DCX-PCI300

Modular Multi-Axis Motion Control System

Motion Control Command Language (MCCL) Reference

Revision 2.1

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

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

Revision	Date	Description
1.0 Pre	3/12/2001	Preliminary release
	4/6/2001	Added preliminary pinouts for –H (high density) MC3XX modules
	4/25/2001	Preliminary PCI300 qualification edits
	5/14/2001	Added - Initializing and Restoring Controller Configuration description
1.0	5/15/2001	Initial release
	5/22/2001	BF320 pinouts
	5/22/2001	BF3XX-H pinouts and high density connectors module mapping
	5/24/2001	IIR filter description
	5/24/2001	Integral gain option description
1.1	5/24/2001	Updated to match firmware revision 1.1a
1.2	7/26/2001	Updated to match firmware revision 1.2a
	7/26/2001	Added support for Motion Integrator
	8/14/2001	Miscellaneous edits
2.0	1/25/2002	Updated to match firmware 2.0a
	1/18/2002	Miscellaneous edits
	1/18/2002	Added support for MCCL program single stepping
	1/30/2002	Updated Auxiliary Encoder homing example
2.1	2/13/2002	Updated to match firmware 2.1a
	2/26/2002	Miscellaneous edits

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

A 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 handle missile control systems, 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 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
- 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
- Reporting the status of an axis including: current position of an axis, target of a move, current following error, and indicating when a move is complete
- Electronic gearing of axes
- Homing an axis

The command set for the DCX-PCI300 is called **MCCL** (**M**otion **C**ontrol **C**ommand **L**anguage) and it supports well over 200 operations. A complete listing and description of the DCX-PCI300 command set can be found in **Chapters 7 – 15** of this manual.

The MCCL command set is made up of two character alphanumeric mnemonics built with two key characters from the description of the operation (eg. "MR" for ***Move Relative***). When the DCX receives a MCCL command (followed by a carriage return) it will be executed immediately.

```
;Example

1MR1000   <Enter>                         Move axis #1 1000 encoder counts
```

Beginning with Chapter 7 of this manual you will find the MCCL command descriptions. The commands are divided into the following categories:

- Setup commands (set velocity, set proportional gain) – Chapter 7
- Mode commands(position mode, contour mode) – Chapter 8
- Motion commands(move relative, find index) – Chapter 9
- Reporting commands(tell position, tell axis status) – Chapter 10
- I/O commands(turn on digital output, configure input as high true) – Chapter 11
- Macro and Multi-tasking commands(execute a macro, terminate a background task – Chapter 12
- Register commands (copy accumulator to user register, multiply accumulator by) – Chapter 13
- Sequence commands (wait for trajectory complete, if accumulator = execute next command) – Chapter 14
- Miscellaneous commands (reset the controller, parameters in decimal mode) – Chapter 15

Who should use MCCL commands?

The target market for the DCX-PCI300 is Windows PC based multi-axis applications requiring proprietary software written in C++/C. In these type of environments the application program issues function calls to PMC's motion control function library (MCAPI). The MCAPI converts the function call into a MCCL command. The device driver then passes the MCCL command to the motion control card. For additional information on the MCAPI please refer to the **DCX-PCI300 MCAPI Reference Manual** and the **DCX-PCI300 User's Manual**.

For these type of applications the machine designer will rarely (if ever) need to use the MCCL commands because PMC provides powerful tools for:

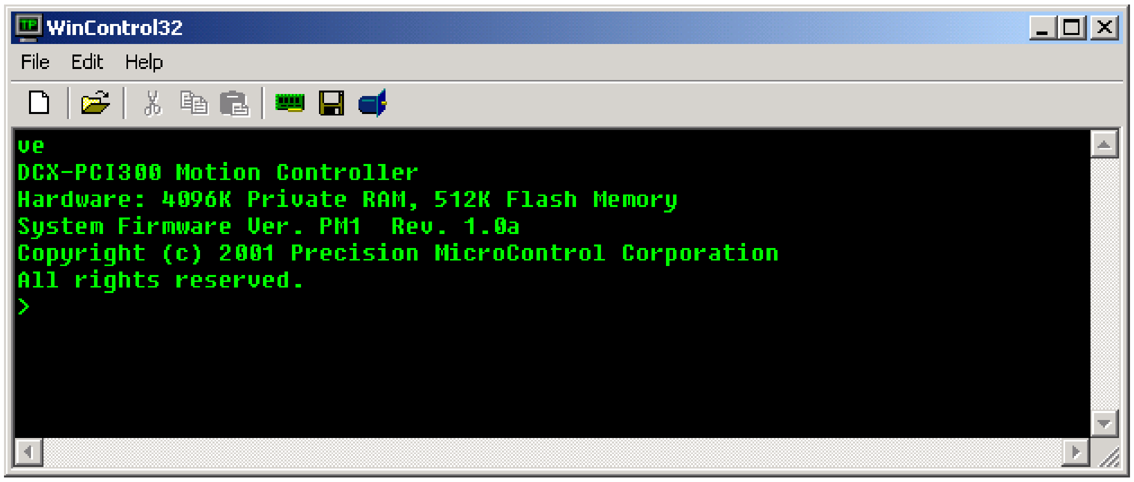
- Integrating the DCX controller and external system components (**Motion Integrator**)
- Tuning the servo (**Servo Tuning program**)
- Exercising the motion control system (**Motor Mover**)

But there are times when the capability to issue MCCL commands directly to the DCX controller can be invaluable. Some examples are:

- When homing routines are programmed as MCCL command sequences (versus MCAPI functions called from an application program) The PC is free to handle other tasks while the motion system is being initialized.
- Identifying whether unexpected system behavior is the fault of hardware, software, or external devices (amplifiers, sensors, feedback devices, etc.)
- When a non programmer needs to exercise system functions in combinations not supported by PMC tools
- Some applications require the motion controller to execute subroutines locally, completely separate from the PC. The DCX-PCI300 is capable of executing MCCL command sequences as background tasks.

WinControl – the MCCL command interface utility

One of the tools included with the MCAPI is the **WinControl** utility. This tool allows the user to issue MCCL commands directly to the DCX-PCI300.



```
ve
DCX-PCI300 Motion Controller
Hardware: 4096K Private RAM, 512K Flash Memory
System Firmware Ver. PM1  Rev. 1.0a
Copyright (c) 2001 Precision MicroControl Corporation
All rights reserved.
>
```

Figure 1: The WinControl MCCL command interface utility

To open the WinControl utility from the Windows Start menu select Programs\Motion Control\Motion Control API\ WinControl.

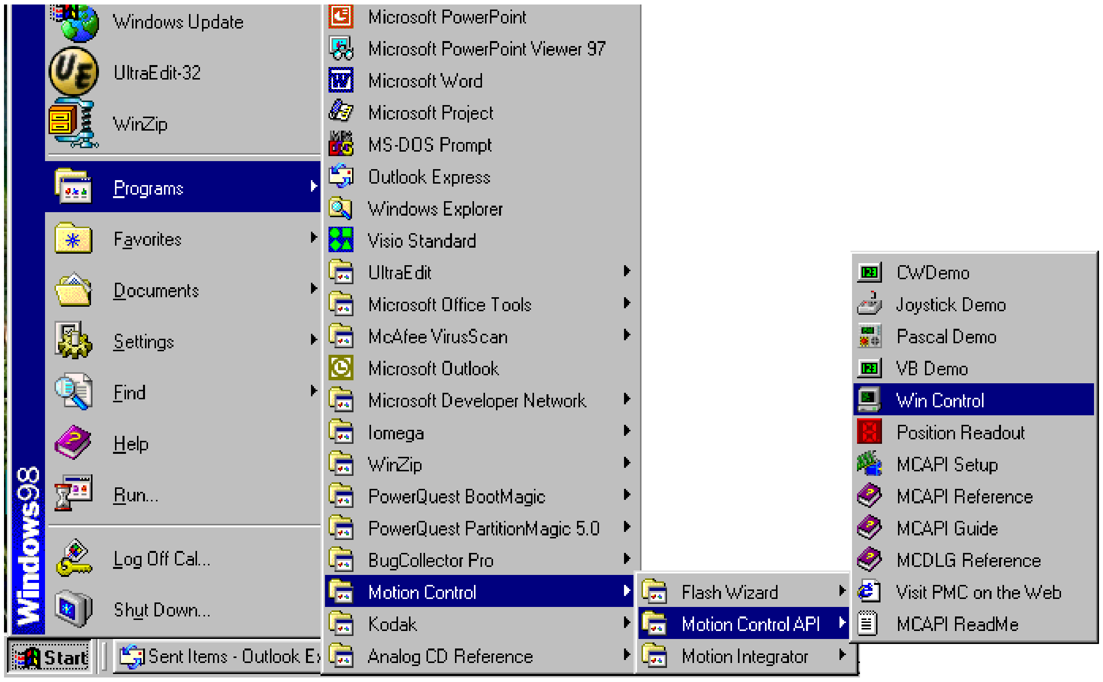
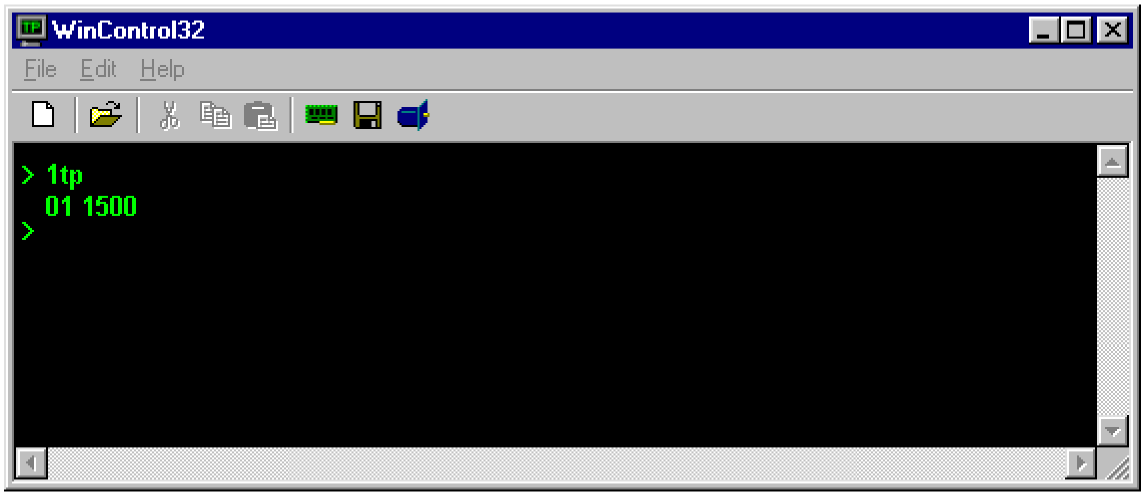


Figure 2: Open WinControl (\Start\Program Files\Motion Control\Motion Control API\WinCtl32)

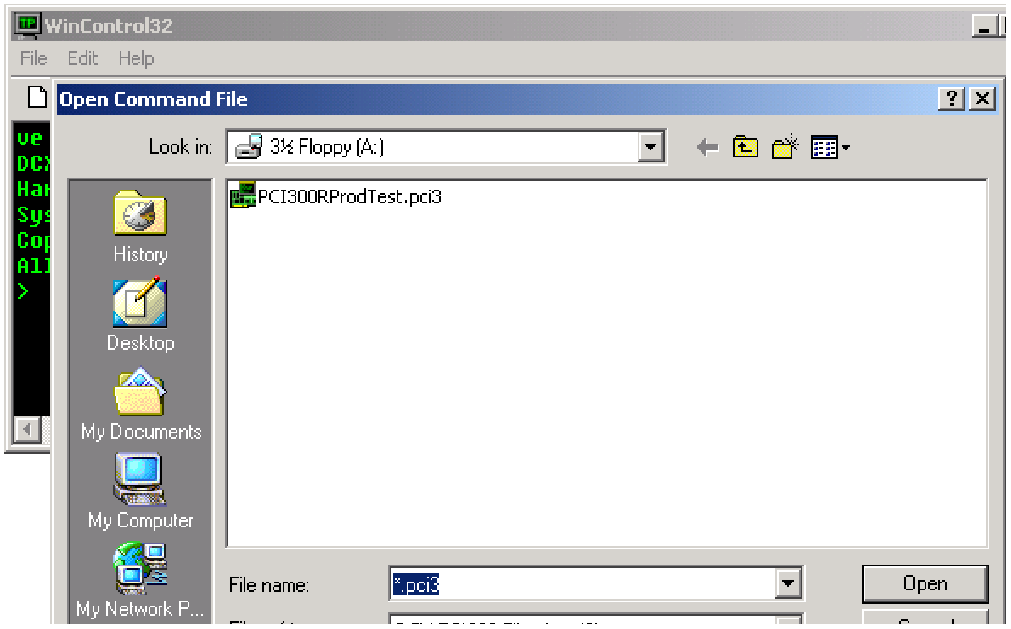
From the PC keyboard, MCCL commands can be entered one character at a time and executed when the user presses the **Enter** key. The user can issue a single MCCL command or multiple MCCL commands separated by commas. In figure #3 the current position of axis #1 is reported by issuing the MCCL command **T**ell **P**osition (***1*****TP** <Enter>).



```
> 1tp
  01 1500
>
```

Figure 3: Report the current position of axis #1 by issuing the MCCL command Tell Position (1TP)

By selecting File and Open from the WinControl menu the user can download a text file containing MCCL commands to the controller.



PCI300RProdTest.pci3

File name: *.pci3

Figure 4: Download a MCCL text file to the DCX-PCI300

Chapter Contents

- MCCL command syntax
- Simple moves with MCCL commands

Chapter

2

DCX Operation Basics

MCCL Command Syntax

All DCX MCCL commands are a 2 character mnemonic. The characters that make the mnemonic are selected from the command description so that the command has a direct correlation to the operation to be performed. For example, the MCCL command that is used to move an axis to an absolute position is:

MA	(**M**ove **A**bsolute).

Any MCCL command that references an axis is preceded by a numeric axis specifier. To issue a move absolute to axis #1:

1**MA**	(axis #1 **M**ove **A**bsolute)

If the numeric axis specifier equals 0 (0MA) then the command will be executed for all axes.

Most DCX commands will also include a parameter value following the mnemonic. To move axis #1 to absolute position 10.25:

1**MA**10.25	(axis #1 **M**ove **A**bsolute to position 10.25)

If no parameter value is given with the command the default value of 0 will be used.

In the command descriptions in this manual, the axis specifier is shown as an italicized ‘*a*’, and the parameter as an italicized ‘*n*’.

A typical command description is shown below:

Move Absolute

MCCL command: *a*MA*n* where: *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: MR, PM

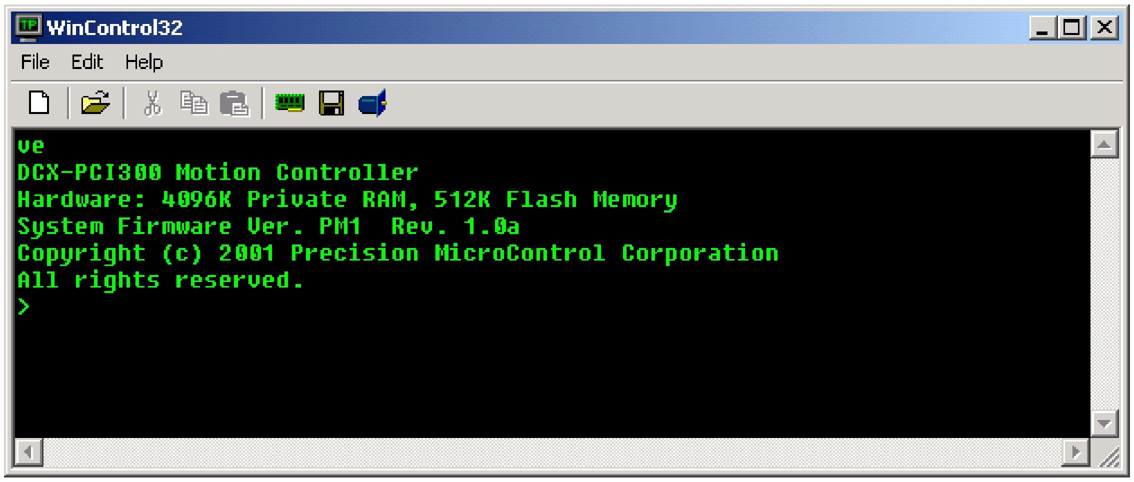
This command generates a motion to an absolute position *n*. A motor number must be specified and that motor must be in the 'on' state for any motion to occur. If the motor is in the off state, only its internal target position will be changed. See the description of **Point to Point Motion** in the **Motion Control** chapter.

The ***MCCL command*** line shows the command syntax and parameter type and/or range

The ***compatibility*** line lists the DCX modules that support the command

The ***see also*** line lists associated MCCL commands

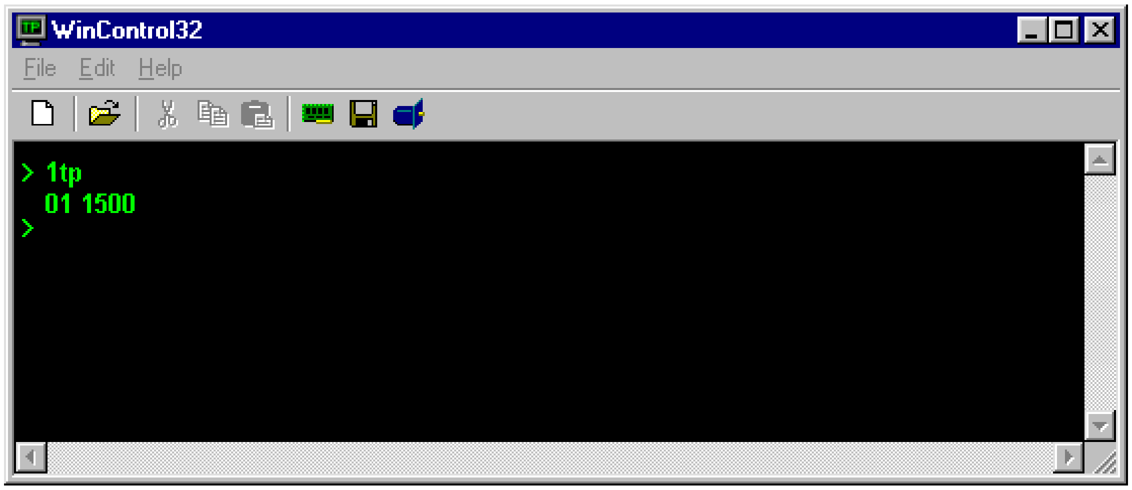
MCCL commands are issued to the DCX via the WinControl utility. This tool allows the user to communicate directly with the controller. The graphic in figure #1 shows the result of executing the **VE** command. This command causes the DCX to report the controller resources and the firmware version.



```
ve
DCX-PCI300 Motion Controller
Hardware: 4096K Private RAM, 512K Flash Memory
System Firmware Ver. PM1  Rev. 1.0a
Copyright (c) 2001 Precision MicroControl Corporation
All rights reserved.
>
```

Figure 5: Using WinControl to display the firmware version of the controller

All axis related MCCL commands will be preceded by an axis specifier, identifying to which axis the operation is intended. The graphic in figure #2 shows the result of issuing the Tell Position (aTP) command to axis number one.



```
> 1tp
  01 1500
>
```

Figure 6: Reporting the position of axis #1

Note that each character typed at the keyboard should be echoed to your display. If you enter an

unrecognized or improperly formatted command the DCX will echo a question mark character, followed by an error code. The **MCCL Error Code** listing can be found in the **Chapter 16** of this manual. On receiving this response, you should re-enter the corrected 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 the **Enter** key is pressed.

Simple moves with MCCL commands

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 command signal to the amplifier/drive, which means and no drive current applied to the motor windings.

> 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 defined by issuing commands to the DCX. The minimum required motor setup parameters include:

- PID filter gains (servo only)
- Trajectory parameters (maximum velocity, minimum velocity, acceleration, and deceleration)
- Allowable following error (servo only)
- Configuring motion limits (hard and soft)

For details on setting up these initial parameters please refer to the **Motion Control** chapter sections titled **Getting Started with Servo's** and **Getting Started with Steppers**. 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 Motor oN (***a*MN**) command. The axis specifier '*a*' of the Motor oN command allows the user to turn on a selected axis or all axes. To enable all, enter the Motor oN command with specifier '*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, which will turn on the Amplifier Enable or Driver Enable output, it should hold steady at one position without moving .

> If the axis is a servo which has not yet been tuned it may not 'hold position'. For assistance with tuning a servo please refer to the descriptions of **Servo Tuning** in the **DCX-PCI300 User's Manual**.

The **T**ell **T**arget (***a*TT**) command will report the position at which the axis should be and the **T**ell **P**osition (***a*TP**) command will report the current actual position of the axis. There are several commands which can used to begin motion, including **M**ove **A**bsolute (MA) and **M**ove **R**elative (MR). 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 which 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 (with no intervening spaces), pressing the Enter 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  <Enter>
```

This command string will cause axis 2 to move from position 1000 to position –1000 a total of 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  <Enter>
```

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 pressing the Enter key. All command strings are retained by the controller until some character other than an Enter key is pressed. This comes in handy for observing the position display during a move. If you enter:

```
1MR1000   <Enter>
1TP   <Enter>
<Enter>
<Enter>
<Enter>
<Enter>
```

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 which allows you to track the position of the axis by simply holding down the Enter key.

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

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

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

Chapter 3

Motion Control

This chapter describes the basic operations typically performed with the DCX-PCI motion controller. 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.

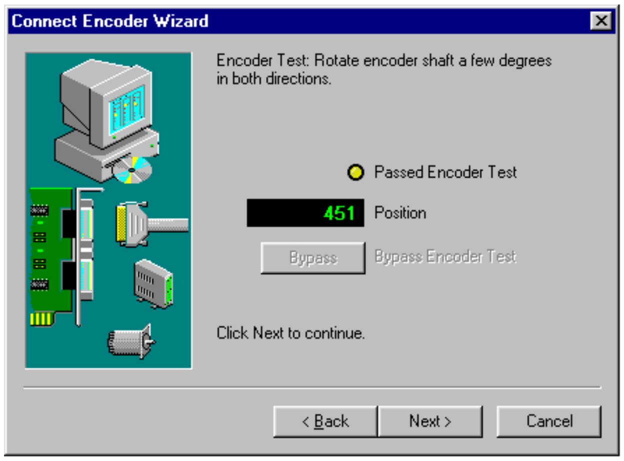
Getting Started with Servo's

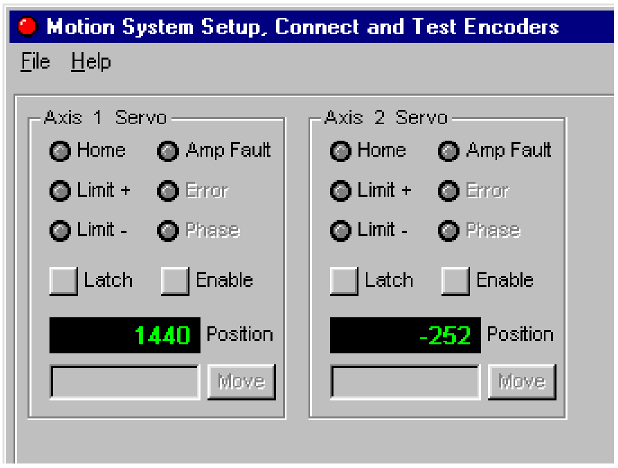
Before a servo motor can be successfully commanded to move certain operations must be performed and parameters must be defined by issuing commands to the DCX. The minimum required setup steps are:

- Verify operation of the encoder
- Define PID filter gains (tune the servo)
- Define Trajectory parameters (maximum velocity, acceleration, and deceleration)
- Set the allowable following error (servo only)
- Configuring motion limits (hard and soft)

Verify operation of the encoder
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.

Testing the encoder with MCCL commands

To test the encoder with MCCL commands:

```
1DH0                                ;Zero the current position of axis #1
1TP                                 ;Report the position of axis #1
```

Manually rotate the motor shaft/encoder, verify that the controller indicates a change of position.

```
1TP                                 ;Report the position of axis #1
```

Manually rotate the motor shaft/encoder in the opposite direction, verify that the controller indicates an appropriate change of position.

```
1TP                                 ;Report the position of axis #1
```

Define PID Filter gains (tune the servo)

To define the PID filter gains you must use the **Servo Tuning** program that is installed along with **Motion Integrator**. Please refer to the **Tuning a Servo** section in **Chapter 6** of the **DCX-PCI300 User's Manual**.

Define Trajectory parameters (maximum velocity, acceleration, and deceleration)

Prior to executing a move command the user must define the trajectory parameters.

- The maximum velocity at which the motor should travel
- The rate at which the motor should accelerate to the maximum velocity
- The rate at which the motor should decelerate to the target
- Select a velocity profile (Trapezoidal or S-curve)

To define the velocity parameters with MCCL commands use the **S**et **V**elocity (***a*****SV*****n***), **S**et **A**cceleration (***a*****SA*****n***), and the **D**eceleration **S**et (***a*****DS*****n***) commands.

```
1SV10000                          ;Set maximum velocity to 10,000 encoder
                                  ;counts per second
1SA100000                         ;Set acceleration to 100,000 encoder counts
                                  ;per second per second
1DS100000                         ;Set deceleration to 100,000 encoder counts
                                  ;per second per second
```

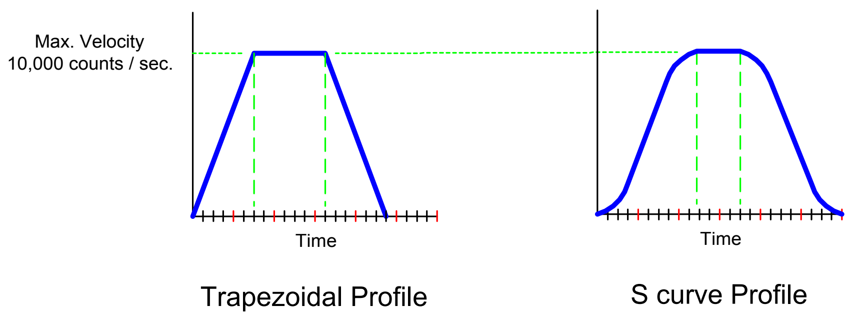
Trapezoidal Profile –

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

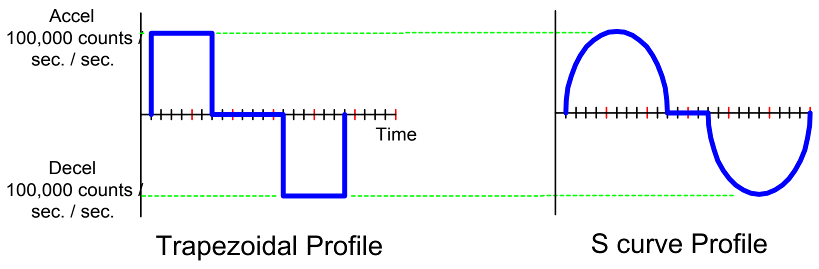
S curve Profile –

'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

DCX Velocity Profiles for Servo Axes



DCX Accel / Decel Profiles for Servo Axes



To define the velocity profile with MCCL commands:

```
1PT              ;Axis #1 moves using trapezoidal profile
2PS              ;Axis #2 moves using S-curve profile
```

Set the allowable following error (servo only)

The following error is the difference between the actual position of an axis and the position where it is supposed to be. Once an axis has been enabled (***a*MN**) the controller calculates the target position of that axis. If at any time there is a difference between the calculated target position and the actual position of an axis the controller will report that difference as the following error.

The DCX provides hard coded following error checking that is enabled when an axis is turned on. The **S**top on **E**rror (***a*SE*n***) command allows the user to define the maximum allowable following error for an axis. If at anytime the following error equals or exceeds this setting, the axis will be disabled (PID filter stops executing, command output set to 0.0 volts). In addition three bits of the axis status word will change state. The Motor On status bit (bit 1) will be cleared, the Motor Error status bit (bit 7) will be set, and the Exceeded Max. Following Error bit (bit 13) will be set. To query the status of an axis use the **T**ell **S**tatus (***a*TS*n***) command.

```
1SE500                          ;Set the max. allowable following error for
axis #1 to 500 encoder counts
```

To disable following error checking issue the Stop on Error command with parameter n = 0.

```
1SE0                            ;Disable following error checking
```

Configuring motion limits (hard and soft)

The DCX provides hard coded error checking for both hard (over travel) and soft (user programmable) limits that is enabled when an axis is turned on. Please refer to the **Defining Motion Limits** section later in this chapter.

Getting Started with Steppers

Before a stepper can be successfully commanded to move certain parameters must be defined by issuing commands to the DCX. The minimum required motor setup parameters include:

- Trajectory parameters (maximum velocity, minimum velocity, acceleration, and deceleration)
- Configuring motion limits (hard and soft)

Define Trajectory parameters (maximum velocity, minimum velocity, acceleration, and deceleration)
Prior to executing a move command the user must define the trajectory parameters.

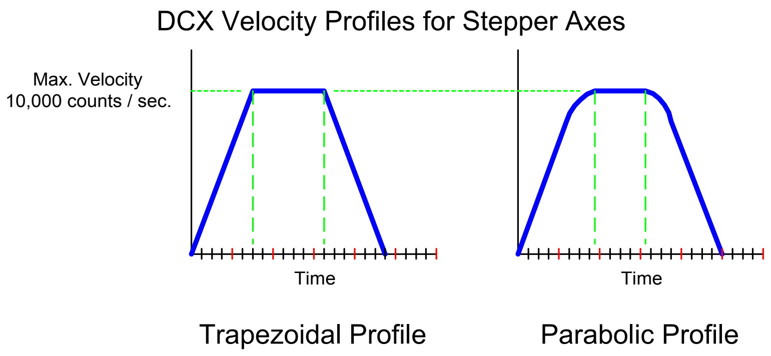
- The maximum velocity at which the motor should travel
- The minimum velocity at which the axis should travel (1% to 10% of maximum velocity)
- The rate at which the motor should accelerate to the maximum velocity
- The rate at which the motor should decelerate to the target
- Select a velocity profile (Trapezoidal or Parabolic)

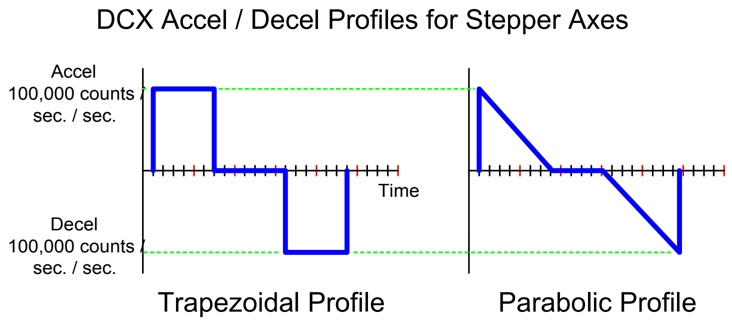
To define the velocity parameters with MCCL commands use the **S**et **V**elocity (***a*SV*n***), set **M**inimum **V**elocity (***aMVn***), **S**et **A**cceleration (***a*SA*n***), and the **D**eceleration **S**et (***a*DS*n***) commands.

```
1SV10000                         ;Set maximum velocity to 10,000 steps per
                                 ;second
1MV1000                          ;set minimum velocity to 1,000 steps per
                                 ;second
1SA100000                        ;Set acceleration to 100,000 steps per second
                                 ;per second
1DS100000                        ;Set deceleration to 100,000 steps per second
                                 ;per second
```

Trapezoidal Profile –
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 –
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





To define the velocity profile with MCCL commands:

```
1PT             ;Axis #1 moves using trapezoidal profile
2PP             ;Axis #2 moves using parabolic profile
```

Configuring motion limits (hard and soft)
The DCX provides hard coded error checking for both hard (over travel) and soft (user programmable) limits that is enabled when an axis is turned on. Please refer to the **Defining Motion Limits** section later in this chapter.

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 susceptible to ‘stalling’, they are not plagued with the familiar open loop stepper system problem of losing steps due to encountering 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 operating a stepper axis in closed loop mode, the open loop operation of the stepper must be verified (as described in the previous section **Getting Started with Steppers**). There are five steps required to then configure a stepper to operate in closed loop mode:

1) Connect and verify the encoder
2) Define the motor steps per rotation / encoder counts per rotation ratio
3) Enable closed loop stepper mode
4) Set the trajectory parameters
5) Tune the axis

Connect and verify the encoder
Connect the stepper's encoder to the module as shown in the following diagram (for detailed wiring information please refer to the **Connectors, Jumpers, and Schematics** chapter in this manual).

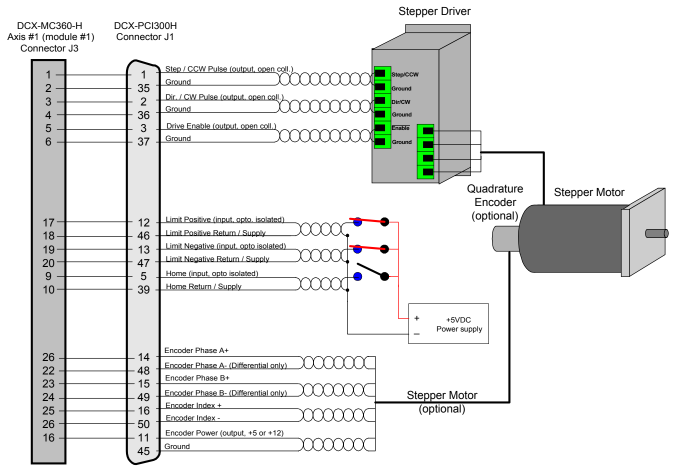
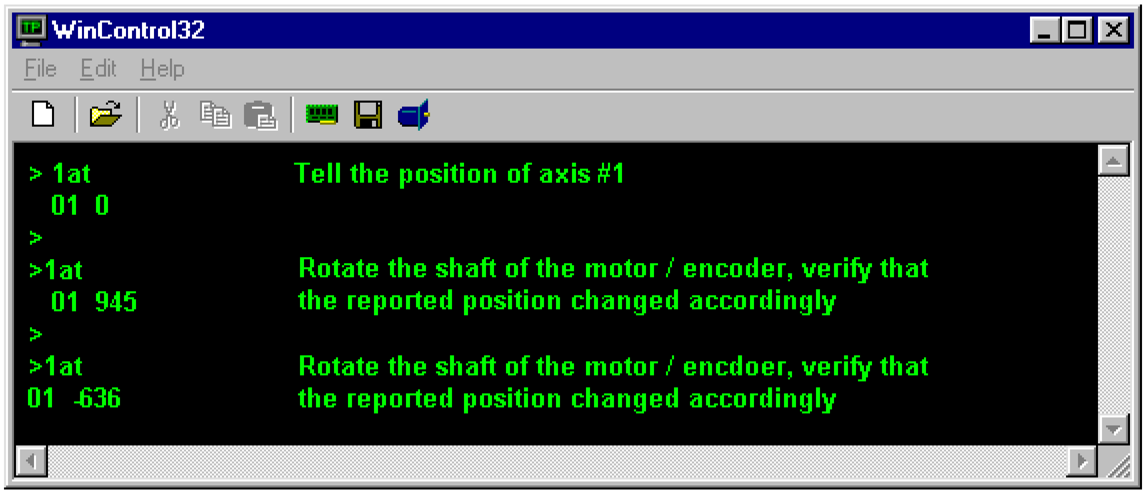

Stepper Driver
DCX-MC360-H
Axis #1 (module #1)
Connector J3
DCX-PCI300H
Connector J1
Step / CCW Pulse (output, open coll.)
Ground
Dir. / CW Pulse (output, open coll.)
Ground
Drive Enable (output, open coll.)
Ground
Step/CCW
Ground
Dir/CW
Ground
Enable
Ground
Quadrature
Encoder
(optional)
Stepper Motor
Limit Positive (input, opto. isolated)
Limit Positive Return / Supply
Limit Negative (input, opto isolated)
Limit Negative Return / Supply
Home (input, opto isolated)
Home Return / Supply
+
–
+5VDC
Power supply
Encoder Phase A+
Encoder Phase A- (Differential only)
Encoder Phase B+
Encoder Phase B- (Differential only)
Encoder Index +
Encoder Index -
Encoder Power (output, +5 or +12)
Ground
Stepper Motor
(optional)

Figure 7: Typical closed loop stepper interconnections

From the WinControl command utility, issuing the **A**uxiliary encoder **T**ell position (***a*****AT**) command, with ***a*** = axis number, will cause the controller to report the current position of the auxiliary encoder. If the encoder is manually rotated in either direction, the position reported should increment or decrement accordingly.



```
> 1at                    Tell the position of axis #1
  01  0
>
>1at                     Rotate the shaft of the motor / encoder, verify that
  01  945                the reported position changed accordingly
>
>1at                     Rotate the shaft of the motor / encdoer, verify that
01  -636                 the reported position changed accordingly
```

Figure 8: Using WinControl to verify the operation of an auxiliary encoder

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 **E**ncoder **S**cale (***a*****ES*****n***) command is used to configure the controller for converting encoder units to step pulses. Parameter ***n*** of the Encoder Scale command is calculated by dividing motor steps per rotation by encoder counts per rotation. For example, if there 1000 encoder counts per rotation (250 line encoder) and the stepper motor has 20,000 steps per rotation, the Encoder Scale parameter ***n*** would be

ES***n*** = motor steps per rotation / encoder counts per rotation.
ES***n*** = 20,000 / 1000
ES***n*** = 20

The Encoder Scale can also be defined from the Servo Setup dialog of the Servo Tuning program.

Enable closed loop stepper mode
To enable closed loop stepper mode issue the Input Mode (***a*IM*n***) command with parameter *n* equal to 1. The MCCL command **TP** (**T**ell **P**osition) and **AT** (**A**ux encoder **T**ell position) will now report the position of the encoder (not the step pulse count), which is the number of encoder counts counted since the axis was last homed / reset. The closed loop stepper mode can also be enabled / disabled from the Servo Setup dialog of the Servo Tuning program.

After switching into or out of closed loop mode you should disable (***a*MF**) and then enable (***a*MN**) the axis to reinitialize the position registers.

Set the trajectory parameters
As with an open loop stepper, the trajectory parameters (***a*SV*n*** - maximum velocity, ***a*SA*n*** - acceleration, ***a*DS*n*** - deceleration, and ***a*MV*n*** - minimum velocity) must be set prior to commanding motion. These trajectory parameters can also be entered from the Servo Setup dialog of the Servo Tuning program 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 used to tune a closed loop stepper.

Step 5 - Toggle the Motor Off and Motor On buttons to initialize the closed loop position registers

Step 6 - Start the move with the **Move +** or **Move -** buttons

Step 7 - Observe the plot of following error during the move

Step 8 - 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 9 - 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 10 - 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.

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 or speed. In this case, once the position error exceeds the value entered using the Stop on Error (***a*****SE*****n***) command (default = 1024) a motor error will occur and the axis will stop. If this happens, the phasing can be changed by either:

- Issuing the PHase (***a*****PH*****n***) command to the axis with a parameter ***n*** = 1
- Selecting the Reverse Phase option in the Servo Tuning Servo Setup dialog
- Swapping 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 using the High Speed (***a*****HS**) command.

```
1HS                                        ;define step rate = HS
1ES25.6                                    ;define steps per rotation (51200)
                                           ; by encoder counts per rotation
                                           ;(200)scaling
1IM1                                       ;enable closed loop stepper mode
1SV35000,1SA170000,1DS170000,1MV3500
                                           ;set trajectory parameters (in encoder
                                           ;units)
1MF,1MN                                    ;turn motor off & on to initialize
                                           ;position registers
```

To disable closed loop stepper operation, issue the Input Mode (***a*****IM*****n***) command with parameter n = 0 or deselect the closed loop check box in the Servo Tuning Servo Setup dialog..

Point to Point Motion

The user can command the DCX to perform a relative or absolute move of an axis. The **M**ove **A**bsolute (***a*****MA*****n***) command will cause the DCX to define the new target position as:

position 0 + *n* (encoder counts or steps)

The **M**ove **R**elative (***a*****MR*****n***) command will cause the DCX to define the target position as:

current position + *n* (encoder counts or steps

```
1MN,2MN                          ;turn on axes 1 and 2
1MA5000                          ;move axis #1 to position 5000
2MR-250                          ;from the current position move axis #2 250
                                 ;counts or steps in the negative direction
```

Note – Prior to executing a absolute or relative move the user must :

- Define PID filter parameters (servo only)
- Define trajectory parameters (max. velocity, accel, & decel)
- Define the allowable following error (servo only)
- Configure the motion limits

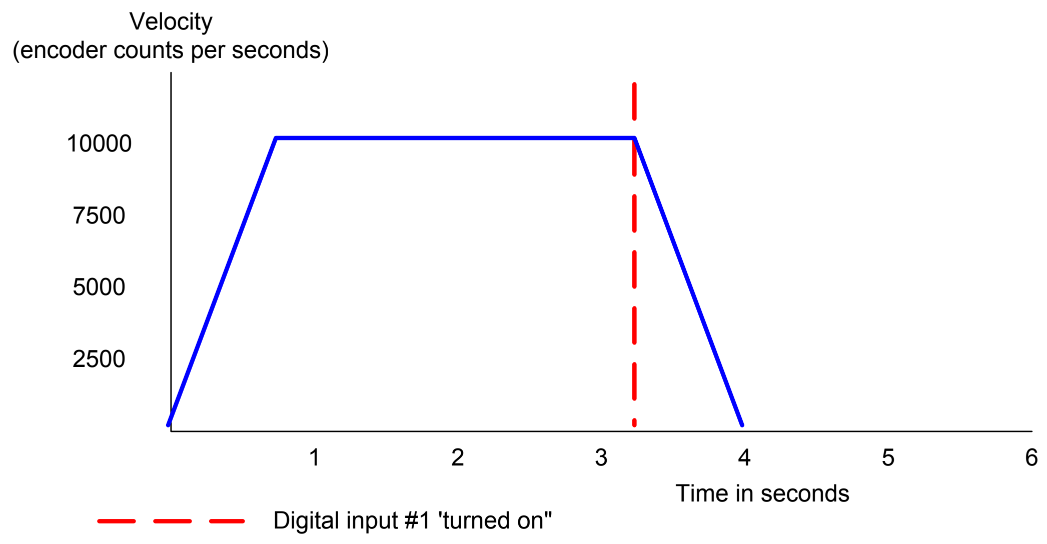
Constant Velocity Motion

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

```
1MN                              ;turn on axis #1
1VM                              ;configure axis #1 for velocity mode
1DI0                             ;move in the positive direction
1GO                              ;Start velocity mode move

WN1                              ;wait for digital input #1 to go active
1ST                              ;Stop the axis
1WS0.1                           ;Dwell for 100 msec's after trajectory complete
```

Note – Prior to executing a velocity mode move the user must :

- Define PID filter parameters (servo only)
- Define trajectory parameters (max. velocity, accel, & decel)
- Define the allowable following error (servo only)
- Configure the motion limits



Contour Motion (arcs and lines)

The DCX supports Linear Interpolated motion with any combination of two to eight 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 **C**ontour **M**ode (***a*****CM*****n***) 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 ***n*** 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.

```
1CM1                            ;Axis #1 is controlling axis in a 3 axis
                                ;contour group
2CM1                            ;Axis #2 is a member of the contour group
3CM1                            ;Axis #3 is a member of the contour group
```

Define the trajectory parameters

The **V**ector **V**elocity (***a*****VV*****n***), **V**ector **A**cceleration (***a*****VA*****n***), and **V**ector **D**eceleration (***a*****VD*****n***) commands 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.

```
1VV10000                          ;Set the vector velocity to 10,000
                                  ;counts/steps per second
1VA50000                          ;Set the vector acceleration to 50,000
                                  ;counts/steps per second per second
1VD50000                          ;Set the vector deceleration to 50,000
                                  ;counts/steps per second per second
```

Define the type of contour move
The **C**ontour **P**ath (***a*****CP*****n***) command is used to define the type of move the contour group will execute. The following types of contour motion are supported:

Parameter n	***Contour move type***	***Description***
0	User defined, 1 to 8 axes	Specifies that this block is a user defined contour path motion. Parameter *a* should be set to the controlling axis number.
1	Linear interpolated move, 1 to 8 axes	Specifies that this block is a linear contour path motion. Parameter *a* should be set to the controlling axis number.
2	Clockwise arc, 2 axes	Specifies that this block is a clockwise arc contour path motion. Parameter *a* should be set to the controlling axis number.
3	Counter Clockwise arc, 2 axes	Specifies that this block is a counter-clockwise arc contour path motion. Parameter *a* should be set to the controlling axis number.

Examples of a linear move and a clockwise arc follow:

```
;Linear move

1CP1,1MA10000,2MA20000,3MR-5000


;Clockwise arc move

1CP2,1CA20000,2CA0,1MA40000,2MA0
```

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 next. 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 seven axes is the controlling axis, only 16 motions can be queued up. The Tell contour (***a*TX**) command will report how many contour moves have been executed since the axes were last configured for contour motion with the **C**ontour **M**ode command (***a*CM*n***). 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 **S**ynchronization o**N** (***a*SN**) command. This command should be issued to the controlling axis **before** issuing any contour moves. Moves issued after the **SN** command will be queued into the contour buffer. To begin executing the moves in the buffer, issue the ***a*GO** command to the controlling axis . To return to normal operation (immediate execution of contour move commands), issue the **N**o **S**ynchronization command (**aNS**) to the controlling axis.

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.

```
1CM1                            ;Axis #1 is controlling axis in a 3 axis
                                ;contour group
2CM1                            ;Axis #2 is a member of the contour group
3CM1                            ;Axis #3 is a member of the contour group

;Linear move #1
1CP1,1MA85000,2MR12000,3MA-33000

;Linear move #2
1CP1,1MA0,2MA0,3MA0

;Linear move #3
1CP1,1MA5000,2MR23000,3MA-16000
```

Arc Motion

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

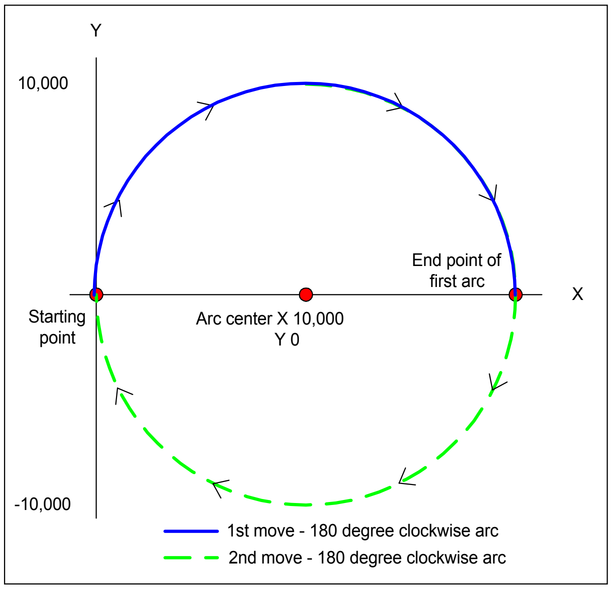
Specify center and end point
Specify radius and end point (not supported by MCAPI)
Specify center and ending angle (not supported by MCAPI)

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 arc **C**enter **A**bsolute (***a*****CA*****n***) and arc **C**enter **R**elative (***a*****CR*****n***) commands are used to specify the center position of the arc. This command also defines which two axes will perform the arc motion. The **M**ove **A**bsolute and/or **M**ove **R**elative 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:

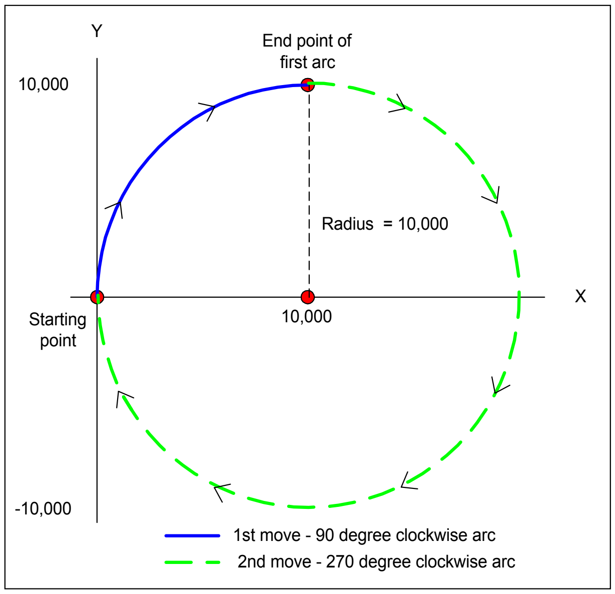
```
1CM1                                ;Axis #1 is controlling axis in a 2 axis
                                    ;contour group
2CM1                                ;Axis #2 is a member of the contour group


;180 degree clockwise arc (defined by arc center absolute coordinates)
;starting at X=0, Y=0 and moving to X=20000, Y=0
1CP2,1CA10000,2CA0,1MA20000,2MA0


;180 degree clockwise arc (defined by arc center relative coordinates)
;starting at X=20000, Y=0 and moving to X=0, Y=0
1CP2,1CR-10000,2CR0,1MR-20000,2MR0
```



Arc motions by specifying the radius and end point

The Radius of aRc (***a*****RR*****n***) command is used to execute an arc move by specifying the radius of an arc. The axis specifier *a* should be the controlling axis for the contour move. The parameter *n* should equal the radius of the arc. If the arc is greater than 180 degrees, the parameter must be expressed as a negative number.

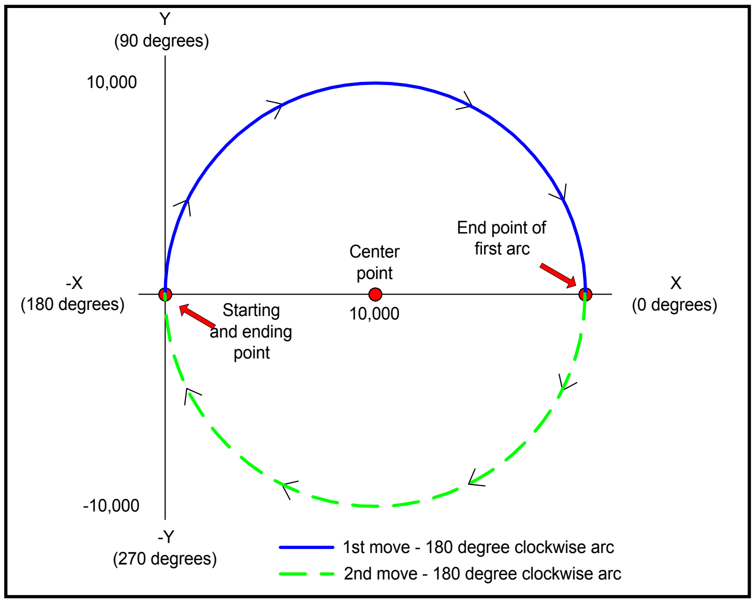
```
1CM1,2CM1                             ;define axis 1 as controlling axis
1CP2,1MR10000,2MR10000,1RR10000       ;90°  clockwise arc, radius = 10000
1CP2,1MR-10000,2MR-10000,1RR-10000    ;270° degree arc, radius = 10000,
                                      ;negative radius parameter indicates
                                      ;arc greater than 180°
```



Arc motions by specifying the center point and ending angle

The arc **E**nding **A**ngle absolute (***a*EA*n***) and arc **E**nding angle **R**elative (***a*ER*n***) commands can be used in conjunction with the arc **C**enter **A**bsolute (***a*CA*n***) and arc **C**enter **R**elative (***a*CR*n***) to execute an arc motion. When using this method to specify an arc, the move absolute and move relative commands are not used. The center point commands define the radius of the arc. The ending arc angle commands define the end point of the arc as an angle relative to the X axis.

```
1CP2,1CA10000,2CA0,1EA0                ;Clockwise arc motion in X & Y
1CP2,1CR-10000,2CR0,1ER180             ;Clockwise arc motion in X & Y
```



Changing the velocity 'on the fly'

'On the fly' velocity changes during contour mode motion are accomplished by using the Velocity Override (***a*****VO*****n***) command. 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 **S**ynchronization o**N** (***a*****SN**) 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 the **GO** command with ***a*** = controlling axis and ***n*** = 1. Motion will continue from the first to the last point in the contour buffer. To return to normal operation, issue the **N**o **S**ynchronization (***a*****NS**) command with ***a*** = the controlling axis.

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 8 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.

```
1CM1,2CM1,3CM1                    ;configure axes 1, 2, & 3 as a contour
                                  ;group with axis #1 as the controlling
                                  ;axis

1CP0,1MA1000,2MA1000,3MA1000,1CD10000
                                  ;execute a user defined contour path with
                                  ;a contour distance of 10,000

1CP0,1MA0,2MA0,3MA0,1CD20000      ;execute a second user defined contour
                                  ;path with a contour distance of 10,000.
                                  ;The CD command parameter n is 10,000 +
                                  ;10,000 = 20,000
```

When executing User Defined Contour Path moves each compound move command must include the **C**ontour **D**istance (***a*****CD*****n***) command. Parameter ***n*** is an absolute value, relative to the positions of the included axes when the **C**ontour **M**ode (***a*****CM*****n***) command was last issued. Re-issuing the Contour Mode command 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 (***a*****CP*****0***) with the **C**ontour **D**istance (***a*****CD*****n***) command with parameter ***n* = *0***, the user can execute multi axis contour moves and limit the maximum velocity of an individual axis. In this mode of operation the Vector Velocity (VV) 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.

```
1SV300,1SA1000,1DS1000
2SV100,2SA1000,2DS1000

1CM1,2CM1
1CP0,1MR1000,2MR1000,1CD0
```

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 Set Velocity (SV) 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 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 **S**et **M**aster (***a*SM*n***) command is used to enable electronic gearing. Axis specifier ***a*** identifies a slave axis and parameter ***n*** identifies the master axis. To slave axes 2 and 3 to axis one:

```
2SM1,3SM1
```

The **S**et **S**lave ratio (***aSSn***) command is used to define the ratio of a slave axis. 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 example defines axis one as the master and sets the slave ratio for axes 2, 3, and 4.

```
;Enable gearing of axis 2, 3, and 4
2SM1,3SM1,4SM1
2SS5
3SS-.1
4SS-1000
```

To terminate electronic gearing issue the Set Master command to each slave axis with parameter n = 0. Enable either position or velocity mode for the slave axes. Complete the operation by issuing the Motor On command .

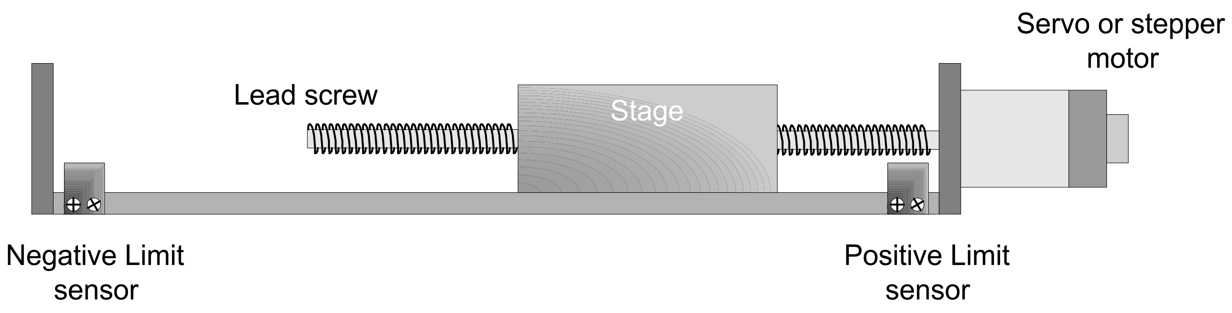
```
2SM0,3SM0,4SM0
2PM,3PM,4VM
1MN,2MN,3MN,4MN
```

Note – if the slave axes are servos, 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).

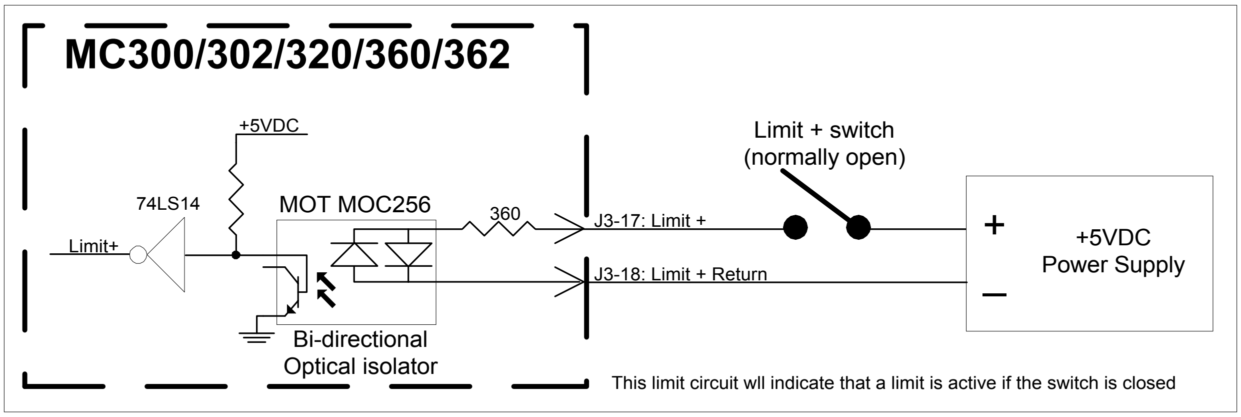
Defining Motion Limits

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.

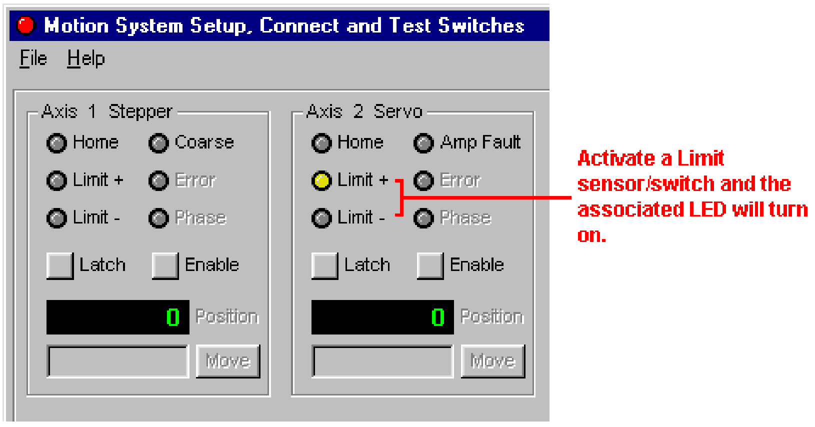


Hard Limits

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.


MC300/302/320/360/362
+5VDC
74LS14
Limit+
MOT MOC256
360
J3-17: Limit +
J3-18: Limit + Return
Bi-directional Optical isolator
Limit + switch (normally open)
+
–
+5VDC Power Supply
This limit circuit wll indicate that a limit is active if the switch is closed

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



Activate a Limit sensor/switch and the associated LED will turn on.

⚠ 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.

Motion Integrators limit LED's display the **current state** (Status bits 28 an 29) not the 'tripped' flag (status bits 17 and 19) of the limit inputs.

To use MCCL commands to report the state of the limit sensor inputs use the Tell Status command and set parameter n to the status bit to be tested:

```
1TS28                             ;report the state of axis #1 limit + sensor
2TS29                             ;report the state of axis #2 limit - sensor
```

The DCX supports two levels of limit switch handling:

- Auto axis disable
- Simple monitoring

The MCCL commands **L**imit **M**ode (***a*LM*n***) and motion **L**imits o**N** (***a*LN*n***) allow 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 by checking the status of the axis. The **LM** command defines how an axis will respond to an active limit.

Parameter n	***Limit Mode - Hard Limit Mode description***
0	Turn off the axis (PID filter disabled - command signal to 0, disable amplifier/driver) when the hard limit input 'goes' active or a soft motion limit is exceeded
1	Stop the axis **abruptly** when the hard limit input goes active
2	Decelerate and stop the axis when the hard limit input goes active
3	Turn off the axis (PID filter disabled - command signal to 0, disable amplifier/driver) when the hard limit input 'goes' active
128 (plus 0, 1, or 2)	Invert the active level of the hard limit input to high true. Typically used for normally closed limit sensors

The **LN** command enables hard coded limit error checking.

Parameter n	***Hard Limits - Limit oN parameter description***
0	Enable both hard limits and soft limits
1	Enable Limit + error checking
2	Enable Limit – error checking
3	Enable Limit + and Limit – error checking

The **LF** command disables hard coded limit error checking.

When limit error checking is enabled by the LM and LN commands, status bits 17 (Limit + tripped), 19 (Limit – tripped), 7 (Motor error) indicate an error condition. For a normally closed limit switch, the LM parameter n must be set to 128 + the stop mode (0, 1, or 2) properly define the active level of the limit circuit.

When a limit event occurs, motion of that axis will stop and the error flags (bits 17 or 19 and 7) will be set and remain set until the motor is turned back on by the **M**otor o**N** (***a*MN**) command. The axis must then be moved out of the limit region with a **M**ove **A**bsolute or **M**ove **R**elative command.

```
;Set both hard limits of axes 1 and 2 to stop smoothly when tripped, Note-
;axis #2 has normally closed sensors so the active level must be inverted

1LM2,2LM130
1LN0,2LN0
```

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 bits 28 and 29 of the status word.

Soft Limits

Soft motion limits allow the user to define an area of travel that will cause a DCX error condition. The allowable range of travel is defined using the **H**igh motion **L**imit (***a*HL*n***) and the **L**ow motion **L**imit (***a*LL*n***) commands. When soft motion limits are enabled by the **L**imit o**N** (***a*LN*n***) command, if an axis is commanded to move to a position that is outside the soft limits, an error condition is indicated (status bit 21 or 23 and bit 7 will be set) and the axis will stop.

Parameter n	***Limit Mode – Soft Limit Mode description***
0	Turn off the axis (PID filter disabled - command signal to 0, disable amplifier/driver) when the soft motion limit is exceeded or a hard limit input 'goes' active
4	Stop the axis **abruptly** when the soft motion limit is exceeded
8	Decelerate and stop the axis when the soft motion limit is exceeded
12	Turn off the axis (PID filter disabled - command signal to 0, disable amplifier/driver) when the soft motion limit is exceeded

To enable both hard and soft limits the value of parameter ***n*** of the LM command is the sum of the hard limit setting plus the soft limit setting

Parameter n	***Soft Limits - Limit oN parameter description***
0	Enable both hard limits and soft limits
1	Enable High Limit error checking
2	Enable Low Limit error checking
3	Enable High and Low Limit – error checking

```
;Set both hard and soft limits of axes 1 and 2 to stop smoothly when tripped,
;Note: axis #2 has normally closed sensors so the active level must be inverted
1LM10,2LM138
1LN0,2LN0
1HL100000,2HL20000
1LL0,2LL-20000
```

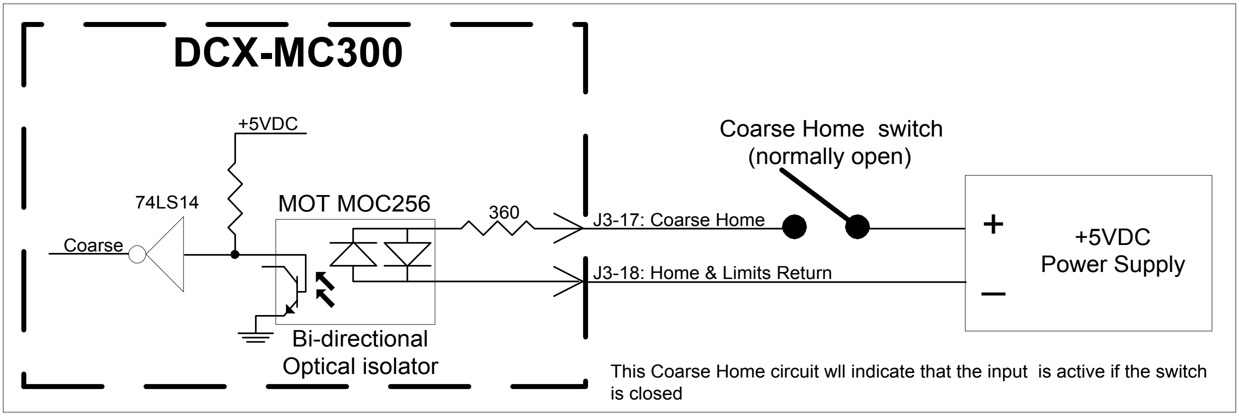
Homing Axes

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 issue the **D**efine **H**ome (***a*****DH*****n***) command 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, the user can define a custom homing sequence with MCCL commands.

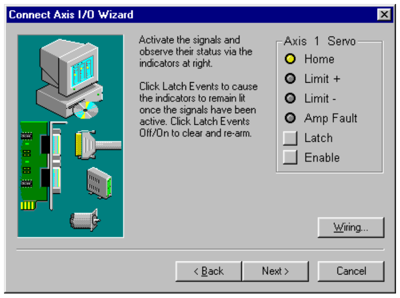
Connecting a Home Sensor

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.


DCX-MC300
+5VDC
74LS14
Coarse
MOT MOC256
360
Bi-directional Optical isolator
Coarse Home switch (normally open)
J3-17: Coarse Home
J3-18: Home & Limits Return
+
–
+5VDC Power Supply
This Coarse Home circuit wll indicate that the input is active if the switch is closed

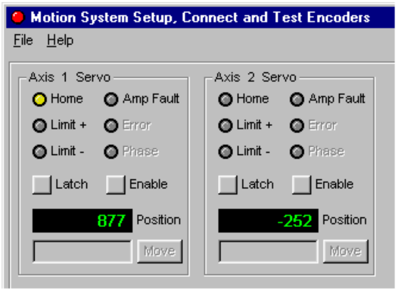
Verifying the operation of the Home Sensor

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.


Connect Axis I/O Wizard

Activate the signals and observe their status via the indicators at right.

Click Latch Events to cause the indicators to remain lit once the signals have been active. Click Latch Events Off/On to clear and re-arm.


Motion System Setup, Connect and Test Encoders

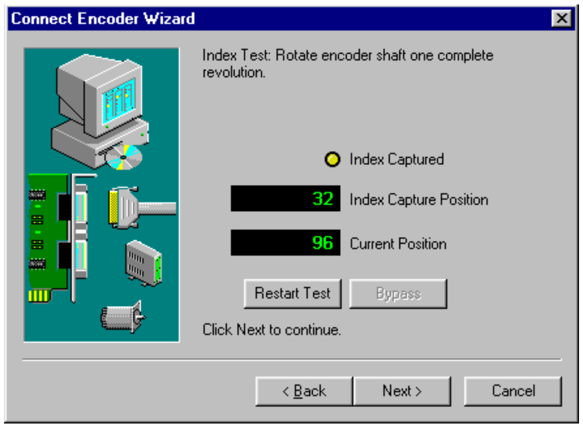
Axis 1 Servo: 877 Position

Axis 2 Servo: -252 Position

To use MCCL commands to report the state of the Coarse Home (servo) or Home (stepper) sensor inputs use the Tell Status command and set parameter n to the status bit to be tested:

```
1TS25                          ;report the state of axis #1 Coarse Home
                               ;sensor input (servo axis)
2TS24                          ;report the state of axis #2 Home sensor
                               ;input (stepper axis)
```

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.


Connect Encoder Wizard

Index Test: Rotate encoder shaft one complete revolution.

Index Captured

32 Index Capture Position

96 Current Position

Click Next to continue.

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

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 MCCL command sequence will home a multi turn rotary stage:

```
MD1,1VM,1DI0,1SV50000,1GO,MJ10          ;start velocity mode move
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
```

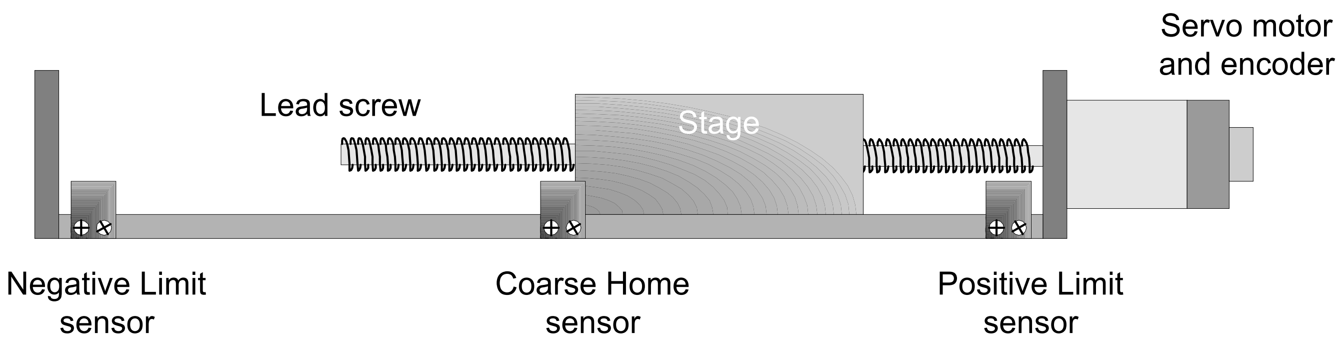
For MCCL homing samples that can be downloaded to the controller and executed please refer to the MotionCD (\PCI300\MCCL\Homing).

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 following flow chart and program sample will move the stage in the positive direction. If the coarse home sensor 'goes active' before the positive limit sensor, the Index Arm and Wait for Index commands will redefine the position of the axis when the index mark is captured. If the positive limit sensor 'goes active', the stage will change direction, until **both** the coarse home sensor **and** the encoder index are active, at which point the position will be redefined.

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

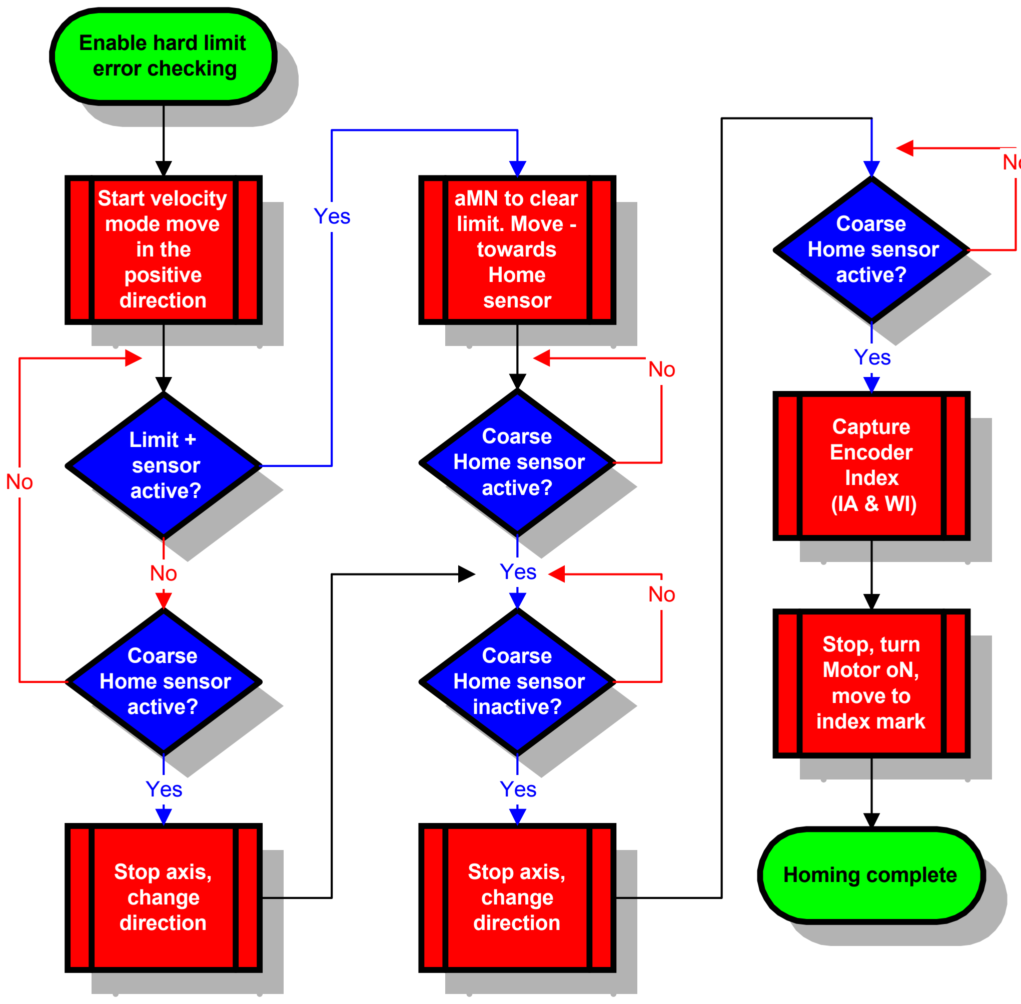

Enable hard limit error checking
Start velocity mode move in the positive direction
Limit + sensor active?
Yes
No
No
Coarse Home sensor active?
Yes
Stop axis, change direction
aMN to clear limit. Move - towards Home sensor
Coarse Home sensor active?
No
Yes
Coarse Home sensor inactive?
No
Yes
Stop axis, change direction
Coarse Home sensor active?
No
Yes
Capture Encoder Index (IA & WI)
Stop, turn Motor oN, move to index mark
Homing complete

Figure 9: Typical homing routine for a closed loop system

```
MD1,1SV50000,1SA100000,1DS100000,1LM2,1LN3,1VM,1DI0,1GO,MJ10
                                              ;begin positive velocity mode move
MD10,LU"STATUS",1RL@0,IS25,MJ11,NO,IS17,MJ15,NO,JR-8
                                              ;test for sensors (home and +limit)
MD11,1ST,1WS.01,1DI1,1GO,MC21,1ST,1WS.01,MJ12
                                              ;move out of coarse home
MD12,MC22,1DI0,1GO,MC23,MJ13                  ;move back into coarse home sensor
MD13,1IA1000,MC24,1WI,1ST,1WS.01,MJ14         ;capture index (position = 1000) then
                                              ;stop
MD14,1PM,1MN,1MA1000,1WS.1                   ;initialize axis, move to index
MD15,1WS.1,1MN,1DI1,1GO,MC23,MC21,1ST,1WS.01,MJ12
                                              ;limit + true, move negative until coarse
                                              ;home true


;homing sub routines
MD21,LU"STATUS",1RL@0,IC25,BK,NO,JR-5         ;test for coarse home false
MD22,1SV10000,1SA100000,1DS100000             ;reduce trajectory parameters
```

```
MD23,LU"STATUS",1RL@0,IS25,BK,NO,JR-5          ;test for coarse home true
MD24,LU"STATUS",1RL@0,IS10,BK,NO,JR-5          ;test for Index Found
```

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

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.

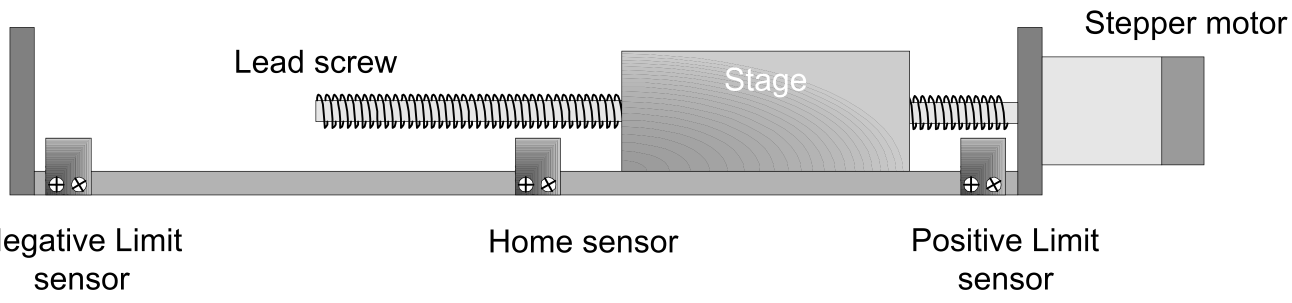
This method **is not recommended** for applications that require **high repeatability and accuracy**. To achieve the highest possible accuracy when using this method, significantly reduce the velocity of the axis while polling for the active state of the limit input.

The following sequences will home an axis at the position where the positive limit sensor 'goes active':

```
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 when limit + not active, define position as
                                     ;0. Move to position -100.
```

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

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 **E**dge **L**atch (***a*****EL*****n***) and **W**ait for **E**dge (***a*****WE**) commands are used to capture the position of the Home sensor active edge.

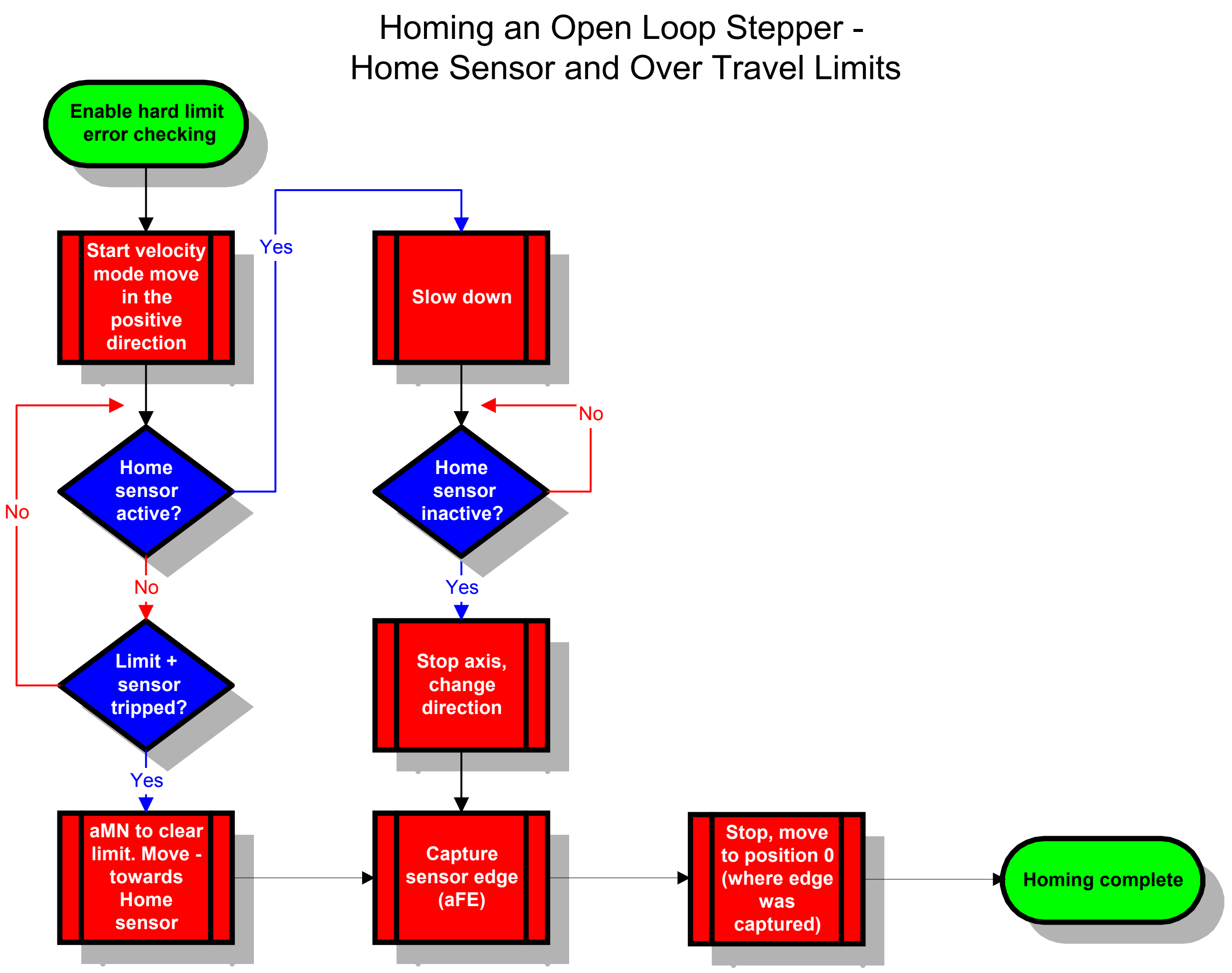

Homing an Open Loop Stepper - Home Sensor and Over Travel Limits
Enable hard limit error checking
Start velocity mode move in the positive direction
Home sensor active?
Yes
No
Limit + sensor tripped?
No
Yes
aMN to clear limit. Move - towards Home sensor
Slow down
Home sensor inactive?
No
Yes
Stop axis, change direction
Capture sensor edge (aFE)
Stop, move to position 0 (where edge was captured)
Homing complete

Figure 10: Typical homing routine for a stepper

```
; 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
                                          ;is 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
```

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

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

```
; 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.
```

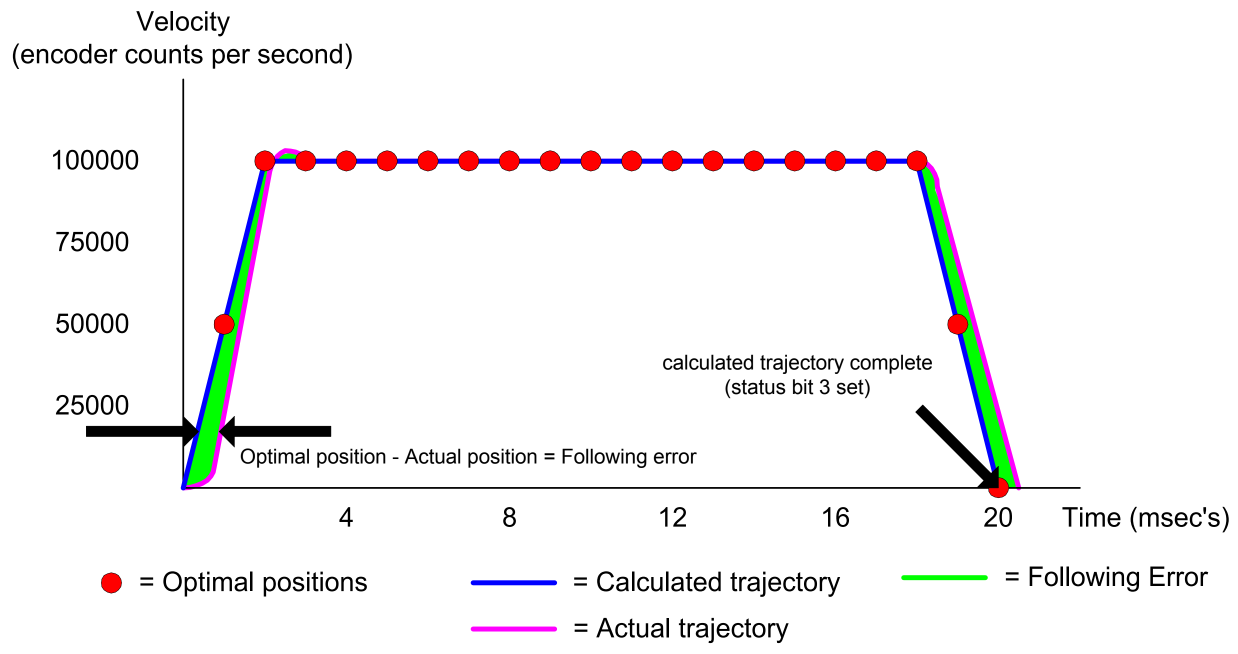
For MCCL homing samples that can be downloaded to the controller and executed please refer to the MotionCD (\PCI300\MCCL\Homing).

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 '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 optimal position is equal to the move target, the 'digital trajectory' of the move has been completed and the Trajectory Complete status bit (bit 3) will be set. This status flag is the conditional component of the **W**ait for **S**top (**a*WS*n**) command. For a closed loop stepper axis when the encoder position is equal to the move target, the 'digital trajectory' of the move has been completed and the Trajectory Complete status bit (bit 3) will be set. For an open loop stepper axis when the step count (pulses issued) is equal to the move target, the 'digital trajectory' of the move has been completed and the Trajectory Complete status bit (bit 3) will be set.

As shown in the diagram above, a following error of a closed loop servo can cause the Trajectory Complete status bit (bit 3) to be set **before the axis has reached its target**. Issuing the Wait for Stop (***a*WS*n***) command with a dwell period defined by parameter ***n*** allows the user to delay additional command execution until the move has been completed (following error = 0). In the example below, the dwell period (parameter ***n***) of the Wait for Stop command is set to 5 msec, giving the axis time to stop and settle.

```
1MR500                          ;move 500 counts
1WS0.005                        ;dwell until trajectory complete + 5 msec's
```

Another method of indicating the end of a move of a servo is to use the **W**ait for **T**arget (***a*WT*n***) command. The Wait for Target command allows the user to define an acceptable range (encoder counts) around the target position of the move. The at target range is defined with two commands, set position **D**ead**B**and (***a*DB*n***) and set **D**elay at **T**arget (***a*DT*n***). Parameter ***n*** of the **DB** command sets the +/- range (in encoder counts) around the move target. Parameter ***n*** of the **DT** command defines how long the axis must remain within that range in order to set the At Target status bit (bit 2) of the axis stats word. The example below sets the At Target range to +/- 5 encoder counts, and it must stay within that range for 50 msec's before the At Target bit will be set.

```
1DB5                                ;deadband range = +/-5
1DT0.05                             ;At target dwell time = 50 msec's
```

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. (1 msec. is the interval between execution of the trajectory generator of the DCX).

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

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).

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

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
```

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.

Chapter Contents

Chapter
4

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’. 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. The **A**uxiliary encoder **T**ell position (***a*AT**) command is used to report the position of an auxiliary encoder. The **G**et au**X**iliary encoder position (***a*GX**) command will load the accumulator (user register 0) with the position of the auxiliary encoder.

```
;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

1MA1675.5                         ;move to position 1675.5
1WS0.01                           ;wait 10 msec's after trajectory complete
1GX                               ;load accumulator with aux. encoder position

IG1674,BK,NO,AS1675,AM-1,1MR@0,1WS0.01
                                  ;if aux. encoder greater than position 1674 the
                                  ; move is complete. If below 1674, subtract 1675.5
                                  ;(target position), multiply by -1, issue relative
                                  ;move and wait for move complete
```

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. The **A**uxiliary encoder **T**ell position (***a*****AT**) command is used to report the position of an auxiliary encoder. The **G**et au**X**iliary encoder position (***a*****GX**) command will load the accumulator (user register 0) with the position of the auxiliary encoder.

```
;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

1MA222.75                         ;move to position 1675.5
1WS0.01                           ;wait 10 msec's after trajectory complete
1GX                               ;load accumulator with aux. encoder position

IG222.5,BK,NO,AS222.75,AM-1,1MR@0,1WS0.01
                                  ;if aux. encoder greater than 222.5 move
                                  ; complete. If below 222.5, subtract 222.75
                                  ;(target ;position), multiply by -1, issue
                                  ; relative move and wait for move complete
```

For additional 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 the reported position of the auxiliary encoder. The repeatability of a system homed using the index mark will be **significantly better** than that of a system that uses only a mechanical switch / electromechanical sensor. The following example will zero the position of a stepper motor at the location of the index mark of the auxiliary encoder:

User scaling other than 1:1 will cause aux. encoder homing to fail. Prior to homing an aux. encoder return all user scaling to default (1:1)

```
;Stepper motor Auxiliary encoder homing routine. Zero the position of both the auxiliary
;encoder and the step count ;register ;at the location of the auxiliary encoder index
;mark
;Note - register #105 must be loaded with the encoder scaling factor (steps ;per
;rotation / encoder counts per rotation)

AL0,AR100,AR101                           ;zero user registers 100 & 101
AL-25.6,AR105                             ;define steps per rotation to encoder
                                          ;counts per rotation scaling (51200 / 2000)

1MN,1VM,1SV50000,1SA1000000,1DS1000000,1DI0,1GO
                                          ;begin moving in velocity mode.
LU"STATUS",1RL@0,IS25,BK,NO,JR-5          ;loop on Coarse Home status bit
1AF,LU"STATUS",1RL@0,IS27,BK,NO,JR-5     ;Enable aux. encoder index mark
                                          ;capture, loop until index captured
1AX,AR100,tr100                           ;load accumulator with encoder
                                          ;position when index occurred, store
                                          ;the value in user register 100
1ST,1WS0.01,1GX,AR101,tr101               ;Stop, load encoder position into
                                          ;register 101
1PM,1MN
AL@101,AS@100,AM@105,1MR@0,1WS0.01
                                          ;Calculate how far the axis moved away
                                          ;from the encoder index. Issue move to
                                          ;the position of the index. Note - The
                                          ;multiply (AM@105) is used to scale
                                          ;axis units to aux. encoder units
TR100,1AT                                 ;compare index position with current
                                          ;position
DH0,1AH0,1MF,1MN,1PM                      ;set step pulse & aux encoder
                                          ;positions to 0
```

Unlike the **F**ind **I**ndex command, which re-defines the position reported by a servos' encoder, the **A**ux. encoder **F**ind index command does not re-define the reported position of the auxiliary encoder. The **AF** command only arms the capture of the aux. encoder index mark, which is then indicated by the Auxiliary encoder index status bit (bit 27)

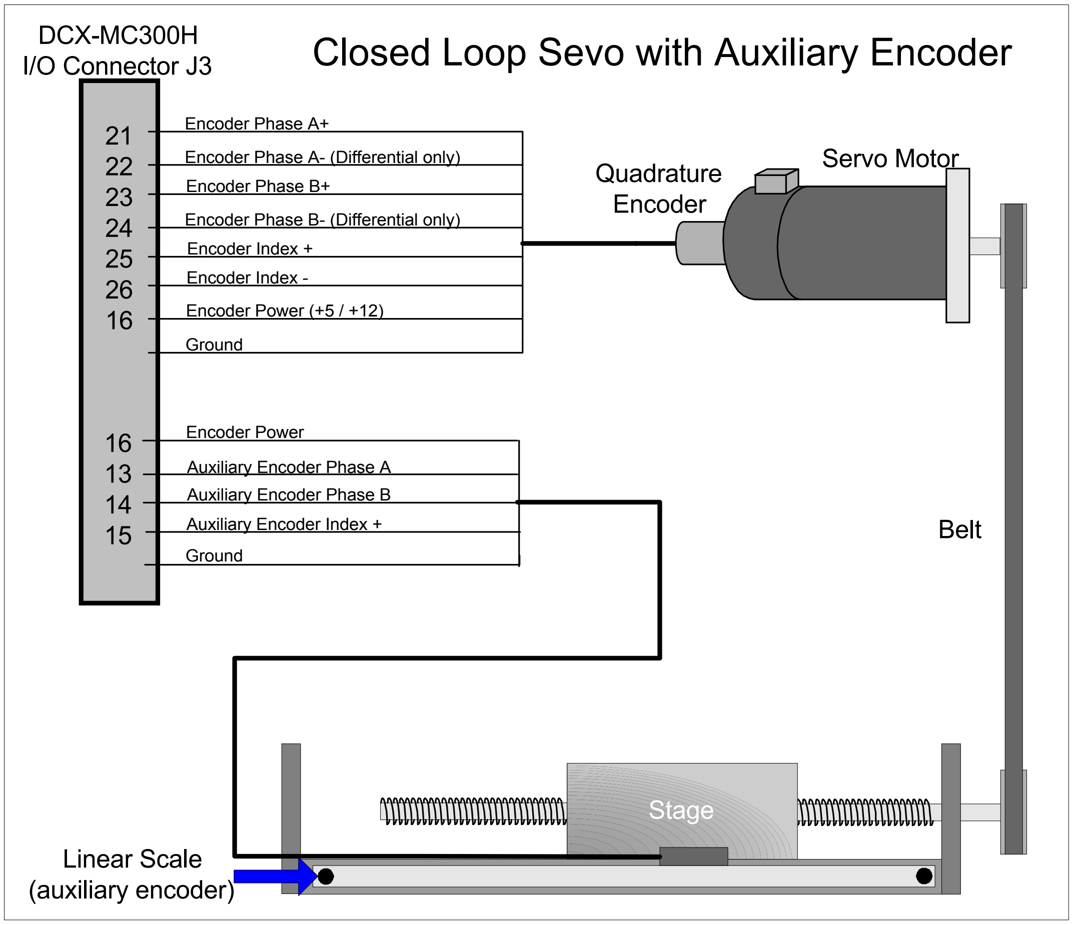
For MCCL homing samples that can be downloaded to the controller and executed please refer to the MotionCD (\PCI300\MCCL\Homing).

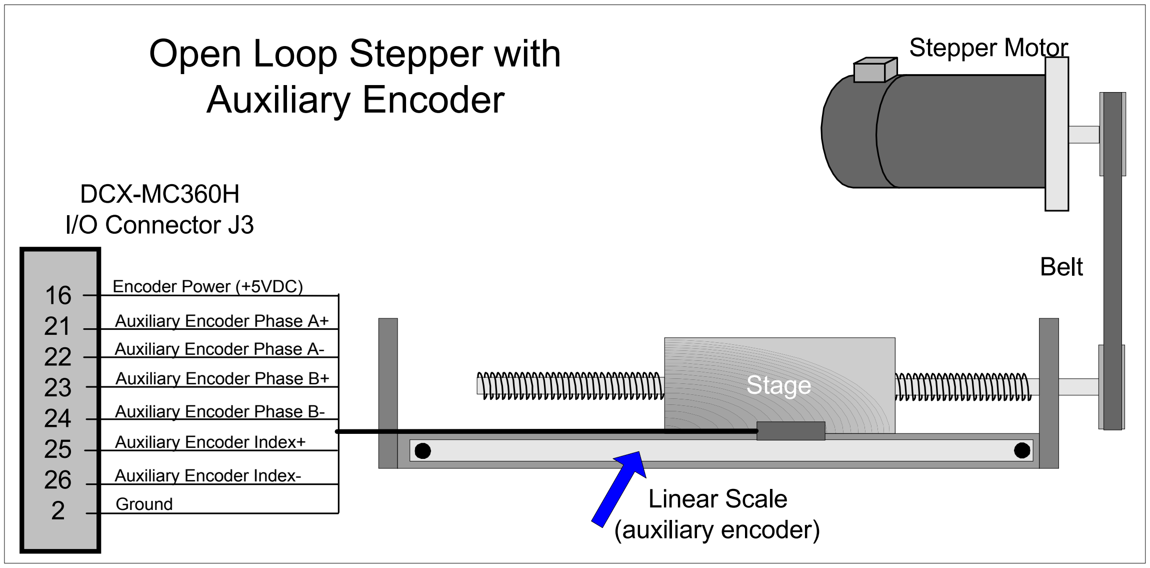
If no encoder index mark output is available, the position of the auxiliary encoder can be redefined at anytime using the **A**uxiliary encoder define **H**ome (***a*AH**) command. The following programming example will re-define the position of the auxiliary encoder of a stepper axis when it is homed.

```
1VM,1SV1000,1DI0,1GO                    ;start moving in velocity mode
1EL0,LU"STATUS",1RL@0,IC10,JR-3,1WE,1ST,1WS.1,1MF,1MN,
1PM,1MA0,1ST,1WS0.01                    ;move to position 0 (location of Home
                                        ;input)
1AH0                                    ;set reported position of aux. encoder
                                        ;to 0
```

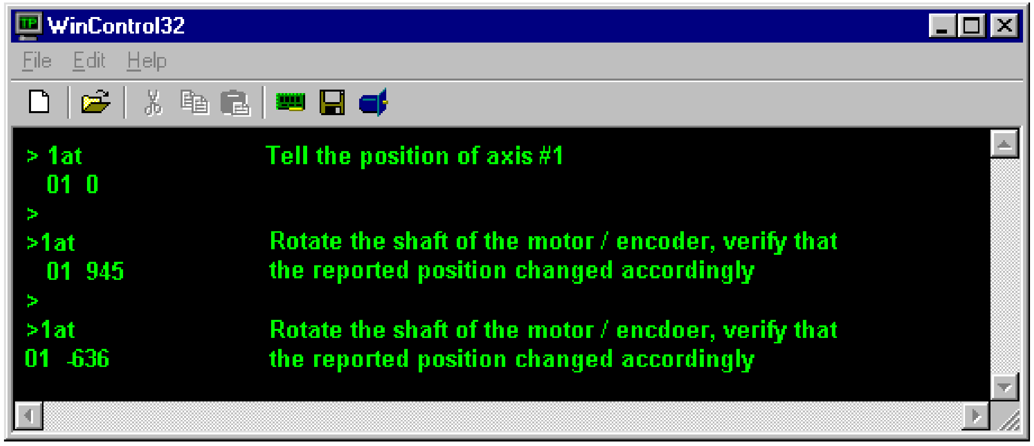
Auxiliary Encoder Connections

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 of the **DCX-PCI300 User's Manual**.


DCX-MC300H
I/O Connector J3
Closed Loop Sevo with Auxiliary Encoder
21
22
23
24
25
26
16
Encoder Phase A+
Encoder Phase A- (Differential only)
Encoder Phase B+
Encoder Phase B- (Differential only)
Encoder Index +
Encoder Index -
Encoder Power (+5 / +12)
Ground
Quadrature Encoder
Servo Motor
16
13
14
15
Encoder Power
Auxiliary Encoder Phase A
Auxiliary Encoder Phase B
Auxiliary Encoder Index +
Ground
Belt
Stage
Linear Scale (auxiliary encoder)


Open Loop Stepper with Auxiliary Encoder
DCX-MC360H I/O Connector J3
16 Encoder Power (+5VDC)
21 Auxiliary Encoder Phase A+
22 Auxiliary Encoder Phase A-
23 Auxiliary Encoder Phase B+
24 Auxiliary Encoder Phase B-
25 Auxiliary Encoder Index+
26 Auxiliary Encoder Index-
2 Ground
Stepper Motor
Belt
Stage
Linear Scale (auxiliary encoder)

Verifying the Operation of the Auxiliary Encoder
Use the WinControl MCCL command utility to verify the proper operation of the DCX module and the auxiliary encoder. The example below details testing an encoder connected to axis number one. To test a different axis, issue the **A**uxiliary encoder **T**ell position (***a*****AT**) command with the appropriate prefix (where *a* = axis number).



```
> 1at                   Tell the position of axis #1
  01  0
>
>1at                    Rotate the shaft of the motor / encoder, verify that
  01  945               the reported position changed accordingly
>
>1at                    Rotate the shaft of the motor / encdoer, verify that
01  -636                the reported position changed accordingly
```

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 **B**acklash compensation **D**istance (***a*****BD*****n***) command is used to initiate backlash compensation. Parameter ***n*** 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 default units for this command parameter are encoder counts, or the units established by the **U**ser **S**cale (***a*****US*****n***) command.

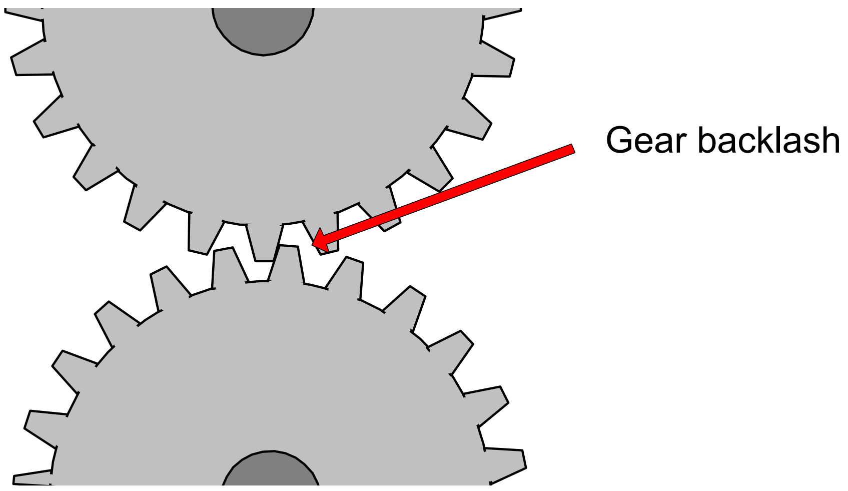
When this feature is enabled, the controller will add or subtract the backlash distance from the motor's commanded position during all subsequent moves. If the motor moves in a positive direction, the distance will be added; if the motor moves in a negative direction, it will be subtracted. When the 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 **T**ell **P**osition, **T**ell **T**arget, and **T**ell **O**ptimal commands will be adjusted to reflect the ideal positions as if no mechanical backlash was present.

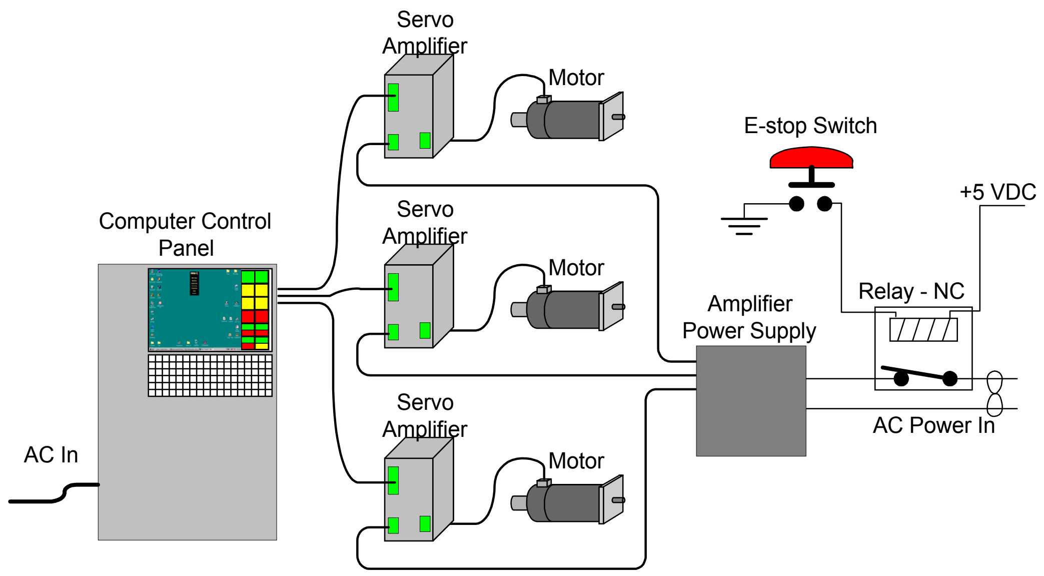
For the example below assume that the system has 200 encoder counts of backlash. This example moves the system to the middle of the backlash range and enables compensation. Note that the compensation value (in encoder counts) defined by the **B**acklash compensation **D**istance (***a*****BD*****n***) is half of the total amount of backlash.

```
1MR-100.0                         ;move to middle of backlash
1WS0.02                           ;let motion finish
1BD100                            ;enable backlash compensation
1BN                               ;enable backlash compensation

1BF                               ;disable 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.

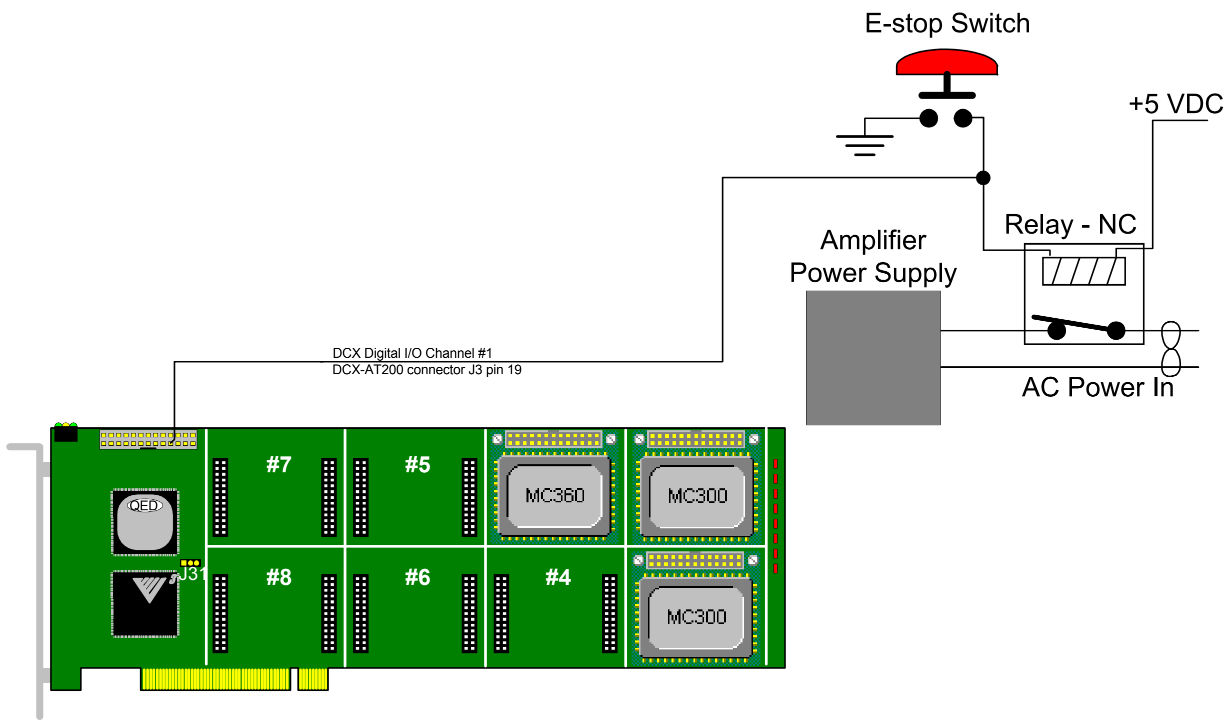


This ‘hard wired’ E-stop circuit uses a relay to disconnect power from the servo amplifiers. The motors and amplifiers would certainly be disabled, but the motion controller and the application program will have no indication that an error condition exists.

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 **W**ait for channel o**N** (**WN*x***) function will stay active in background until the input ‘goes true’.

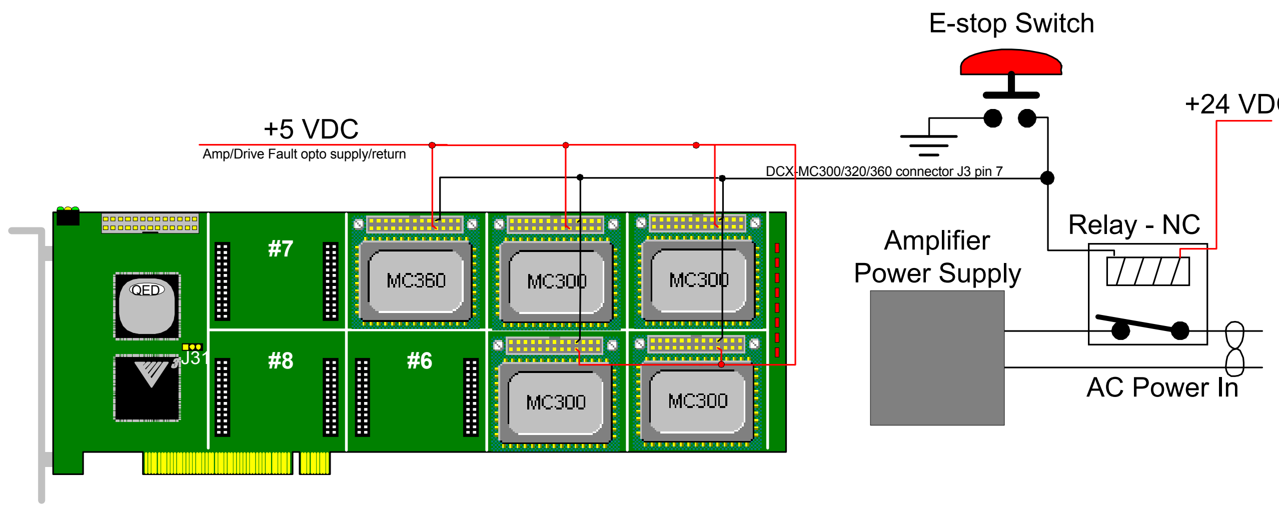


```
AL0,AR100                          ;load 0 into user register 100, this register
                                   ;will be used to indicate when an E-stop has
                                   ;occurred


GT,WN1,AL1,AR100,0MF               ;execute background task that: monitors the
                                   ;E-stop input, sets the E-stop flag register
                                   ;(100), and turns off all axes
```

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 or pin 7) of **each module**. Auto shut down of motion upon activation of the E-stop switch is enabled by the amplifier **F**ault input o**N** (***a*****FN*****n***). This function can be disabled by issuing the amplifier **F**ault input o**F**f (***a*****FF*****n***). When the E-stop switch is activated:

1) The axis is disabled (PID loop terminated, Amplifier Enable output turned off)
2) The Amplifier Fault Tripped status bit (bit 15) will be set for each axis
3) The Motor Error status bit (bit 7) will be set for each axis

When the E-stop condition has been cleared, motion can be resumed after issuing the **M**otor o**N** (***a*****MN**) with the axis specifier ***a*** set to 0..



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 (***a*VM**) move.

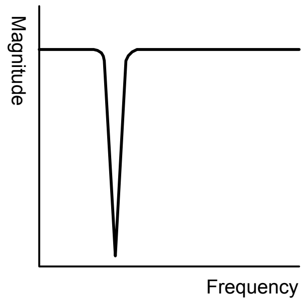
Prior to executing a velocity mode move in which the encoder position may rollover the axis **must** be homed (Find Index) to **position 0**. Defining a offset to the home position will cause the axis to pause at the rollover point.

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

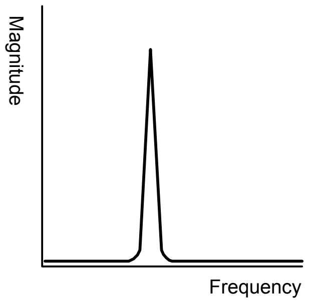
The DCX-MC302 Dual Servo Control module does not support user defined filters.

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

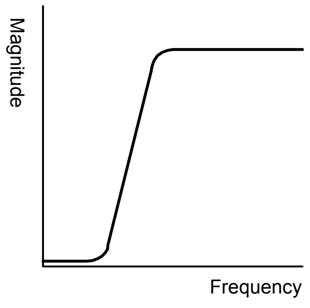
- Notch filter, otherwise known as a Band Stop filter, allows the user to define a specific frequency to be attenuated.
- Low Pass filter - removes the high frequency response of a servo system.
- High Pass filter - removes the low frequency response from the system.
- Band Pass filter - blocks both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



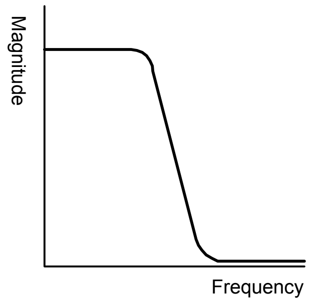
Notch Filter



Band Pass



High Pass



Low Pass

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

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

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:

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

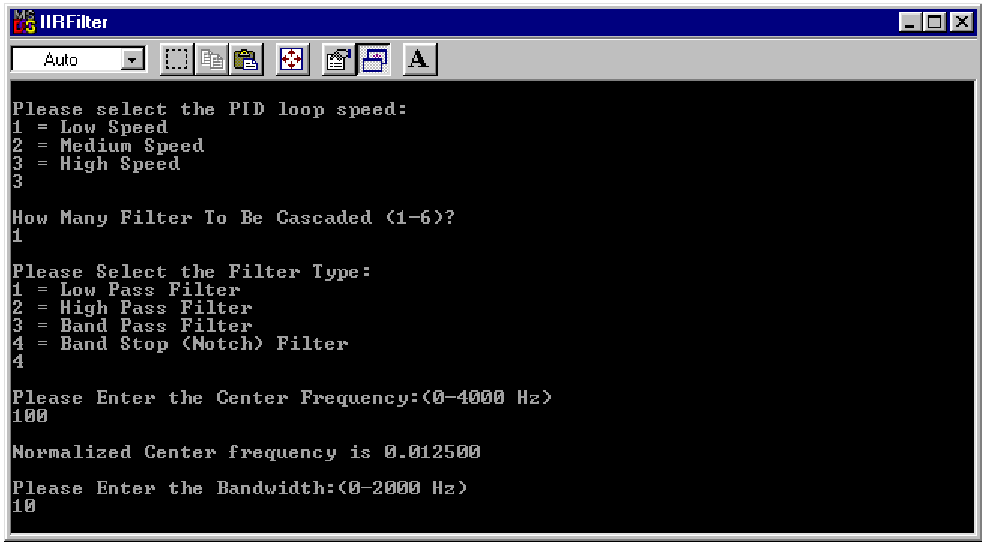
The DCX-PCI300-H supports as many as two IIR filters per servo axis. Usually this would means that the user could define two Notch filters, the controller does support combining two different filter types (Notch & Low Pass, Notch and High Pass, etc...).

The setting for 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.



```
Please select the PID loop speed:
1 = Low Speed
2 = Medium Speed
3 = High Speed
3

How Many Filter To Be Cascaded (1-6)?
1

Please Select the Filter Type:
1 = Low Pass Filter
2 = High Pass Filter
3 = Band Pass Filter
4 = Band Stop (Notch) Filter
4

Please Enter the Center Frequency:(0-4000 Hz)
100

Normalized Center frequency is 0.012500

Please Enter the Bandwidth:(0-2000 Hz)
10
```

Figure 11: 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
```

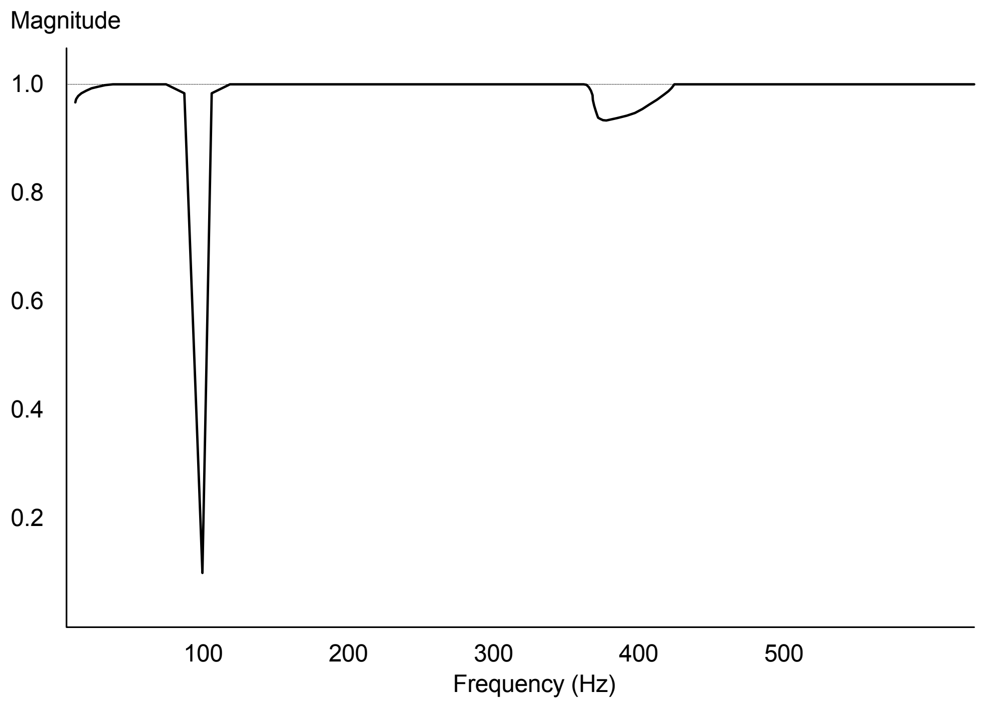


Figure 12: 100 Hz Notch filter frequency response

Download the file to the controller with WinControl. To enable the filter issue the **Y**es IIR **F**ilter command. Issue the Motor oN command to enable the PID filter.

```
1YF
1MN
```

To disable the filter issue the **N**o IIR **F**ilter (*a*NF) command.

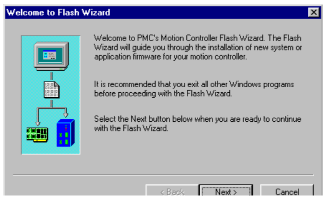
The servo will perform significantly different with the IIR filter enabled so you will need to re-tune the axis.

The Save and Restore functions of MCAPI 3.01 do not support the IIR filter parameters. Each time the controller is initialized 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.

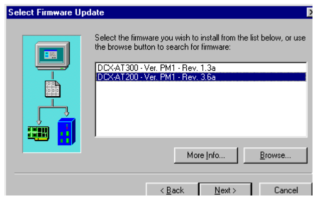
PMC's **Flash Wizard** is a windows utility that allows the user to easily update the operational code. Code updates are available from the **MotionCD** or from PMC's web site **www.pmccorp.com**.


Welcome to Flash Wizard

Welcome to PMC's Motion Controller Flash Wizard. The Flash Wizard will guide you through the installation of new system or application firmware for your motion controller.

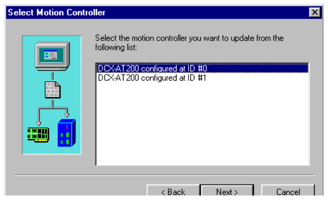
It is recommended that you exit all other Windows programs before proceeding with the Flash Wizard.

Select the Next button below when you are ready to continue with the Flash Wizard.


Select Firmware Update

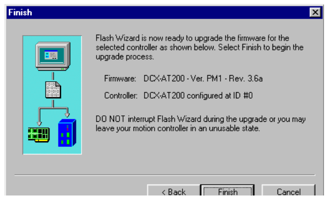
Select the firmware you wish to install from the list below, or use the browse button to search for firmware:

DCX-AT300 - Ver. PM1 - Rev. 1.3a
DCX-AT200 - Ver. PM1 - Rev. 3.6a


Select Motion Controller

Select the motion controller you want to update from the following list:

DCX-AT200 configured at ID #0
DCX-AT200 configured at ID #1

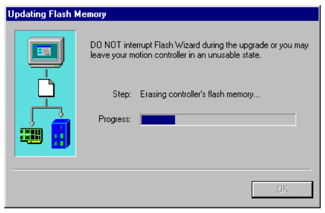

Finish

Flash Wizard is now ready to upgrade the firmware for the selected controller as shown below. Select Finish to begin the upgrade process.

Firmware: DCX-AT200 - Ver. PM1 - Rev. 3.6a

Controller: DCX-AT200 configured at ID #0

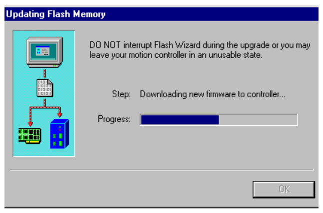
DO NOT interrupt Flash Wizard during the upgrade or you may leave your motion controller in an unusable state.


Updating Flash Memory

DO NOT interrupt Flash Wizard during the upgrade or you may leave your motion controller in an unusable state.

Step: Erasing controller's flash memory...

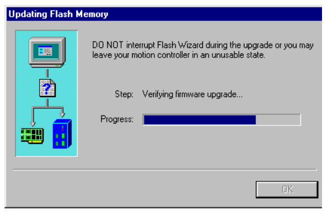
Progress:


Updating Flash Memory

DO NOT interrupt Flash Wizard during the upgrade or you may leave your motion controller in an unusable state.

Step: Downloading new firmware to controller...

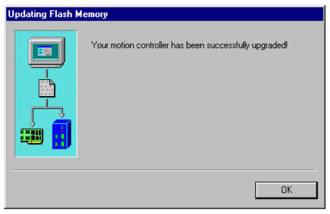
Progress:


Updating Flash Memory

DO NOT interrupt Flash Wizard during the upgrade or you may leave your motion controller in an unusable state.

Step: Verifying firmware upgrade...

Progress:


Updating Flash Memory

Your motion controller has been successfully upgraded!

Requires Flash Wizard 2.20 or higher

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 292).

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 settings by selecting **Auto Initialize** from the **File** menu.

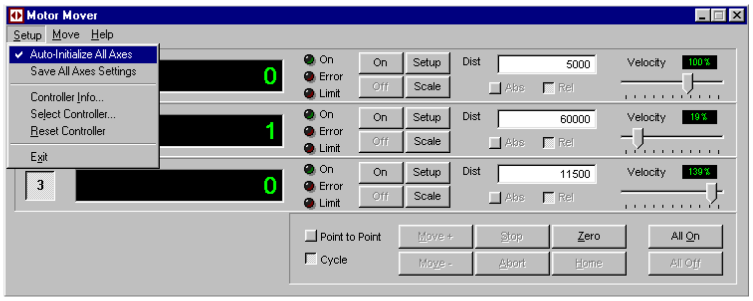


Figure 13: Launch Motor Mover with user defined controller setting by selecting Auto Initialize

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.

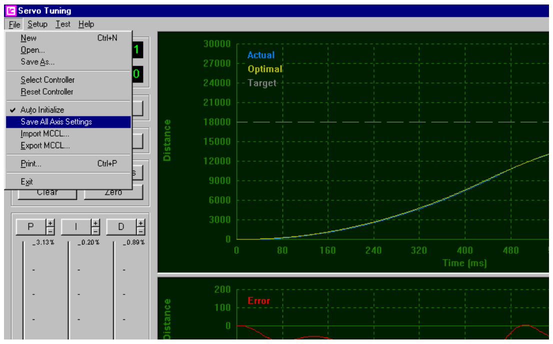


Figure 14: Saving axis settings to mcapi.ini from PMC's Servo Tuning program

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

Learning/Teaching Points

As many as 256 points can be stored for ***each axis*** in the DCX's point memory by using the **L**earn **P**osition (***a*LP*n***) and **L**earn **T**arget (***a*LT*n***) commands. The **LP** command will store the current position of an axis. The **LT** command will store the commanded target position of an axis.
For some applications, using **LT** command 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 by the **M**ove to **P**oint (***a*MP*n***) command. Parameter n indicates to which stored point the axis should move.

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

;Storing current positions

1MN                                 ;Turn on the axis
1DH0                                ;define current position as 0
1MR250,1WS0.02                      ;move and wait
1LP1                                ;store current position as learned point #1
1MR500,1WS0.01                      ;move and wait
1LP2                                ;store current position as learned point #2
1MR1000,1WS0.01                     ;move and wait
1LP3                                ;store current position as learned point #3


;Storing target positions

2MF                                 ;turn of the axis
2DH0                                ;define current position as 0
2MR100                              ;move and wait
2LT1                                ;store commanded position as learned point #1
2MR100                              ;move and wait
2LT2                                ;store commanded position as learned point #2
2MR100                              ;move and wait
2LT3                                ;store commanded position as learned point #3


;Move axes 1 and 2 through previously stored positions. Dwell for ¼ second at each
;point

1MN,2MN
1MA0,2MA0,0ws.25
0MP1,0WS.25
0MP2,0WS.25
0MP3,0WS.25
```

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 Move to Point (aMPn) 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.

```
1CM1,2CM1,3CM1
```

```
    1MP1,1MP2,1MP3                             ;move continuously through the three stored points
```

Pause and Resume Motion

Not supported at this time

The **S**ave **C**onfiguration (***a*SC*n*)** and **R**estore **C**onfiguration (***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 modules do not support Position Capture. When configured for **closed loop stepper** control the **DCX-MC360** does 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 **C**apture **B**egin (***a*CB*n***) command 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 *n* equals 0 or 1 the module will capture only one position. If parameter *n* equals 2 the module will capture two positions, and so on. When the number of positions captured = *n*, bit 11 of the axis status (Position Captured Flag) will be set. To disable position capture issue the **C**apture **B**egin command with parameter *n* expressed as a negative number. Captured positions are reported by using the **D**isplay **R**ecorded position (***a*DR*n***) command. To load the number of captured positions into the accumulator use the **C**apture **G**et count (***a*CG**) command

```
;Use the MCCL command Capture Begin to capture 10 positions

1CB10                             ;capture 10 positions
1MR10000                          ;start moving
LU"STATUS",1RL@0,IS11,BK,NO,JR-5
                                  ;loop until Position Capture Flag =1
```

Position Compare

The DCX-MC302, DCX-MC320, and DCX-MC362 modules do not support Position Compare. When configured for **closed loop stepper** control the **DCX-MC360** does 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 issue the **B**egin **C**ompare (***a*****BC*****n***) command with parameter *n* = -1. This will terminate any current compare operation and initializes the compare index to 0. Parameter ***n*** of the **L**oad Compare (***a*****LC*****n***) command is used to define the compare position. This command can be issued as many as 512 times. To initiate position compare issue the **B**egin **C**ompare command with parameter ***n*** equal to the number of defined compare positions. The Begin Compare command causes the first compare position to be loaded into the compare register. 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 Breakpoint reached flag of the axis status will be set.

Compare at incremental distances
For compare events at fixed distances of travel use the **N**ext **C**ompare (***a*****NC*****n***) command. As with al compare operations the **L**oad **C**ompare command (***a*****LC*****n***) is used to define the first compare position. The **N**ext **C**ompare command is used to define the increment between compare events. To start fixed interval compare issue the Begin Compare command with parameter ***n* = 0**. After the motors position equals the first compare position, the parameter to the Next Compare command is added to (the parameter can be a positive or negative number) the previous compare position. At each compare position the output will be activated. To disable fixed interval position compare issue the Begin Compare command with parameter ***n* = -1**.

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 Output mode for Compare command (OC, code=320). The parameter to this command selects one of the following modes:

Parameter	***Output Mode***
0*	**Disabled (default)**
1*	Static
2*	Toggle
3*	One-shot

*The active level of the output can be switched by adding a value of 128 to the OC command parameter.

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.

In the static mode, when a breakpoint command is issued (WP, WR, IP or IR), the output will go to the in-active level and remain there until the breakpoint position is reached. When the position is reached the output will go to the active level, and remain there until a succeeding breakpoint command. This output mode should not be used for the compare from memory or compare incremental functions.

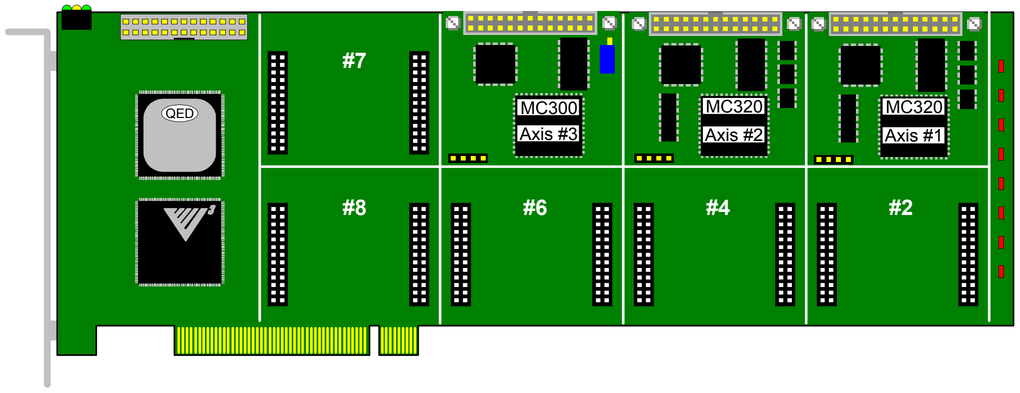
Selecting the toggle output mode will cause the output to switch between in-active and active levels with each compare or breakpoint position reached. The initial state of the output can be set to in-active by selecting the Disabled mode momentarily, and then setting it back to toggle mode. Reaching the first compare or breakpoint position will cause the output to go to the active level. At the second position or breakpoint, it will switch back to the in-active level, and so on.

Selecting the one-shot output mode will cause the compare output to generate an active level pulse of a fixed period at each compare or breakpoint position. The period of the pulse is set by issuing the Output Period command (OP, code=321). The parameter to this command is in units of seconds. The module hardware has a one-shot timer that can generate a pulse period of from 1 microsecond to 1 second. For time periods less then 50 milliseconds the timer has 1 microsecond resolution. For time periods greater than or equal to 50 milliseconds, it has 50 microsecond resolution. With either short or long duration pulses, the output signal is guaranteed to go active within the 1/2 microsecond latency of the compare function (not counting the delay for the optical isolator).

To determine how many compares have occurred, either from memory or incremental, the Get Compare count command (GC, code=325) is issued to the axis. This will cause the number to be stored in the accumulator (user register 0). Once in the accumulator, it can be displayed with the Tell Register command (TR) or used for conditional testing.

Physical Assignment of Axes Numbers

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.



Using the Use Physical (***a*UP*n***) 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 (***a*UP*n***) 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). Captured position data is typically used to analyze servo motor performance and PID loop tuning parameters. PMC's Servo Tuning utility uses the MCAPI version of this command (***MCCaptureData( )***) to capture and display the performance of a servo. The **R**ecord axis data (**aPRn**) command begins the capture of axis data for:

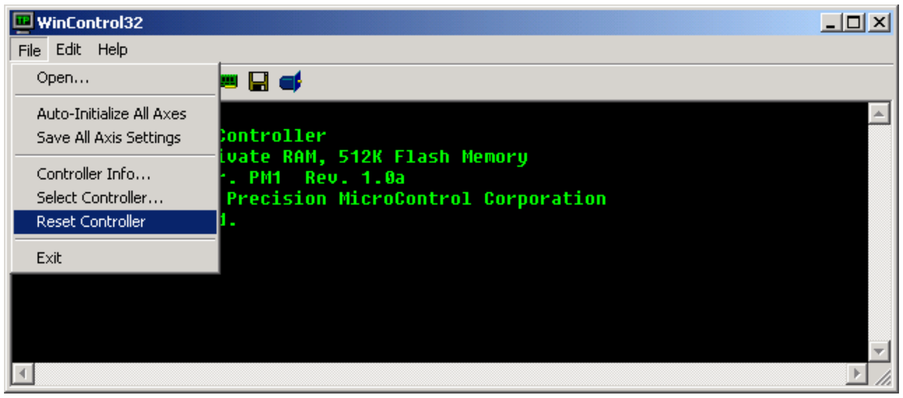
- Actual Position versus time
- Optimal Position versus time
- Following error versus time
- DAC output versus time (DCX-MC300 and MC320)

The time base (8 KHz, 4 KHz, 2 KHz) for captured data is set by High Speed, Medium Speed, or Low Speed commands. The **D**isplay **R**ecorded position (***a*DR*n***), **D**isplay **O**ptimal position (***a*DO*n***), and the **D**isplay **R**ecorded DAC output (***a*DQ*n***) commands are used to report axis data. For C/C++ application programs the function ***MCGetCapturedData( )*** is used to retrieve the captured data. The following example captures 100 data points from axis 3, then displays the first 10 captured positions.

```
3DH0
3MA5000,3PR100,3WS.1

AL0,AR100                         ;define register #100 as recorded position
                                  ;pointer

1DR@100,AL@100,AA1,AR100,RP9      ;display the first 10 recorded positions
```

Resetting the DCX

The DCX supports software controlled reset. The **R**ese**T** (***a*****RT**) command can be used to reset a specific axis (***a*** = axis number). To reset the entire DCX-PCI300 motion controller, select Reset Controller from the WinControl File menu. 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.



```
ontroller
ivate RAM, 512K Flash Memory
. PM1  Rev. 1.0a
 Precision MicroControl Corporation
1.
```

Figure 15: Resetting the DCX-PCI 300

Resetting the DCX-PCI300 will cause the controller to revert to default setting (PID, vel/accel/decel, limits, etc. For information restoring the user defined settings please refer to the **Initializing and Restoring Controller Configuration** section in this chapter.

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

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.

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 (2000:1)
3US2000.0;

;Use the MCCL command Set Master to configure axis 3 as a slave to axes 1 and 2.
3SM33

1MN,2MN,3MN

;Execute 1st linear move
1CM1,2CM1

;Linear move, first side of triangle
CP1,1MA1000,2MA0

; wait for end of contour move, lift blade, rotate blade, lower blade
0WS0.1
4MR1000,4WS0.1
3MR333,3WS0.1
4MR-1000,4WS0.1

;Linear move, second side of triangle
CP1,1MA500,2MA1000

; wait for end of contour move, lift blade, rotate blade, lower blade
0WS0.1
4MR1000,4WS0.1
3MR333,3WS0.1
```

```
4MR-1000,4WS0.1

;Linear move, third side of triangle
CP1,1MA0,2MA0

;wait for end of contour move, lift blade, rotate blade, lower blade
0WS0.1
4MR1000,4WS0.1
3MR333,3WS0.1
4MR-1000,4WS0.1

;!!! now disable tangential knife control !!!
3SM0,3PM,3MN
```

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. This mode of threading uses the encoder index mark of the master axis to trigger motion of the slave.

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 **S**et **M**aster 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
2US2000

;Set scaling of the slave axis
1US4000

1MA0,1WS0.1                      ;move slave to starting position

;Set the slave ratio. This is the lead or pitch when cutting a thread.
2SS0.1

;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
2SM258

;Set the target position. This is the position at which slave mode is ;terminated and
axis #1 will stop.
1MA0

;Start master axis moving in torque mode.
2SQ0,2QM,2SQ3.0

;Arm the index capture of the master axis. When the index pulse occurs, the
```

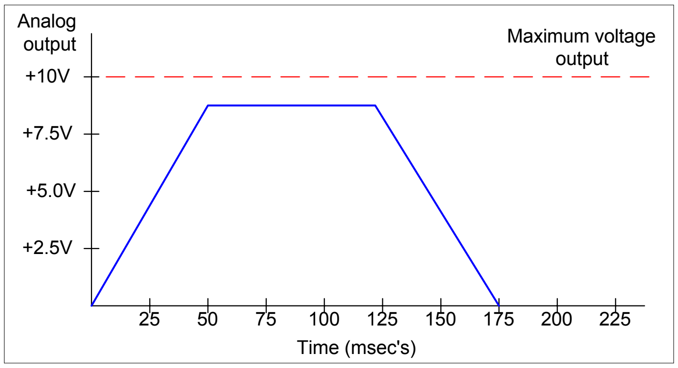
```
;slave will begin tracking the master axis until // the slave reaches its
;target position.
2IA

;This command sequence will repeat until auxiliary status bit 22 is clear,
;indicating that the slave has reached its target.
LU"AUXSTAT",1RL@0,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

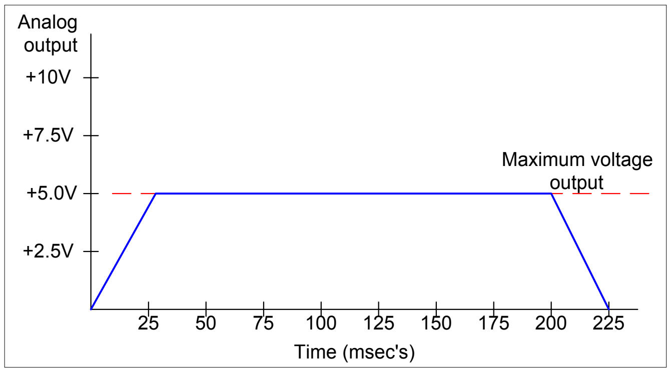
Torque Mode Output Control

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 **S**et tor**Q**ue (***a*SQ*n***) command allows the user to limit the output signal to a specific level. The following diagram depicts a simple position mode move of 1000 encoder counts with the default torque setting of 10 volts (no limit).



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

```
1SQ5.0
1MR1000
```



Servo Modules as simple D/A output with encoder reader
The tor**Q**ue **M**ode (***a*****QM*****n***) command 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.

```
1SQ0                                  ;set DAC output to 0 volts
1QM                                   ;enable torque mode
1SQ2.5                                ;axis 1 output to 2.5V (MC300)
2SQ7.5                                ;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 defined as:

The integral term **accumulates** the position error for servos and generates an output signal to reduce the position error to zero.

For a typical servo system, 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 description in the DCX-PCI300 User's Manual
- the Servo tuning tutorials on PMC's MotionCD

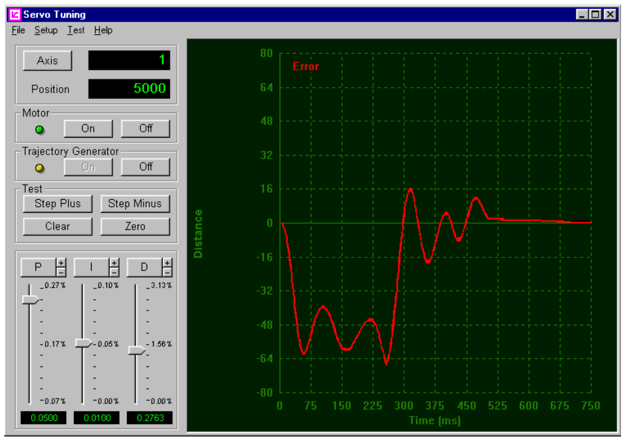
For some applications, integral gain has a tendency to cause bounce or oscillation of the command signal during a move. This tendency can be is especially problematic in:

- Systems with high and or irregular friction
- Systems with unbalanced loads
- Systems with unbalanced and or high offset amplifiers

The Integral Option (***a*****IO*****n***) command provides the user with the option of determining how the PID filter will use integral gain.

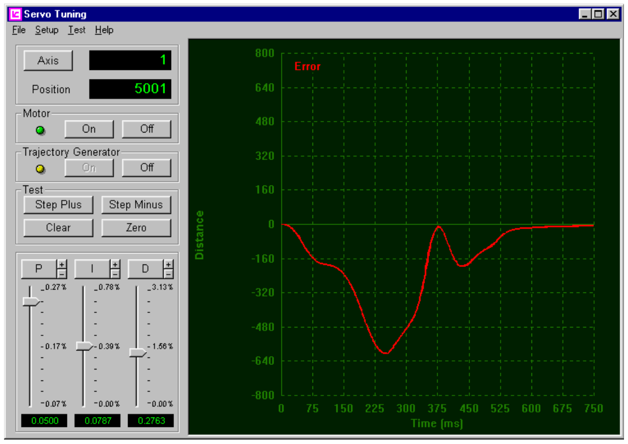
n =	***Integral term function***	***Notes – (all other servo parameters remaining unchanged)***
0	**Integral term always on (default)**	Smallest following error during move. As the integral term is increased the command output / following error will tend to bounce
1	Freezes accumulation of integration 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)
2	Zero integration term when motion begins.	Most stable command signal / servo

	When the calculated trajectory (trajectory complete, status bit 3 = 1) has completed, enable the integration term	performance during the move. Largest following error during the move. Not acceptable for applications with unbalanced load.



Axis 1
Position 5000
P 0.0500
I 0.0100
D 0.2763

With Integral turned on all the time the following error (red trace) is relatively small, but not extremely stable



Axis 1
Position 5001
P 0.0500
I 0.0787
D 0.2763

With Integral turned off until the move is complete the following error (red trace) is much larger but not much more stable

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.

Setting Move (Encoder/Step) Units
The User Scale (aUSn) command is used to define 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:

```
3US1000
3MN
```

Prior to issuing the **US** command the parameters to all motion commands for a particular axis are rounded to the nearest integer. A new encoder/step count scales is defined by the **US** command but does not take effect until the **M**otor o**N** command has been issued. Motion targets are multiplied by the ratio prior to rounding to determine the correct encoder position. Issuing the **T**ell **P**osition command will report 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 **U**ser **R**ate (***a*****UR*****n***) command sets the time unit for velocity, acceleration and deceleration values. The DCX defaults to counts/steps per second. If velocities are to be in units of inches per minute, the user time unit is a minute. Parameter ***n*** of the **UR** command 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 parameter ***n*** should be set to 60 seconds/1 minute = 60 (1UR60). The Motor oN command must be issued before the user rate will take effect.

```
        3UR60
        3MN
```

Typical Rate values

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

Defining the Time Base for Wait commands
The default unit for dwell commands (**WA**it, **W**ait for **S**top, **W**ait for **T**arget) is seconds. The **U**ser **T**ime (**UT*n***) command allows the user to change the units for dwell commands. Parameter **n** is the number of 1 second periods in the user's unit of time. If the user prefers time parameters in units of minutes, parameter n = 60 should be issued.

```
    1UT60
```

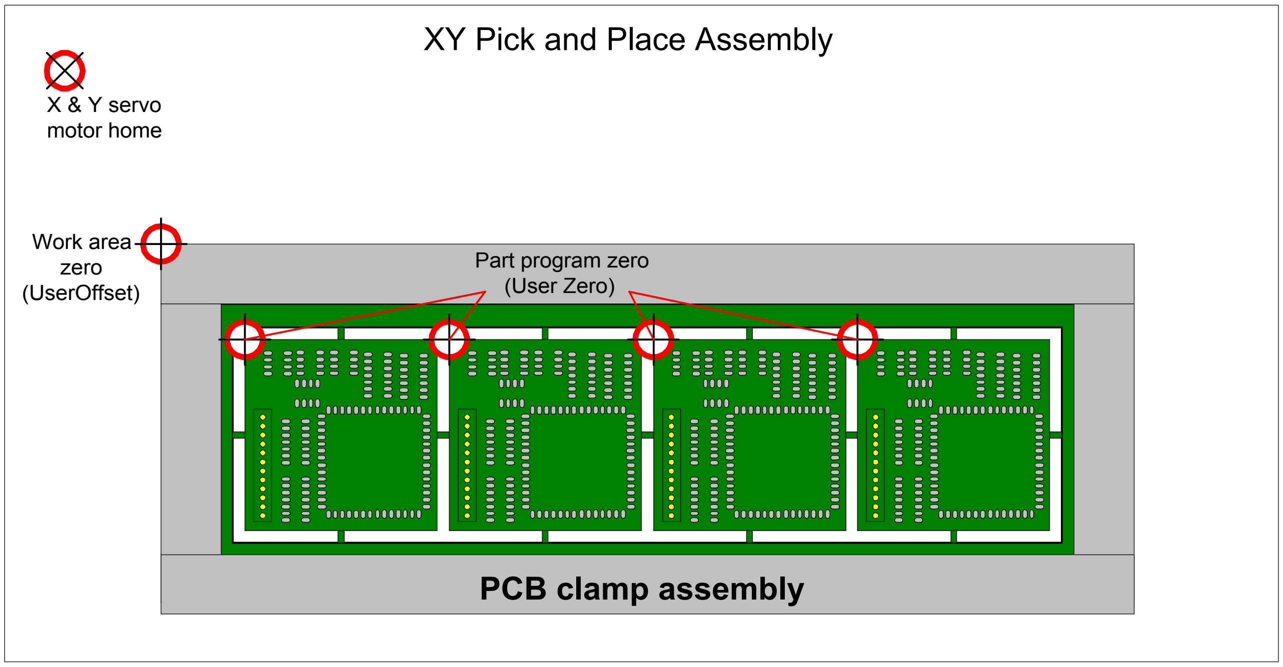
The **UT** command only effects the dwell time in the **task or command interface** from which it was issued. Each command interface (Binary, ASCII) and tasks maintain separate dwell time settings.

Defining a System/Machine zero
The **U**ser **O**ffset (***a*UO*n***) command allows the user to define a 'work area' zero position of the axis. Use parameter ***n*** to define 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 **U**ser **S**caling command. The **U**ser **O**ffset command does not change the index or home position of the servo or stepper motor, it only establishes an arbitrary zero position for the axis.

```
        3UO12.25                        ;define user = 12.25 inches
```

Defining a Part Zero
The **U**ser **Z**ero (***a*UZ*n***) command would typically be used in conjunction with the **U**ser **O**ffset 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 **UO** command is used to define the 'work area' zero of the PCB. The **UZ** command 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.

```
    3UO12.25                        ;define offset to 12.25 inches
    3UZ1.25                         ;define 'part zero' to 1.25 inches
```


XY Pick and Place Assembly
X & Y servo motor home
Work area zero (UserOffset)
Part program zero (User Zero)
PCB clamp assembly

Defining the output constant for velocity gain
The **U**ser output **K**onstant (***a*UK*n***) command allows the user to define the units to be used for setting the **V**elocity **G**ain (feed forward) parameters.

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 6 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 5

General Purpose I/O

DCX Motherboard Digital I/O

The DCX-PCI300 Motion Controller motherboard has 16 undedicated 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 can be configured via software as high true or low true (default = 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.

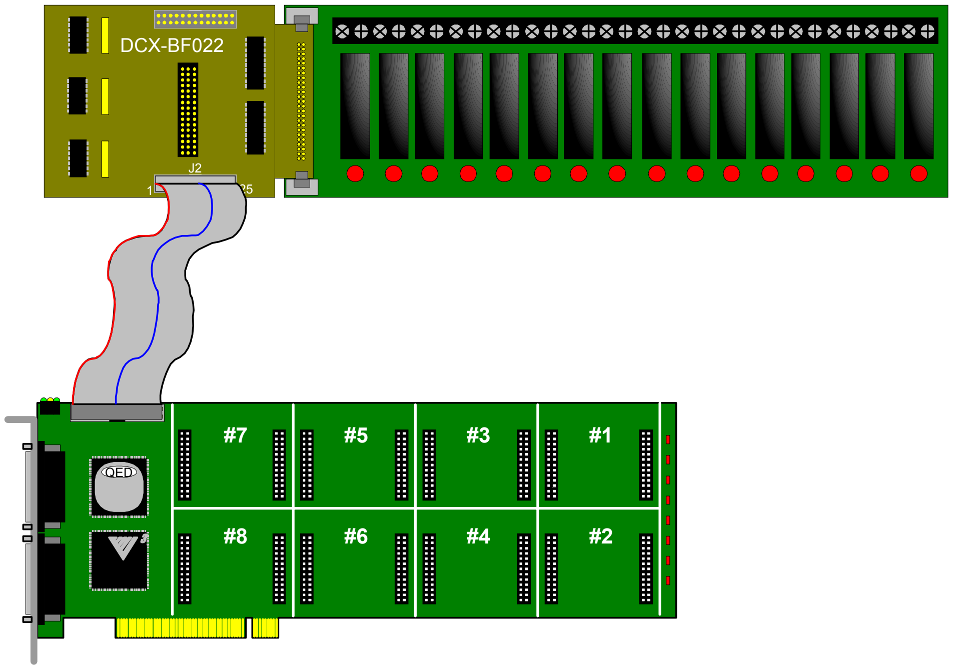


Figure 16: **A DCX-BF022 is used to interface DCX digital I/O to an OPTO22 relay rack**

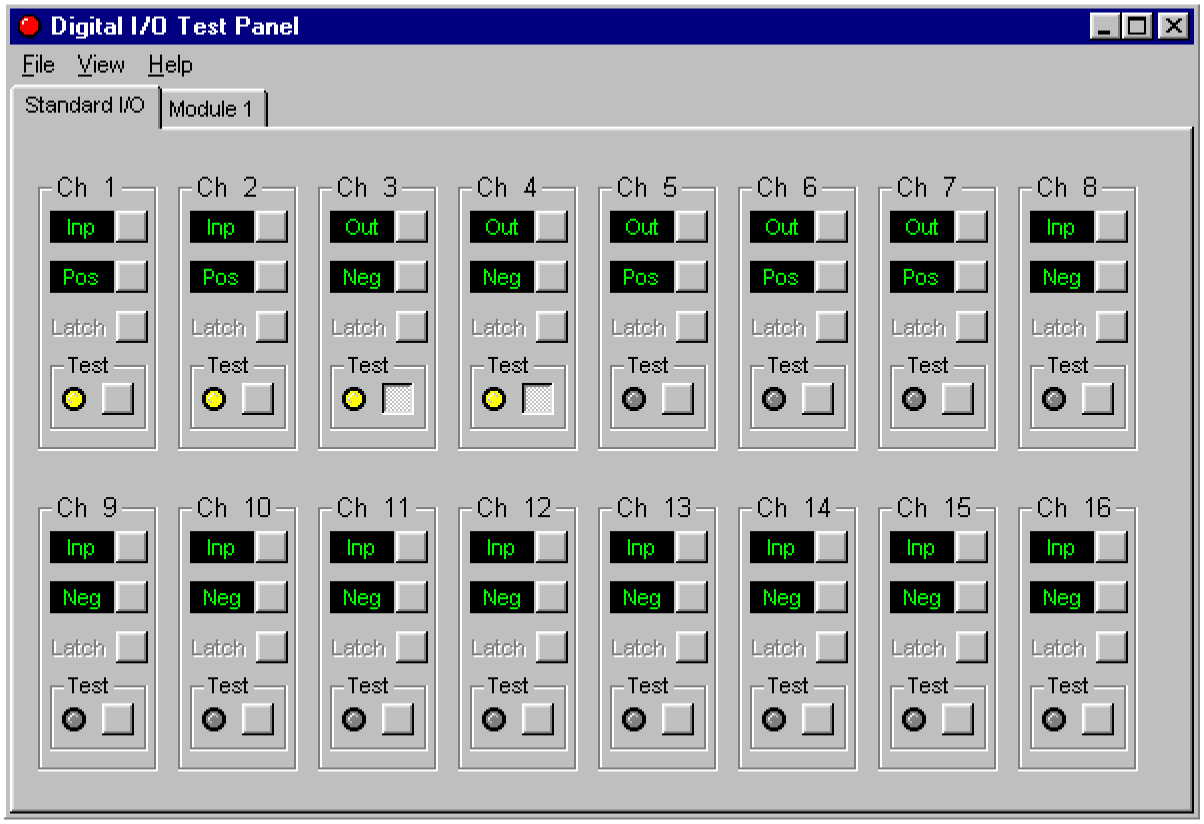
As shown above, the DCX-BFO22 plugs directly into the relay rack's 50 pin header connector and then connects to the DCX-PCI300 via a 26 conductor ribbon cable. Note that the relays are numbered sequentially starting from 0, while the DCX digital I/O channels are numbered sequentially starting with 1.

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-MC400 digital I/O channels is accomplished using either PMC's Motion Integrator software or the **C**hannel **H**igh (**CHx**) and/or **C**hannel **L**ow (**CLx)** commands. 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:

High true/Positive logic or Low true/Negative logic

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.

Using the DCX Digital I/O

After configuring the digital I/O the following MCCL commands are available for activating and monitoring the digital I/O:

CF Channel oFf

MCCL command:	CF*x* *x* = Channel number
compatibility:	MC400
see also:	CN

Causes channel x to go to "off" state. If the channel has been configured for "high true", the channel will be at a logic low (less that 0.4 volts DC) after this command is executed. If it has been configured for "low true", the channel will be at a logic high (greater than 2.4 volts DC).

CH Channel High

MCCL command :	CH*x* *x* = Channel number or 0
compatibility:	MC400
see also:	CL

Causes digital I/O channel x to be configured for "high true" logic. This means that the I/O channel will be at a high logic level (greater than 2.4 volts DC) when the channel is "on”, and at a low logic level (less than 0.4 volts DC) when the channel is "off". Note that issuing this command will not cause the I/O channel to change its current state. Issuing this command without specifying a channel will cause all channels present on the DCX to be configured as "high true". If parameter *x* = 0 all digital I/O channels will be configured for high true logic.

CI Channel In

MCCL command:	CI*x* *x* = Channel number
compatibility:	MC400
see also:	CT

Used to configure digital I/O channel x of a MC400 module as an input. All digital I/O channels on the MC400 default to inputs on power-on or reset. If they are subsequently changed to outputs with the Channel ouT command, they can be returned to inputs with the Channel In command. The state of a digital I/O channel can be viewed with the Tell Channel command.

CL Channel Low

MCCL command :	CL*x* *x* = Channel number or 0
compatibility:	MC400
see also:	CH

Causes digital I/O channel x to be configured for "low true" logic. This means that the I/O channel will be at a low logic level (less than 0.4 volts DC) when the channel is "on", and at high logic level (greater than 2.4 volts DC) when the channel is "off". Note that issuing this command will not cause the I/O channel to change its current state. Issuing this command without specifying a channel will cause all channels present on the DCX to be configured as "low true". If parameter *x* = 0 all digital I/O channels will be configured for low true logic.

CN Channel oN

MCCL command: CN*x* *x* = Channel number
compatibility: MC400
see also: CF

Causes channel x to go to "on" state. If the channel has been configured for "high true", the channel will be at a logic high (greater than 2.4 volts DC) after this command is executed. If it has been configured for "low true", the channel will be at a logic low (less that 0.4 volts DC).

CT Channel ouT

MCCL command: CT*x* *x* = Channel number
compatibility: MC400
see also: CI

Used to configure digital I/O channel x of a MC400 module as an output. The DCX will turn the channel "off" before changing it to an output.

DF Do if channel oFf

MCCL command: DF*x* x = Channel number

Used for conditional execution of commands. If digital I/O channel *x* is "off", commands that follow on the command line or in the macro will be executed. Otherwise the rest of the command line or macro will be skipped.

```
DF2,1MR1000                         ;If channel 2 is off move 1000
```

DN Do if channel ‘x’ is oN

MCCL command: DN*x* x = Channel number

Used for conditional execution of commands. If digital I/O channel *x* is "on", commands that follow on the command line or in the macro will be executed. Otherwise the rest of the command line or macro will be skipped.

```
DN2,1MR1000                         ;If channel 2 is off move 1000
```

IF If channel oFf do next command, else skip 2 commands

MCCL command: IF*x* x = Channel number

Used for conditional execution of commands. If digital I/O channel *x* is "off", command execution will continue with the command following the IF command. Otherwise the two commands following the IF command will be skipped, and command execution will continue from the third command. See the description of **Digital I/O** in the **DCX General Purpose I/O** chapter.

```
IF5,MJ10,NO,MJ11                        ;If digital input #5 is off jump to
                                        ;macro 10, otherwise jump to macro 11
```

IN If channel ’ oN do next command, else skip 2 commands

MCCL command: IN*x* x = Channel number

Used for conditional execution of commands. If digital I/O channel *x* is "on", command execution will continue with the command following the IN command. Otherwise the two commands following the IN command will be skipped, and command execution will continue from the third command. See the description of **Digital I/O** in the **DCX General Purpose I/O** chapter.

```
        IN5,MJ10,NO,MJ11                              ;If digital input #5 is on jump to
                                                      ;macro 10, otherwise jump to macro 11
```

Tell **C**hannel — TC

MCCL command: TC*x*   *x* = Channel number or 0
compatibility: MC400
see also:

Reports the on/off status of each digital I/O line. This data is reported separately for each channel. The DCX responds by displaying the channel number and a "1" if the channel is "on", or a "0" if the channel is "off". If parameter *x* = 0 the state of all digital I/O channels will be reported.

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 Analog I/O Module all analog input channels are numbered sequentially as a group. Likewise, all analog output channels are numbered sequentially as a group. 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.

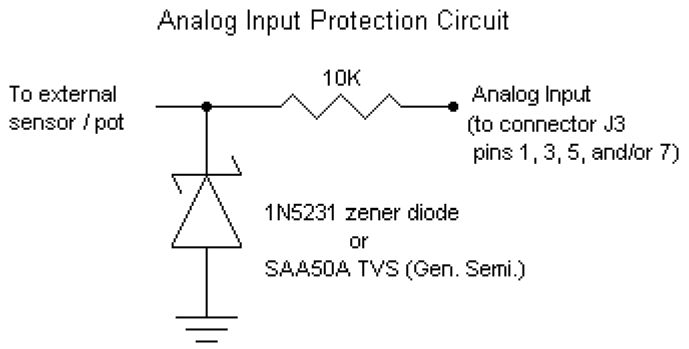
Because the DCX controller board is implemented in digital electronics, all analog input signals must be converted into a representative numerical value. This function is done by an Analog to Digital Converter (ADC) on the DCX-MC500. Similarly, analog output signals originate on the DCX board as numerical values. These numbers must be written to a Digital to Analog Converter (DAC) on the DCX-MC500, 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 DCX-MC500/MC510 analog input channels. 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.


Analog Input Protection Circuit
To external sensor / pot
10K
Analog Input (to connector J3 pins 1, 3, 5, and/or 7)
1N5231 zener diode
or
SAA50A TVS (Gen. Semi.)

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.

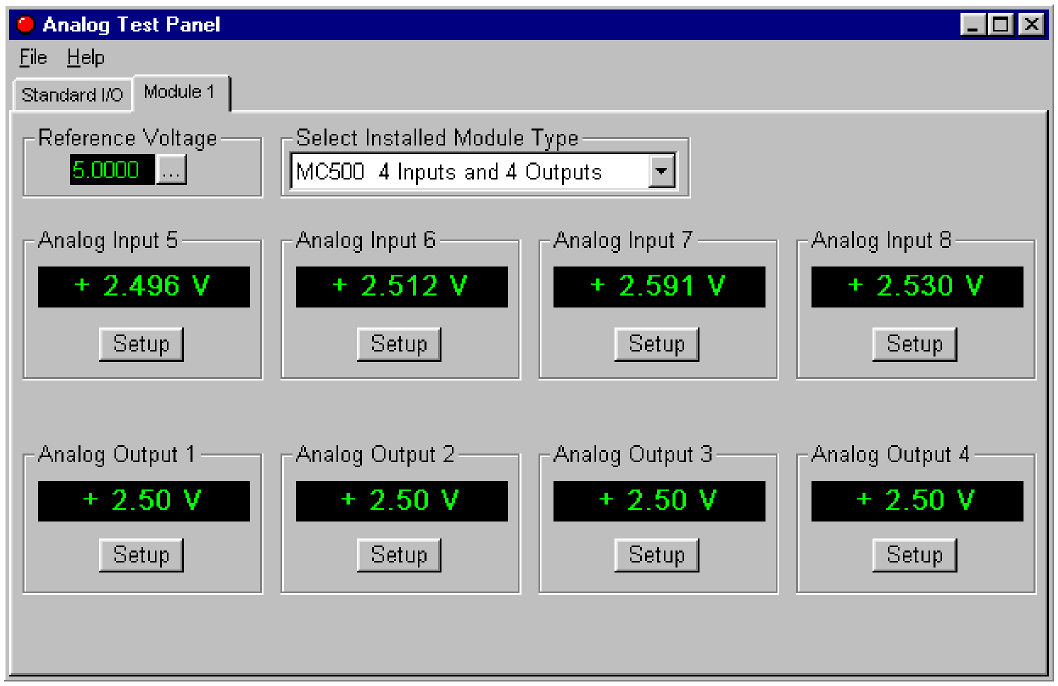
The outputs from the DAC on the DCX-MC500 module are voltage levels in the range 0 to +5 volts. These outputs have no gain or offset adjustment. These signals are available on pins 10, 12, 14 and 16 of the module J3 connector. For a complete signal pinout please refer to the chapter titled **Connectors, Jumpers, and Schematics**.

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 MCCL commands. 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.



Reference Voltage: 5.0000

Select Installed Module Type: MC500 4 Inputs and 4 Outputs

Analog Input 5: + 2.496 V
Analog Input 6: + 2.512 V
Analog Input 7: + 2.591 V
Analog Input 8: + 2.530 V

Analog Output 1: + 2.50 V
Analog Output 2: + 2.50 V
Analog Output 3: + 2.50 V
Analog Output 4: + 2.50 V

The following MCCL commands functions are available for setting and monitoring the MC500 analog I/O:

GA Get Analog

MCCL command: GA*x* x = Channel number
compatibility: MC500, MC520
Performs analog to digital conversion on the specified input channel and places the result into the Accumulator (User Register 0). Analog channels are numbered starting with 1.

OA Output Analog

MCCL command: OAx x = Channel number
compatibility MC500, MC520
Sets the specified analog output channel to the value stored in the Accumulator (User Register 0). The analog output channels on any installed MC500 modules are numbered consecutively starting with channel 1. The contents of the Accumulator should be in the range 0 to 4095.

TA Tell Analog

MCCL command: TAx *x* = Channel number *p* = 0, 1, 2, 3, ... (# of MC500/510 modules X 4)

compatibility: MC500, MC510

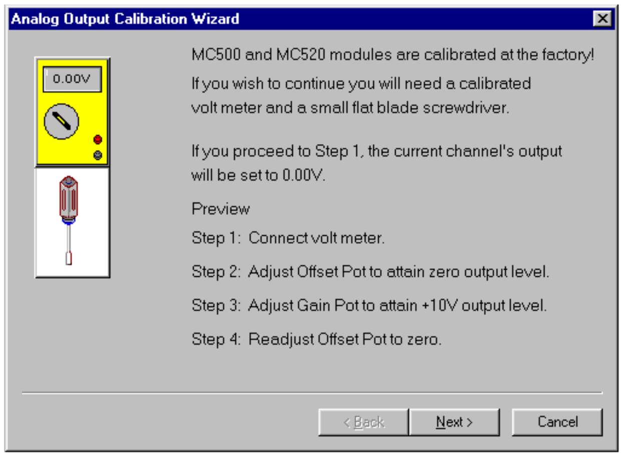
see also:

Reports the digitized analog input signals to MC500 and MC510 modules. The analog input channels on any installed MC500/510 modules will be numbered sequentially starting with channel 1. For each of these channels, the TA command will display a number between 0 and 4096. These numbers are the ratio of the analog input voltage to the reference input voltage multiplied by 4096.

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 which 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**.



Analog Output Calibration Wizard

MC500 and MC520 modules are calibrated at the factory!

If you wish to continue you will need a calibrated volt meter and a small flat blade screwdriver.

If you proceed to Step 1, the current channel's output will be set to 0.00V.

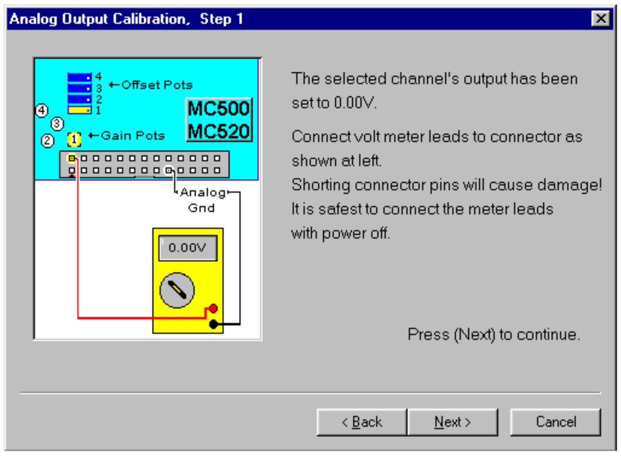
Preview

Step 1: Connect volt meter.

Step 2: Adjust Offset Pot to attain zero output level.

Step 3: Adjust Gain Pot to attain +10V output level.

Step 4: Readjust Offset Pot to zero.



Analog Output Calibration, Step 1

The selected channel's output has been set to 0.00V.

Connect volt meter leads to connector as shown at left.

Shorting connector pins will cause damage! It is safest to connect the meter leads with power off.

Press (Next) to continue.

The analog outputs can also be calibrated using MCCL command sequences. 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

Chapter
6

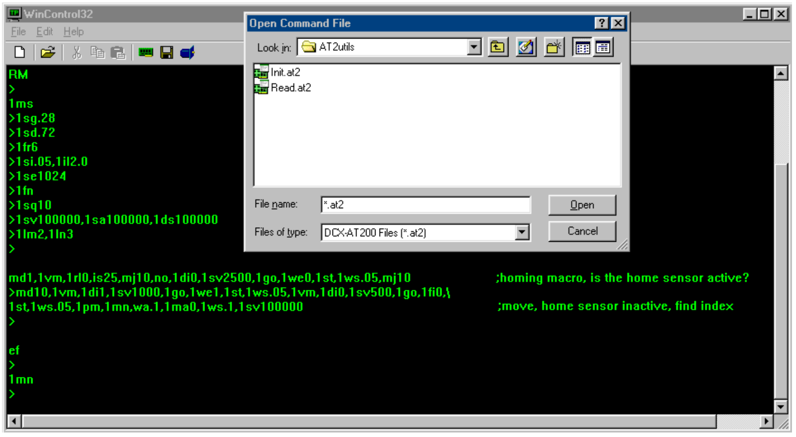
Working with MCCL Commands

Downloading MCCL Text Files

Motion Control Command Language (MCCL) command sequences can be downloaded as text files to the DCX-PCI300. If these command sequences are not defined as macro's (**MD*n***) then the commands will be executed as they are received by the card. If the command sequences are defined as macro's they will be stored in the memory of the DCX-PCI300 for execution at a later time.

While most motion control applications will utilize the high level language (C++, VB, Delphi, LabVIEW, etc..) function calls to program the operation of the machine, downloaded MCCL text files are typically used for initial system integration, defining homing routines, and programming background tasks.

The following graphic is a screen capture of PMC's WinControl . This utility provides the user with a direct interface to the DCX., A MCCL text file (init.AT2) containing servo parameters and a homing routine have been downloaded to the DCX using the File – Open menu options.



```
RM
>
1ms
>1sg.28
>1sd.72
>1fr6
>1si.05,1il2.0
>1se1024
>1fn
>1sq10
>1sv100000,1sa100000,1ds100000
>1lm2,1ln3
>

md1,1vm,1rl0,is25,mj10,no,1di0,1sv2500,1go,1we0,1st,1ws.05,mj10          ;homing macro, is the home sensor active?
>md10,1vm,1di1,1sv1000,1go,1we1,1st,1ws.05,1vm,1di0,1sv500,1go,1fi0,\
1st,1ws.05,1pm,1mn,wa.1,1ma0,1ws.1,1sv100000                              ;move, home sensor inactive, find index
>

ef
>
1mn
>
```

Figure 17: Downloading a MCCL text file via WinControl

Note: Any characters that are preceded by a semicolon are treated as documenting commands. These documenting character strings are displayed by WinControl but they are 'stripped' from the file and are not be passed to the DCX.

Building MCCL Macro Sequences

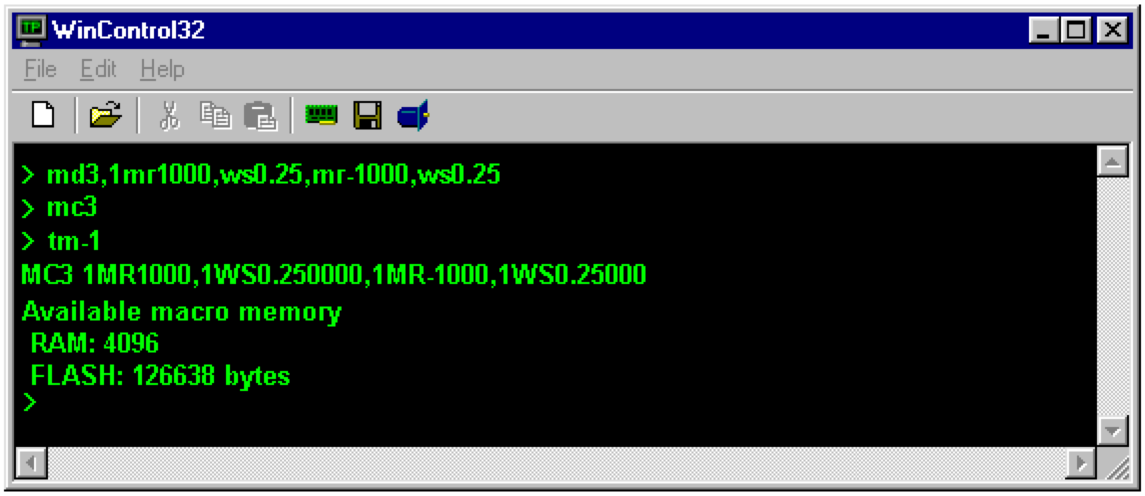
A powerful feature of the DCX is the ability to define MCCL (Motion Control Command Language) command sequences as macros. This simply means defining a mnemonic that will execute a user defined sequence of commands. For example:

```
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) 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.



```
> md3,1mr1000,ws0.25,mr-1000,ws0.25
> mc3
> tm-1
MC3 1MR1000,1WS0.250000,1MR-1000,1WS0.25000
Available macro memory
 RAM: 4096
 FLASH: 126638 bytes
>
```

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**.

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.

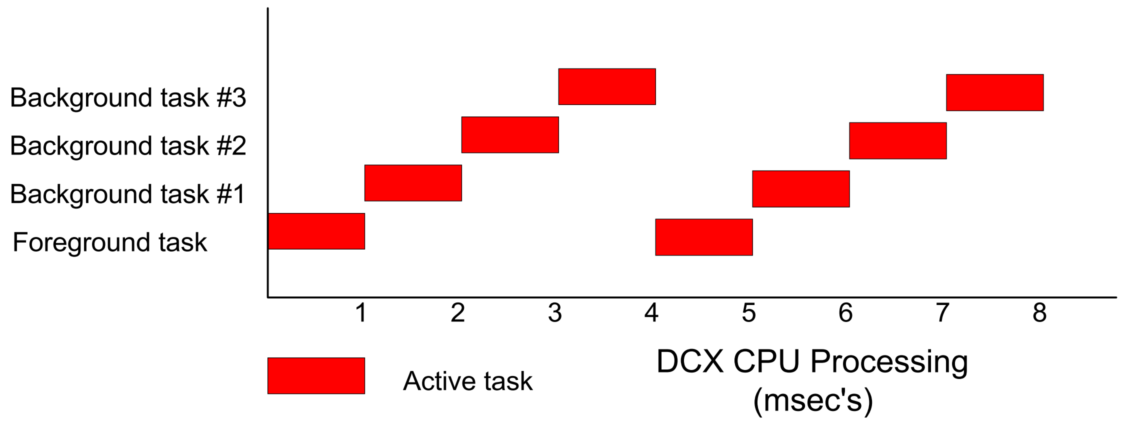
Another way to create a background task is to place the Generate Task command as the first command in a command line, using a parameter of 0. This instructs the command interpreter to take all the commands that follow the Generate Task command and cause them to run as a background task. The commands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

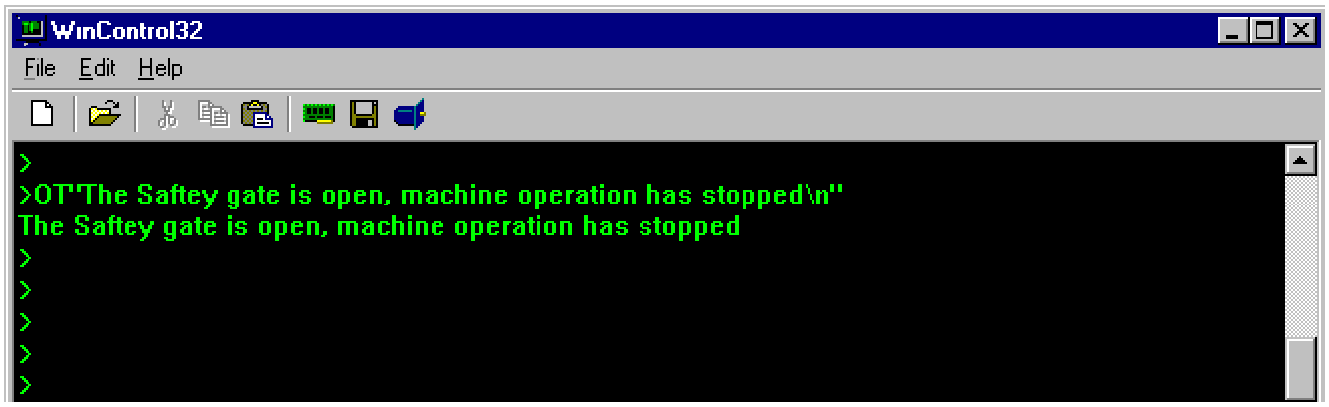
Outputting Formatted Message Strings

The DCX supports the outputting of formatted text strings from the ASCII interface using the Output Text commands. The two commands supported are:

Output Text with integer values (OT"   ")
Output text with Double values (OD"   ")

The syntax of these two commands are patterned after standard 'C' function 'printf'. For specific 'printf' description please refer to the Microtech Research Inc. MCC960 compiler documentation. The message to be displayed should be delimited by double quotes. Please refer to the examples below:

```
OT"The Safety gate is open, machine operation has stopped \n"
                                        ;output simple text message,
                                        ; \n = line feed
```



```
>
>OT"The Saftey gate is open, machine operation has stopped\n"
The Saftey gate is open, machine operation has stopped
>
>
>
>
>
```

As with typical implementations of 'C' print statements, the DCX supports variables. Prior to executing the output text command, load the accumulator with the data to be included as a variable. In the following example, the Output Double (OD" ") command is used to report the current position of axis one as a floating point value. The % character indicates that a variable stored in the accumulator will be included in the text message. The 'f' indicates that the variable is a floating point value. The '\r' calls for a carriage return at the end of the message, \n calls for a line feed.

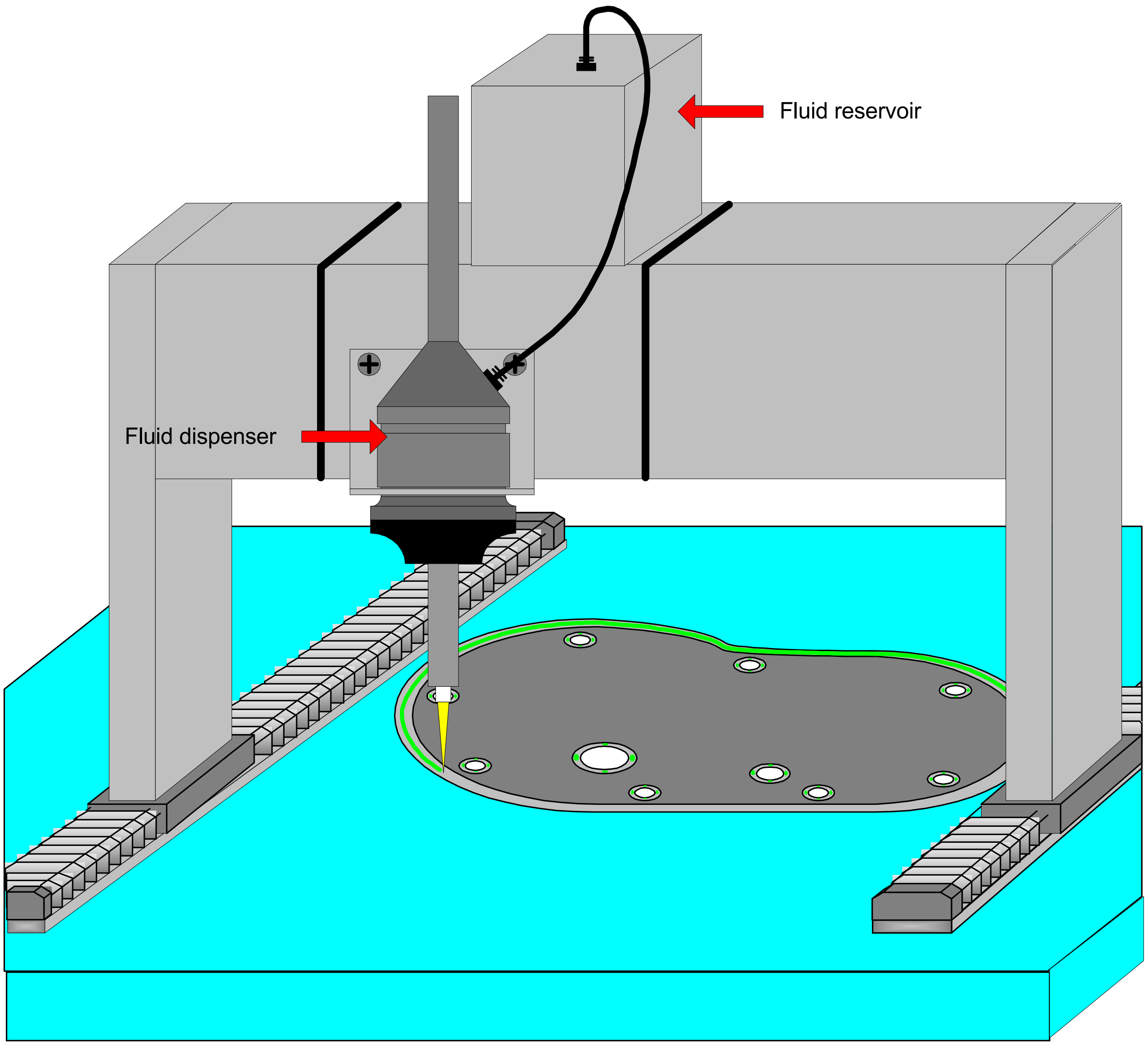
```
1RD20,OD"The current position of Axis #1 %f \n"
                                        ;load the accumulator with the
                                        ;position of axis #1. Output a text
                                        ;message displaying the position of
                                        ;axis #1 (floating point value),
                                        ;carriage return
```

To output integer variable use %d. For a line feed versus a carriage return us \n.

PLC I/O Control using MCCL Sequence Commands

PLC control and DCX Digital I/O
The following graphic depicts a fluid dispensing system. A liquid adhesive is applied to a gasket. Two linear axes are mounted to the table top (see the slides with bellows) and are slaved together to move the dispensing head back and forth. The fluid dispensing head is moved left and right by a third axis. For this application there is no Z (up and down) axis motion, the operator manually positions the dispense head for the proper height. The following Digital I/O are used dispensing the liquid:

Inputs:	Outputs:
Fluid reservoir empty (dig. I/O #1)	Turn on the dispense valve air supply (dig. I/O #4)
Dispense valve empty (dig. I/O #2)	Dispense liquid – turn on Archimedes motor (dig I/O #5)
Valve busy being primed (dig. I/O #3)	Stop dispense – reverse Archimedes motor (dig I/O #6)
	Prime the valve (dig I/O #7)



A ‘PC based’ application program is used by the operator to initialize and operate the dispensing system. The ‘Fluid Reservoir’ and ‘Dispense Valve” sensors are monitored by MCCL macro’s executing as background tasks. If either of these sensors ‘goes active’, a DCX User Register flag will be set, The application program will then notify the operator to remedy the error condition. The following macro sequence will monitor the state of the two sensors:

```
MD300,IF1,MJ301,NO,IF2,MJ302,NO,WA.1,JR-7
                                        ;pole sensors for error condition
MD301,1VO0,AL1,AR201,OT”The fluid reservoir level is low, add fluid and prime
the dispense valve \n”                  ;stop the motion (Velocity
                                        ;Override =0), set Input Sensor Error
                                        ;Flag, output ASCII error message
MD302,1VO0,AL2,AR201,OT”The dispense valve fluid level is low, add fluid and
prime the dispense valve \n”            ;stop the motion (Velocity
                                        ;Override =0), set Input Sensor Error
                                        ;flag, output ASCII error message

GT300,AR200                             ;execute sensor monitoring macros as a
                                        ;background task. Load the task ID in
                                        ;register #200.


AL0,AR201,1VO100,GT300,AR200            ;after error condition cleared, resume
                                        ;operation
```

Prior to initiating a dispensing operation:

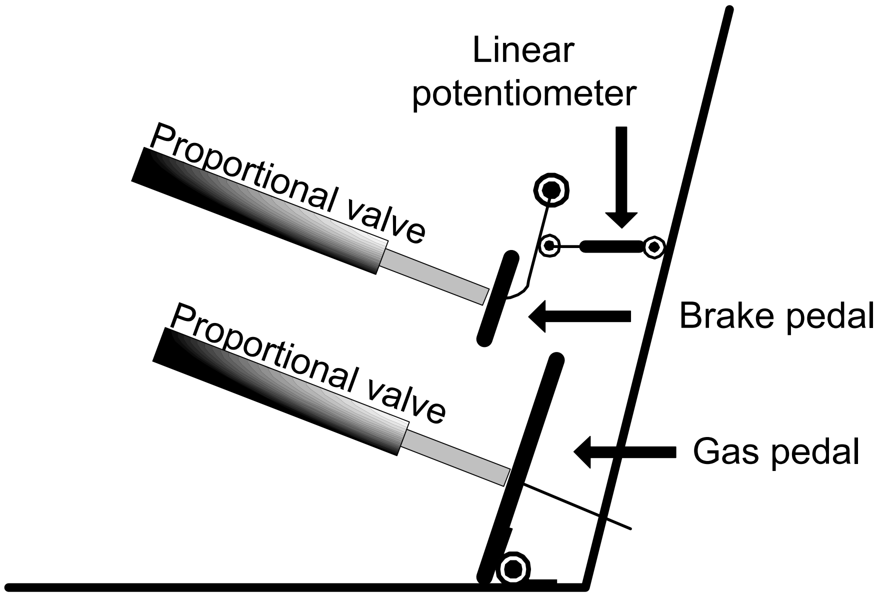
Move the axes to the starting position (handled by the “PC based’ application program)
Verify that no error conditions exist
Begin fluid output operation
Begin motion (handled by the “PC based’ application program)

A sequence of DCX macro’s control the dispensing of fluid.

```
MD400,AL@201,IE1,MJ401,NO,IE2,NO,MJ402,MJ403
                                        ;check for input sensors active (error
                                        ;condition)
MD401,OT”The fluid reservoir level is low /n”
MD402,OT”The dispense valve fluid is low /n”


MD403,CN4,WA.150,CN5                    ;begin fluid dispense

MD404,CF5,CF6,WA.150,CF4                ;terminate fluid dispense
```

PLC Control and DCX Analog I/O

Remote operation of a automobile provides a simple example analog I/O control. Two Proportional Pneumatic valves are used to control the velocity and the braking. One valve is positioned to depress the ‘gas pedal’ to control the vehicle speed. The other is positioned to depress the ‘brake pedal’. An analog tachometer is connected to the drive train to provide the vehicle speed feedback. A linear potentiometer is mounted to the back side of the brake pedal to provide the feedback of the position/force of the braking action.



Vehicle Speed Control
An analog output of 0.0V will cause the proportional valve that depresses the gas pedal to be fully retracted (no velocity). An analog output of +5.0V causes the valve to fully extend (pedal to the metal). The user defines a ‘look up table’ that is used to equate a DAC value to the speed of the vehicle. The user’s application program then writes the appropriate DAC (speed) value into DCX User Register #100 and sets the new speed command flag (register 102). For the purposes of this example, if the difference between the commanded speed and the actual speed of the vehicle is greater than 5%, the DAC output will be adjusted accordingly. The following DCX User Registers are used to store and manipulate data for this application:

User Register 100	;current speed command
User Register 101	;A/D conversion of the output of the tachometer
User Register 102	;new programmed ‘speed command’ flag
User Register 103	;difference between the speed command and the tachometer ;feedback (signed value)
User Register 104	;difference between the speed command and the tachometer ;feedback (unsigned value)
User Register 105	;+/- speed command tolerance (5%)

The following macro commands will output the user's 'speed command' and adjust the vehicle velocity:

```
MD10,AL@100,OA1,MJ11                      ;output the DAC value
MD11,WA.10,GA5,AR101,MJ12                 ;wait 100 msec's, load the output of
                                          ;the tachometer
MD12,AL@100,AS@101,AR103,AV3,AR104,MJ13 ;find the signed and unsigned value of
                                          ;the difference between the 'speed
                                          ;command' and the tachometer feedback
MD13,AL@100,AM.05,AR105,MJ14              ;calculate 5% of the 'speed command'
MD14,AL@104,IB@105,MJ10,NO,MJ15           ;is vehicle speed within 5% of
                                          ;commanded speed?
MD15,AL@103,IB0,MJ16,NO,IG0,MJ17          ;is vehicle velocity too fast or too
                                          ;slow?
MD16,AL@100,AS@104,AL@100,AL0,AR102,MJ10
                                          ;decrease 'speed command ' by 5%
MD17,AL@100,AA@104,AL@100,AL0,AR102,MJ10
                                          ;increase 'speed command ' by 5%

GT10,AR200                                ;execute the 'speed control' macro
                                          ;sequence as a background task, store
                                          ;Task ID# in user register 200
```

Braking Control

An analog output of 0.0V will cause the proportional valve that depresses the gas pedal to be fully retracted (no brake pedal pressure). An analog output of +5.0V causes the valve to fully extend (full brake pressure). The user defines a 'look up table' that is used to equate a DAC value to the brake pedal pressure. The user's application program then writes the appropriate DAC (brake) value into DCX User Register #111 and sets the braking flag (register 101). For the purposes of this example, if the difference between the commanded brake pressure and the brake pedal position feedback is greater than 5%, the DAC output will be adjusted accordingly. The following DCX User Registers are used to store and manipulate data for this application:

User Register 110	;"Braking' flag register, 1=braking
User Register 111	;current DAC braking command
User Register 113	;Brake pedal linear potentiometer feedback
User Register 114	;difference between the brake command and the linear potentiometer ;feedback (signed value)
User Register 115	;difference between the brake command and the linear potentiometer ;feedback (unsigned value)
User Register 116	;+/- brake command tolerance (20%, 10%, or 5%)

The following macro commands will output the user's 'brake command' and adjust the brake pedal position:

```
MD20,AL@110,IE1,MJ21,NO,WA.1,RP           ;braking flag set?
MD21,AL@111,OA2,MJ22                      ;output DAC 'brake command'
MD22,WA.1,GA6,AR113,MJ23                  ;wait 100 msec's, load the A/D linear
                                          ;potentiometer value
MD23,AL@111,AS113,AR114,AV3,AR115,MJ24    ;find the signed and unsigned value of
                                          ;the difference between the 'brake
                                          ;command' and the brake pedal
                                          ;potentiometer feedback
MD24,AL@111,AM.05,AR116,MJ25              ;calculate 5% of the 'speed command'
MD25,AL@115,IB@116,MJ32,NO,MJ26           ;is brake pedal within 5% of command
                                          ;position?
MD26,AL@114,IB0,MJ28,NO,IG0,MJ27          ;is brake pedal pressure too much or
                                          ;too little?
```

```
MD27,AL@111,AS@115,AL@110,MJ20          ;decrease 'brake pedal pressure' by 5%
MD28,AL@111,AA@1115,AL@110,MJ20         ;increase 'brake pedal pressure' by 5%

GT20,AR201                              ;execute the 'braking' macro sequence
                                        ;as a background task. Store the Task
                                        ;ID# in user register 201
```

MCCL Command Quick Reference Tables

Setup Commands

MCCL	Code	Description
AG	E2h	set Acceleration feed-forward Gain
AH	EAh	Auxiliary encoder define Home
BD	DOh	Backlash compensation Distance
DB	76h	set position DeadBand
DG	E3h	set Deceleration feed-forward Gain
DH	23h	Define Home
DI	24h	DIrection
DS	75h	Deceleration Set
DT	C5h	Delay at Target
ES		Encoder counts / Steps Scale
FC	40h	Full Current
FF	33h	amplifier Fault input ofF
FL	151h	IIR Filter Load coefficients
FN	32h	amplifier Fault input oN
FR	27h	set derivative sampling period
HC	41h	Half Current
HL	D3h	set motion High Limit
HS	E8h	set High Speed
IL	28h	set Integration Limit
IO	205h	Integral option
LA	330h	Load commutation phase A
LB	331h	load commutation phase B
LD	333h	Load commutation Divisor
LE	332h	Load commutation encoder prescale
LF	36h	motion Limits ofF
LL	D2h	set motion Low Limit
LM	34h	Limit Mode
LN	35h	motion Limits oN
LR	334h	Load commutation repeat count
LS	36h	set Low Speed
MS	E7h	set Medium Speed
MV	C4h	set Minimum Velocity
NF	150h	No IIR Filter
OB	C3h	set the Output deadBand
OM	D8h	set Output Mode
OO	CBh	set the Output Offset
PH	73h	set servo output PHase
SA	2Bh	Set Acceleration
SD	2Ch	Set Derivative gain
SE	19h	Stop on Error
SF	3Eh	Step Full
SG	2Dh	Set prop. Gain of motor
SH	3Fh	Step Half
SI	2Eh	Set Integral gain
SQ	74h	Set TorQue
SS		Set Slave ratio
SV	2Fh	Set Velocity
UA	9Ch	Use as default Axis
UK	D7h	set User output constant
UO	B3h	set User Offset
UP	9Dh	Use Physical axis
UR	B1h	set User Rate conversion
US	AFh	set User Scale
UT	B2h	set User Time conversion
UZ	B0h	set User Zero
VA	ADh	set Vector Acceleration

Setup Commands continued

MCCL	Code	Description
VD	AEh	set Vector Deceleration
VG	77h	set Velocity Gain
VO	E0h	set Velocity Override
VV	ACh	set Vector Velocity
YF	14Fh	Yes IIR Filter
ZF	153h	Zero IIR Filter coefficients

Motion Commands

MCCL	Code	Description
AB	Ah	ABort
AF	EBh	Auxiliary encoder arm/Find index
BC	144h	Begin position Compare
BF	CFh	Backlash compensation oFf
BN	CEh	Backlash compensation oN
CA	B4h	arc Center Absolute
CB	148h	position Capture Begin
CD	C1h	Contour Distance
CP	C0h	Contour Path
CR	B5h	arc Center Relative
EA		arc Ending Angle absolute
EL	E4h	home Edge Latch
ER		arc Ending angle Relative
FE	Bh	Find Edge
FI	Ch	Find Index
GH	Dh	Go Home
GO	Eh	GO
HO	Fh	HOme
IA	5Dh	Index Arm
LC	142h	Load Compare positions
LP	70h	Learn Position
LT	71h	Learn Target
MA	10h	Move Absolute
MF	11h	Motor ofF
MN	13h	Motor oN
MP	14h	Move to Point
MR	15h	Move to Point
NC	143h	Next Compare
NS	ABh	No Synchronization
OC	140h	Output mode for Compare
OP	141h	Output Period
PP	EDh	Profile Parabolic
PR		Record motion data
PS	EEh	Profile S-curve
PT	EFh	Profile Trapezoidal
RC	115h	Restore Configuration
RR		aRc Radius
SC	114h	Save Configuration
SN	AAh	Synchronization oN
ST	16h	STop

Mode Commands

MCCL	Code	Description
CM	1Bh	enable Contour Mode (arcs and lines)
GM	8h	enable Gain Mode (no velocity profile)
IM	72h	Input Mode (closed loop stepper)
PM	17h	enable Position Mode
SM		enable Master/Slave mode
QM	9H	enable TorQue mode
VM	18h	enable Velocity Mode

Reporting Commands

MCCL	Code	Description
AT	E9h	Auxiliary encoder Tell position
AZ	ECh	Auxiliary encoder tell index
CG	149h	Capture Get count
DO		Display recorded optimal position
DQ		Display recorded DAC output
DR		Display recorded actual position
GC	145h	Get the position compare count
TA	49h	Tell Analog to digital converter
TB	5Bh	Tell Breakpoint position
TC	4Ah	Tell Channel
TD	4Bh	Tell Derivative gain
TE		Tell command interface Error
TF	4Dh	Tell Following error
TG	4Eh	Tell proportional Gain
TI	4Fh	Tell Integral gain
TK	5Ch	Tell velocity gain
TL	50h	Tell integration Limit
TM	51h	Tell stored Macros
TO	59h	Tell Optimal
TP	52h	Tell Position
TQ	D1h	Tell torQue (aSQn)
TR	57h	Tell Register n
TS	53h	Tell Status
TT	54h	Tell Target
TV	55h	Tell Velocity
TX	58h	Tell contouring count
TZ	5Ah	Tell index position
VE	56h	tell VErsion

Macro Commands

MCCL	Code	Description
BK	79h	BreaK
ET	FBh	Escape Task
GT	FAh	Generate Task
MC	2h	Macro Call
MD	3h	Macro Definition
MJ	5h	Macro Jump
RM	4h	Reset Macros
TM	51h	Tell Macros

Register Commands

MCCL	Code	Description
AA	85h	Accumulator Add
AC	8Ch	Accumulator Complement
AD	88h	Accumulator Divide
AE	8Fh	Accumulator logical Exclusive or
AL	82h	Accumulator Load
AM	87h	Accumulator Multiply
AN	8Dh	Accumulator logical aNd with n,
AO	83h	Accumulator logical Or with n
AR	84h	copy Accumulator to Register n
AS	86h	Accumulator Subtract
AV	8Bh	Accumulator eValuate
AX	E1h	get Aux. indeX position
GA	F8h	Get Analog value
GD		Get module ID
GF	152h	Get IIR Filter coefficients
GU	89h	Get the default axis
GX	F7h	Get auXiliary encoder position
LU	81h	Look Up motor table variable
OA	F9h	Output Analog value
RA	83h	copy Register to Accumulator
RB	96h	Read Byte into accumulator
RD	93h	Read Double into accumulator
RL	98h	Read Long into accumulator
RV	92h	Read float into accumulator
RW	97h	Read Word into accumulator
SL	90h	Shift Left accumulator n bits
SR	91h	Shift Right accumulator n bits
TR	57h	Tell contents of Register n
WB	99h	Write accumulator Byte to n
WD	95h	Write accumulator double to n
WL	9Bh	Write accumulator Long to n
WV	94h	Write accumulator float to n
WW	9Ah	Write accumulator Word to n

I/O Commands

MCCL	Code	Description
CF	1Fh	Channel ofF
CH	42h	Channel High true logic
CI	20h	Channel In
CL	43h	Channel Low true logic
CN	21h	Channel oN
CT	22h	Channel ouT
GA		Get Analog
OA		Output Analog
TA	49h	Tell the value of Analog input
TC	4Ah	Tell state of digital Channel
WF	67	Wait for channel ofF
WN	68	Wait for channel oN

Sequence Commands

MCCL	Code	Description
DF	6B	Do if channel ofF
DN	6A	Do if channel oN
IB	A5	If Below do next command
IC	A1	If Clear, do next command
IE	A2	If Equals do next command
IF	6D	If channel ofF do next command
IG	A4	If accumulator is Greater do next
IN	6C	If channel oN do next command
IP	60	Interrupt on absolute Position
IR	61	Interrupt on Relative position
IS	A0	If bit Set do next command
IU	A3	If Unequal do next command
JP	6	JumP to command absolute
JR	7	Jump to command Relative
RP	64	RePeat
WA	65	WAit (time)
WE	66	Wait for Edge
WF	67	Wait for channel ofF
WI	5E	Wait for Index
WN	68	Wait for channel oN
WP	62	Wait for absolute Position
WR	63	Wait for Relative position
WS	63	Wait for Stop
WT	C6	Wait for Target

Miscellaneous Commands

MCCL	Code	Description
DM	3Ch	Decimal Mode
DW	FDh	Disable Watchdog
EF	25h	Echo ofF
EN	26h	Echo oN
FD		Format text with Doubles
FT		Format Text with Integers
HM	3Dh	Hexadecimal Mode
NO	78h	No Operation
OD		Output text with Doubles
OT		Output Text with integers
PC	80h	set Prompt Character
RT	2Ah	ReseT system

Reading Data from DCX Memory

A group of read commands are available for accessing the internal Motor Tables of the DCX. These commands provide an easy method of moving motor data in and out of the Accumulator (user register 0).

The **L**ook **U**p (**LU*s***) command is used to load the internal address of a motor table entry into the accumulator. String parameter *s* defines the variable name of the target motor table entry. A Read command (RB, RW, RL, RV, or RD) is then used to load the accumulator with the data from the target motor table entry. The Read command must include the axis specifier ‘a’. The type of command to use (byte, double, long, float or word), is determined by the type of data to be accessed and is listed below.

a**RB*****n*** Read Byte (8 bit) at memory location n into accumulator (ACC = (n))
a**RD*****n*** Read Double at memory location n into accumulator (ACC = (n))
a**RL*****n*** Read Long (32 bit) at memory location n into accumulator (ACC = (n))
a**RV*****n*** Read float at memory location n into accumulator (ACC = (n))
a**RW*****n*** Read Word (16 bit) at memory location n into accumulator (ACC = (n))

Examples of using the read commands to access the motor tables are shown below.

To load the 32 bit status of axis 2 into the accumulator, issue the following command sequence:

```
LU”STATUS”,2RL@0                    ;load the motor table address for axis
                                    ;status into the accumulator. Load the
                                    ;32 bit status word of axis #2 into the
                                    ;accumulator
```

To load the 64 bit current position of axis 3 into the accumulator, issue the following command:

```
LU”POSITION”,3RD@0                  ;load the motor table address for current
                                    ;position into the accumulator. Load the
                                    ;accumulator with the 64 bit current
                                    ;position of  axis #3
```

Motor Table Variables – 32 bit integer (long)

Motor Table Variable Description	***Variable Name***
Module Base Address	MODADDR
Motor Status – primary	STATUS
Motor Status - auxiliary	AUXSTAT
Position - Adjustment (index + Offset)	CNTADJ
Position – Current (raw count)	POSCOUNT
Position - Index Count	IDXCOUNT
Position - Optimal (raw count)	CMDCOUNT
Position – Target (raw count)	TGTCOUNT

Motor Table Variables – 64 bit floating point (double)

Motor Table Variable Description	***Variable Name***
Backlash count	BACKLASH
Dead band	DBAND
Encoder counts / Steps scaling	ENCSCALE
Following Error Setting - maximum	MAXERROR
Position - Current	POSITION
Position – Target	TARGET
Position - Optimal	OPTIMAL
Position - Breakpoint	BRKPOS
Programmed Acceleration	PGMACC
Programmed Deceleration	PGMDEC
Programmed Minimum Velocity	MINVEL
Programmed Velocity	PGMVEL
Scaling - User	SCALE
Scaling - User Offset	OFFSET
Scaling - User Output Constant	OUTCONST
Scaling - User Rate Conversion	RATE
Scaling - User Zero	ZERO
Soft Motion Limit Setting (low)	LOLIM
Soft Motion Limit Setting (high)	HILIM
Velocity - Current	CURVEL

Motor Table Variables – 32 bit floating point (float)

Motor Table Variable Description	***Variable Name***
A/D channel #1	ADCIN1
A/D channel #2	ADCIN1
Feed Forward - Velocity Gain	KVEL
Feed Forward - Acceleration Gain	KACC
Feed Forward - Deceleration Gain	KDEC
PID - Proportional Gain setting	KPOS
PID - Derivative Gain setting	KDER
PID - Integral Gain	KINT
PID - Integration Limit	ILIM
Torque Limit – max. output	MAXTRQ
Velocity Override	VELOVRD

Motor Table Variables – 16 bit integer (word)

Motor Table Variable Description	***Variable Name***
Axis Number	AXISNUM
IIR filter coefficients, maximum	COEFMAX
IIR filter coefficients, qty. defined	COEFCNT
Master Axis	MASTER
Input Mode	INPUTMODE
Integral Term Options	INTEGRALOPT
Module - Axis	MODAXIS
Module – Base address	MODADDR
Module - Location	MODULE
Module Status	MODST
Module - Type	MODTYPE
Derivative Term Sampling Frequency	SFREQ
Timer - Wait Stop	WAITSTOP
Timer - Wait Target	WAITTARGET

Single Stepping MCCL Programs

While the DCX is executing any Motion Control Command Language (MCCL) macro program via WinControl 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
```

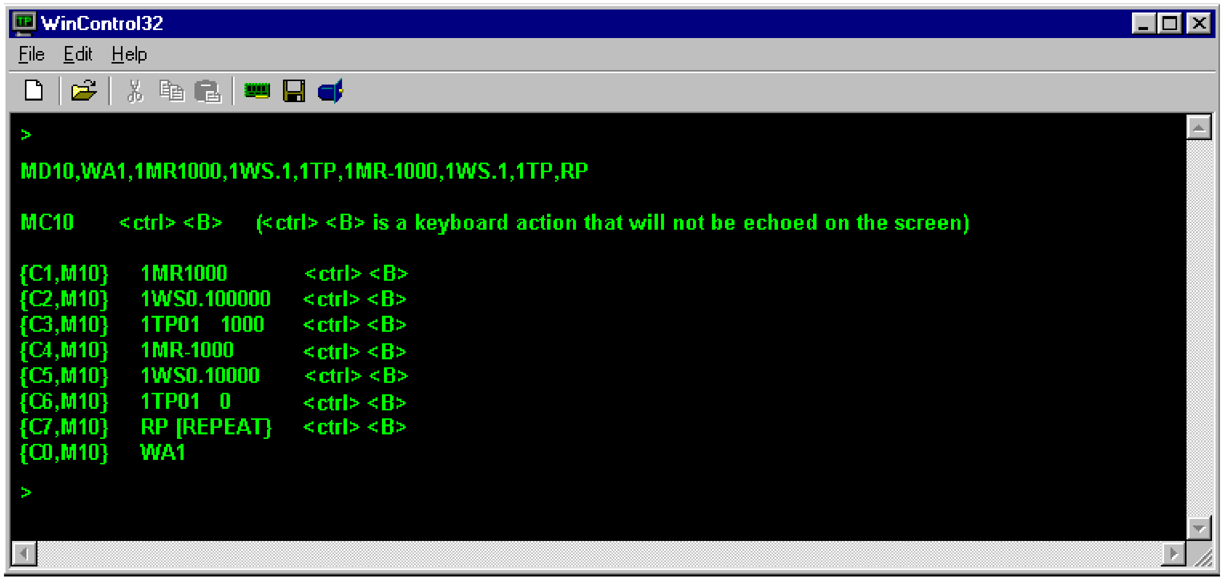
This sample program will: wait for 1 second, move 1000 encoder counts, report the position 100 msec's after the calculated trajectory is complete, move -1000 encoder counts, report the position 100 msec's after the calculated trajectory is complete, repeat the command sequence.

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



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

MC10      <ctrl> <B>    (<ctrl> <B> is a keyboard action that will not be echoed on the screen)

{C1,M10}    1MR1000         <ctrl> <B>
{C2,M10}    1WS0.100000     <ctrl> <B>
{C3,M10}    1TP01   1000    <ctrl> <B>
{C4,M10}    1MR-1000        <ctrl> <B>
{C5,M10}    1WS0.10000      <ctrl> <B>
{C6,M10}    1TP01   0       <ctrl> <B>
{C7,M10}    RP [REPEAT]     <ctrl> <B>
{C0,M10}    WA1

>
```

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

DCX User Registers

The DCX contains 256 general purpose global registers that can be used for; storing command parameters, performing math computations and controlling command execution. The registers are numbered 0 through 255, with register 0 being the 'accumulator'. The accumulator (register 0) is used by all commands that manipulate register data.

Each register can hold a 32 bit integer, a 32 bit single precision floating point number, or a 64 bit double precision floating point number. A register will be loaded with the double precision floating point number if the **A**ccumulator **L**oad (**AL*n***) command is issued with a parameter containing a decimal point. Otherwise, the register will be loaded with a 32 bit integer. When executing commands that perform math operations on the accumulator (AA, AD, AM, ...), the result will have the same precision as the command parameter or the accumulator (prior to the command), whichever is more precise. Since the 32 bit integer is considered to be the least precise, multiplying an integer by a floating point number will always result in a floating point number. If a floating point indirect parameter is used for a command that does not support floating point parameters (eg. CN, LM, PC,...), the register contents will be rounded to the nearest integer prior to use.

Typically the user issues commands with 'immediate' parameters (ie: the parameter ‘n’ is a constant). The user can also issue commands, specifying that the parameter is the contents of a register. This is done by replacing the command parameter with the register number preceded with an '@' sign. For example, the command "1MR@10" will cause the DCX to move axis 1 by the number stored in register 10. The use of a register specifier can be used in any command as the parameter. The DCX ***does not*** support the use of the '@' sign in front of an axis number. The following commands are available for working with the registers:

MCCL Command	Description
AAn	Accumulator Add (ACC = ACC + n)
ACn	Accumulator Complement, bit wise (ACC = !ACC)
ADn	Accumulator Divide (ACC = ACC/n)
AEn	Accumulator logical Exclusive or with n, bit wise (ACC = ACC eor n)
ALn	Accumulator Load with constant n (ACC = n)
AMn	Accumulator Multiply(ACC = ACC x n)
ANn	Accumulator logical aNd with n, bit wise (ACC = ACC and n)
AOn	Accumulator logical Or with n, bit wise (ACC = ACC or n)
ARn	copy Accumulator to Register n (REGn = ACC)
ASn	Accumulator Subtract (ACC = ACC - n)
GAx	Get Analog value (ACC = channel x)
aGX	Get auXiliary encoder position (ACC = axis a auxiliary encoder)
IBn	If accumulator is Below (>) n, do next command, else skip 2 commands
ICn	If bit n of accumulator is Clear, do next command, else skip 2 commands
IEn	If accumulator Equals constant n, do next command, else skip 2 commands
IGn	If accumulator is Greater than ‘n’, do next command, else skip 2 commands
OAx	Output Analog value (channel x = ACC)
ISn	If bit n of accumulator is Set, do next command, else skip 2 commands
IUn	If accumulator is Unequal to ‘n’, do next command, else skip 2 commands
RAn	copy Register n to Accumulator (ACC = REGn)
SLn	Shift Left accumulator n bits (ACC = ACC << n)
SRn	Shift Right accumulator n bits (ACC = ACC >> n)
TRn.p	Tell contents of Register n
TR.p	Tell contents of accumulator (register 0)

DCX Scratch Pad Memory

Not supported

Over and above what is available by using the User Registers, the DCX also provides an 8KB space allocated for user scratch pad memory. The allocate **ME**mory (**ME*n***) command is used to format the memory space for user operations. The parameter n defines the number of bytes to be allocated for use.

Upon executing the memory allocate command, the accumulator will be loaded with the address of the first byte of allocated memory. The following commands are used to write data from the accumulator into the allocated memory locations:

WBn	Write accumulator low Byte (8 bit) to memory location n ((n) = ACC)
WDn	Write accumulator Double to absolute memory location n ((n) = ACC)
WLn	Write accumulator Long (32 bit) to memory location n ((n) = ACC)
WVn	Write accumulator float to absolute memory location n ((n) = ACC)
WWn	Write accumulator low Word (16 bit) to memory location n ((n) = ACC)

The following commands are used to read data from the allocated memory into the accumulator:

RBn	Read Byte (8 bit) at memory location n into accumulator (ACC = (n))
RDn	Read Double at memory location n into accumulator (ACC = (n))
RLn	Read Long (32 bit) at memory location n into accumulator (ACC = (n))
RVn	Read float at memory location n into accumulator (ACC = (n))
RWn	Read Word (16 bit) at memory location n into accumulator (ACC = (n))

The **F**ree **M**emory (**FM*n***) command returns previously allocated memory and returns it to the ‘heap’ from which it was allocated. The parameter n of this command must be the same as the value that was loaded into the accumulator upon issuing the memory allocated (ME) command.

Chapter Contents

Chapter 7

Setup Commands

AG Acceleration Gain

MCCL command: *a*AG*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320
see also: DG, VG

This command sets the acceleration feed-forward gain for a servo. The product of this gain and the motor's calculated acceleration will be summed into the controller's DAC output. The acceleration gain is only applied while a motor is accelerating, when it decelerates the deceleration gain term is used (see DG command).

comment: Acceleration and deceleration feed-forwards are not calculated when a motor is in contour mode.

```
      1AG10.0                                   ;Sets Acceleration gain for axis 1 to
                                                ;10.0
```

AH Auxiliary encoder define Home

MCCL command: *a*AH*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC320, MC360
see also: DH

This command causes axis *a* auxiliary encoder position to be set to *n*. This encoder input is available on both the MC300 and MC360 modules, and is used for loop closure when a MC360 is controlling a closed loop stepper. The auxiliary encoder of a MC300 is used for position verification only, it cannot be used for dual loop positioning. For defining the home position of the primary encoder, see the Define Home command.

BD Backlash compensation Distance

MCCL command: *a*BD*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: BF, BN

Use this command to set the distance required to nullify the effects of mechanical backlash in the system. The command parameter should be equal to half the amount the motor must move to take up backlash when it changes direction. The units for this command parameter are encoder counts, or the units established by the User Scale command for the axis.

Once the backlash compensation distance is set, issuing the Backlash compensation oN command will cause the controller to add or subtract the 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. See the description on backlash compensation in the **Application Solutions** chapter of this manual.

DB set position DeadBand

MCCL command: *a*DB*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320
see also: DT, WT

This command sets the position dead band that is used by the controller to determine when a servo axis is 'At Target'. In order for the At Target flag in the motor status to be set, a servo must remain within the specified dead band of the current target position for a period of time specified with the Delay at Target (aDTn) command.

DG Deceleration Gain

MCCL command: *a*DG*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320
see also: AG, VG

This command sets the deceleration feed-forward gain for a servo. The product of this gain and the motor's calculated deceleration will be summed into the controller's DAC output. The deceleration gain is only applied while a motor is decelerating. When it accelerates the acceleration gain term is used (see AG command).
comment: Acceleration and deceleration feed-forwards are not calculated when a motor is in contour mode.

```
example:  1DG10.0                         ;Sets Deceleration gain for axis 1 to
                                          ;10.0
```

DH **D**efine **H**ome

MCCL command: *a*DH*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: FI, IA, WI

Defines the current position of a motor to be n. From then on, all positions reported for that motor will be relative to that point.

DI **DI**rection

MCCL command: *a*DI*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: GO, VM

Sets the move direction of a motor when in velocity mode. A parameter value of 0 results in motion in the positive direction, a value of 1 causes motion in the negative direction.

DS **D**eceleration **S**et

MCCL command: *a*DS*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: SA, SV

Defines the deceleration rate for an axis. The default units for the command parameter are encoder counts (or steps) per second per second.

DT **D**elay at **T**arget

MCCL command: *a*DT*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320
see also: DB, WT

This command sets the time period during which a servo must remain within the position dead band of the target for the 'At Target' flag in the motor status to be set.

ES **E**ncoder **S**cale

MCCL command: *a*ES*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC360
see also: IM

Defines the steps per rotation / encoder counts per rotation for a closed loop stepper axis. This parameter must be set before commanding closed loop stepper motion. The default value is 1.

FC **F**ull **C**urrent

MCCL command : *a*FC *a* = Axis number
compatibility: MC360, MC362
see also: HC

Causes Full/Half Current output signal of a stepper module to go low.

FF amplifier **F**ault o**F**f

MCCL command: *a*FF *a* = Axis number
compatibility: MC300, MC320, MC360
see also: FN

Disables the Amplifier Fault input of a servo control module or the Driver Fault input of a stepper control module. See description of amplifier Fault input oN command (FN), for further details.

FL **IIR Filter Load coefficients**

MCCL command : *a*FL*n* *a* = Axis number n = real
compatibility: MC300, MC320
see also: GF, NF, YF, ZF

Downloads the user defined IIR filter coefficients, which can be used to implement a notch or low pass filter. See the description of **User Defined Filters (Notch and Low Pass)** in the **Application Solutions chapter**.

FN amplifier **F**ault o**N**

MCCL command : aFN *a* = Axis number
compatibility: MC300, MC320, MC360
see also: FF

Enables the Amplifier Fault input of a servo control module or the Driver Fault input of a stepper control module. If the input goes active after this command is executed, the motor will be turned off and the amplifier fault tripped flag in servo status will be set. The tripped flag will remain set until the motor is turned back on with the MN command.

FR set the derivative sampling period

MCCL command: *a*FR*n* *a* = Axis number *n* = integer >= 0
compatibility: MC300, MC302, MC320
see also: SD, SG

Helps tune servo loop to the inertial characteristics of system. High inertial loads normally require a longer period and low inertial loads a shorter period. The default value is 0 (0.00025 seconds). For a value of n, the derivative sampling period will be (n +1) * (sample period). See the High Speed command for a discussion of the sample period on servos. See **Tuning the Servo** section in the **Motion Control** chapter.

HC **H**alf **C**urrent

MCCL command: *a*HC *a* = Axis number
compatibility: MC360, MC362
see also: FC

Causes Full/Half Current output signal of a stepper module to go high.

HL High motion soft Limit

MCCL command: *a*HL*n* *a* = Axis number *n* = integer or real
compatibility: MC300, MC302, MC320, MC360, MC362
see also: LF, LL, LM, LN

This command sets the high limit for motion. After this command is issued, and the motion limit is enabled with the Limit oN (aLNn) command, the command parameter is used as a 'soft' limit for all motion of the axis. If the desired or true position of the axis is greater than this limit, and the axis is being commanded to move in the positive direction, the Soft Motion Limit High and the Motor Error flags in the motor status will be set. The axis will also be turned off, stopped abruptly, or stopped smoothly, depending upon the mode set by the Limit Mode command. Please refer to the **Motion Limits** description in the **Motion Control** chapter.

comment: When the axis is in contouring mode, this limit will be tripped anytime the desired or true position is greater than the limit regardless of commanded direction. Thus, to move an axis out of the limit region, it must be placed in a non-contour mode. When one or more axes are moving in contour mode, and one of the axes experiences a limit trip, all axes associated with the motion will be turned off or stopped.

HS High Speed

MCCL command: *a*HS *a* = Axis number
compatibility: MC300, MC302, MC320, MC360, MC362
see also: LS, MS

This command has a different effect depending on whether it is issued to a servo or stepper motor axis. For a servo axis, it sets the servo feedback loop to 8 KHz update rate. For a stepper motor axis, it sets the maximum pulse rate to 5.0 Million Pulses/Sec.

IL Integration Limit

MCCL command: *a*IL*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320
see also: SI, SG

Limits level of power that integral gain can use to reduce the position error. The default units for the command parameter are (encoder counts) * (sample interval). See the description of **Tuning the Servo** section in the **Motion Control** chapter.

IO Integral Option

MCCL command : *a*SI*n* *a* = Axis number *n* = integer 0, 1, or 2
compatibility: MC300, MC302, MC320, MC360
see also: SI, WS

The integral term accumulates the position error for servos (and closed loop stepper) and generates an output signal to reduce the position error to zero. The integral gain determines the magnitude of this term. The default value is zero. The **I**ntegral **O**ption command allows the user to define how the integral term of the PID filter functions while a servo is moving.

n =	***Integral term function***
0	**Integral term always on (default)**
1	Freezes accumulation of integration term during movement. Integration will continued once the calculated trajectory (trajectory complete, status bit 3 = 1) has been completed.
2	Zero integration term when motion begins. When the calculated trajectory (trajectory complete, status bit 3 = 1) has completed, enable the integration term

See the description of **Turning off Integral gain during a move** in the **Application Solutions** chapter.

LA Load commutation phase A

MCCL command: *a*LA*n* *a* = Axis number *n* = integer > -360 < 360
compatibility: MC320
see also: LB,LD,LE,LR

Defines the phase shift for phase A DAC command output. For additional information on brushless motor commutation please refer to application note **AN1004 – Brushless AC Motor Commutation** available on PMC's **MotionCD**.

LB Load commutation phase B

MCCL command: *a*LB*n* *a* = Axis number *n* = integer > -360 < 360
compatibility: MC320
see also: LA,LD,LE,LR

Defines the phase shift for phase B DAC command output. For additional information on brushless motor commutation please refer to application note **AN1004 – Brushless AC Motor Commutation** available on PMC's **MotionCD**.

LD Load commutation encoder Divisor constant

MCCL command: *a*LD*n* *a* = Axis number *n* = integer > 0
compatibility: MC320
see also: LA, LB, LE, LR

Defines the encoder divisor to be used for brushless motor commutation. For additional information on brushless motor commutation please refer to application note **AN1004 – Brushless AC Motor Commutation** available on PMC's **MotionCD**.

LE Load commutation Encoder prescale constant

MCCL command: *a*LE*n* *a* = Axis number *n* = 1, 2, 4, 8, 16, 32, 64, 128, 256
compatibility: MC320
see also: LD,LR,LA,LB

Defines prescaling to be used for brushless motor commutation. For additional information on brushless motor commutation please refer to application note **AN1004 – Brushless AC Motor Commutation** available on PMC's **MotionCD**.

LF motion Limits oFf

MCCL command: *a*LF*n* *a* = Axis number *n* = 0 – 15
compatibility: MC300, MC302, MC320, MC360, MC362
see also: LN, LM

Disables one or more 'hard' limit switch inputs or 'soft' position limits for an axis. The parameter to this command determines which limits will be disabled. The coding of the parameter is the same as for the motion Limits oN command (LN). See the description on **Motion Limits** in the **Motion Control** chapter.

LL Low motion soft Limit

MCCL command: *a*LL*n* *a* = Axis number *n* = integer or real
compatibility: MC300, MC302, MC320, MC360, MC362
see also: LF, LH, LM, LN

This command sets the low limit for motion. After this command is issued, and the motion limit is enabled with the Limit oN (aLNn) command, the command parameter is used as a 'soft' limit for all motion of the axis. If the desired or true position of the axis is less than this limit, and the axis is being commanded to move in the negative direction, the Soft Motion Limit Low and the Motor Error flags in the motor status will be set. The axis will also be turned off, stopped abruptly, or stopped smoothly, depending upon the mode set by the Limit Mode command. See the description of **Motion Limits** in the **Motion Control** chapter.

comment: When the axis is in contouring mode, this limit will be tripped anytime the desired or true position is less than the limit regardless of commanded direction. Thus, to move an axis out of the limit region, it must be placed in a non-contour mode. When one or more axes are moving in contour mode, and one of the axes experiences a limit trip, all axes associated with the motion will be turned off or stopped.

LM Limit Mode

MCCL command: *a*LM*n* *a* = Axis number *n* = integer 0 - 15
compatibility: MC300, MC302, MC320, MC360, MC362
see also: LF, LN

This command is used to select how the DCX will react when a 'hard' limit switch or a 'soft' position limit is tripped by an axis. The command parameter should be formed by adding a value of 1, 2, or 3 for the hard limit switch mode, to a value of 4, 8, or 12 for the soft position limit mode. In all cases the Motor Error and one of Limit Tripped flags in the status word will be set when a limit event occurs. This will prevent the DCX from moving the motor until a Motor oN command is issued. See the description of **Motion Limits** in the **Motion Control** chapter.

Limit Mode (aLMn) command

Desired action	***Parameter n =***
Turn motor off (disable PID) when **hard limit sensor 'goes' active** or soft motion limit is exceeded	0,0 *
Stop the motor abruptly (under PID control) when **hard limit sensor 'goes' active** or soft motion limit is exceeded	1,4 *
Decelerate and stop the motor (under PID control) when **hard limit sensor 'goes' active** or the soft motion limit is exceeded. Use the current deceleration	2,8 *

setting.	
Invert the active level of the hard limit input. Typically used for normally closed hard limit sensors	128 **

* Values in red are for defining the Limit Mode for hard limits. Values in black are for defining the mode for soft motion limits. When using both hard and soft limits, parameter n should equal hard limit parameter n + soft limit parameter n.
** For inverted active level hard limits parameter n = 128 plus desired mode

```
1LM130                          ;Axis #1 Limit mode = decelerate & stop (n=2)
                                ; + invert active level(n=128)
```

LN Limits oN

MCCL command: *a*LN*n*   *a* = Axis number   *n* = 0 - 15
compatibility: MC300, MC302, MC320, MC360, MC362
see also: LF, LM

This command is used to enable the 'hard' limit switch inputs and/or the 'soft' position limits of an axis. If a limit switch input goes active after it has been enabled by this command, and the motor has been commanded to move in the direction of that switch, the Motor Error and one of the Hard Limit Tripped Flags will be set in the motor status. At the same time the motor will be turned off or stopped (depending on the value of parameter *n* of the **L**imit **M**ode command). If a soft motion limit is enabled, and the respective axis is commanded to move beyond the motion limits set by the High motion Limit and the Low motion Limit commands, the Motor Error and one of the Soft Limit Tripped Flags will be set. At the same time the motor will be turned off or stopped (depending on the value of parameter *n* of the **L**imit **M**ode command). The flags will remain set until the motor is turned back on with the MN command. Once the motor is turned back on, it can be moved out of the limit region with any of the standard motion commands. The parameter to this command determines which of the hard and soft limits will be enabled. See the description of **Motion Limits** in the **Motion Control** chapter.

The **LN** command enables hard coded limit error checking.

Parameter n	***Enable Hard & Soft Limits - Limit oN parameter n description***
0*	Enable both hard limits (+/-) and soft limits (high & low)
1**	Enable hard limit + error checking
2**	Enable hard limit – error checking
3**	Enable hard limit + and hard limit – error checking
4**	Enable high soft limit error checking
8**	Enable low soft limit error checking
12**	Enable high & low soft limit error checking

* If parameter n = 0 both hard and soft limit error checking will be enabled.
** Values in red are for enabling limit error checking for hard limits. if both hard and soft limits are to be used the parameter ***n*** should equal hard limit parameter ***n*** + soft limit parameter ***n***.

```
1LN0                         ;Axis #1 - enable hard and soft limits

2LN7                         ;Axis #2 - enable both hard limits (n=3) and
                             ;high soft motion limit (n=4)
```

LR Load commutation encoder Repeat count

MCCL command: *a*LR*n* *a* = Axis number *n* = integer > 0
compatibility: MC320
see also: LA, LB, LD, LE

Defines the number of encoder counts in a commutation cycle. For additional information on brushless motor commutation please refer to application note **AN1004 – Brushless AC Motor Commutation** available on PMC's **MotionCD**.

LS Low Speed

MCCL command: *a*LS *a* = Axis number
compatibility: MC300, MC302, MC320, MC360, MC362
see also: HS, MS

This command has a different effect depending on whether it is issued to a servo or stepper motor axis. For a servo axis, it sets the feedback loop to 2 KHz update rate. For a stepper motor axis, it sets the maximum pulse rate to 78 Thousand Pulses/Sec.

MS Medium Speed

MCCL command: *a*MS *a* = Axis number
compatibility: MC300, MC302, MC320, MC360, MC362
see also: HS, LS

This command has a different effect depending on whether it is issued to a servo or stepper motor axis. For a servo axis, it sets the feedback loop to 4KHz update rate. For a stepper motor axis, it sets the maximum pulse rate to 625 Thousand Pulses/Sec.

MV set Minimum Velocity

MCCL command: *a*MV*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC360, MC362
see also: SV

Sets the minimum velocity for a given stepper motor axis. The purpose of this command is to set an initial and final velocity for motion of stepper motors. Below this velocity a full stepping motor is 'cogging' between steps. The default units for the command parameter are steps per second. This command will have no effect on servos.

NF No IIR Filter

MCCL command : *a*NF *a* = Axis number
compatibility: MC300, MC320
see also: GF, FL, YF, ZF

Disables the IIR filter, which can be used to implement a notch or low pass filter. See the description of **User Defined Filters (Notch and Low Pass)** in the **Application Solutions chapter**.

OB Output dead Band

MCCL command: *a*OB*n* *a* = Axis number *n* = integer or real > 0, <=10
compatibility: MC300, MC302, MC320
see also: OO

This command can be used to simulate a 'frictionless servo system'. Parameter ***n*** must be a positive real value between 0 and 10. Parameter ***n*** modifies the commanded analog output to a servo. This value will be added to or subtracted from the analog output. If the calculated command output voltage is positive, the voltage is increased by the output deadband voltage (n). If the calculated output voltage is negative, the command voltage is decreased by the deadband amount (n) . And if the calculated output is zero, the output will be 0 volts.

OM Output Mode

MCCL command: *a*OM*n* *a* = Axis number *n* = integer 0, 1, 2,
compatibility: MC300, MC360, MC362
see also:

This command is used to set a servo or stepper module's output mode. The available modes are listed in the following tables.

n	*MC300 Output Mode*
0	Bipolar Analog output, -10V to +10V
1	Unipolar Analog output, 0V to +10V, direction J3 pin 7

n	*MC360 / MC362 Output Mode*
0	Pulse and Direction outputs (default)
1	CW and CCW Pulse Outputs

OO Output Offset

MCCL command: *a*OO*n* *a* = Axis number *n* = integer or real >= -10, <= +10
compatibility: MC300, MC302, MC320
see also: OD

This command is used to provide software programmability of the zero point of a servo output. Similar to adjusting an offset potentiometer, the parameter *n* will redefine the 'no commanded motion' output level.

PH set the servo output PHasing

MCCL command: *a*PH*n* *a* = Axis number *n* = 0 or 1
compatibility: MC300, MC302, MC320, MC360
see also: OM

This command is used to set a servo or closed loop stepper module's output phasing. The phase of the output will determine whether the module drives the axis in a direction that reduces position error, or increases it. The module defaults to standard phasing, which is the same as issuing this command with a parameter of 0. The module output can be set to reverse phase by issuing this command with a parameter of 1. This command has no effect on the command output polarity when operating in Torque Mode (aQM).

SA Set Acceleration

MCCL command: *a*SA*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: DS, SV

Set the maximum acceleration rate for a given axis. The default units for the command parameter are encoder counts (or steps) per second per second.

SD Set Derivative gain

MCCL command: *a*SD*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320
see also: FR, IL, SI, SG

This command is used to set the derivative gain of a servo's feedback loop. Increasing the derivative gain has the effect of dampening oscillations. See the description of **Tuning the Servo** in the **Motion Control** chapter.

SE Stop on following Error

MCCL command : *a*SE*n* *a* = Axis number *n* = integer or real <= 0
compatibility: MC300, MC302, MC320
see also:

Used to set the maximum following or position error for a servo. Once this command is issued and the motor is on, if the servo position error exceeds the specified value the motor error flag in servo status will be set, and the servo will be turned off. The error flag will remain set until the motor is turned back on with the MN command. Issuing this command with parameter *n* = 0 will disable errors on excessive following errors.

SF Step Full

MCCL command: *a*SF *a* = Axis number
compatibility: MC360, MC362
see also: SH

Causes Full/Half Step output signal of a stepper module to go low. This command is typically used to disable ‘micro stepping’ of a stepper driver.

SG Set Proportional gain

MCCL command : *a*SG*n* *a* = Axis number *n* = integer or real >= 0.000153, <=10
compatibility: MC300, MC302, MC320
see also: IL, SI, SD

This command is used to set the proportional gain of a servo's feedback loop. Increasing the proportional gain has the effect of stiffening the force holding a servo in position. The parameter to this command has default units of volts per encoder count. This command should not be used for open loop stepper axes. See the description of **Tuning the Servo** in the **Motion Control** chapter.

SH Step Half

MCCL command: *a*SH *a* = Axis number
compatibility: MC360, MC362
see also: SF

Causes Full/Half Step output signal of stepper module to go high. This command is typically used to enable ‘micro stepping’ of a stepper driver.

SI Set the Integral gain

MCCL command : *a*SI*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320
see also: SI, SG

The integral term accumulates the position error for servos and generates an output signal to reduce the position error to zero. The integral gain determines the magnitude of this term. The default value is zero. Note that Integration Limit (IL) command must be set to a nonzero value before integral gain will have any effect. See the description of **Tuning the Servo** in the **Motion Control** chapter.

SQ Set torQue

MCCL command: *a*SQ*n* *a* = Axis number *n* = integer or real >= -10, <= 10
compatibility: MC300, MC302, MC320
see also: QM, PM, VM

Sets maximum output level for servos. When an axis is placed in torque mode, this command sets a continuous output level. The default units for the command parameter are volts. See the description of **Torque Mode Output Control** in the **Applications Solutions** chapter.

SS Set the ratio of Slave axis

MCCL command : *a*SS*n* *a* = Axis number *n* = integer or real > 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: SM

This command specifies the ratio at which the slave axis (designated by *a*) will move relative to a changed in encoder counts (or steps) of the master axis. As soon as the Set Master command is issued, the slave axis will begin tracking the master axis with the programmed ratio. The controller makes the position calculations using the optimal positions of the master and slave axes when the Set Master command was issued as the starting point. See the description of **Electronic Gearing** in the **Motion Control** chapter.

SV Set Velocity

MCCL command : *a*SV*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: SA, DS

Set the maximum velocity for a given axis. The default units for the command parameter are encoder counts (or steps) per second.

UA set the defa**U**lt **A**xis

MCCL command: UA*n* *n* = integer > 0, <= 8
compatibility: MC300, MC302, MC320, MC360, MC362
see also:
The DCX-PCI300 defaults to setting the default axis to zero. If the user executes a motion or setup command with the axis specifier missing, the default axis will be used. In most cases a motion or setup command issued to axis zero commands that operation to all axes. By defining a non-zero default axis, the user can execute 'generic' macro's (no axis number specified) to any axis.

This command is used to define a default axis. After issuing this command, any commanded move, setup, etc. command that utilizes an axis designator (*a*) will execute the command to the axis specified by parameter n. To query the controller as to the current default axis use the **G**et defa**U**lt axis (**GU**) command.

```
MD10,MR1000                             ;Macro 10 will execute a relative move
                                        ;of 1000 counts to the default axis
                                        ;(defined by the User Axis command).
                                        ;Note that the move command does not
                                        ;include the axis designator a.
UA1,MC10                                ;Define axis #1 as the default axis,
                                        ;call macro ten to move 1000 counts
UA2,MC10                                ;Define axis #2 as the default axis,
                                        ;call macro ten to move 1000 counts
```

UK **U**ser **K**onstant

MCCL command: *a*UK*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also:
This command is used to define the units to be used for setting the feed forward (velocity, acceleration, & deceleration), output deadband, and output offset parameters. The default setting is 1.0. This command does not take effect until after a **M**otor o**N** (***a*****MN**) command. See the description of **Defining User Units** in the **Application Solutions** chapter.

UO **U**ser **O**ffset

MCCL command: aUOn *a* = Axis number *n* = integer or real
compatibility: MC300, MC302, MC320, MC360, MC362
see also:
This command is used to define a 'work area zero' position. Use parameter n to define the distance from the servo or stepper home position, to the 'work area zero' position. This offset distance must use the same units as currently defined by the User Scaling command. This command does not take effect until after a **M**otor o**N** (***a*****MN**) command. See the description of **Defining User Units** in the **Application Solutions** chapter.

UP Use Physical axis addressing

MCCL command: *a*UP*n* *a* = Axis number *n* = integer > 0, < 8
compatibility: MC300, MC302, MC320, MC360, MC362
see also:

This command is used to reassign the axis number of a DCX motion module. The value ***a*** should equal the new axis designator. The parameter ***n*** should equal the current physical location of the motor module. Prior to reassigning axis numbers all default axis assignments must be cleared by issuing the **UP** command with no values expressed for ***a*** or ***n***. If dual axis modules are installed and the UP command is issued with parameter *n* = 0 both of the axes must be reassigned, otherwise the second axis of the dual module would cease to exist. When assigning axis numbers for dual modues the UP command for both axes must be issued at the same time. See the description of **Physical Assignment of Axes Numbers** in the **Application Solutions** chapter.

UR User Rate

MCCL command: *a*UR*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also:

This command is used to configure an axis for commands in user units. The default setting is 1.0. This command does not take effect until after a **M**otor o**N** (***a*****MN**) command. See the description of **Defining User Units** in the **Application Solutions** chapter.

US User Scale

MCCL command : *a*US*n* *a* = Axis number *n* = integer or real
compatibility: MC300, MC302, MC320, MC360, MC362
see also:

This command is used to configure an axis for commands in user units. The default setting is 1.0. This command does not take effect until after a **M**otor o**N** (***a*****MN**) command. See the description of **Defining User Units** in the **Application Solutions** chapter.

UT User Time

MCCL command: *a*UT*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also:

This command is used to define the units of time for Wait commands (WA, WS, WT). The default setting is seconds. This command does not take effect until after a **M**otor o**N** (***a*****MN**) command. Note – The **UT** command only effects the time base in the **task or command interface** from which it was issued. See the description of **Defining User Units** in the **Application Solutions** chapter.

UZ set the User Zero position

MCCL command: *a*UZ*n* *a* = Axis number *n* = integer or real
compatibility: MC300, MC302, MC320, MC360, MC362
see also:

This command is used to define a part program zero position. This command does not take effect until after a **M**otor o**N** (***a*****MN**) command. See the description of **Defining User Units** in the **Application Solutions** chapter.

VA Vector Acceleration

MCCL command: *a*VA*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: CP, VD, VV

This command specifies the acceleration rate for motion along a contour path. It should be issued to the controlling axis prior to the first Contour Path command. It can also be issued to the controlling axis while motion is in progress, but it will take effect immediately, and be used for all succeeding motion. See the description of **Contour Motion (lines and arcs)** in the **Motion Control chapter**.

VD Vector Deceleration

MCCL command: *a*VD*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: CP, VA, VV

This command specifies the deceleration rate for motion along a contour path. It should be issued to the controlling axis prior to the first Contour Path command. It can also be issued to the controlling axis while motion is in progress, but it will take effect immediately, and be used for all succeeding motion. See the description of **Contour Motion (lines and arcs)** in the **Motion Control chapter**.

VG Velocity Gain

MCCL command : aVGn *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: AG, DG

Sets the feed forward gain of the servo PID-FF loop. The default units for the parameter to this command are volts per encoder counts per second. For example, if the Velocity gain of a servo is set to 0.0001, the feed forward component of the modules output will be 1 volt at a speed of 10000 encoder counts per second (0.0001 * 10000 = 1). The parameter to this command can be a positive or negative number. This command should not be used for open loop stepper motors.

VO Velocity Override

MCCL command : aVOn *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: CP, VV

Sets a multiplying factor that will be applied to the velocity of both servo and stepper motors. The default setting (multiplying factor) for this command is 1. This command does not support jogging. For contour moves (linear, circular) the axis identifier ‘a’ should be the axis number of the ‘controlling’ axis of the contour group. See the description of **Contour Motion (lines and arcs)** in the **Motion Control chapter**.

VV Vector Velocity

MCCL command: *a*VV*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: CP, VA, VD

This command specifies the maximum velocity for motion along a contour path. It should be issued to the controlling axis prior to the first Contour Path command. When a Contour Path command is issued, the current vector velocity will be stored with the move in the motion table. The Vector Velocity command can also be issued to the controlling axis while motion is in progress, but it won't have any effect on the contour path motions already issued. To adjust the velocity of motions already in progress, use the Velocity Override command. See the description of **Contour Motion (lines and arcs)** in the **Motion Control chapter**.

YF Yes IIR Filter

MCCL command : *a*YF *a* = Axis number
compatibility: MC300, MC320
see also: GF, NF, FL, ZF

Enables the IIR filter, which can be used to implement a notch or low pass filter. See the description of **User Defined Filters (Notch and Low Pass)** in the **Application Solutions chapter**.

ZF Zero IIR Filter coefficients

MCCL command : *a*ZF *a* = Axis number
compatibility: MC300, MC320
see also: GF, NF, FL, YF

Clears all IIR filter coefficients to zero and resets the filter load index to 0. See the description of **User Defined Filters (Notch and Low Pass)** in the **Application Solutions chapter**.

Chapter Contents

Chapter 8

Mode Commands

CM Contour Mode

MCCL command: *a*CM*n* *a* = Axis number *n* = integer > 0, <8
compatibility: MC300, MC302, MC320, MC360, MC362
see also: CP

This command places a servo or stepper motor in the Contour Mode of operation. The parameter *n* to this command specifies the controlling axis for a group of axes to be included in contour path commands. The controlling axis should be the lowest numbered axis in the group. Contour Mode is terminated by issuing a Position (aPM) or Velocity Mode (aVMn) command to all axes in the group. The controlling axis should be taken out of contour mode last. See the description of **Contour Motion (lines and arcs)** in the **Motion Control chapter**.

GM enable Gain Mode

MCCL command: *a*GM *a* = Axis number
compatibility: MC300, MC302, MC320
see also: IL, SD, SG, SI

This command places a servo in the Gain Mode of operation. In this mode, the servo can be commanded to execute moves to specific positions. However, no velocity profile (maximum velocity, acceleration, or deceleration) will be calculated. The servo will be driven to the new target based **only upon** the output of the PID loop.

IM Input Mode

MCCL command: *a*IM*n* *a* = Axis number *n* = 0 or 1 *n* = 0, 1, 128, 129
compatibility: MC360
see also:

The Input Mode command issued with a parameter of 1 enables closed loop stepper motion. Issued with *n* = 0 disables closed loop motion. See the description of **DCX Stepper Basics** in the **Motion Control** chapter. If parameter *n* MSB is set to 1 (128) the user can change open loop stepper PID parameters

PM **P**osition **M**ode

MCCL command : *a*PM *a* = Axis number
compatibility: MC300, MC302, MC320, MC360, MC362
see also: MA, MR

This command places a servo or stepper motor in the Position Mode of operation. In this mode, it can be commanded to execute moves to specific positions. The moves will be carried out using a trapezoidal, parabolic or S-curve velocity profile. When in the Position Mode with trapezoidal velocity profile selected, servos can change the move destination while the move is in progress. With stepper motors, the destination can be changed as long as it doesn't require the motors direction to change. If it is necessary to change the direction of a stepper motor, it must first be stopped and a new move command issued. Upon start up, or after a Reset, motors will be placed in the Position Mode. See the description of **Point to Point Motion** in the **Motion Control** chapter.

SM **S**et **M**aster/Slave mode

MCCL command: *a*SM*n* *a* = Axis number *n* = integer > 0, <= 101
compatibility: MC300, MC302, MC320, MC360, MC362
see also: SS

This command will cause axis ‘a’ to be "slaved" to a "master" axis n with a ratio specified by the Set Slave ratio command. Alternatively, this command can slave one axis to two master axes for tangential knife control in cutter applications. In this case the command parameter is determined by the following algorithm:

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

Issuing this command with a parameter of zero to the slave axis, will terminate the connection to the master axis. See the Master/Slave Motion section of this manual for further details. See the descriptions of **Electronic Gearing** and **Tangential Knife Control** in this manual chapter.

QM tor**Q**ue **M**ode

MCCL command: *a*QM *a* = Axis number
compatibility: MC300, MC302, MC320
see also: SQ

This command places a servo (not valid for open or closed loop steppers) in the Torque Mode of operation. This command does **not imply** that the torque generated by or current across the motor is monitored or controlled by the DCX controller. In this mode, the output drive signal to the motor will hold a constant level as specified with **S**et tor**Q**ue command. The parameter to this command has default units of volts. In this mode of operation, the servo motor control module (MC300, MC302, MC320) is ‘turned into’ a programmable power supply. As such, the change of position as indicated by the encoder of an axis, while still recorded by the servo module, will have no affect on the operation of the controller. See the description of **Torque Mode Output Control** in the **Applications Solutions** chapter.

VM Velocity Mode

MCCL command: aVM *a* = Axis number
compatibility: MC300, MC302, MC320, MC360, MC362
see also: DI, GO

This command places a motor in the Velocity Mode of operation. In this mode, the motor can be commanded to move in either direction at a given velocity. The motor will move in that direction until commanded to stop. In Velocity Mode the user can specify the direction for the motor to move using the DIrection (DI) command. While a motor is moving, the user can issue new direction or velocity commands. The acceleration or deceleration rate at which the motor velocity will change is determined by the Set Acceleration (SA) and Deceleration Set (DS) commands. See the description of **Continuous Velocity Motion** in the **Motion Control** chapter.

Chapter Contents

Chapter
9

Motion Commands

AB ABort motion

MCCL command:	*a*AB *a* = Axis number (0 = Abort motion on all axes)
compatibility:	MC300, MC302, MC320, MC360, MC362
see also:	ST

This command serves as an emergency stop. For a servo, motion stops abruptly but leaves the position feedback loop (PID) and the amplifier enabled. For a stepper motor, the pulses from the module will be disabled immediately. For both servos and stepper motors, the target position of the axis is set equal to the present position. This command can be issued to a specific axis, or can be issued to all axes simultaneously by using an axis specifier of 0.

```
example:  2AB                              ;causes the motion of axis 2 to be
                                           ;aborted
```

AF Auxiliary encoder Find index mark

MCCL command :	*a*AF *a* = Axis number
compatibility:	MC300, MC320, MC360
see also:	AH, AX

This command is used to capture the index position of an auxiliary encoder. After issuing the AF command, when the auxiliary encoder index mark is next asserted status bit 27 will be set. The AX command (get **A**uxiliary encoder inde**X** position) can then be used to load the encoder index position into the accumulator. The AF only arms the capturing the auxiliary encoder index, it does not delay additional command execution. See the description of **Auxiliary Encoders** in the **Application Solutions** chapter.

BC Begin Compare

MCCL command: *a*BC*n* *a* = Axis number *n* = integer
compatibility: MC300, MC360
see also: CG, DR, HS, LC, LS, MS

This command is used to start or terminate a position compare event. As many as 512 compare positions can be stored for DCX each motion control module. To start a position compare event issue the Begin Compare command with parameter *n* = the number of compare position. If parameter *n* is expressed as a negative number the position compare operation will be terminated. See the description of **Position Compare** in the **Application Solutions** chapter.

BF Backlash compensation oFf

MCCL command : *a*BF *a* = Axis number
compatibility: MC300, MC302, MC320
see also: BD, BN

Use this command to disable backlash compensation. As soon as this command is executed, the motor will move to its uncompensated position. See the description of **Backlash Compensation** in the **Application Solutions** chapter.

BN Backlash compensation oN

MCCL command: *a*BN *a* = Axis number
compatibility: MC300, MC302, MC320
see also: BD, BF

Use this command to enable backlash compensation. It should be issued after the backlash compensation distance has set been with the BD command. Prior to issuing the Backlash Compensation On command, 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 Tell Position, Tell Target and Tell Optimal commands will be adjusted to reflect the ideal positions (as if no mechanical backlash were present). See the description of **Backlash Compensation** in the **Application Solutions** chapter.

CA arc Center Absolute

MCCL command: *a*CA*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: CM, CP

This command is used to specify the center of an arc for a Contour Path motion. Since the arc motion is performed by two axes, this command (or the arc Center Relative command) should occur twice in a Contour Path command that initiates the arc motion. The parameter to this command specifies the center of the arc for the selected axis in absolute user units. See the description of **Contour Motion** in the **Motion Control** chapter.

CB Capture Begin

MCCL command: *a*CB*n* *a* = Axis number *n* = integer
compatibility: MC300, MC360
see also: CG, DR, HS, LS, MS

This command is used to store the position of an axis when the encoder index pulse transitions to active. As many as 512 captured position points can be stored for DCX each motion control module. When parameter n is an integer between 1 and 512 position capture will begin. If parameter *n* is expressed as a negative number position capture will be terminated. See the description of **Position Capture** in the **Application Solutions** chapter.

CD Contour Distance

MCCL command : *a*CD*n* *a* = Axis number of the controlling axis *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: CM, CP

For user defined contour moves, this command is used to specify the distance as measured along the path from the contour path starting point to the end of the next motion. For typical orthogonal (X, Y, Z) geometry, the DCX calculates the contour move distance ($\sqrt{X^2+Y^2+Z^2}$) based on the target positions specified by the move absolute and/or move relative commands. Parameter n of the Contour Distance command allows the user to enter a custom contour distance to be used for trajectory generation. The value n is used as an ending point for the contour path motion and to determine the proper velocity over the motion segment. See the description of **Contour Motion** in the **Motion Control** chapter.

CP define the Contour Path

MCCL command: *a*CP*n* *a* = Axis number *n* = integer 0, 1, 2, or 3
compatibility: MC300, MC302, MC320, MC360, MC362
see also: CM

This command is used to form a 'compound' command that specifies a multi axis motion. The compound command must begin with the Contour Path command, followed by a variable number of other motion commands. The axis number used with the Contour Path command must be the controlling axis in a group of motors that have been previously placed in Contour Mode. The parameter to the Contour Path command selects either a linear, arc or 'user defined' motion. The table below list the type of motion each parameter value specifies, and the acceptable commands that can be include in the compound command. See the description of **Contour Motion** in the **Motion Control** chapter.

```
1CP1,1GH,2GH,3GH,1VV60000
1CP1,1MA10000,2MA20000,3MR-5000,1VV30000
1CP1,1GH,2GH,3GH
1CP2,1CA20000,2CA0,1MA40000,2MA0
1CP3,1CR-20000,2CR0,1MR-40000,2MR0
```

Parameter n	*Motion Type*	*Compatible Commands*
0	User defined	CD,GH,MA,MR,VV
1	Linear	GH,MA,MR,VV
2	Clockwise arc	CA,CR,GH,MA,MR,VV
3	Counter-Clockwise arc	CA,CR,GH,MA,MR,VV

CR arc **C**enter **R**elative

MCCL command: *a*CR*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: CM, CP

This command is used to specify the center of an arc for a Contour Path motion. Since the arc motion is performed by two axes, this command (or the arc Center Absolute command) should occur twice in a Contour Path command that initiates the arc motion. The parameter to this command specifies the center of the arc for the selected axis in user units, relative to its' target position prior to beginning the arc motion. See the description of **Contour Motion** in the **Motion Control** chapter.

EA arc **E**nding **A**ngle absolute

MCCL command: *a*EA*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: CM, CP

This command is used to specify the ending angle (end point) of a contour arc move. The parameter *n* is expressed as an absolute angle relative to when the axes where last homed. This command would be used in conjunction with the Center Absolute, Center Relative, or aRc Radius commands. See the description of **Contour Motion** in the **Motion Control** chapter.

EL home **E**dge **L**atch

MCCL command : *a*EL*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC360, MC362
see also: FE, WE

This command is used to arm the capturing of the home sensor position of an open loop stepper. When combined with Wait for Edge (***a*WE**) it performs the same operation as the Find Edge command. After the Edge Latch command is issued, when the home sensor is activated, the step count position of the home sensor will be captured and the Home found status bit (status bit 10) will be set. The Wait for Edge and Motor oN commands are then issued to define the location of the home sensor as position *n*. The difference between using the Edge Latch and Wait for Edge commands versus the Find Edge command is that Find Edge will stall the command interpreter (no additional command execution) until the home sensor is captured. The Edge Latch command arms the capture of the home sensor but does not stop additional command execution. See the description of **Homing Axes** in the **Motion Control** chapter.

ER arc **E**nding angle **R**elative

MCCL command: *a*ER*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: CM, CP

This command is used to specify the ending angle (end point) of a contour arc move. The parameter *n* is expressed as an angle relative to when the axes where last homed. This command would be used in conjunction with the Center Absolute, Center Relative, or aRc Radius commands. See the description of **Contour Motion** in the **Motion Control** chapter.

FE Find Edge

MCCL command: *a*FE*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC360, MC362
see also: EL, WE

This command is used to reference (home) the reported position of an open loop stepper to the home sensor. This is a conditional command and will not complete execution until the activated home sensor has been captured. The FE command performs two operations, capturing of the home sensor event, and storing the step count position of the home sensor in preparation for redefing the reported position of the sensor. Upon completion of this command (and after stopping the motor), issuing the Motor oN command will cause the step count position of the sensor to be redefined to position *n.* The current position will be redefined to *n* plus or minus the distance in step counts from the sensor.

If for any reason the activation of the home sensor is not captured the controller will stall and no additional commands will be executed. In this case to clear the FE and unlock the command interpreter enter the escape key. To home a open loop servo without the possibility of locking up the controller use the Edge Latch (aELn) command to enable capturing of the sensor location and the Wait for Edge (aWE) command to store the step count position of the sensor in preparation for redefing the reported position of the sensor. To avoid stalling the command interpreter it is recommended that the FE command only be issued from a macro running as a background task. Note: The FE command does not start or stop motion, it is up to the user to initiate motion prior to issuing the find edge command. See the description of **Homing Axes** in the **Motion Control** chapter.

FI Find Index

MCCL command: *a*FI*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360 (closed loop only)
see also: DH, FE, IA, WI

This command is used to reference (home) the reported position of a closed loop system to the index mark of an encoder. This is a conditional command and will not complete execution until the index mark of the encoder has been captured. The FI command performs two operations, capturing of the encoder index event, and storing the encoder position of the index in preparation for redefing the reported position of the encoder. Upon completion of this command (and after stopping the motor), issuing the Motor oN command will cause the encoder position of the index to be redefined to position *n.* The current position will be redefined to *n* plus or minus the distance in encoder counts from the index mark.

If for any reason the encoder index mark is not captured the controller will stall and no additional commands will be executed. In this case to clear the FI and unlock the command interpreter enter the escape key. To home a closed loop system without the possibility of locking up the controller use the Index Arm (aIAn) command to enable capturing of the index mark location and the Wait for Index (aWI) command to store the encoder position of the index in preparation for redefing the reported position of the encoder. To avoid stalling the command interpreter it is recommended that the FI command only be issued from a macro running as a background task. Note: The FI command does not start or stop motion, it is up to the user to initiate motion prior to issuing the find index command.

Since an index pulse may occur at numerous points of a servo's travel (once per revolution of a rotary encoder), typical closed loop systems will require a coarse home signal to "qualify" the index pulse. The encoder and / or coarse home sensor would then be adjusted so that the index pulse occurs while the sensor is active. See the description of **Homing Axes** in the **Motion Control** chapter.

GH Go Home

MCCL command: *a*GH *a* = Axis number
compatibility: MC300, MC302, MC320, MC360, MC362
see also: MA, MC, MD

Causes the specified axis or axes to move to the offset position that was specified when the last DH, IA & WI, EL & WE, FI, or FE command was issued. This is equivalent to a Move Absolute command, where the destination is 0 or the offset of the home position.

GO GO

MCCL command: *a*GO*n* *a* = Axis number *n* = integer 0 or 1
compatibility: MC300, MC302, MC320, MC360, MC362
see also: CM, VM

Causes one or all axes to begin motion in velocity or contour mode. In contour mode, synchronization must be on. The parameter to this command is only used for contour mode, and determines whether the motions will be linearly interpolated (n = 0), or a cubic spline (n = 1).

HO HOme

MCCL command: *a*HO *a* = Axis number
compatibility: MC300, MC302, MC320, MC360, MC362
see also: MC, MD

This command will cause a user defined macro to be executed. It is up to the user to define the macro to carry out the appropriate homing sequence for that motor (see Find Edge and Find Index commands). Issuing 1HO will cause macro 1 to be executed, issuing 2HO will cause macro 2 to be executed, and so on. Issuing this command with no motor specified will cause macro 9 to be executed. See the description of **Homing Axes** in the **Motion Control** chapter.

IA Index Arm

MCCL command : *a*IA*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360
see also: FI, WI

This command is used to arm the index capture function of a closed loop axis. When combined with Wait for Index (***a*WI**) it performs the same operation as the Find Index command. After the Index Arm command is issued, when the index pulse occurs, the encoder position of the index mark will be captured and the Encoder Index status bit (status bit 10) will be set. The Wait for Index and Motor oN commands are then issued to define the location of the index pulse as position *n*. The difference between using the Index Arm and Wait for Index commands versus the Find Index command is that Find Index will stall the command interpreter (no additional command execution) until the index is captured. The Index Arm command arms the capture of the index but does not stop additional command execution. See the description of **Homing Axes** in the **Motion Control** chapter.

LC Load Compare

MCCL command: *a*LC*n* *a* = Axis number *n* = integer
compatibility: MC300, MC360
see also: BG, CG, DR, HS, LS, MS

This command is used to store the compare positions. As many as 512 position points can be stored for DCX each motion control module. See the description of **Position Compare** in the **Application Solutions** chapter.

LP Learn Position

MCCL command: *a*LP*n* *a* = Axis number *n* = integer >= 0, <=255
compatibility: MC300, MC302, MC320, MC360, MC362
see also: LT, MP

Used for storing the current position of one or more axes in the DCX's point memory. Positions stored in the point memory can be used by the Move to Point command to repeat a stored motion pattern. The command parameter n specifies the entry in the point memory where the position will be stored.

If the LP command is issued with an axis specifier of 0, the positions of all axes on the DCX board will be stored in the point memory. If the command is issued with a non-zero axis specifier, only the position of that axis will be stored in the point memory. No other positions in the point memory will be changed. See the description of **Learning/ Teaching Points** in the **Application Solutions** chapter.

LT Learn Target

MCCL command: *a*LT*n* *a* = Axis number *n* = integer >= 0, <=255
compatibility: MC300, MC302, MC320, MC360, MC362
see also: LP, MP

Similar to the LP command, but stores the axes' target position (versus actual position). Motion of an axis is not required for storing target positions. This makes it possible to download coordinates from a host computer or CAD system.

Turn off the motor drive outputs with the MF command, then send motion commands prior to the LT command. Targets stored in the point memory can be used by the Move to Point command to repeat a stored motion pattern. The command parameter n specifies the entry in the point memory where the position will be stored. If the LT command is issued with an axis specifier of 0, the targets of all axes on the DCX board will be stored in the point memory. If the command is issued with a non-zero axis specifier, only the target of that axis will be stored in the point memory. No other targets in the point memory will be changed. See the description of **Learning/ Teaching Points** in the **Application Solutions** chapter.

MA Move Absolute

MCCL command : *a*MA*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360, MC362
see also: MR, PM

This command generates a motion to absolute position *n*. 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. See the description of **Point to Point Motion** in the **Motion Control** chapter.

MF Motor oFf

MCCL command : *a*MF *a* = Axis number
compatibility: MC300, MC302, MC320, MC360, MC362
see also: MN

Issuing this command will place one or all servos and stepper motors in the "off" state. For servos, the Analog Signal will go to the null level, the servo loop (PID) will terminate, and the Amplifier Enable output will go inactive. For stepper motors, the Motor On output will go inactive. This command can be used to prevent unwanted motion or to allow manual positioning of the servo or stepper motor.

MN Motor oN

MCCL command: aMN *a* = Axis number
compatibility: MC300, MC302, MC320, MC360, MC362
see also: MF

Use this command to place one or all servos and stepper motors in the on state. If an axis is off when this command is issued, the target and optimal (commanded) positions will be set to the motor's current position. This can cause a change in the axis' reported position based on new user units. At the same time, a servo module's Amplifier Enable or a stepper motor module's drive enable output signal will go active. This has the effect of causing servo and stepper motors to hold their current position. If an axis is already on when this command is issued, the position values will be set for the current user units, but the commanded encoder or pulse position will not be changed.

MP Move to Point

MCCL command: *a*MP*n* *a* = Axis number *n* = integer >= 0, <=255
compatibility: MC300, MC302, MC320, MC360, MC362
see also: LP, LT

Used for moving one or more axes to a previously stored point. The command parameter n specifies which entry in the DCX's point memory is to be used as the destination of the move. If the MP command is issued with an axis specifier of 0, all axes will move to the positions stored in the point memory for that point. If the command is issued with a non-zero axis specifier, only that axis will move to the position in the point memory. No other axes will be commanded to move. Points can be stored in the point memory with the Learn Point (LP) and Learn Target LT) commands. See the description of **Learning/ Teaching Points** in the **Application Solutions** chapter.

MR Move Relative

MCCL command : *a*MR*n* *a* = Axis number *n* = integer or real
compatibility: MC300, MC302, MC320, MC360, MC362
see also: MA, PM

This command generates a motion of relative distance *n*. 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. See the description of **Point to Point Motion** in the **Motion Control** chapter.

NC Next Compare position

MCCL command: *a*NC*n* *a* = Axis number *n* = integer or real
compatibility: MC300, MC360
see also: BG, CG, DR, HS, LS, MS

This command is used to define a fixed increment distance that will be used to for position compare. See the description of **Position Compare** in the **Application Solutions** chapter.

NS No Synchronization

MCCL command: *a*NS *a* = Axis number
compatibility: MC300, MC302, MC320, MC360, MC362
see also: SN

This command turns synchronization off in contour path motions. It should be issued to the controlling axis of the contour group. See the description of **Contour Motion** in the **Motion Control** chapter.

OC Output mode for Compare

MCCL command: *a*OC*n* *a* = Axis number *n* = integer 0, 1, 2, 3
compatibility: MC300, MC360
see also: BG, NC, OP

This command is used to define the mode of operation for the Position Compare output. The possible values for parameter n are:

Parameter	***Output Mode***
0	**Disabled (default)**
1	Static
2	Toggle
3	One-shot

See the description of **Position Compare** in the **Application Solutions** chapter.

OP Output compare Period

MCCL command: *a*OP*n* *a* = Axis number *n* = integer or real > 0
compatibility: MC300, MC360
see also: BG, NC,OC

This command is used to define the period of the Position Compare output when configured for one-shot mode. The parameter to this command is in units of seconds. The module hardware has a one-shot timer that can generate a pulse period of from 1 microsecond to 1 second. For time periods less then 50 milliseconds the timer has 1 microsecond resolution. For time periods greater than or equal to 50 milliseconds, it has 50 microsecond resolution. With either short or long duration pulses, the output signal is guaranteed to go active within the 1/2 microsecond latency of the compare function (not counting the delay for the optical isolator). See the description of **Position Compare** in the **Application Solutions** chapter.

PP Parabolic Profile

MCCL command: *a*PP *a* = Axis number
compatibility: MC360, MC362
see also: PS, PT

This command causes the respective stepper motor to perform ***point to point*** motions with a triangular acceleration profile. The resulting velocity profile is parabolic. Motion with this profile is limited to ***position and contour*** mode moves, where the acceleration, deceleration, , velocity, and destination ***don't change during the move***. See the description of **Getting Started with Servo's and Getting Started with Steppers** in the **Motion Control** chapter.

PR Record axis data

MCCL command : *a*PR*n* *a* = Axis number *n* = integer > 0, <=512
compatibility: MC300, MC302, MC320
see also:

This command is used to begin the recording of motion data (actual position, optimal position, and DAC output) for an axis. See the description of **Record and display Motion Data** in the **Application Solutions** chapter.

PS Profile S-curve

MCCL command: *a*PS *a* = Axis number
compatibility: MC300, MC302, MC320
see also: PP, PT

This command causes the respective servo motor to perform ***point to point*** motions with a sinusoidal acceleration profile. The resulting velocity profile is trapezoidal with rounded corners, thus the name S-curve. Motion with this profile is limited to ***position and contour*** mode moves, where the acceleration, deceleration, velocity, and destination ***don't change during the move***. See the description of **Getting Started with Servo's and Getting Started with Steppers** in the **Motion Control** chapter.

PT Profile Trapezoidal

MCCL command:	*a*PT *a* = Axis number
compatibility:	MC300, MC302, MC320, MC360, MC362
see also:	PS, PT

This command causes the respective servo or stepper motor to perform point to point motions with a constant acceleration profile. The resulting velocity profile is trapezoidal. When motion is being performed with this profile, the acceleration, velocity, and destination can be changed at any time during the move. See the description of **Getting Started with Servo's and Getting Started with Steppers** in the **Motion Control** chapter.

RC Restore Configuration

MCCL command:	*a*RC*n* n = Axis number *n* = integer > 0, <=127
compatibility:	MC300, MC302, MC320, MC360, MC362
see also:	SC

Not supported

RR Radius of aRc

MCCL command:	*a*RR*n* *a* = Axis number *n* = integer or real >= 0
compatibility:	MC300, MC302, MC320, MC360, MC362
see also:	CM, CP

This command specifies the radius of a contour mode arc. For an arc of less than 180 degrees the parameter n should be a positive value equal to the radius of the arc. For an arc of greater than 180 degrees the parameter n should be a negative value equal to the radius of the arc. See the description of **Contour Motion** in the **Motion Control** chapter.

SC Save Configuration

MCCL command:	*a*SC*n* *n* = Axis number *n* = integer > 0, <=127
compatibility:	MC300, MC302, MC320, MC360, MC362
see also:	RC

Not supported

SN Synchronization oN

MCCL command:	aSN *a* = Axis number
compatibility:	MC300, MC302, MC320, MC360, MC362
see also:	GO, CP, NS

This command turns synchronization on for contour path motion. This command can be issued to the controlling axis prior to Contour Path commands. With synchronization on, no motion will occur when a Contour Path command is issued, until a succeeding GO command is issued to the controlling axis. See the description of **Contour Motion** in the **Motion Control** chapter.

ST STop

MCCL command: *a*ST *a* = Axis number (0 = Stop motion on all axes)
compatibility: MC300, MC302, MC320, MC360, MC362
see also: AB, MF

This command is used to stop one or all motors. It differs from the Abort command in that motors will decelerate at their preset rate, instead of stopping abruptly. This command can be issued to a specific axis, or can be issued to all axes simultaneously by using an axis specifier of 0. See the description of **Continuous Velocity Motion** in the **Motion Control** chapter.

Chapter Contents

Chapter 10

Reporting Commands

The commands in this section are used to display the current values of internal controller data. Some of these values are 'real' numbers that must be displayed with fractional parts. In order to provide compatibility with older products that don't support real numbers, and to provide flexibility in the display format, certain reporting commands accept a parameter that sets the number of digits displayed to the right of the decimal point. These commands will show a 'p' as a parameter in their descriptions.

For ASCII command interfaces, *p* can be replaced with a number between 0 and 1 and the tenths digit will be interpreted as the number of decimal digits to display to the right of the decimal point. If no parameter is used with the command, or a parameter of 0 is used, the reply to the command will be an integer with no decimal point. Example:

```
 ;If axis 1 position is 123.4567
   1TP;       DCX replies 123
   1TP0;      DCX replies 123
   1TP.1;     DCX replies 123.4
   1TP.3;     DCX replies 123.456
```

For the Binary command interface, the reporting commands that have a 'p' listed as their parameter will accept an integer value of 0, 1 or 2 in place of *p*. A value of 0 will generate an integer reply, a value of 1 will generate a 64 bit floating point reply, and a value of 2 will generate a 32 bit floating point reply. See the appendix describing the DCX Binary Command Interface for more details on these reply formats.

AT **A**uxiliary encoder **T**ell position

MCCL command: *a*AT*p* a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility: MC300, MC320, MC360
see also: AH

Reports the absolute position of the auxiliary encoder of an axis. To read the primary encoder or stepper position, see the Tell Position command.

AZ Auxiliary encoder tell index

MCCL command: *a*AZ*p* a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility: MC300, MC320, MC360
see also:

Reports the position where the auxiliary encoder's index pulse was observed. This position is relative to the encoder's position when the controller was reset or an Auxiliary encoder define Home command was issued to the axis.

CG Capture Get count

MCCL command: *a*CG = Axis number
compatibility: MC300, MC360
see also: CB, TR

Loads the number of captured position points into the accumulator (user register 0). The Tell Register command is then used to report the value. See the description of **Position Capture** in the **Application Solutions** chapter.

DO Display the recorded Optimal position

MCCL command: aDOp *a* = Axis number *p* = integer >= 0, < 512
compatibility: MC300, MC302, MC320
see also: PR

This command is used to report the captured optimal position of an axis. See the description of **Record and display Motion Data** in the **Application Solutions** chapter.

DQ Display the recorded DAC output

MCCL command: aDQp *a* = Axis number *p* = integer >= 0, < 512
compatibility: MC300, MC302, MC320
see also: PR

This command is used to report the captured DAC output of an axis. See the description of **Record and display Motion Data** in the **Application Solutions** chapter.

DR Display Recorded position

MCCL command: aDRp *a* = Axis number *p* = integer >= 0, < 512
compatibility: MC300, MC302, MC320
see also: PR

This command is used to report the captured actual position of an axis. See the description of **Record and display Motion Data** in the **Application Solutions** chapter.

GC Get the Compare count

MCCL command :	*a*GC *a* = Axis number
compatibility:	MC300, MC360
see also:	BG, DR, HS, LS, MS

Loads the number of position events into the accumulator (user register 0). The Tell Register command is then used to report the value. See the description of **Position Compare** in the **Application Solutions** chapter.

TA Tell Analog

MCCL command:	TAx *x* = Channel number *p* = 0, 1, 2, 3, ... (# of MC500/510 modules X 4)
compatibility:	MC500, MC510
see also:	

Reports the digitized analog input signals to MC500 and MC510 modules. The analog input channels on any installed MC500/510 modules will be numbered sequentially starting with channel 1. For each of these channels, the TA command will display a number between 0 and 4096. These numbers are the ratio of the analog input voltage to the reference input voltage multiplied by 4096. See the description of **Analog Inputs** in the **DCX General Purpose I/O** chapter.

TB Tell Breakpoint

MCCL command:	aTBp a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility:	MC300, MC302, MC320, MC360, MC362
see also:	IP, IR

Reports the position where the breakpoint for a motor is placed. Breakpoints are placed with the IP, IR, WP and WR commands. The interpretation of the command parameter p is explained at the beginning of this section.

TC Tell digital Channel

MCCL command:	TC*x* *x* = Channel number
compatibility:	MC400
see also:	

Reports the on/off status of each digital I/O line. This data is reported separately for each channel. The DCX responds by displaying the channel number and a "1" if the channel is "on", or a "0" if the channel is "off".

TD Tell Derivative gain

MCCL command:	aTDp a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility:	MC300, MC302, MC320
see also:	SD

Reports the derivative gain setting for a servo.

TE Tell command interpreter Error

MCCL command: TE
compatibility: Not applicable
see also:
Reports the last command interpreter error (syntax error, invalid character, etc.). For a listing of error codes please refer to the **MCCL Error Codes** chapter.

TF Tell Following error

MCCL command: *a*TF*p* a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility: MC300, MC302, MC320
see also: SE
Reports the current following error of a servo. This error is the difference between the commanded position (calculated by the trajectory generator) and the current position.

TG Tell proportional Gain

MCCL command: *a*TG*p* a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility: MC300, MC302, MC320
see also: SG
Reports the proportional gain setting for a servo.

TI Tell Integral gain

MCCL command: *a*TI*p* a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility: MC300, MC302, MC320
see also: set Integral gain
Reports the integral gain setting for a servo.

TK Tell velocity Konstant

MCCL command: aTKp a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility: MC300, MC302, MC320
see also: VG
Reports the velocity constant for a servo. This is the value that was set with the Velocity Gain command. For a closed loop stepper axis this command reports the Velocity Gain that was set during initialization.

TL Tell integral Limit setting

MCCL command: *a*TL*p* a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility: MC300, MC302, MC320
see also: IL
Reports the integral limit setting for a servo.

TM Tell Macros

MCCL command: TM*n* *n* = integer >= -1, <= 1000
compatibility: N/A
see also: MD, RM

Displays the commands which make up any macros which have been defined. If n = -1, all macros will be displayed. Since macros may be defined in any sequence, the TM command is useful for confirming the existence and/or contents of macro commands. In addition to the contents of macros, this command will also show the amount of memory available for macro storage. See the description of **Macro Commands** in the **Working with MCCL Commands** chapter.

TO Tell Optimal

MCCL command: *a*TO*p* a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility: MC300, MC302, MC320, MC360, MC362
see also:

Reports the desired position for servos and current position for steppers. For servos, the reported value will be different than the position reported by the TP command if a following error is present.

TP Tell Position

MCCL command: *a*TP*p* a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility: MC300, MC302, MC320, MC360, MC362
see also: DH, FI

Reports the absolute position of axis a. It may be used to monitor motion during both Motor oN (MN) and Motor ofF (MF) states. The interpretation of the command parameter p is explained at the beginning of this section.

TQ Tell torQue

MCCL command: *a*TQ*p* a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility: MC300, MC302, MC320
see also: QM, SQ

Reports the current setting for the Set torQue command. See the description of the **Torque Mode Output Control** in the **Application Solutions** chapter.

TR Tell Register '*n*'

MCCL command: TR*n* *n* = integer >= 0, <= 255
compatibility: N/A
see also: AL, AR

Displays the contents of User Register *n*. When the command parameter is set to 0 (or not specified), this command reports the contents of User Register zero, which is the accumulator.

TS Tell axis Status

MCCL command : *a*TS*n* a = Axis number *n*= integer

compatibility: MC300, MC302, MC320, MC360, MC362

see also:

Reports the status of an axis. If the command parameter is 0, the response is coded into a single 32 bit value. If the parameter has a value between 1 and 31 inclusive, the state of the respective bit is displayed as a '0' for reset, and a '1' for set. Using a command parameter greater that 32 results in formatted status displays. Status flags that are not valid for steppers are indicated by an asterisk. The meaning of each bit is listed below:

Bit number	***Description***
0	Busy (motor data being updated)
1	Motor On
2	At Target
3	Trajectory Complete (Optimal = Target)
4	Direction (0 = positive, 1 = negative)
5	Reserved
6	Motor homed ***
7	Motor Error (Limit +/- tripped, max. following error exceeded) *
8	Looking For Index (FI, WI) *
9	Looking For Edge (FE, WE)
10	Index found (closed loop) / Edge found (open loop stepper)
11	Position Capture flag
12	Breakpoint Reached (IP, IR, WP, WR)
13	Exceeded Max. Following Error *
14	Servo Amplifier / Stepper Driver Fault Enabled
15	Servo Amplifier / Stepper Driver Fault Tripped
16	Hard Limit Positive Input Enabled
17	Hard Limit Positive Tripped
18	Hard Limit Negative Input Enabled
19	Hard Limit Negative Tripped
20	Soft Motion Limit High Enabled
21	Soft Motion Limit High Tripped
22	Soft Motion Limit Low Enabled
23	Soft Motion Limit Low Tripped
24	Encoder Index (closed loop) / Home (open loop stepper) ****
25	Encoder Coarse home (current state)
26	Servo Amplifier / Stepper Driver Fault (current state)
27	Auxiliary Encoder Index
28	Limit Positive Input Active (current state)
29	Limit Negative Input Active (current state)
30	Null **
31	Reserved

* Not supported by open loop stepper axes

** Supported by MC360 only

*** Set after completed homing procedure (closed loop IA & WI or open loop EL & We)

**** Defaults to displaying the current state of Index / Home input. Issuing Index Arm (Index) or Edge Latch (Home) commands will cause axis to set and latch the bit 24 once the input has been activated. Issue Motor off / Motor On commands to return to reporting the current state of the input.

```
example:  DM                            ;Place DCX in Decimal Output Mode
      1TS                               ;report the status of axis #1


  DCX returns:   01  268439566    ;status =
                                        ;bit 28 set - limit + input active
                                        ;but limits error checking is not
                                        ;enabled (bit 16 cleared)
                                        ;bit 12 set - breakpoint reached
                                        ;bit 3 set - trajectory complete
                                        ;bit 2 set - axis At target
                                        ;bit 1 set – motor on


example:  HM                            ;Place DCX in Hexadecimal Output Mode
      1TS                               ;report the status of axis #1


  DCX returns:   01  1000100E           ;status =
                                        ;bit 28 set - limit + input active
                                        ;but limits error checking is not
                                        ;enabled (bit 16 cleared)
                                        ;bit 12 set - breakpoint reached
                                        ;bit 3 set - trajectory complete
                                        ;bit 2 set - axis At target
                                        ;bit 1 set – motor on


example:       1TS32
```

```
DCX returns:
MOTOR STATUS:
Motor On
At Target
Trajectory Complete
Direction = Positive
Jogging Disabled
Not Homed
No Motor Error
Not Looking For Index
Not Looking For Edge
Breakpoint Reached
Max. Following Error Not Exceeded
Amplifier Fault Disabled
Hard Motion Limit Positive Disabled
Hard Motion Limit Negative Disabled
Soft Motion Limit High Disabled
Soft Motion Limit Low Disabled
Index Input = 1
Coarse Home Input = 0
Amplifier Fault Input = 0
Auxiliary Encoder Index = 0
Limit Positive Input = 1
Limit Negative Input = 0
User Input 1 = 0
User Input 2 = 0
```

Axis Auxiliary Status

Bit number	Description
0	Hard Motion Limit Mode = Stop abrupt
1	Hard Motion Limit Mode = Decelerate to a stop
2	Soft Motion Limit Mode = Stop abrupt
3	Soft Motion Limit Mode = Decelerate to a stop
4	Rate = Low
5	Rate = Medium
6	Rate = High
7	IIR Filter = On
8	Synchronization = On
9	Synchronization = Ready
10	Full Step (stepper on) = On
11	Full Current (stepper only) = On
12	Servo Phasing = Standard
13	Intermediate position (Stepper only) = Yes
14	Backlash Compensation = On
15	Open loop PID enabled
16	Backlash Compensation distance +
17	Backlash Compensation distance -
18	Axis slaved to master = Yes
19	Axis slaved to contour = Yes
20	Axis slaved to master encoder = Yes
21	Axis slaved to master index mark = Yes
22	Positive Limit Invert = On
23	Negative Limit Invert = On

To report the state of all Auxiliary Axis Status bits issued the Tell Status command with parameter *n* = 33:

```
example:       1TS33
```

```
example:       1TS34
```

```
DCX returns:
Motor status: 100100c
Auxiliary status: 20
Position count: 0 *
Optimal count: 0 *
Index count: 0 *
Position: 0.000000 *
Target: 0.000000 *
Optimal position: 0.000000 *
Break position: 0.000000 *
Dead band: 0.000000 *
Maximum following error: 1024.000000 *
Motion limits: Low: 0.000000 *    High: 0.000000 *
User Scale: 1.000000
User Zero: 0.000000
User Offset: 0.000000
User Rate Conv.: 1.000000
User output constant: 1.000000
Programmed velocity: 10000.000000 *
Programmed acceleration: 10000.000000 *
Programmed deceleration: 10000.000000 *
Minimum velocity: 0.000000 *
Current velocity: 0.000000 *
Velocity override: 1.000000
Module ADC Input 1: 0.0    Input2: 0.000000
```

* Value reported in user defined units

TT Tell Target

MCCL command: *a*TT*p* a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility: MC300, MC302, MC320, MC360, MC362
see also:
Reports target position. This is the absolute position to which the servo or stepper motor was last commanded to move. It may be specified directly with the Move Absolute (MA) command or indirectly with the Move Relative (MR) command. The interpretation of parameter *p* is explained at the beginning of this section.

TV Tell Velocity

MCCL command : aTVp a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility: MC300, MC302, MC320, MC360, MC362
see also: HS, LS, MS
Reports the current velocity of a servo or stepper motor. The value is reported in units of encoder counts per servo loop update.

TX Tell contouring count

MCCL command: aTX *a* = Axis number
compatibility: MC300, MC302, MC320, MC360, MC362
see also: CP
Reports the current contour path motion that an axis is performing. The value that the DCX replies is only valid for the controlling axis in a group of axes performing contoured path motion. After the Contour Mode command is issued to an axis, the TX command will have a reply value of 0. For each Linear or User Defined Contour Path motion that the controller completes, the contouring count will be incremented by one. For Arc Contour Path motions, the count will be incremented by 2. By counting the number of Contour Path commands that have been issued to the controller (1 for linear, 2 for arc), and comparing it to the response from the TX command, the user can determine on what segment of a continuous path motion the motors are on. The contour count is stored as a 32 bit value (2,147,483,647). To reset the contour count value and avoid ‘wrap around’, the user should stop motion and issue the Contour Mode command.

TZ Tell index position

MCCL command: *a*TZ*p* a = Axis number *p* = 0, .1, .2, .3, .4, .5
compatibility: MC300, MC302, MC320
see also:
Reports the position where the index pulse was observed. This position is relative to the encoder's position when the controller was reset or a Define Home command was issued to the axis.

VE tell firmware **VE**rsion

MCCL command: VE
compatibility: N/A
see also:
Reports the revision level of the firmware running on the DCX. This command also displays the amount of memory installed on the DCX motion controller motherboard.

example: VE

```
DCX returns:
DCX-PCI300 Motion Controller
Hardware: 4096K Private RAM, 512K Flash Memory
System Firmware Ver. PM1 Rev. 1.0a
Copyright (c) 1994-2001 Precision MicroControl Corporation
All rights reserved.
```

Chapter Contents

Chapter 11

I/O Commands

CF Channel oFf

MCCL command: CF*x* *x* = Channel number
compatibility: MC400
see also: CN

Causes channel x to go to "off" state. If the channel has been configured for "high true", the channel will be at a logic low (less that 0.4 volts DC) after this command is executed. If it has been configured for "low true", the channel will be at a logic high (greater than 2.4 volts DC).

CH Channel High

MCCL command : CH*x* *x* = Channel number or 0
compatibility: MC400
see also: CL

Causes digital I/O channel x to be configured for "high true" logic. This means that the I/O channel will be at a high logic level (greater than 2.4 volts DC) when the channel is "on". and at a low logic level (less than 0.4 volts DC) when the channel is "off". Note that issuing this command will not cause the I/O channel to change its current state. Issuing this command without specifying a channel will cause all channels present on the DCX to be configured as "high true". If parameter *x* = 0 all digital I/O channels will be configured for high true logic.

CI Channel In

MCCL command: CI*x* *x* = Channel number
compatibility: MC400
see also: CT

Used to configure digital I/O channel x as an input. All digital I/O channels on the DCX default to inputs on power-on or reset. If they are subsequently changed to outputs with the Channel ouT command, they can be returned to inputs with the Channel In command. The state of a digital I/O channel can be viewed with the Tell Channel command.

CL Channel Low

MCCL command : CL*x* *x* = Channel number or 0
compatibility: MC400
see also: CH

Causes digital I/O channel x to be configured for "low true" logic. This means that the I/O channel will be at a low logic level (less than 0.4 volts DC) when the channel is "on", and at high logic level (greater than 2.4 volts DC) when the channel is "off". Note that issuing this command will not cause the I/O channel to change its current state. Issuing this command without specifying a channel will cause all channels present on the DCX to be configured as "low true". If parameter *x* = 0 all digital I/O channels will be configured for low true logic.

CN Channel oN

MCCL command: CN*x* *x* = Channel number
compatibility: MC400
see also: CF

Causes channel x to go to "on" state. If the channel has been configured for "high true", the channel will be at a logic high (greater than 2.4 volts DC) after this command is executed. If it has been configured for "low true", the channel will be at a logic low (less that 0.4 volts DC).

CT Channel ouT

MCCL command: CT*x* *x* = Channel number
compatibility: MC400
see also: CI

Used to configure digital I/O channel x as an output. The DCX will turn the channel "off" before changing it to an output.

DF Do if channel oFf

MCCL command: DF*x* x = Channel number

Used for conditional execution of commands. If digital I/O channel *x* is "off", commands that follow on the command line or in the macro will be executed. Otherwise the rest of the command line or macro will be skipped. See the description of **Digital I/O** in the **DCX General Purpose I/O** chapter.

```
DF2,1MR1000                         ;If channel 2 is off move 1000
```

DN Do if channel ‘x’ is oN

MCCL command: DN*x* x = Channel number

Used for conditional execution of commands. If digital I/O channel *x* is "on", commands that follow on the command line or in the macro will be executed. Otherwise the rest of the command line or macro will be skipped. See the description of **Digital I/O** in the **DCX General Purpose I/O** chapter.

```
DN2,1MR1000                         ;If channel 2 is off move 1000
```

GA Get Analog

MCCL command: GA*x* x = Channel number
compatibility: MC500, MC520
Performs analog to digital conversion on the specified input channel and places the result into the Accumulator (User Register 0). Analog channels are numbered starting with 1.

IF If channel oFf do next command, else skip 2 commands

MCCL command: IF*x* x = Channel number
Used for conditional execution of commands. If digital I/O channel *x* is "off", command execution will continue with the command following the IF command. Otherwise the two commands following the IF command will be skipped, and command execution will continue from the third command. See the description of **Digital I/O** in the **DCX General Purpose I/O** chapter.

```
        IF5,MJ10,NO,MJ11                              ;If digital input #5 is off jump to
                                                      ;macro 10, otherwise jump to macro 11
```

IN If channel ' oN do next command, else skip 2 commands

MCCL command: IN*x* x = Channel number
Used for conditional execution of commands. If digital I/O channel *x* is "on", command execution will continue with the command following the IN command. Otherwise the two commands following the IN command will be skipped, and command execution will continue from the third command. See the description of **Digital I/O** in the **DCX General Purpose I/O** chapter.

```
        IN5,MJ10,NO,MJ11                              ;If digital input #5 is on jump to
                                                      ;macro 10, otherwise jump to macro 11
```

OA Output Analog

MCCL command: OA*n* *n* = integer or real
compatibility MC500, MC520
Sets the specified analog output channel to the value stored in the Accumulator (User Register 0). The analog output channels on any installed MC500 modules are numbered consecutively starting with channel 1. The contents of the Accumulator should be in the range 0 to 4095.

RC Reset Counter

MCCL command: RC
compatibility: N/A
see also:

Not supported

TA Tell Analog

MCCL command:	TAx *x* = Channel number *p* = 0, 1, 2, 3, ... (# of MC500/510 modules X 4)
compatibility:	MC500, MC510
see also:	

Reports the digitized analog input signals to MC500 and MC510 modules. The analog input channels on any installed MC500/510 modules will be numbered sequentially starting with channel 1. For each of these channels, the TA command will display a number between 0 and 4096. These numbers are the ratio of the analog input voltage to the reference input voltage multiplied by 4096. See the description of **Analog Inputs** in the **DCX General Purpose I/O** chapter.

TC Tell Channel

MCCL command:	TC*x* *x* = Channel number or 0
compatibility:	MC400
see also:	

Reports the on/off status of each digital I/O line. This data is reported separately for each channel. The DCX responds by displaying the channel number and a "1" if the channel is "on", or a "0" if the channel is "off". If parameter *x* = 0 the state of all digital I/O channels will be reported.

Chapter Contents

Chapter 12

Macro and Multi-tasking Commands

BK BreaK

MCCL command: BK
see also: GT, TR

Execution of this command will cause the rest of the command line or macro to be skipped. This command is used in conjunction with the If oN and If ofF commands to implement conditional execution.

To enable single stepping of a MCCL program use 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.

ET Escape Task

MCCL command: ET*n* *n* = integer > = 0
see also: GT, TR

This command is used to terminate a 'background task' that was created with the Generate Task command. The parameter to this command must be the task identifier that was placed in the accumulator (user register 0) of the task that issued the Generate Task command. A background task can use this command to terminate itself, but it must first acquire its identifier from the 'parent' task through a global register. Note that the task that interprets and executes commands received from the command interfaces cannot be terminated. See the description of **Multi-Tasking** in the **DCX Operation** chapter.

GT Generate Task

MCCL command: GT*n* *n* = integer > = 0, < = 1000
see also: ET, MC, MD, TR

This command will cause macro *n* to be executed as a background task. Alternatively, this command can precede a sequence of commands. In this case, the commands following the Generate Task command will be executed as a background task. After this command is issued, an identifier for the background task will be placed in the accumulator (register 0) of the task that issued the command. This identifier can be used as the parameter to the Escape Task command to terminate the background task. See the description of **Multi-Tasking** in the **DCX Operation** chapter.

MC Macro Call

MCCL command: MC*n* *n* = integer > = 0, < = 1000
see also: ET, MD

This command may be used to execute a previously defined macro command. If there is no macro defined by the number n, an error message will be displayed. Macro Call Commands can also be used in compound commands with other commands in the instruction set. In addition, a macro command can call another macro command, which in turn can call another macro command, and so on. See the description of **Macro Command** in the **DCX Operation** chapter.

MD Macro Define

MCCL command: MD*n* *n* = integer > = 0, < = 1000
see also: ET, GT, MD, RM

Used to define a new macro. This is done by placing the Macro Define command as the first command in a sequence of commands. All commands following the Macro Define command will be included in the macro. See the description of **Macro Command** in the **DCX Operation** chapter.

Macros will erased if power to the board is turned off. A macro can be redefined but the memory space occupied by the previous version of the macro will not be reused until a Reset Macro command is issued. Thus, if macro *n* already exists when a Macro Define command for that macro is issued, the previously defined macro will be replaced by the new macro definition.

MJ Macro Jump

MCCL command: MJ*n* *n* = integer > = 0, < = 1000
see also: ET, GT, MD

Jumps to a previously defined macro. This command differs from the Macro Call command in that execution will not return to the command following the MJ command. See the description of **Macro Command** in the **DCX Operation** chapter.

NO No Operation

MCCL command: NO

This command does nothing. It can be used to cause short delays in command line executions or as a filler in sequence commands.

TM Tell Macros

MCCL command: TM*n* *n* = integer >= -1, <= 1000
see also: MD, RM

Displays the commands which make up any macros which have been defined. If n = -1, all macros will be displayed. Since macros may be defined in any sequence, the TM command is useful for confirming the existence and/or contents of macro commands. In addition to the contents of macros, this command will also show the amount of memory available for macro storage, both in RAM and FLASH memory. See the description of **Macro Command** in the **DCX Operation** chapter.

Reset Macros

MCCL command: RM

This command will initialize the memory space used for storage of macro commands. It has the effect of erasing currently defined macros from memory. It is also the only way in which macro commands can be removed from memory after they are defined. It is always a good idea to use the Reset Macro command (RM) before setting up a new set of macro commands. See the description of **Macro Commands** in the **Working with MCCL Commands** chapter.

Tell Macros

MCCL command: TM*n* *n* = integer > = -1, < = 1000

Displays the commands which make up any macros which have been defined. If *n* = -1, all macros will be displayed. Since macros may be defined in any sequence, the TM command is useful for confirming the existence and/or contents of macro commands. See the description of **Macro Commands** in the **Working with MCCL Commands** chapter.

Chapter Contents

Chapter
13

Register Commands

AA Accumulator Add

MCCL command: AAn *n* = integer or real

Performs ACC = ACC + *n*, the addition of the command parameter ***n*** to the Accumulator (User Register 0). If the command parameter is in integer format, the result is stored in the Accumulator as a 32 bit integer. If the command parameter is in real format, the result is stored in the Accumulator (User Register 0 and 1) as a 64 bit real value.

AC Accumulator Complement, bit wise

MCCL command: AC

Performs ACC = !ACC, the bit wise logical complement of the Accumulator (User Register 0). The result is stored in the Accumulator as a 32 bit integer.

AD Accumulator Divide

MCCL command: ADn *n* = integer or real

Performs ACC = ACC/*n*, the division of the Accumulator (User Register 0) by the command parameter. If the command parameter is in integer format, the result is stored in the Accumulator as a 32 bit integer. If the command parameter is in real format, the result is stored in the Accumulator (User Register 0 and 1) as a 64 bit real value. No operation is done if the command parameter is zero.

AE Accumulator logical Exclusive or with ‘*n*’ , bit wise

MCCL command: AE*n* *n* = integer or real

Performs ACC = ACC ^ *n*, the bit wise logical exclusive or'ing of the Accumulator (User Register 0) with the command parameter. The result is stored in the Accumulator as a 32 bit integer.

AL Accumulator load

MCCL command: AL*n* *n* = integer or real

Loads the Accumulator (User Register 0) with *n* . If the command parameter is an integer (no decimal point or exponent label) the Accumulator will be marked as containing a 32 bit integer, otherwise it will be marked as containing a 64 bit real value.

```
AL1234567890                    ;Load 1234567890 into the accumulator
AL1234.56789                    ;Load 1234.56789 into the accumulator
AL0.123456789e4                 ;Load 1234.56789 into the accumulator
```

AM Accumulator Multiply

MCCL command: AM*n* *n* = integer or real

Performs ACC = ACC * *n*, the multiplication of the Accumulator (User Register 0) by the command parameter. If the command parameter is in integer format, the result is stored in the Accumulator as a 32 bit integer. If the command parameter is in real format, the result is stored in the Accumulator (User Register 0 and 1) as a 64 bit real value.

AN Accumulator logical 'aNd' the '*n*' , bit wise

MCCL command: AN*n* *n* = integer or real

Performs ACC = ACC & *n*, the bit wise logical AND of the Accumulator (User Register 0) with the command parameter. The result is stored in the Accumulator as a 32 bit integer.

AO Accumulator logical 'Or' with '*n*' , bit wise

MCCL command : AO*n* *n* = integer or real

Performs ACC = ACC | *n*, the bit wise logical OR of the Accumulator (User Register 0) with the command parameter. The result is stored in the Accumulator as a 32 bit integer.

AR copy Accumulator to Register

MCCL command: ARn *n* = integer or real

Copies the contents of the Accumulator (User Register 0) to the User Register specified by *n*. The contents of the Accumulator are unaffected by this command.

AS Accumulator Subtract

MCCL command: AS*n* *n* = integer or real

Performs ACC = ACC - *n*, the subtraction of the command parameter from the Accumulator (User Register 0). If the command parameter is in integer format, the result is stored in the Accumulator as a 32 bit integer. If the command parameter is in real format, the result is stored in the Accumulator (User Register 0 and 1) as a 64 bit real value.

AV Accumulator eValuate

MCCL command: AVn *n* = integer >= 0, <=25

Performs a unary operation on the contents of the Accumulator (User Register 0), placing the result in the Accumulator, overwriting the original contents. Parameter *n* specifies the desired operation. The table below list the available operations and the respective command parameter to use. The result that is stored in the Accumulator (1<= *n* <= 25) will be a 64 bit real in all cases except the Convert to ASCII operation which returns an integer.

Parameter n =	*Operation*	*Return type*
1	Convert to ASCII (Address placed in ACC)	Integer
2	Change Sign	Double
3	Absolute Value	Double
4	Ceiling	Double
5	Floor	Double
6	Fraction	Double
7	Round	Double
8	Square	Double
9	Square Root	Double
10	Sine	Double
11	Cosine	Double
12	Tangent	Double
13	Arc Sine	Double
14	Arc Cosine	Double
15	Arc Tangent	Double
16	Hyperbolic Sine	Double
17	Hyperbolic Cosine	Double
18	Hyperbolic Tangent	Double
19	Exponent	Double
20	Log	Double
21	Log10	Double
22	Load Pi	Double
23	Load 2 * Pi	Double
24	Load Pi/2	Double
25	Convert double register contents to an integer	Integer

AX get Auxiliary encoder indeX position

MCCL command: *a*AXn a = Axis number
compatibility: MC300, MC320, MC360
see also: GX

Loads the accumulator with the position of the auxiliary encoder when the index pulse was last captured (using the Auxiliary encoder Find command). The value is relative to the auxiliary encoder's position when the controller was reset or an Auxiliary encoder Home command was last issued to the axis.

GA Get Analog

MCCL command: GA*x* x = Channel number

Performs analog to digital conversion on the specified input channel and places the result into the Accumulator (User Register 0). Analog channels are numbered starting with 1.

GD Get the module iD

MCCL command: *a*GD*x* *a* = integer > 0, <= 8

Loads the accumulator with the type of motor module associated with an axis number

Module Type	ID code
MC300	2
MC302	22
MC320	162
MC360	3
MC362	23
none	15

GF Get IIR Filter coefficients

MCCL command : *a*GF*n* *a* = Axis number n = integer (coefficient #)
compatibility: MC300, MC320
see also: FL, NF, YF, ZF

Loads IIR filter coefficient number *n* into the accumulator. See the description of **User Defined Filters (Notch and Low Pass)** in the **Application Solutions chapter**.

GU Get defaUlt axis

MCCL command: GU

The DCX-PCI300 defaults to setting the default axis to zero. If the user executes a motion or setup command with the axis specifier missing, the default axis will be used. In most cases a motion or setup command issued to axis zero commands that operation to all axes. By defining a non-zero default axis, the user can execute ‘generic’ macro’s (no axis number specified) to any axis.
This Get default axis is used to report the current default axis by placing the current setting into the accumulator. The default axis is defined by using the setup command set the defa**U**lt **A**xis (**UA*n***).

GX Get the position of the auxiliary encoder

MCCL command: *a*GX *a* = Axis number
compatibility: MC300, MC320, MC360
see also: AT, AX

This command reads the auxiliary encoder associated with axis a and places the value into the Accumulator (User Register 0).

LU Look Up variable

MCCL command: LU*s* s= *string parameter ("variable name")*

Loads the accumulator with the memory location for a motor table data entry. For additional information including a complete listing of variable names please refer to the description of **Reading Data from DCX Memory** in **Chapter 6** of this manual.

OA Output Analog

MCCL command: OAx x = integer or real

compatibility MC500, MC520

Sets the analog output of channel *x* to the value stored in the Accumulator (User Register 0). The analog output channels on any installed MC500 modules are numbered consecutively starting with channel 1. The contents of the Accumulator should be in the range 0 to 4095.

RA copy Register to Accumulator

MCCL command: RAn *n* = integer or real

Copies the contents of the User Register *n* into the Accumulator (User Register 0). The original contents of the accumulator is overwritten, while the contents of the source User Register are unaffected.

RB Read the Byte at absolute memory location '*n*' into the accumulator

MCCL command: *a*RB*n* *a* = Axis number *n* = integer

This command will copy the contents of the byte located at absolute memory address *n* into the Accumulator (User Register 0). Alternatively, if an axis number is specified with the command, the contents of a byte located within that axes' motor table will be copied into the accumulator. In this case the command parameter specifies the offset of the byte from the beginning of that axes motor table. The **Reading DCX Memory** section of this chapter lists the offsets of all data in the motor tables. The upper bits of the Accumulator are cleared when the byte data is copied into it.

RD Read the Double (64 bit real) value at absolute memory location '*n*' into the accumulator

MCCL command: *a*RD*n* *a* = Axis number *n* = real

This command will copy the contents of the Double (64 bit real) located at absolute memory address *n* into the Accumulator (User Register 0). Alternatively, if an axis number is specified with the command, the contents of a Double located within that axes' motor table will be copied into the accumulator. In this case the command parameter specifies the offset of the Double from the beginning of that axes motor table. The **Reading DCX Memory** section of this chapter lists the offsets of all data in the motor tables.

RL Read the Long (32 bit integer) value at absolute memory location '*n*' into the accumulator

MCCL command: *a*RL*n* *a* = Axis number *n* = integer

This command will copy the contents of the Long (32 bit integer) located at absolute memory address n into the Accumulator (User Register 0). Alternatively, if an axis number is specified with the command, the contents of a Long located within that axes' motor table will be copied into the accumulator. In this case the command parameter specifies the offset of the Long from the beginning of that axes motor table. The **Reading DCX Memory** section of this chapter lists the offsets of all data in the motor tables.

RV Read the float (32 bit real) value at absolute memory location '*n*' into the accumulator

MCCL command: *a*RV*n* *a* = Axis number *n* = real

This command will copy the contents of the Float (32 bit real) located at absolute memory address *n* into the Accumulator (User Register 0). Alternatively, if an axis number is specified with the command, the contents of a Float located within that axes' motor table will be copied into the accumulator. In this case the command parameter specifies the offset of the Float from the beginning of that axes motor table. The **Reading DCX Memory** section of this chapter lists the offsets of all data in the motor tables.

RW Read the Word (16 bit integer) value at absolute memory location '*n*' into the accumulator

MCCL command: *a*RW*n* *a* = Axis number *n* = integer

This command will copy the contents of the Word (16 bit integer) located at absolute memory address *n* into the Accumulator (User Register 0). Alternatively, if an axis number is specified with the command, the contents of a Word located within that axes' motor table will be copied into the accumulator. In this case the command parameter specifies the offset of the Word from the beginning of that axes motor table. The **Reading DCX Memory** section of this chapter lists the offsets of all data in the motor tables.

SL Shift Left accumulator by '*n*' bits

MCCL command: SL*n* *n* = integer > 0, < = 31

Performs ACC = ACC << *n*, the logical shift of the Accumulator (User Register 0) to the left. The command parameter specifies the number of bits to shift the accumulator. Zero bits will be shifted in on the right. The result is stored in the Accumulator as a 32 bit integer.

SR Shift Right accumulator by '*n*' bits

MCCL command: SR*n* *n* = integer > 0, < = 31

Performs ACC = ACC >> *n* , the logical shift of the Accumulator (User Register 0) to the right. The command parameter specifies the number of bits to shift the accumulator. Zero bits will be shifted in on the left. The result is stored in the Accumulator as a 32 bit integer.

TR Tell Register '*n*'

MCCL command: TR*n*   *n* = integer >= 0, <= 256
compatibility: N/A
see also: AL, AR

Displays the contents of User Register *n*. When the command parameter is set to 0 (or not specified), this command reports the contents of User Register zero, which is the accumulator.

WB Write the low Byte in the accumulator to absolute memory location '*n*'

Not supported

MCCL command: WB*n*   *n* = integer

This command will copy the low byte of the accumulator (User Register 0) to the byte located at absolute memory address *n*.

WD Write the Double (64 bit real) value in the accumulator to absolute memory location '*n*'

Not supported

MCCL command: WD*n*   *n* = real

This command will copy a Double (64 bit real) from the accumulator (User Register 0 and 1) to absolute memory address *n*.

WL Write the Long (32 bit integer) value in the accumulator to absolute memory location '*n*'

Not supported

MCCL command: WL*n*   *n* = integer

This command will copy a Long (32 bit integer) from the accumulator (User Register 0) to absolute memory address *n*.

WV Write the float (32 bit real) value in the accumulator to absolute memory location '*n*'

Not supported

MCCL command: WV*n*   *n* = real

This command will copy a float (32 bit real) from the accumulator (User Register 0) to absolute memory address *n*.

WW Write the low Word (16 bit integer) value in the accumulator to absolute memory location '*n*'

Not supported

MCCL command: WW*n*   *n* = integer

This command will copy the low Word (16 bit integer) of the accumulator (User Register 0) to absolute memory address *n*.

Chapter Contents

Chapter 14

Sequence (If/Then) Commands

DF Do if channel oFf

MCCL command: DF*x* x = Channel number

Used for conditional execution of commands. If digital I/O channel *x* is "off", commands that follow on the command line or in the macro will be executed. Otherwise the rest of the command line or macro will be skipped. See the description of **Digital I/O** in the **DCX General Purpose I/O** chapter.

```
        DF2,1MR1000                         ;If channel 2 is off move 1000
```

DN Do if channel ‘x’ is oN

MCCL command: DN*x* x = Channel number

Used for conditional execution of commands. If digital I/O channel *x* is "on", commands that follow on the command line or in the macro will be executed. Otherwise the rest of the command line or macro will be skipped. See the description of **Digital I/O** in the **DCX General Purpose I/O** chapter.

```
        DN2,1MR1000                         ;If channel 2 is off move 1000
```

IB If the accumulator is Below ‘n’, execute the next command, else skip 2 commands

MCCL command: IBn *n* = integer or real

Used for conditional execution of commands. If the contents of the accumulator (User Register 0) is less than *n*, command execution will continue with the command following the IB command. Otherwise the two commands following the IB command will be skipped, and command execution will continue from the third command. See the description of **Digital I/O** in the **DCX General Purpose I/O** chapter.

```
        IB0,MJ10,NO,MJ11                    ;If the accumulator contents is less
                                            ;than 10 jump to macro 10, otherwise
                                            ;jump to macro 11
```

IC — **I**f bit 'n' of the accumulator is **C**lear (equal to 0), execute the next command, else skip 2 commands

MCCL command: IC*n* *n* = integer >= 0, <= 31

Used for conditional execution of commands. If the contents of the accumulator (User Register 0) has bit *n* reset, command execution will continue with the command following the IC command. Otherwise the two commands following the IC command will be skipped, and command execution will continue from the third command.

```
        IC3,MJ10,NO,MJ11                         ;If accumulator bit 3 is cleared jump
                                                 ;to macro 10, otherwise jump to macro
                                                 ;11
```

IE — **I**f the accumulator **E**quals "n", execute the next command, else skip 2 commands

MCCL command: IE*n* *n* = integer or real

Used for conditional execution of commands. If the contents of the accumulator (User Register 0) equals *n*, command execution will continue with the command following the IE command. Otherwise the two commands following the IE command will be skipped, and command execution will continue from the third command.

```
        IE0,MJ10,NO,MJ11                            ;If accumulator contents equals 0 jump
                                                    ;to macro 10, otherwise jump to macro
                                                    ;11
```

IF — **I**f channel o**F**f do next command, else skip 2 commands

MCCL command: IF*x* x = Channel number

Used for conditional execution of commands. If digital I/O channel *x* is "off", command execution will continue with the command following the IF command. Otherwise the two commands following the IF command will be skipped, and command execution will continue from the third command. See the description of **Digital I/O** in the **DCX General Purpose I/O** chapter.

```
        IF5,MJ10,NO,MJ11                            ;If digital input #5 is off jump to
                                                    ;macro 10, otherwise jump to macro 11
```

IG — **I**f the accumulator is **G**reater than 'n' execute the next command, else skip 2 commands

MCCL command: IG*n* *n* = integer or real

Used for conditional execution of commands. If the contents of the accumulator (User Register 0) is greater than *n*, command execution will continue with the command following the IG command. Otherwise the two commands following the IG command will be skipped, and command execution will continue from the third command. See the description of **Digital I/O** in the **DCX General Purpose I/O** chapter.

```
        IG0,MJ10,NO,MJ11                                ;If the accumulator contents is
                                                        ;greater than 0 jump to macro 10,
                                                        ;otherwise jump to macro 11
```

IN — **I**f channel ' o**N** do next command, else skip 2 commands

MCCL command: IN*x* x = Channel number

Used for conditional execution of commands. If digital I/O channel *x* is "on", command execution will continue with the command following the IN command. Otherwise the two commands following the IN command will be skipped, and command execution will continue from the third command. See the description of **Digital I/O** in the **DCX General Purpose I/O** chapter.

```
        IN5,MJ10,NO,MJ11                                ;If digital input #5 is on jump to
                                                        ;macro 10, otherwise jump to macro 11
```

IP — **I**nterrupt (set breakpoint reached flag) on absolute **P**osition

MCCL command: IP*n* *n* = integer or real

compatibility: MC300, MC302, MC320, MC360, MC362

This command is used to indicate when an axis has reached a specific position. The position is specified by parameter *n* as a relative distance from the axis home position. When the specified position has been reached, the DCX will set the "breakpoint reached" flag in the motor status for that axis. The IP command can be issued to an axis before or after it has been commanded to move.

For MC360 stepper axes the setting of the Breakpoint Reached Flag is based on the number of step pulses issued to the stepper driver, not the number of counts from the auxiliary encoder. The Interrupt on absolute Position command is not supported by MC360 modules configured as a closed loop stepper.

IR — **I**nterrupt (set breakpoint reached flag) upon reaching **R**elative position

MCCL command: IR*n* *n* = integer or real

compatibility: MC300, MC302, MC320, MC360, MC362

This command is used to indicate when an axis has reached a specific position. The position is specified by parameter *n* as a relative distance from the target position established by the last motion command. When the specified position has been reached, the DCX will set the "breakpoint reached" flag in the status for that axis. The IR command can be issued to an axis before or after it has been commanded to move.

For MC360 stepper axes the setting of the Breakpoint Reached Flag is based on the number of step pulses issued to the stepper driver, not the number of counts from the auxiliary encoder. The Interrupt on Relative Position command is not supported by MC360 modules configured as a closed loop stepper.

IS — **I**f bit 'n' of the accumulator is **S**et execute the next command, else skip 2 commands

MCCL command: IS*n* *n* = integer >= 0, <= 31

Used for conditional execution of commands. If the contents of the accumulator (User Register 0) has bit *n* set, command execution will continue with the command following the IS command. Otherwise the two commands following the IS command will be skipped, and command execution will continue from the third command.

```
        IS3,MJ10,NO,MJ11                          ;If accumulator bit 3 is set jump to
                                                  ;macro 10, otherwise jump to macro 11
```

IU — **I**f the accumulator is **U**nequal to "n" execute the next command, else skip 2 commands

MCCL command: IU*n* *n* = integer or real

Used for conditional execution of commands. If the contents of the accumulator (User Register 0) does not equal *n*, command execution will continue with the command following the IU command. Otherwise the two commands following the IU command will be skipped, and command execution will continue from the third command.

```
        IU0,MJ10,NO,MJ11                          ;If accumulator contents is unequal to
                                                  ;0 jump to macro 10, otherwise jump to
                                                  ;macro 11
```

JP — **J**um**P** to command absolute

MCCL command: JP*n* *n* = integer

Jumps to the specified command in the current command string or macro. Commands are numbered consecutively starting with 0.

```
        IE0,JP5,NO,1MR1000,1WS,1MR2000,1WS        ;If accumulator equals 0 jump to
                                                  ;1MR2000
```

JR — **J**ump to command **R**elative

MCCL command: JRn *n* = integer

Jumps forward or backward by *n* commands in the current command string or macro. Specifying a positive value will cause a forward jump in the command string or macro. Specifying a negative value will cause a backward jump. A jump of relative 0 will cause the command to jump to itself.

```
        1MR1000,1WS.005,IE0,JR-3                  ;If accumulator equals 0 jump to
                                                  ;1MR1000
```

RP — **R**e**P**eat

MCCL command: RPn *n* = integer > = 0, < = 2,147,483,647

This command causes all the commands preceding the RP command to be executed *n* + 1 times. If *n* is not specified or is 0 then the commands are repeated indefinitely. Note - There can be only one RP command in a command string or macro.

```
        TP,RP999                                  ;Display the position of axis #1, 1000
                                                  ;times
```

WA WAit

MCCL command: WAn *n* = integer or real >= 0

Insert a wait period of *n* seconds before going on to the next command. If this command was issued from an ASCII interface, it can be aborted by sending an Escape character.

```
1TP,WA0.1,RP9                          ;Display the position of axis #1, 10
                                       ;times with a delay of one tenth of a
                                       ;second between displays
```

WE Wait for Edge

MCCL command: *a*WE *a* = Axis number
compatibility: MC360, MC362
see also: EL, FE

This command is used in conjunction with the Edge Latch command to home an open loop stepper. When combined, the EL and WE commands perform the same operation as Find Edge (FE) without the negative side effect of possibly stalling the command interpreter. The WE command should not be issued until:

1) The motor is moving towards the home sensor mark
2) The EL command has been issued
3) The home sensor has been captured (status bit 10 - Home found is set)

After the Edge Latch command is issued, when the home sensor activates, the step count position of the sensor will be captured and the Home found status bit (status bit 10) will be set. The Wait for Edge and Motor oN commands are then issued to define the location of the home sensor as position *n*. See the description of **Homing Axes** in the **Motion Control** chapter.

WF Wait for digital channel oFf

MCCL command: WF*x* x = Channel number
compatibility: MC400
see also: WN

Wait until digital I/O channel x is "off" before continuing to the next command on the command line or in the macro. If this command was issued from an ASCII interface, it can be aborted by sending an Escape character.

WI Wait for encoder Index mark

MCCL command: *a*WIn *a* = Axis number *n* = integer or real >= 0
compatibility: MC300, MC302, MC320, MC360
see also: FI, IA

This command is used in conjunction with the Index Arm (IA) command to home a closed loop axis. When combined, the IA and WI commands perform the same operation as Find Index (FI) without the negative side effect of possibly stalling the command interpreter. The WI command should not be issued until:

1) The motor is moving towards the index mark

2) The IA command has been issued
3) The Index mark has been captured (status bit 10 - Index found is set)

After the Index Arm command is issued, when the index pulse occurs, the encoder position of the index mark will be captured and the Encoder Index status bit (status bit 10) will be set. The Wait for Index and Motor oN commands are then issued to define the location of the index pulse as position *n*. See the description of **Homing Axes** in the **Motion Control** chapter.

WN Wait for digital channel oN

MCCL command: WN*x* x = Channel number
compatibility: MC400
see also: WF

Wait until digital I/O channel x is "on" before continuing to the next command on the command line or in the macro. If this command was issued from an ASCII interface, it can be aborted by sending an Escape character.

WP Wait for absolute Position

MCCL command: *a*WP*n* *n* = integer or real
compatibility: MC300, MC302, MC320, MC360, MC362
see also:

This command is used to delay command execution until axis *a* has reached a specific position. The position is specified by the command parameter as a relative distance from the home position of the axis. When the specified position has been reached, the DCX will set the "breakpoint reached" flag in the status for that axis, and then continue execution of commands following WP. The WP command will typically be issued to an axis after it has been commanded to move. If this command was issued from an ASCII interface, it can be aborted by sending an Escape character.

For MC360 stepper axes the setting of the Breakpoint Reached Flag is based on the number of step pulses issued to the stepper driver, not the number of counts from the auxiliary encoder. The Wait for absolute Position command is not supported by MC360 modules configured as a closed loop stepper.

WR Wait for Relative position

MCCL command: *a*WR*n* *n* = integer or real
compatibility: MC300, MC302, MC320, MC360, MC362
see also:

This command is used to delay command execution until axis *a* has reached a specific position. The position is specified by the command parameter as a relative distance from the target position established by the last motion command. When the specified position has been reached, the DCX will set the "breakpoint reached" flag in the status for that axis, and then continue execution of commands following WR. The WR command will typically be issued to an axis after it has been commanded to move. If this command was issued from an ASCII interface, it can be aborted by sending an Escape character.

For MC360 stepper axes the setting of the Breakpoint Reached Flag is based on the number of step pulses issued to the stepper driver, not the number of counts from the auxiliary encoder. The Wait for Relative position command is not supported by MC360 modules configured as a closed loop stepper.

WS Wait for Stop

MCCL command: *a*WS*n* *n* = integer or real
compatibility: MC300, MC302, MC320, MC360, MC362
see also: WA, WT

Will delay execution of the next command in the sequence until the Trajectory Complete status bit (bit 3) for axis *a* (or all axes if axis specifier a = 0) has been set. The command parameter *n* specifies an additional time period (in seconds) that the controller will wait before continuing execution of the commands following WS. This additional time (or dwell) period is typically used to allow the following error of a servo to be resolved by the PID filter.

The trajectory complete bit of a closed loop servo will be set when the calculated (Optimal) position = the Target position. The trajectory complete bit of a closed loop stepper will be set when the Encoder position = the Target position. The trajectory complete bit of an open loop stepper will be set when the Step count position = the Target position.

```
3MR1000,WS0.1,MR-1000                   ;Perform a forward then backward
                                        ;motion sequence
```

comment: If the WS command was not used in the above example, there would be no motion of the axis. The reason being that the target position would simply be changed twice. The computer would add 1000 counts to the target position then subtract the same amount. This would take place far quicker than the axis could begin moving.

WT Wait for Target

MCCL command: *a*WT*n* *n* = integer or real
compatibility: MC300, MC302, MC320
see also: DB, DT, WS

This command will delay additional command execution until servo axis *a* (or all servo axes if axis specifier a = 0) has settled within the user defined target range (At Target status bit, bit 2). The target range is defined by the DeadBand (aDBn) and Delay at Target (aDTn) commands. Parameter *n* specifies an additional time period (in seconds) that the controller will wait before continuing execution of the commands following WT. This additional time (or dwell) period is typically used to allow the for additional settling time of a servo. When controlling steppers (MC360 & MC362) use the Wait for Stop command

The conditions for a closed loop axis servo to have reached its' target, are that it remains within the position **D**ead**B**and range ***n*** for the time period specified by the **D**elay at **T**arget parameter '***n***'. The Wait for Target command should not be used for axes in contour mode.

```
2DB5,2DT0.01                            ;deadband range = +/- 5, at target
                                        ;timer = 0.01 seconds
3MR1000,WT0.1,MR-1000                   ;Perform a forward then backward
                                        ;motion sequence
```

comment: If the WT command was not used in the above example, there would be no motion of the axis. The reason being that the target position would simply be changed twice. The computer would add 1000 counts to the target position then subtract the same amount. This would take place far quicker than the axis could begin moving.

Chapter Contents

Chapter 15

Miscellaneous Commands

DM Decimal Mode

MCCL command: DM
see also: HM

Input and output numbers in decimal format.
comment: The Decimal Mode command must be "executed" by the DCX before commands can be issued with decimal formatted parameters. The Decimal Mode (DM) and Hexadecimal Mode (HM) commands cannot be in the same command string.

DW Disable Watchdog

MCCL command: DM
see also:

Disable the processor watchdog circuit.
comment: This command is reserved for factory use only.

EF Echo oFf

MCCL command: EFn *n* = 0, 1, 2, or 3
compatibility:
see also: EF

Causes the DCX not to echo characters received through an ASCII command port.
Only data specifically requested from the DCX will be transmitted. Normally used when operating with the host or terminal in half duplex mode. Parameter *n* selects the terminating character or characters that will be transmitted with command replies. The following table lists the available options.

Parameter n	*Terminating Characters*
0	No change from current setting
1	Carriage Return (ASCII 13) Only
2	Linefeed (ASCII 10) Only
3	Carriage Return and Linefeed (ASCII 13 and 10)

EN Echo oN

MCCL command: EN*n* *n* = 0, 1, 2, or 3
compatibility:
see also: EF

Causes all characters received through an ASCII command port to be echoed to that port as received. Normally used when operating with a or terminal in full duplex mode. Parameter *n* selects the terminating character or characters that will be transmitted with command replies. The following table lists the available options.

Parameter n	*Terminating Characters*
0	No change from current setting
1	Carriage Return (ASCII 13) Only
2	Linefeed (ASCII 10) Only
3	Carriage Return and Linefeed (ASCII 13 and 10)

FD Output text with Doubles

MCCL command: FDs s = *string parameter*
see also: FT, OD, OT

This command places a formatted message string and double precision values into DCX memory. Upon completion of this command the memory address where the formatted message is stored is available in the accumulator (register 0). For additional information please refer to the description of **Outputting Formatted Message Strings** in the **Working with MCCL Commands chapter**.

FT Output Text with integers

MCCL command: FDs s = *string parameter*
see also: FD, OD, OT

This command places a formatted message string and integer values into DCX memory. Upon completion of this command the memory address where the formatted message is stored is available in the accumulator (register 0). For additional information please refer to the description of **Outputting Formatted Message Strings** in the **Working with MCCL Commands chapter**.

HE display the supported MCCL commands

MCCL command: HE
explanation: Reports the valid DCX command mnemonics for the installed software version.

HM Hexadecimal Mode

MCCL command: HM
see also: DM

Input and output numbers in hexadecimal format.
comment: The Hexadecimal Mode command must be executed by the DCX before commands can be issued with hexadecimal formatted parameters. The Hexadecimal Mode (HM) and Decimal Mode (DM) and commands cannot be in the same command string. If a command parameter is to be entered in hexadecimal format, and the number starts with either A, B, C, D, E, or F, it must be preceded by a '0' (zero).

NO No Operation

MCCL command: NO

This command does nothing. It can be used to cause short delays in command line executions or as a filler in sequence commands.

OD Output text with Doubles

MCCL command: ODs s = *string parameter*
see also: FD, FT, OT

This command allows the user to send formatted message strings and double precision values to the ASCII interface (WinControl). For additional information please refer to the description of **Outputting Formatted Message Strings** in the **Working with MCCL Commands chapter**.

OT Output Text with integers

MCCL command: OTs s = *string parameter*
see also: FD, FT, OD

This command allows the user to send formatted message strings and integer values to the ASCII interface (WinControl). For additional information please refer to the description of **Outputting Formatted Message Strings** in the **Working with MCCL Commands chapter**.

PC Prompt Character

MCCL command: PCn *n* = integer >0, <= 255

This command sets the character that will be sent out an ASCII command port when the DCX completes execution of a command issued to that port. The parameter to this command is the ASCII code for the character. Issuing the command with a parameter of 0 will inhibit any character from being sent. The default prompt character is '>' (ASCII 62 decimal).

RT ReseT

MCCL command: *a*RT *a* = Axis number (0 resets the entire controller)
compatibility: MC300, MC302, MC320, MC360, MC362, MC400, MC5X0
see also: Default Settings in the Appendix

Performs a reset of the entire controller or a specific axis. If an axis number is specified when the command is issued, just that axis will be reset. If no axis is specified, the entire controller and all installed axes will be reset. When an axis is reset, the default conditions such as acceleration and velocity will be restored, and the axes will be placed in the "off" state. If the reset command is issued to a dual axis (MC302, MC362) both that axis and its companion axis will be reset.

Chapter Contents

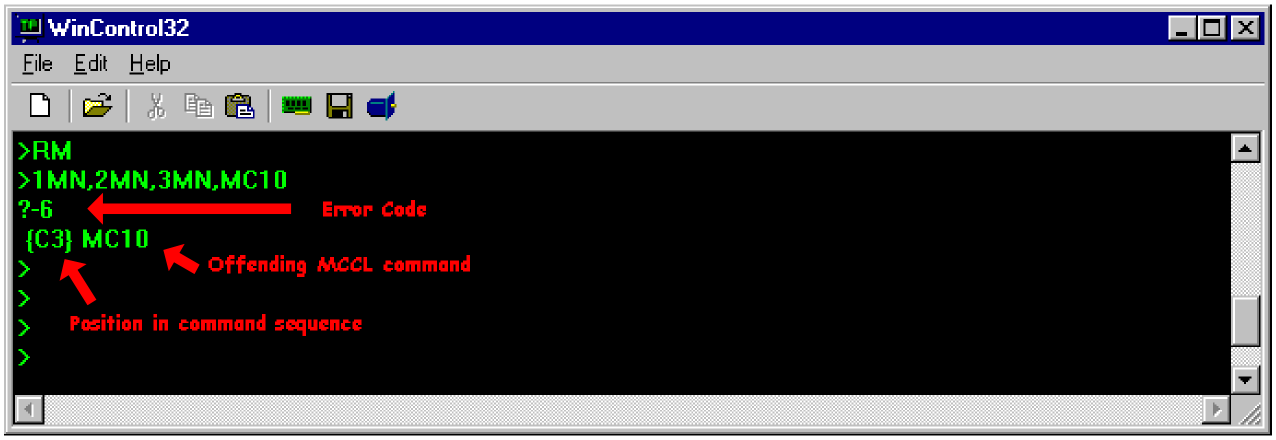
Chapter 16

Controller 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.



```
>RM
>1MN,2MN,3MN,MC10
?-6
 {C3} MC10
>
>
>
>
```

Chapter Contents

Chapter

17

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, Gain, and Joystick
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 Using External Reference		 +/- 3 +/- 1/2	 LSB LSB
Input Full-Scale Error Using Internal Reference Using External Reference		 +/- 15 +/- 1/2	 LSB LSB
Input Zero Temp. Coefficient		0.5	ppm/C
Input Differential Nonlinearity		+/- 1	LSB
Input Total Unadjusted Error Using Internal Reference Using External Reference		 +/- 15 +/- 1	
Input Voltage Range Using Internal Reference Using External Reference	 0.0 0.0	 5.0 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 -10.0	5.0 +10.0	V V

* These values are for 0 to +5.0 volt outputs

Chapter Contents

Chapter 18

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 CA-26IDS2-F-SPT 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**

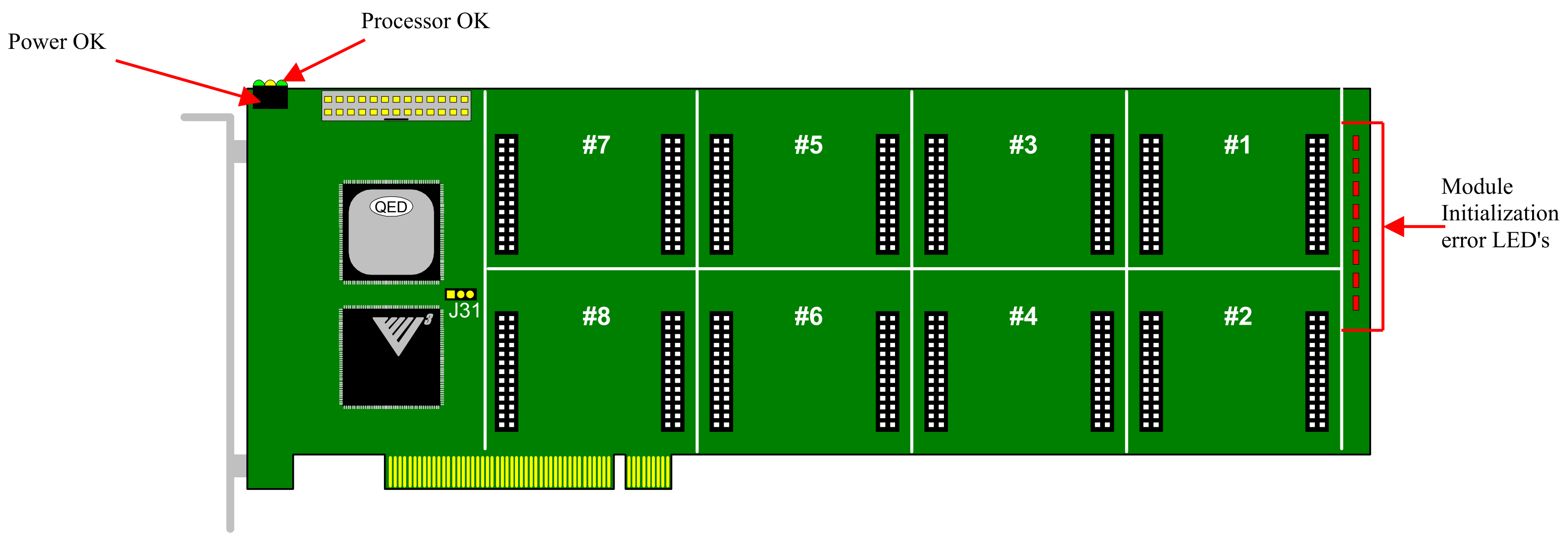
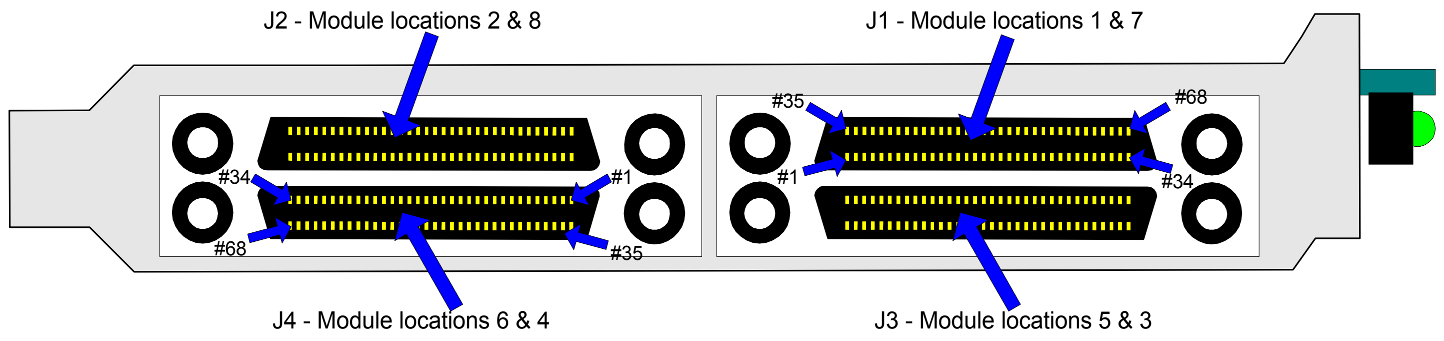


Figure 18: DCX-PCI300-R motherboard (ribbon cable version)


J2 - Module locations 2 & 8
J1 - Module locations 1 & 7
#35
#68
#34
#1
#1
#34
#68
#35
J4 - Module locations 6 & 4
J3 - Module locations 5 & 3

VHDCI connectors as viewed from the back of the computer
(component side down)

Figure 19: 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: 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 (**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 CA-26IDS2-F-SPT 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-

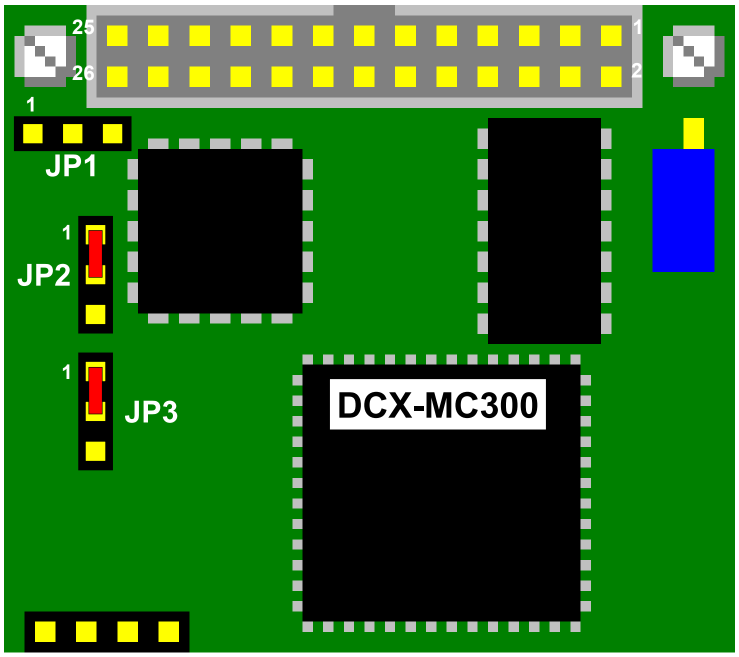
JP3 – Encoder Power Select (+5VDC or +12 VDC)

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.)

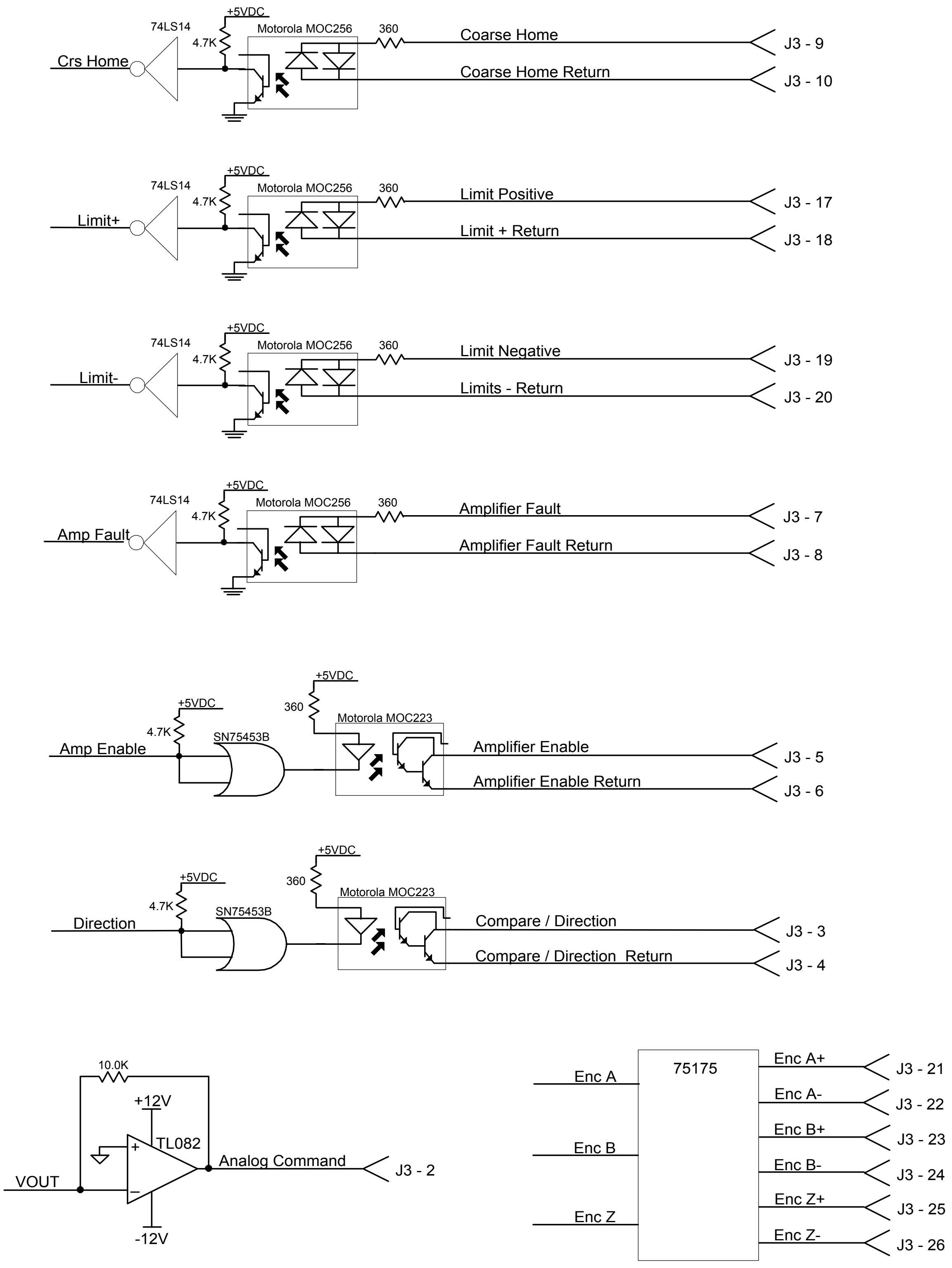
DCX-MC300 Module Output Offset Potentiometer

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.

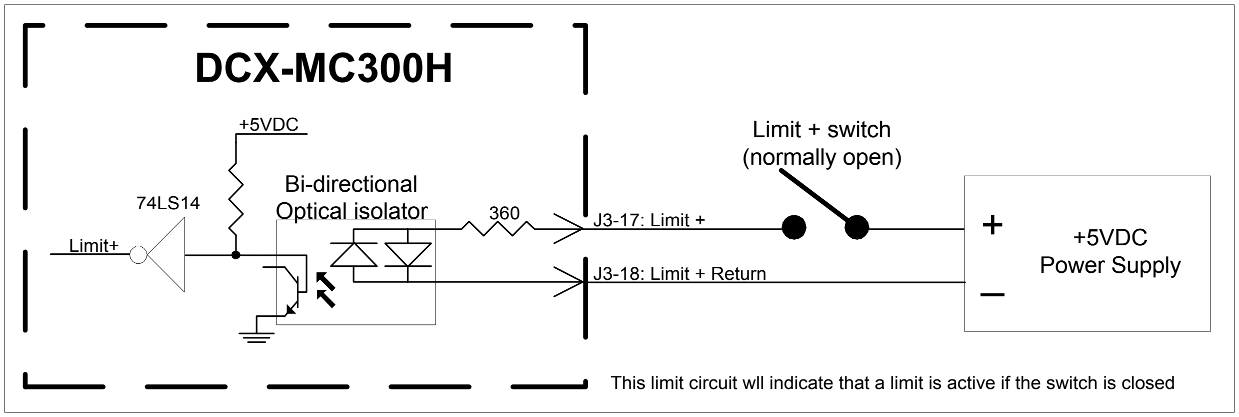
DCX-MC300 Module Layout



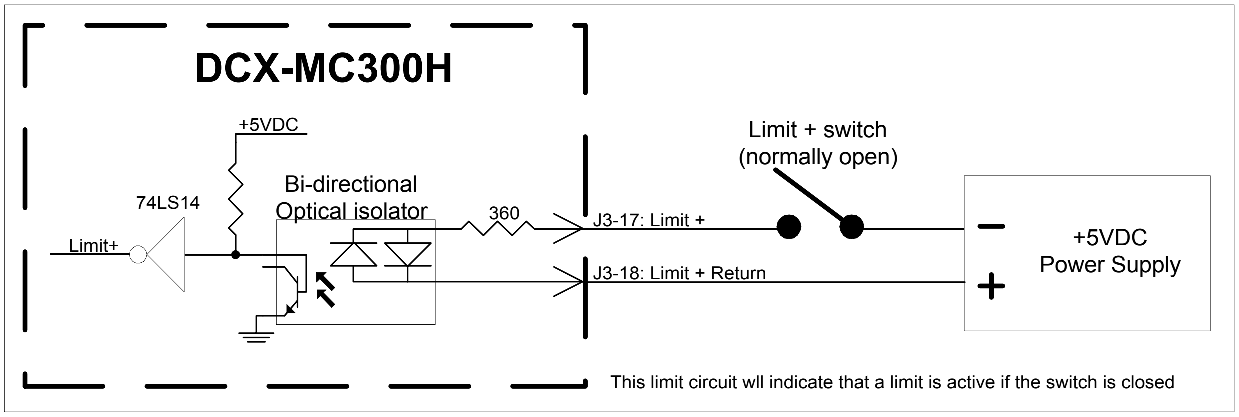
DCX-MC300H Axis I/O Interface Schematic



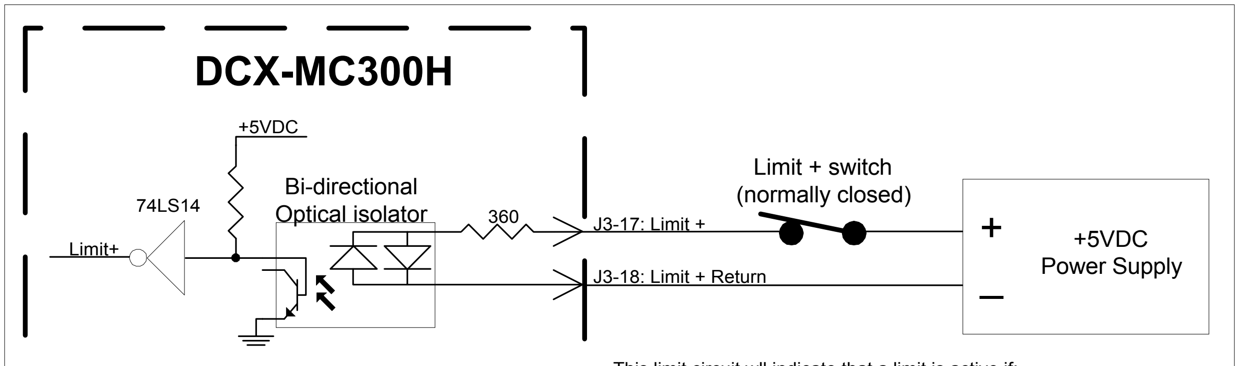
DCX-MC300H Optically Isolated Inputs Wiring Examples



This limit circuit wll indicate that a limit is active if the switch is closed



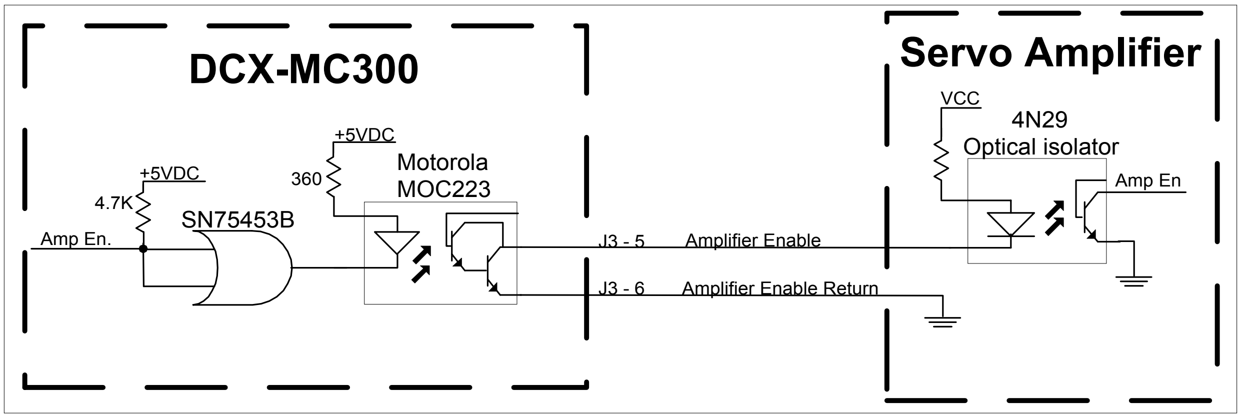
This limit circuit wll indicate that a limit is active if the switch is closed



This limit circuit wll indicate that a limit is active if:
1) The switch is open
2) Any component in the circuit fails (power supply, bi-directional opto isolator, broken wire, etc...

This is not the default configuration of the DCX, issue the **MC_LIMIT_INVERT** parameter of the ***MCSetLimits( )*** function.

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).

Primary 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

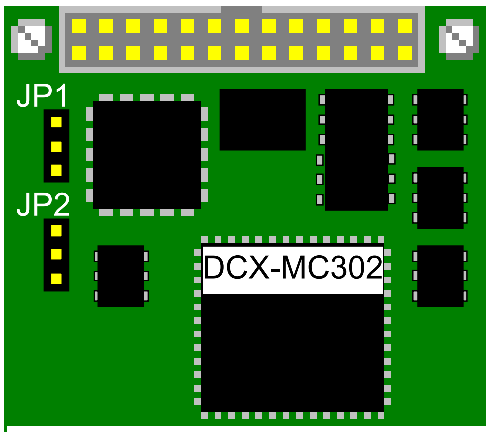
JP1 – Axis 1 Encoder Power Select (+5VDC or +12 VDC)

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.**)

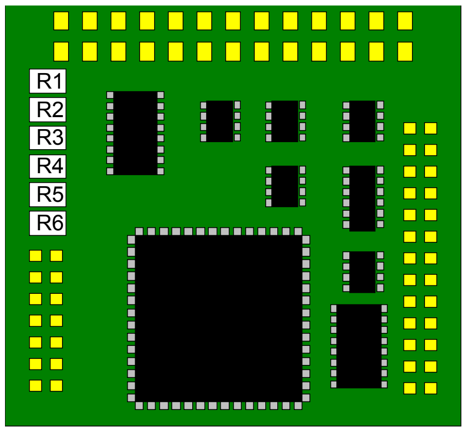
JP2 – Axis 2 Encoder Power Select (+5VDC or +12 VDC)

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

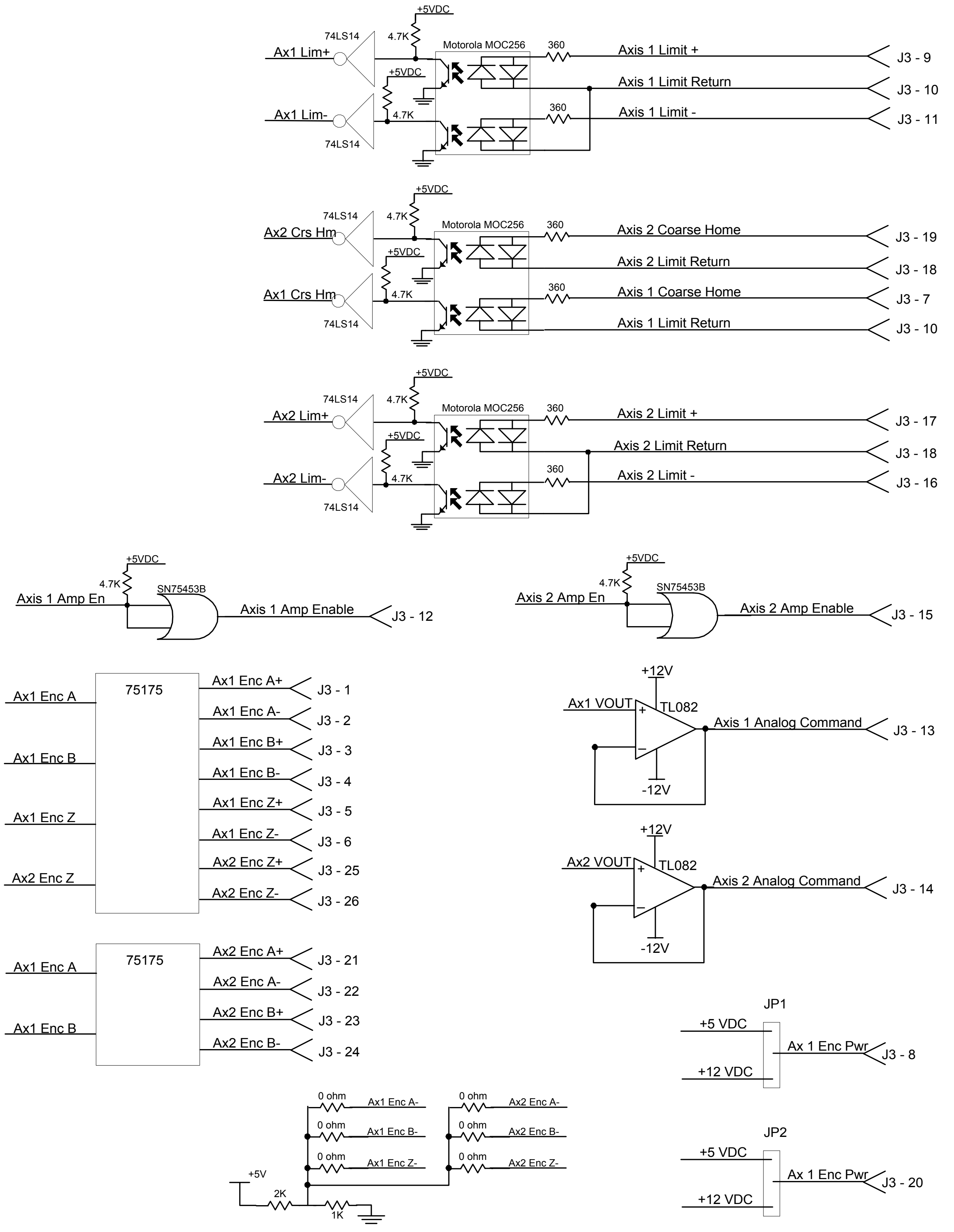


DCX-MC302 top side

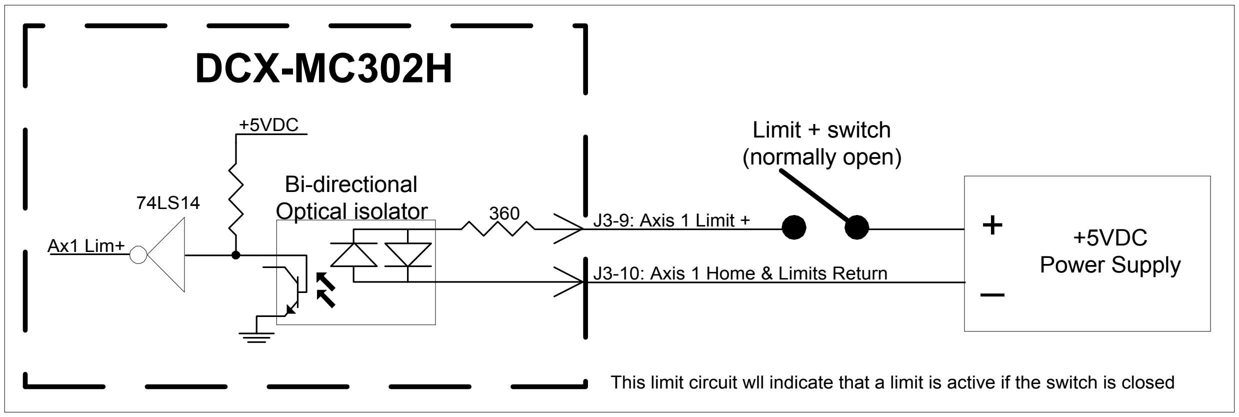


DCX-MC302 bottom side: remove R1 – R6 for differential encoder

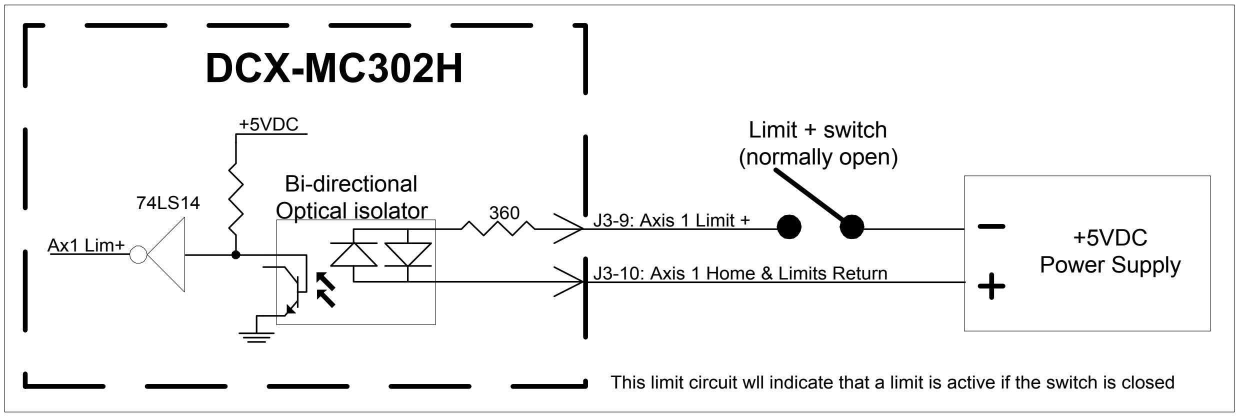
DCX-MC302H Axis I/O Interface Schematic



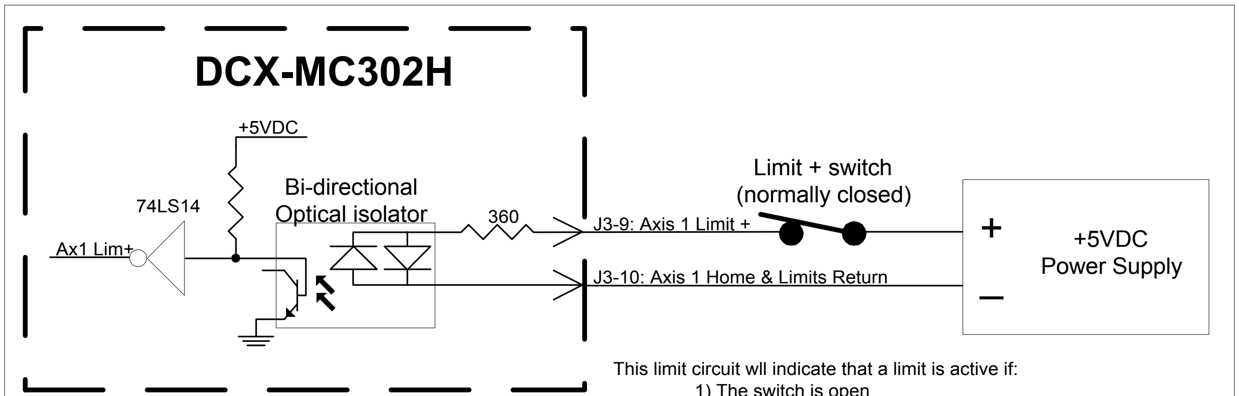
DCX-MC302 Optically Isolated Inputs Wiring Examples



This limit circuit wll indicate that a limit is active if the switch is closed



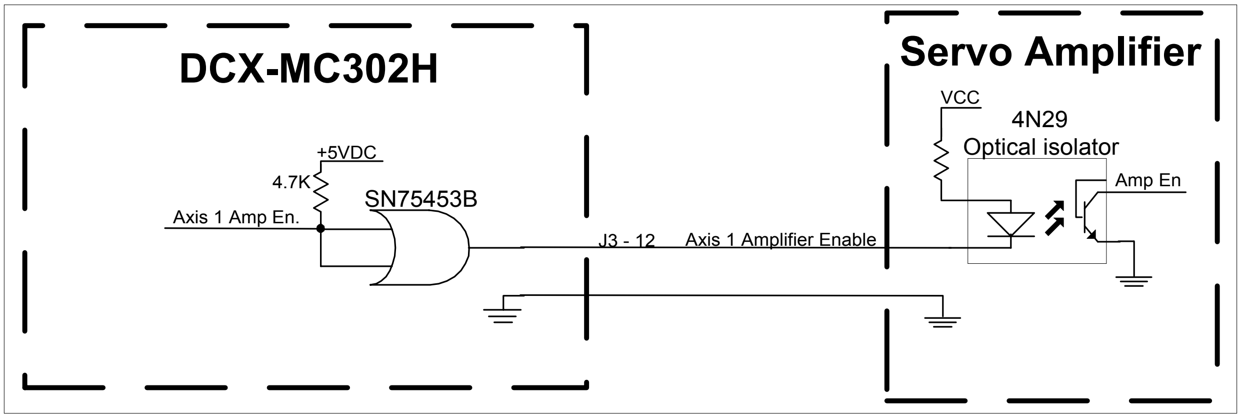
This limit circuit wll indicate that a limit is active if the switch is closed



This limit circuit wll indicate that a limit is active if:
1) The switch is open
2) Any component in the circuit fails (power supply, bi-directional opto isolator, broken wire, etc...

This is not the default configuration of the DCX, issue the **MC_LIMIT_INVERT** parameter of the ***MCSetLimits( )*** function.

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 CA-26IDS2-F-SPT 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-**

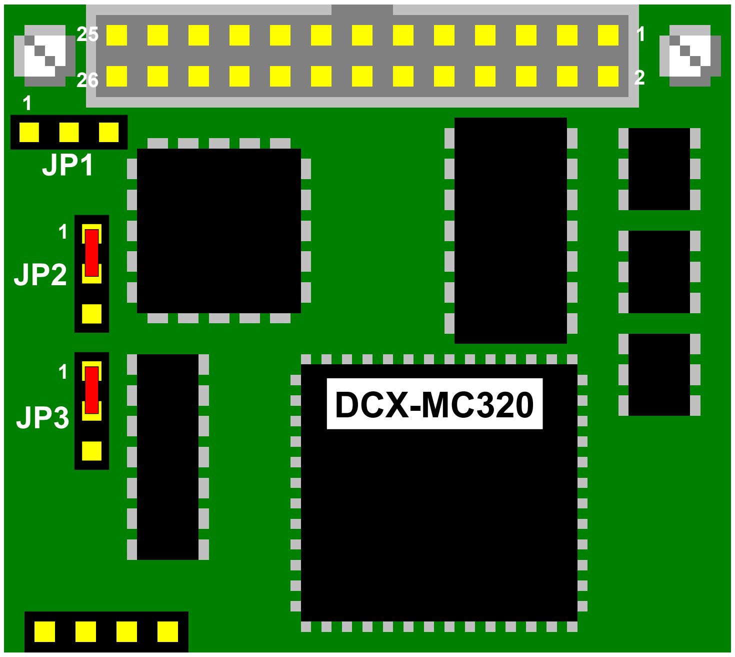
JP2 – Encoder Index Active Level Select)

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-

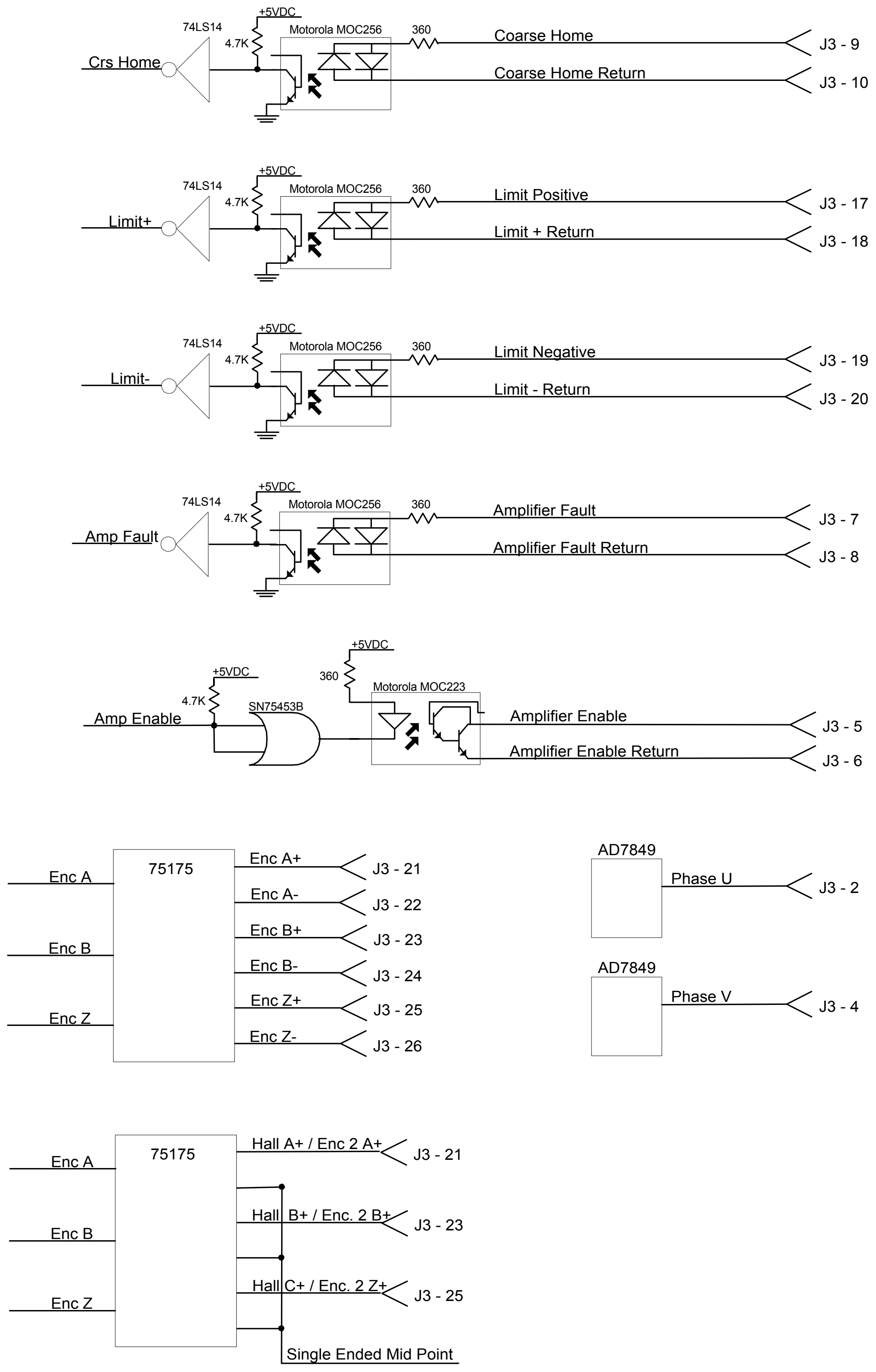
JP3 – Encoder Power Select (+5VDC or +12 VDC)

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.**)

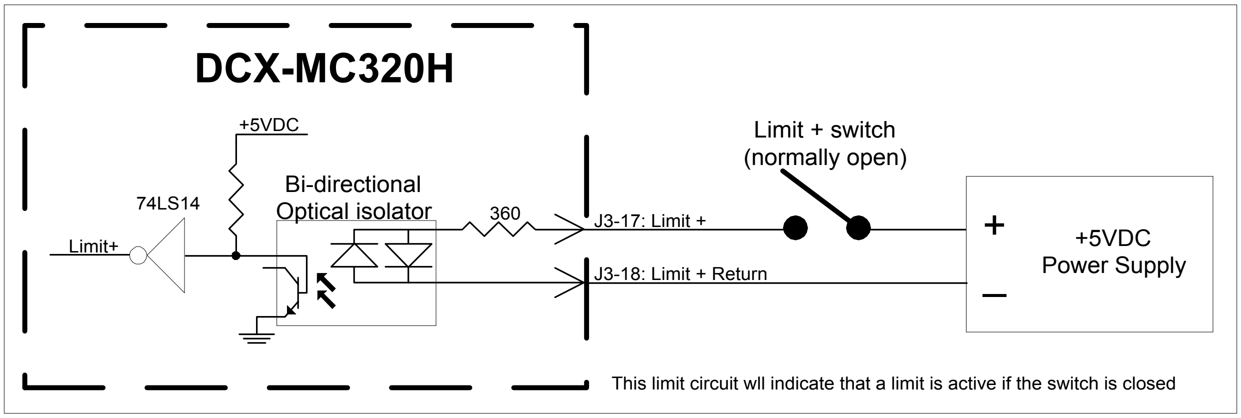
DCX-MC320 Module Layout



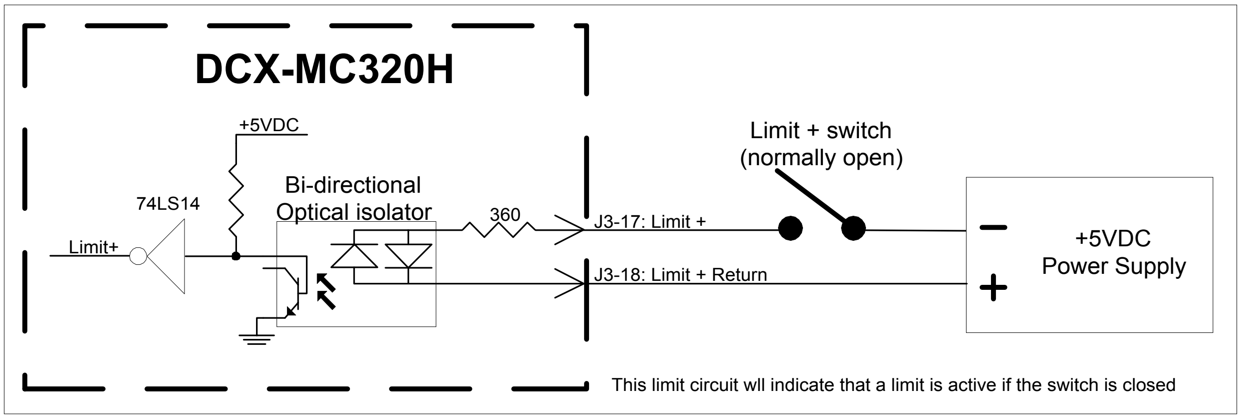
DCX-MC320H Axis I/O Interface Schematic



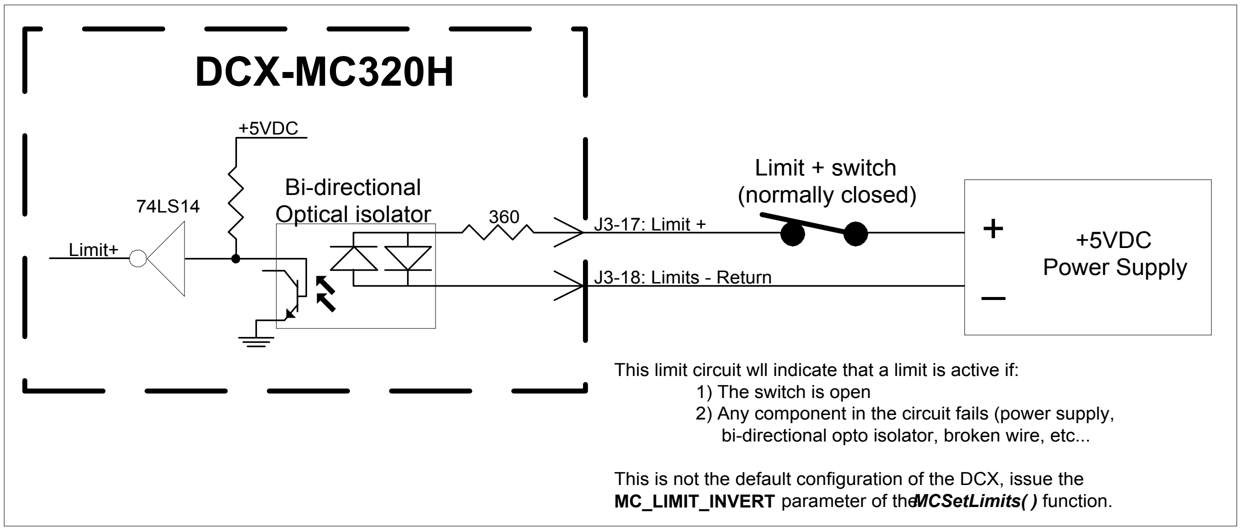
DCX-MC320H Optically Isolated Inputs Wiring Examples



This limit circuit wll indicate that a limit is active if the switch is closed



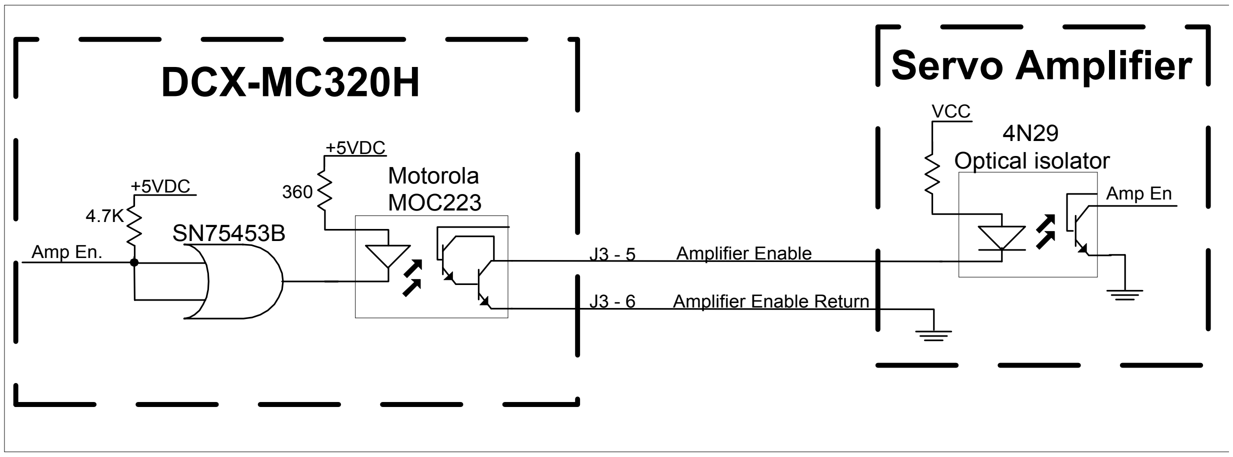
This limit circuit wll indicate that a limit is active if the switch is closed



This limit circuit wll indicate that a limit is active if:
1) The switch is open
2) Any component in the circuit fails (power supply, bi-directional opto isolator, broken wire, etc...

This is not the default configuration of the DCX, issue the **MC_LIMIT_INVERT** parameter of the ***MCSetLimits( )*** function.

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 CA-26IDS2-F-SPT 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-**

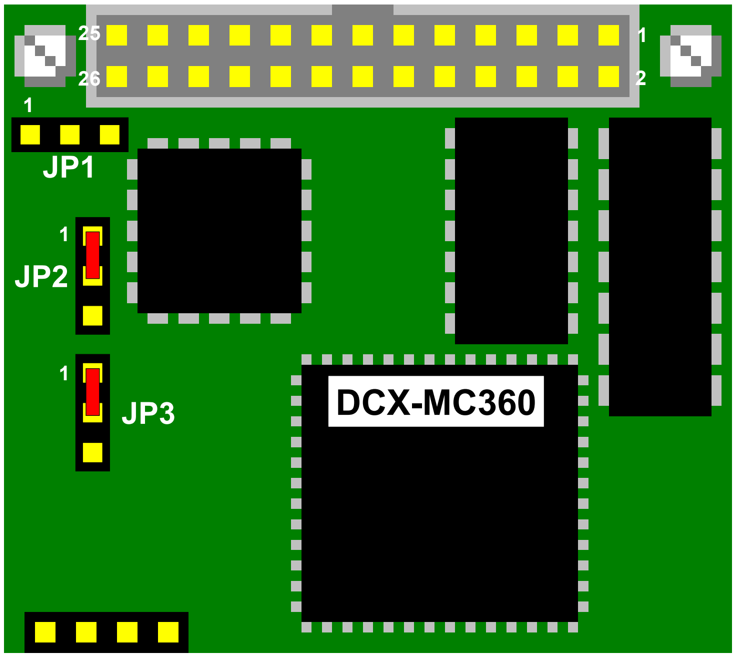
JP2 – Auxiliary Encoder Index Active Level Select

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-

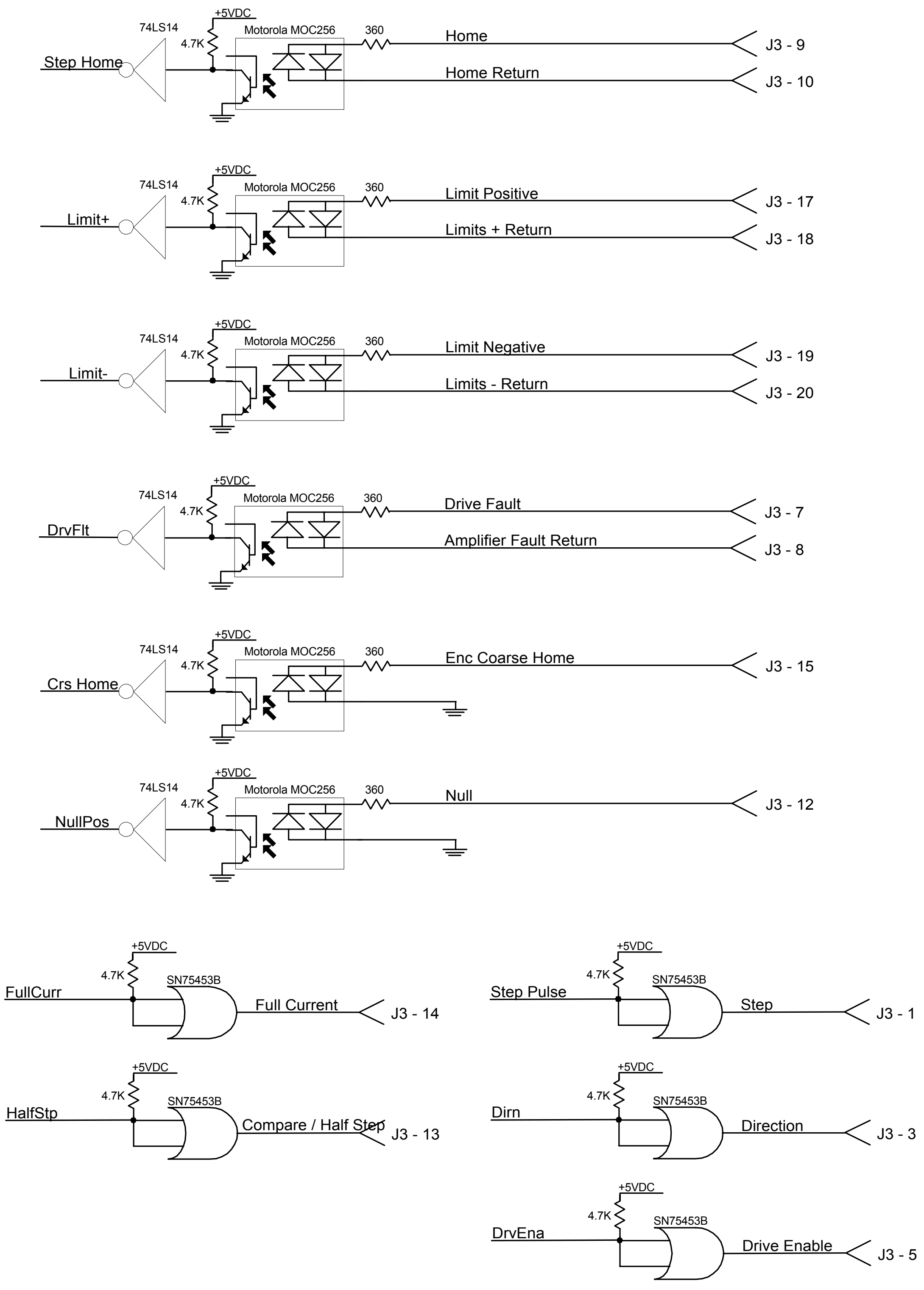
JP3 – Auxiliary Encoder Power Select (+5VDC or +12 VDC)

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**)

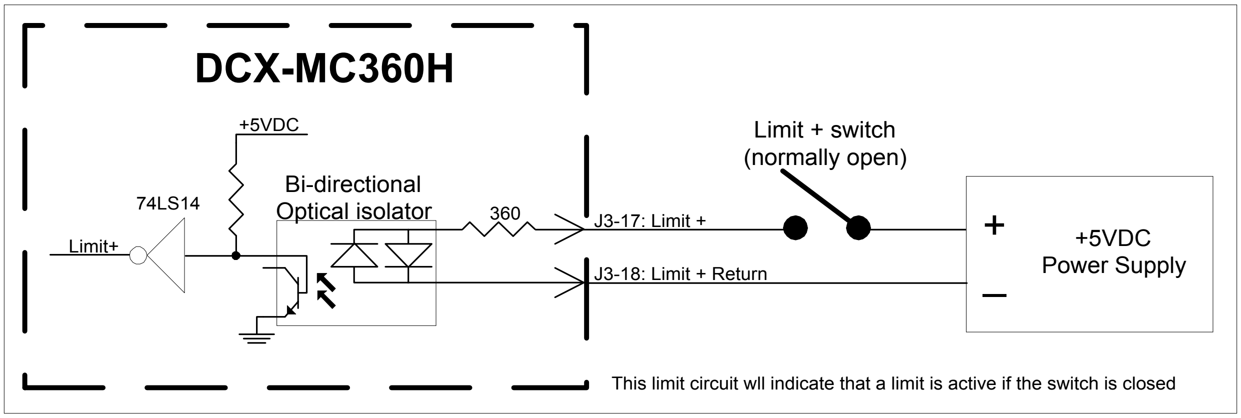
DCX-MC360 Module Layout



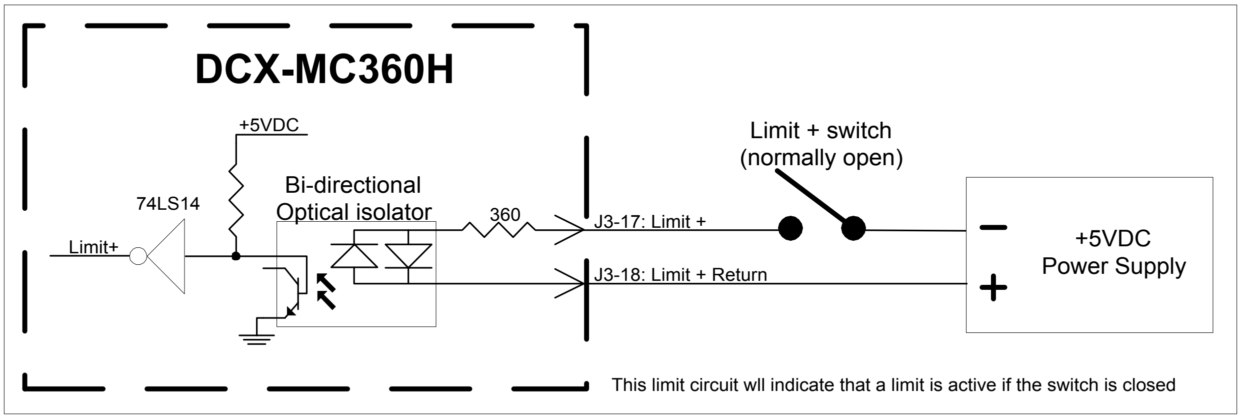
DCX-MC360H Axis I/O Interface Schematic



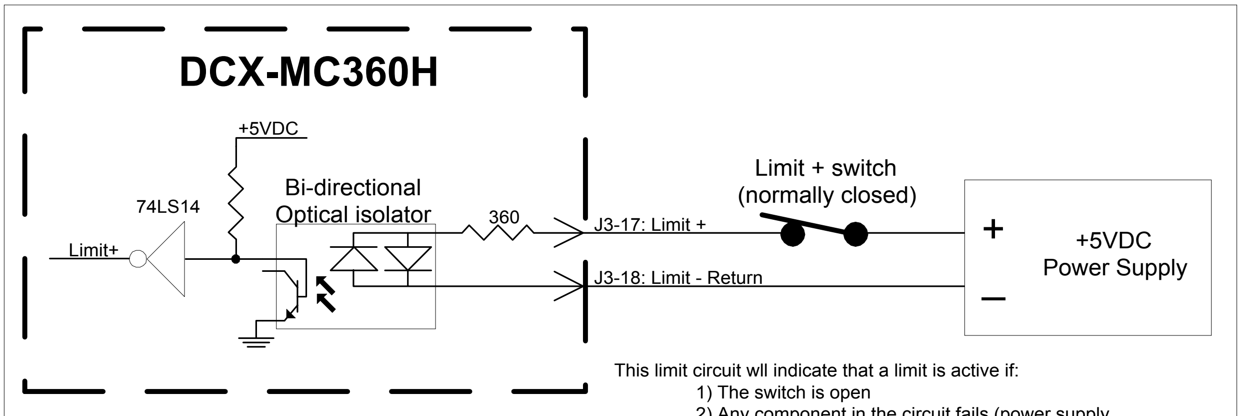
DCX-MC360H Optically Isolated Inputs Wiring Examples



This limit circuit wll indicate that a limit is active if the switch is closed



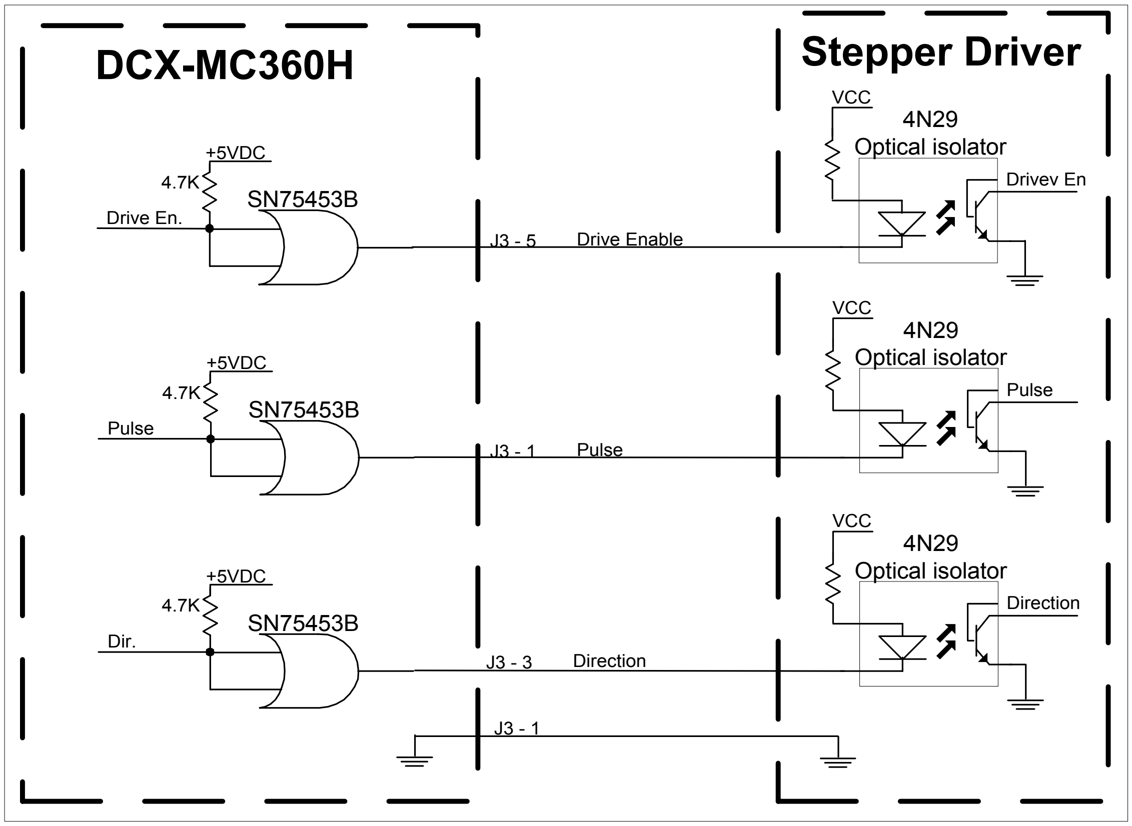
This limit circuit wll indicate that a limit is active if the switch is closed



This limit circuit wll indicate that a limit is active if:
1) The switch is open
2) Any component in the circuit fails (power supply, bi-directional opto isolator, broken wire, etc...

This is not the default configuration of the DCX, issue the **MC_LIMIT_INVERT** parameter of the***MCSetLimits( )*** function.

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.

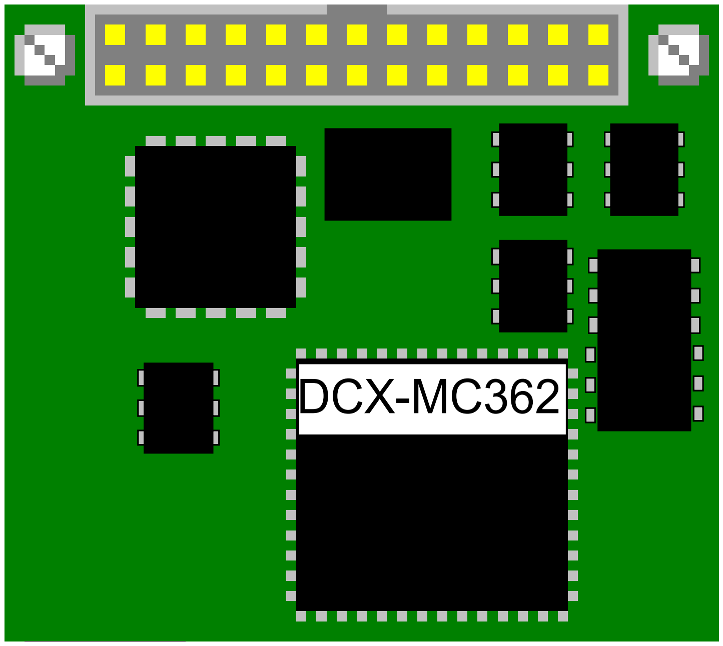
This signal is driven by a high current open collector driver and is suitable for direct connection to optically isolated inputs commonly found on a servo amplifier. Because of the characteristics of open collector drivers, no voltages will be present on these output signals unless pull-up resistors are connected to them.

DCX-MC362-H High Density connector signal map

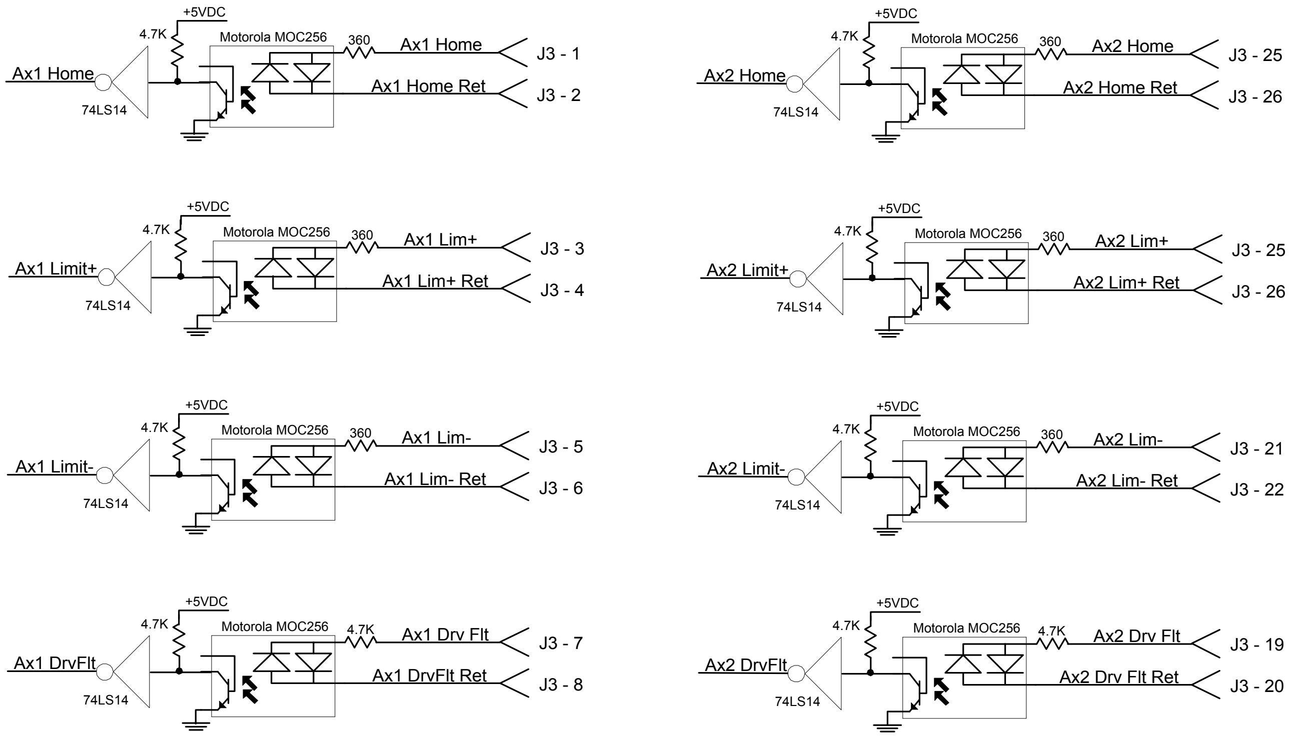
Module #1	Module #2	Module #3	Module #4	Module #5	Module #6	Module #7	Module #8	J3 Pin #	Description
J1 – 1	J2 – 1	J3 - 19	J4 - 19	J3 – 1	J4 – 1	J1 – 19	J2 - 19	1	Axis 1 Home: input (optically isolated)
J1 – 35	J2 - 35	J3 – 53	J4 – 53	J3 – 35	J4 – 35	J1 – 53	J2 – 53	2	Axis 1 Home opto isolator supply/return
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

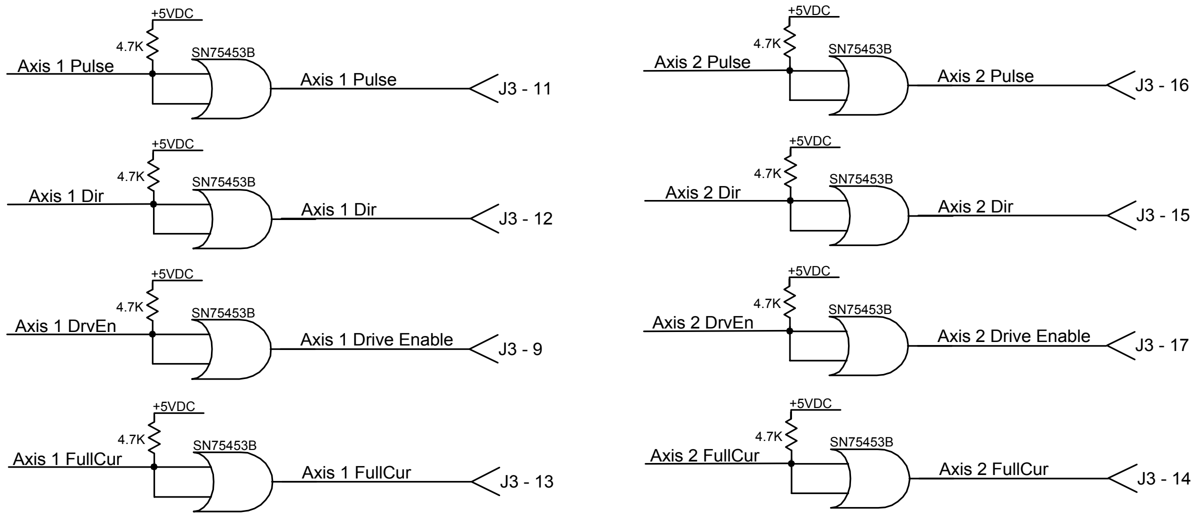
For a more complete signal description please refer to the previous five pages
Mating cable connector components: Amp: 787801-1 (offset connector), 788362-1 (backshell, offset)

DCX-MC362 Module Layout

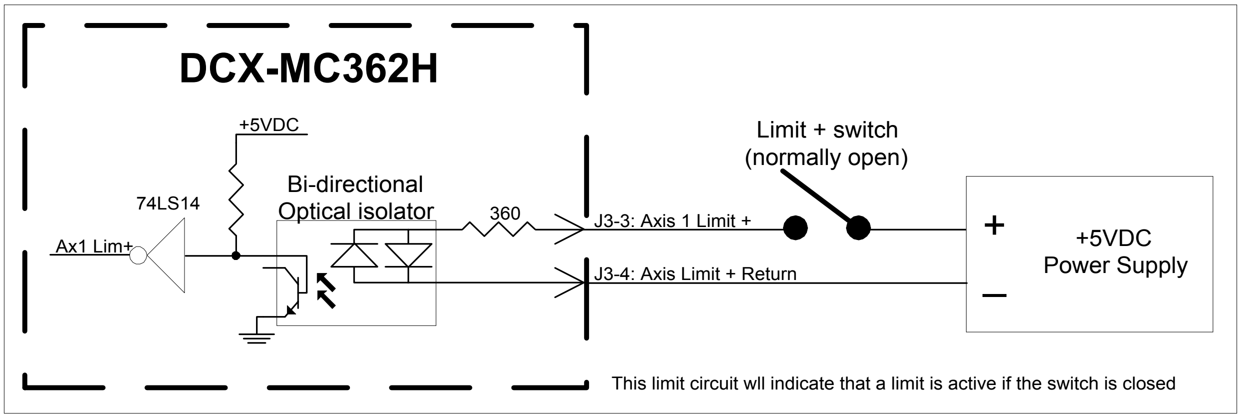


DCX-MC362H Axis I/O Interface Schematic

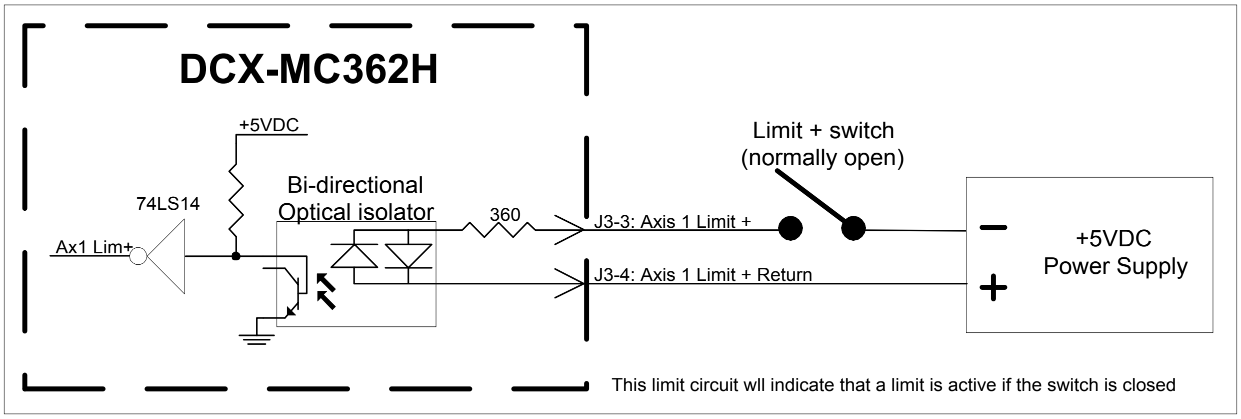




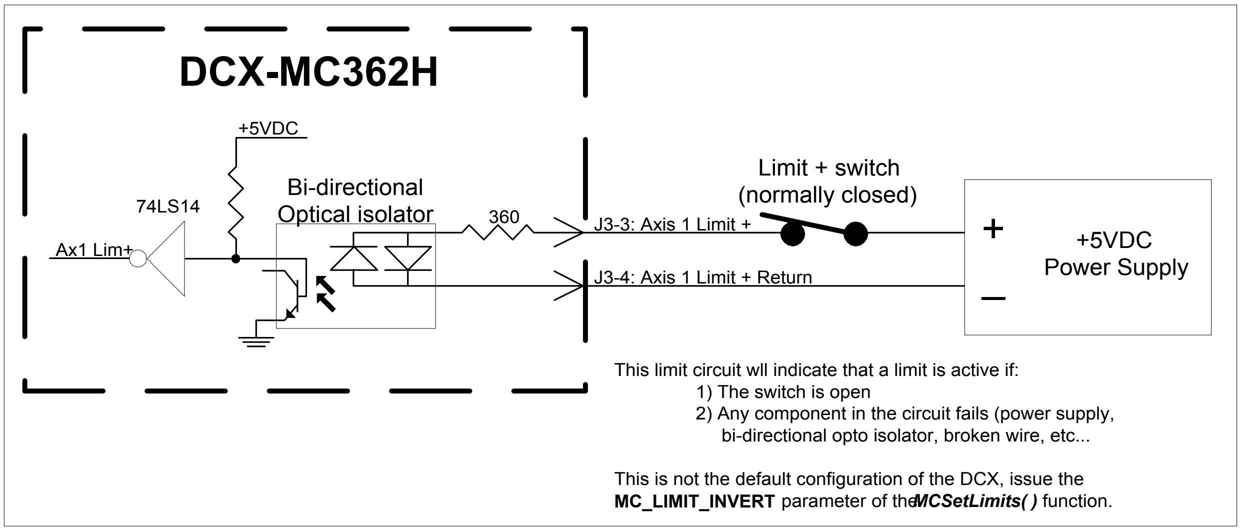
DCX-MC36H Optically Isolated Inputs Wiring Examples



This limit circuit wll indicate that a limit is active if the switch is closed



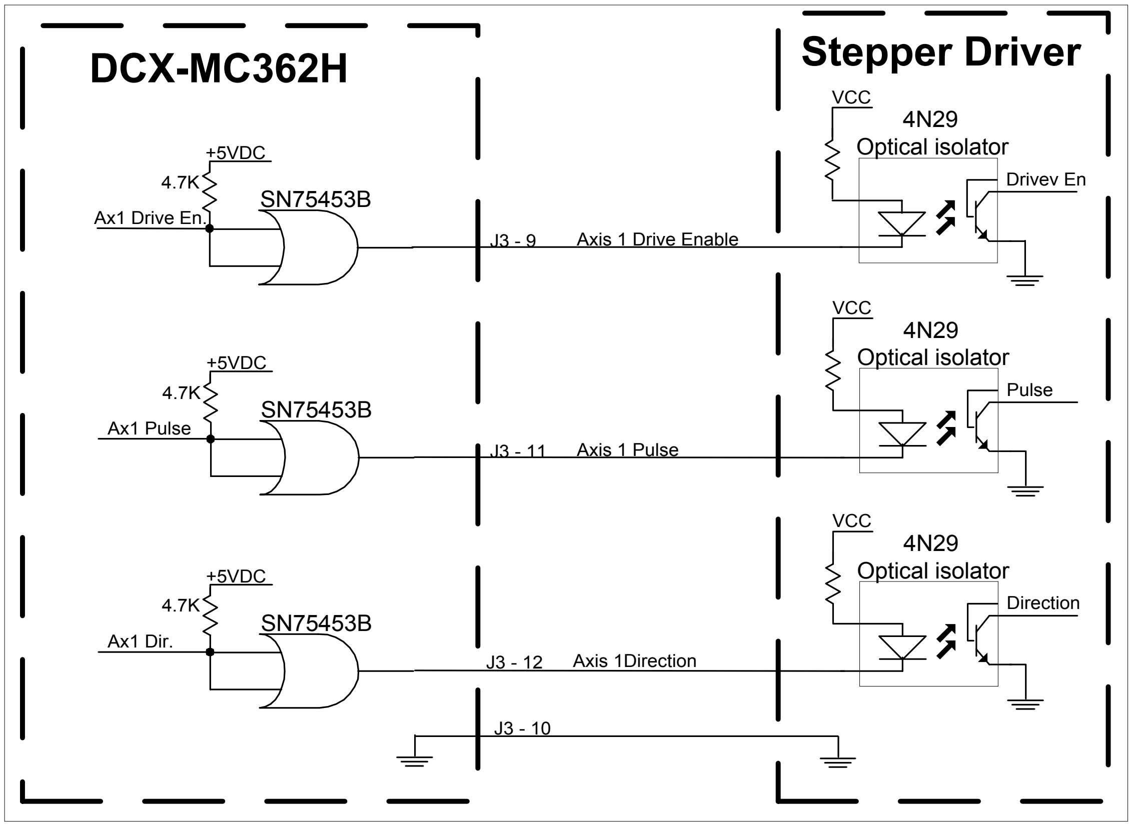
This limit circuit wll indicate that a limit is active if the switch is closed



This limit circuit wll indicate that a limit is active if:
1) The switch is open
2) Any component in the circuit fails (power supply, bi-directional opto isolator, broken wire, etc...

This is not the default configuration of the DCX, issue the **MC_LIMIT_INVERT** parameter of the ***MCSetLimits( )*** function.

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

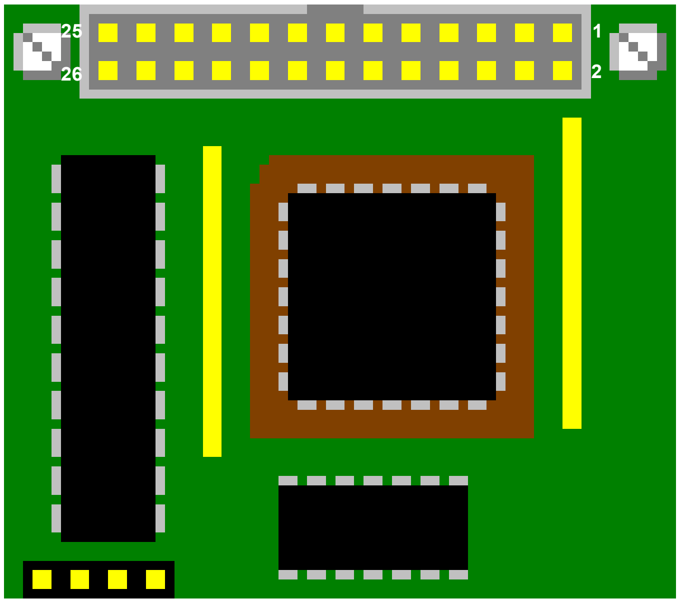
For a more complete signal description please refer to the previous five pages
Mating cable connector components: Amp: 787801-1 (offset connector), 788362-1 (backshell, offset)

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 CA-26IDS2-F-SPT 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

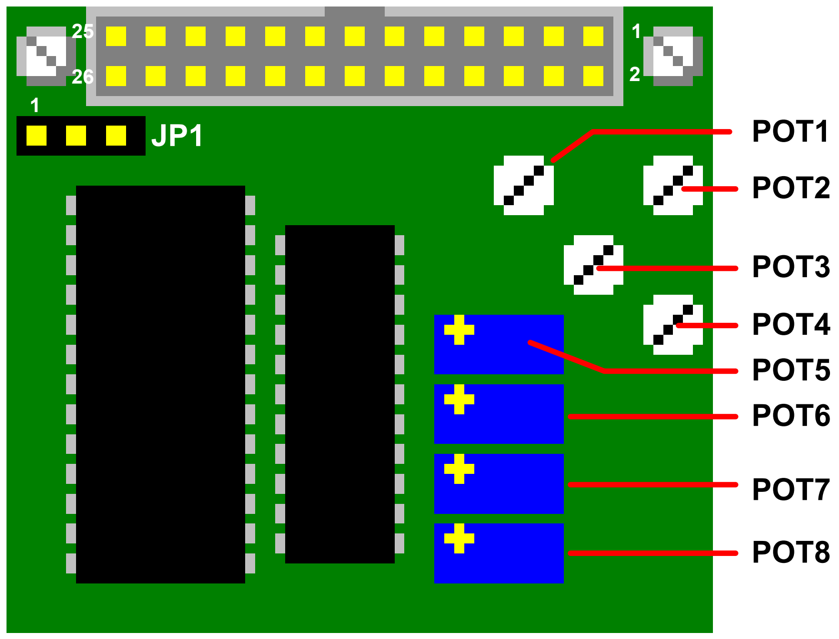
Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N CA-26IDS2-F-SPT or equivalent

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 CA-26IDS2-F-SPT 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 CA-26IDS2-F-SPT or equivalent

DCX-BF022 Configuration Jumpers - configuration in **bold type** denotes default factory shipping configuration

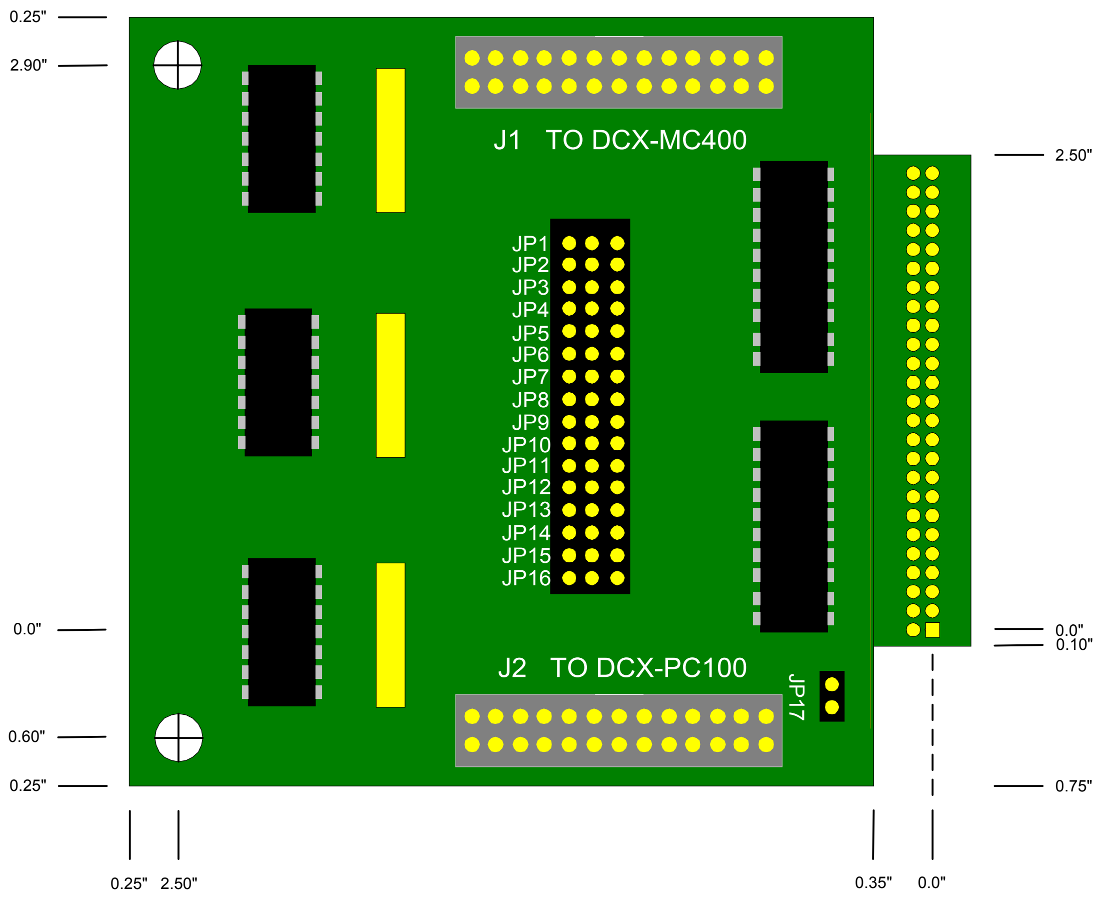
JP1 – JP16 Configure Digital channel as Input or Output

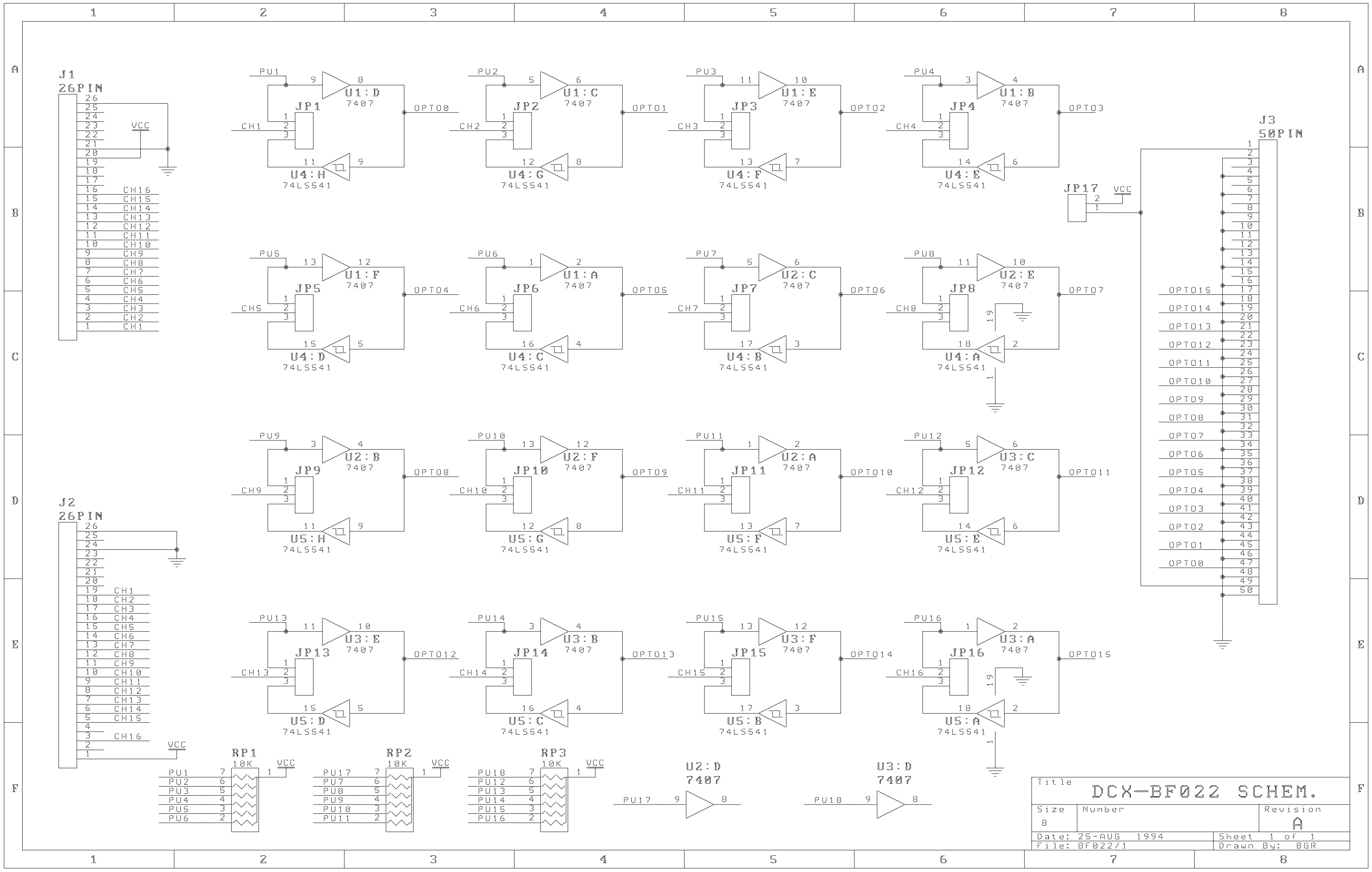
Pins	*Description*
1 to 2	**Configure channel as Output**
2 to 3	Configure channel as an Input

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

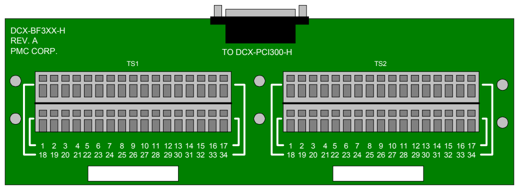
DCX-BF022 Interface layout



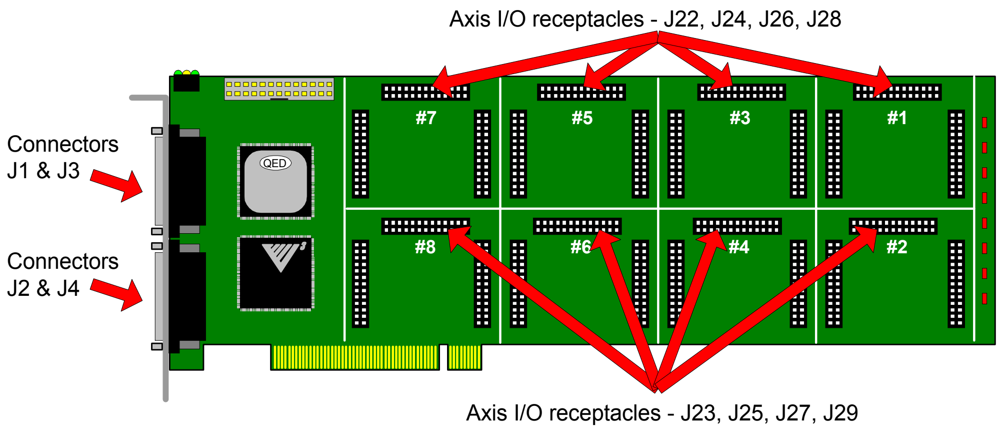

J1
26PIN
J2
26PIN
J3
50PIN
JP1
JP2
JP3
JP4
JP5
JP6
JP7
JP8
JP9
JP10
JP11
JP12
JP13
JP14
JP15
JP16
JP17
U1:D
U1:C
U1:E
U1:B
U1:F
U1:A
U2:C
U2:E
U2:B
U2:F
U2:A
U3:C
U3:E
U3:B
U3:F
U3:A
U4:H
U4:G
U4:F
U4:E
U4:D
U4:C
U4:B
U4:A
U5:H
U5:G
U5:F
U5:E
U5:D
U5:C
U5:B
U5:A
7407
74LS541
OPTO0
OPTO15
CH1
CH16
PU1
PU18
VCC
RP1
RP2
RP3
10K
U2:D
U3:D
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

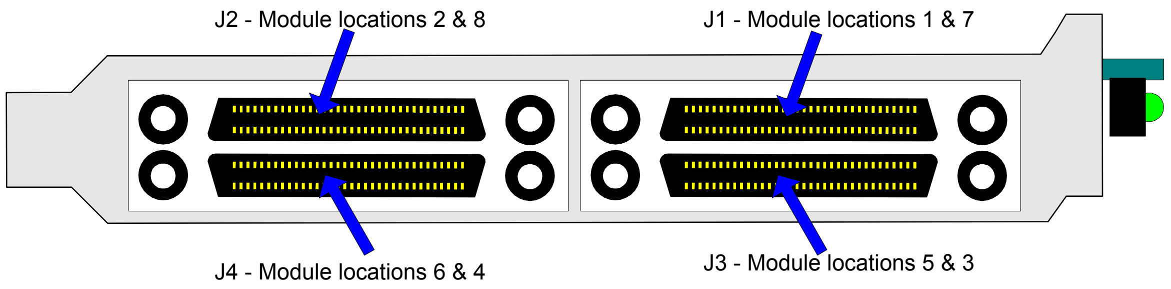
The DCX-BF3XX-H provides easy to use terminal strip contacts for –H DCX modules (MC300-H, MC320-H, MC360-H, MC400-H, MC500-H).



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:


Axis I/O receptacles - J22, J24, J26, J28
Connectors J1 & J3
Connectors J2 & J4
QED
#7
#5
#3
#1
#8
#6
#4
#2
Axis I/O receptacles - J23, J25, J27, J29

The diagram below details how the DCX-PCI300-H module locations 1 – 8 (receptacles J22 – J29) map into the high density connectors J1 – J4.

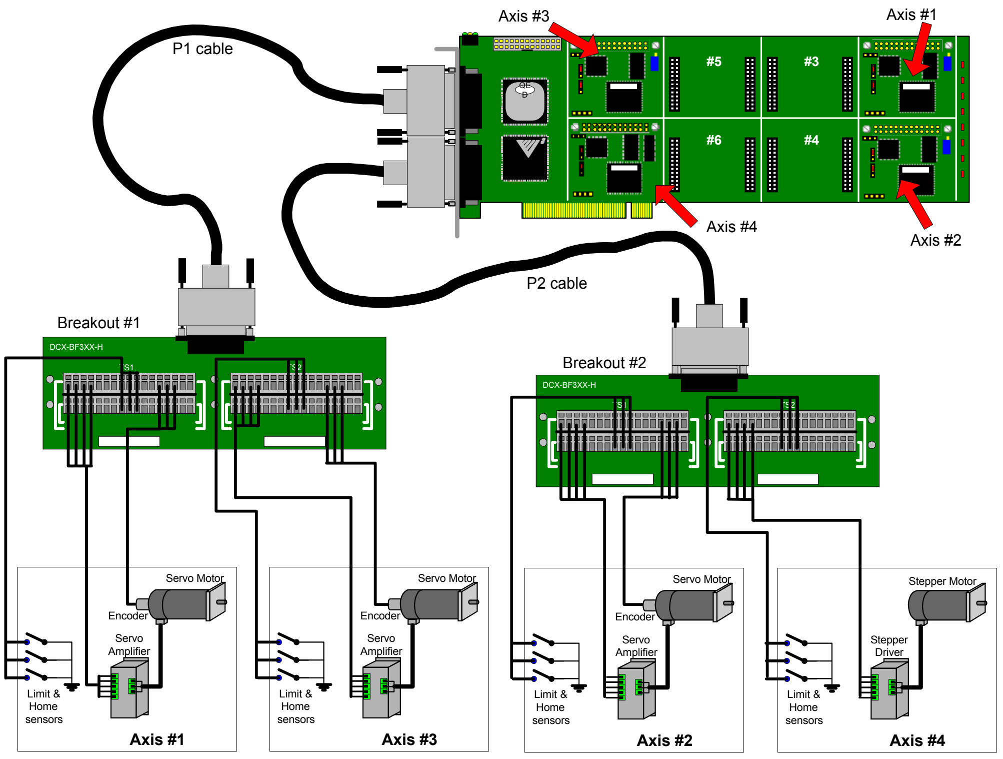

J2 - Module locations 2 & 8
J1 - Module locations 1 & 7
J4 - Module locations 6 & 4
J3 - Module locations 5 & 3

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.

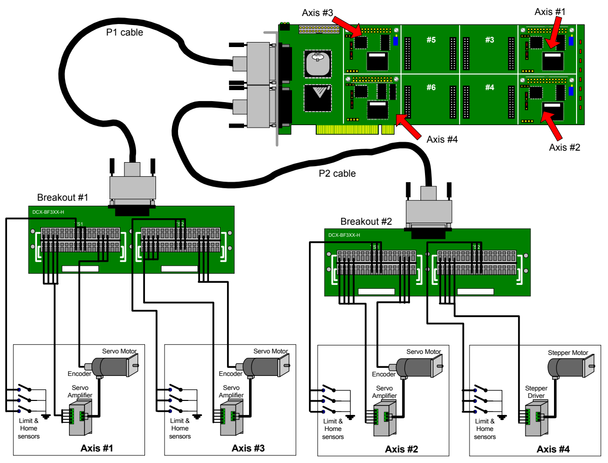


DCX-BF3XX-H signals pinout (when using single axis or I/O modules)
The following table details the pinouts –H DCX modules.

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)

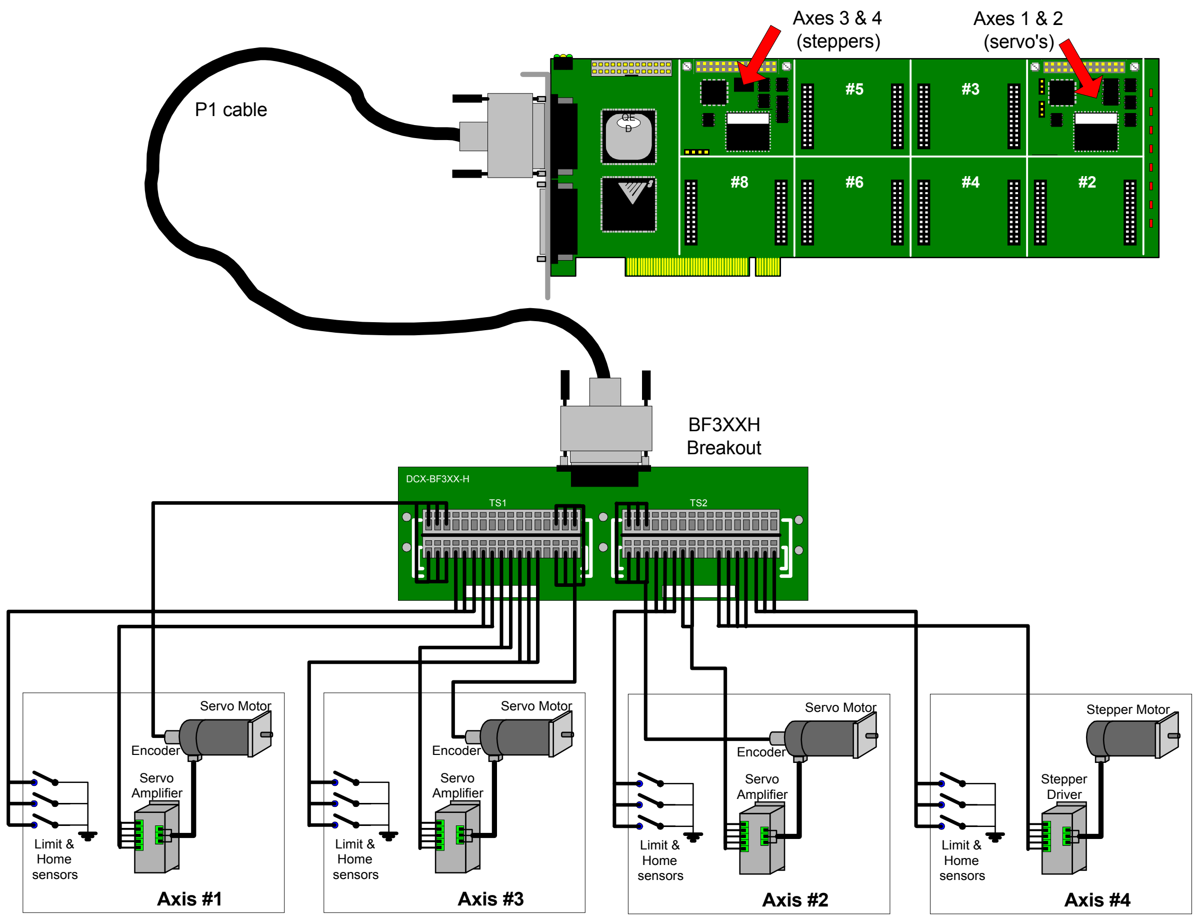
TS2	Signal
1	Axis 4 – Step
18	Axis 4 – Ground
2	Axis 4 – Direction
19	Axis 4 – Ground
3	Axis 4 – Drive En
20	Axis 4 – Ground
4	Axis 4 – Drive Fault
21	Axis 4 – Ground
5	Axis 4 – Home
22	Axis 4 – Home ret
6	Ground
23	
7	
24	Ground
8	Ground
25	
9	Axis 4 – Full/Half cur
26	Ground
10	Ground
27	
11	
28	Ground
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

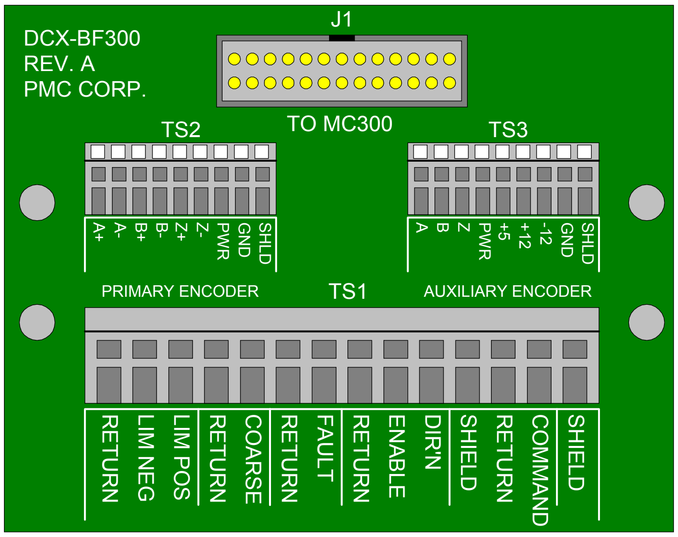
Axis #3 (module 2)
BF3XX-H TS2 contacts
1-8 & 18-25.

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

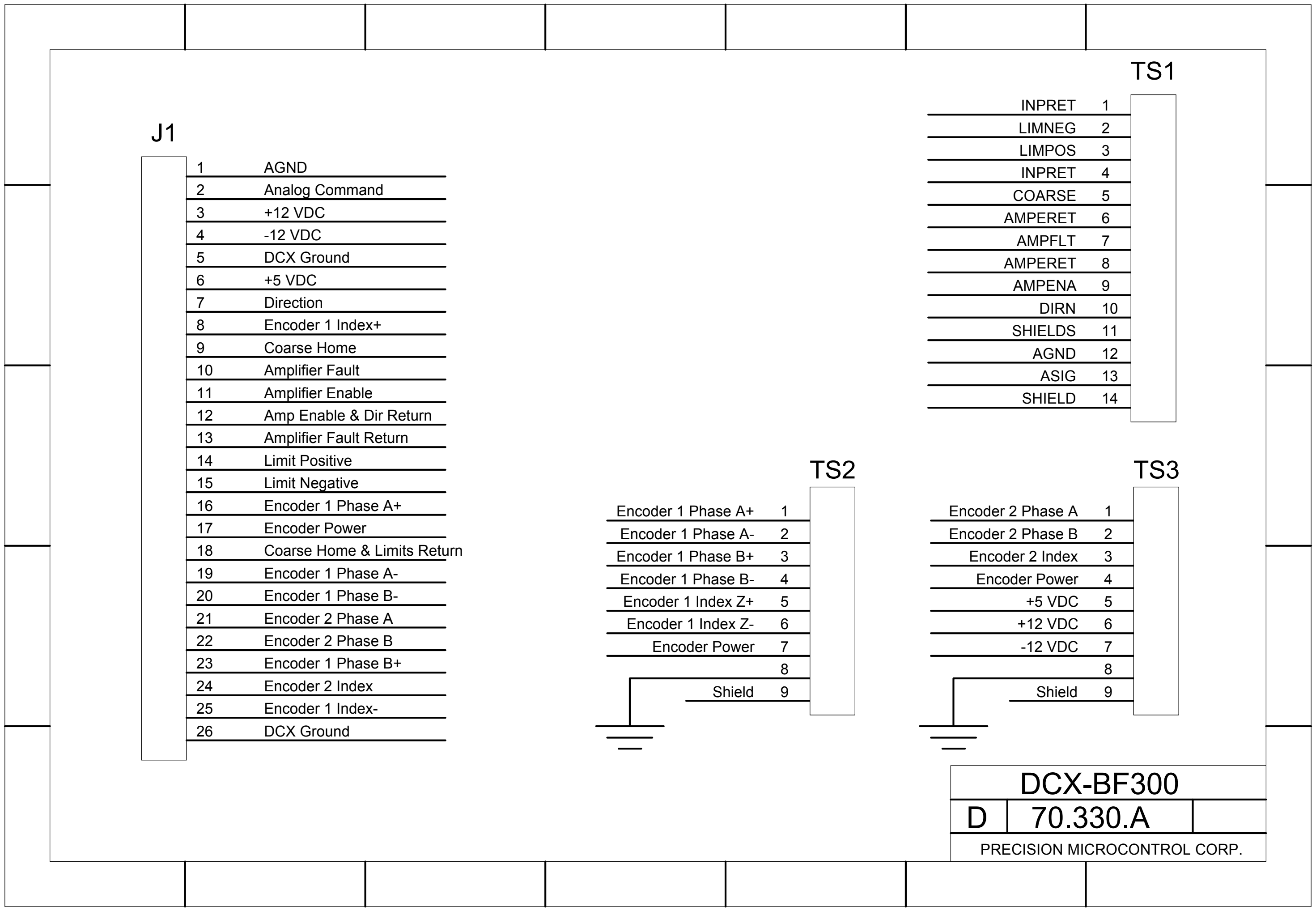
Terminal strip TS3

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

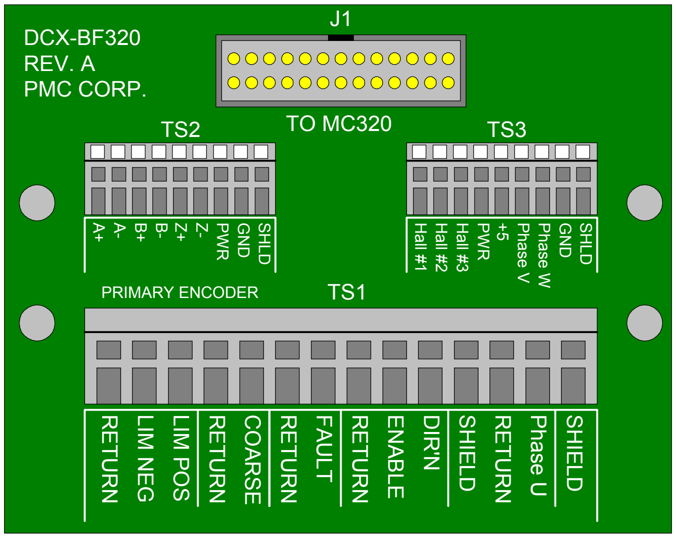
DCX-BF300-R to DCX-MC300 Connections (continued):

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

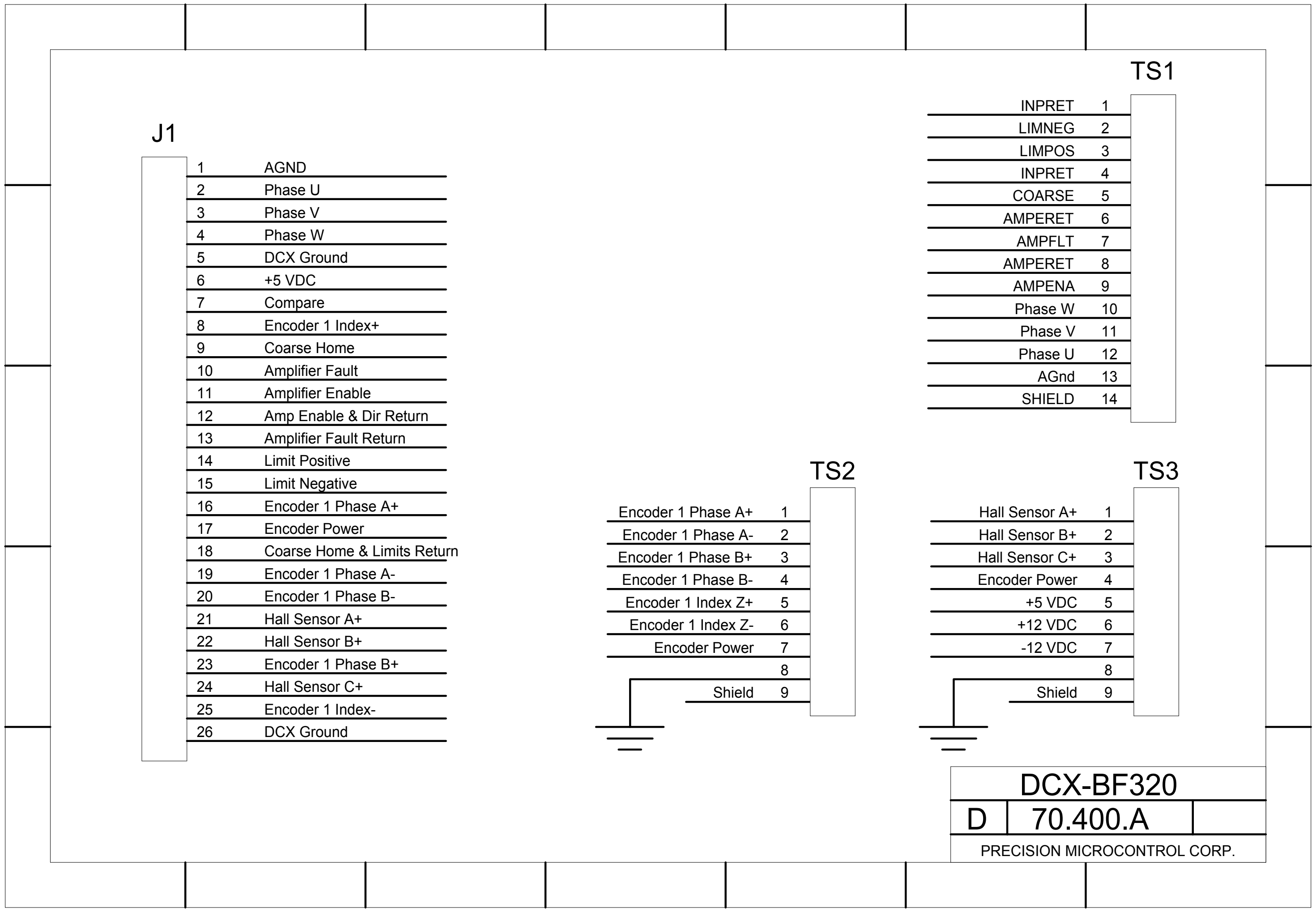
Terminal strip TS3

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

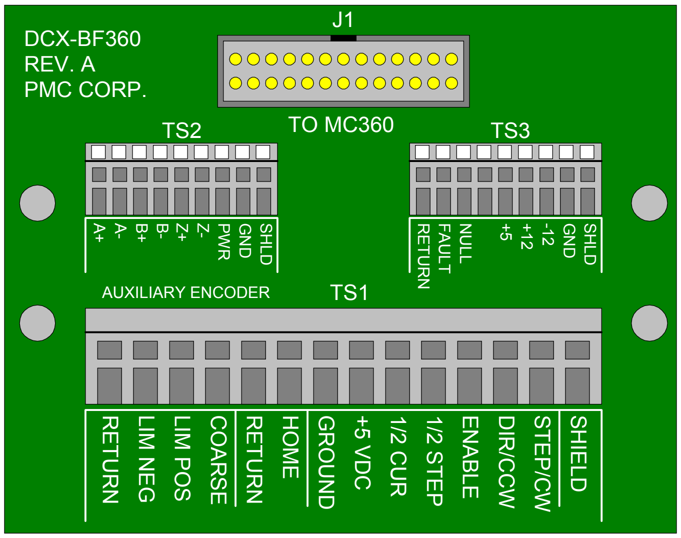
DCX-BF320-R to DCX-MC320 Connections (continued):

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

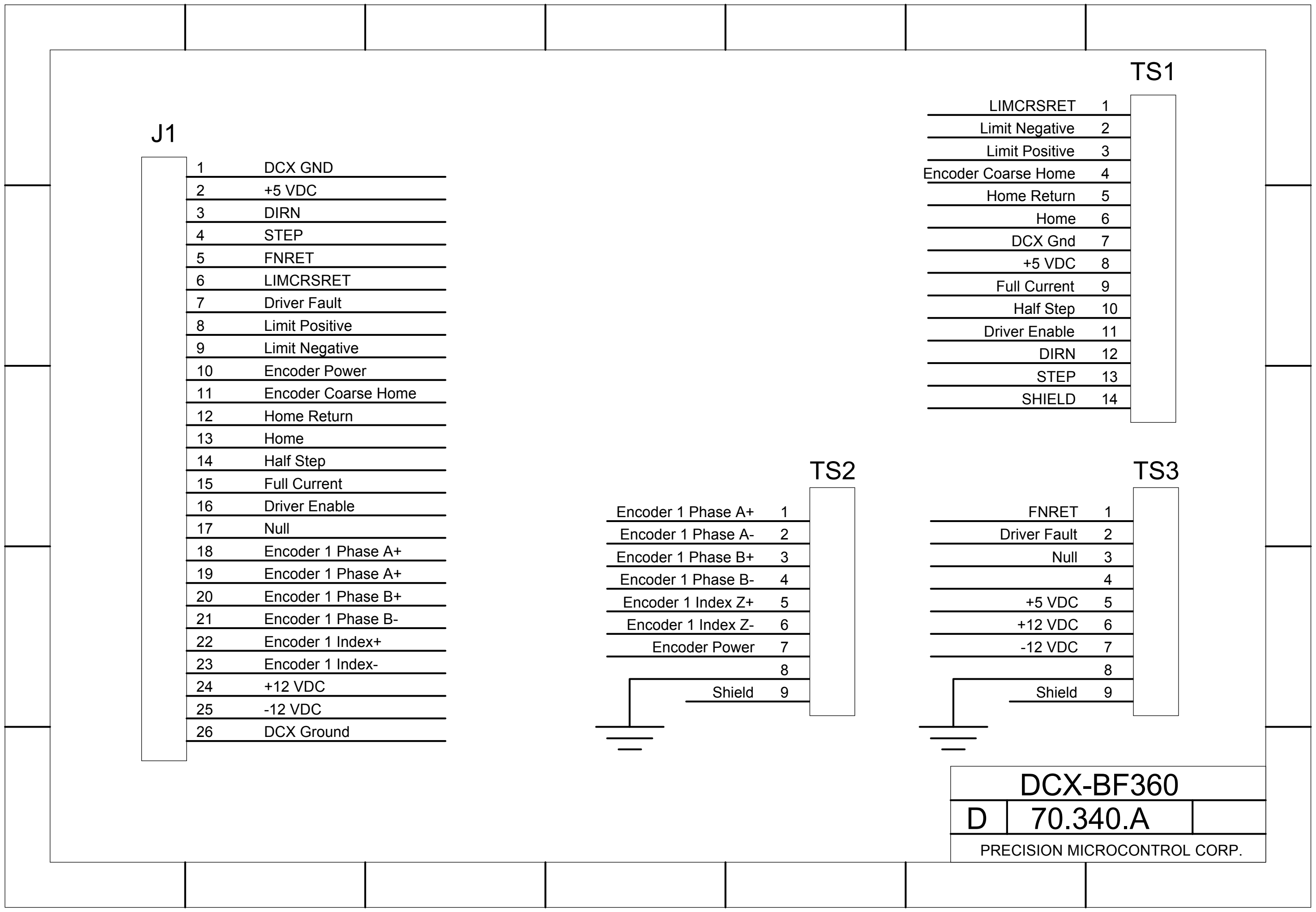
Terminal strip TS3

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

19

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.

Appendix Contents

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-MC302	.4	.02	.02	A
DCX-MC320	.4	*	*	A
DCX-MC360	.4	-----	-----	A
DCX-MC362	.4	-----	-----	A
DCX-MC400	.25	-----	-----	A
DCX-MC500	.1	*	*	A
DCX-MF300	.01	*	*	A
DCX-MF310	.16	-----	-----	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
Encoder counts / Steps scaling ratio	1
Proportional Gain	.2
Derivative Gain	.1
Integral Gain	.01
Integration Limit	50
Maximum Following Error	1024
Motion Limits	disabled
Soft Motion Limits - Low Limit setting	0
Soft Motion Limits - High Limit setting	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

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