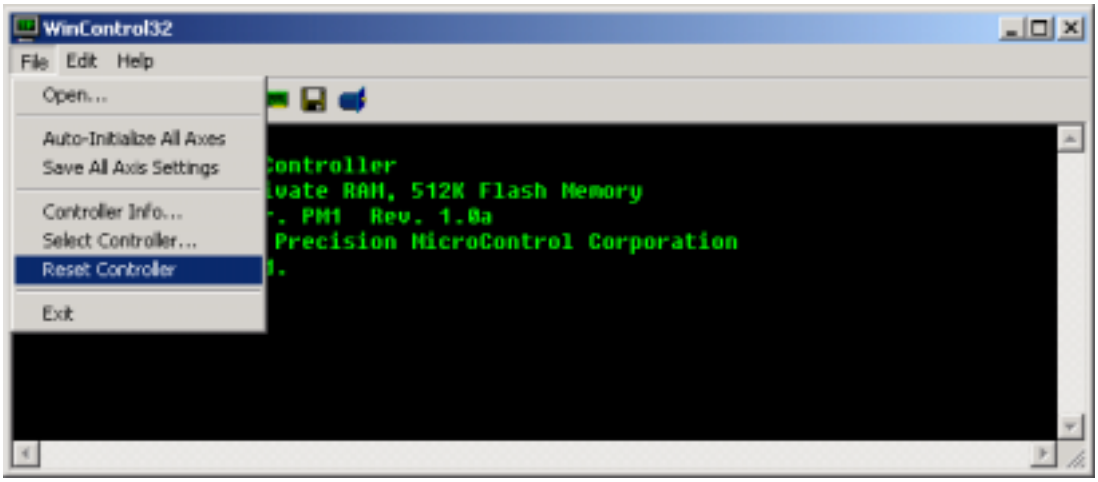


Figure 43: Resetting the DCX-PCI300

Resetting the DCX-PCI300 from a user application program (with ***MCReset( )***) or from one of a PMC's software programs (by selecting ***Reset Controller*** from: Motor Mover, WinControl, Servo Tuning, etc...) will cause the controller to revert to default settings (PID, velocity, accel/decel, limits, etc...). For information restoring the user defined settings please refer to the **Initializing and Restoring Controller Configuration** section in this chapter.

In the event of a 'hang up' of the application program and/or controller, the application program may fail to resume operation after issuing the ***MCReset( )*** function. The user will have to terminate and then re-open the application program.

Until the DCX has fully re-initialized the Reset Relay will be energized.

Single Stepping MCCL Programs

While the DCX is executing any Motion Control Command Language (MCCL) macro program, the user can enable single step mode by entering <ctrl> <B>. Each time this keyboard sequence is entered, the next MCCL command in the program sequence will be executed. The following macro program will be used for this example of single stepping:

```
MD10,WA1,1MR1000,1WS.1,1TP,1MR-1000,1WS.1,1TP,RP
```

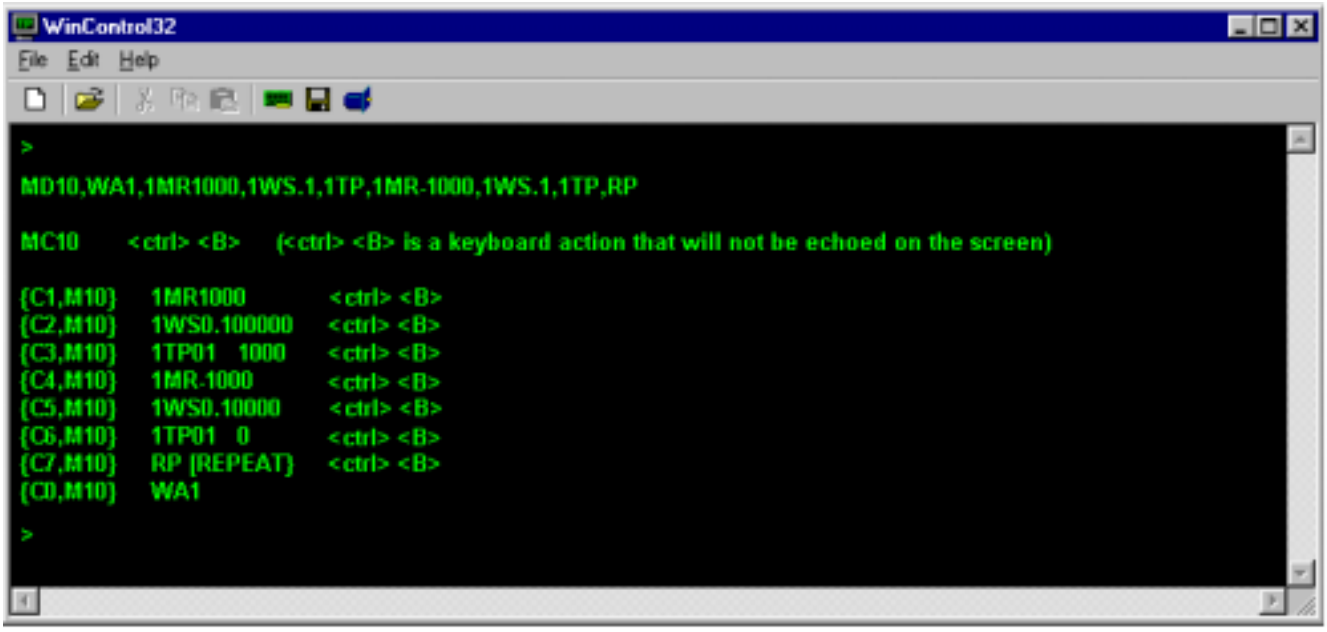
This sample program will: wait for 1 second, move 1000 encoder counts, report the position 100 msec's after the calculated trajectory is complete, move -1000 encoder counts, report the position 100 msec's after the calculated trajectory is complete, repeat the command sequence.

This command sequence can be entered directly into the memory of the DCX by typing the command sequence in the terminal interface program WinCtl32.exe or by downloading a text file via WinControl's file menu.

To begin single step execution of the above example macro enter MC10 (call macro #10) then <ctrl> <B> the following will be displayed:

{C1,MC10} 1MR1000 <

The display format of single step mode is: {Command #,Macro #} Next command to be executed



To end single stepping and return to immediate MCCL command execution press <Enter>. To abort the MCCL program enter <Escape>. Single step mode is not supported for a MCCL sequence that is executing as a background task.

Single stepping can also be enabled from within a MCCL program by using the break command immediately followed by a "string" parameter. When the break command is executed the controller will

display the characters in the string (inside the quotation marks) and then delay additional command execution until the space bar (execute next command and then delay) or the enter key (terminate single stepping and resume program execution) are selected. In the following example axis one will move 1000 counts, report the position, move –1000 counts, report the position, halt command execution until the space bar is entered, repeat one time.

```
MC10 1MR1000,1WS0.100000,1TP,1MR-1000,1WS0.100000,1TP,BK"wait",RP1

>mc10
01 997
01 0
BREAK AT COMMAND 6, MACRO 10
wait
 {C7,M10} RP10 [REPEAT] <
   <space bar>
01 997
01 0
BREAK AT COMMAND 6, MACRO 10
wait
 {C7,M10} RP10 [REPEAT] <
>
```

Note: Firmware revision 1.6c or higher is required for single step mode

Tangential Knife Control

Not supported at this time

A variation of Master/Slave mode supports using the position of two master axes to control the position of a third axis. The slave's optimal position will equal the arctangent of the ratio of the master axes' velocities. If the master axes are driving an X-Y table, the slave's position will equal the table's direction of travel. This dual master capability can be used to control the knife in cutting applications. This function is only available when the slave is a servo, and the two master axes, which can be servos or steppers, are in contour mode.

The current release of the Motion Control API (3.2.0000) does not provide a high level function call that enables tangential knife control. The following description uses the MCAPI OEM low level function ***pmccmdex( )*** to issue the MCCL command Set Master (***aSMn***) with a parameter ***n***, which configures the axis that controls the rotation of the knife.

Future releases of the MCAPI will resolve this lack of support.

Set the scaling of the knife axis to one unit equals 360 degrees of rotation of the knife. Issue the Set Master (aSMn) command to the slave axis with a parameter *n* that specifies the two master axes. The value of the Set Master parameter should be calculated as follows:

parameter n = master 1 axis number + (master 2 axis number x 16)

With two master operation, the slave axis will begin to track the master axis's direction when the first (and subsequent) contour mode move is issued. The blade of the knife will remain tangential to the contour path. To terminate the master and slave connections between the axes, issue the Set Master command to the slave axis with a parameter of 0, followed by either the Position Mode (PM) or the Velocity Mode (VM) command. If a significant change in direction (like a corner) of the X and/or Y axes occurs the knife will instantaneously. If this is undesirable, lift the blade, place the slave in position mode, re-position the blade, and lower the blade.

The following example will cut a 5 inch square out of a piece of linoleum. Axes 1 and 2 (X and Y respectively) are designated as the two master axes. Axis 3 will position the knife. Axis four (Z) is used to lift the knife at a corner, where an instantaneous change of direction in X and/or Y would be undesirable.

```
// define scaling of axis 3, 2000 encoder counts per revolution sets 1 unit to
// 1 ;revolution
//
MCGetScale( hCtlr, 3, &Scaling );
Scaling.Scale = 2000.0;
MCSetScale( hCtlr, 3, &Scaling );

// Use the MCCL command Set Master to configure axis 3 as a slave to axes 1 and 2.
// Header file MCAPI.H must be included
//
if (pmcrdy( hCtlr )) {
   arg = 33;
   if (pmccmdex( hCtlr, 3, SM, &arg, MC_TYPE_LONG ) == MCERR_NOERROR) {
   }
}


// turn on axes 1, 2, 3, & 4
//
MCEnableAxis( hCtlr, 1, MC_ALL_AXES );



//Execute 1st linear move
//
MCSetOperatingMode( hCtlr, 1, 1, MC_MODE_CONTOUR );  // axis 1 contour mode

//  Linear move, first side of triangle
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_LIN, 1 );
   MCMoveAbsolute( hCtlr, 1, 1000.0 );
   MCMoveAbsolute( hCtlr, 2, 0.0 );
MCBlockEnd( hCtlr, NULL );

// wait for end of contour move, lift blade, rotate blade, lower blade
//
MCWaitForStop( hCtlr, 1, 0.1 );
MCMoveRelative( hCtlr, 4, 1000.0 );
MCWaitForStop( hCtlr, 4, 0.1 );
MCMoveRelative( hCtlr, 3, 0.333 );
MCWaitForStop( hCtlr, 3, 0.1 );
MCMoveRelative( hCtlr, 4, -1000.0 );
MCWaitForStop( hCtlr, 4, 0.1 );


//  Linear move, second side of triangle
```

```
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_LIN, 1 );
   MCMoveAbsolute( hCtlr, 1, 500.0 );
   MCMoveAbsolute( hCtlr, 2, 1000.0 );
MCBlockEnd( hCtlr, NULL );

// wait for end of contour move, lift blade, rotate blade, lower blade
//
MCWaitForStop( hCtlr, 1, 0.1 );
MCMoveRelative( hCtlr, 4, 1000.0 );
MCWaitForStop( hCtlr, 4, 0.1 );
MCMoveRelative( hCtlr, 3, 0.333 );
MCWaitForStop( hCtlr, 3, 0.1 );
MCMoveRelative( hCtlr, 4, -1000.0 );
MCWaitForStop( hCtlr, 4, 0.1 );


//  Linear move, third side of triangle
//
MCBlockBegin( hCtlr, MC_BLOCK_CONTR_LIN, 1 );
   MCMoveAbsolute( hCtlr, 1, 0.0 );
   MCMoveAbsolute( hCtlr, 2, 0.0 );
MCBlockEnd( hCtlr, NULL );

// wait for end of contour move, lift blade, rotate blade, lower blade
//
MCWaitForStop( hCtlr, 1, 0.1 );
MCMoveRelative( hCtlr, 4, 1000.0 );
MCWaitForStop( hCtlr, 4, 0.1 );
MCMoveRelative( hCtlr, 3, 0.333 );
MCWaitForStop( hCtlr, 3, 0.1 );
MCMoveRelative( hCtlr, 4, -1000.0 );
MCWaitForStop( hCtlr, 4, 0.1 );


!!! now disable tangential knife control !!!
```

Threading Operations

Not supported at this time

Threading operations require not only tight synchronization between the primary axes, but also the ability to begin motion of the slave axis relative to a specific position of the master. The DCX implementation of threading uses the encoder index mark of the master axis to trigger motion of the slave.

The current release of the Motion Control API (3.2.0000) does not provide a high level function call for enabling threading operations. The following description uses the MCAPI OEM low level function ***pmccmdex( )*** to issue the MCCL command Set Master (***aSMn***) with a parameter ***n***, which configures the DCX controller for threading.

Future releases of the MCAPI will resolve this lack of support.

To enable Master/Slave Threading mode, issue the Set Master (aSMn) command where:

a = the axis number of the slave
n = the axis number of the master + 2

A move absolute, move relative, or go home command can also be issued to the slave axis to set a target position where the axis will be taken out of slave mode. The Index Arm or Find Index command must be issued to the master axis after the Set Master command has been issued to the slave axis. The slave will be synchronized to the master's position when its encoder index pulse occurs. In the following example the spindle (master) is axis #2 and the thread cutting tool is positioned by axis #1 (slave).

```
// Set scaling of master axis. For the spindle, this would typically be set to
// the number of encoder counts per revolution.
//


MCGetScale( hCtlr, 2, &Scaling );
Scaling.Scale = 2000.0;
MCSetScale( hCtlr, 2, &Scaling );


//Set scaling of the slave axis
//
MCGetScale( hCtlr, 1, &Scaling );
Scaling.Scale = 4000.0;
MCSetScale( hCtlr, 1, &Scaling );

MCMoveAbsolute( hCtlr, 1, 0.0 );        // move slave to starting position


MCWaitForStop( hCtlr, 1, 0.1 );         // wait till we're there



// Set the slave ratio. This is the lead or pitch when cutting a thread.
//
MCEnableGearing( hCtlr, 1, 2, 0.1, TRUE );


// Use the MCCL command Set Master to configure axis 2 as a slave to axis 1.
// Enable threading by n = 2 + 256. Header file MCAPI.H must be included
//
if (pmcrdy( hCtlr )) {
   arg = 258;
   if (pmccmdex( hCtlr, 3, SM, &arg, MC_TYPE_LONG ) == MCERR_NOERROR) {
   }
}



// Set the target position. This is the position at which slave mode is
// terminated and axis #1 will stop.
//
MCMoveAbsolute( hCtlr, 1, 1.0 );

// Start master axis moving in torque mode.
//
MCSetTorque( hCtlr, 2, 3.0 );

// Arm the index capture of the master axis. When the index pulse occurs, the
// slave will begin tracking the master axis until // the slave reaches its
// target position.
2IA
```

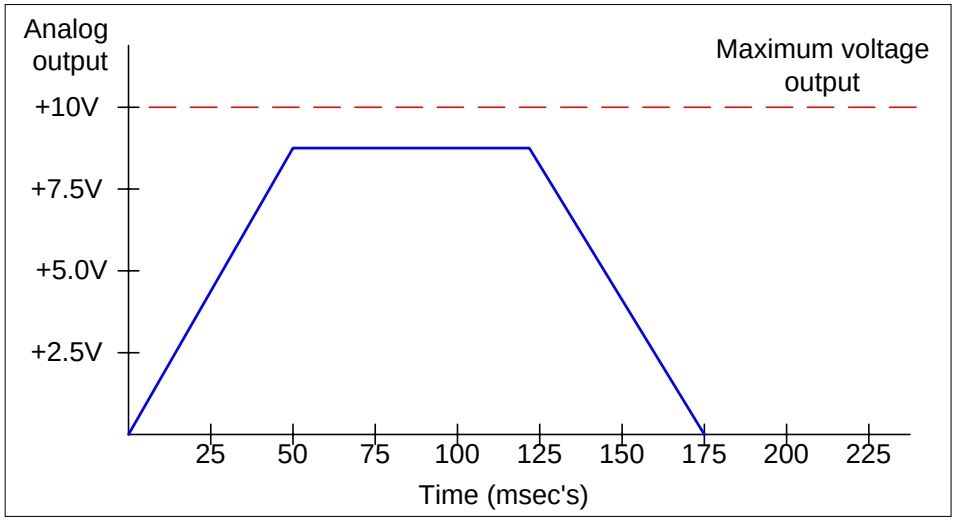
```
// This command sequence will repeat until auxiliary status bit 22 is clear,
// indicating that the slave has reached its target.
//
1RL16,IS22,JR-2,NO,2SQ0
```

The following bits of the axis auxiliary status word are used for monitoring the status of the slave axis during a threading operation:

Bit 22 = Axis is slaved to master's encoder position
Bit 23 = Axis is slaved and waiting for master's index mark

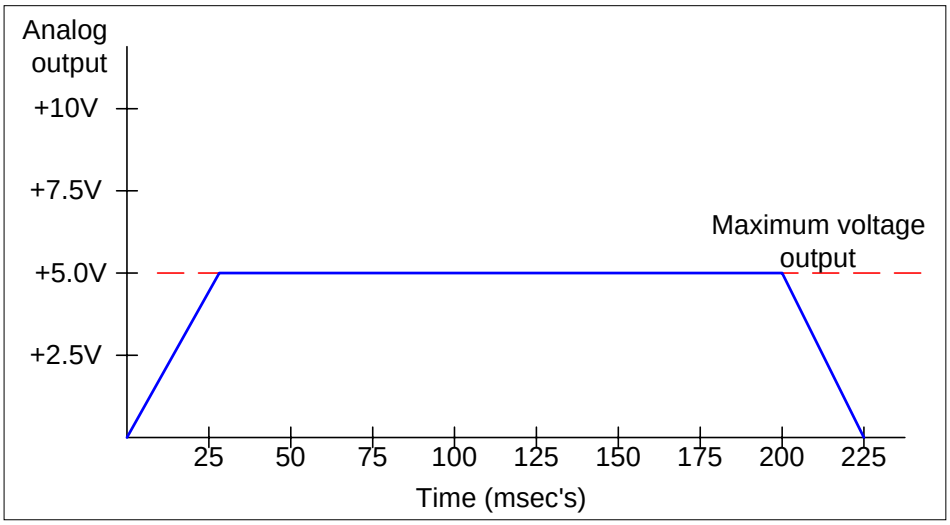
Torque Mode Output Control

The DCX servo modules (MC300, MC302, & MC320) provide two methods of ***directly and completely*** controlling the Torque/Velocity of a axis. When executing closed loop servo motion in Position or Velocity mode, the ***MCSetTorque( )*** command allows the user to limit the output signal or duty cycle to a specific level. The following graph depicts a simple position mode move of 1000 encoder counts with the default torque setting of 10 volts (no limit).



The graphic below depicts the same 1000 encoder count move, but the maximum voltage output has been limited to 5.0 volts.

```
MCSetTorque( hCtlr, 1, 5.0 );
MCMoveRelative( hCtlr, 1, 1000.0 );
```



Servo Modules as simple D/A output with encoder reader

Selecting Torque mode using the ***MCSetOperatingMode( )*** function allows the user to directly write values to the servo control DAC. This mode does not support closed loop servo control, but the user can read the position of the encoder at any time.

```
MCSetOperatingMode( hCtlr, 1, 0, MC_MODE_VELOCITY );
MCSetTorque( hCtlr, 1, 2.5 );          ;axis 1 output to 2.5V (MC300)
MCSetTorque( hCtlr, 1, 7.5 );          ;set duty cycle to 75% (MC320)
```

Turning off Integral gain during a move

State of the art servo controllers primarily use Proportional gain to determine the current/velocity command signal that the controller applies to the servo amplifier during a move. For motion control applications integral gain is used primarily to reduce the static position error **at the end of a move**. For additional information about servo tuning and integral gain please refer to :

- the Servo Tuning in the **Motion Control** chapter of this manual
- the Servo tuning tutorials on PMC's MotionCD

For some applications, integral gain has a tendency to cause bounce or oscillation of the command signal during a move. This tendency can be is especially problematic in:

- High gain servo systems
- Systems with high and / or irregular friction
- Systems with unbalanced loads
- Systems with unbalanced and / or high offset amplifiers

The following graphic shows the typical response of a high gain servo system when integral gain is enabled through out the move. Even though the following error never exceeds 10 encoder counts during the 100,000 count move, a significant oscillation (+/- 10 counts) occurred.

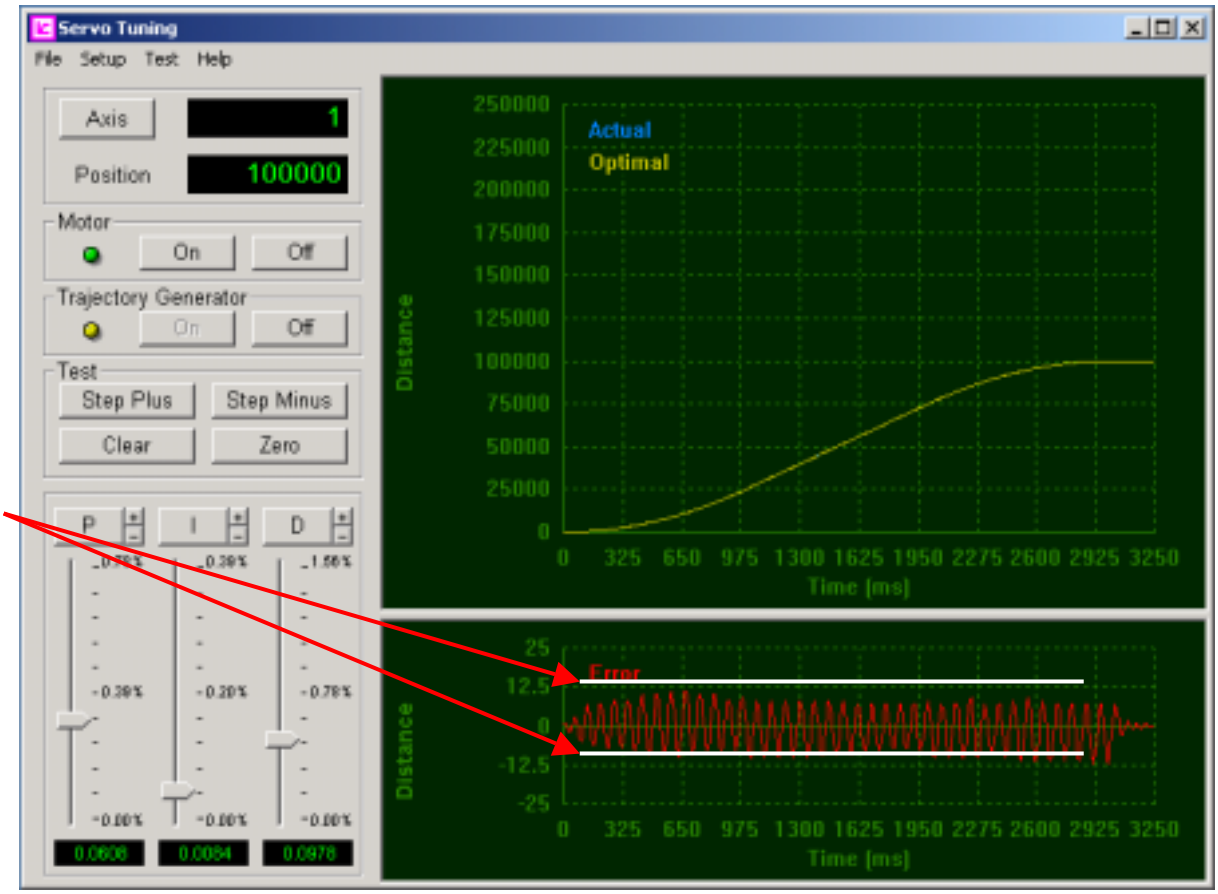


Figure 44: Typical servo response when integral gain is enabled throughout the move

By disabling the integral gain term until after the trajectory is complete (desired position = target position) the same move is accomplished with a following error of +/- 3 counts versus +/- 10 counts.

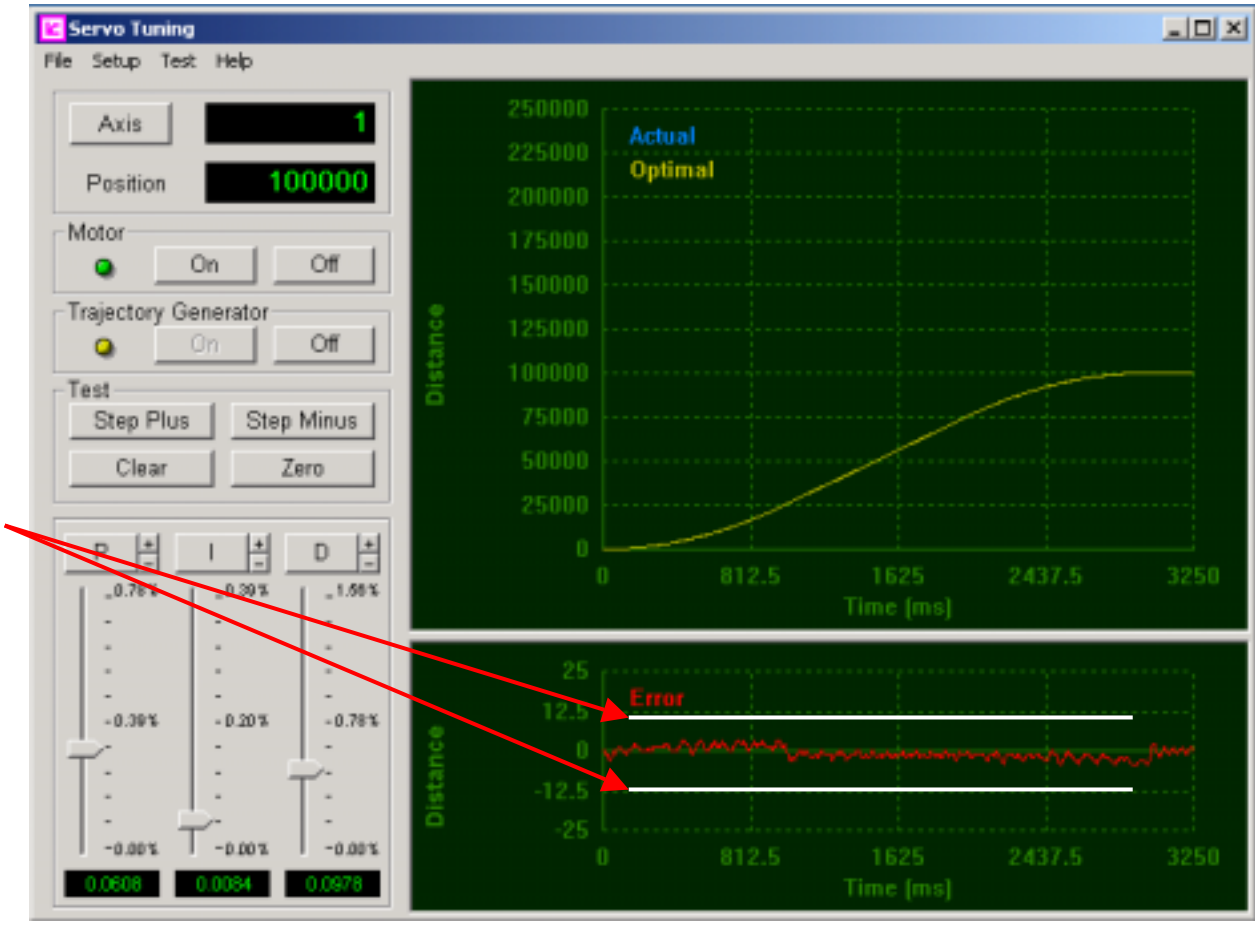


Figure 45: Typical servo response when integral gain is disabled until the calculated is complete

The ***IntegralOption*** member of the ***MCFilterEx*** structure allows the user to select from three different mode of integral gain operation for servo or closed loop stepper.

IntegralOptiont value	***Notes – (all other servo parameters remaining unchanged)***
MC_INT_NORMAL - integral term always on (default)	Smallest following error during move. As the integral term is increased the command output / following error will tend to bounce
MC_INT_FREEZE - Freezes accumulation of integration term during movement. Integration will continued once the calculated trajectory (trajectory complete, status bit 3 = 1) has been completed.	Ideal for applications with unbalanced loads (robotic arm with vertical axis, hoist)
MC_INT_ZERO - Zero and freeze accumulation of the integration term when motion begins. When the calculated trajectory (trajectory complete, status bit 3 = 1) has completed, enable the integration term	Most stable command signal / servo performance during the move. Largest following error during the move. Not acceptable for applications with unbalanced load.

From PMC application programs like Servo Tuning and Motor Mover the integral gain mode can be selected from the Servo Setup Dialog.

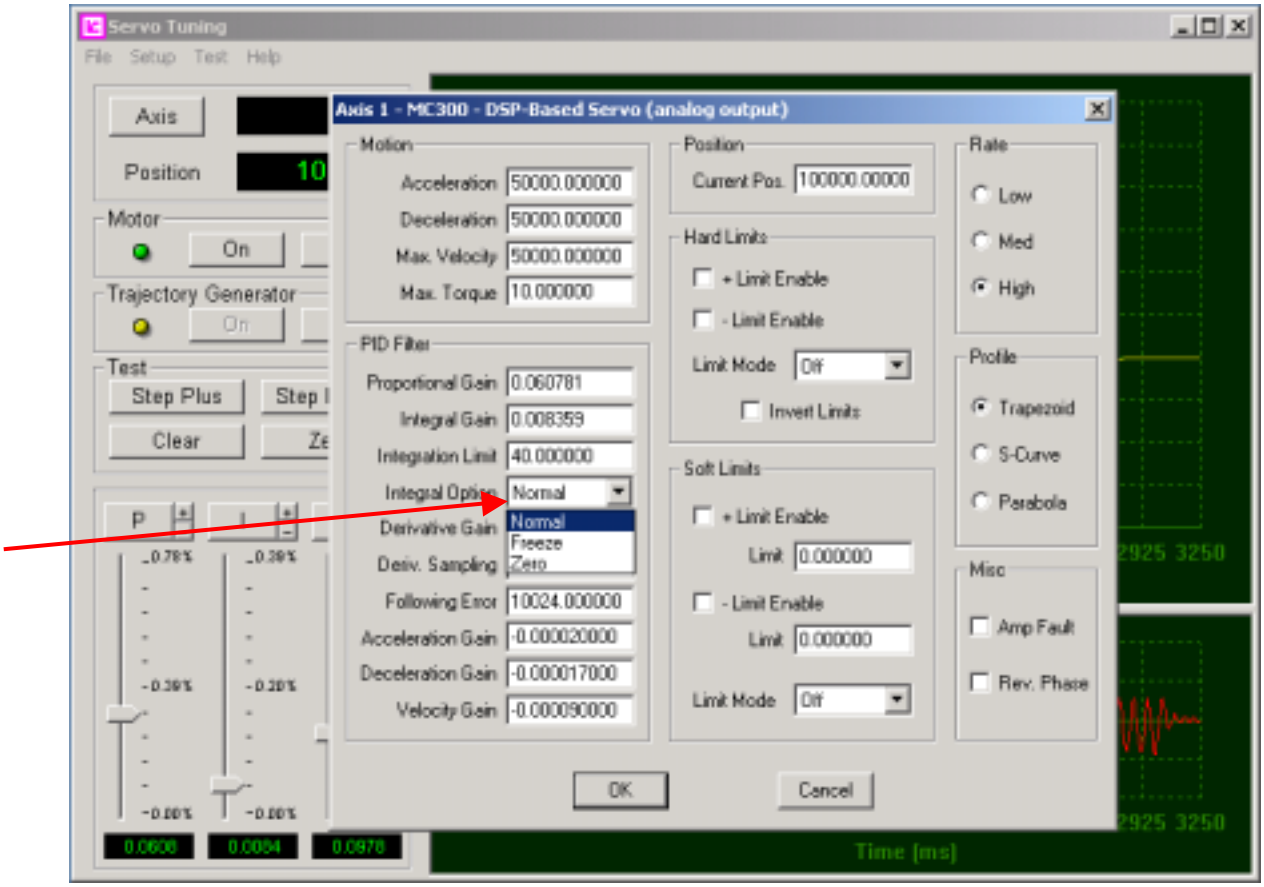


Figure 46: Using Servo Tuning's Servo Setup Dialog to set the integral gain mode of operation

Upgrading from a DCX-AT200 motion control system

For most motion control applications the DCX-PCI300 Modular Multi-Axis motion control system offers significant advantages over its predecessor, the DCX-AT200 system. The PCI300 enhancements include:

Servo motor control

- Texas Instrument DSP, 40 MHz, 16 bit, zero wait state (MC300 & MC320) versus 12 MHz, 8/16 bit micro controller (MC200)
- 16 bit DAC output (MC300 & MC320) versus 12 bit DAC (MC200)
- 8 KHz, 4 KHz, or 2 KHz servo loop rate (MC300 & MC320) versus 4 KHz (no integral term), 2 KHz, or 1 KHz (MC200)
- 10 MHz encoder frequency (MC300 & MC320) versus 1 MHz encoder frequency (MC200)
- High speed Position Capture: from 1 to 512 positions, 8 KHz (125 msec.) max frequency
- Position Compare: Open collector output, 1 to 512 user defined compare positions or fixed increment distance
- Bi-directional Optical isolation (MC300 & MC320) versus TTL level inputs (MC200)
- 32 bit Floating point PID parameters (MC300 & MC320) versus 16 bit integer PID parameters (MC200)

Stepper motor control

- Texas Instrument DSP, 40 MHz, 16 bit, zero wait state (MC360) versus 12 MHz, 8/16 bit micro controller (MC260)
- 5 MHz maximum step rate (MC360) versus 1 MHz maximum step rate (MC260)
- High speed Position Capture: from 1 to 512 positions, 8 KHz (125 msec.) max frequency
- Position Compare: Open collector output, 1 to 512 user defined compare positions or fixed increment distance
- Bi-directional Optical isolation (MC360) versus TTL level inputs (MC260)

Upgrading to the DCX-MC300 servo control module

The DCX-MC300 is similar in function to the DCX-MC200 servo control module. Other than the addition of Position Capture and Compare signals and the optical isolator supply/return lines, the pin-out of the MC200 and MC300 are the same. The changes that must be made when replacing a MC200 with a MC300 are:

- The PID parameters will need to be changed (the axis will need to be re-tuned)
- The axis inputs (Coarse Home, Limit +, Limit -, Amplifier Fault) use bi-directional optical isolators. These circuits operate with voltage levels from +12 to +24 VDC. See the wiring examples in the **Defining Motion Limits** and **Homing Axes** sections of the **Motion Control** chapter and in the **DCX-MC300** section of the **Connectors, Jumpers, and Schematics** chapter.
- The Amplifier Enable output circuit uses an optical isolator/open collector driver (versus basic TTL gate). The Amplifier Enable return (J3 pin 12) must be referenced to the return/ground of the servo amplifier. The Amplifier Enable output requires an external pull-

up (+5 to +24 VDC). See the wiring examples in the **DCX-MC300** section of the **Connectors, Jumpers, and Schematics** chapter.
- The DCX-MC300 does not provide a connection for the Index – output of an auxiliary encoder.

Upgrading to the DCX-MC360 stepper control module
The DCX-MC360 is similar in function to the DCX-MC260 stepper control module. Other than the addition of Position Capture and Compare signals and the optical isolator supply/return lines, the pin-out of the MC260 and MC360 are the same. The changes that must be made when replacing a MC260 with a MC360 are:

- The axis inputs (Home, Limit +, Limit -, Drive Fault, Null) use bi-directional optical isolators. These circuits operate with voltage levels from +12 to +24 VDC. See the wiring examples in the **Defining Motion Limits** and **Homing Axes** sections of the **Motion Control** chapter and in the **DCX-MC360** section of the **Connectors, Jumpers, and Schematics** chapter.
- The Driver Enable output circuit uses an open collector driver (versus basic TTL gate). The ground of the module (J3 pin 1 and/or 26) must be referenced to the return/ground of the stepper driver. The Driver Enable output requires an external pull-up (+5 to +24 VDC). See the wiring examples in the **DCX-MC300** section of the **Connectors, Jumpers, and Schematics** chapter.
- The Stopped output (J3 pin 7) has been replaced with the Drive Fault input
- The Jog input (J3 pin 10) has been replaced by the Auxiliary Encoder Power output
- The TTL output circuits for Full/Half Step (J3 pin 14) and Full/Half Current (J3 pin 15) now use open collector drivers. These outputs require an external pull-up (+5 to +24 VDC). See the wiring examples in the **DCX-MC300** section of the **Connectors, Jumpers, and Schematics** chapter.
- The Auxiliary Encoder Index – connection, which was connector J3 pin 22 is now found on connector J3 pin 23.
- An Auxiliary Encoder Index + connection, which was not available on the DCX-MC260, is now available on connector J3 pin 22 of the DCX-MC360
- The Auxiliary Encoder Coarse Home input, which was found on pin 23 of the DCX-MC260, is now available on pin 11 of the DCX-MC360.
- Due to the increased maximum step rate of the DCX-MC360, the user may need to change the step rate range setting of an application program that used a DCX-MC260.

Defining User Units

When power is applied or the DCX is reset, it defaults to encoder counts or stepper pulses as its units for motion command parameters. If the user issues a move command to a servo with a target of 1000, the DCX will move the servo 1000 encoder counts. If the user issues the same command to a stepper motor, it will move 1000 motor steps.

In many applications there is a more convenient unit of measure than the encoder counts of the servo or steps of the stepper motor. If there is a fixed ratio between the encoder counts or steps and the desired 'user units', the DCX can be programmed with this ratio and it will perform conversions implicitly during command execution.

Defining user units is accomplished with the function ***MCSetScale( )***, which uses the **MCSCALE** data structure. This function provides a way of setting all scaling parameters with a single function call

using an initialized **MCSCALE** structure. To change scaling, call ***MCGetScale( )***, update the **MCSCALE** structure, and write the changes back using ***MCSetScale( )***.

MCScale Data Structure

```
typedef struct {


double    Constant;       // Define output constant
   double    Offset;             // Define the work area zero
   double    Rate;               // Define move (vel., accel, decel) time
units
   double    Scale;          // Define encoder scaling
   double    Zero;               // Define part zero
   double    Time;               // Define time scale

} MCMOTION;
```

Setting Move (Encoder/Step) Units

The value of the **Scale** member is the number of encoder counts or steps per user unit. For example, if the servo encoder on axis 1 has 1000 quadrature counts per rotation, and the mechanics move 1 inch per rotation of the servo, then to setup the controller for user units of inches:

```
MCSCALE Scaling;

MCGetScale( hCtlr, 3, &Scaling );
Scaling.Scale = 1000.0;                 // 1000 encoder counts/inch
MCSetScale( hCtlr, 3, &Scaling );
```

Prior to issuing the **Scale** member, the parameters to all motion commands for a particular axis are rounded to the nearest integer. After setting a new encoder scale and calling **MCEnableAxis( )** to initialize the axis, motion targets are multiplied by the ratio prior to rounding to determine the correct encoder position. Calling the MCGetPosition( ) will load the scaled encoder position.

Note – setting a user scale other than 1:1 will also scale trajectory settings (Velocity, acceleration, and deceleration) but not PID settings.

Trajectory Time Base

The value of the **Rate** member sets the time unit for velocity, acceleration and deceleration values, to a time unit selected by the user. If velocities are to be in units of inches per minute, the user time unit is a minute. The value of the **Rate** member is the number of seconds per 'user time unit'. If the velocity, acceleration and deceleration are to be specified in units of inches per minute and inches per minute per minute for axis 1, then the **Rate** value should be set to 60 seconds/1 minute = 60 (1UR60). The function **MCEnableAxis( ) must** be issued before the user rate will take effect.

```
MCSCALE Scaling;

MCGetScale( hCtlr, 3, &Scaling );
Scaling.Rate = 60.0;                    // set rate to inches per minute
MCSetScale( hCtlr, 3, &Scaling );
```

Typical Rate values

Time Unit	User Rate Conversion
second	1 (default)
minute	60
hour	3600

Defining the Time Base for Wait commands
For the ***MCWait( )***, ***WaitForStop( )*** and ***WaitForTarget( )*** functions, the default units are seconds. By setting the member **Time**, these three commands can be issued with parameters in units of the user's preference. The parameter to member is the number of 1 second periods in the user's unit of time. If the user prefers time parameters in units of minutes, **Time** = 60 should be issued.

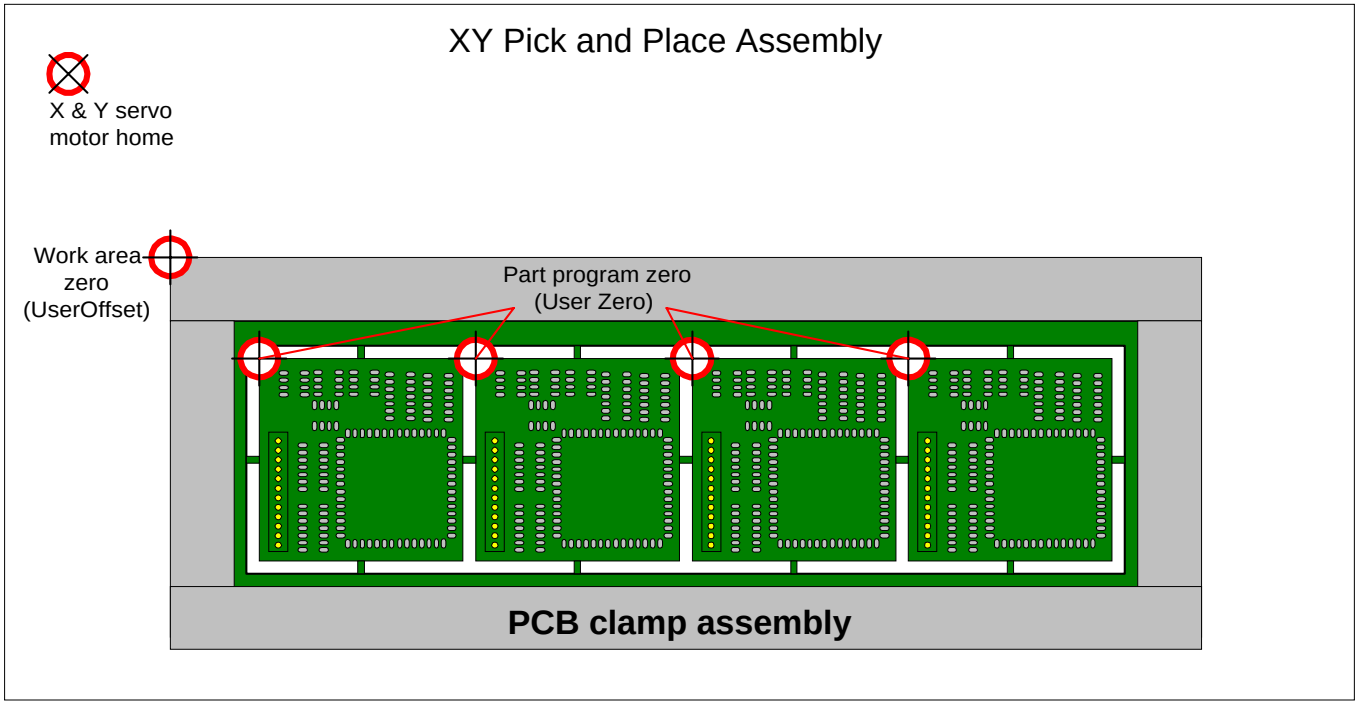
```
MCSCALE Scaling;

MCGetScale( hCtlr, &Scaling );
Scaling.Time = 60.0;                    // set Wait time unit to minutes
MCSetScale( hCtlr, &Scaling );
```

Defining a System/Machine zero
The member **Offset** allows the user to define a 'work area' zero position of the axis. The **Offset** value should be the distance from the servo or stepper motor home position, to the machine zero position. This offset distance must use the same units as currently defined by set User Scaling command. **Offset** does not change the index or home position of the servo or stepper motor, it only establishes an arbitrary zero position for the axis.

```
MCSCALE Scaling;

MCGetScale( hCtlr, 3, &Scaling );
Scaling.Offset = 12.25;                 // define offset to 12.25 inches
MCSetScale( hCtlr, 3, &Scaling );
```

Defining a Part Zero
The member **Zero** would typically be used in conjunction with **Offset** to define a 'part zero' position. A PCB (Printed Circuit Board) pick and place operation is a good example of how this function would be used. After a new PCB is loaded and clamped into place the X and Y axes would be homed. The **Offset** member is used to define the 'work area' zero of the PCB. The Zero member is used to define the 'part program' or 'local' zero position. This way a single 'part placement program' can be developed for the PCB type, and a 'step and repeat' operation can be used to assemble multiple part assemblies.

```
MCSCALE Scaling;

MCGetScale( hCtlr, 3, &Scaling );
Scaling.Offset = 12.25;                 // define offset to 12.25 inches
Scaling.Zero = 1.25;                    // define 'part zero' to 1.25 inches
MCSetScale( hCtlr, 3, &Scaling );
```



Defining the output constant for velocity gain
The member **Constant** allows the user to define the units to be used for setting the Velocity Gain parameters. Please refer to the description of **Using Velocity Gain** in the **Application Solutions** chapter of this user manual.

DCX Watchdog

The DCX incorporates a watchdog circuit to protect against improper CPU operation.

After a reset or power cycle, once the firmware (operational code) has been loaded by the operating system (approximately 3 seconds), the watchdog circuit is enabled.

If the DCX processor fails to properly execute firmware code for a period of 10 msec's, the watchdog circuit will 'time out' and the on-board reset will be latched by the ‘watchdog reset relay’. This in turn will hold the DCX modules in a constant state of reset. All motor outputs (+/- 10V & Step/Direction) will be disabled. When the watchdog circuit has tripped, the green **Run LED** will be disabled. To clear the watchdog error either:

Cycle power to the computer ***(recommended)***
Reset the computer

Chapter Contents

Chapter 7

General Purpose I/O

DCX Motherboard Digital I/O

The DCX-PCI300 Motion Controller motherboard has 16 general purpose digital I/O channels. Channels 1 – 8 are TTL inputs and channels 9 – 16 are TTL outputs. These signals can be accessed on connector J3 of the motherboard. The **DCX-PCI300** section of the **Connectors, Jumpers, and Schematics** chapter includes a pin-out for this connector. Each digital channel is configured via software (high true or low true).

Interfacing to the 'Outside World'
The TTL digital I/O channels can be connected directly to external circuits if output loading **(1ma maximum sink/source)** and input voltages **(0.0V to +5.0V)** are within acceptable limits.

The DCX Digital I/O channels are not suitable for driving optical isolators, relays solenoids, etc...

Alternatively, a DCX-BFO22 interface board can be used to connect the module's I/O to a relay rack in order to provide optically isolated inputs and outputs.

The DCX-BFO22 interface board provides a convenient means of connecting the DCX-PCI300 TTL digital I/O channels to a 16 position relay rack available from two manufacturers, Opto22 (P/N PB16H) and Grayhill (P/N 70RCK16-HL). These relay racks accept up to 16 optically isolated input or output modules for interfacing with external electrical systems. Using one of these relay racks and a DCX-BFO22, an optically isolated I/O module can be connected to each of the DCX's digital I/O channels.

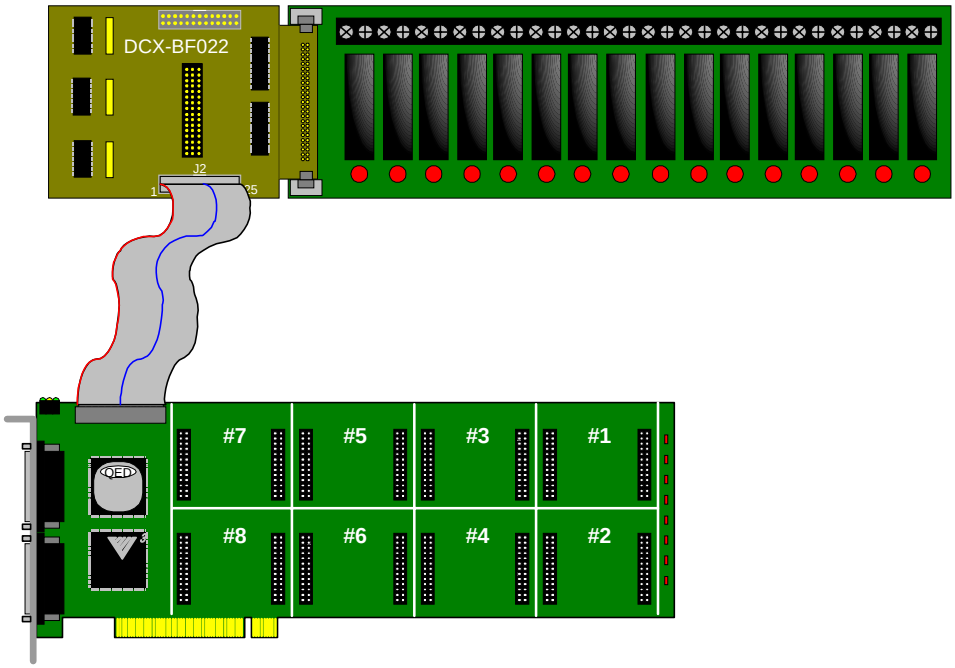


Figure 47:A DCX-BF022 is used to interface DCX digital I/O to an OPTO22 relay rack

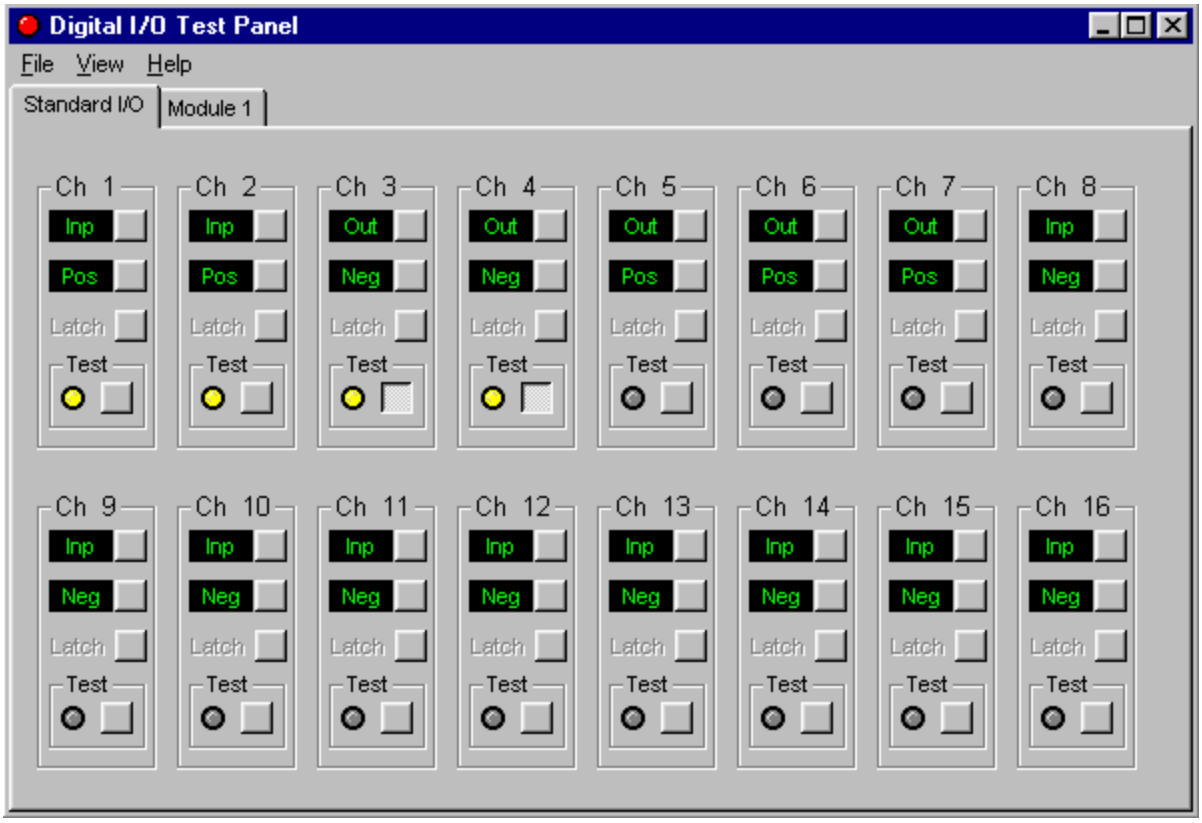
As shown above, the DCX-BFO22 plugs directly into the relay rack's 50 pin header connector and then connects to the DCX-PCI300 via a 26 conductor ribbon cable. Note that the relays are numbered sequentially starting from 0, while the DCX digital I/O channels are numbered sequentially starting with 1.

Although the relay rack has screw terminals for connecting a logic supply, it is not necessary to make this connection. By installing a shorting block on jumper JP17 of the BFO22, the 5 volt supply of the DCX will be supplied to the relay rack.

For detailed information on configuring the DCX-BF022, please refer to the schematic and jumper table in the DCX-BF022 Appendix in this user manual.

Configuring the DCX Digital I/O

The configuration of both the DCX-PCI300 and the DCX-MC400 digital I/O channels is accomplished using either PMC's Motion Integrator software or the MCAPI function ***MCConfigureDigitalIO( )***. The screen shot that follows shows the Motion Integrator Digital I/O test panel. This tool is used to both configure each I/O channel and then verify its operation. A comprehensive on-line help document is provided.



Each channel is individually programmable as:

Input (MC_DIO_INPUT) or Output (MC_DIO_OUTPUT)
High true/Positive logic (MC_DIO_HIGH) or Low true/Negative logic (MC_DIO_LOW)

The 16 channels of the DCX-PCI300 motherboard are defined as channels 1 – 16. If one or more DCX-MC400 Digital I/O modules are installed, the additional I/O channels are assigned to succeeding channel/numbers in blocks of 16 (e.g. 17-32, 33-48, etc.). All I/O channels accept the same configuration, monitoring and control.

Note – If a BFO22 interface and relay rack are connected to the DCX Digital I/O, a MC_DIO_LOW command set to ALL_AXES should be issued to the DCX. This will cause "normally open" relays to turn on when the Channel oN command is issued, and off when the Channel oFf command is issued.

This example configures all the digital I/O channels on a controller for output, then turns each channel on (in order) for a half second.

```
MCPARAM Param;

MCGetMotionConfig( hCtlr, &Param );

for (i = 1; i <= Param.DigitalIO; i++) {
   MCConfigureDigitalIO( hCtlr, i, MC_DIO_OUPUT | MC_DIO_HIGH );

for (i = 1; i <= Param.DigitalIO; i++) {
   MCEnableDigitalIO( hCtlr, i, TRUE );
   MCWait( hCtlr, 0.5 );
   MCEnableDigitalIO( hCtlr, i, FALSE );
}
```

Using the DCX Digital I/O

After configuring the Digital I/O channels, three MCAPI functions are available for activating and monitoring the digital I/O:

MCEnableDigitalIO( )	set digital output channel state
MCGetDigitalIO( )	get digital input channel state
MCWaitForDigitalIO( )	wait for digital input channel to reach specific state

Enable Digital IO

Turns the specified digital I/O on or off, depending upon the value of *bState*.

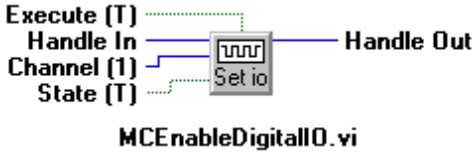
TRUE Turns the channel on.
FALSE Turns the channel off.

The I/O channel selected must have previously been configured for output using the ***MCConfigureDigitalIO( )*** command. Note that depending upon how a channel has been configured "on" (and conversely "off") may represent either a high or a low voltage level.

compatibility: MC400
see also: Configure Digital IO

C++ Function: void MCEnableDigitalIO(HCTRLR hCtlr, WORD wChannel, short int bState);
Delphi Function: procedure MCEnableDigitalIO(hCtlr: HCTRLR; wChannel: Word; bState: SmallInt);
VB Function: Sub MCEnableDigitalIO (ByVal hCtrlr As Integer, ByVal channel As Integer, ByVal state As Integer)
MCCL command: CF, CN

LabVIEW VI:



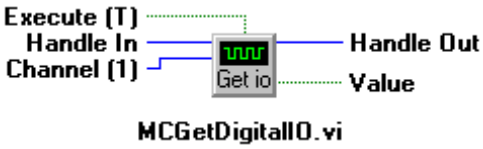
Get Digital IO

Returns the current state of the specified digital I/O channel. This function will read the current state of both input and output digital I/O channels. Note that this function simply reports if the channel is "on" or "off"; depending upon how a channel has been configured "on" (and conversely "off") may represent either a high or a low voltage level.

compatibility: MC400
see also:

C++ Function: short int MCGetDigitalIO(HCTRLR hCtlr, WORD wChannel);
Delphi Function: function MCGetDigitalIO(hCtlr: HCTRLR; wChannel: Word): SmallInt;
VB Function: Function MCGetDigitalIO (ByVal hCtrlr As Integer, ByVal channel As Integer) As Integer
MCCL command : TC

LabVIEW VI:



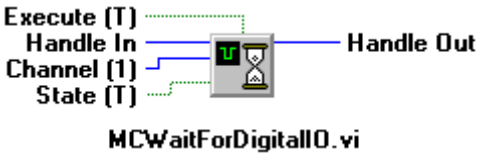
Wait for Digital IO

Waits for the specified digital I/O channel to go on or off, depending upon the value of bState.

compatibility: MC400
see also: Wait for digital channel on

C++ Function: void MCWaitForDigitalIO(HCTRLR hCtlr, WORD wChannel, short int bState);
Delphi Function: procedure MCWaitForDigitalIO(hCtlr: HCTRLR; wChannel: Word; bState: SmallInt);
VB Function: Sub MCWaitForDigitalIO (ByVal hCtrlr As Integer, ByVal channel As Integer, ByVal state As Integer)
MCCL command: WF, WN

LabVIEW VI:



This example configures all the digital I/O channels on a controller for output, then turns each channel on (in order) for a half second.

```
MCPARAM Param;

MCGetMotionConfig( hCtlr, &Param );

for (i = 1; i <= Param.DigitalIO; i++) {
   MCConfigureDigitalIO( hCtlr, i, MC_DIO_OUPUT | MC_DIO_HIGH );

for (i = 1; i <= Param.DigitalIO; i++) {
   MCEnableDigitalIO( hCtlr, i, TRUE );
   MCWait( hCtlr, 0.5 );
   MCEnableDigitalIO( hCtlr, i, FALSE );
}



//
//  Next re-configure channel 3 for input, and put up a message
//  box based on the input state
//
if (MCConfigureDigitalIO( hCtlr, 3, MC_DIO_INPUT | MC_DIO_HIGH )) {
   val = MCGetDigitalIO( hCtlr, 3 );
   if (val)            // MessageBox is a Windows API function
      MessageBox( hParent, "Channel 3 input voltage high (>2.4VDC)",
                  "MCAPI Sample", MB_ICONINFORMATION );
   else
      MessageBox( hParent, "Channel 3 input voltage low (<0.4VDC)",
                  "MCAPI Sample", MB_ICONINFORMATION );
   }
```

DCX Module Analog I/O

The DCX-MC500 Analog I/O Module provides additional analog I/O capability to a DCX Motion Controller. One or more of these modules can be installed in any available position on a DCX motherboard. Analog input channels can be used to monitor signal levels from external sensors. Output channels can be used to control external devices.

Three models of the DCX-MC500 are available:

Part Number	***Description***
DCX-MC500	4 Inputs and 4 Outputs
DCX-MC510	4 Inputs
DCX-MC520	4 Outputs

On each DCX-MC500/510 Analog I/O Module all analog input channels are numbered sequentially in groups of four. Likewise, all analog output channels are numbered sequentially in groups of four. When installed on the DCX-PCI300, the MC500/510 in the **lowest module location** will have its 4 analog input channels defined as 1 – 4. The four analog inputs of a MC500/510 installed in the next lowest module location will be defined as channels 5 – 8.

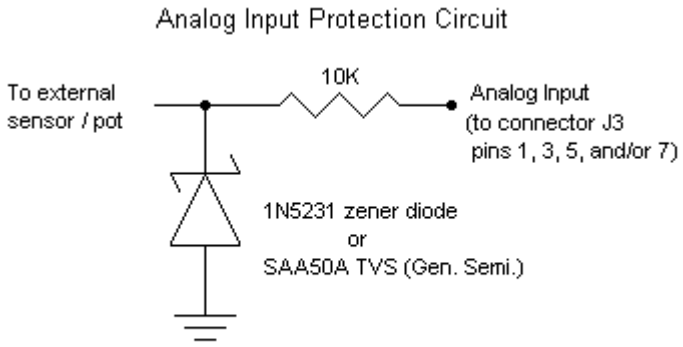
Because the DCX controller board is implemented in digital electronics, all analog input signals must be converted into a representative numerical value. This function is done by an Analog to Digital Converter (ADC) on the DCX-MC500/510. Similarly, analog output signals originate on the DCX board as numerical values. These numbers must be written to a Digital to Analog Converter (DAC) on the DCX-MC500/520, which converts them to a corresponding analog output signal level.

The DCX-MC500 is designed to accurately measure voltage levels on the input channels. These inputs are very high impedance with leakage currents less than 10 nano amps. The output channels are designed to provide signals with accurate voltage levels. The current requirement from these outputs should not exceed ***10 milliamps***.

Each of the analog input and analog output channels has 12 bits of resolution. This means that the digital value read from the ADC, or the digital value written to DAC, must be in the range 0 to 4095. For both inputs and outputs, a digital value of 0 translates to the lowest analog voltage. A digital value of 4095 translates to the highest analog voltage.

Input signals on pins 1, 3, 5 and 7 of the module J3 connector are wired directly to the ADC. No amplification or clamping to the input voltage range is provided on the module.

A voltage level greater than 5.6 volts will damage the analog input channels of a DCX-MC5X0 module. The schematic below is recommended to protect an analog input from damage due to an over voltage condition. This circuit will limit the maximum voltage applied to the A/D converter to 5.6 VDC.



In some applications, the signals from a sensor may not be absolute voltage levels, but proportional to some reference voltage. In these cases, it may be desirable to supply the reference signal to the ADC on the module through pin 18 of the J3 connector (and setting jumper JP1 accordingly). This will result in a "ratiometric" conversion of the input signal relative to the reference voltage.

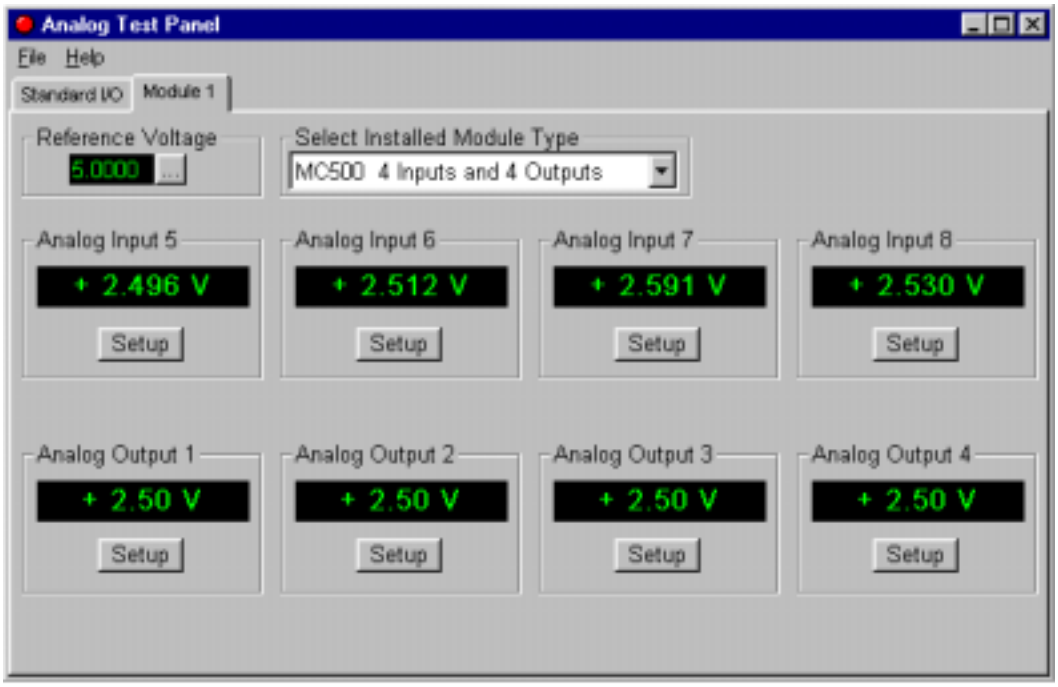
The outputs from the DAC on the DCX-MC500 module are voltage levels in the range 0 to +5 volts. These outputs have no gain or offset adjustment. These signals are available on pins 10, 12, 14 and 16 of the module J3 connector.

The outputs from the DAC are also connected to operational amplifiers on the module, which offset and amplify them to provide a +/-10 volt range. Each of these outputs has a 20 turn trim pot for offset adjustment, and a single turn pot for gain adjustment. The offset pot provides a minimum 0.5 volt adjustment, and the gain pot provides a nominal 2% range adjustment. These output signals are available on pins 2, 4, 6 and 8 of the module J3 connector.

After reset the outputs of the DCX-MC500 will be initialized to their mid-scale point. For the 0 to +5 volt outputs, this will be ***2.5 volts***. For the -10 to +10 volt outputs, this will be ***0.0*** volts.

Using the Analog I/O

The configuration and operation of the DCX-MC5X0 analog I/O channels is accomplished using either PMC's Motion Integrator program or the MCAPI functions ***MCSetAnalog( )*** , ***MCGetAnalog( )***. The screen capture that follows shows the Motion Integrator Analog I/O test panel. This tool is used to both configure each I/O channel and then verify its operation. A comprehensive on-line help document is provided.



Two MCAPI functions are available for setting and monitoring the MC500 analog I/O:

MCSetAnalog( )	set digital output channel state
MCGetAnalogIO( )	get digital input channel state

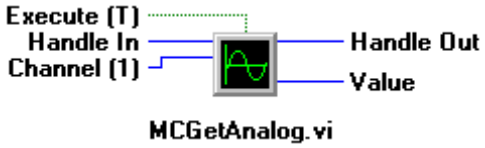
Get Analog

Reads the digitized input state of the specified input *wChannel*. The four 8-bit analog input channels accessed on connectors J3 are numbered 1,2,3 and 4. For each of these channels, this function will read a number between 0 and 255. These numbers are the ratio of the analog input voltage to the reference input voltage multiplied by 256. The reference voltage for the first four channels must be supplied to the DCX on the J3 connector pin 23, and can be any voltage between 0 and +5 volts DC. The analog input channels on any installed MC500 modules will be numbered sequentially starting with channel 5. See the description of **Analog Inputs** in the **DCX General Purpose I/O** chapter.

compatibility: MC500, MC510
see also: Set Analog

C++ Function: WORD MCGetAnalog(HCTRLR hCtlr, WORD wChannel);
Delphi Function: function MCGetAnalog(hCtlr: HCTRLR; wChannel: Word): Word;
VB Function: Function MCGetAnalog (ByVal hCtrlr As Integer, ByVal channel As Integer) As Integer
MCCL command: TA

LabVIEW VI:



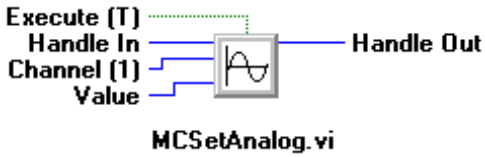
Set Analog

Sets the output level of an analog channel. Analog output ports on MC500 and MC520 Analog Modules accept values in the range of 0 to 4095 counts (12 bits). This range of values corresponds to an output voltage of 0 to 5V or -10 to +10V, depending upon how the output is configured (See the description of **Analog Inputs** in the **DCX General Purpose I/O** chapter).

compatibility: MC500, MC520
see also: Get Analog

C++ Function: void MCSetAnalog(HCTRLR hCtlr, WORD wChannel, WORD wValue);
Delphi Function: procedure MCSetAnalog(hCtlr: HCTRLR; wChannel, value: Word);
VB Function: Sub MCSetAnalog (ByVal hCtrlr As Integer, ByVal channel As Integer, ByVal Value As Integer)
MCCL command: OA

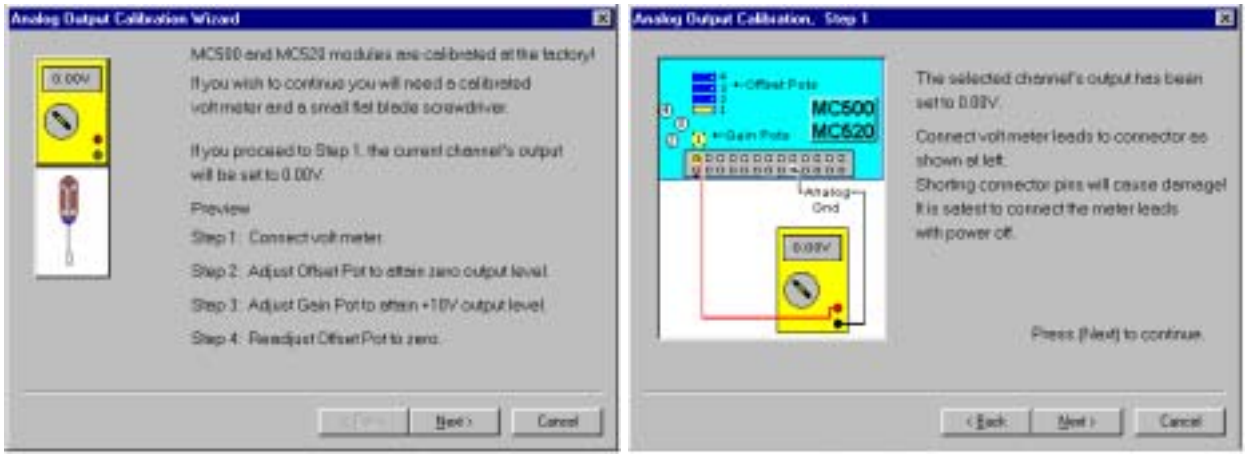
LabVIEW VI:



Calibrating the MC500/MC520 +/- 10V Analog Outputs:

The analog inputs of the DCX-MC500 require no calibration, and the only option is use of the internal +5, or an external, reference voltage. The analog outputs with the 0 to +5 volt range also have no adjustments. The reference for the DAC is fixed to the internal reference voltage.

The four 0.0 to +5.0 analog outputs require no calibration. The four +10 to –10 volt analog outputs are calibrated at the factory. There are four single turn trim pots that are used to adjust the gain of each of the four analog outputs. There are also four 20 turn trim pots for adjusting the offsets of each of the analog outputs. It is **strongly recommended** that the +10 to –10 volt outputs be calibrated using the **Motion Integrator Calibration Wizard**.



The analog outputs can also be calibrated using MCCL command sequences. For a description of **MCCL** commands and the **WinControl command interface utility** please refer to the MCCL section of the appendix at the end of this user manual. Refer to the module layout diagram in the **Connectors, Jumpers, and Schematics** chapter of this user manual. Using the following command sequence, and reading the analog output voltage level with a voltmeter, an analog output can be calibrated to provide the specified -10 to +10 volt range:

```
AL0,OAn,WA2,AL2048,OAn,WA2,AL4095,OAn,WA2,RP
```

where: n = channel number = 1, 2, 3, 4, ...

This command sequence will cycle the specified analog output from the minus limit, to the mid-point, to the positive limit. There is a 2 second delay at each voltage level, during which the voltmeter can settle and display the current reading.

The first step in calibrating an analog output is to adjust the gain using the single turn pot to achieve a 20.00 volt "swing". This is the difference between the most positive level reading, and the most negative level reading. It is not necessary for the two readings to be centered about 0 volts for this step.

The second step is to adjust the offset using the 20 turn pot. This adjustment will place the mid-point of analog output at the 0 volt level. When the output changes to the mid- point level turn the pot to achieve a 0.000 volt reading.

After the second step of the calibration procedure, the output swing should still be 20.00 volts. If not, repeat steps 1 and 2 again.

Chapter Contents

- Motherboard: DCX-PCI300
- DCX-MC300 - +/- 10 Volt Analog Servo Motor Control Module
- DCX-MC302 – Dual +/- 10 Volt Analog Servo Motor Control Module
- DCX-MC320 - Brushless Servo Commutation Control Module
- DCX-MC360 - Stepper Motor Control Module
- DCX-MC362 – Dual Stepper Motor Control Module
- DCX-MC400 - 16 channel Digital I/O Module
- DCX-MC5X0 - Analog I/O Module

Chapter 8

DCX Specifications

Motherboard: DCX-PCI300

Function	15 Axis Motion Controller
Installation	Intel PC compatible computer
Configuration	8 User Installed Modules
Main Processor	QED 5231 200MHz MIPS RISC
Processor Clock	192 MHz
Memory	512k x 8 bit Flash Memory
	1Meg X 32 Synchronous Dynamic Ram
Processor Fault Detection	Watchdog Circuit with Reset Relay
Status LED's	Power, Reset, Run, General Purpose (8)
Standard Communication Interface	PCI Bus 4 Kilobytes dual ported memory in Memory Address Space 'Plug and Play' dynamic addressing
Undedicated Digital I/O Channels	16 TTL (0 – 5 VDC), 1ma max. sink/source with 4.7K ohm pull up to +5V 2 groups (8 inputs, 8 outputs)
Connection options	DCX-PCI300-H - VHDCI Ultra SCSI (SCSI V) DCX-PCI300-R – 26 conductor, dual row, ribbon cable
Required Supply Voltages	+5,+12 and -12 vdc
Form Factor	Full Size PCI card (4.2" x 12.28")
Operating Temperature range	0 degrees C to 60 degrees C
Weight	10 oz + 1.2 oz per module (approx.)

DCX-MC300 - +/- 10 Volt Analog Servo Motor Control Module

Function	Closed Loop Servo Controller with Dual Encoder Inputs
Installation	DCX-PCI300 Motion Control Motherboard
Operating Modes	Position, Velocity, Contouring, Torque, and Gain
Filter Algorithm	PID with Velocity and Acceleration Feed-Forwards
Filter Update Rate	8, 4 or 2 KHz, software selectable
Trajectory Generator	Trapezoidal, Parabolic or S-Curve Independent Acceleration and Deceleration
Command output	Analog Signal (+/- 10 vdc @ 10 ma, 16 bit)
Position Feedback	Incremental Encoder with Index
Position and Velocity Resolution	32 bit
Primary Encoder	
Encoder and Index Inputs	Differential or single ended, -7 to +7 vdc max.
Encoder Count Rate	10,000,000 Quadrature Counts/Sec.
Encoder Supply Voltage	+5 or +12 vdc, jumper selectable
Auxiliary Encoder	
Encoder and Index Inputs	Differential or single ended, -7 to +7 vdc max.
Encoder Count Rate	10,000,000 Quadrature Counts/Sec.
Encoder Supply Voltage	+5 or +12 vdc, jumper selectable
Axis Inputs	Limit+, Limit-, Coarse Home, Amplifier Fault Optically isolated (Motorola MOC256)
Voltage range	+2.5V to +7.5V
Minimum current required	10 ma
Axis Outputs	Amplifier Enable, Direction Optically isolated Open Collector (Motorola MOC223)
Maximum voltage	30V
Maximum current sink	125ma
Connection options	DCX-MC300-H - VHDCI Ultra SCSI (SCSI V) DCX-MC300-R – 26 conductor, dual row, ribbon cable
Operating Temperature range	0 degrees C to 60 degrees C

DCX-MC302 – Dual +/- 10 Volt Servo Motor Control Module

Function	Dual Closed Loop Servo Controller
Installation	DCX-PCI300 Motion Control Motherboard
Operating Modes	Position, Velocity, Contouring, Torque, and Gain
Filter Algorithm	PID with Velocity and Acceleration Feed-Forwards
Filter Update Rate	8, 4 or 2 KHz, software selectable
Trajectory Generator	Trapezoidal, Parabolic or S-Curve Independent Acceleration and Deceleration
Command output	Axis 1 - Analog Signal (+/- 10 vdc @ 10 ma, 16 bit) Axis 2 - Analog Signal (+/- 10 vdc @ 10 ma, 16 bit)
Position Feedback	Incremental Encoder with Index
Position and Velocity Resolution	32 bit
Encoder	
Encoder and Index Inputs	Axis 1 - Differential or single ended, -7 to +7 vdc max. Axis 2 - Differential or single ended, -7 to +7 vdc max.
Encoder Count Rate	10,000,000 Quadrature Counts/Sec.
Encoder Supply Voltage	Axis 1 - +5 or +12 vdc, jumper selectable Axis 2 - +5 or +12 vdc, jumper selectable
Axis Inputs	Axis 1 - Limit+, Limit-, Coarse Home, Amplifier Fault Optically isolated (Seimens ILDC256) Axis 2 - Limit+, Limit-, Coarse Home, Amplifier Fault Optically isolated (Seimens ILDC256)
Voltage range	+2.5V to +7.5V
Minimum current required	10 ma
Axis Outputs	Axis 1 - Amplifier Enable Open Collector (TI 75453B) Axis 2 - Amplifier Enable Open Collector (TI 75453B)
Maximum voltage	30V
Maximum current sink	125ma
Connection options	DCX-MC302-H - VHDCI Ultra SCSI (SCSI V)
Operating Temperature range	0 degrees C to 60 degrees C

DCX-MC320 - Brushless Servo Commutation Control Module

Function	Closed Loop Servo Controller with Dual Encoder Inputs
Installation	DCX-PCI300 Motion Control Motherboard
Operating Modes	Position, Velocity, Contouring, Torque, and Gain
Filter Algorithm	PID with Velocity and Acceleration Feed-Forwards
Filter Update Rate	8, 4 or 2 KHz, software selectable
Trajectory Generator	Trapezoidal, Parabolic or S-Curve Independent Acceleration and Deceleration
Command output	Phase A (+/- 10 vdc @ 10 ma, 16 bit) Phase B (+/- 10 vdc @ 10 ma, 16 bit)
Position Feedback	Incremental Encoder with Index
Position and Velocity Resolution	32 bit
Primary Encoder	
Encoder and Index Inputs	Differential or single ended, -7 to +7 vdc max.
Encoder Count Rate	10,000,000 Quadrature Counts/Sec.
Encoder Supply Voltage	+5 or +12 vdc, jumper selectable
Hall Sensor / Auxiliary Encoder	
Encoder and Index Inputs	Differential or single ended, -7 to +7 vdc max.
Encoder Count Rate	10,000,000 Quadrature Counts/Sec.
Encoder Supply Voltage	+5 or +12 vdc, jumper selectable
Axis Inputs	Limit+, Limit-, Coarse Home, Amplifier Fault Optically isolated (Motorola MOC256)
Voltage range	+2.5V to +7.5V
Minimum current required	10 ma
Axis Outputs	Amplifier Enable, Optically isolated Open Collector (Motorola MOC223)
Maximum voltage	30V
Maximum current sink	125ma
Connection options	DCX-MC320-H - VHDCI Ultra SCSI (SCSI V) DCX-MC320-R – 26 conductor, dual row, ribbon cable
Operating Temperature range	0 degrees C to 60 degrees C

DCX-MC360 - Stepper Motor Control Module

Function	Open or Closed Loop Stepper Controller
Installation	DCX-PCI300 Motion Control Motherboard
Operating Modes	Position, Velocity, and Contouring
Trajectory Generator	Trapezoidal, Parabolic or S-Curve Independent Acceleration and Deceleration
Position Feedback	Incremental Encoder with Index (for closed loop stepper operation or position verification of an open loop stepper)
Position and Velocity Resolution	32 bit
Step Outputs	Pulse/Direction or CW/CCW (software selectable), 50% duty cycle open collector drivers (max. 30V, 125ma current sink)
Step Rates (Software Selectable)	High Speed - 153 Steps/Sec. - 5.0M Steps/Sec. Medium Speed - 20 Steps/Sec. - 625K Steps/Sec. Low Speed - .1 Steps/Sec. – 78K Steps/Sec.
Axis Inputs	Limit+, Limit-, Home, Drive Fault (Optically isolated Motorola MOC256)
Voltage range	+2.5V to +7.5V
Minimum current required	10 ma
Axis Outputs	Drive Enable, Full/Half Current (Open Collector TI 75453B)
Maximum voltage	30V
Maximum current sink	125ma
Connection options	DCX-MC360-H - VHDCI Ultra SCSI (SCSI V) DCX-MC360-R – 26 conductor, dual row, ribbon cable
Operating Temperature range	0 degrees C to 60 degrees C

DCX-MC362 – Dual Stepper Motor Control Module

Function	Dual Open Loop Stepper Controller
Installation	DCX-PCI300 Motion Control Motherboard
Operating Modes	Position, Velocity, and Contouring
Trajectory Generator	Trapezoidal, Parabolic or S-Curve Independent Acceleration and Deceleration
Position Feedback	None
Position and Velocity Resolution	32 bit
Step Outputs	Axis 1 - Pulse/Direction – CW/CCW (software selectable), 50% duty cycle, open collector drivers (max. 30V, 125ma current sink) Axis 2 - Pulse/Direction – CW/CCW (software selectable), 50% duty cycle, open collector drivers (max. 30V, 125ma current sink)
Step Rates (Software Selectable)	High Speed - 153 Steps/Sec. - 5.0M Steps/Sec. Medium Speed - 20 Steps/Sec. - 625K Steps/Sec. Low Speed - .1 Steps/Sec. – 78K Steps/Sec.
Axis Inputs	Axis 1 - Limit+, Limit-, Home, Drive Fault Optically isolated (Motorola MOC256) Axis 2 - Limit+, Limit-, Home, Drive Fault Optically isolated (Motorola MOC256)
Voltage range	+2.5V to +7.5V
Minimum current required	10 ma
Axis Outputs	Axis 1 - Drive Enable, Full/Half Current, Open Collector (TI 75453B) Axis 2 - Drive Enable, Full/Half Current, Open Collector (TI 75453B)
Maximum voltage	30V
Maximum current sink	125ma
Connection options	DCX-MC362-H - VHDCI Ultra SCSI (SCSI V)
Operating Temperature range	0 degrees C to 60 degrees C

DCX-MC400 - 16 channel Digital I/O Module

Function	16 Channel Digital I/O module
Installation	DCX-PCI300 Motion Control Motherboard
Channels	16, individually programmable as input s or outputs
Output low voltage (min)	0.0 volt
Output high voltage (min)	2.4 volt
Current sink	1 ma max
Current source	1 ma max.
Input Low voltage	-0.3V min. to 0.8V max.
Input High voltage	2.0V min. to 5.3V max.
Input termination	4.7K ohm pull up to +5V per channel
Relay rack interface	DCX-BF022
Connection options	DCX-MC400-H - VHDCI Ultra SCSI (SCSI V) DCX-MC400-R – 26 conductor, dual row, ribbon cable
Operating Temperature range	0 degrees C to 60 degrees C

.

DCX-MC5X0 - Analog I/O Module

Function	DCX-MC500 – 4 A/D channels, 4 D/A channels DCX-MC510 – 4 A/D channels DCX-MC520 – 4 D/A channels
Installation	DCX-PCI300 Motion Control Motherboard
Inputs resolution	12 bit
Input voltage range	0.0V to +5.0V
Output resolution	12 bit
Output voltage range	0.0V to +5.0V (@ 5ma), -10V to +10V (@ 5ma)
Output Offset Adjustment	20 turn trim pot
Output Full Scale Adjustment	single turn trim pot
Connection options	DCX-MC5__0-H - VHDCI Ultra SCSI (SCSI V) DCX-MC5__0-R – 26 conductor, dual row, ribbon cable
Operating Temperature range	0 degrees C to 60 degrees C

DCX-MC500 Electrical Specifications

Parameter	Min.	Max	Unit
Input Resolution	12		Bits
Input Conversion Rate		10	KHz
Input Zero Error			
Using Internal Reference		+/- 3	LSB
Using External Reference		+/- 1/2	LSB
Input Full-Scale Error			
Using Internal Reference		+/- 15	LSB
Using External Reference		+/- 1/2	LSB
Input Zero Temp. Coefficient		0.5	ppm/C
Input Differential Nonlinearity		+/- 1	LSB
Input Total Unadjusted Error			
Using Internal Reference		+/- 15	
Using External Reference		+/- 1	
Input Voltage Range			
Using Internal Reference	0.0	5.0	
Using External Reference	0.0	Vref	
Input Capacitance		8	
Input Leakage Current		100	
External Reference Voltage	4.0	6.0	

Parameter	Min.	Max	Unit
Output Resolution	12		Bits
Output Zero Code Error *			LSB
Output Full Scale Error *			LSB
Output Nonlinearity *			LSB
Output Total Unadjusted Error *			LSB
Output Voltage Range	0.0	5.0	V
	-10.0	+10.0	V

* These values are for 0 to +5.0 volt outputs

Chapter Contents

Chapter 9

Connectors, Jumpers, and Schematics

DCX-PCI300 Motion Control Motherboard

Status LED Indicators

LED #	*Color*	*Description*
D1	Green	+5 VDC logic supply OK
D2	Yellow	DCX Reset active
D3	Green	Run (processor fault or watchdog tripped if off)
L1	Red	Motor Module #1 initialization error (will blink when reset)
L2	Red	Motor Module #2 initialization error (will blink when reset)
L3	Red	Motor Module #3 initialization error (will blink when reset)
L4	Red	Motor Module #4 initialization error (will blink when reset)
L5	Red	Motor Module #5 initialization error (will blink when reset)
L6	Red	Motor Module #6 initialization error (will blink when reset)
L7	Red	Motor Module #7 initialization error (will blink when reset)
L8	Red	Motor Module #8 initialization error (will blink when reset)

(Refer to diagram at the end of this appendix)

General Purpose I/O (Digital I/O and Analog inputs) Connector J5

Pin #	*Description*
1	+5 VDC
2	RESET RELAY CONTACT #1 *
3	DIGITAL OUTPUT CHANNEL 16
4	RESET RELAY CONTACT #2 *
5	DIGITAL OUTPUT, CHANNEL 15
6	DIGITAL OUTPUT, CHANNEL 14
7	DIGITAL OUTPUT, CHANNEL 13
8	DIGITAL OUTPUT, CHANNEL 12
9	DIGITAL OUTPUT, CHANNEL 11
10	DIGITAL OUTPUT, CHANNEL 10
11	DIGITAL OUTPUT, CHANNEL 09
12	DIGITAL INPUT, CHANNEL 08
13	DIGITAL INPUT, CHANNEL 07
14	DIGITAL INPUT, CHANNEL 06
15	DIGITAL INPUT, CHANNEL 05
16	DIGITAL INPUT, CHANNEL 04
17	DIGITAL INPUT, CHANNEL 03
18	DIGITAL INPUT, CHANNEL 02
19	DIGITAL INPUT, CHANNEL 01
20	NO CONNECT
21	+12 VDC
22	NO CONNECT
23	NO CONNECT
24	GROUND
25	-12 VDC
26	GROUND

Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N 26IDS2-C-SPT-SR or equivalent

*** -** Reset Relay contacts (normally open). The relay is energized (contacts 1 and 2 connected) when the DCX-PCI300 is held in reset.

Alternative +12 volt supply connector (not supported at this time)

Pin #	*Description*
1	
2	
3	
4	

Mating Connector:_____

J31 – +12 volt supply input select

Pins	*Description*
Open	+12 volt supply provided via connector J33
2 to 3	**+12 volt supply provided via PCI bus**

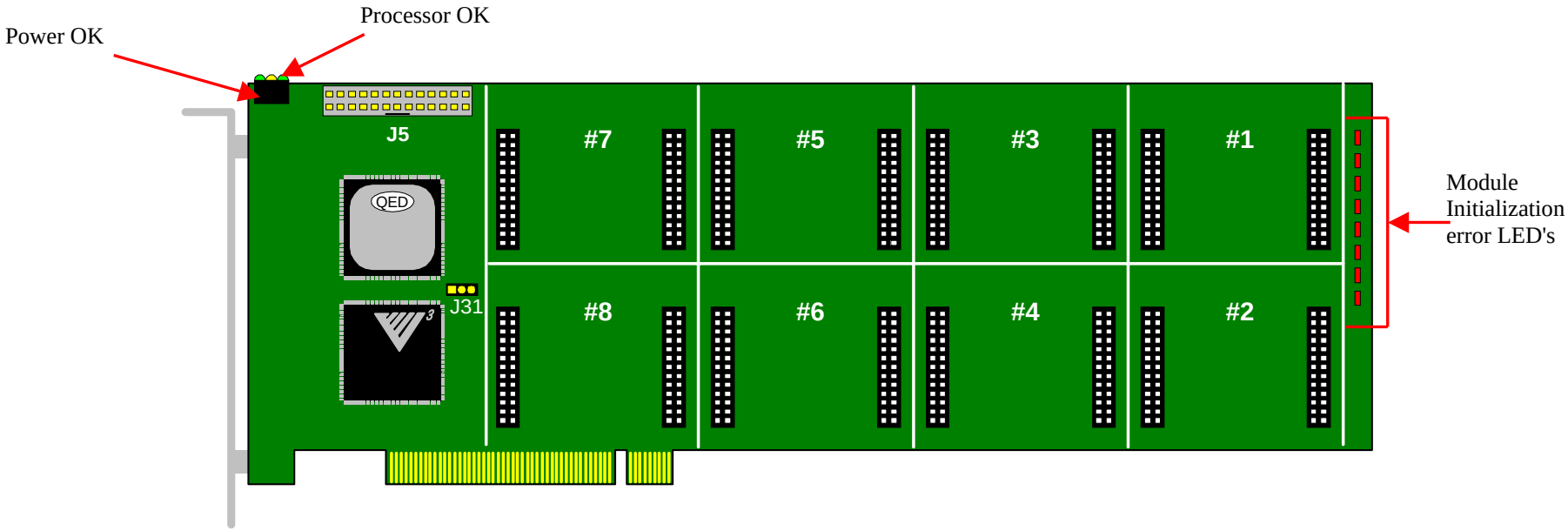
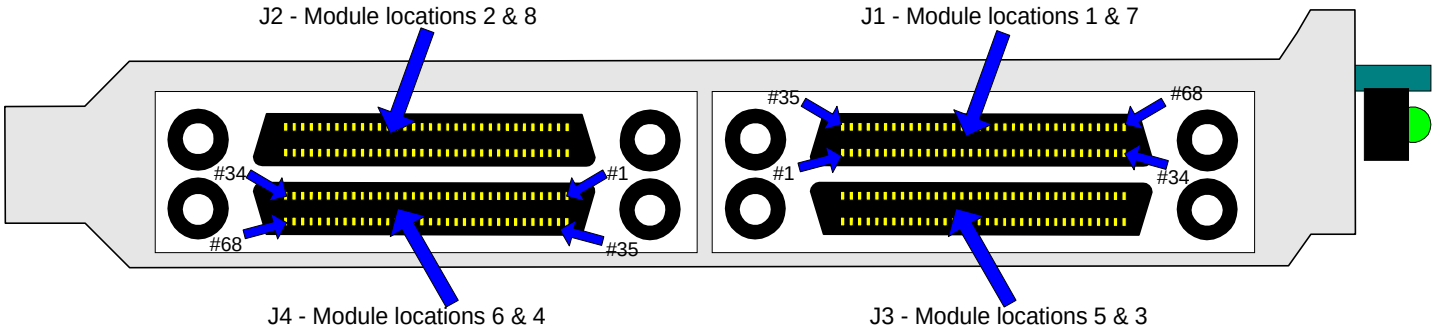


Figure 48: DCX-PCI300-R motherboard (ribbon cable version)



VHDCI connectors as viewed from the back of the computer
(component side down)

Figure 49: DCX-PCI300-H high density connectors pin numbering

DCX-MC300 +/- 10V Servo Motor Control Module

SIGNAL DESCRIPTIONS:

Analog Command Return
connection point: MC300-H J3 - pin 1, MC300-R J3 - pin 1
signal type: ground
notes:
explanation: Provides the signal ground for the modules Analog Command Signal output. This return path is common to the ground plane of the DCX motherboard, but is connected in such a way as to reduce digital noise. Typical servo amplifiers will have a connection for the analog command (or Ref-) return where this signal should be connected.

Analog Command Output
connection point: MC300-H J3 - pin 2, MC300-R J3 - pin 2
signal type: +/- 10V analog, 16 bit
notes: connects to servo amplifier motor command input (Ref+)
explanation: This module output signal is used to control the servo amplifier's output. When connected to the command input of a velocity mode amplifier, the voltage level on this signal should cause the amplifier to drive the servo at a proportional velocity. For current mode amplifiers, the voltage level should cause a proportional current to be supplied to the servo. In its default Bipolar output mode, the module provides an analog signal that is in the range -10 to +10 volts, with 0 volts being the null output level. Positive voltages indicate a desired velocity or current in one direction. Negative voltages indicate velocity or current in the opposite direction. By using the Output Mode command, the output can be changed to Unipolar, where the analog signal range is 0 to +10 volts, and a separate signal is used to indicate the desired direction of velocity or current. The maximum drive current of this signal is +/-10 milliamps.

Compare / Direction Output
connection point: MC300-H J3 - pin 3, MC300-R J3 - pin 7
signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: external pull-up required
explanation:
Compare – Used to indicate when a position compare event has occurred. See the description of **Position Compare** in the **Application Solutions chapter**.

Direction - For servo drives requiring a Unipolar output. The velocity or current command input consists of a magnitude signal and a separate direction signal . The magnitude signal is provided by the modules Analog Command Signal (J3 pin 2) previously described, while this signal provides a digital direction command.

This signal is driven by a high current open collector driver and is suitable for direct connection to optically isolated inputs commonly found on a servo amplifier. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless pull-up resistors are connected to them. When the axis is moving in the positive direction the output will be pulled low. When the axis is moving in the negative direction the open collector driver will be turned off and the output will be pulled high.

Coarse Home Input
connection point: MC300-H J3 - pin 9, MC300-R J3 - pin 9
signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range
notes: Supply/Return using INPRET (J3 pin 18)
explanation: This module input is used to determine the proper zero position of the servo. In servo systems that use rotary encoders with index outputs, an index pulse is generated once per rotation of the encoder. While this signal occurs at a very repeatable angular position on the encoder, it may occur many times within the motion range of the servo. In these cases, a Coarse Home switch connected to this module input can be used to qualify which index pulse is the true zero position of the servo. By setting this switch to be activated near the end of travel of the servo, and using DCX motion commands to position the servo within this region prior to searching for the index pulse, a unique zero position for the servo can be determined. The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W). The minimum current required to turn on the optical isolator is **10ma**. Bi-directional optical isolator wiring examples are provided later in this section.

Amplifier Fault Input
connection point: MC300-H J3 - pin 7, MC300-R J3 - pin 10
signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range
notes: Supply/Return using AMPFRET (J3 pin 13)
explanation: - This module input is designed to be connected to the servo amplifiers Fault or Error output signal. The state of this signal will appear as a status bit in the servo's status word. The **EnableAmpFault** member of the **MCMotion** structure will enable the module to shut off the axis if the Amplifier Fault input is active. No further servo motion will occur until the fail signal is deactivated and the axis is enabled. The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Amplifier Enable Output
connection point: MC300-H J3 - pin 5, MC300-R J3 - pin 11
signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: external pull-up required
explanation: - This module output signal should be connected to the enable input of the servo amplifier. When the DCX is turned on or reset, this signal will immediately go to its' inactive high level. When the ***MCEnableAxis( )*** is called, this signal will go to its' active low level. Anytime there is an error on the respective servo axis, including ***exceeding the following error, a limit switch input activated or the Amplifier Fault input activated***, the Amplifier Enable signal will be deactivated. This signal can also be deactivated by the Motor oFf command.

This signal is driven by a high current open collector driver and is suitable for direct connection to optically isolated inputs commonly found on a servo amplifier. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless pull-up resistors are connected to them.

Limit Positive and Limit Negative Inputs
connection point: Limit Positive: MC300-H J3 - pin 17, MC300-R J3 - pin 14
Limit Negative: MC300-H J3 - pin 19, MC300-R J3 - pin 15
signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range
notes: MC300-H Limit Positive Supply/Return J3 pin 18
MC300-H Limit Negative Supply/Return J3 pin 20
MC300-R Limits Supply/Return J3 pin 18
explanation: The limit switch inputs are used to cause the DCX to stop a servo's motion when it reaches the end of travel. If the servo is in position mode, the axis will only be stopped if it is moving in the direction of an activated limit switch. In all other modes, the servo will be stopped regardless of the direction it is moving if either limit switch is activated. There are three modes of stopping (decelerate to a stop, stop immediately, turn off the axis) that can be configured by the ***MCSetLimits( )***. The limit switch inputs can be enabled and disabled by ***MCSetLimits( )***. See the description of **Motion Limits** in the **Motion Control** chapter.

The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Position Capture / Auxiliary Encoder Index +
connection point: MC300-H J3 - pin 15, MC300-R J3 - pin 24
signal type: TTL or Differential driver output (-7V to +7V)
notes:
explanation: -
Position Capture – Used to initiate the capture of position data. See the description of **Position Capture** in the **Application Solutions chapter**.

Auxiliary Encoder Index + - This input signal can be used to define the home position of an auxiliary encoder.

Primary Encoder Inputs (Phase A+, Phase -, Phase B+, Phase B-, Index+, Index-)
connection point: see pin-out table
signal type: TTL or Differential driver output (-7V to +7V)
notes: The encoder power jumper JP3 sets the ‘mid point’ for the differential receiver
explanation: These input signals should be connected to an incremental quadrature encoder for supplying position feedback information for the servo controller. The plus (+) and minus (-) signs refer to the two sides of differential inputs. By setting jumpers JP1 and JP2 appropriately, the plus signal inputs can be configured for single ended inputs.

Auxiliary Encoder Inputs (Phase A, Phase B, Index+, Index-)
connection point: see pin-out table
signal type: TTL or Differential driver output (-7V to +7V)
notes:
explanation: - These input signals can be used for an auxiliary encoder.

Encoder Power Output
connection point: J3 pin 17MC300-H J3 - pin 16, MC300-R J3 - pin 17
signal type: +5 VDC PC power supply output or +12 VDC PC power supply output
notes: The encoder power jumper JP3 selects +5VDC or +12VDC (**max. load 250 mA**).
explanation: This module pin provides a convenient supply voltage connection for the encoders. The jumper JP3 located on the module can be used to connect either the +5 or +12 volt supply to the Encoder Power pin. The setting of this jumper also selects the threshold voltage for the module's single ended phase and index encoder inputs. When JP1 is set for +5 volts, the threshold will be 2.5 volts, for +12 volts, the threshold will be +6 volts. The threshold voltage determines at what voltage the input changes between on and off.

SUPPLY CONNECTIONS (+5, +12, -12, GROUND) - These module pins provide access to the DCX supply voltages.

DCX-MC300-H High Density connector signal map

Module #1	Module #2	Module #3	Module #4	Module #5	Module #6	Module #7	Module #8	J3 Pin #	Description
J1 – 1	J2 – 1	J3 - 19	J4 - 19	J3 – 1	J4 – 1	J1 – 19	J2 - 19	1	Analog Command return
J1 – 35	J2 - 35	J3 – 53	J4 – 53	J3 – 35	J4 – 35	J1 – 53	J2 – 53	2	Analog Command output
J1 – 2	J2 – 2	J3 – 20	J4 – 20	J3 – 2	J4 – 2	J1 – 20	J2 – 20	3	Compare / Direction: output
J1 – 36	J2 - 36	J3 – 54	J4 – 54	J3 – 36	J4 – 36	J1 – 54	J2 – 54	4	Compare / Direction return
J1 – 3	J2 – 3	J3 - 21	J4 - 21	J3 – 3	J4 – 3	J1 – 21	J2 - 21	5	Amplifier Enable: output
J1 – 37	J2 - 37	J3 – 55	J4 – 55	J3 – 37	J4 – 37	J1 – 55	J2 – 55	6	Amp Enable return
J1 – 4	J2 – 4	J3 – 22	J4 – 22	J3 – 4	J4 – 4	J1 – 22	J2 – 22	7	Amplifier Fault: input
J1 – 38	J2 - 38	J3 – 56	J4 – 56	J3 – 38	J4 – 38	J1 – 56	J2 – 56	8	Amp Fault opto isolator supply/return
J1 – 5	J2 – 5	J3 – 23	J4 – 23	J3 – 5	J4 – 5	J1 – 23	J2 – 23	9	Coarse Home: input
J1 – 39	J2 - 39	J3 – 57	J4 – 57	J3 – 39	J4 – 39	J1 – 57	J2 – 57	10	Coarse Home return
J1 – 6	J2 – 6	J3 – 24	J4 – 24	J3 – 6	J4 – 6	J1 – 24	J2 – 24		Ground
J1 – 40	J2 - 40	J3 – 58	J4 – 58	J3 – 40	J4 – 40	J1 – 58	J2 – 58	11	Reserved
J1 – 7	J2 – 7	J3 – 25	J4 – 25	J3 – 7	J4 – 7	J1 – 25	J2 – 25	12	Reserved
J1 – 41	J2 - 41	J3 – 59	J4 – 59	J3 – 41	J4 – 41	J1 – 59	J2 – 59		Ground
J1 – 8	J2 – 8	J3 – 26	J4 – 26	J3 – 8	J4 – 8	J1 – 26	J2 – 26		Ground
J1 – 42	J2 - 42	J3 – 60	J4 – 60	J3 – 42	J4 – 42	J1 – 60	J2 – 60	13	Auxiliary Encoder Phase A+: input
J1 – 9	J2 – 9	J3 – 27	J4 – 27	J3 – 9	J4 – 9	J1 – 27	J2 – 27	14	Auxiliary Encoder Phase B+: input
J1 – 43	J2 - 43	J3 – 61	J4 – 61	J3 – 43	J4 – 43	J1 – 61	J2 – 61		Ground
J1 – 10	J2 - 10	J3 – 28	J4 – 28	J3 – 10	J4 – 10	J1 – 28	J2 – 28		Ground
J1 – 44	J2 - 44	J3 – 62	J4 – 62	J3 – 44	J4 – 44	J1 – 62	J2 – 62	15	Position Capture + / Aux. Encoder Index+
J1 – 11	J2 - 11	J3 - 29	J4 - 29	J3 – 11	J4 – 11	J1 – 29	J2 - 29	16	Encoder Power: output (**max. load 250 mA**)
J1 – 45	J2 - 45	J3 – 63	J4 – 63	J3 – 45	J4 – 45	J1 – 63	J2 – 63		Ground
J1 – 12	J2 - 12	J3 – 30	J4 – 30	J3 – 12	J4 – 12	J1 – 30	J2 – 30	17	Limit Positive: input
J1 – 46	J2 - 46	J3 – 64	J4 – 64	J3 – 46	J4 – 46	J1 – 64	J2 – 64	18	Limit Positive opto isolator supply/return
J1 – 13	J2 - 13	J3 – 31	J4 – 31	J3 – 13	J4 – 13	J1 – 31	J2 – 31	19	Limit Negative: input
J1 – 47	J2 - 47	J3 – 65	J4 – 65	J3 – 47	J4 – 47	J1 – 65	J2 – 65	20	Limit Negative opto isolator supply/return
J1 – 14	J2 - 14	J3 – 32	J4 – 32	J3 – 14	J4 – 14	J1 – 32	J2 – 32	21	Primary Encoder Phase A+: input *
J1 – 48	J2 - 48	J3 – 66	J4 – 66	J3 – 48	J4 – 48	J1 – 66	J2 – 66	22	Primary Encoder Phase A-: input
J1 – 15	J2 - 15	J3 – 33	J4 – 33	J3 – 15	J4 – 15	J1 – 33	J2 – 33	23	Primary Encoder Phase B+: input*
J1 – 49	J2 - 49	J3 – 67	J4 – 67	J3 – 49	J4 – 49	J1 – 67	J2 – 67	24	Primary Encoder Phase B-: input
J1 – 16	J2 - 16	J3 – 34	J4 – 34	J3 – 16	J4 – 16	J1 – 34	J2 – 34	25	Primary Encoder Index +:input
J1 – 50	J2 - 50	J3 – 68	J4 – 68	J3 – 50	J4 – 50	J1 – 68	J2 – 68	26	Primary Encoder Index -:input
J1 – 17	J2 – 17			J3 – 17	J4 – 17				Ground
J1 – 51	J2 - 51			J3 – 51	J4 – 51				Ground
J1 – 18	J2 – 18			J3 – 18	J4 – 18				Ground
J1 – 52	J3 – 52			J3 – 52	J4 – 52				Ground

For a more complete signal description please refer to the previous five pages
Mating cable connector components: Amp: 787801-1 (offset connector), 788362-1 (backshell, offset)

DCX-MC300-R Module connector

J3 connector pin-out (Motor command, encoders, and axis I/O)

Pin #	Description
1	Analog Command return (analog ground)
2	Analog Command output (output, +/-10 V)
3	+12 VDC (**250 mA max.**)
4	-12 VDC (**50 mA max.**)
5	Ground
6	+5 VDC (**250 mA max.**)
7	Compare / Direction: output (open collector, 100ma max., 30V max.)
8	Primary Encoder Index +:input (active high)
9	Coarse Home: input (optically isolated, 12V – 24V, 15ma min.)
10	Amplifier Fault: input (optically isolated, 12V – 24V, 15ma min.)
11	Amplifier Enable: output (open collector, 100ma max., 30V max.)
12	Amp Enable & Direction return
13	Amp Fault opto isolator supply/return
14	Limit Positive: input (optically isolated, 12V – 24V, 15ma min.)
15	Limit Negative: input (optically isolated, 12V – 24V, 15ma min.)
16	Primary Encoder Phase A+: input *
17	Encoder Power: output (+5VDC or +12VDC, see jumper JP3) (**max. load 250 mA**)
18	Coarse Home & Limits opto isolator supply/return
19	Primary Encoder Phase A-: input
20	Primary Encoder Phase B-: input
21	Auxiliary Encoder Phase A+: input
22	Auxiliary Encoder Phase B+: input
23	Primary Encoder Phase B+: input*
24	Position Capture + / Auxiliary Encoder Index+: input (active high)
25	Primary Encoder Index-: input (active low)
26	Ground

* Use A+ and B+ for single-ended ENCODER INPUTS

Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N 26IDS2-C-SPT-SR or equivalent

DCX-MC300 Module Configuration Jumpers - configuration in **bold type** denotes default factory shipping configuration

JP1 – Encoder type (single ended or differential)

Pins	*Description*
1 to 2 to 3	Single ended encoder, A, B, Z (three pin jumper provided)
open	**Differential encoder, A+, A-, B+, B-**

JP2 – Encoder Index Active Level Select)

Pins	*Description*
1 to 2	Single ended Index, Z+ (Active high)
2 to 3	**Single ended Index, Z- (active low)**
open	Differential Index, Z+ and Z-

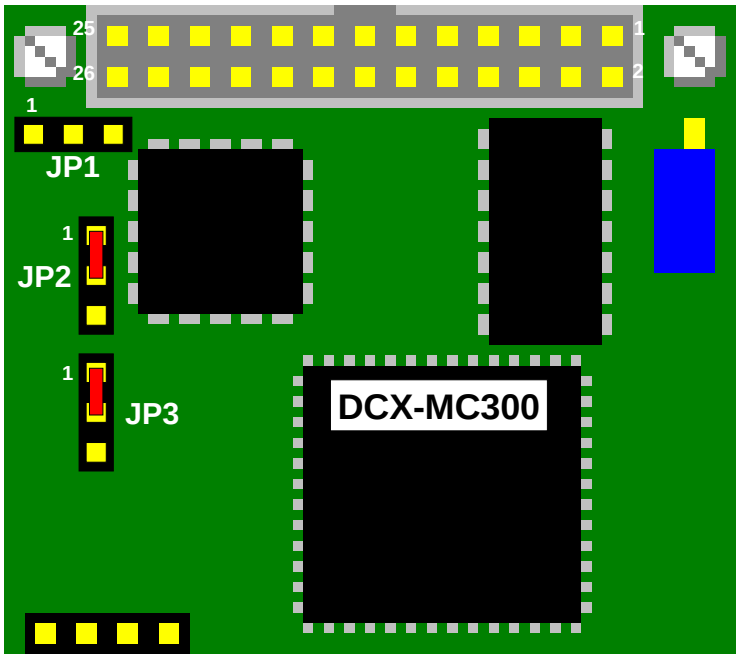
JP3 – Encoder Power Select (+5VDC or +12 VDC)

Pins	*Description*
1 to 2	**+5 VDC encoder supply on J3 pin 16/17** (**250 mA max.**)
2 to 3	+12 VDC encoder supply on J3 pin 16/17 (**250 mA max.**)

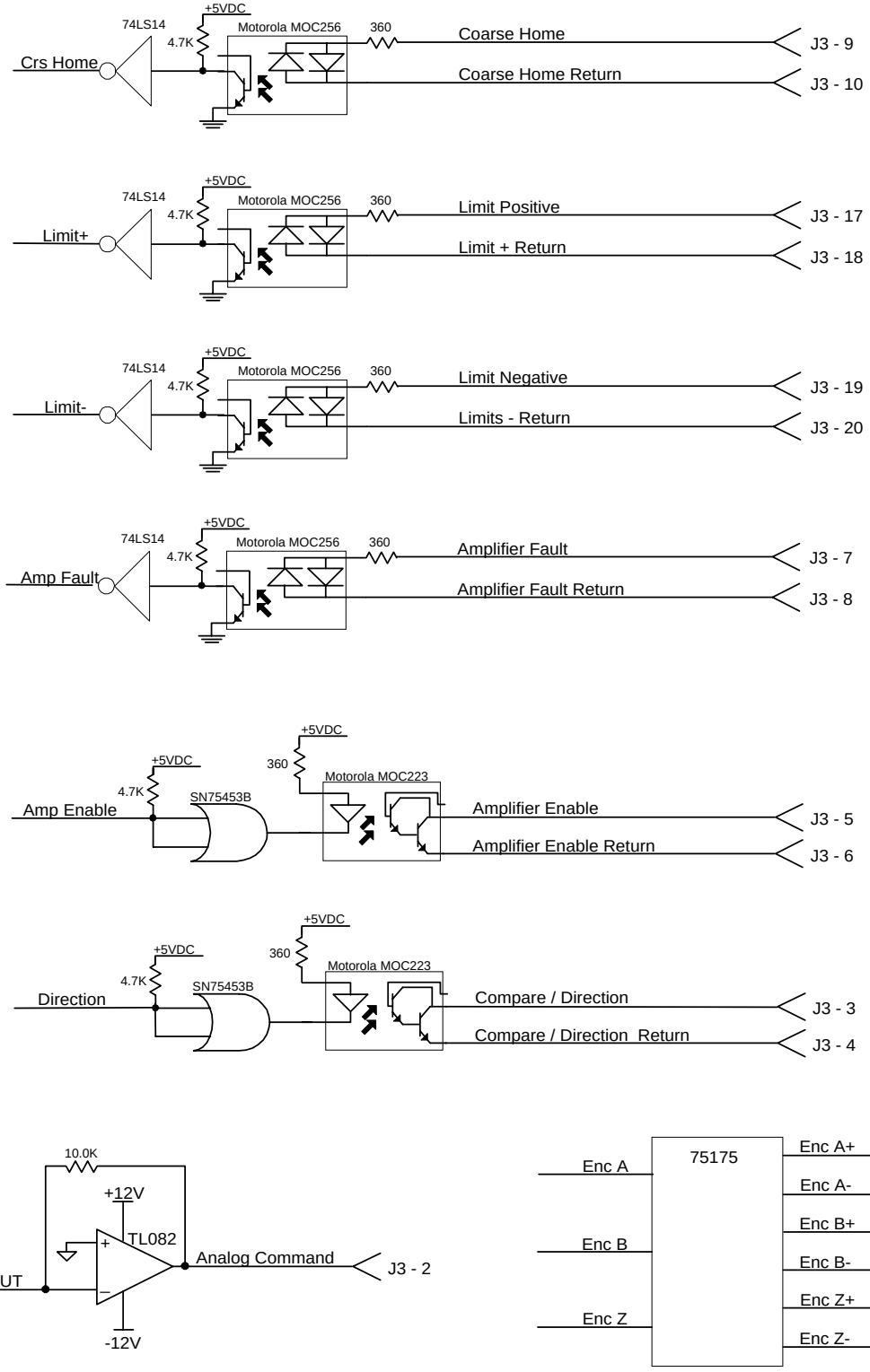
DCX-MC300 Module Output Offset Potentiometer

This multi-turn trimming potentiometer can be used to add an offset to the module's analog output. The range of this adjustment is approximately +/-1.0 volts.

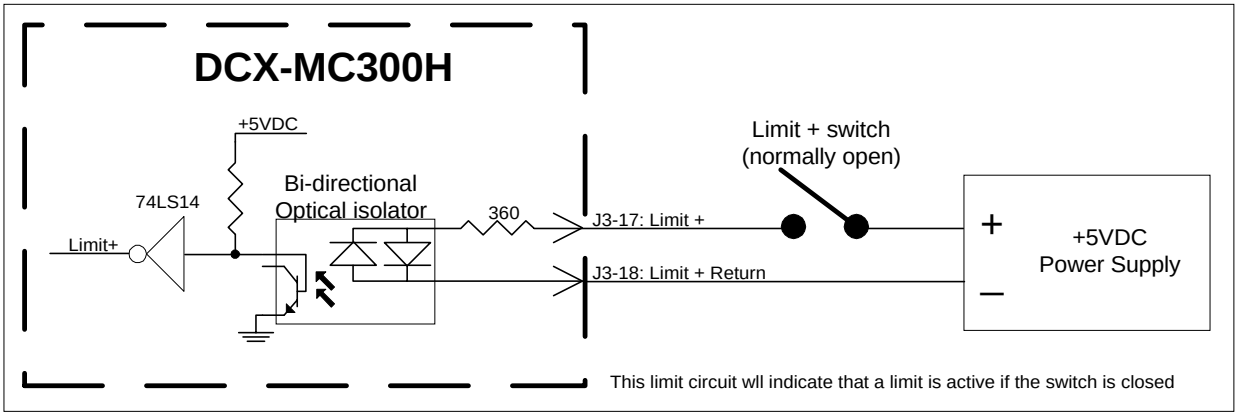
DCX-MC300 Module Layout

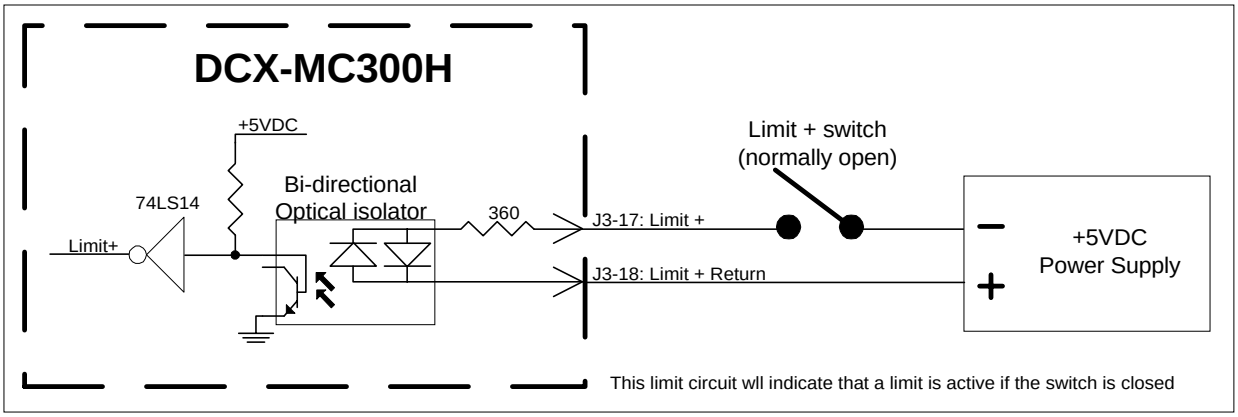


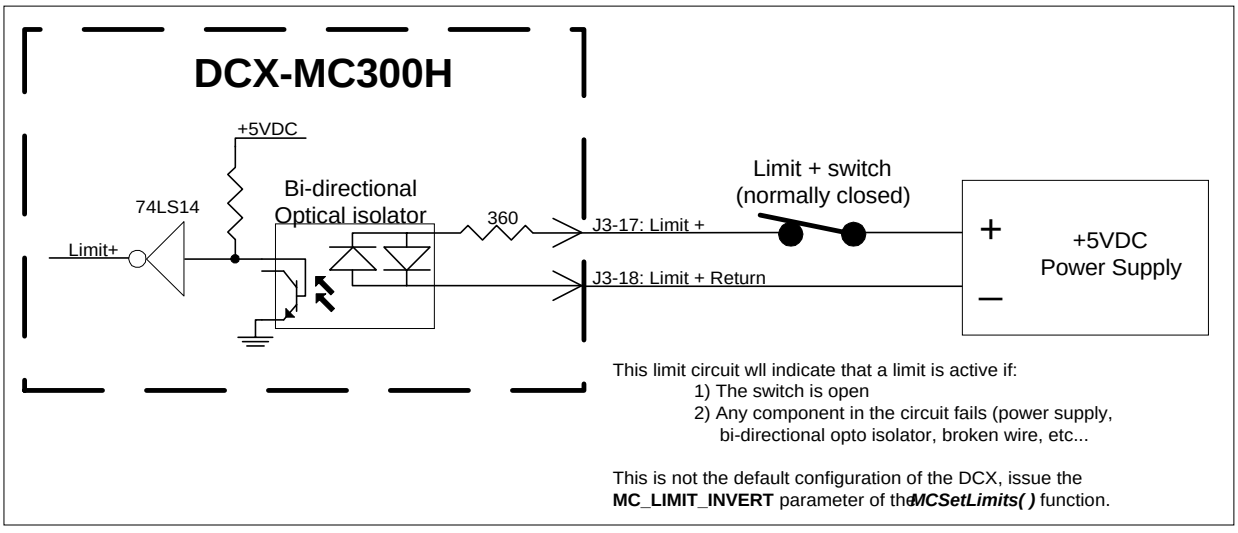
DCX-MC300H Axis I/O Interface Schematic



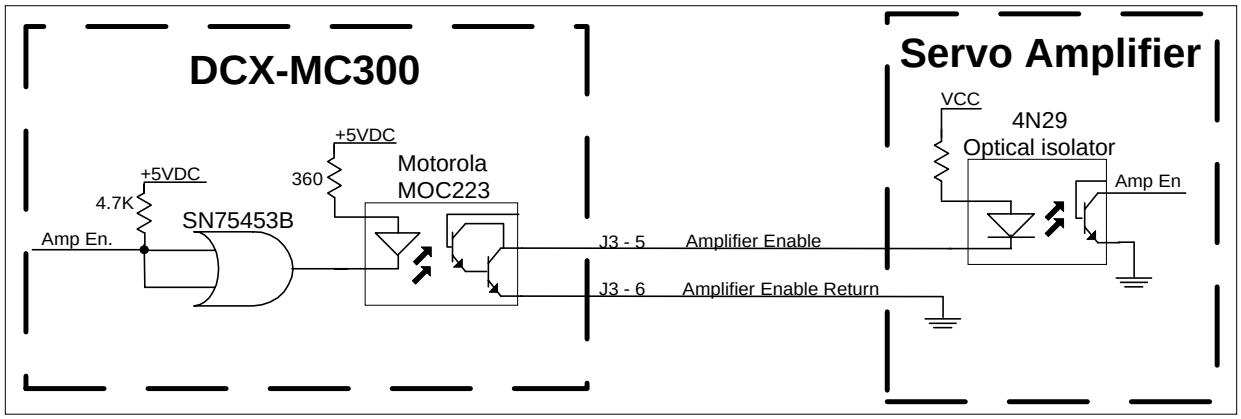
DCX-MC300H Optically Isolated Inputs Wiring Examples







DCX-MC300H Open Collector Driver Wiring Examples



DCX-MC302 Dual Axis +/- 10V Servo Motor Control Module

SIGNAL DESCRIPTIONS:

Analog Command Return
connection point: Axis 1: VHDCI connector pin 8 (no connection on module J3 connector)
Axis 2: VHDCI connector pin 43 (no connection on module J3 connector)
signal type: ground
notes:
explanation: Provides the signal ground for the modules Analog Command Signal output. This return path is common to the ground plane of the DCX motherboard, but is connected in such a way as to reduce digital noise. Typical servo amplifiers will have a connection for the analog command (or Ref-) return where this signal should be connected.

Analog Command Output
connection point: Axis 1: J3 - pin 13
Axis 2: J3 – pin 14
signal type: +/- 10V analog, 16 bit
notes: connects to servo amplifier motor command input (Ref+)
explanation: This module output signal is used to control the servo amplifier's output. When connected to the command input of a velocity mode amplifier, the voltage level on this signal should cause the amplifier to drive the servo at a proportional velocity. For current mode amplifiers, the voltage level should cause a proportional current to be supplied to the servo. In its default Bipolar output mode, the module provides an analog signal that is in the range -10 to +10 volts, with 0 volts being the null output level. Positive voltages indicate a desired velocity or current in one direction. Negative voltages indicate velocity or current in the opposite direction. The maximum drive current of this signal is +/-10 milliamps.

Coarse Home Input
connection point: Axis 1: J3 - pin 7
Axis 2: J3 – pin 19
signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range
notes: Axis 1 Supply/Return: A1Limret (J3 pin 10)
Axis 2 Supply/Return: A2Limret (J3 pin 18)
explanation: This module input is used to determine the proper zero position of the servo. In servo systems that use rotary encoders with index outputs, an index pulse is generated once per rotation of the encoder. While this signal occurs at a very repeatable angular position on the encoder, it may occur many times within the motion range of the servo. In these cases, a Coarse Home switch connected to this module input can be used to qualify which index pulse is the true zero position of the servo. By setting this switch to be activated near the end of travel of the servo, and using DCX motion commands to position the servo within this region prior to searching for the index pulse, a unique zero position for the servo can be determined. The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Amplifier Enable Output
connection point: Axis 1: J3 - pin 12
Axis 2: J3 – pin 15
signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: external pull-up required
explanation: - This module output signal should be connected to the enable input of the servo amplifier. When the DCX is turned on or reset, this signal will immediately go to its' inactive high level. When the ***MCEnableAxis( )*** is called, this signal will go to its' active low level. Anytime there is an error on the respective servo axis, including ***exceeding the following error, a limit switch input activated or the Amplifier Fault input activated***, the Amplifier Enable signal will be deactivated. This signal can also be deactivated by the **M**otor o**F**f command.

This signal is driven by a high current open collector driver and is suitable for direct connection to optically isolated inputs commonly found on a servo amplifier. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless pull-up resistors are connected to them.

Limit Positive and Limit Negative Inputs
connection point: Axis 1 Limit Positive: J3 - pin 9, Axis 2 Limit Positive: J3 - pin 17
Axis 1 Limit Negative: J3 - pin 11, Axis 2 Limit Positive: J3 - pin 16
signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range
notes: Axis 1 Limits +/- Supply/Return: A1Limret (J3 pin 10)
Axis 2 Limits +/- Supply/Return: A2Limret (J3 pin 18)
explanation: The limit switch inputs are used to cause the DCX to stop a servo's motion when it reaches the end of travel. If the servo is in position mode, the axis will only be stopped if it is moving in the direction of an activated limit switch. In all other modes, the servo will be stopped regardless of the direction it is moving if either limit switch is activated. There are three modes of stopping (decelerate to a stop, stop immediately, turn off the axis) that can be configured by the ***MCSetLimits( )***. The limit switch inputs can be enabled and disabled by ***MCSetLimits( )***. See the description of **Motion Limits** in the **Motion Control** chapter.

The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Encoder Inputs (Phase A+, Phase A-, Phase B+, Phase B-, Index+, Index-)
connection point: see pin-out tables
signal type: TTL or Differential driver output (-7V to +7V)
notes:
explanation: These input signals should be connected to an incremental quadrature encoder for supplying position feedback information for the servo controller. The plus (+) and minus (-) signs refer to the two sides of differential inputs. The default shipping configuration is for using a differential encoder. For a single ended encoder add 0 ohm resistors (Axis 1- R1, R2, R3; Axis 2-R4, R5, R6).

Encoder Power Output

connection point: Axis 1: J3 - pin 8
Axis 2: J3 – pin 20

signal type: +5 VDC PC power supply output or +12 VDC PC power supply output

notes: Axis 1: jumper JP1 selects +5VDC or +12VDC (**max. load 250 mA**)
Axis 2: jumper JP2 selects +5VDC or +12VDC (**max. load 250 mA**)

explanation: This module pin provides a convenient supply voltage connection for the encoders. The jumper can be used to connect either the +5 or +12 volt supply to the Encoder Power pin.

DCX-MC302-H High Density connector signal map

Module #1	Module #2	Module #3	Module #4	Module #5	Module #6	Module #7	Module #8	J3 Pin #	Description
J1 – 1	J2 – 1	J3 - 19	J4 - 19	J3 – 1	J4 – 1	J1 – 19	J2 - 19	1	Axis 1 Encoder Phase A+: input *
J1 – 35	J2 - 35	J3 – 53	J4 – 53	J3 – 35	J4 – 35	J1 – 53	J2 – 53	2	Axis 1 Encoder Phase A-: input
J1 – 2	J2 – 2	J3 – 20	J4 – 20	J3 – 2	J4 – 2	J1 – 20	J2 – 20	3	Axis 1 Encoder Phase B+: input*
J1 – 36	J2 - 36	J3 – 54	J4 – 54	J3 – 36	J4 – 36	J1 – 54	J2 – 54	4	Axis 1 Encoder Phase B-: input
J1 – 3	J2 – 3	J3 - 21	J4 - 21	J3 – 3	J4 – 3	J1 – 21	J2 - 21	5	Axis 1 Encoder Index +:input
J1 – 37	J2 - 37	J3 – 55	J4 – 55	J3 – 37	J4 – 37	J1 – 55	J2 – 55	6	Axis 1 Encoder Index-: input
J1 – 4	J2 – 4	J3 – 22	J4 – 22	J3 – 4	J4 – 4	J1 – 22	J2 – 22	7	Axis 1 Coarse Home: input (optically isolated)
J1 – 38	J2 - 38	J3 – 56	J4 – 56	J3 – 38	J4 – 38	J1 – 56	J2 – 56	8	Axis 1 Encoder Power (+5/+12) (**250 mA max.**)
J1 – 5	J2 – 5	J3 – 23	J4 – 23	J3 – 5	J4 – 5	J1 – 23	J2 – 23	9	Axis 1 Limit Positive: input (optically isolated)
J1 – 39	J2 - 39	J3 – 57	J4 – 57	J3 – 39	J4 – 39	J1 – 57	J2 – 57	10	Axis 1 Coarse Home & Limits supply/return
J1 – 6	J2 – 6	J3 – 24	J4 – 24	J3 – 6	J4 – 6	J1 – 24	J2 – 24		Ground
J1 – 40	J2 - 40	J3 – 58	J4 – 58	J3 – 40	J4 – 40	J1 – 58	J2 – 58	11	Axis 1 Limit Negative: input (optically isolated)
J1 – 7	J2 – 7	J3 – 25	J4 – 25	J3 – 7	J4 – 7	J1 – 25	J2 – 25	12	Axis 1 Amplifier Enable: output (open collector)
J1 – 41	J2 - 41	J3 – 59	J4 – 59	J3 – 41	J4 – 41	J1 – 59	J2 – 59		Ground
J1 – 8	J2 – 8	J3 – 26	J4 – 26	J3 – 8	J4 – 8	J1 – 26	J2 – 26		Ground
J1 – 42	J2 - 42	J3 – 60	J4 – 60	J3 – 42	J4 – 42	J1 – 60	J2 – 60	13	Axis 1 Analog Command output (+/-10 V)
J1 – 9	J2 – 9	J3 – 27	J4 – 27	J3 – 9	J4 – 9	J1 – 27	J2 – 27	14	Axis 2 Analog Command output (+/-10 V)
J1 – 43	J2 - 43	J3 – 61	J4 – 61	J3 – 43	J4 – 43	J1 – 61	J2 – 61		Ground
J1 – 10	J2 - 10	J3 – 28	J4 – 28	J3 – 10	J4 – 10	J1 – 28	J2 – 28		Ground
J1 – 44	J2 - 44	J3 – 62	J4 – 62	J3 – 44	J4 – 44	J1 – 62	J2 – 62	15	Axis 2 Amplifier Enable: output (open collector)
J1 – 11	J2 - 11	J3 - 29	J4 - 29	J3 – 11	J4 – 11	J1 – 29	J2 - 29	16	Axis 2 Limit Negative: input (optically isolated)
J1 – 45	J2 - 45	J3 – 63	J4 – 63	J3 – 45	J4 – 45	J1 – 63	J2 – 63		Ground
J1 – 12	J2 - 12	J3 – 30	J4 – 30	J3 – 12	J4 – 12	J1 – 30	J2 – 30	17	Axis 2 Limit Positive: input (optically isolated)
J1 – 46	J2 - 46	J3 – 64	J4 – 64	J3 – 46	J4 – 46	J1 – 64	J2 – 64	18	Axis 2 Coarse Home & Limits supply/return
J1 – 13	J2 - 13	J3 – 31	J4 – 31	J3 – 13	J4 – 13	J1 – 31	J2 – 31	19	Axis 2 Coarse Home: input (optically isolated)
J1 – 47	J2 - 47	J3 – 65	J4 – 65	J3 – 47	J4 – 47	J1 – 65	J2 – 65	20	Axis 2 Encoder Power (+5/+12) (**250 mA max.**)
J1 – 14	J2 - 14	J3 – 32	J4 – 32	J3 – 14	J4 – 14	J1 – 32	J2 – 32	21	Axis 2 Encoder Phase A+: input *
J1 – 48	J2 - 48	J3 – 66	J4 – 66	J3 – 48	J4 – 48	J1 – 66	J2 – 66	22	Axis 2 Encoder Phase A-: input
J1 – 15	J2 - 15	J3 – 33	J4 – 33	J3 – 15	J4 – 15	J1 – 33	J2 – 33	23	Axis 2 Encoder Phase B+: input*
J1 – 49	J2 - 49	J3 – 67	J4 – 67	J3 – 49	J4 – 49	J1 – 67	J2 – 67	24	Axis 2 Encoder Phase B-: input
J1 – 16	J2 - 16	J3 – 34	J4 – 34	J3 – 16	J4 – 16	J1 – 34	J2 – 34	25	Axis 2 Encoder Index +:input
J1 – 50	J2 - 50	J3 – 68	J4 – 68	J3 – 50	J4 – 50	J1 – 68	J2 – 68	26	Axis 2 Encoder Index-: input
J1 – 17	J2 – 17			J3 – 17	J4 – 17				Ground
J1 – 51	J2 - 51			J3 – 51	J4 – 51				Ground
J1 – 18	J2 – 18			J3 – 18	J4 – 18				Ground
J1 – 52	J3 – 52			J3 – 52	J4 – 52				Ground

For a more complete signal description please refer to the previous five pages
Mating cable connector components: Amp: 787801-1 (offset connector), 788362-1 (backshell, offset)

DCX-MC302 Module Configuration Jumpers - configuration in **bold type** denotes default factory shipping configuration

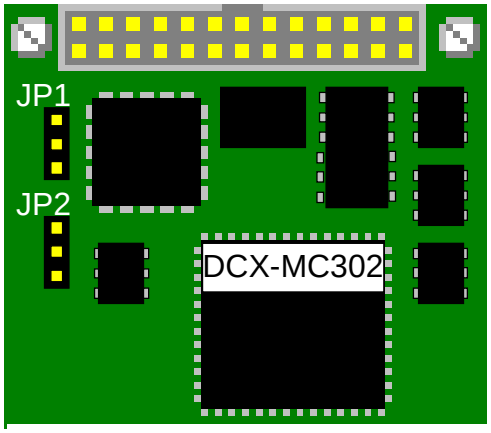
JP1 – Axis 1 Encoder Power Select (+5VDC or +12 VDC)

Pins	*Description*
1 to 2	**+5 VDC encoder supply on J3 pin 8** (**250 mA max.**)
2 to 3	+12 VDC encoder supply on J3 pin 8 (**250 mA max.**)

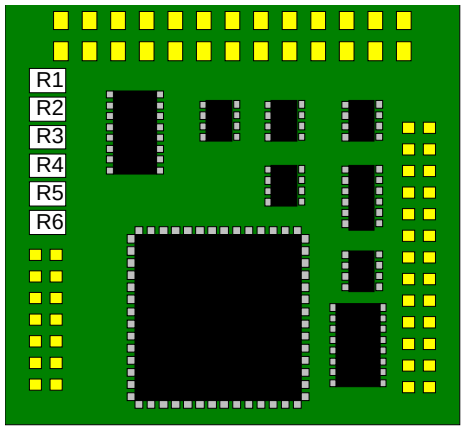
JP2 – Axis 2 Encoder Power Select (+5VDC or +12 VDC)

Pins	*Description*
1 to 2	**+5 VDC encoder supply on J3 pin 20** (**250 mA max.**)
2 to 3	+12 VDC encoder supply on J3 pin 20 (**250 mA max.**)

DCX-MC302 Module Layout

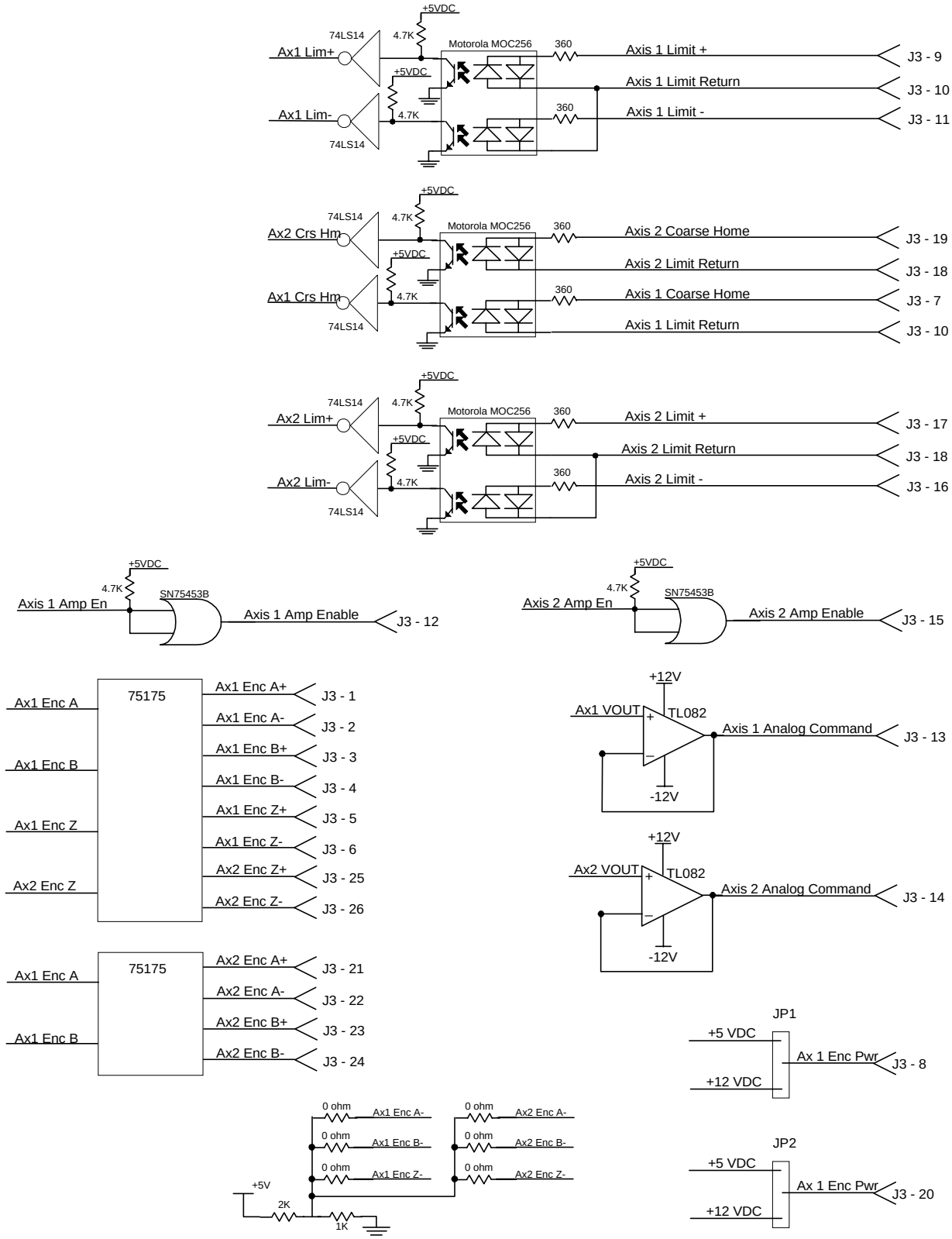


DCX-MC302 top side

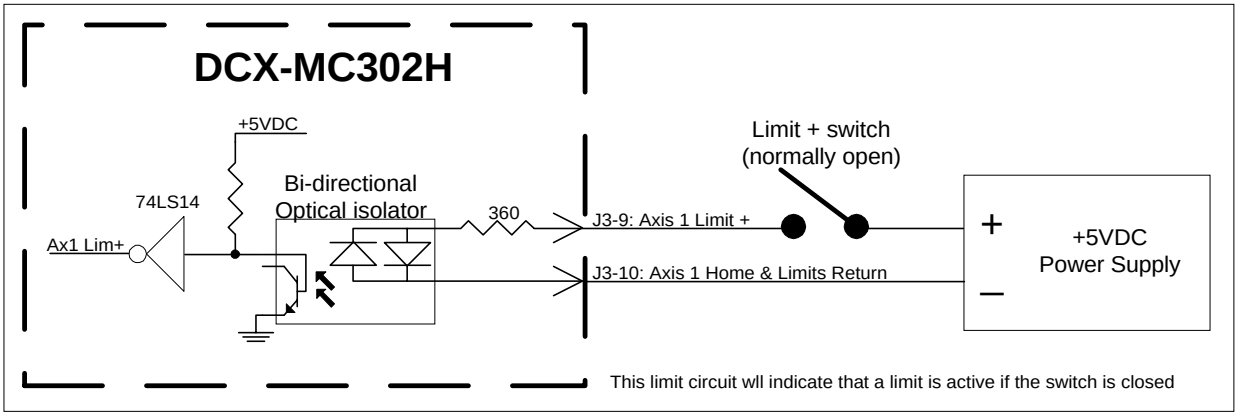


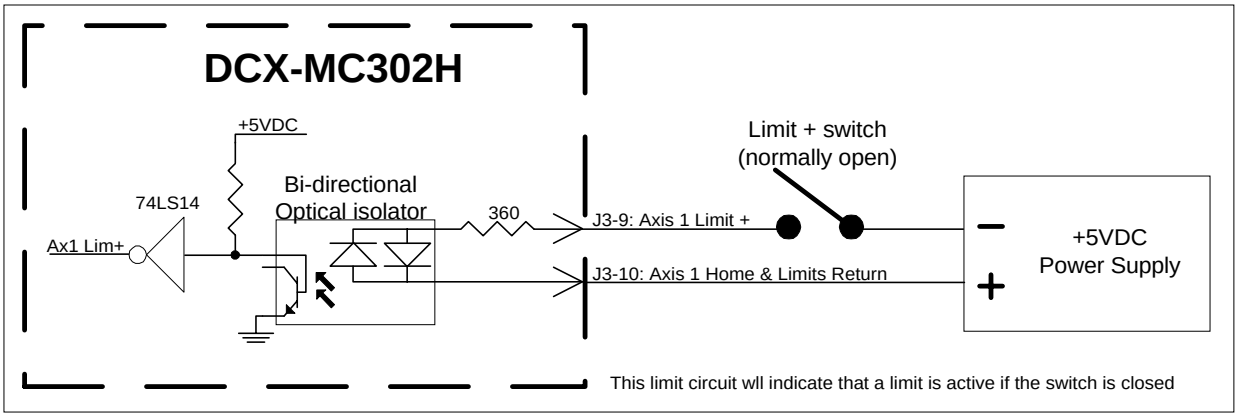
DCX-MC302 bottom side: remove R1 – R6 for differential encoder

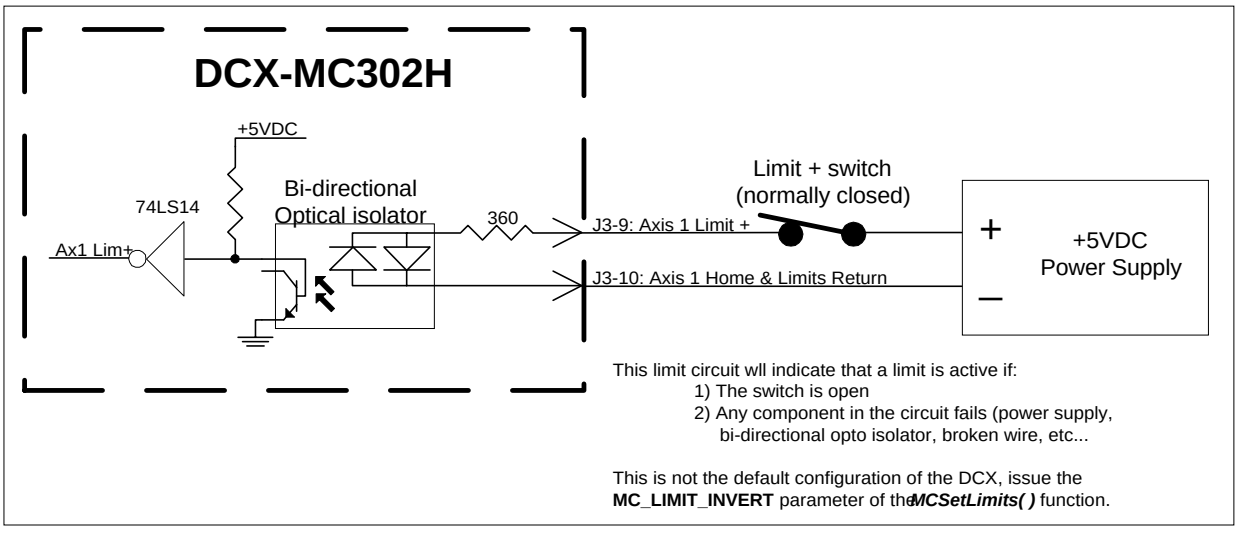
DCX-MC302H Axis I/O Interface Schematic



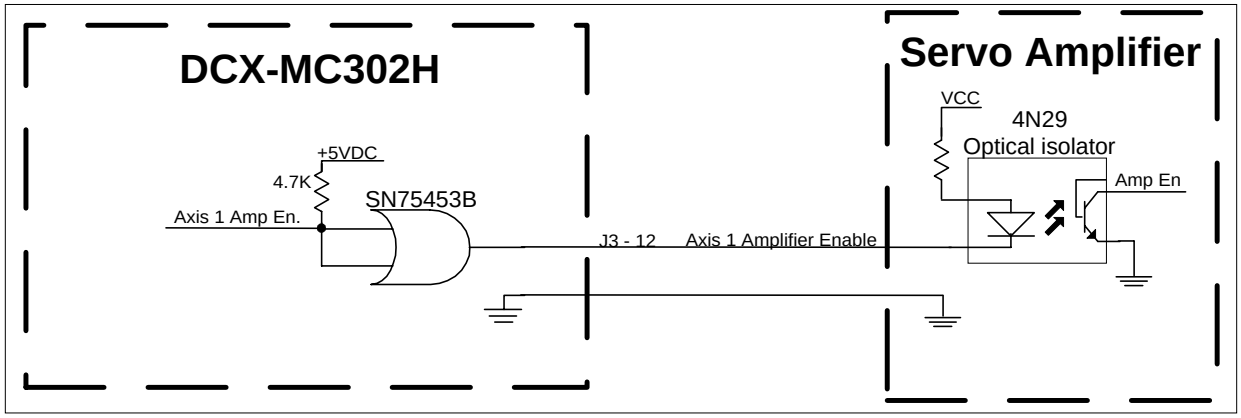
DCX-MC302 Optically Isolated Inputs Wiring Examples







DCX-MC302H Open Collector Driver Wiring Examples



DCX-MC320 Brushless Servo Commutation Control Module

The description of how to set up and operate the MC320 Commutation module was not available when this document was printed.

Please refer to Application Note **AN1004** - **Brushless AC Motor Commutation**. This PDF document is available on PMC's **MotionCD** (Other Docs and Tools/AppNOTES/Explore AppNOTES/AN1004.PDF) or from PMC's web site (www.pmccorp.com)

SIGNAL DESCRIPTIONS:

Analog Command Return
connection point: MC320-H J3 - pin 1 & 3, MC320-R J3 - pin 1
signal type: ground
notes:
explanation: Provides the signal ground for the modules Analog Command Signal output. This return path is common to the ground plane of the DCX motherboard, but is connected in such a way as to reduce digital noise. Typical servo amplifiers will have a connection for the analog command (or Ref-) return where this signal should be connected.

Phase U Torque Command Output
connection point: MC320-H J3 - pin 2, MC320-R J3 - pin 2
signal type: +/- 10V analog, 16 bit
notes: connects to servo amplifier motor command input (Ref+)
explanation: This module output signal is used to control the torque of the U winding of a brushless servo. The maximum drive current of this signal is +/-10 milliamps.

Phase V Torque Command Output
connection point: MC320-H J3 - pin 4, MC320-R J3 - pin 3
signal type: +/- 10V analog, 16 bit
notes: connects to servo amplifier motor command input (Ref+)
explanation: This module output signal is used to control the torque of the V winding of a brushless servo. The maximum drive current of this signal is +/-10 milliamps.

Phase W Torque Command Output
connection point: MC320-H not supported, MC320-R J3 - pin 1J3 - pin 4
signal type: +/- 10V analog, 16 bit
notes: connects to servo amplifier motor command input (Ref+)
explanation: This module output signal is used to control the torque of the W winding of a brushless servo. The maximum drive current of this signal is +/-10 milliamps.

Coarse Home Input
connection point: MC320-H J3 - pin 9, MC320-R J3 - pin 9
signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range
notes: MC320-H Supply/Return J3 pin 10
MC320-R Supply/Return J3 pin 18
explanation: This module input is used to determine the proper zero position of the servo. In servo systems that use rotary encoders with index outputs, an index pulse is generated once per rotation of the encoder. While this signal occurs at a very repeatable angular position on the encoder, it may occur many times within the motion range of the servo. In these cases, a Coarse Home switch connected to this module input can be used to qualify which index pulse is the true zero position of the servo. By setting this switch to be activated near the end of travel of the servo, and using DCX motion commands to position the servo within this region prior to searching for the index pulse, a unique zero position for the servo can be determined. The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Amplifier Fault Input
connection point: MC320-H J3 - pin 7, MC320-R J3 – pin 10
signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range
notes: MC320-H Supply/Return J3 pin 8
MC320-R Supply/Return J3 pin 13
explanation: - This module input is designed to be connected to the servo amplifiers Fault or Error output signal. The state of this signal will appear as a status bit in the servo's status word. The **EnableAmpFault** member of the **MCMotion** structure will enable the module to shut off the axis if the Amplifier Fault input is active. No further motion will occur until the fail signal is deactivated and the axis is enabled. The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Amplifier Enable Output
connection point: MC320-H J3 - pin 5, MC320-R J3 – pin 11
signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: external pull-up required
explanation: - This module output signal should be connected to the enable input of the servo amplifier. When the DCX is turned on or reset, this signal will immediately go to its' inactive high level. When the ***MCEnableAxis( )*** is called, this signal will go to its' active low level. Anytime there is an error on the respective servo axis, including ***exceeding the following error, a limit switch input activated or the Amplifier Fault input activated***, the Amplifier Enable signal will be deactivated. This signal can also be deactivated by the Motor oFf command.

This signal is driven by a high current open collector driver and is suitable for direct connection to optically isolated inputs commonly found on a servo amplifier. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless pull-up resistors are connected to them.

Limit Positive and Limit Negative Inputs
connection point: Limit Positive: MC320-H J3 - pin 17, MC320-R J3 - pin 14
Limit Negative: MC320-H J3 - pin 19, MC320-R J3 - pin 15
signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range
notes: MC320-H Limit Positive Supply/Return J3 pin 18
MC320-H Limit Negative Supply/Return J3 pin 20
MC320-R Limits Supply/Return J3 pin 18
explanation: The limit switch inputs are used to cause the DCX to stop a servo's motion when it reaches the end of travel. If the servo is in position mode, the axis will only be stopped if it is moving in the direction of an activated limit switch. In all other modes, the servo will be stopped regardless of the direction it is moving if either limit switch is activated. There are three modes of stopping (decelerate to a stop, stop immediately, turn off the axis) that can be configured by the ***MCSetLimits( )***. The limit switch inputs can be enabled and disabled by ***MCSetLimits( )***. See the description of **Motion Limits** in the **Motion Control** chapter.

The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Primary Encoder Inputs (Phase A+, Phase -, Phase B+, Phase B-, Index+, Index-)
connection point: see pin-out table
signal type: TTL or Differential driver output (-7V to +7V)
notes: The encoder power jumper JP3 sets the 'mid point' for the differential receiver
explanation: These input signals should be connected to an incremental quadrature encoder for supplying position feedback information for the servo controller. The plus (+) and minus (-) signs refer to the two sides of differential inputs. By setting jumpers JP1 and JP2 appropriately, the plus signal inputs can be configured for single ended inputs.

Hall Effect Sensor A, B, and C Inputs
connection point: see pin-out table
signal type: TTL or Differential driver output (-7V to +7V)
notes:
explanation: - These input signals can be used for interfacing to Hall effect sensors.

Encoder Power Output
connection point: MC300-H J3 - pin 16, MC300-R J3 - pin 17
signal type: +5 VDC PC power supply output or +12 VDC PC power supply output
notes: The encoder power jumper JP3 selects +5VDC or +12VDC (**250 mA max.**)
explanation: This module pin provides a convenient supply voltage connection for the encoders. The jumper JP3 located on the module can be used to connect either the +5 or +12 volt supply to the Encoder Power pin. The setting of this jumper also selects the threshold voltage for the module's single ended phase and index encoder inputs. When JP1 is set for +5 volts, the threshold will be 2.5 volts, for +12 volts, the threshold will be +6 volts. The threshold voltage determines at what voltage the input changes between on and off.

SUPPLY CONNECTIONS (+5, GROUND) - These module pins provide access to the DCX supply voltages.

DCX-MC320-H High Density connector signal map

Module #1	Module #2	Module #3	Module #4	Module #5	Module #6	Module #7	Module #8	J3 Pin #	Description
J1 – 1	J2 – 1	J3 - 19	J4 - 19	J3 – 1	J4 – 1	J1 – 19	J2 - 19	1	Ground
J1 – 35	J2 - 35	J3 – 53	J4 – 53	J3 – 35	J4 – 35	J1 – 53	J2 – 53	2	Phase U Torque Command: output
J1 – 2	J2 – 2	J3 – 20	J4 – 20	J3 – 2	J4 – 2	J1 – 20	J2 – 20	3	Ground
J1 – 36	J2 - 36	J3 – 54	J4 – 54	J3 – 36	J4 – 36	J1 – 54	J2 – 54	4	Phase V Torque Command: output
J1 – 3	J2 – 3	J3 - 21	J4 - 21	J3 – 3	J4 – 3	J1 – 21	J2 - 21	5	Amplifier Enable: output
J1 – 37	J2 - 37	J3 – 55	J4 – 55	J3 – 37	J4 – 37	J1 – 55	J2 – 55	6	Amplifier Enable return
J1 – 4	J2 – 4	J3 – 22	J4 – 22	J3 – 4	J4 – 4	J1 – 22	J2 – 22	7	Amplifier Fault: input
J1 – 38	J2 - 38	J3 – 56	J4 – 56	J3 – 38	J4 – 38	J1 – 56	J2 – 56	8	Amplifier Fault return
J1 – 5	J2 – 5	J3 – 23	J4 – 23	J3 – 5	J4 – 5	J1 – 23	J2 – 23	9	Coarse Home: input
J1 – 39	J2 - 39	J3 – 57	J4 – 57	J3 – 39	J4 – 39	J1 – 57	J2 – 57	10	Coarse Home return
J1 – 6	J2 – 6	J3 – 24	J4 – 24	J3 – 6	J4 – 6	J1 – 24	J2 – 24		Ground
J1 – 40	J2 - 40	J3 – 58	J4 – 58	J3 – 40	J4 – 40	J1 – 58	J2 – 58	11	Reserved
J1 – 7	J2 – 7	J3 – 25	J4 – 25	J3 – 7	J4 – 7	J1 – 25	J2 – 25	12	Reserved
J1 – 41	J2 - 41	J3 – 59	J4 – 59	J3 – 41	J4 – 41	J1 – 59	J2 – 59		Ground
J1 – 8	J2 – 8	J3 – 26	J4 – 26	J3 – 8	J4 – 8	J1 – 26	J2 – 26		Ground
J1 – 42	J2 - 42	J3 – 60	J4 – 60	J3 – 42	J4 – 42	J1 – 60	J2 – 60	13	Hall sensor A+ / Aux. Encoder Phase A+
J1 – 9	J2 – 9	J3 – 27	J4 – 27	J3 – 9	J4 – 9	J1 – 27	J2 – 27	14	Hall sensor B+ / Aux. Encoder Phase B+
J1 – 43	J2 - 43	J3 – 61	J4 – 61	J3 – 43	J4 – 43	J1 – 61	J2 – 61		Ground
J1 – 10	J2 - 10	J3 – 28	J4 – 28	J3 – 10	J4 – 10	J1 – 28	J2 – 28		Ground
J1 – 44	J2 - 44	J3 – 62	J4 – 62	J3 – 44	J4 – 44	J1 – 62	J2 – 62	15	Hall Sensor C+
J1 – 11	J2 - 11	J3 - 29	J4 - 29	J3 – 11	J4 – 11	J1 – 29	J2 - 29	16	Encoder Power: output (**max. load 250 mA**)
J1 – 45	J2 - 45	J3 – 63	J4 – 63	J3 – 45	J4 – 45	J1 – 63	J2 – 63		Ground
J1 – 12	J2 - 12	J3 – 30	J4 – 30	J3 – 12	J4 – 12	J1 – 30	J2 – 30	17	Limit Positive: input
J1 – 46	J2 - 46	J3 – 64	J4 – 64	J3 – 46	J4 – 46	J1 – 64	J2 – 64	18	Limit Positive return
J1 – 13	J2 - 13	J3 – 31	J4 – 31	J3 – 13	J4 – 13	J1 – 31	J2 – 31	19	Limit Negative: input
J1 – 47	J2 - 47	J3 – 65	J4 – 65	J3 – 47	J4 – 47	J1 – 65	J2 – 65	20	Limit Negative return
J1 – 14	J2 - 14	J3 – 32	J4 – 32	J3 – 14	J4 – 14	J1 – 32	J2 – 32	21	Primary Encoder Phase A+: input *
J1 – 48	J2 - 48	J3 – 66	J4 – 66	J3 – 48	J4 – 48	J1 – 66	J2 – 66	22	Primary Encoder Phase A-: input
J1 – 15	J2 - 15	J3 – 33	J4 – 33	J3 – 15	J4 – 15	J1 – 33	J2 – 33	23	Primary Encoder Phase B+: input*
J1 – 49	J2 - 49	J3 – 67	J4 – 67	J3 – 49	J4 – 49	J1 – 67	J2 – 67	24	Primary Encoder Phase B-: input
J1 – 16	J2 - 16	J3 – 34	J4 – 34	J3 – 16	J4 – 16	J1 – 34	J2 – 34	25	Primary Encoder Index +:input
J1 – 50	J2 - 50	J3 – 68	J4 – 68	J3 – 50	J4 – 50	J1 – 68	J2 – 68	26	Primary Encoder Index-: input
J1 – 17	J2 – 17			J3 – 17	J4 – 17				Ground
J1 – 51	J2 - 51			J3 – 51	J4 – 51				Ground
J1 – 18	J2 – 18			J3 – 18	J4 – 18				Ground
J1 – 52	J3 – 52			J3 – 52	J4 – 52				Ground

For a more complete signal description please refer to the previous five pages
Mating cable connector components: Amp: 787801-1 (offset connector), 788362-1 (backshell, offset)

DCX-MC320-R Module connector

J3 connector pin-out (Motor command, encoders, and axis I/O)

Pin #	Description
1	Torque Command Return (Ground)
2	Phase U Torque Command: output (10ma max.)
3	Phase V Torque Command: output (10ma max.)
4	Phase W Torque Command: output (10ma max.)
5	Ground
6	+5 VDC (**250 mA max.**)
7	Reserved
8	Primary Encoder Index +:input (active high)
9	Coarse Home: input (optically isolated, 12V – 24V, 15ma min.)
10	Amplifier Fault: input (optically isolated, 12V – 24V, 15ma min.)
11	Amplifier Enable: output (open collector, 100ma max., 30V max.)
12	Amp Enable & Direction return
13	Amp Fault opto isolator supply/return
14	Limit Positive: input (optically isolated, 12V – 24V, 15ma min.)
15	Limit Negative: input (optically isolated, 12V – 24V, 15ma min.)
16	Primary Encoder Phase A+: input *
17	Encoder Power: output (+5VDC or +12VDC, see jumper JP3) (**250 mA max.**)
18	Coarse Home & Limits opto isolator supply/return
19	Primary Encoder Phase A-: input
20	Primary Encoder Phase B-: input
21	Hall sensor A+
22	Hall sensor B+
23	Primary Encoder Phase B+: input*
24	Hall Sensor C+
25	Primary Encoder Index-: input (active low)
26	Ground

* Use A+ and B+ for single-ended Encoder inputs

Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N 26IDS2-C-SPT-SR or equivalent

DCX-MC320 Module Configuration Jumpers - configuration in **bold type** denotes default factory shipping configuration

JP1 – Encoder type (single ended or differential)

Pins	*Description*
1 to 2 to 3	Single ended encoder, A, B, Z (three pin jumper provided)
open	**Differential encoder, A+, A-, B+, B-**

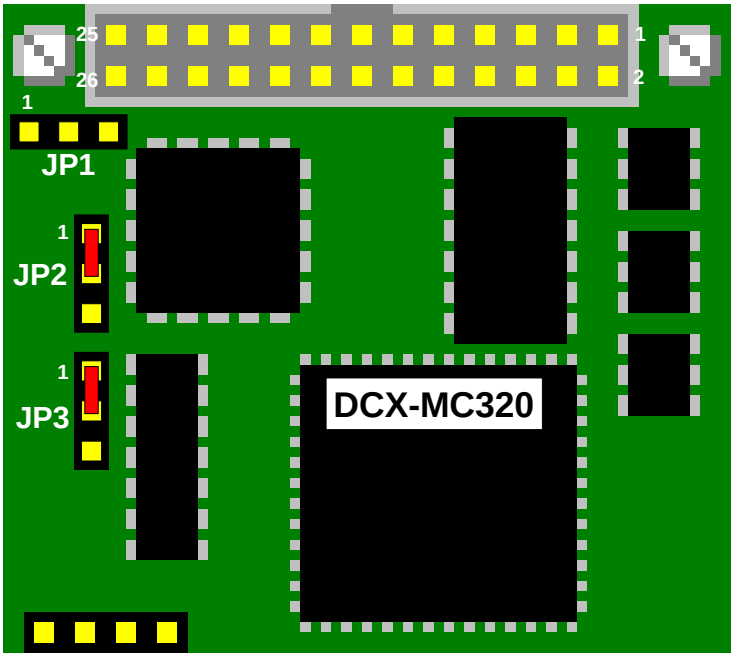
JP2 – Encoder Index Active Level Select)

Pins	*Description*
1 to 2	Single ended Index, Z+ (Active high)
2 to 3	**Single ended Index, Z- (active low)**
open	Differential Index, Z+ and Z-

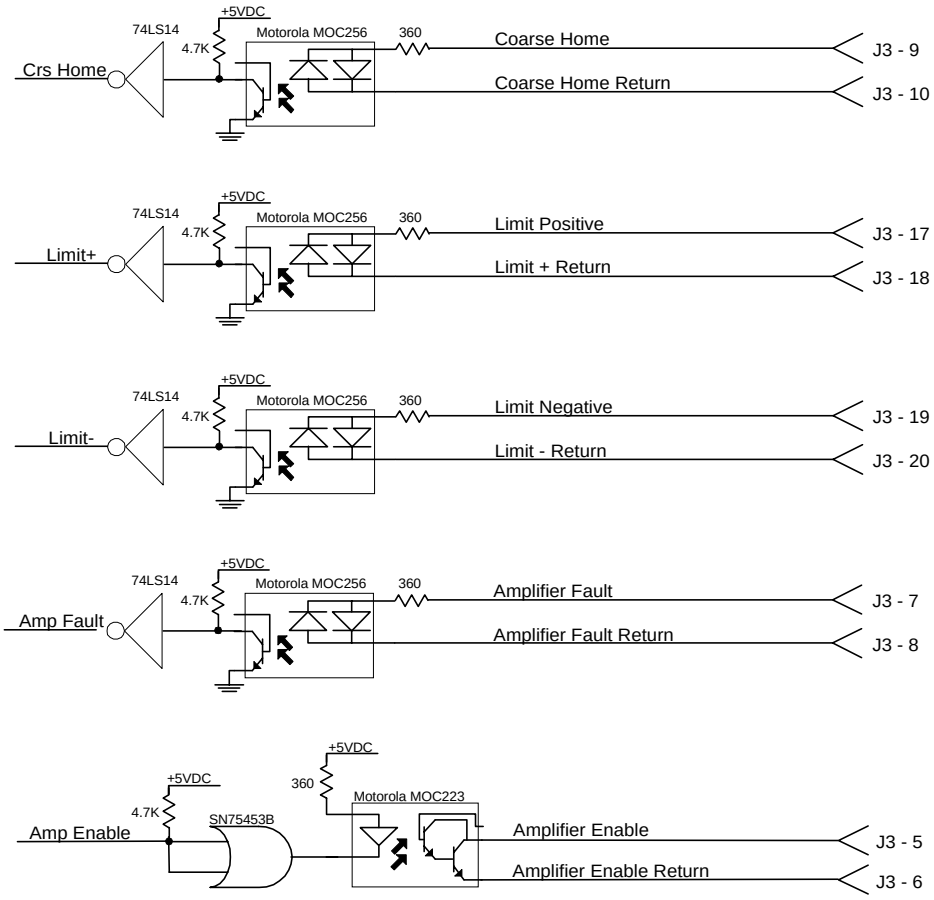
JP3 – Encoder Power Select (+5VDC or +12 VDC)

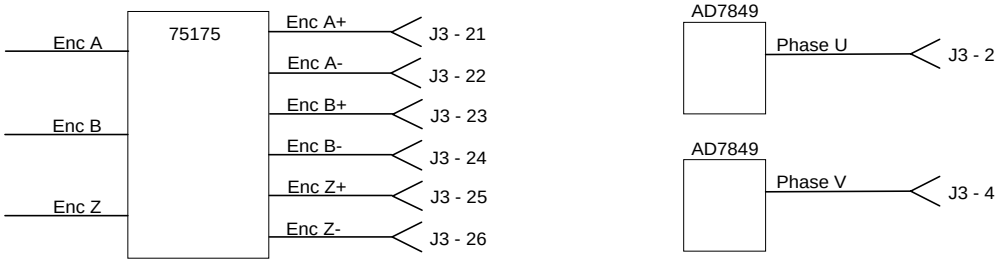
Pins	*Description*
1 to 2	**+5 VDC encoder supply on J3 pin 16/17** (**250 mA max.**)
2 to 3	+12 VDC encoder supply on J3 pin 16/17 (**250 mA max.**)

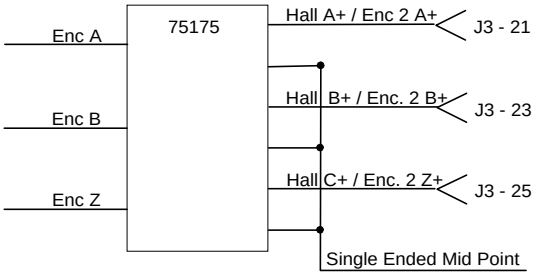
DCX-MC320 Module Layout



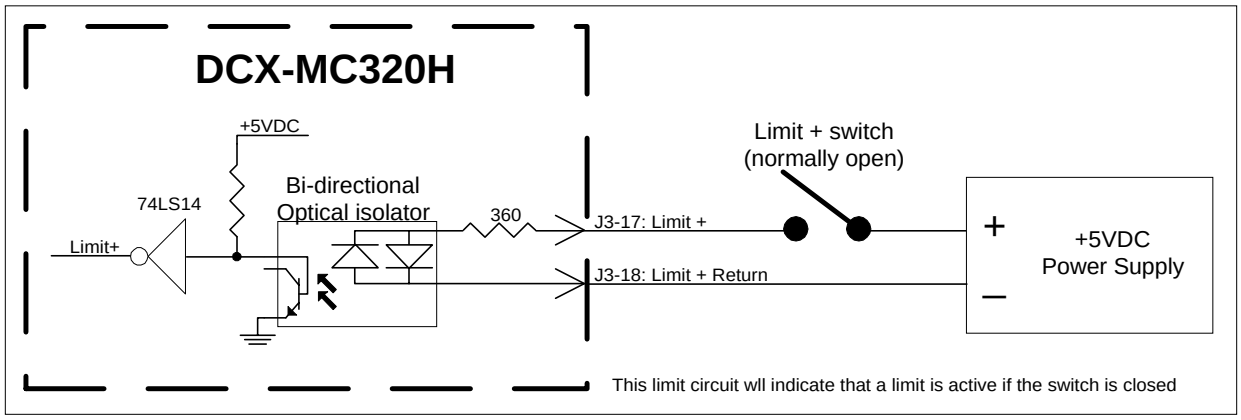
DCX-MC320H Axis I/O Interface Schematic

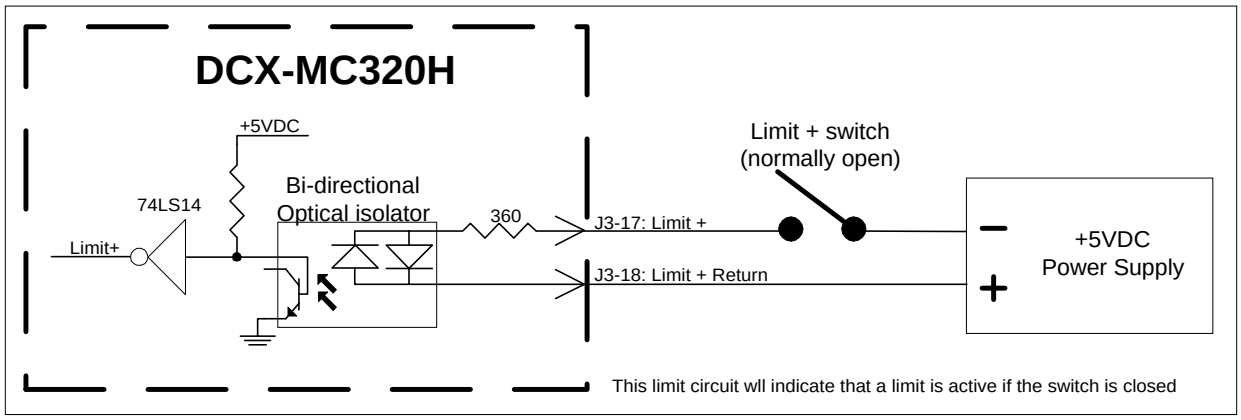


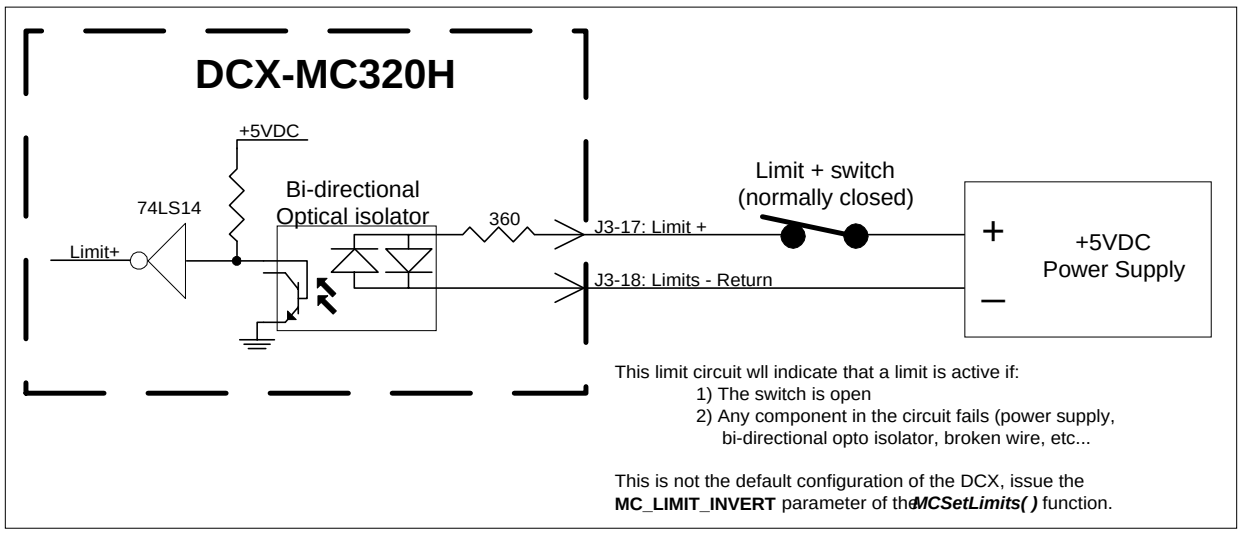




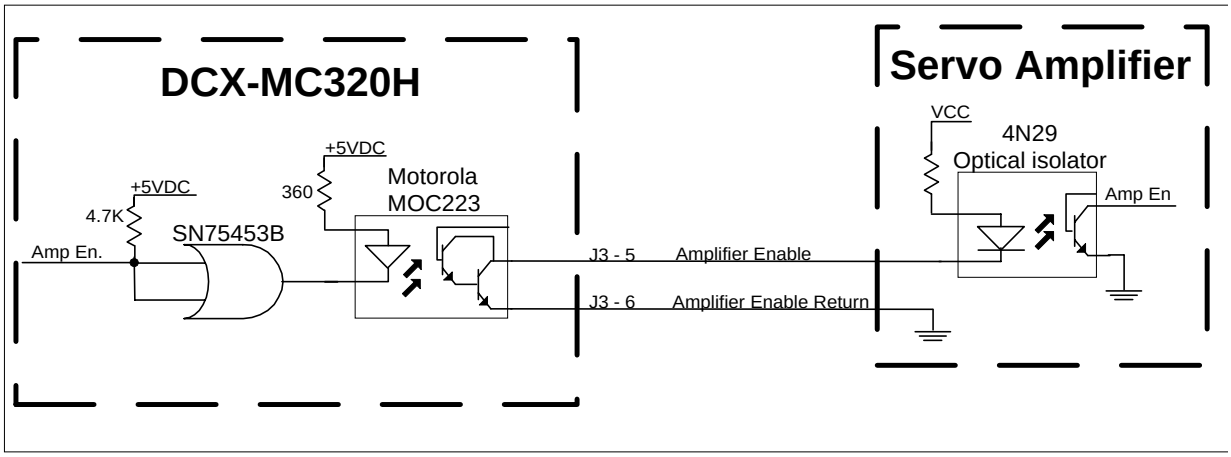
DCX-MC320H Optically Isolated Inputs Wiring Examples







DCX-MC320H Open Collector Driver Wiring Examples



DCX-MC360 Stepper Motor Control Module

SIGNAL DESCRIPTIONS:

Pulse and Direction Outputs
connection point: Direction / CW: MC360-H J3 - pin 3, MC360-R J3 - pin 3
Pulse / CCW: MC360-H J3 - pin 1, MC360-R J3 - pin 4
signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: external pull-up required
explanation: In the control of a stepper motor, the two primary control signals are Pulse and Direction (or CW Pulse and CCW Pulse). These signals are connected to the external stepper motor driver that supplies current to the motor windings. In order for the stepper module to move the motor one step, a pulse is generated on one of these signals.

Both of these signals are driven by high current open collector drivers and are suitable for direct connection to optically isolated inputs commonly found on stepper motor drivers. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless pull-up resistors are connected to them.

Pulse: The motor driver should advance the motor by one increment for each pulse. The motor may advance a full step, a half step, or a micro step. This is determined by the mode of the stepper motor driver. The Pulse signal is normally high, and is pulled low at the beginning of a step. It stays low for one half the step period (50% duty cycle), and then goes back high. When it is time for the next step, the signal will be pulled low again.

Direction: This signal indicates the direction the motor will move. When the stepper is incrementing the current position (moving positive) this signal will remain high (pulled up). When the stepper is decrementing the current position (moving negative) this signal will be pulled low.

The function ***MCSetModuleOutputMode( )*** is used to change the operation of these signals to CW and CCW Pulse. In this mode, pulses will be generated on the CW Pulse output when the current position is increasing, and on the CCW Pulse output when the current position is decreasing.

Drive Enable Output
connection point: MC360-H J3 - pin 5, MC360-R J3 - pin 16
signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: external pull-up required
explanation: This output will be pulled low when an axis is enabled (***MCEnableAxis( )***). It will remain low until: the axis is disabled or an error condition exists (limits tripped).

This signal is driven by a high current open collector driver and is suitable for direct connection to optically isolated inputs commonly found on a servo amplifier. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless pull-up resistors are connected to them.

Limit Positive and Limit Negative Inputs

connection point: Limit Positive: MC360-H J3 - pin 17, MC320-R J3 - pin 8
Limit Negative: MC360-H J3 - pin 19, MC320-R J3 - pin 9
signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range
notes: MC360-H Limit Positive Supply/Return J3 pin 18
MC360-H Limit Negative Supply/Return J3 pin 20
MC360-R Limits Supply/Return J3 pin 6

explanation: The limit switch inputs are used to cause the DCX to stop a servo's motion when it reaches the end of travel. If the servo is in position mode, the axis will only be stopped if it is moving in the direction of an activated limit switch. In all other modes, the servo will be stopped regardless of the direction it is moving if either limit switch is activated. There are three modes of stopping (decelerate to a stop, stop immediately, turn off the axis) that can be configured by the ***MCSetLimits( )***. The limit switch inputs can be enabled and disabled by ***MCSetLimits( )***. See the description of **Motion Limits** in the **Motion Control** chapter.

The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Home Input

connection point: MC360-H J3 - pin 9, MC360-R J3 - 13
signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range
notes: MC360-H Home Supply/Return J3 pin 10
MC360-R Home Supply/Return J3 pin 12

explanation: This input is used to set the zero position of a stepper motor. It is typically connected to a sensor/switch that is activated at a fixed position in the motor's range of motion. The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Compare / Full/Half Step Output

connection point: MC360-H J3 - pin 13, MC360-R J3 - 14
signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: external pull-up required
explanation:

Compare – Used to indicate when a position compare event has occurred. See the description of **Position Compare** in the **Application Solutions chapter**.

Full/Half Step –This signal is used if the stepper driver has a digital input to select between or full/micro (or full/half) step modes. The default condition of this signal is to be inactive (pulled high). Setting the **MC_STEP_FULL** parameter of the **MCMotion** structure will cause the signal to be pulled low.

This signal is driven by a high current open collector driver and is suitable for direct connection to optically isolated inputs commonly found on a servo amplifier. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless pull-up resistors are connected to them.

Full/Half Current Output
connection point: MC360-H J3 - pin 14, MC360-R J3 - 15
signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: external pull-up required
explanation: This signal is used if the stepper driver has a digital input for current control. The default condition of this signal is to be inactive (pulled high). Setting the **MC_CURRENT_FULL** parameter of the **MCMotion** structure will cause the signal to be pulled low.

This signal is driven by a high current open collector driver and is suitable for direct connection to optically isolated inputs commonly found on a servo amplifier. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless pull-up resistors are connected to them.

Drive Fault Input
connection point: MC360-H J3 - pin 7, MC360-R J3 - 7
signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range
notes: MC360-H Drive Fault Supply/Return J3 pin 8
MC360-R Drive Fault Supply/Return J3 pin 5
explanation: This module input is designed to be connected to the Fault or Error output signal of a stepper driver. The state of this signal will appear as a status bit in the servo's status word. The **EnableAmpFault** member of the **MCMotion** structure will enable the module to shut off the axis if the Drive Fault input is active. No further motion will occur until the fault signal is deactivated and the axis has been enabled. The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Null Input
connection point: MC360-H J3 - pin 12, MC360-R J3 - 17
signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range
notes: MC360-H Null VHDCI connector pin 41 (no connect on module J3 connector)
MC360-R Null Supply/Return J3 pin 5
explanation: In order to switch from micro stepping to full stepping without the motor shifting position, the motor should be micro stepped to the "Null" Position. This is the position where the output of the amplifier will not change if it is switched between full and micro stepping. If the stepper amplifier provides an output signal that indicates when the motor is at a null position, the DCX can monitor this signal on the Null Position input of the module.

The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Position Capture +/- / Auxiliary Encoder Index +/-
connection point: Position Cap. + / Aux. Enc. Index+: MC360-H J3 - pin 25, MC320-R J3 - pin 22
Position Cap. - / Aux. Enc. Index-: MC360-H J3 - pin 26, MC320-R J3 - pin 23
signal type: TTL or Differential driver output (-7V to +7V)
notes:
explanation: -
Position Capture +/- – Used to initiate the capture of position data. See the description of **Position Capture** in the **Application Solutions chapter**.

Auxiliary Encoder Index +/- - This input signal can be used to define the home position of an auxiliary encoder.

Auxiliary Encoder Coarse Home Input
connection point: MC360-H J3 - pin 15, MC360-R J3 - 11
signal type: Bi-directional optical isolator, 15ma min. 12V – 24V range
notes: MC360-H Coarse Home VHDCI connector pin 10 (no connect on module J3 connector)
MC360-R Null Supply/Return J3 pin 6
explanation: This input is used to 'home' the auxiliary encoder by qualifying the index mark. It is typically connected to a switch that is activated at a fixed position in the motors motion range. See the description of **Homing an Axis** in the **Motion Control** chapter.

The input device is a bi-directional optical isolator. The allowable voltage range for this signal is 12VDC to 24VDC. The minimum current required to turn on the optical isolator is 10ma. Bi-directional optical isolator wiring examples are provided later in this section.

Auxiliary Encoder Inputs (Phase A, Phase B, Index+, Index-)
connection point: see pin-out table
signal type: TTL or Differential driver output (-7V to +7V)
notes:
explanation: - These input signals can be used for an auxiliary encoder.

Auxiliary Encoder Power Output
connection point: MC300-H J3 - pin 16, MC300-R J3 - pin 10
signal type: +5 VDC PC power supply output or +12 VDC PC power supply output
notes: The encoder power jumper JP3 selects +5VDC or +12VDC (**250 mA max.**)
explanation: This module pin provides a convenient supply voltage connection for the auxiliary encoder. The jumper JP3 located on the module can be used to connect either the +5 or +12 volt supply to the Encoder Power pin. The setting of this jumper also selects the threshold voltage for the module's single ended phase and index encoder inputs. When JP1 is set for +5 volts, the threshold will be 2.5 volts, for +12 volts, the threshold will be +6 volts. The threshold voltage determines at what voltage the input changes between on and off.

SUPPLY CONNECTIONS (+5, +12, -12, GROUND) - These module pins provide access to the DCX supply voltages.

DCX-MC360-H High Density connector signal map

Module #1	Module #2	Module #3	Module #4	Module #5	Module #6	Module #7	Module #8	J3 Pin #	Description
J1 – 1	J2 – 1	J3 - 19	J4 - 19	J3 – 1	J4 – 1	J1 – 19	J2 - 19	1	Step or CCW Pulse: output
J1 – 35	J2 - 35	J3 – 53	J4 – 53	J3 – 35	J4 – 35	J1 – 53	J2 – 53	2	Ground
J1 – 2	J2 – 2	J3 – 20	J4 – 20	J3 – 2	J4 – 2	J1 – 20	J2 – 20	3	Direction or CW Pulse: output
J1 – 36	J2 - 36	J3 – 54	J4 – 54	J3 – 36	J4 – 36	J1 – 54	J2 – 54	4	Ground
J1 – 3	J2 – 3	J3 - 21	J4 - 21	J3 – 3	J4 – 3	J1 – 21	J2 - 21	5	Driver Enable: output
J1 – 37	J2 - 37	J3 – 55	J4 – 55	J3 – 37	J4 – 37	J1 – 55	J2 – 55	6	Ground
J1 – 4	J2 – 4	J3 – 22	J4 – 22	J3 – 4	J4 – 4	J1 – 22	J2 – 22	7	Drive Fault: input
J1 – 38	J2 - 38	J3 – 56	J4 – 56	J3 – 38	J4 – 38	J1 – 56	J2 – 56	8	Drive Fault return
J1 – 5	J2 – 5	J3 – 23	J4 – 23	J3 – 5	J4 – 5	J1 – 23	J2 – 23	9	Home: input
J1 – 39	J2 - 39	J3 – 57	J4 – 57	J3 – 39	J4 – 39	J1 – 57	J2 – 57	10	Home return
J1 – 6	J2 – 6	J3 – 24	J4 – 24	J3 – 6	J4 – 6	J1 – 24	J2 – 24		Ground
J1 – 40	J2 - 40	J3 – 58	J4 – 58	J3 – 40	J4 – 40	J1 – 58	J2 – 58	11	Reserved
J1 – 7	J2 – 7	J3 – 25	J4 – 25	J3 – 7	J4 – 7	J1 – 25	J2 – 25	12	Null Position: input
J1 – 41	J2 - 41	J3 – 59	J4 – 59	J3 – 41	J4 – 41	J1 – 59	J2 – 59		Ground
J1 – 8	J2 – 8	J3 – 26	J4 – 26	J3 – 8	J4 – 8	J1 – 26	J2 – 26		Ground
J1 – 42	J2 - 42	J3 – 60	J4 – 60	J3 – 42	J4 – 42	J1 – 60	J2 – 60	13	Compare / Full/Half Step: output
J1 – 9	J2 – 9	J3 – 27	J4 – 27	J3 – 9	J4 – 9	J1 – 27	J2 – 27	14	Full/Half Current: output
J1 – 43	J2 - 43	J3 – 61	J4 – 61	J3 – 43	J4 – 43	J1 – 61	J2 – 61		Ground
J1 – 10	J2 - 10	J3 – 28	J4 – 28	J3 – 10	J4 – 10	J1 – 28	J2 – 28		Ground
J1 – 44	J2 - 44	J3 – 62	J4 – 62	J3 – 44	J4 – 44	J1 – 62	J2 – 62	15	Aux. Encoder Coarse Home: input
J1 – 11	J2 - 11	J3 - 29	J4 - 29	J3 – 11	J4 – 11	J1 – 29	J2 - 29	16	Aux. Enc. Power: output (**max. load 250 mA**)
J1 – 45	J2 - 45	J3 – 63	J4 – 63	J3 – 45	J4 – 45	J1 – 63	J2 – 63		Ground
J1 – 12	J2 - 12	J3 – 30	J4 – 30	J3 – 12	J4 – 12	J1 – 30	J2 – 30	17	Limit Positive: input
J1 – 46	J2 - 46	J3 – 64	J4 – 64	J3 – 46	J4 – 46	J1 – 64	J2 – 64	18	Limit Positive return
J1 – 13	J2 - 13	J3 – 31	J4 – 31	J3 – 13	J4 – 13	J1 – 31	J2 – 31	19	Limit Negative: input
J1 – 47	J2 - 47	J3 – 65	J4 – 65	J3 – 47	J4 – 47	J1 – 65	J2 – 65	20	Limit Negative return
J1 – 14	J2 - 14	J3 – 32	J4 – 32	J3 – 14	J4 – 14	J1 – 32	J2 – 32	21	Auxiliary Encoder Phase A+: input
J1 – 48	J2 - 48	J3 – 66	J4 – 66	J3 – 48	J4 – 48	J1 – 66	J2 – 66	22	Auxiliary Encoder Phase A-: input
J1 – 15	J2 - 15	J3 – 33	J4 – 33	J3 – 15	J4 – 15	J1 – 33	J2 – 33	23	Auxiliary Encoder Phase B+: input
J1 – 49	J2 - 49	J3 – 67	J4 – 67	J3 – 49	J4 – 49	J1 – 67	J2 – 67	24	Auxiliary Encoder Phase B-: input
J1 – 16	J2 - 16	J3 – 34	J4 – 34	J3 – 16	J4 – 16	J1 – 34	J2 – 34	25	Position Capture + / Aux. Encoder Index+: input
J1 – 50	J2 - 50	J3 – 68	J4 – 68	J3 – 50	J4 – 50	J1 – 68	J2 – 68	26	Position Capture - / Aux. Encoder Index-: input
J1 – 17	J2 – 17			J3 – 17	J4 – 17				Ground
J1 – 51	J2 - 51			J3 – 51	J4 – 51				Ground
J1 – 18	J2 – 18			J3 – 18	J4 – 18				Ground
J1 – 52	J3 – 52			J3 – 52	J4 – 52				Ground

For a more complete signal description please refer to the previous five pages
Mating cable connector components: Amp: 787801-1 (offset connector), 788362-1 (backshell, offset)

DCX-MC360-R Module connector

J3 connector pin-out (Motor command, encoders, and axis I/O)

Pin #	Description
1	Ground
2	+5 VDC (**max. load 250 mA**)
3	Direction or CW Pulse: output (open collector, 100ma max., 30V max.) *
4	Pulse or CCW Pulse: output (open collector, 100ma max., 30V max.) *
5	FNRET: Drive Fault and Null opto isolator supply/return
6	LIMCRSRET: Coarse Home & Limits opto isolator supply/return
7	Drive Fault: input (opto isolator, 15ma min. current, 30V max.)
8	Limit Positive: input (opto isolator, 15ma min. current, 30V max.)
9	Limit Negative: input (opto isolator, 15ma min. current, 30V max.)
10	Auxiliary Encoder Power: output (+5VDC or +12VDC, see jumper JP3) (**max. load 250 mA**)
11	Aux. Encoder Coarse Home: input (opto isolator, 15ma min. current, 30V max.)
12	HOMRET: Home opto isolator supply/return
13	Home: input (opto isolator, 15ma min. current, 30V max.)
14	Compare / Full/Half Step: output (open collector, 100ma max., 30V max.)
15	Full/Half Current: output (open collector, 100ma max., 30V max.)
16	Driver Enable: output (open collector, 100ma max., 30V max.)
17	Null Position: input (opto isolator, 15ma min. current, 30V max.)
18	Auxiliary Encoder Phase A+: input
19	Auxiliary Encoder Phase A-: input
20	Auxiliary Encoder Phase B+: input
21	Auxiliary Encoder Phase B-: input
22	Position Capture + / Auxiliary Encoder Index+: input (active high)
23	Position Capture - / Auxiliary Encoder Index-: input (active low)
24	+12 VDC (**max. load 250 mA**)
25	-12 VDC (**max. load 50 mA**)
26	Ground

* These signals default to DIRECTION and PULSE, use ***MCSetModuleOutputMode( )*** to change to CW and CCW PULSE.

Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N 26IDS2-C-SPT-SR or equivalent

DCX-MC360 Module Configuration Jumpers - configuration in **bold type** denotes default factory shipping configuration

JP1 – Encoder type (single ended or differential)

Pins	*Description*
1 to 2 to 3	Single ended encoder, A, B, Z (three pin jumper provided)
open	**Differential encoder, A+, A-, B+, B-**

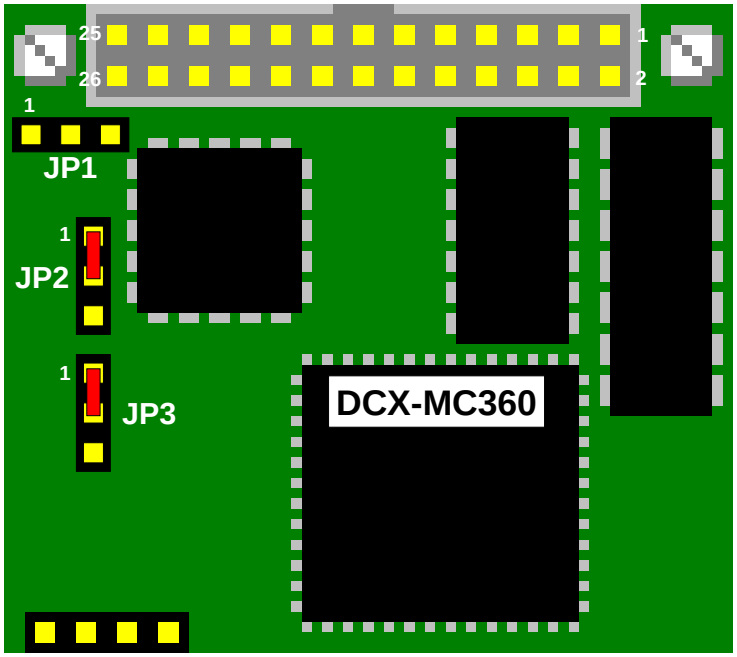
JP2 – Auxiliary Encoder Index Active Level Select

Pins	*Description*
1 to 2	**Single ended Index, Z+ (Active high)**
2 to 3	Single ended Index, Z- (active low)
open	Differential Index, Z+ and Z-

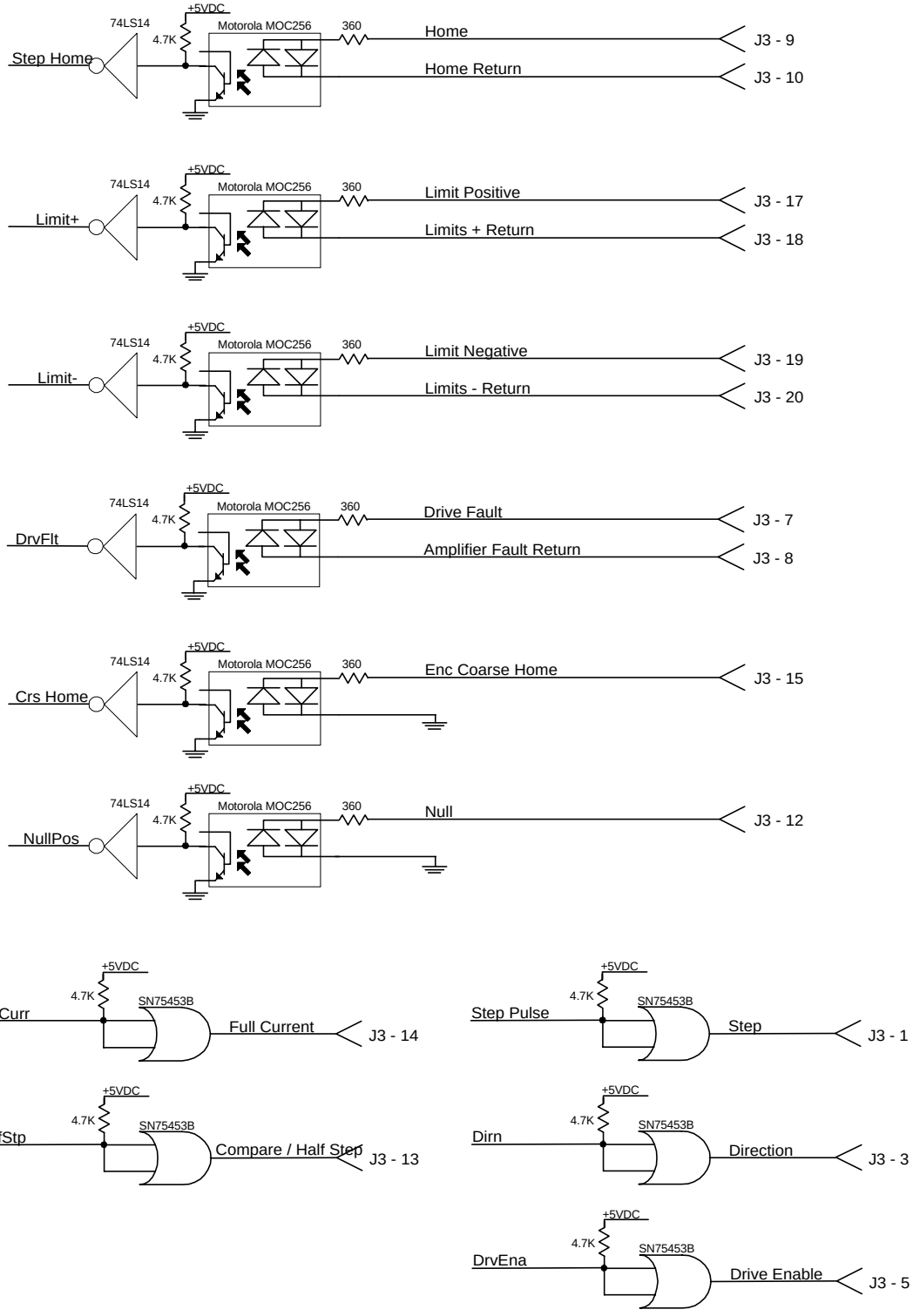
JP3 – Auxiliary Encoder Power Select (+5VDC or +12 VDC)

Pins	*Description*
1 to 2	**+5 VDC encoder supply on J3 pin 16/10** (**max. load 250 mA**)
2 to 3	+12 VDC encoder supply on J3 pin 16/10 (**max. load 250 mA**)

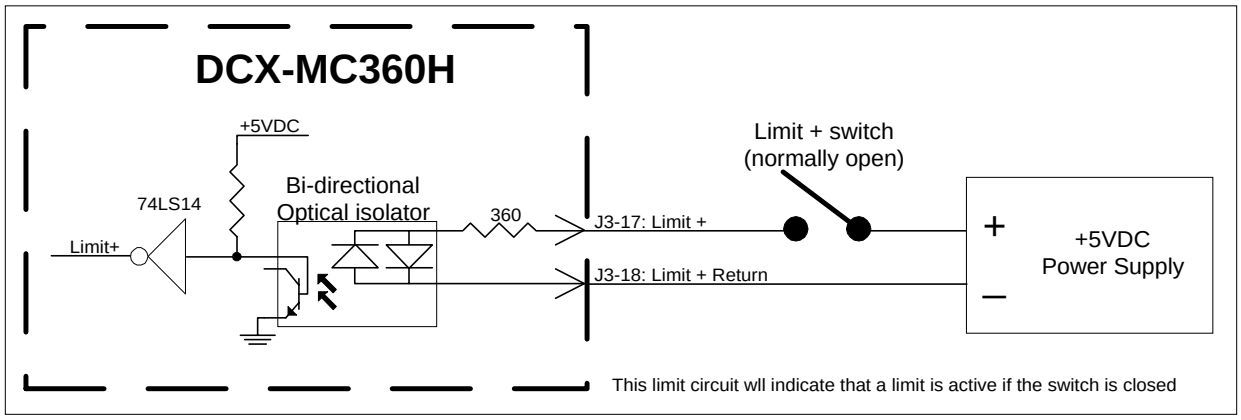
DCX-MC360 Module Layout

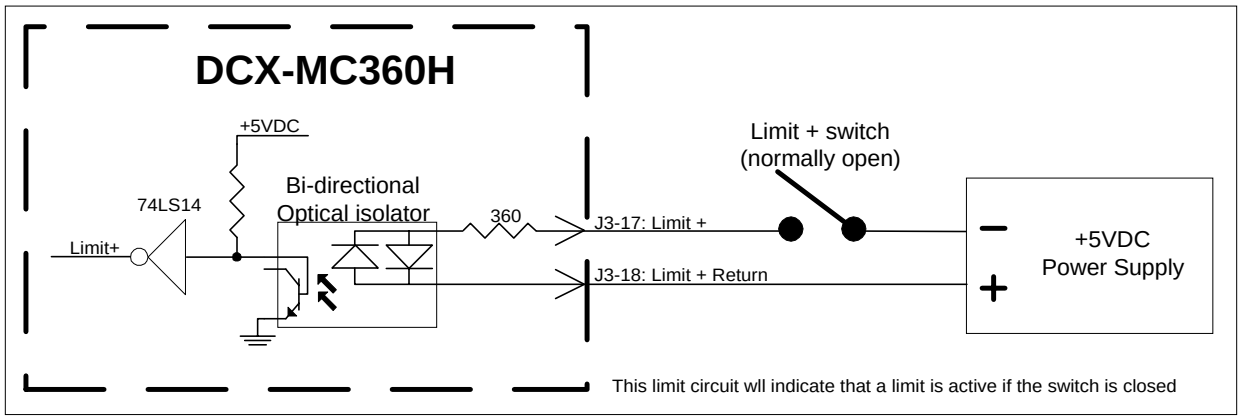


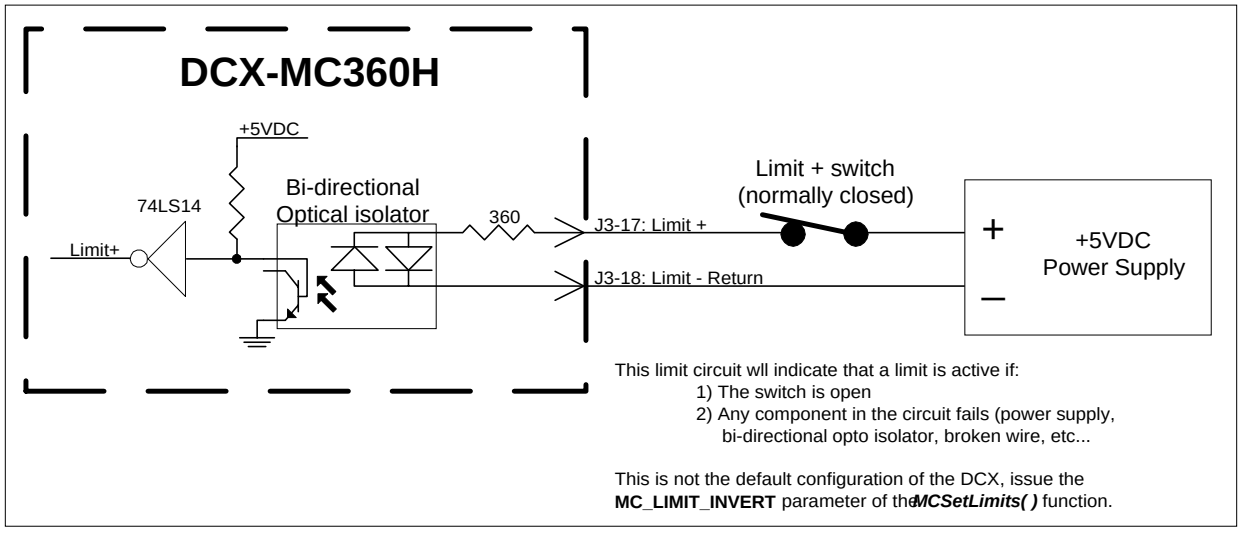
DCX-MC360H Axis I/O Interface Schematic



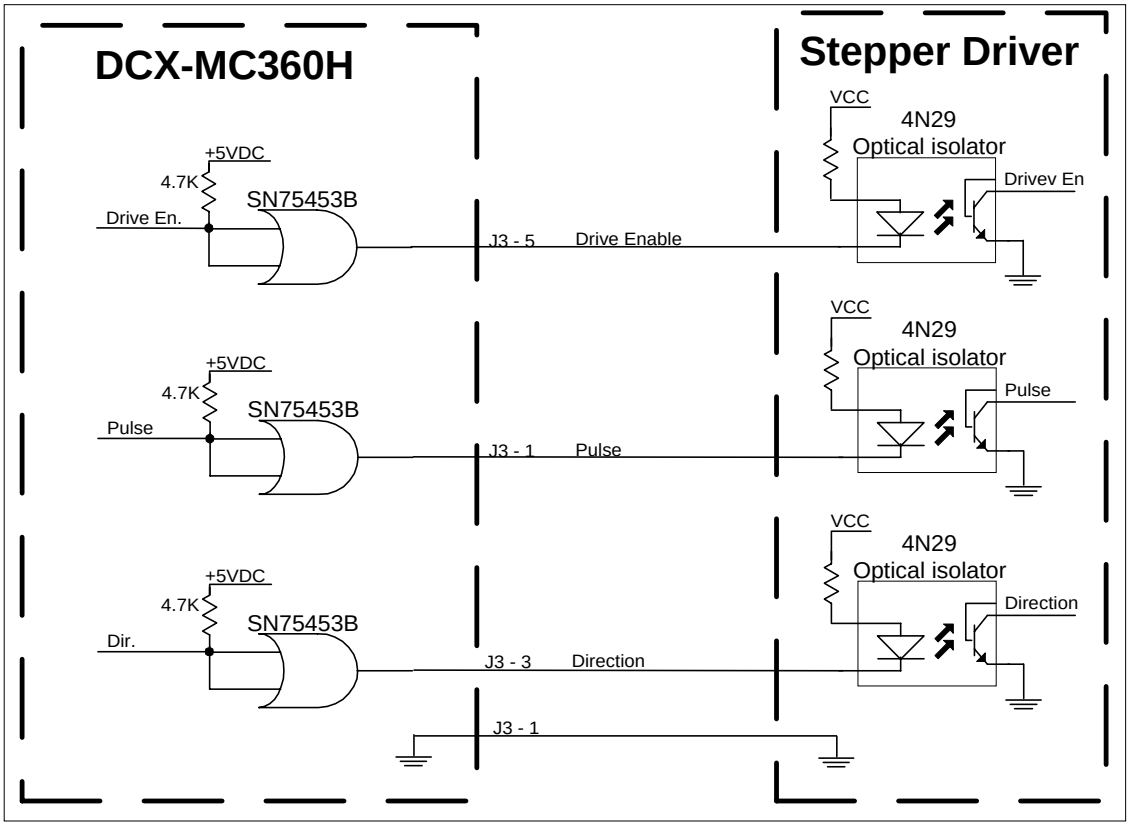
DCX-MC360H Optically Isolated Inputs Wiring Examples







DCX-MC360H Open Collector Driver Wiring Examples



DCX-MC362 Dual Axis Stepper Motor Control Module

SIGNAL DESCRIPTIONS:

Pulse and Direction Outputs
connection point: Axis 1 Direction/CW – J3 pin 12
Axis 1 Pulse/CCW – J3 pin 11
Axis 2 Direction/CW – J3 pin 15
Axis 2 Pulse/CCW – J3 pin 16
signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: external pull-up required
explanation: In the control of a stepper motor, the two primary control signals are Pulse and Direction (or CW Pulse and CCW Pulse). These signals are connected to the external stepper motor driver that supplies current to the motor windings. In order for the stepper module to move the motor one step, a pulse is generated on one of these signals.

Both of these signals are driven by high current open collector drivers and are suitable for direct connection to optically isolated inputs commonly found on stepper motor drivers. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless pull-up resistors are connected to them.

Pulse: The motor driver should advance the motor by one increment for each pulse. The motor may advance a full step, a half step, or a micro step. This is determined by the mode of the stepper motor driver. The Pulse signal is normally high, and is pulled low at the beginning of a step. It stays low for one half the step period (50% duty cycle), and then goes back high. When it is time for the next step, the signal will be pulled low again.

Direction: This signal indicates the direction the motor will move. When the stepper is incrementing the current position (moving positive) this signal will remain high (pulled up). When the stepper is decrementing the current position (moving negative) this signal will be pulled low.

The function ***MCSetModuleOutputMode( )*** is used to change the operation of these signals to CW and CCW Pulse. In this mode, pulses will be generated on the CW Pulse output when the current position is increasing, and on the CCW Pulse output when the current position is decreasing.

Drive Enable Output
connection point: Axis 1 - J3 pin 9
Axis21 - J3 pin 17
signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: external pull-up required
explanation: This output will be pulled low when an axis is enabled (***MCEnableAxis( )***). It will remain low until: the axis is disabled or an error condition exists (limits tripped).

This signal is driven by a high current open collector driver and is suitable for direct connection to optically isolated inputs commonly found on a servo amplifier. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless pull-up resistors are connected to them.

Limit Positive and Limit Negative Inputs

connection point: Axis 1 Limit Positive – J3 pin 3
Axis 1 Limit Negative – J3 pin 5
Axis 2 Limit Positive – J3 pin 23
Axis 2 Limit Negative – J3 pin 21

signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range

notes: Axis 1 Limit + Supply/Return - J3 pin 4
Axis 1 Limit - Supply/Return - J3 pin 6
Axis 2 Limit + Supply/Return - J3 pin 24
Axis 2 Limit - Supply/Return - J3 pin 22

explanation: The limit switch inputs are used to cause the DCX to stop a servo's motion when it reaches the end of travel. If the servo is in position mode, the axis will only be stopped if it is moving in the direction of an activated limit switch. In all other modes, the servo will be stopped regardless of the direction it is moving if either limit switch is activated. There are three modes of stopping (decelerate to a stop, stop immediately, turn off the axis) that can be configured by the ***MCSetLimits( )***. The limit switch inputs can be enabled and disabled by ***MCSetLimits( )***. See the description of **Motion Limits** in the **Motion Control** chapter.

The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Home Input

connection point: Axis 1 - J3 - pin 1
Axis 2 - J3 - pin 25

signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range

notes: Axis 1 Supply/Return - J3 pin 2
Axis 2 Supply/Return - J3 pin 26

explanation: This input is used to set the zero position of a stepper motor. It is typically connected to a sensor/switch that is activated at a fixed position in the motor's range of motion. The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Drive Fault Input

connection point: Axis 1 - J3 - pin 7
Axis 2 - J3 - pin 19

signal type: Bi-directional optical isolator, 10ma min. 2.5V – 7.5V range

notes: Axis 1 Supply/Return - J3 pin 8
Axis 2 Supply/Return - J3 pin 19

explanation: This module input is designed to be connected to the Fault or Error output signal of a stepper driver. The state of this signal will appear as a status bit in the servo's status word. The **EnableAmpFault** member of the **MCMotion** structure will enable the module to shut off the axis if the Drive Fault input is active. No further motion will occur until the fault signal is deactivated and the axis has been enabled. The input device is a bi-directional optical isolator. The allowable voltage range for this signal is **2.5 VDC to 7.5 VDC**. For I/O systems operating at higher voltage levels add an external resistor (12 volt I/O = 1.1K, 5%, 1/4W. 24 volt I/O = 4.3K, 5%, 1/4W).

Full/Half Current Output
connection point: Axis 1 - J3 - pin 13
Axis 2 - J3 - pin 14
signal type: Open collector, current sink, 100ma max. current sink, 30V max.
notes: external pull-up required
explanation: This signal is used if the stepper driver has a digital input for current control. The default condition of this signal is to be inactive (pulled high). Setting the **MC_CURRENT_FULL** parameter of the **MCMotion** structure will cause the signal to be pulled low.

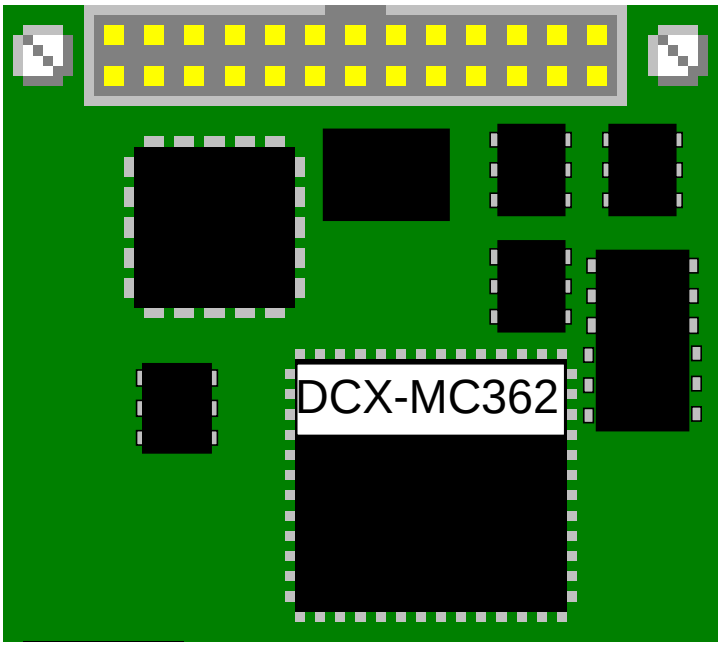
This signal is driven by a high current open collector driver and is suitable for direct connection to optically isolated inputs commonly found on a servo amplifier. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless pull-up resistors are connected to them.

DCX-MC362-H High Density connector signal map

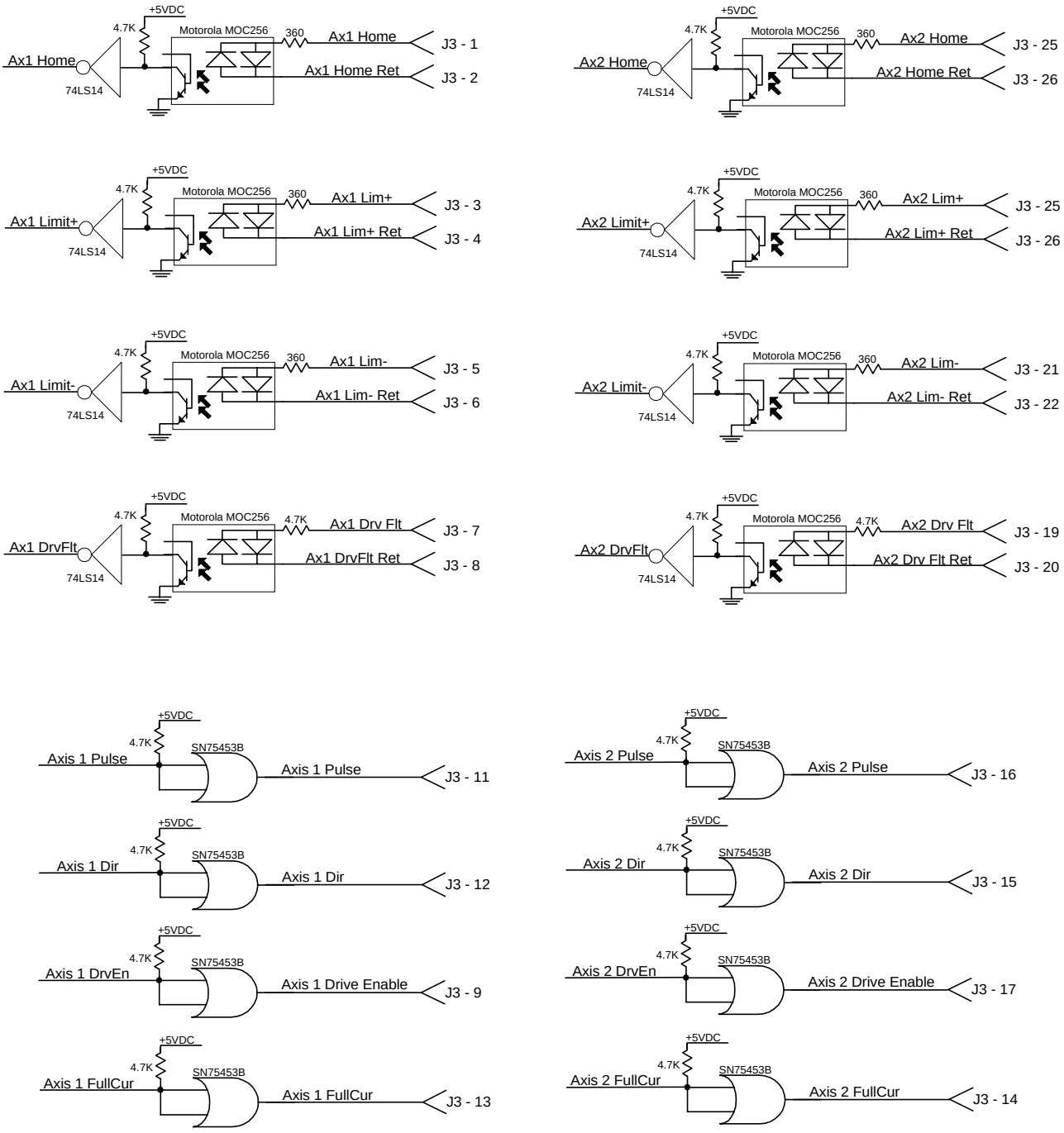
Module #1	Module #2	Module #3	Module #4	Module #5	Module #6	Module #7	Module #8	J3 Pin #	Description
J1 – 1	J2 – 1	J3 - 19	J4 - 19	J3 – 1	J4 – 1	J1 – 19	J2 - 19	1	Axis 1 Home: input (optically isolated)
J1 – 35	J2 - 35	J3 – 53	J4 – 53	J3 – 35	J4 – 35	J1 – 53	J2 – 53	2	Axis 1 Home opto isolator supply/return
J1 – 2	J2 – 2	J3 – 20	J4 – 20	J3 – 2	J4 – 2	J1 – 20	J2 – 20	3	Axis 1 Limit Positive: input (optically isolated)
J1 – 36	J2 - 36	J3 – 54	J4 – 54	J3 – 36	J4 – 36	J1 – 54	J2 – 54	4	Axis 1 Limit Positive opto isolator supply/return
J1 – 3	J2 – 3	J3 - 21	J4 - 21	J3 – 3	J4 – 3	J1 – 21	J2 - 21	5	Axis 1 Limit Negative: input (optically isolated)
J1 – 37	J2 - 37	J3 – 55	J4 – 55	J3 – 37	J4 – 37	J1 – 55	J2 – 55	6	Axis 1 Limit Negative opto isolator supply/return
J1 – 4	J2 – 4	J3 – 22	J4 – 22	J3 – 4	J4 – 4	J1 – 22	J2 – 22	7	Axis 1 Drive Fault: input (optically isolated)
J1 – 38	J2 - 38	J3 – 56	J4 – 56	J3 – 38	J4 – 38	J1 – 56	J2 – 56	8	Axis 1 Drive Fault opto isolator supply/return
J1 – 5	J2 – 5	J3 – 23	J4 – 23	J3 – 5	J4 – 5	J1 – 23	J2 – 23	9	Axis 1 Driver Enable: output (open collector)
J1 – 39	J2 - 39	J3 – 57	J4 – 57	J3 – 39	J4 – 39	J1 – 57	J2 – 57	10	Ground
J1 – 6	J2 – 6	J3 – 24	J4 – 24	J3 – 6	J4 – 6	J1 – 24	J2 – 24		Ground
J1 – 40	J2 - 40	J3 – 58	J4 – 58	J3 – 40	J4 – 40	J1 – 58	J2 – 58	11	Axis 1 Pulse or CCW Pulse: output
J1 – 7	J2 – 7	J3 – 25	J4 – 25	J3 – 7	J4 – 7	J1 – 25	J2 – 25	12	Axis 1 Direction or CW Pulse: output
J1 – 41	J2 - 41	J3 – 59	J4 – 59	J3 – 41	J4 – 41	J1 – 59	J2 – 59		Ground
J1 – 8	J2 – 8	J3 – 26	J4 – 26	J3 – 8	J4 – 8	J1 – 26	J2 – 26		Ground
J1 – 42	J2 - 42	J3 – 60	J4 – 60	J3 – 42	J4 – 42	J1 – 60	J2 – 60	13	Axis 1 Full/Half Current: output
J1 – 9	J2 – 9	J3 – 27	J4 – 27	J3 – 9	J4 – 9	J1 – 27	J2 – 27	14	Axis 2 Full/Half Current: output
J1 – 43	J2 - 43	J3 – 61	J4 – 61	J3 – 43	J4 – 43	J1 – 61	J2 – 61		Ground
J1 – 10	J2 - 10	J3 – 28	J4 – 28	J3 – 10	J4 – 10	J1 – 28	J2 – 28		Ground
J1 – 44	J2 - 44	J3 – 62	J4 – 62	J3 – 44	J4 – 44	J1 – 62	J2 – 62	15	Axis 2 Direction or CW Pulse: output
J1 – 11	J2 - 11	J3 - 29	J4 - 29	J3 – 11	J4 – 11	J1 – 29	J2 - 29	16	Axis 2 Pulse or CCW Pulse: output
J1 – 45	J2 - 45	J3 – 63	J4 – 63	J3 – 45	J4 – 45	J1 – 63	J2 – 63		Ground
J1 – 12	J2 - 12	J3 – 30	J4 – 30	J3 – 12	J4 – 12	J1 – 30	J2 – 30	17	Axis 2 Driver Enable: output
J1 – 46	J2 - 46	J3 – 64	J4 – 64	J3 – 46	J4 – 46	J1 – 64	J2 – 64	18	Ground
J1 – 13	J2 - 13	J3 – 31	J4 – 31	J3 – 13	J4 – 13	J1 – 31	J2 – 31	19	Axis 2 Drive Fault: input (optically isolated)
J1 – 47	J2 - 47	J3 – 65	J4 – 65	J3 – 47	J4 – 47	J1 – 65	J2 – 65	20	Axis 2 Drive Fault opto isolator supply/return
J1 – 14	J2 - 14	J3 – 32	J4 – 32	J3 – 14	J4 – 14	J1 – 32	J2 – 32	21	Axis 2 Limit Negative: input (optically isolated)
J1 – 48	J2 - 48	J3 – 66	J4 – 66	J3 – 48	J4 – 48	J1 – 66	J2 – 66	22	Axis 2 Limit Negative opto isolator supply/return
J1 – 15	J2 - 15	J3 – 33	J4 – 33	J3 – 15	J4 – 15	J1 – 33	J2 – 33	23	Axis 2 Limit Positive: input (optically isolated)
J1 – 49	J2 - 49	J3 – 67	J4 – 67	J3 – 49	J4 – 49	J1 – 67	J2 – 67	24	Axis 2 Limit Positive opto isolator supply/return
J1 – 16	J2 - 16	J3 – 34	J4 – 34	J3 – 16	J4 – 16	J1 – 34	J2 – 34	25	Axis 2 Home: input (optically isolated)
J1 – 50	J2 - 50	J3 – 68	J4 – 68	J3 – 50	J4 – 50	J1 – 68	J2 – 68	26	Axis 2 Home opto isolator supply/return
J1 – 17	J2 – 17			J3 – 17	J4 – 17				Ground
J1 – 51	J2 - 51			J3 – 51	J4 – 51				Ground
J1 – 18	J2 – 18			J3 – 18	J4 – 18				Ground
J1 – 52	J3 – 52			J3 – 52	J4 – 52				Ground

For a more complete signal description please refer to the previous five pages
Mating cable connector components: Amp: 787801-1 (offset connector), 788362-1 (backshell, offset)

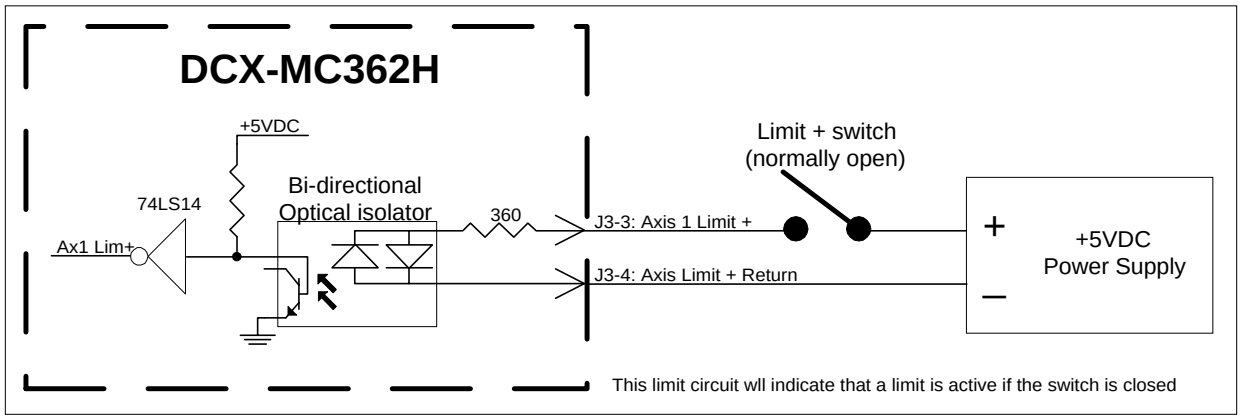
DCX-MC362 Module Layout

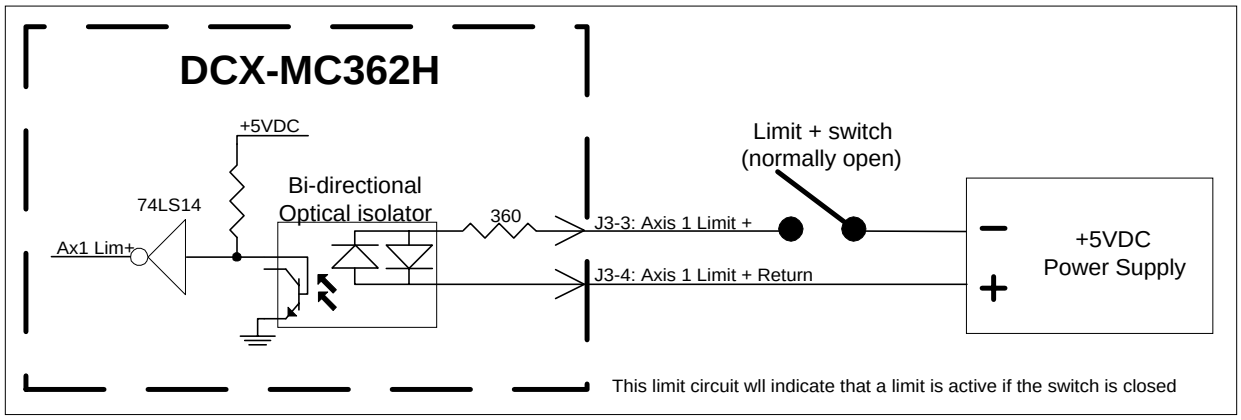


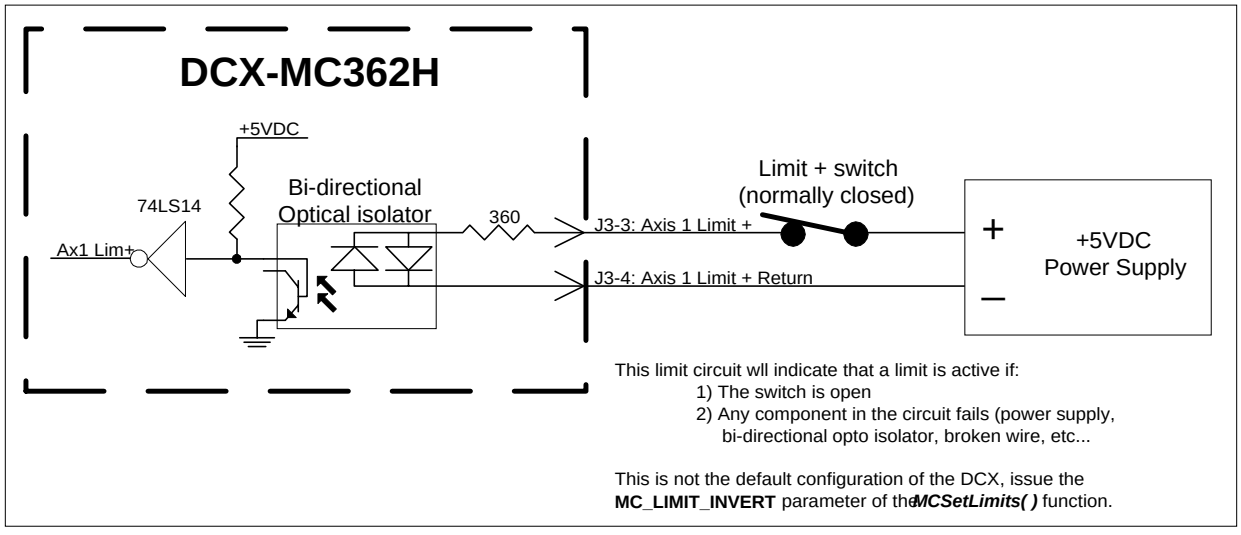
DCX-MC362H Axis I/O Interface Schematic



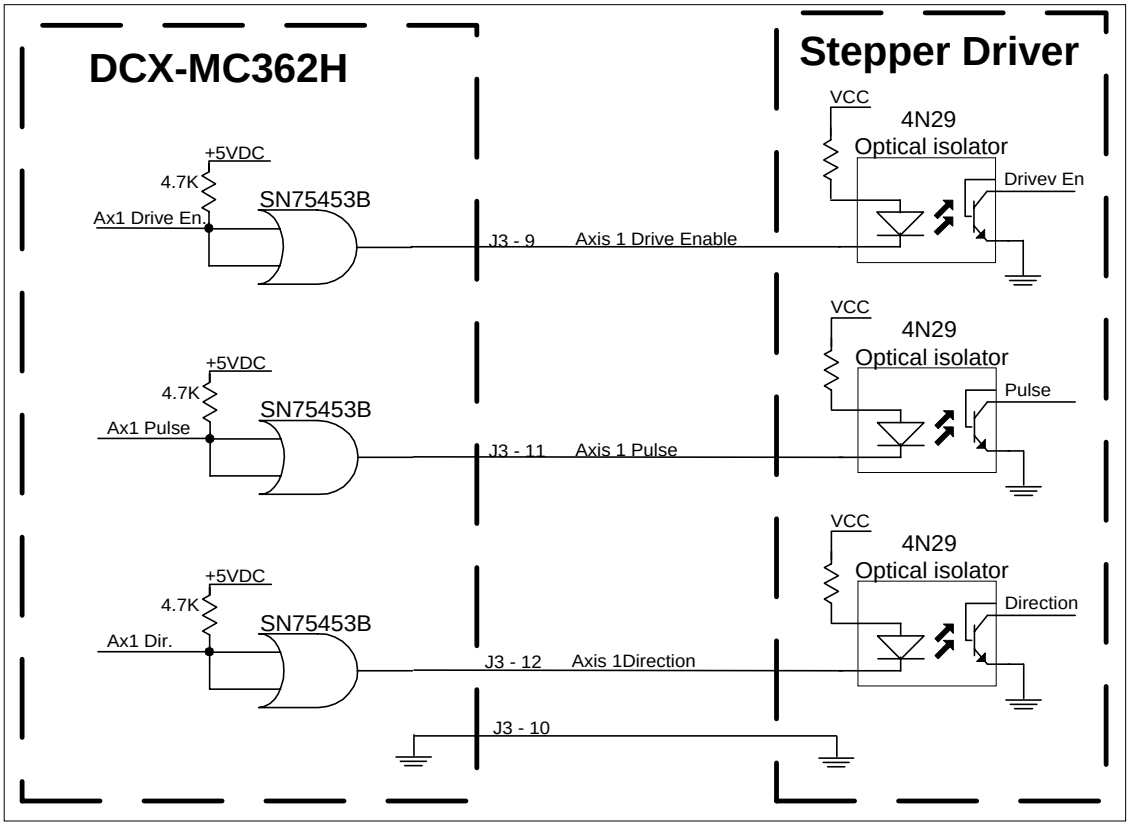
DCX-MC362H Optically Isolated Inputs Wiring Examples







DCX-MC362H Open Collector Driver Wiring Examples



DCX-MC400 Digital I/O Module

DCX-MC400 Electrical Specifications

Parameter	*Min.*	*Max*	*Unit*
Digital Input – High voltage	2.0	5.3	V
Digital Input – Low voltage	-0.3	0.8	V
Digital Output – High voltage	2.4		V (current source 0.25ma)
Digital Output – Low voltage		0.4	V (current source 2.0ma)
Input leakage		+/- 10.0	uA

DCX-MC400-H High Density connector signal map

Module #1	Module #2	Module #3	Module #4	Module #5	Module #6	Module #7	Module #8	J3 Pin #	Description
J1 – 1	J2 – 1	J3 - 19	J4 - 19	J3 – 1	J4 – 1	J1 – 19	J2 - 19	1	Ground
J1 – 35	J2 - 35	J3 – 53	J4 – 53	J3 – 35	J4 – 35	J1 – 53	J2 – 53	2	Digital I/O channel #1
J1 – 2	J2 – 2	J3 – 20	J4 – 20	J3 – 2	J4 – 2	J1 – 20	J2 – 20	3	Ground
J1 – 36	J2 - 36	J3 – 54	J4 – 54	J3 – 36	J4 – 36	J1 – 54	J2 – 54	4	Digital I/O channel #2
J1 – 3	J2 – 3	J3 - 21	J4 - 21	J3 – 3	J4 – 3	J1 – 21	J2 - 21	5	Ground
J1 – 37	J2 - 37	J3 – 55	J4 – 55	J3 – 37	J4 – 37	J1 – 55	J2 – 55	6	Digital I/O channel #3
J1 – 4	J2 – 4	J3 – 22	J4 – 22	J3 – 4	J4 – 4	J1 – 22	J2 – 22	7	Ground
J1 – 38	J2 - 38	J3 – 56	J4 – 56	J3 – 38	J4 – 38	J1 – 56	J2 – 56	8	Digital I/O channel #4
J1 – 5	J2 – 5	J3 – 23	J4 – 23	J3 – 5	J4 – 5	J1 – 23	J2 – 23	9	Ground
J1 – 39	J2 - 39	J3 – 57	J4 – 57	J3 – 39	J4 – 39	J1 – 57	J2 – 57	10	Digital I/O channel #5
J1 – 6	J2 – 6	J3 – 24	J4 – 24	J3 – 6	J4 – 6	J1 – 24	J2 – 24		Ground
J1 – 40	J2 - 40	J3 – 58	J4 – 58	J3 – 40	J4 – 40	J1 – 58	J2 – 58	11	Digital I/O channel #6
J1 – 7	J2 – 7	J3 – 25	J4 – 25	J3 – 7	J4 – 7	J1 – 25	J2 – 25	12	Digital I/O channel #7
J1 – 41	J2 - 41	J3 – 59	J4 – 59	J3 – 41	J4 – 41	J1 – 59	J2 – 59		Ground
J1 – 8	J2 – 8	J3 – 26	J4 – 26	J3 – 8	J4 – 8	J1 – 26	J2 – 26		Ground
J1 – 42	J2 - 42	J3 – 60	J4 – 60	J3 – 42	J4 – 42	J1 – 60	J2 – 60	13	Digital I/O channel #8
J1 – 9	J2 – 9	J3 – 27	J4 – 27	J3 – 9	J4 – 9	J1 – 27	J2 – 27	14	Digital I/O channel #9
J1 – 43	J2 - 43	J3 – 61	J4 – 61	J3 – 43	J4 – 43	J1 – 61	J2 – 61		Ground
J1 – 10	J2 - 10	J3 – 28	J4 – 28	J3 – 10	J4 – 10	J1 – 28	J2 – 28		Ground
J1 – 44	J2 - 44	J3 – 62	J4 – 62	J3 – 44	J4 – 44	J1 – 62	J2 – 62	15	Digital I/O channel #10
J1 – 11	J2 - 11	J3 - 29	J4 - 29	J3 – 11	J4 – 11	J1 – 29	J2 - 29	16	Digital I/O channel #11
J1 – 45	J2 - 45	J3 – 63	J4 – 63	J3 – 45	J4 – 45	J1 – 63	J2 – 63		Ground
J1 – 12	J2 - 12	J3 – 30	J4 – 30	J3 – 12	J4 – 12	J1 – 30	J2 – 30	17	Ground
J1 – 46	J2 - 46	J3 – 64	J4 – 64	J3 – 46	J4 – 46	J1 – 64	J2 – 64	18	Digital I/O channel #12
J1 – 13	J2 - 13	J3 – 31	J4 – 31	J3 – 13	J4 – 13	J1 – 31	J2 – 31	19	Ground
J1 – 47	J2 - 47	J3 – 65	J4 – 65	J3 – 47	J4 – 47	J1 – 65	J2 – 65	20	Digital I/O channel #13
J1 – 14	J2 - 14	J3 – 32	J4 – 32	J3 – 14	J4 – 14	J1 – 32	J2 – 32	21	Ground
J1 – 48	J2 - 48	J3 – 66	J4 – 66	J3 – 48	J4 – 48	J1 – 66	J2 – 66	22	Digital I/O channel #14
J1 – 15	J2 - 15	J3 – 33	J4 – 33	J3 – 15	J4 – 15	J1 – 33	J2 – 33	23	Ground
J1 – 49	J2 - 49	J3 – 67	J4 – 67	J3 – 49	J4 – 49	J1 – 67	J2 – 67	24	Digital I/O channel #15
J1 – 16	J2 - 16	J3 – 34	J4 – 34	J3 – 16	J4 – 16	J1 – 34	J2 – 34	25	Ground
J1 – 50	J2 - 50	J3 – 68	J4 – 68	J3 – 50	J4 – 50	J1 – 68	J2 – 68	26	Digital I/O channel #16
J1 – 17	J2 – 17			J3 – 17	J4 – 17				Ground
J1 – 51	J2 - 51			J3 – 51	J4 – 51				Ground
J1 – 18	J2 – 18			J3 – 18	J4 – 18				Ground
J1 – 52	J3 – 52			J3 – 52	J4 – 52				Ground

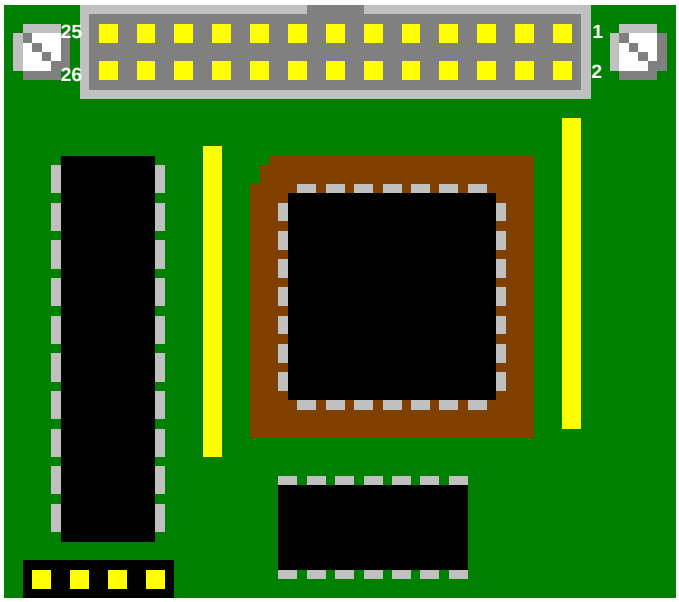
For a more complete signal description please refer to the previous five pages
Mating cable connector components: Amp: 787801-1 (offset connector), 788362-1 (backshell, offset)

DCX-MC400-R connector pin-out

Pin #	*Description*
1	Digital I/O channel #1
2	Digital I/O channel #2
3	Digital I/O channel #3
4	Digital I/O channel #4
5	Digital I/O channel #5
6	Digital I/O channel #6
7	Digital I/O channel #7
8	Digital I/O channel #8
9	Digital I/O channel #9
10	Digital I/O channel #10
11	Digital I/O channel #11
12	Digital I/O channel #12
13	Digital I/O channel #13
14	Digital I/O channel #14
15	Digital I/O channel #15
16	Digital I/O channel #16
17	Reserved
18	Reserved
19	Reserved
20	+5 VDC
21	Ground
22	Reserved
23	Reserved
24	Reserved
25	Reserved
26	Ground

Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N 26IDS2-C-SPT-SR or equivalent

DCX-MC400 Module layout



DCX-MC500/510/520 Analog I/O Module

DCX-MC500-H/510-H/520-H High Density connector signal map

Module #1	Module #2	Module #3	Module #4	Module #5	Module #6	Module #7	Module #8	J3 Pin #	Description
J1 – 1	J2 – 1	J3 - 19	J4 - 19	J3 – 1	J4 – 1	J1 – 19	J2 - 19	1	Ground
J1 – 35	J2 - 35	J3 – 53	J4 – 53	J3 – 35	J4 – 35	J1 – 53	J2 – 53	2	Channel 1 Output (-10 to +10 volts)
J1 – 2	J2 – 2	J3 – 20	J4 – 20	J3 – 2	J4 – 2	J1 – 20	J2 – 20	3	Ground
J1 – 36	J2 - 36	J3 – 54	J4 – 54	J3 – 36	J4 – 36	J1 – 54	J2 – 54	4	Channel 2 Output (-10 to +10 volts)
J1 – 3	J2 – 3	J3 - 21	J4 - 21	J3 – 3	J4 – 3	J1 – 21	J2 - 21	5	Ground
J1 – 37	J2 - 37	J3 – 55	J4 – 55	J3 – 37	J4 – 37	J1 – 55	J2 – 55	6	Channel 3 Output (-10 to +10 volts)
J1 – 4	J2 – 4	J3 – 22	J4 – 22	J3 – 4	J4 – 4	J1 – 22	J2 – 22	7	Ground
J1 – 38	J2 - 38	J3 – 56	J4 – 56	J3 – 38	J4 – 38	J1 – 56	J2 – 56	8	Channel 4 Output (-10 to +10 volts)
J1 – 5	J2 – 5	J3 – 23	J4 – 23	J3 – 5	J4 – 5	J1 – 23	J2 – 23	9	Ground
J1 – 39	J2 - 39	J3 – 57	J4 – 57	J3 – 39	J4 – 39	J1 – 57	J2 – 57	10	External A/D reference input (see jumper JP1)
J1 – 6	J2 – 6	J3 – 24	J4 – 24	J3 – 6	J4 – 6	J1 – 24	J2 – 24		Ground
J1 – 40	J2 - 40	J3 – 58	J4 – 58	J3 – 40	J4 – 40	J1 – 58	J2 – 58	11	Channel 1 Output (0 to +5 volts)
J1 – 7	J2 – 7	J3 – 25	J4 – 25	J3 – 7	J4 – 7	J1 – 25	J2 – 25	12	Channel 2 Output (0 to +5 volts)
J1 – 41	J2 - 41	J3 – 59	J4 – 59	J3 – 41	J4 – 41	J1 – 59	J2 – 59		Ground
J1 – 8	J2 – 8	J3 – 26	J4 – 26	J3 – 8	J4 – 8	J1 – 26	J2 – 26		Ground
J1 – 42	J2 - 42	J3 – 60	J4 – 60	J3 – 42	J4 – 42	J1 – 60	J2 – 60	13	Channel 3 Output (0 to +5 volts)
J1 – 9	J2 – 9	J3 – 27	J4 – 27	J3 – 9	J4 – 9	J1 – 27	J2 – 27	14	Channel 4 Output (0 to +5 volts)
J1 – 43	J2 - 43	J3 – 61	J4 – 61	J3 – 43	J4 – 43	J1 – 61	J2 – 61		Ground
J1 – 10	J2 - 10	J3 – 28	J4 – 28	J3 – 10	J4 – 10	J1 – 28	J2 – 28		Ground
J1 – 44	J2 - 44	J3 – 62	J4 – 62	J3 – 44	J4 – 44	J1 – 62	J2 – 62	15	+12 VDC
J1 – 11	J2 - 11	J3 - 29	J4 - 29	J3 – 11	J4 – 11	J1 – 29	J2 - 29	16	-12 VDC
J1 – 45	J2 - 45	J3 – 63	J4 – 63	J3 – 45	J4 – 45	J1 – 63	J2 – 63		Ground
J1 – 12	J2 - 12	J3 – 30	J4 – 30	J3 – 12	J4 – 12	J1 – 30	J2 – 30	17	No connect
J1 – 46	J2 - 46	J3 – 64	J4 – 64	J3 – 46	J4 – 46	J1 – 64	J2 – 64	18	No connect
J1 – 13	J2 - 13	J3 – 31	J4 – 31	J3 – 13	J4 – 13	J1 – 31	J2 – 31	19	Ground
J1 – 47	J2 - 47	J3 – 65	J4 – 65	J3 – 47	J4 – 47	J1 – 65	J2 – 65	20	Channel 1 Input (0 to +5 volts)
J1 – 14	J2 - 14	J3 – 32	J4 – 32	J3 – 14	J4 – 14	J1 – 32	J2 – 32	21	Ground
J1 – 48	J2 - 48	J3 – 66	J4 – 66	J3 – 48	J4 – 48	J1 – 66	J2 – 66	22	Channel 2 Input (0 to +5 volts)
J1 – 15	J2 - 15	J3 – 33	J4 – 33	J3 – 15	J4 – 15	J1 – 33	J2 – 33	23	Ground
J1 – 49	J2 - 49	J3 – 67	J4 – 67	J3 – 49	J4 – 49	J1 – 67	J2 – 67	24	Channel 3 Input (0 to +5 volts)
J1 – 16	J2 - 16	J3 – 34	J4 – 34	J3 – 16	J4 – 16	J1 – 34	J2 – 34	25	Ground
J1 – 50	J2 - 50	J3 – 68	J4 – 68	J3 – 50	J4 – 50	J1 – 68	J2 – 68	26	Channel 4 Input (0 to +5 volts)
J1 – 17	J2 – 17			J3 – 17	J4 – 17				Ground
J1 – 51	J2 - 51			J3 – 51	J4 – 51				Ground
J1 – 18	J2 – 18			J3 – 18	J4 – 18				Ground
J1 – 52	J3 – 52			J3 – 52	J4 – 52				Ground

For a more complete signal description please refer to the previous five pages
Mating cable connector components: Amp: 787801-1 (offset connector), 788362-1 (backshell, offset)

DCX-MC5X0-R connector pin-out

Pin #	*Description*
1	Channel 1 Input (0 to +5 volts)
2	Channel 1 Output (-10 to +10 volts)
3	Channel 2 Input (0 to +5 volts)
4	Channel 2 Output (-10 to +10 volts)
5	Channel 3 Input (0 to +5 volts)
6	Channel 3 Output (-10 to +10 volts)
7	Channel 4 Input (0 to +5 volts)
8	Channel 4 Output (-10 to +10 volts)
9	Reserved
10	Channel 1 Output (0 to +5 volts)
11	Reserved
12	Channel 2 Output (0 to +5 volts)
13	Reserved
14	Channel 3 Output (0 to +5 volts)
15	Reserved
16	Channel 4 Output (0 to +5 volts)
17	Analog Ground
18	External A/D reference input (see jumper JP1)
19	+12 VDC
20	-12 VDC
21	No connect
22	No connect
23	+5 VDC
24	+5 VDC
25	Digital Ground
26	Digital Ground

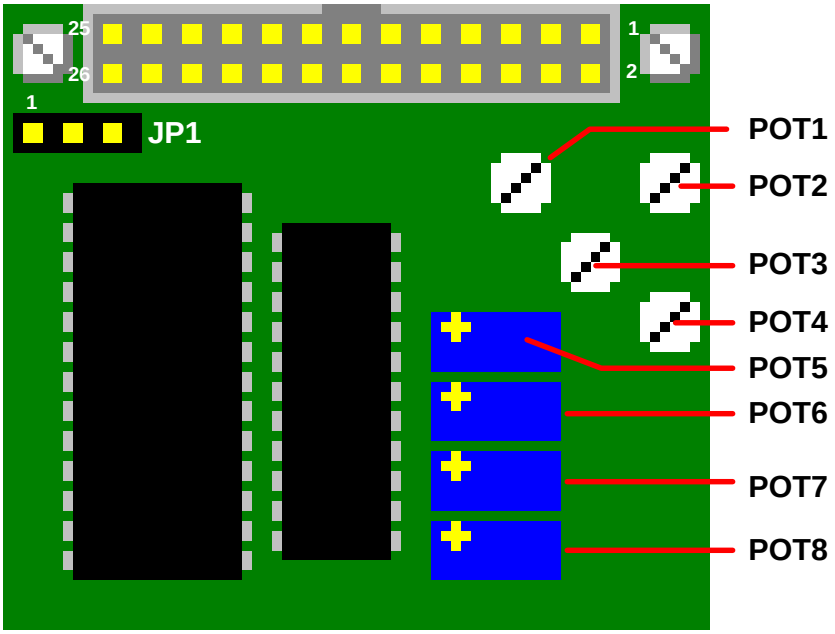
Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N 26IDS2-C-SPT-SR or equivalent

DCX-MC500/510/520 Module Configuration Jumpers - configuration in **bold type** denotes default factory shipping configuration

JP1 – A/D reference select (external reference or on board +5 VDC reference)

Pins	*Description*
1 to 2	Use external reference (supplied by user on J3 pin 18)
2 to 3	**Use the on board +5 VDC reference**

DCX-MC500 Module layout



DCX-BF022 Relay Rack Interface

J1 connector pin-out - The signals are arranged to interface the DCX-MC400 directly to an OPTO 22 relay rack.

Pin #	***Description***
1	Digital I/O channel #1
2	Digital I/O channel #2
3	Digital I/O channel #3
4	Digital I/O channel #4
5	Digital I/O channel #5
6	Digital I/O channel #6
7	Digital I/O channel #7
8	Digital I/O channel #8
9	Digital I/O channel #9
10	Digital I/O channel #10
11	Digital I/O channel #11
12	Digital I/O channel #12
13	Digital I/O channel #13
14	Digital I/O channel #14
15	Digital I/O channel #15
16	Digital I/O channel #16
17	No connect
18	No connect
19	No connect
20	+5 VDC
21	Ground
22	No connect
23	No connect
24	No connect
25	No connect
26	Ground

Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N 26IDS2-C-SPT-SR or equivalent

J2 connector pin-out - The signals are arranged to interface the DCX-PCI300 General Purpose I/O (connector J3) directly to an OPTO 22 relay rack.

Pin #	***Description***
1	+5 VDC
2	No connect
3	Digital I/O channel #16
4	No connect
5	Digital I/O channel #15
6	Digital I/O channel #14
7	Digital I/O channel #13
8	Digital I/O channel #12
9	Digital I/O channel #11
10	Digital I/O channel #10
11	Digital I/O channel #9
12	Digital I/O channel #8
13	Digital I/O channel #7
14	Digital I/O channel #6
15	Digital I/O channel #5
16	Digital I/O channel #4
17	Digital I/O channel #3
18	Digital I/O channel #2
19	Digital I/O channel #1
20	No connect
21	No connect
22	No connect
23	No connect
24	Ground
25	No connect
26	Ground

Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N 26IDS2-C-SPT-SR or equivalent

DCX-BF022 Configuration Jumpers - configuration in **bold type** denotes default factory shipping configuration

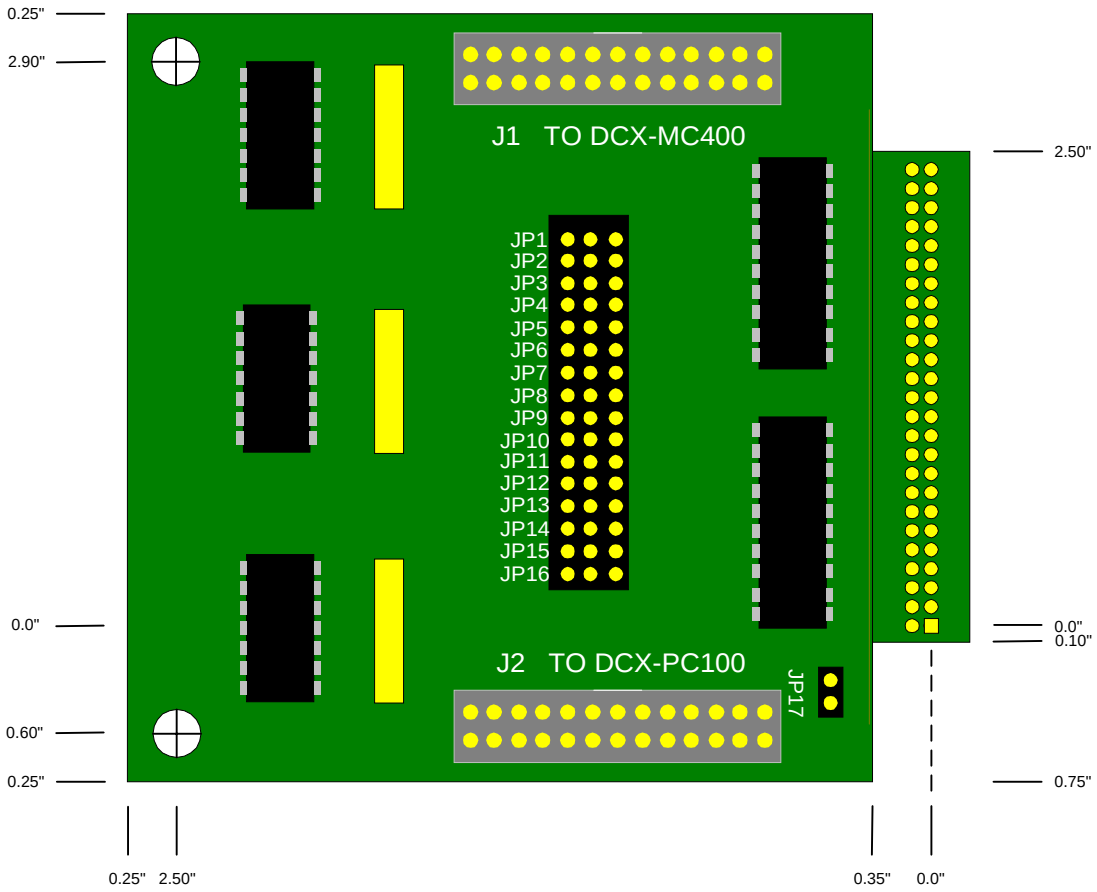
JP1 – JP16 Configure Digital channel as Input or Output

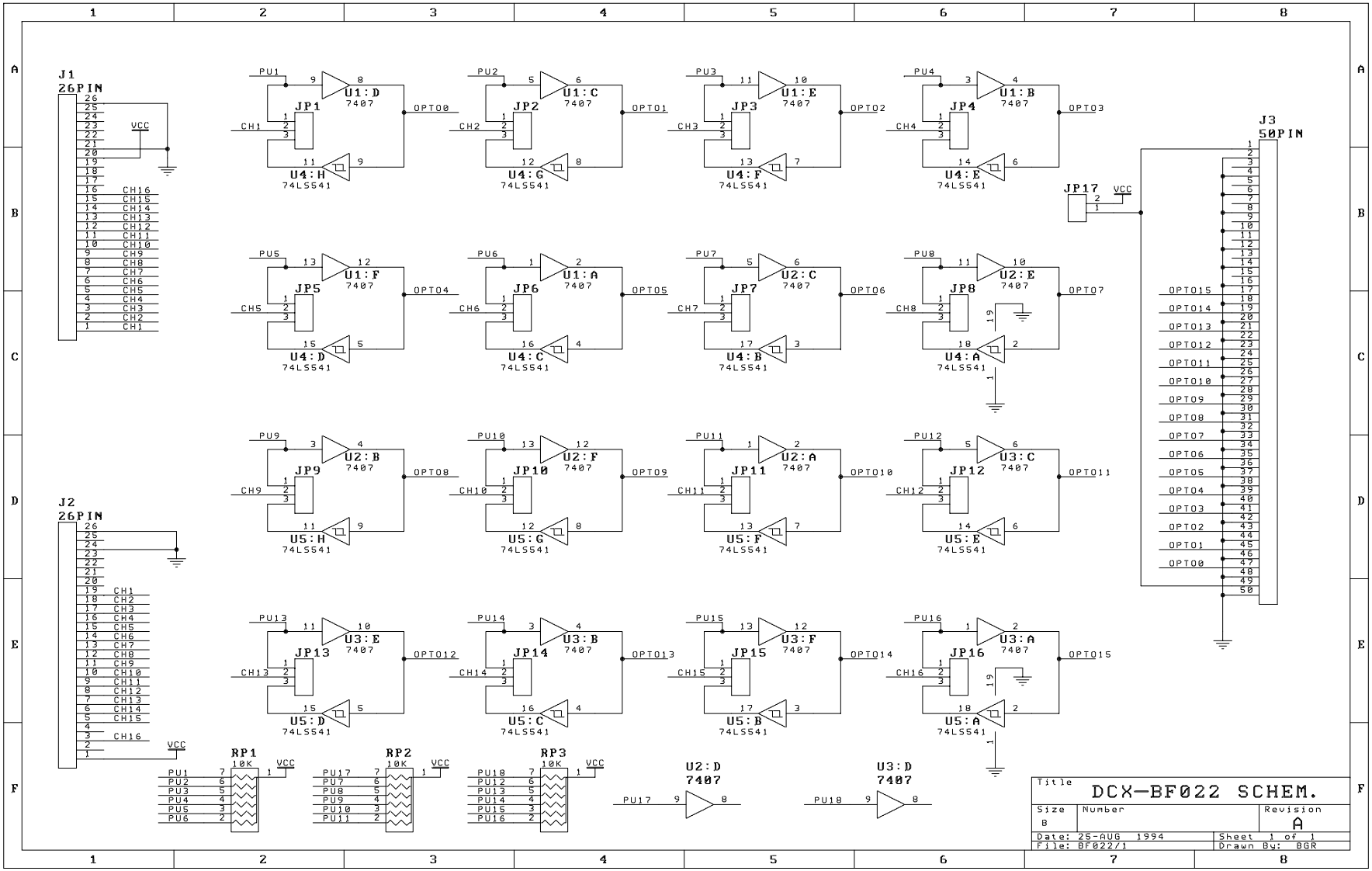
Pins	*Description*
1 to 2	**Configure channel as Output**
2 to 3	Configure channel as an Input

JP17 – Select Relay Rack supply source

Pins	*Description*
1 to 2	**DCX provides +5 VDC Relay Rack supply**
2 to 3	Relay Rack has separate +5 VDC supply

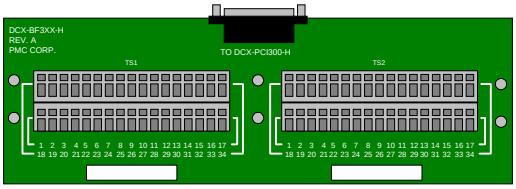
DCX-BF022 Interface layout



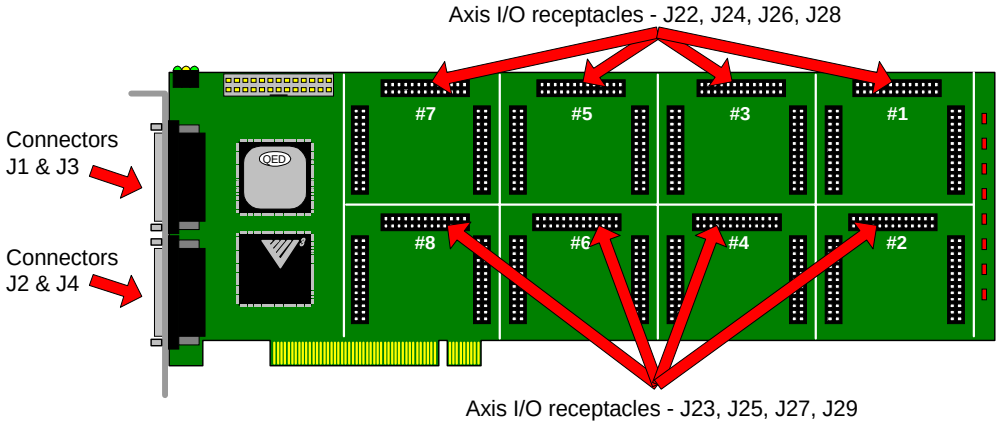

J1
26PIN
J2
26PIN
J3
50PIN
JP1
JP2
JP3
JP4
JP5
JP6
JP7
JP8
JP9
JP10
JP11
JP12
JP13
JP14
JP15
JP16
JP17
U1:D
U1:C
U1:E
U1:B
U1:F
U1:A
U2:C
U2:E
U2:B
U2:F
U2:A
U3:C
U3:E
U3:B
U3:F
U3:A
U4:H
U4:G
U4:F
U4:E
U4:D
U4:C
U4:B
U4:A
U5:H
U5:G
U5:F
U5:E
U5:D
U5:C
U5:B
U5:A
74LS541
7407
RP1
RP2
RP3
10K
VCC
U2:D
7407
U3:D
7407
Title
DCX-BF022 SCHEM.
Size
B
Number
Revision
A
Date: 25-AUG 1994
File: BF022/1
Sheet 1 of 1
Drawn By: BGR

DCX-BF3XX-H High Density Breakout Assembly

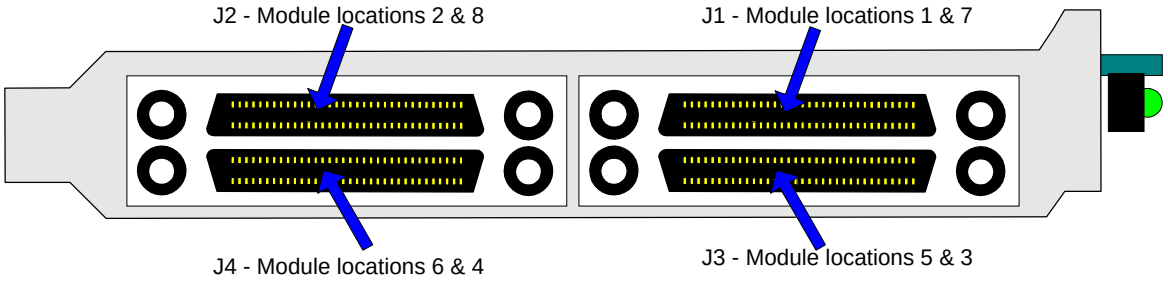
The DCX-BF3XX-H provides easy to use terminal strip contacts for –H DCX modules (MC300-H, MC320-H, MC360-H, MC400-H, MC500-H).



DCX modules can be installed into any one of eight module locations on the DCX-PCI300-H motherboard. The axis I/O signals travel through the inner layers of the DCX-PCI300-H to the high density connectors (J1, J2, J3, and J4).The module, receptacle, and connector locations of a DCX-PCI300-H are shown in the following graphic:



The diagram below details how the DCX-PCI300-H module locations 1 – 8 (receptacles J22 – J29) map into the high density connectors J1 – J4.

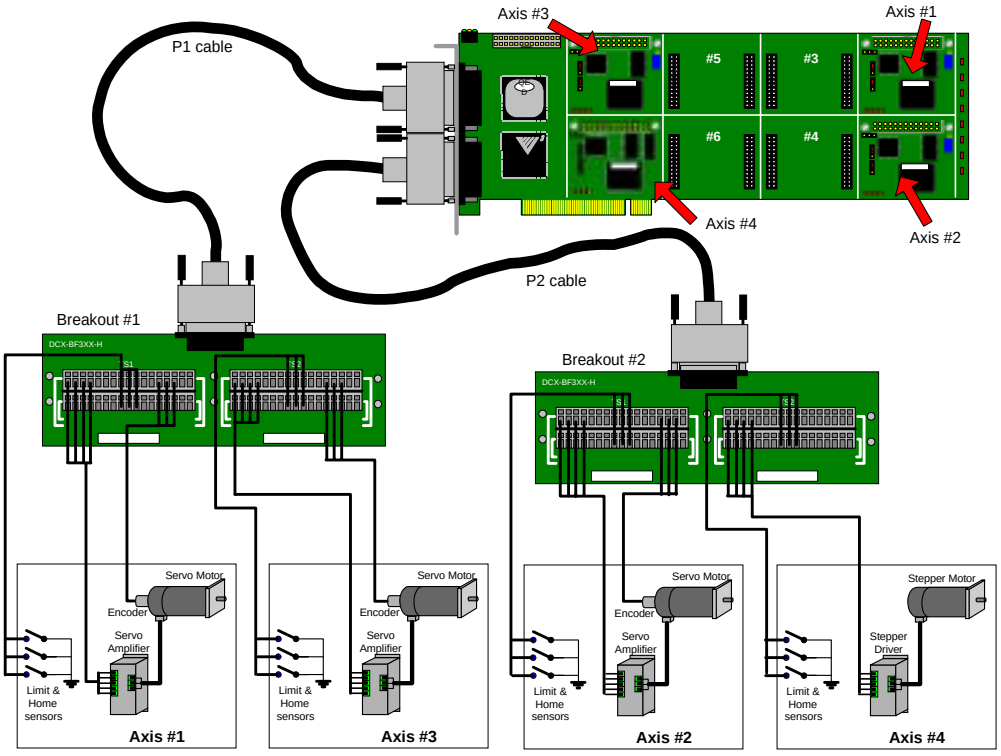


VHDCI connectors as viewed from the back of the computer
(component side down)

Each DCX-BF3XX-H breakout assembly provides contact points for two module locations. The following table details how DCX-PCI300-H motherboard module locations are associated with a DCX-BF3XX-H terminal strip (TS1 or TS2)

DCX-PCI300-H Module location	High Density connector #	Interconnect cable #
1	J1	P1
7	J1	P1
5	J3	P3
3	J3	P3
6	J4	P4
4	J4	P4
2	J2	P2
8	J2	P2

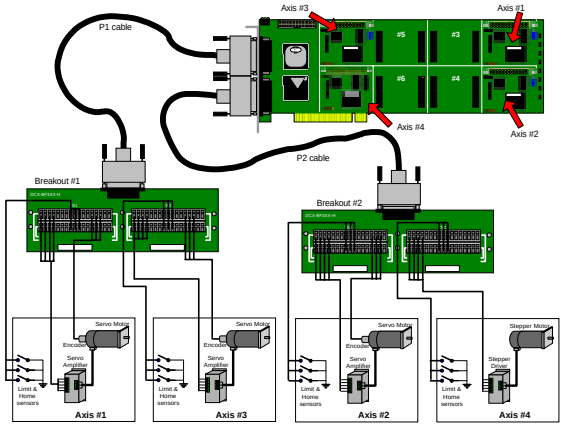
The following diagram details the typical interconnections for a four axis system, three servo's (DCX-MC300-H servo modules in locations 1, 2, & 7) and one stepper (DCX-MC360-H stepper module in location 8). The modules could be installed sequentially into locations #1 - #4, but the system would then require four cables and four DCX-BF3XX-H breakouts instead of two.



DCX-BF3XX-H signals pinout (when using single axis or I/O modules)
The following table details the pinouts –H DCX modules.

BF3XX-H TS1 or TS2	MC300-H	MC320-H	MC360-H	MC400-H	MC500-H
1	Command return	Ground	Step / CCW Pulse	Ground	Ground
18	Command output	Phase U Command	Ground	Digital I/O #1	Output 1 (-10 to +10)
2	Com./Dir. output	Ground	Dir. / CW Pulse	Ground	Ground
19	Com./Dir. return	Phase V Command	Ground	Digital I/O #2	Output 2 (-10 to +10)
3	Amp. Enable output	Amp. Enable output	Driver En. output	Ground	Ground
20	Amp Enable return	Amp. Enable return	Ground	Digital I/O #3	Output 3 (-10 to +10)
4	Amp. Fault input	Amp. Fault: input	Drive Fault: input	Ground	Ground
21	Amp Fault sup./ret.	Amp. Fault return	Drive Fault return	Digital I/O #4	Output 4 (-10 to +10)
5	Coarse Home input	Coarse Home input	Home: input	Ground	Ground
22	Coarse Home ret.	Coarse Home ret.	Home return	Digital I/O #5	External reference
6	Ground	Ground	Ground	Ground	Ground
23	Reserved	Reserved	Reserved	Digital I/O #6	Output 1 (0 to +5)
7	Reserved	Pos. Com. output	Null Position: input	Digital I/O #7	Output 2 (-10 to +10)
24	Ground	Ground	Ground	Ground	Ground
8	Ground	Ground	Ground	Ground	Ground
25	Aux. Enc. A+	Hall A+/Aux Enc A+	Compare / Full/Half Step: output	Digital I/O #8	Output 3 (0 to +5)
9	Aux. Enc. B	Hall B+/Aux Enc B+	Full/Half Current: output	Digital I/O #9	Output 4 (-10 to +10)
26	Ground	Ground	Ground	Ground	Ground
10	Ground	Ground	Ground	Ground	Ground
27	Pos. Capture + / Aux. Enc. Index+	Hall C+/ Pos Cap +	Aux. En Crs Home	Digital I/O #10	+12 VDC
11	Encoder Power	Encoder Power	Aux. Enc. Power	Digital I/O #11	-12 VDC
28	Ground	Ground	Ground	Ground	Ground
12	Limit + input	Limit + input	Limit + input	Ground	
29	Limit + sup./return	Limit + sup./return	Limit + sup/return	Digital I/O #12	
13	Limit Negative input	Limit Negative input	Limit - input	Ground	Ground
30	Limit - supply/return	Limit - supply/return	Limit – sup/return	Digital I/O #13	Input 1 (0 to +5)
14	Prim. Enc. A+	Prim. Enc. A+	Aux. Enc. A+	Ground	Ground
31	Prim. Enc. A-	Prim. Enc. A-	Aux. Enc. A-	Digital I/O #14	Input 2 (0 to +5
15	Prim. Enc. B+	Prim. Enc. B+	Aux. Enc. B+	Ground	Ground
32	Prim. Enc. B-	Prim. Enc. B-	Aux. Enc. B-	Digital I/O #15	Input 3 (0 to +5)
16	Prim. Enc. Index +	Prim. Enc. Index +	Pos. Cap. + / Aux. Enc. Index +	Ground	Ground
33	Prim. Enc. Index -	Prim. Enc. Index -	Pos. Cap. - / Aux. Enc. Index-	Digital I/O #16	Input 4 (0 to +5)
17	Ground	Ground	Ground	Ground	Ground
34	Ground	Ground	Ground	Ground	Ground

Example: DCX-BF3XX-H connections for a four axes system (single axis modules)
Here is an example of the typical connections for a four axes system (3 servo's and one stepper). A larger (more detailed) view of the interconnect drawing can be found earlier in this section.



BF3XX-H #1 – Contacts for axis #1 (a MC300-H installed in module location #1)

TS1	Signal
1	Axis 1 – Analog ret
18	Axis 1 – Command
2	Axis 1 – Comp. / Dir
19	Axis 1 – Com/Dir ret
3	Axis 1 – Amp En
20	Axis 1 – Amp En. ret
4	Axis 1 – Amp Fault
21	Axis 1 – Amp Flt ret
5	Axis 1 – Coarse Hm
22	Axis 1 – Crs Hm ret
6	Ground
23	
7	
24	Ground
8	Ground
25	
9	
26	Ground
10	Ground
27	Axis 1 – Pos Cap
11	Axis 1 – Enc Pwr
28	Ground
12	Axis 1 – Limit +
29	Axis 1 – Limit + ret
13	Axis 1 – Limit -
30	Axis 1 – Limit – ret
14	Axis 1 – Encoder A+
31	Axis 1 – Encoder A+
15	Axis 1 – Encoder B+
32	Axis 1 – Encoder B-
16	Axis 1 – Index +
33	Axis 1 – Index -
17	Ground
34	Ground

BF3XX-H #1 – Contacts for axis #3 (a MC300-H installed in module location #7)

TS2	Signal
1	Axis 3 – Analog ret
18	Axis 3 – Command
2	Axis 3 – Comp. / Dir
19	Axis 3 – Com/Dir ret
3	Axis 3 – Amp En
20	Axis 3 – Amp En. ret
4	Axis 3 – Amp Fault
21	Axis 3 – Amp Flt ret
5	Axis 3 – Coarse Hm
22	Axis 3 – Crs Hm ret
6	Ground
23	
7	
24	Ground
8	Ground
25	
9	
26	Ground
10	Ground
27	Axis 3 – Pos Cap
11	Axis 3 – Enc Pwr
28	Ground
12	Axis 3 – Limit +
29	Axis 3 – Limit + ret
13	Axis 3 – Limit -
30	Axis 3 – Limit – ret
14	Axis 3 – Encoder A+
31	Axis 3 – Encoder A+
15	Axis 3 – Encoder B+
32	Axis 3 – Encoder B-
16	Axis 3 – Index +
33	Axis 3 – Index -
17	Ground
34	Ground

BF3XX-H #2 – Contacts for axis #2 (a MC300-H installed in module location #2)

TS1	Signal
1	Axis 2 – Analog ret
18	Axis 2 – Command
2	Axis 2 – Comp. / Dir
19	Axis 2 – Com/Dir ret
3	Axis 2 – Amp En
20	Axis 2 – Amp En. ret
4	Axis 2 – Amp Fault
21	Axis 2 – Amp Flt ret
5	Axis 2 – Coarse Hm
22	Axis 2 – Crs Hm ret
6	Ground
23	
7	
24	Ground
8	Ground
25	
9	
26	Ground
10	Ground
27	Axis 2 – Pos Cap
11	Axis 2 – Enc Pwr
28	Ground
12	Axis 2 – Limit +
29	Axis 2 – Limit + ret
13	Axis 2 – Limit -
30	Axis 2 – Limit – ret
14	Axis 2 – Encoder A+
31	Axis 2 – Encoder A+
15	Axis 2 – Encoder B+
32	Axis 2 – Encoder B-
16	Axis 2 – Index +
33	Axis 2 – Index -
17	Ground
34	Ground

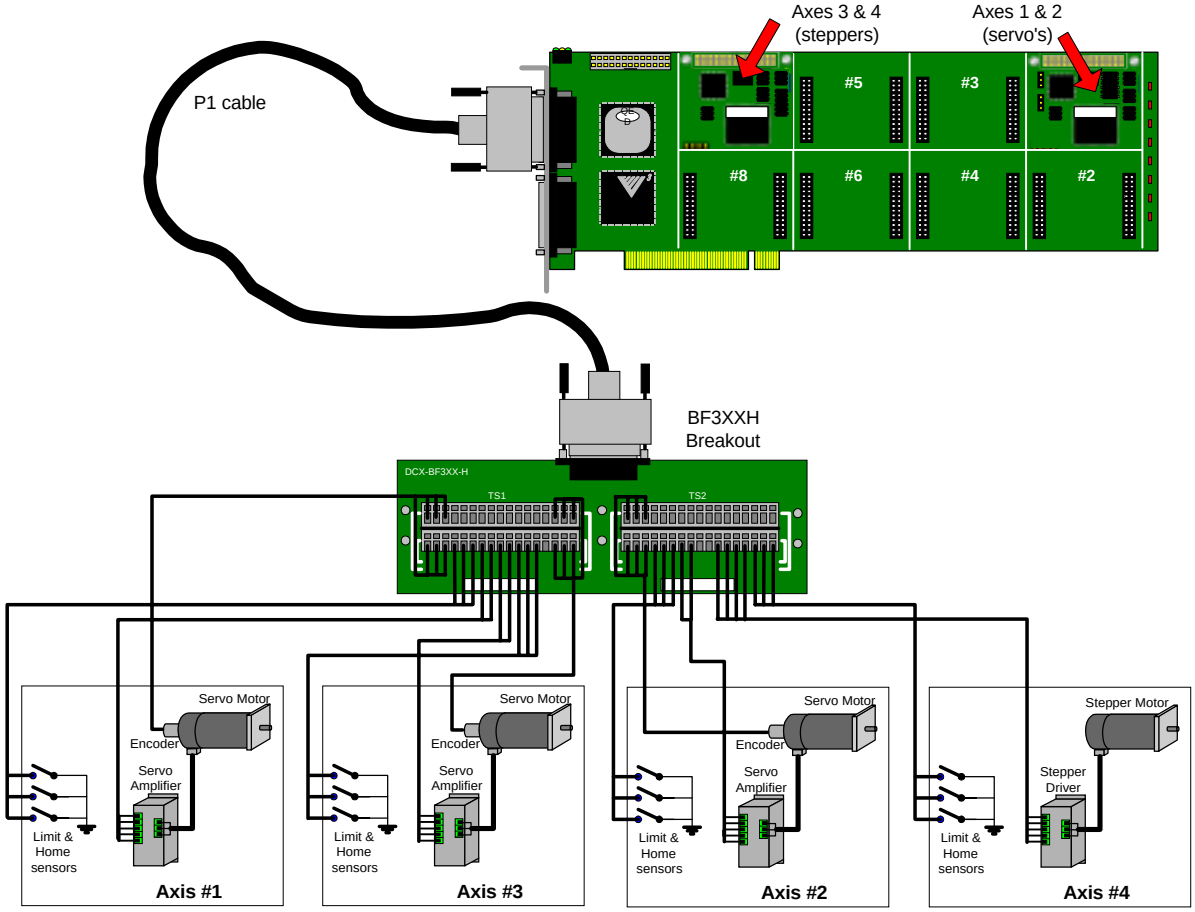
BF3XX-H #2 – Contacts for axis #4 (a MC360-H installed in module location #8)

TS2	Signal
1	Axis 4 – Step
18	Axis 4 – Ground
2	Axis 4 – Direction
19	Axis 4 – Ground
3	Axis 4 – Drive En
20	Axis 4 – Ground
4	Axis 4 – Drive Fault
21	Axis 4 – Ground
5	Axis 4 – Home
22	Axis 4 – Home ret
6	Ground
23	
7	
24	Ground
8	Ground
25	
9	Axis 4 – Full/Half cur
26	Ground
10	Ground
27	
11	
28	Ground
12	Axis 4 – Limit +
29	Axis 4 – Limit + ret
13	Axis 4 – Limit -
30	Axis 4 – Limit – ret
14	
31	
15	
32	
16	
33	
17	Ground
34	Ground

DCX-BF3XX-H signals pinout (when using dual axis modules)
The following table details the pinouts of –H Dual Axis DCX modules.

BF3XX-H TS1 or TS2	MC302-H	MC362-H
1	Axis 1 Encoder A+	Axis 1 Home
18	Axis 1 Encoder A-	Axis 1 Home sup/return
2	Axis 1 Encoder B+	Axis 1 Limit +
19	Axis 1 Encoder B-	Axis 1 Limit + sup/return
3	Axis 1 Encoder Index +	Axis 1 Limit -
20	Axis 1 Encoder Index -	Axis 1 Limit - sup/return
4	Axis 1 Coarse Home	Axis 1 Drive Fault
21	Axis 1 Encoder Power	Axis 1 Fault sup/return
5	Axis 1 Limit +	Axis 1 Drive Enable
22	Axis 1 inputs sup./return	Ground
6	Ground	Ground
23	Axis 1 Limit -	Axis 1 Pulse / CCW
7	Axis 1 Amp. Enable	Axis 1 Dir. / CW
24	Ground	Ground
8	Ground	Ground
25	Axis 1 Analog Command	Axis 1 Full Current
9	Axis 2 Analog Command	Axis 2 Full Current
26	Ground	Ground
10	Ground	Ground
27	Axis 2 Amp. Enable	Axis 2 Dir. / CW
11	Axis 2 Limit -	Axis 2 Pulse / CCW
28	Ground	Ground
12	Axis 2 Limit +	Axis 2 Drive Enable
29	Axis 2 inputs sup./return	Ground
13	Axis 2 Coarse Home	Axis 2 Drive Fault
30	Axis 2 Encoder Power	Axis 2 Fault sup/return
14	Axis 2 Encoder A+	Axis 2 Limit -
31	Axis 2 Encoder A-	Axis 2 Limit - sup/return
15	Axis 2 Encoder B+	Axis 2 Limit +
32	Axis 2 Encoder B-	Axis 2 Limit + sup/return
16	Axis 2 Encoder Index +	Axis 2 Home
33	Axis 2 Encoder Index -	Axis 2 Home sup/return
17	Ground	Ground
34	Ground	Ground

Example: DCX-BF3XX-H connections for a four axes system (dual axis modules)
Here is an example of the typical connections for a four axes system (2 servo's and 2 steppers) using dual axis modules. The DCX-MC302 (dual axis servo) is installed in module location #1 and the DCX-MC362 (dual axis stepper) is installed in module location #7.



Axis #1 (module #1)
BF3XX-H TS1 contacts
1-8 & 18–25.

TS1	Signal
1	Axis 1 Encoder A+
18	Axis 1 Encoder A-
2	Axis 1 Encoder B+
19	Axis 1 Encoder B-
3	Axis 1 Index +
20	Axis 1 Index -
4	Axis 1 Coarse Home
21	Axis 1 Encoder Pwr
5	Axis 1 Limit +
22	Axis 1 inputs return
6	Ground
23	Axis 1 Limit -
7	Axis 1 Amp. Enable
24	Ground
8	Ground
25	Axis 1 Command

Axis #2 (module #1)
BF3XX-H TS1 contacts
9-17 & 26-34.

TS2	Signal
9	Axis 2 Command
26	Ground
10	Ground
27	Axis 2 Amp. Enable
11	Axis 2 Limit -
28	Ground
12	Axis 2 Limit +
29	Axis 2 inputs return
13	Axis 2 Coarse Home
30	Axis 2 Encoder Pwr
14	Axis 2 Encoder A+
31	Axis 2 Encoder A-
15	Axis 2 Encoder B+
32	Axis 2 Encoder B-
16	Axis 2 Index +
33	Axis 2 Index -
17	Ground
34	Ground

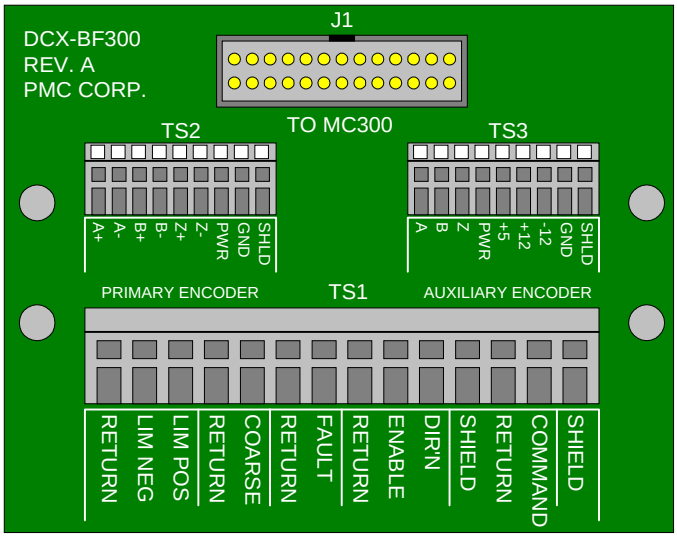
Axis #3 (module 2)
BF3XX-H TS2 contacts
1-8 & 18-25.

TS1	Signal
1	Axis 3 Encoder A+
18	Axis 3 Encoder A-
2	Axis 3 Encoder B+
19	Axis 3 Encoder B-
3	Axis 3 Index +
20	Axis 3 Index -
4	Axis 3 Coarse Home
21	Axis 3 Encoder Pwr
5	Axis 3 Limit +
22	Axis 3 inputs return
6	Ground
23	Axis 3 Limit -
7	Axis 3 Amp. Enable
24	Ground
8	Ground
	Axis 3 Command

Axis #4 (module 2)
BF3XX-H TS2 contacts
9-17 & 26-34.

TS2	Signal
9	Axis 2 Full Current
26	Ground
10	Ground
27	Axis 2 Dir. / CW
11	Axis 2 Pulse / CCW
28	Ground
12	Axis 2 Drive Enable
29	Ground
13	Axis 2 Drive Fault
30	Axis 2 Fault return
14	Axis 2 Limit -
31	Axis 2 Limit - return
15	Axis 2 Limit +
32	Axis 2 Limit + return
16	Axis 2 Home
33	Axis 2 Home return
17	Ground
34	Ground

DCX-BF300-R Servo Module Breakout Assembly



DCX-BF300-R to DCX-MC300-R Connections:

Terminal strip TS1

Pin	Description
1	Crs Home & Limits return
2	Limit -
3	Limit +
4	Crs Home & Limits return
5	Coarse Home
6	Amp Fault supply/return
7	Amplifier Fault
8	Amp Enable/Dir. return
9	Amplifier Enable
10	Direction
11	Shield
12	Analog Ground
13	Analog Command output
14	Shield

Terminal strip TS2

Pin	Description
1	Prim. Encoder Phase A+
2	Prim. Encoder Phase A-
3	Prim. Encoder Phase B+
4	Prim. Encoder Phase B-
5	Prim. Encoder Index+
6	Prim. Encoder Index-
7	Encoder Power
8	Ground
9	Shield

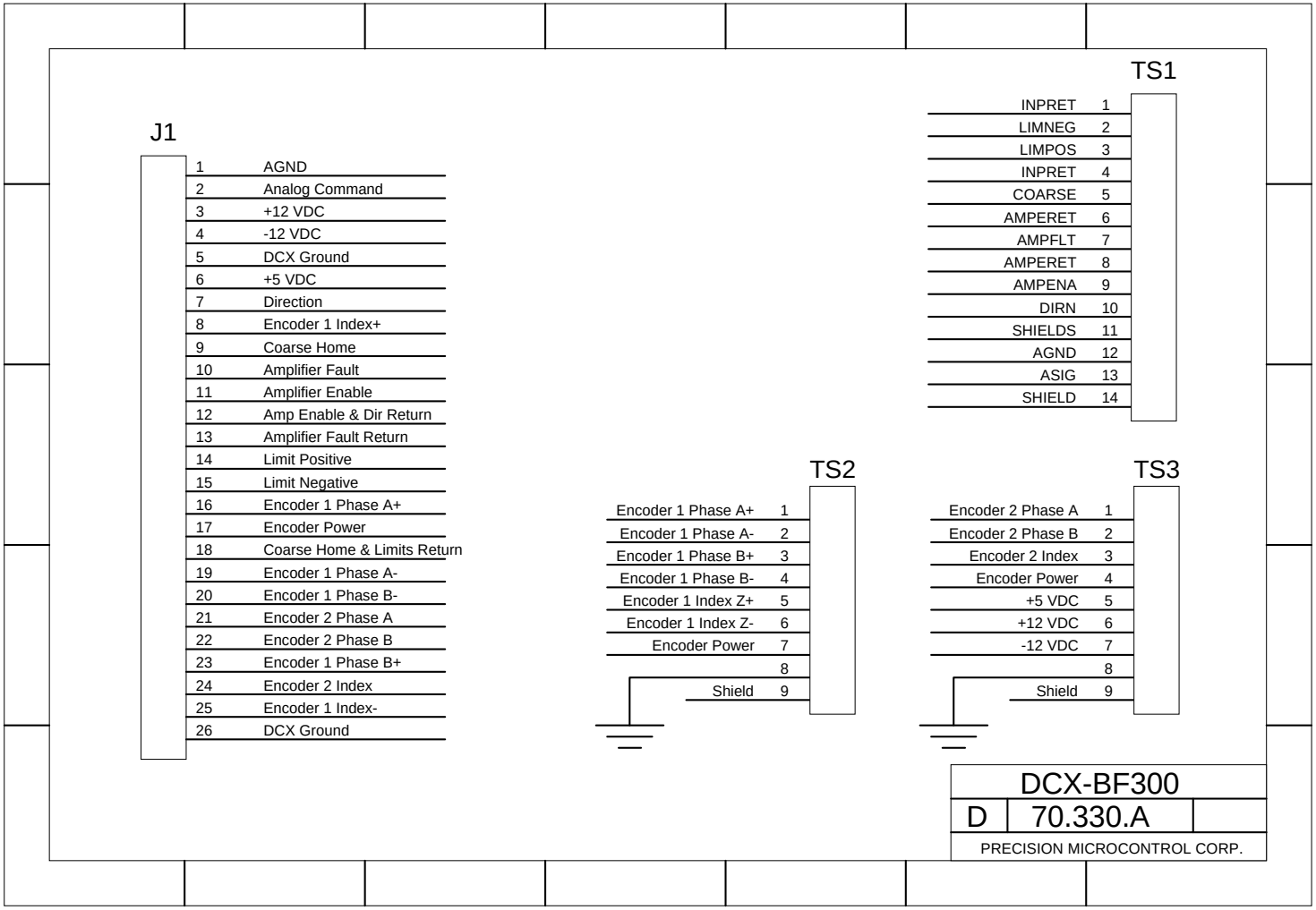
Terminal strip TS3

Pin	Description
1	Aux. Encoder Phase A+
2	Aux. Encoder Phase B+
3	Aux. Encoder Index Z+
4	Encoder Power
5	+5 VDC
6	+12 VDC
7	-12 VDC
8	Ground
9	Shield

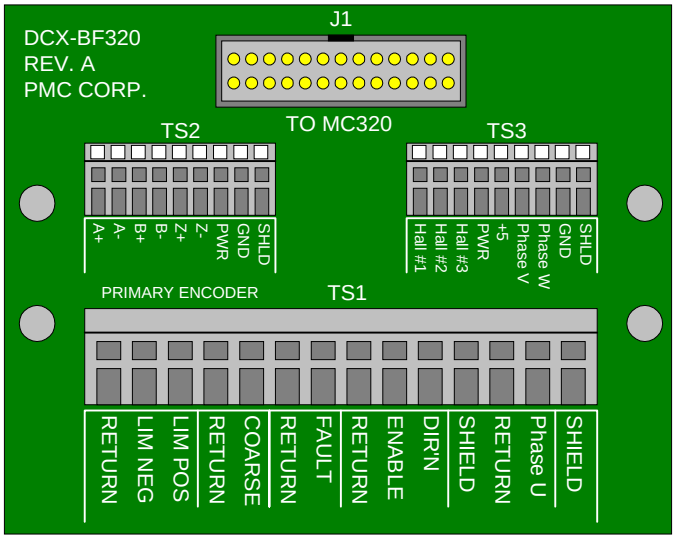
DCX-BF300-R to DCX-MC300 Connections (continued):

Connector J1: From MC300

Pin	*Description*
1	Analog Ground
2	Analog Command output
3	+12 VDC
4	-12 VDC
5	Ground
6	+5 VDC
7	Direction
8	Primary Encoder Index +
9	Coarse Home
10	Amplifier Fault
11	Amplifier Enable
12	Amp Enable/Dir. return
13	Amp Fault supply/return
14	Limit +
15	Limit -
16	Prim. Encoder Phase A+
17	Encoder Power
18	Crs Home & Limits return
19	Prim. Encoder Phase A-
20	Prim. Encoder Phase B-
21	Aux. Encoder Phase A
22	Aux. Encoder Phase B
23	Prim. Encoder Phase B+
24	Aux. Encoder Index+
25	Prim. Encoder Index-
26	Ground


J1
1 AGND
2 Analog Command
3 +12 VDC
4 -12 VDC
5 DCX Ground
6 +5 VDC
7 Direction
8 Encoder 1 Index+
9 Coarse Home
10 Amplifier Fault
11 Amplifier Enable
12 Amp Enable & Dir Return
13 Amplifier Fault Return
14 Limit Positive
15 Limit Negative
16 Encoder 1 Phase A+
17 Encoder Power
18 Coarse Home & Limits Return
19 Encoder 1 Phase A-
20 Encoder 1 Phase B-
21 Encoder 2 Phase A
22 Encoder 2 Phase B
23 Encoder 1 Phase B+
24 Encoder 2 Index
25 Encoder 1 Index-
26 DCX Ground
TS1
INPRET 1
LIMNEG 2
LIMPOS 3
INPRET 4
COARSE 5
AMPERET 6
AMPFLT 7
AMPERET 8
AMPENA 9
DIRN 10
SHIELDS 11
AGND 12
ASIG 13
SHIELD 14
TS2
Encoder 1 Phase A+ 1
Encoder 1 Phase A- 2
Encoder 1 Phase B+ 3
Encoder 1 Phase B- 4
Encoder 1 Index Z+ 5
Encoder 1 Index Z- 6
Encoder Power 7
8
Shield 9
TS3
Encoder 2 Phase A 1
Encoder 2 Phase B 2
Encoder 2 Index 3
Encoder Power 4
+5 VDC 5
+12 VDC 6
-12 VDC 7
8
Shield 9
DCX-BF300
D 70.330.A
PRECISION MICROCONTROL CORP.

DCX-BF320-R Servo Module Breakout Assembly



DCX-BF320-R to DCX-MC320-R Connections:

Terminal strip TS1

Pin	Description
1	Crs Home & Limits return
2	Limit -
3	Limit +
4	Crs Home & Limits return
5	Coarse Home
6	Amp Fault supply/return
7	Amplifier Fault
8	Amp Enable/Dir. return
9	Amplifier Enable
10	Phase W
11	Phase V
12	Phase U
13	Analog Ground
14	Shield

Terminal strip TS2

Pin	Description
1	Prim. Encoder Phase A+
2	Prim. Encoder Phase A-
3	Prim. Encoder Phase B+
4	Prim. Encoder Phase B-
5	Prim. Encoder Index+
6	Prim. Encoder Index-
7	Encoder Power
8	Ground
9	Shield

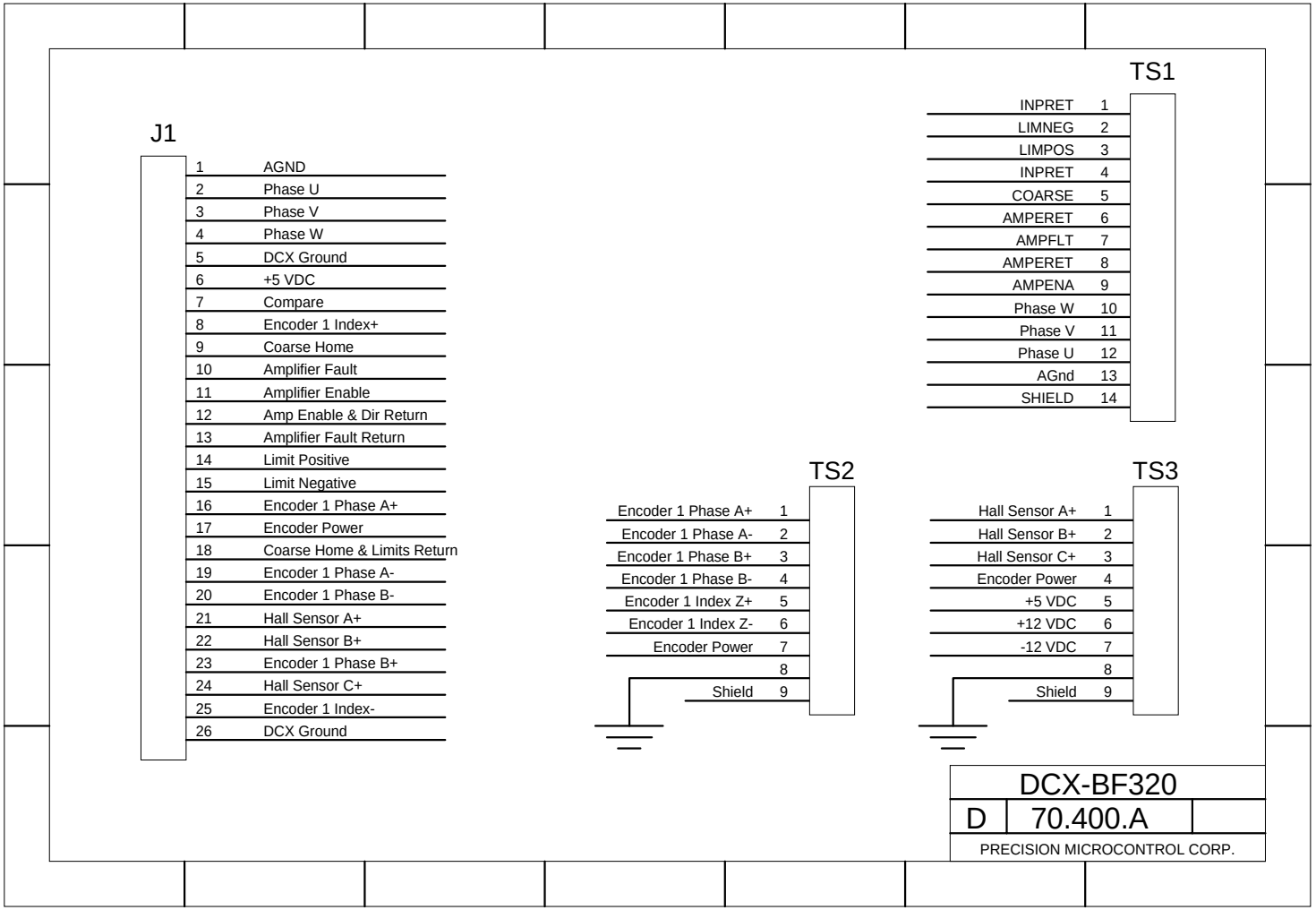
Terminal strip TS3

Pin	Description
1	Hall Sensor A+
2	Hall Sensor B+
3	Hall Sensor C+
4	Encoder Power
5	+5 VDC
6	+12 VDC
7	-12 VDC
8	Ground
9	Shield

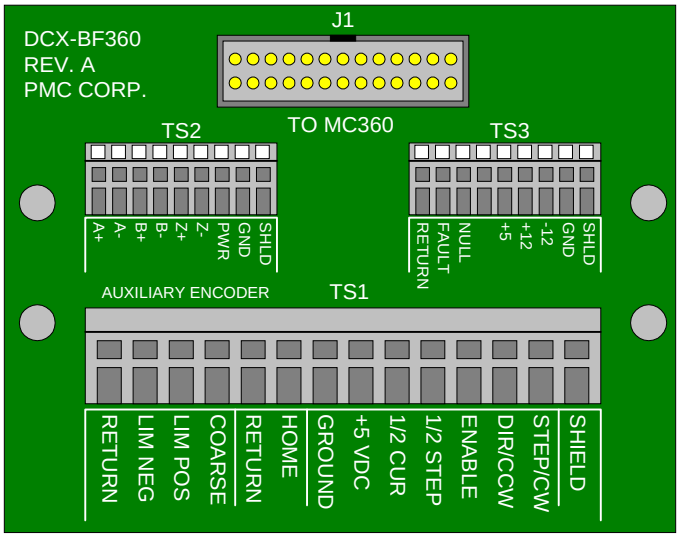
DCX-BF320-R to DCX-MC320 Connections (continued):

Connector J1: From MC300

Pin	*Description*
1	Analog Ground
2	Phase U
3	Phase V
4	Phase W
5	Ground
6	+5 VDC
7	Compare
8	Primary Encoder Index +
9	Coarse Home
10	Amplifier Fault
11	Amplifier Enable
12	Amp Enable/Dir. return
13	Amp Fault supply/return
14	Limit +
15	Limit -
16	Prim. Encoder Phase A+
17	Encoder Power
18	Crs Home & Limits return
19	Prim. Encoder Phase A-
20	Prim. Encoder Phase B-
21	Hall Sensor A+
22	Hall Sensor B+
23	Prim. Encoder Phase B+
24	Hall Sensor C+
25	Prim. Encoder Index-
26	Ground


J1
1 AGND
2 Phase U
3 Phase V
4 Phase W
5 DCX Ground
6 +5 VDC
7 Compare
8 Encoder 1 Index+
9 Coarse Home
10 Amplifier Fault
11 Amplifier Enable
12 Amp Enable & Dir Return
13 Amplifier Fault Return
14 Limit Positive
15 Limit Negative
16 Encoder 1 Phase A+
17 Encoder Power
18 Coarse Home & Limits Return
19 Encoder 1 Phase A-
20 Encoder 1 Phase B-
21 Hall Sensor A+
22 Hall Sensor B+
23 Encoder 1 Phase B+
24 Hall Sensor C+
25 Encoder 1 Index-
26 DCX Ground
TS1
INPRET 1
LIMNEG 2
LIMPOS 3
INPRET 4
COARSE 5
AMPERET 6
AMPFLT 7
AMPERET 8
AMPENA 9
Phase W 10
Phase V 11
Phase U 12
AGnd 13
SHIELD 14
TS2
Encoder 1 Phase A+ 1
Encoder 1 Phase A- 2
Encoder 1 Phase B+ 3
Encoder 1 Phase B- 4
Encoder 1 Index Z+ 5
Encoder 1 Index Z- 6
Encoder Power 7
8
Shield 9
TS3
Hall Sensor A+ 1
Hall Sensor B+ 2
Hall Sensor C+ 3
Encoder Power 4
+5 VDC 5
+12 VDC 6
-12 VDC 7
8
Shield 9
DCX-BF320
D 70.400.A
PRECISION MICROCONTROL CORP.

DCX-BF360-R Stepper Module Breakout Assembly



DCX-BF360-R to DCX-MC360-R Connections:

Terminal strip TS1

Pin	Description
1	Crs Home & Limits return
2	Limit -
3	Limit +
4	Aux Encoder Crs Home
5	Home return
6	Home
7	Ground
8	+5 VDC
9	Full/Half Current
10	Full/Half Step
11	Drive Enable
12	Direction
13	Step
14	Shield

Terminal strip TS2

Pin	Description
1	Aux. Encoder Phase A+
2	Aux. Encoder Phase A-
3	Aux. Encoder Phase B+
4	Aux. Encoder Phase B-
5	Aux. Encoder Index+
6	Aux. Encoder Index-
7	Encoder Power
8	Ground
9	Shield

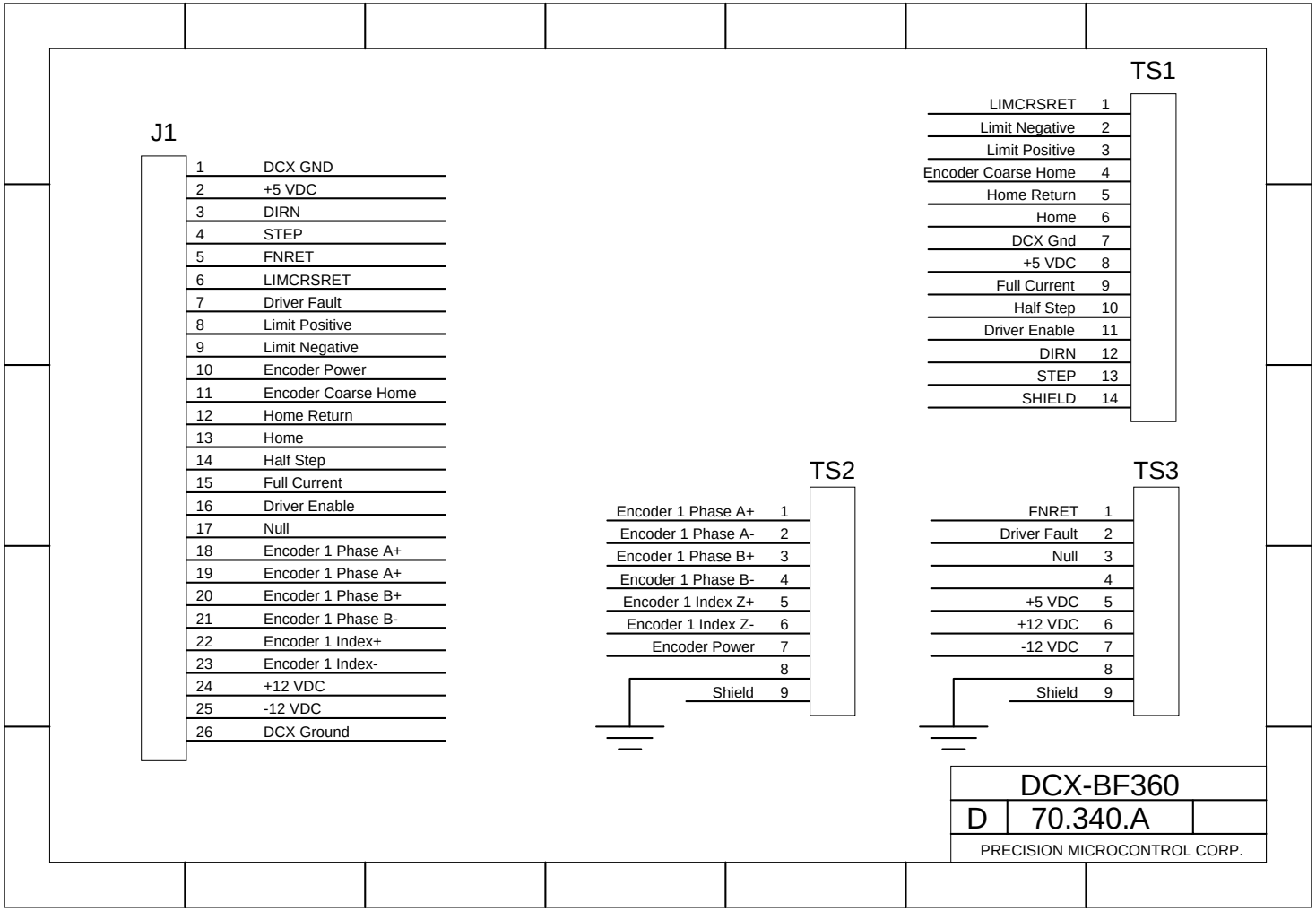
Terminal strip TS3

Pin	Description
1	FNRET
2	Driver Fault
3	Null
4	
5	+5 VDC
6	+12 VDC
7	-12 VDC
8	Ground
9	Shield

DCX-BF300-R to DCX-MC360 Connections (continued):

Connector J1: From MC360

Pin	*Description*
1	Ground
2	+5 VDC
3	Direction
4	Pulse / CCW Pulse
5	FNRET
6	LIMCRSRET
7	Drive Fault
8	Limit Positive
9	Limit Negative
10	Auxiliary Encoder Power
11	Aux. Enc Coarse Home
12	HOMRET
13	Home
14	Full/Half Step
15	Full/Half Current
16	Driver Enable
17	Null Position
18	Aux Encoder Phase A+
19	Aux Encoder Phase A-
20	Aux Encoder Phase B+
21	Aux Encoder Phase B-
22	Auxiliary Encoder Index+
23	Auxiliary Encoder Index-
24	+12 VDC
25	-12 VDC
26	Ground


J1
1 DCX GND
2 +5 VDC
3 DIRN
4 STEP
5 FNRET
6 LIMCRSRET
7 Driver Fault
8 Limit Positive
9 Limit Negative
10 Encoder Power
11 Encoder Coarse Home
12 Home Return
13 Home
14 Half Step
15 Full Current
16 Driver Enable
17 Null
18 Encoder 1 Phase A+
19 Encoder 1 Phase A+
20 Encoder 1 Phase B+
21 Encoder 1 Phase B-
22 Encoder 1 Index+
23 Encoder 1 Index-
24 +12 VDC
25 -12 VDC
26 DCX Ground
TS1
LIMCRSRET 1
Limit Negative 2
Limit Positive 3
Encoder Coarse Home 4
Home Return 5
Home 6
DCX Gnd 7
+5 VDC 8
Full Current 9
Half Step 10
Driver Enable 11
DIRN 12
STEP 13
SHIELD 14
TS2
Encoder 1 Phase A+ 1
Encoder 1 Phase A- 2
Encoder 1 Phase B+ 3
Encoder 1 Phase B- 4
Encoder 1 Index Z+ 5
Encoder 1 Index Z- 6
Encoder Power 7
8
Shield 9
TS3
FNRET 1
Driver Fault 2
Null 3
4
+5 VDC 5
+12 VDC 6
-12 VDC 7
8
Shield 9
DCX-BF360
D 70.340.A
PRECISION MICROCONTROL CORP.

Chapter Contents

Chapter

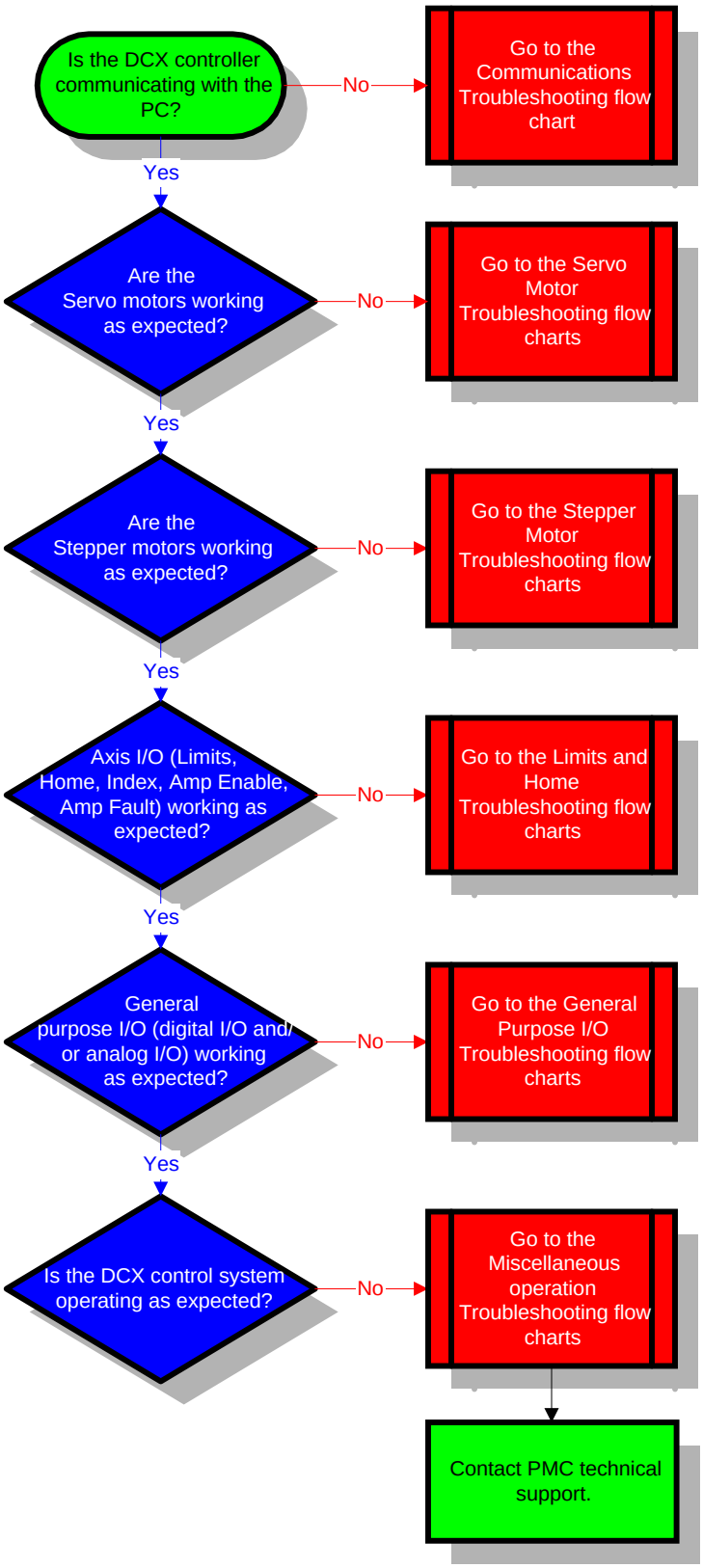
10

Troubleshooting

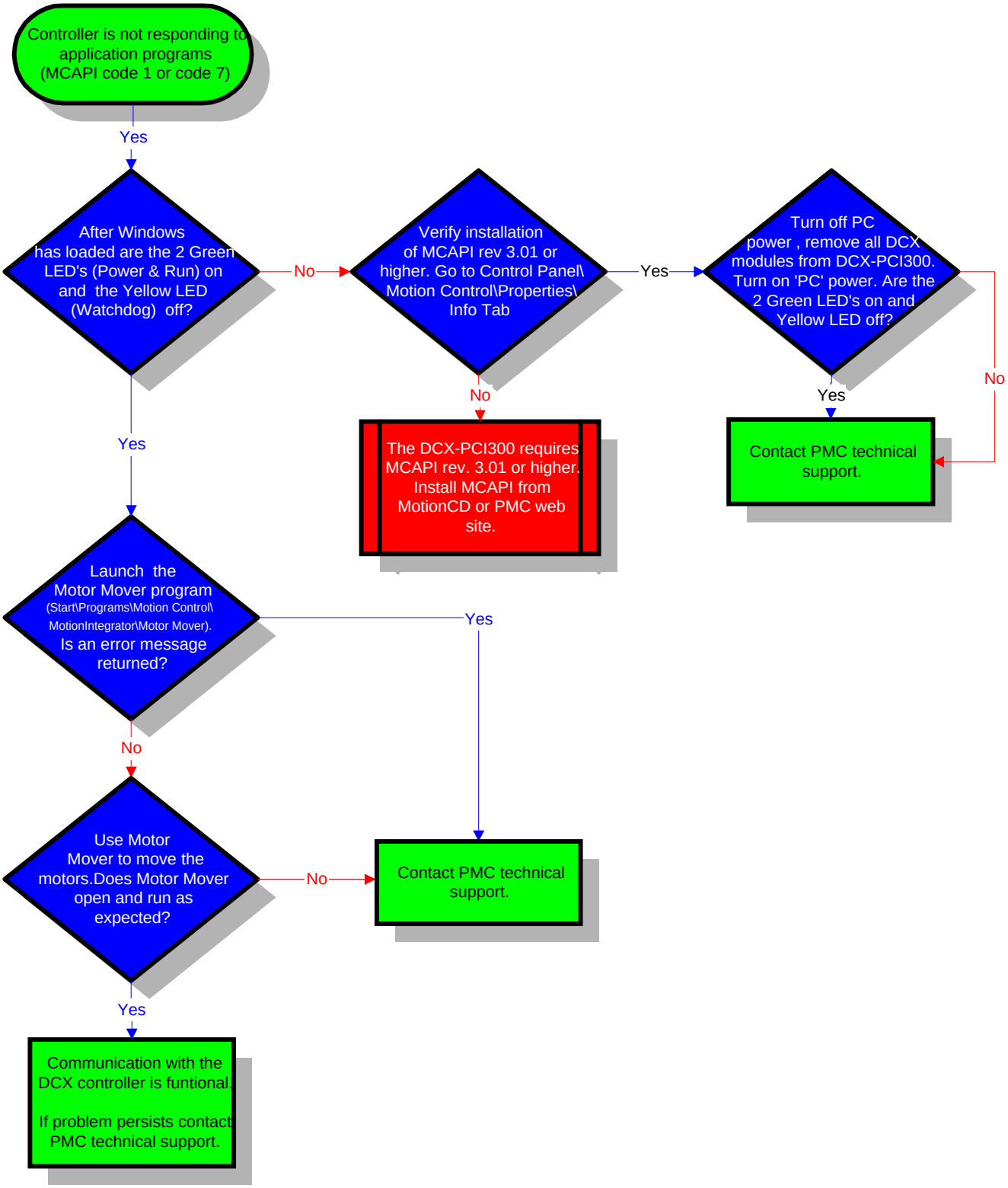
On the following pages you will find troubleshooting flow charts to assist the with diagnosis of motion control system failures.

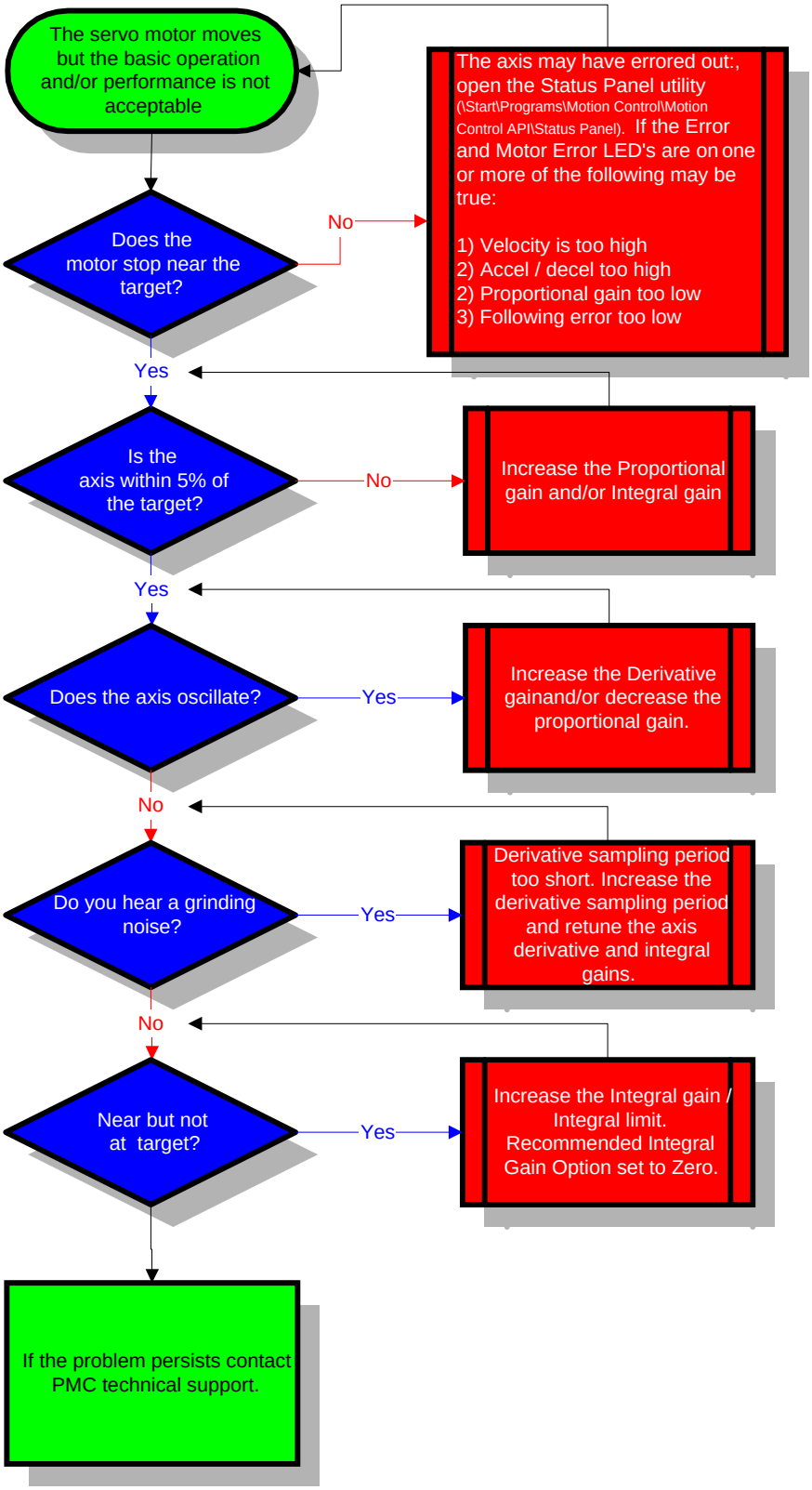
The steps described in these flow charts will direct the user to PMC programs (Motion Integrator, Motor Mover, CWdemo, etc...) and utilities (Servo Tuning, WinControl) that are used to diagnose and resolve system operation.

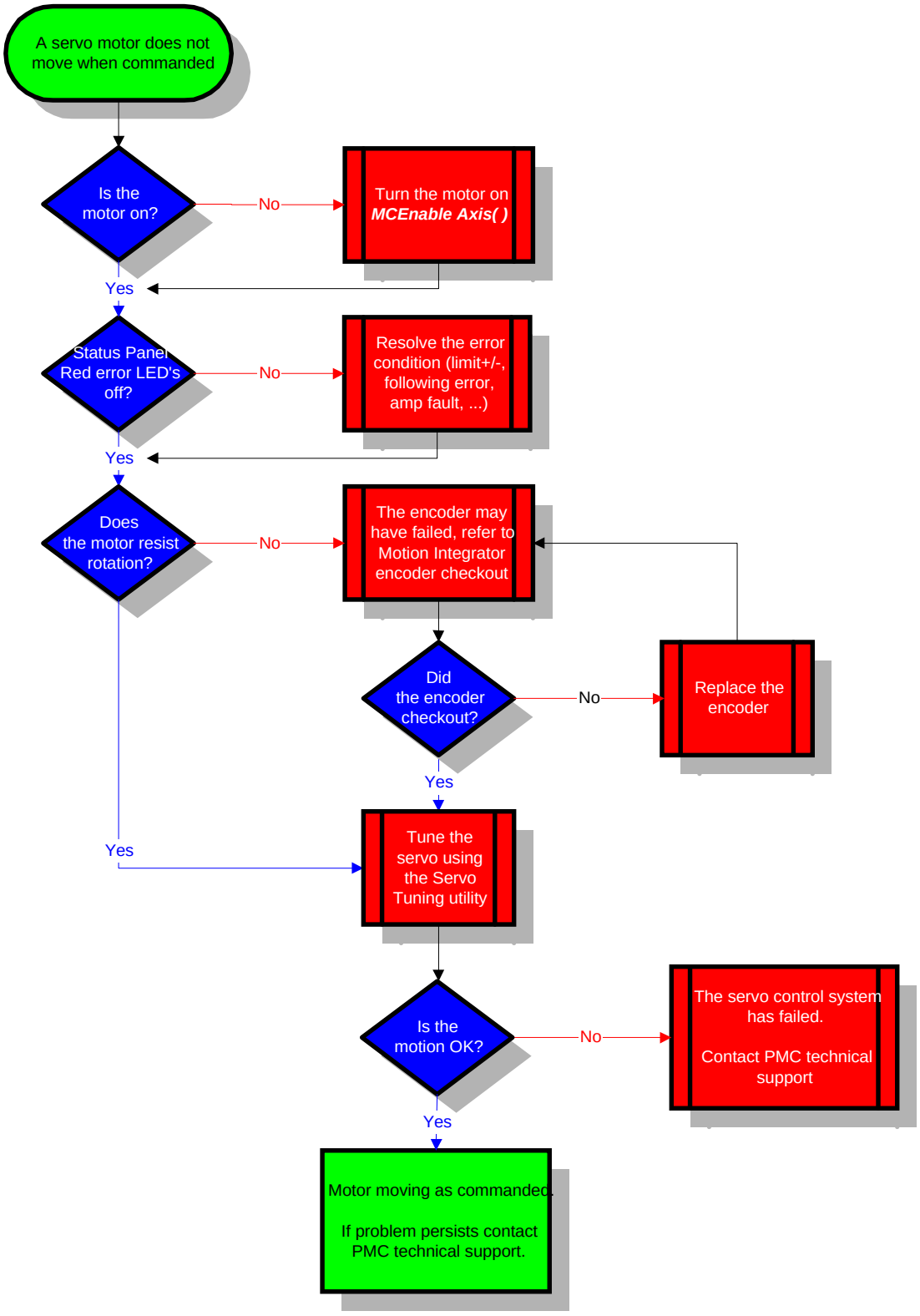
DCX System Troubleshooting

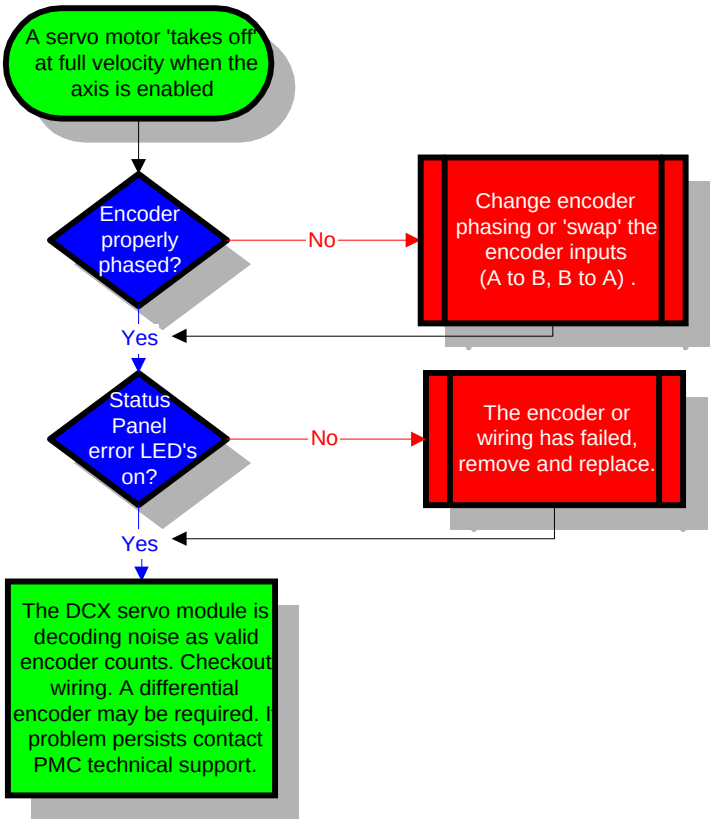


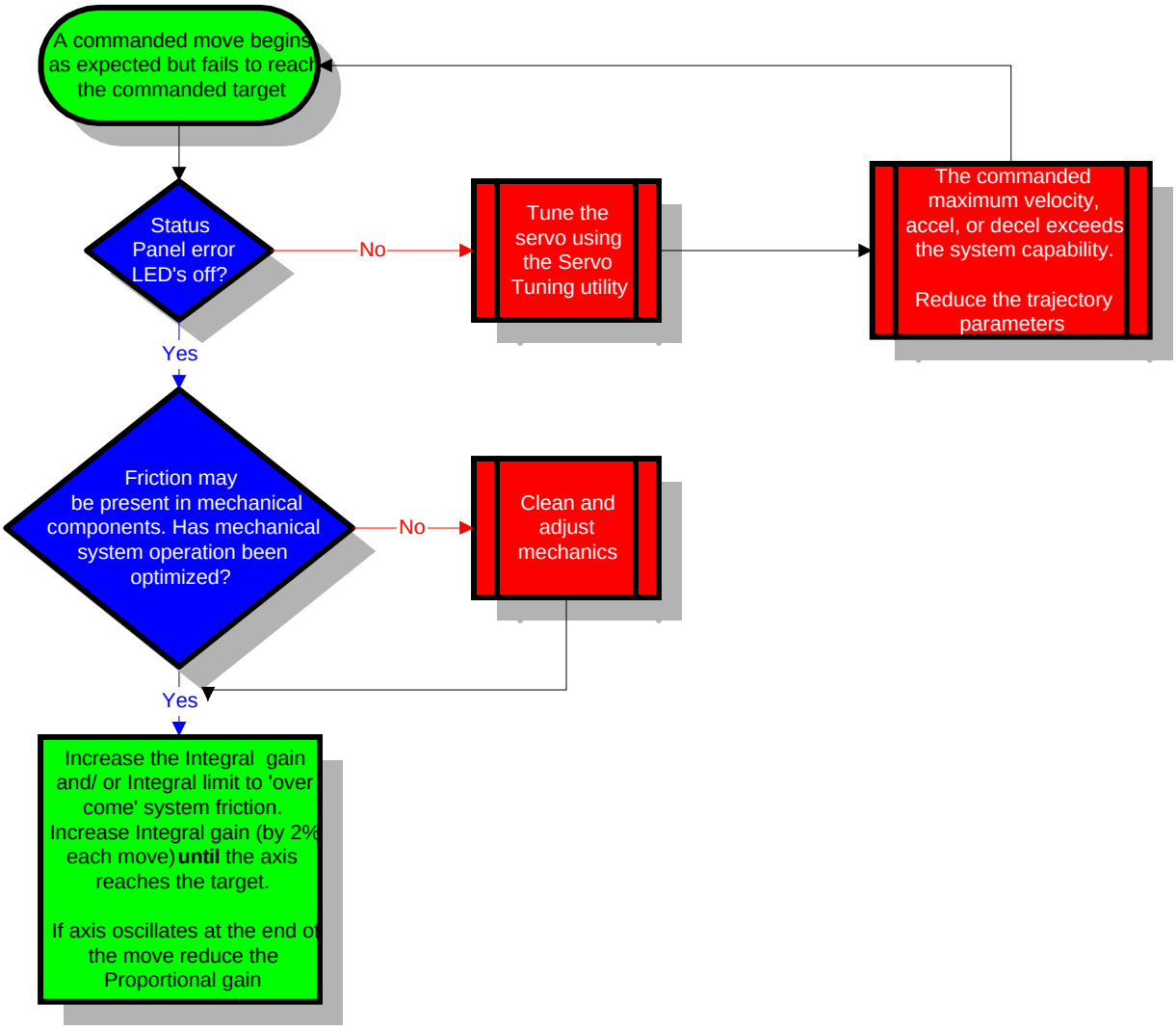
Communications Troubleshooting



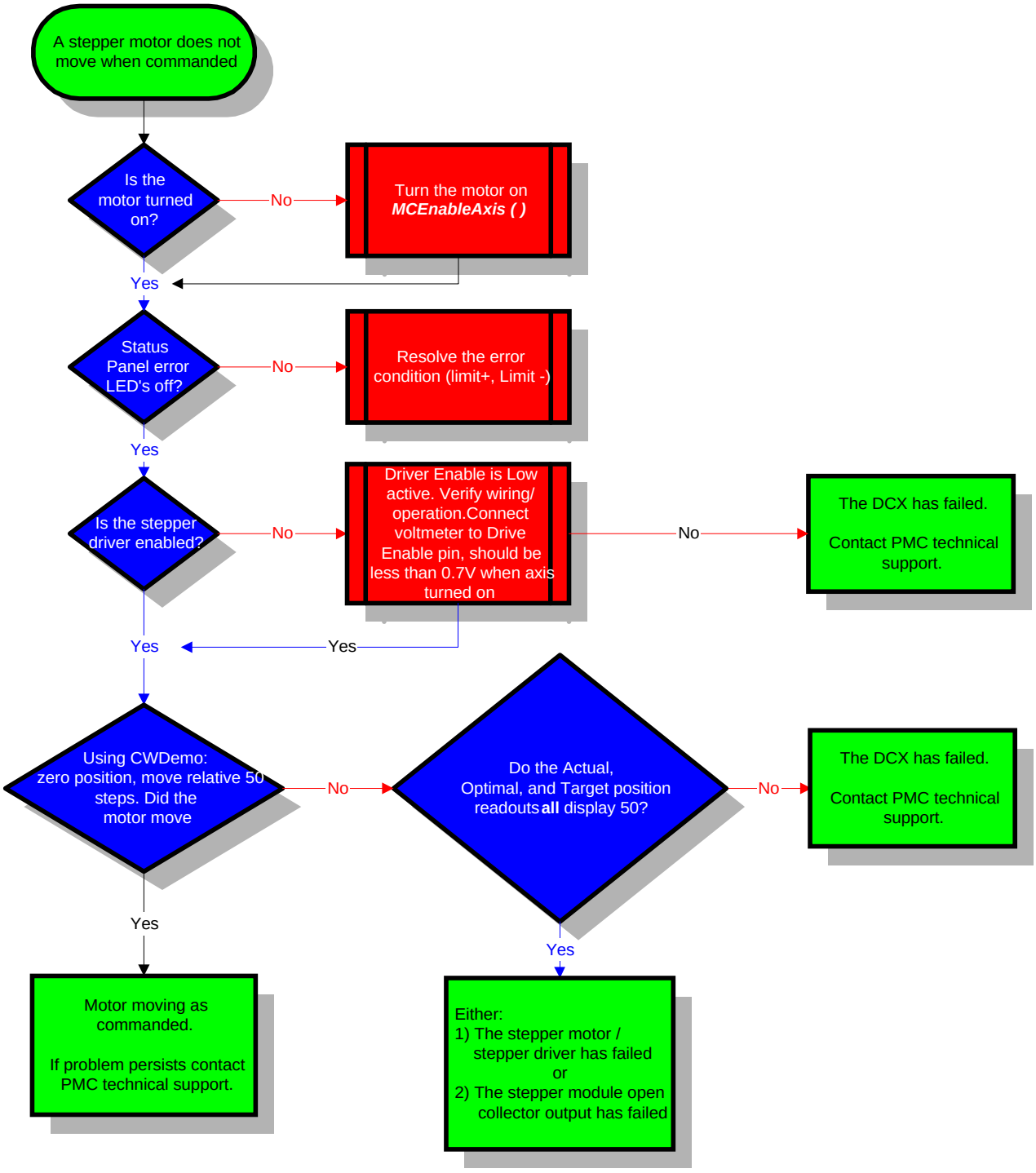

The servo motor moves but the basic operation and/or performance is not acceptable
Does the motor stop near the target?
No
The axis may have errored out:, open the Status Panel utility (\Start\Programs\Motion Control\Motion Control API\Status Panel). If the Error and Motor Error LED's are on one or more of the following may be true:
1) Velocity is too high
2) Accel / decel too high
2) Proportional gain too low
3) Following error too low
Yes
Is the axis within 5% of the target?
No
Increase the Proportional gain and/or Integral gain
Yes
Does the axis oscillate?
Yes
Increase the Derivative gainand/or decrease the proportional gain.
No
Do you hear a grinding noise?
Yes
Derivative sampling period too short. Increase the derivative sampling period and retune the axis derivative and integral gains.
No
Near but not at target?
Yes
Increase the Integral gain / Integral limit. Recommended Integral Gain Option set to Zero.
If the problem persists contact PMC technical support.


A servo motor does not move when commanded
Is the motor on?
No
Turn the motor on MCEnable Axis( )
Yes
Status Panel Red error LED's off?
No
Resolve the error condition (limit+/-, following error, amp fault, ...)
Yes
Does the motor resist rotation?
No
The encoder may have failed, refer to Motion Integrator encoder checkout
Did the encoder checkout?
No
Replace the encoder
Yes
Yes
Tune the servo using the Servo Tuning utility
Is the motion OK?
No
The servo control system has failed.
Contact PMC technical support
Yes
Motor moving as commanded.
If problem persists contact PMC technical support.

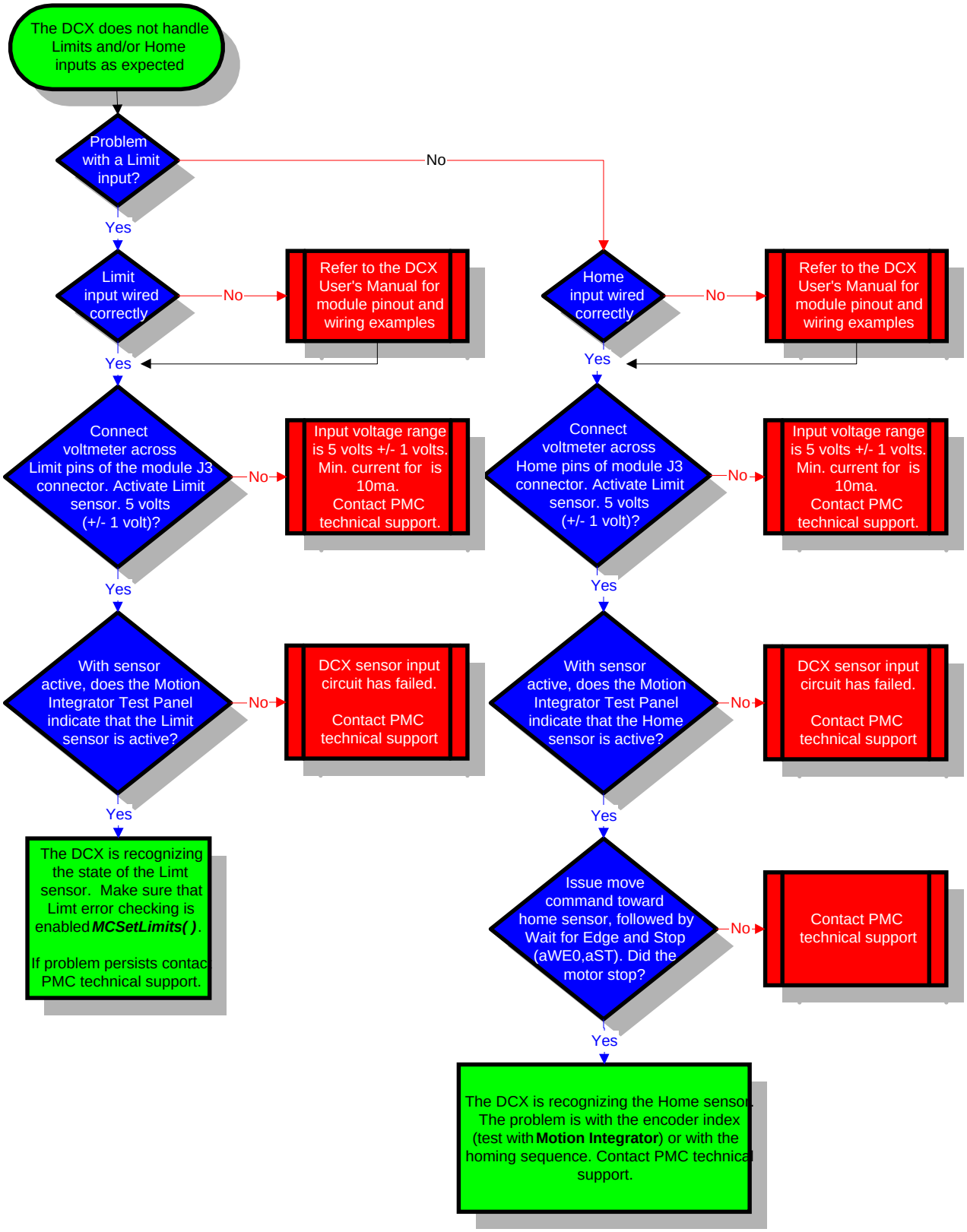

A servo motor 'takes off' at full velocity when the axis is enabled
Encoder properly phased?
No
Change encoder phasing or 'swap' the encoder inputs (A to B, B to A) .
Yes
Status Panel error LED's on?
No
The encoder or wiring has failed, remove and replace.
Yes
The DCX servo module is decoding noise as valid encoder counts. Checkout wiring. A differential encoder may be required. I problem persists contact PMC technical support.


A commanded move begins as expected but fails to reach the commanded target
Status Panel error LED's off?
No
Tune the servo using the Servo Tuning utility
The commanded maximum velocity, accel, or decel exceeds the system capability.
Reduce the trajectory parameters
Yes
Friction may be present in mechanical components. Has mechanical system operation been optimized?
No
Clean and adjust mechanics
Yes
Increase the Integral gain and/ or Integral limit to 'over come' system friction.
Increase Integral gain (by 2% each move) until the axis reaches the target.
If axis oscillates at the end of the move reduce the Proportional gain

Troubleshooting - Open Loop Stepper Motion chart #1



Troubleshooting - Limits and Home



Chapter Contents

Chapter

11

Controller Error Codes

Both the MCAPI and the Motion Control Command Language (MCCL) provide error code and interface status information to the user.

MCAPI Error Codes

MCAPI defined error messages are listed numerically in the table below. Where possible corrective action is included in the description column. Please note that many MCAPI function descriptions also include information regarding errors that are specific to that function.

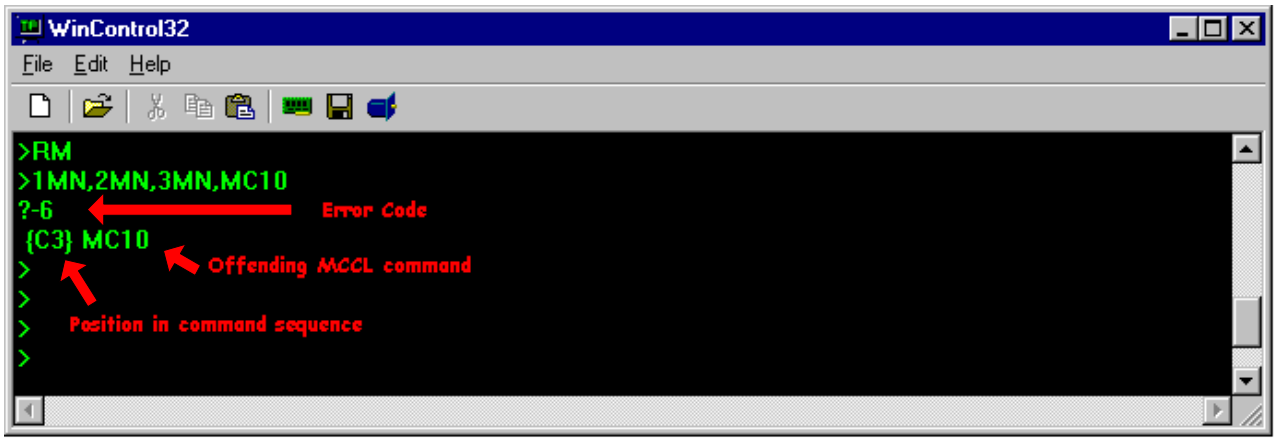
Error	Constant	Description
0	MCERR_NOERROR	No error has occurred
1	MCERR_NO_CONTROLLER	No controller assigned at this ID. Use MCSETUP to configure a controller.
2	MCERR_OUT_OF_HANDLES	MCAPI driver out of handles. The driver is limited to 32 open handles. Applications that do not call MCClose( ) when they exit may leave handles unavailable, forcing a reboot.
3	MCERR_OPEN_EXCLUSIVE	Cannot open - another application has the controller opened for exclusive use
4	MCERR_MODE_UNAVAIL	Controller already open in different mode. Some controller types can only be open in one mode (ASCII or binary) at a time
5	MCERR_UNSUPPORTED_MODE	Controller doesn't support this mode for MCOpen( ) - i.e. ASCII or binary
6	MCERR_INIT_DRIVER	Couldn't initialize the device driver
7	MCERR_NOT_PRESENT	Controller hardware not present
8	MCERR_ALLOC_MEM	Memory allocation error. This is an internal memory allocation problem with the DLL, contact Technical Support for assistance
9	MCERR_WINDOWSERROR	A windows function returned an error - use GetLastError () under WIN32 for details
10		reserved
11	MCERR_NOTSUPPORTED	Controller doesn't support this feature
12	MCERR_OBSOLETE	Function is obsolete
13	MCERR_AXIS_TYPE	Axis type doesn't support this feature
14	MCERR_CONTROLLER	Invalid controller handle
15	MCERR_WINDOW	Invalid window handle
16	MCERR_AXIS_NUMBER	Axis number out of range
17	MCERR_ALL_AXES	Cannot use MC_ALL_AXES for this function
18	MCERR_RANGE	Parameter was out of range
19	MCERR_CONSTANT	Constant value inappropriate
20	MCERR_UNKNOWN_REPLY	Unexpected or unknown reply
21	MCERR_NO_REPLY	Controller failed to reply
22	MCERR_REPLY_SIZE	Reply size incorrect
23	MCERR_REPLY_AXIS	Wrong axis for reply
24	MCERR_REPLY_COMMAND	Reply is for different command
25	MCERR_TIMEOUT	Controller failed to respond
26	MCERR_BLOCK_MODE	Block mode error. Caused by calling MCBlockEnd( ) without first calling MCBlockBegin( ) to begin the block
27	MCERR_COMM_PORT	Communications port (RS232) driver reported an error
28	MCERR_CANCEL	User canceled action (such as when an MCDLG dialog box is dismissed with the CANCEL button
29	MCERR_NOT_INITIALIZED	Feature was not correctly initialized before being enabled or used

MCCL Error Codes

When executing MCCL (Motion Control Command Language) command sequences the command interpreter will report the following error code when appropriate:

Description	Error code
No error	0
Unrecognized command	1
Bad command format	2
I/O error	3
Command string to long	4
Command Parameter Error	-1
Command Code Invalid	-2
Negative Repeat Count	-3
Macro Define Command Not First	-4
Macro Number Out of Range	-5
Macro Doesn't Exist	-6
Command Canceled by User	-7
Contour Path Command Not First	-8
Contour Path Command Parameter Invalid	-9
Contour Path Command Doesn't Specify an AXIS	-10
Axis error (over travel error, max. following error exceeded	-13
No axis specified	-14
Axis not assigned	-15
Axis already assigned	-16
Axis duplicate assigned	-17
Insufficient memory	-18
Unrecognized variable name	-19
Invalid background task ID	-20
Command not supported	-21

Many error code reports will not only include the error code but also the offending command. In the following example the Reset Macro command was issued. This command clears all macro's from memory. The next command sequence turns on 3 motors and then calls macro 10. The command MC10 is a valid command but with no macros in memory error code –6 is displayed.



Chapter Contents

Chapter

12

Printing a PDF Document

Introduction to PDF
PDF stands for Portable Document Format. It is the defacto standard for transporting electronic documents. PDF files are based on the PostScript language imaging model. This enables sharp, color-precise printing on almost all printers.

Printing a complete PDF document
It is **not recommended** that large PDF documents be printed on personal computer printers. The 'wear and tear' incurred by these units, coupled with the difficulties of two sided printing, typically resulting in degraded performance of the printer and a whole lot of wasted paper. PMC recommends that PDF document be printer by a full service print shop that uses digital (computer controlled) copy systems with paper collating/sorting capability.

Printing selected pages of a PDF document
While viewing a PDF document with Adobe Reader (or Adobe Acrobat), any page or range of pages can be printed by a personal computer printer by:

Selecting the printer icon on the tool bar
Selecting **Print** from the Adobe **File** menu

Paper
The selection of the paper type to be used for printing a PDF document should be based on the target market for the document. For a user's manual with extensive graphics that is printed on both sides of a page the minimum recommended paper type is 24 pound. A heavier paper stock (26 – 30 pound) will reduce the 'bleed through' inherent with printed graphics. Typically the front and back cover pages are printed on heavy paper stock (50 to 60 pound).

Binding
Unlike the binding of a book or catalog, a user's manual distributed in as a PDF file will typically use

'comb' or 'coil' binding. This service is provided by most full service print shops. Coil binding is suitable for documents with no more than 100 pieces of paper (24 pound). Comb binding is acceptable for documents with as many as 300 pieces of paper (24 pound). Most print shops stock a wide variety of 'combs'. The print shop can recommend the appropriate 'comb' based on the number of pages.

Pricing
The final cost for printing and binding a PDF document is based on:

- Quantity per print run
- Number of pages
- Paper type

The price range for printing and binding a PDF document similar to this user manual will be $15 to $30 (printed in Black & White) in quantities of 1 to 10 pieces.

Obtaining a Word 2000 version of this user manual
This user document was written using Microsoft's Word 2000. Qualified OEM's, Distributors, and Value Added Reps (VAR's) can obtain a copy of this document for

- Editing
- Customization
- Language translation.

Please contact Precision MicroControl to obtain a Word 2000 version of this document.

Chapter 13

Glossary

Accuracy - A measure of the difference between the expected position and actual position of a motion system.

Actuator - Device that creates mechanical motion by converting energy to mechanical energy.

Axis Phasing - An axis is properly phased when a commanded move in the positive direction causes the encoder decode circuitry of the controller to increment the reported position of the axis.

Back EMF - The voltage generated when a permanent magnet motor is rotated. This voltage is proportional to motor speed and is present regardless of whether the motor windings are energized or de-energized.

Closed Loop - A broadly applied term, relating to any system in which the output is measured and compared to the input. The output is then adjusted to reach the desired condition. In motion control, the term typically describes a system utilizing a velocity and/or position transducer to generate correction signals in relation to desired parameters.

Command Set – Defines the operations that can be executed by the motion controller

Commutation - The action of applying currents or voltages to the proper motor phases in order to produce optimum motor torque.

Critical Damping - A system is critically damped when the response to a step change in desired velocity or position is achieved in the minimum possible time with little or no overshoot.

DAC - The digital-to-analog converter (DAC) is the electrical interface between the motion controller and the motor amplifier. It converts the digital voltage value computed by the motion controller into an analog voltage. The more DAC bits, the finer the analog voltage resolution. DACs are available in three common sizes: 8, 12, and 16 bit. The bit count partitions the total peak-to-peak output voltage

swing into 256, 4096, or 65536 DAC steps, respectively.

Dead Band - A range of input signals for which there is no system response.

Driver - Electronics that convert step and direction inputs to high power currents and voltages to drive a step motor. The step motor driver is analogous to the servo motor amplifier.

Dual Loop Servo – A servo system that combines a velocity mode amplifier/tachometer with a position loop controller/encoder. It is recommended that the encoder not be directly coupled to the motor. The linear scale encoder should be mounted on the external mechanics, as closely coupled as possible to the ‘end effector’

Duty Cycle - For a repetitive cycle, the ratio of on time to total time:

Efficiency - The ratio of power output to power input.

Encoder - A type of feedback device that converts mechanical motion into electrical signals to indicate actuator position or velocity.

End Effector – The point of focus of a motion system. The tools with which a motion system will work. Example: The leading edge of the knife is the *end effector* of a three axis (XYZ) system designed to cut patterns from vinyl.

Feed Forward - Defines a specific voltage level output from a motion controller, which in turn commands a velocity mode amplifier to rotate the motor at a specific velocity.

Following Error - The difference between the calculated desired trajectory position and the actual position.

Friction - A resistance to motion caused by contacting surfaces. Friction can be constant with varying speed (Coulomb friction) or proportional to speed (viscous friction).

Holding Torque - Sometimes called static torque, holding torque specifies the maximum external torque that can be applied to a stopped, energized motor without causing the rotor to rotate continuously.

Inertia - The measure of an object's resistance to a change in its current velocity. Inertia is a function of the object's mass and shape.

Kd - K is a generally accepted variable used to represent gain, an arbitrary multiplier, or a constant. The lower case ‘d’ designates derivative gain.

Ki - K is a generally accepted variable used to represent gain, an arbitrary multiplier, or a constant. The lower case ‘i’ designates integral gain.

Kp - K is a generally accepted variable used to represent gain, an arbitrary multiplier, or a constant. The lower case ‘p’ designates proportional gain.

Limits - Motion system sensors (hard limits) or user programmable range (soft limits) that alert the motion controller that the physical end of travel is being approached and that motion should stop.

MCAPI - The Motion Control Application Programming Interface - this is the programming interface used by Windows programmers to control PMC's family of motion control cards.

MCCL - Motion Control Command Language - this is the command language used to program PMC's family of motion control cards.

Micro-Stepping - Stepper drive systems have a fixed number of electromechanical detents or steps. Micro stepping is an electronic technique to break each detent or step into smaller parts. This results in higher positional resolution and smoother operation.

Open Loop – A control system in which the control output is not referenced or scaled to an external feedback.

Position Error - see following error.

Position Move - Unlike a velocity move, a position move includes a predefined stopping position. The trajectory generator will determine when to begin deceleration in order to ensure the actual stopping point is at the desired target position.

PWM - Pulse Width Modulation is a method of controlling the average current in a motor’s phase windings by varying the duty cycle of transistor switches.

Repeatability - The degree to which the positioning accuracy for a given move performed repetitively can be duplicated.

Resonance - A condition resulting from energizing a motor at a frequency at or close to the motor's natural frequency.

Resolution - The smallest positioning increment that can be achieved.

Resolver - A type of feedback device that converts mechanical position into an electrical signal. A resolver is a variable transformer that divides the impressed AC signal into sine and cosine output signals. The amplitude of these signals represents the absolute position of the resolver shaft.

Servo - An automatic system in which the output is constantly compared with the input through some form of feedback. The error (or difference) between the two quantities can be used to bring about the desired amount of control.

Servo tuning – the process in which the appropriate gain values for the PID filter are determined

Slew - That portion of a move made at constant, non-zero velocity.

Step Response - An instantaneous command to a new position. Typically used for tuning a closed

loop system, ramping (velocity, acceleration, and deceleration) is not applied nor calculated for the move.

Tachometer - A device attached to a moving shaft that generates a voltage signal directly proportional to rotational speed.

Torque -

Velocity Mode Amplifier – An amplifier that requires a tachometer to provide the feedback used to close the velocity loop within the amplifier.

Velocity Move - A move where no final stopping position is given to the motion controller. When a start command is issued the motor will rotate indefinitely until it is commanded to stop.

Appendix Contents

- Power Supply Requirements
- Default Settings

Appendix

Power Supply Requirements

Part Number	*+5 VDC*	*+12 VDC*	*-12 VDC*	*Unit*
DCX-PCI300	0.9	-----	-----	A
DCX-MC300	.4	.01	.01	A
DCX-MC320	.4	.01	.01	A
DCX-MC360	.4	-----	-----	A
DCX-MC400	.25	-----	-----	A
DCX-MC500	.1	*	*	A

* Current depends on output loading

Default Settings

Description	Setting
Programmed Velocity	10,000
Programmed Acceleration	10,000
Programmed Deceleration	10,000
Minimum Velocity	1,000
Current Velocity	0
Velocity Gain	0
Acceleration Gain	0
Deceleration Gain	0
Velocity Override	1
Torque Limit	10
Proportional Gain	.2
Derivative Gain	.1
Integral Gain	.01
Integration Limit	50
Maximum Following Error	1024
Motion Limits	disabled
Low Limit of Movement	0
High Limit of Movement	0
Servo Loop Rate	MS
Stepper Pulse Range	HS
Position Count	0
Optimal Count	0
Index Count	0
Auxiliary Status	0
Position	0
Target	0
Optimal Position	0
Breakpoint Position	0
Position Dead band	0
User Scale	1
User Zero	0
User Offset	0
User Rate Conversion	1
User Output Constant	1
Sampling Frequency	0
Slave Ratio	1

Index

D

E

F

G

H

I

J

L

M

S

T

U

V

W

Precision MicroControl Corporation
2075-N Corte del Nogal
Carlsbad, CA 92009-1415 USA

Tel: (760) 930-0101
Fax: (760) 930-0222

www.pmccorp.com

Information: info@pmccorp.com
Technical Support: support@pmccorp.com