DCX-VM200

Modular Multi-Axis Motion Controller

User's Manual

Revision 4.0

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

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

www.pmccorp.com

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

LIMITED WARRANTY

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

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

Information in this document is subject to change without notice.

Acrobat and Acrobat Reader are registered trademarks of Adobe Corporation.

Table of Contents

User manual revision history

Revision	*Date*	*Description*
1.1a	11/1/94	Initial release (revision level set to match the current firmware)
4.0	1/13/2K	Added firmware revisions to rev. 4.0a Converted to Word - User Manual Format Changed manual organization structure Added Application Solutions chapter Added numerous feature / application descriptions Added Troubleshooting flow charts Added cubic spline interpolation to contouring. Added Profile Parabolic command. Added Move Absolute command description. Added description of backlash compensation function and commands. Added description of file commands. Added description of plotting commands. Added appendix describing HPGL commands. Added PDF document printing description Made miscellaneous corrections and changes.

Chapter 1

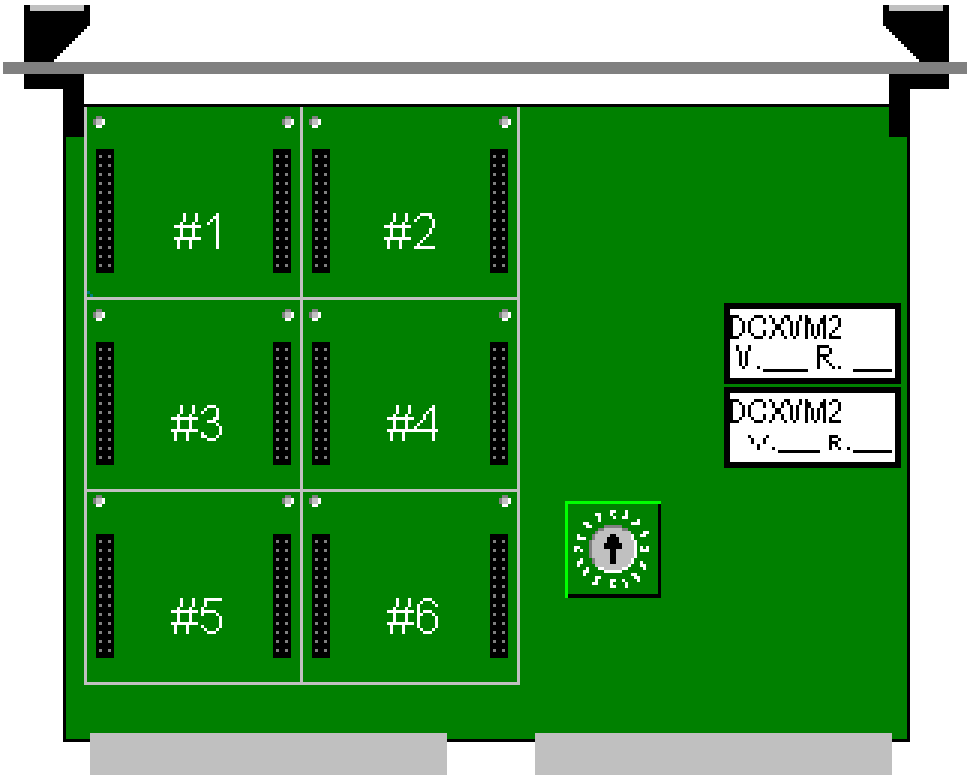
Introduction

This document describes the installation and use of the DCX-VM200 Modular Multi-Axis Motion Control System. In a typical application, the DCX is installed in a 6U VME chassis. A VME CPU will run a high level language application program, written by the machine builder. The application program issues motion control commands via the VME address and data bus to one or more DCX controllers. The DCX executes these motion functions independent of the host.

The modular architecture of the DCX system allows the user to '**mix and match**' components to meet the specific requirements of each application. The DCX system controls the motion of any combination of servos and stepper motors simultaneously. In addition the DCX system supports; direct motor drive (no servo amplifier required) of small servo motors, expandable digital and analog I/O, and stand alone applications.

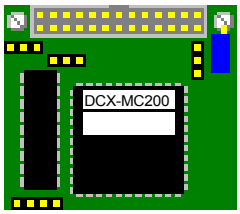
The term DCX refers to a system consisting of from 1 to 7 circuit boards assembled together to form a motion control assembly. The main component of the assembly is the DCX-VM200 "motherboard". It is a 6U peripheral card (size (approximately 9 1/4" X 6 3/8" x 13") peripheral card.

The DCX-VM200 motherboard is the platform upon which the DCX motion control system is built. It communicates with the VME host via the P1 address and data bus. On board dual ported memory is used to pass data between the DCX-VM200 and the host CPU. The on board CPU (Motorola 68020) allows the DCX-VM200 to operate autonomously from the host, freeing the VME to process critical events while the DCX handles all motion control.

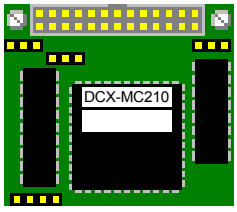


On this DCX-VM200 motherboard, the user can install as many as 6 smaller "daughter boards" known as "DCX modules". There are seven DCX module types available which allow the DCX system to be configured for a variety of applications. A key feature of the DCX system is its ability to sense which DCX modules are present. This results in easy system configuration; simply install whatever modules the application calls for. The logic on the motherboard will adjust its' operation accordingly. The DCX system also offers the OEM machine builder the option of routing all of the motion control (+/- 10V, PWM+/PWM-, and Step/Direction), feedback (Encoder phase A+, A-, B+, B-, Z+, and Z-), and I/O signals to the 'outside world' via the P2 connector to a custom back plane.:

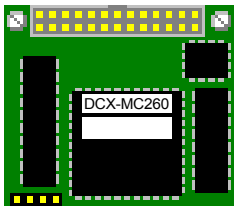
DCX Motion Control Modules



DCX-MC200 / DCX-MC200V - Servo Motor Control Module
+/- 10 volt, 12 bit analog command output for use with a servo amplifier/drive
Inputs - Encoder Coarse Home, Limit +, and Limit -, Amplifier Fault
Output – Amplifier Enable
Quadrature Incremental Encoder Interface
Primary - Single ended (A, B, Z) or Differential (A+, A-, B+, B-, Z+, Z-)
Auxiliary - Single ended (A, B, Z+, Z-)

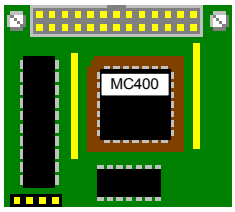


DCX-MC210 / DCX-MC210V - Servo Motor Control Module
PWM output for direct drive of small motors (12W, 12 volt @ 1 A)
Inputs - Encoder Coarse Home, Limit +, and Limit -, Amplifier Fault
Output – Amplifier Enable
Quadrature Incremental Encoder Interface
Primary - Single ended (A, B, Z) or Differential (A+, A_, B+, B-, Z+, Z-)
Auxiliary - Single ended (A, B, Z+, Z-)

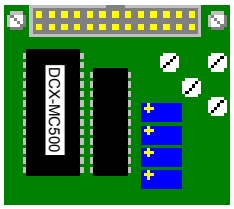


DCX-MC260 / DCX-MC260V – Stepper Motor Control Module
Pulse/CCW and Direction/CW outputs for use with a stepper driver
Inputs - Home, Limit +, and Limit -, Null
Outputs – Drive Enable, Half/Full step, Full/Half current
Quadrature Incremental Encoder Interface
Auxiliary - Single ended (A, B, Z) or Differential (A+, A_, B+, B-, Z+, Z-)

DCX General Purpose I/O Modules



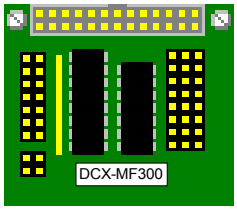
DCX-MC400 / DCX-MC400V – 16 Channel Digital I/O Expansion module
Each channel is individually programmable as either an input or output
TTL level (0 – 5 volt, 2 ma sink/source)



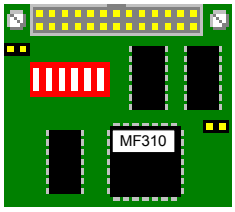
DCX-MC500 / DCX-MC500V – Analog I/O Expansion module
Inputs – 4 channels, 0 – 5 volts, 12 bit
Outputs – 4 channels, 0 – 5 volts and/or –10 - +10 volts, 12 bit

Options:
MC510 – 4 input channels only
MC520 – 4 output channels only

DCX Auxiliary Communication Modules for Stand Alone Applications

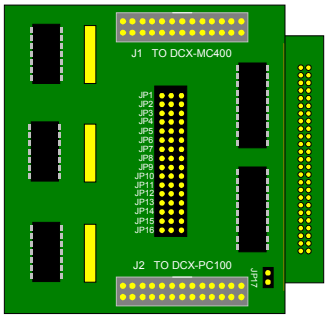


DCX-MF300 / DCX-MF300V – RS232 Interface Module
ASCII command interface for stand alone applications



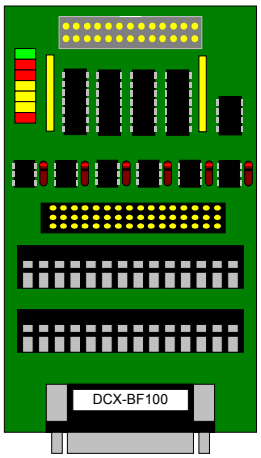
DCX-MF310 / DCX-MF310V – IEEE-488 Interface Module
ASCII command interface for stand alone applications

DCX Interconnect and Isolation Assemblies



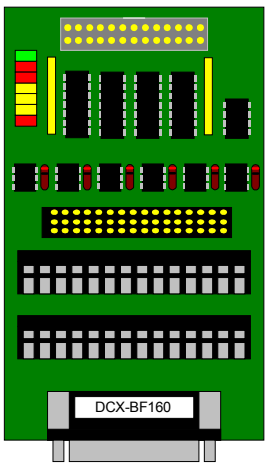
DCX-BF022 – Opto 22 Relay Rack Interface
16 digital I/O channels, each channel individually configured via user installed jumper as input or output. Connects to DCX digital I/O via 26 conductor ribbon cable.

DCX Interconnect and Isolation Assemblies (continued)



DCX-BF100 – Opto isolation and Interconnect assembly for DCX Servo Motor Control Modules (DCX-MC200, DCX-MC210)
Opto isolated inputs – Enc. Coarse Home, Limit +, Limit -, Amp Fault
Open collector output – Amplifier Enable
Differential receiver for Index +, Index –
External system connections via DB25 or two 14 contact screw terminal strip

LED indicators for:
- Amplifier enable
- Encoder Coarse Home
- Limit +
- Limit –
- Amplifier Fault



DCX-BF160 – Opto isolation and Interconnect assembly for DCX Stepper Motor Control Module (DCX-MC260)
Opto isolated inputs – Home, Limit +, Limit -, Null, Enc. Coarse Home
Open collector output – Drive Enable
Differential receiver for Index +, Index –
External system connections via DB25 or two 14 contact screw terminal strip

LED indicators for:
- Drive enable
- Home
- Encoder Coarse Home
- Limit +
- Limit –
- Null Position

Chapter Contents

Chapter 2

Installation

In a typical DCX installation the DCX motion controller is installed in an available slot of a VME chassis. Power (+5V, +12V, and –12V), Ground reference, and communications (Address, Data, and Read/Write control signals) are supplied via the P1 connector of the VME bus.

For non 'VME host' based applications, the on-board intelligence of the DCX allows it to be used as a 'stand-alone' controller. Optional RS-232 or IEEE-4888 communication interfaces allow a host computer to communicate with the DCX for programming and/or operator control. For stand-alone installation information please refer to **Stand-alone Application** section in the **Appendix** at the end of this user manual.

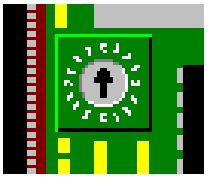
'VME host' based Application Installation

The basic steps of installing a DCX motion controller for 'VME host' based applications are as follows:

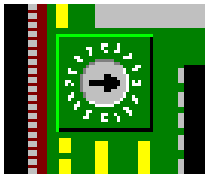
Select the memory address of the DCX
Communicating with the DCX controller
Install and wire DCX servo and stepper motion control modules
Install and wire digital and analog I/O modules
Verify servo operation (refer to **DCX Servo Basics** in the **Motion Control Chapter**)
Verify stepper operation (refer to **DCX Stepper Basics** in the **Motion Control Chapter**)
Verify I/O operation (refer to **DCX General Purpose I/O Chapter**)

Setting the memory address of the DCX
When a DCX board is plugged in to a VME system, it is necessary that the DCX's dual ported memory to be configured to occupy a specific 4K space somewhere within the 24 bit address range 0h to FFFFFFh. Note that the DCX only recognizes address lines A1 through A23 on the VME bus. Data transfers will occur on data lines D0 through D15. The default address of the dual ported memory, which is set by jumper JP8, is 100000H to 100FFFh. Jumper JP8 supports memory address ranges from 100000 hex to F0000 hex. A table in appendix B details the possible settings of jumper JP8.

The 17 position rotary SW1 is used to define at which 4K offset the DCX will reside. Assuming that jumper JP8 is at its factory default setting, the DCX will occupy 4096 bytes somewhere between 100000 hex and 10FFFF hex in the VME systems memory space. If the rotary switch SW1 on the motherboard is set to 0, the DCX will occupy 100000 hex through D00FFF hex. If it is set to 1, it will occupy D01000 hex through D01FFF hex, and so on. In the host's memory map, the lowest numbered memory location that a DCX board occupies is referred to as the 'base' address. The two graphics below show the memory switch setting for a DCX at address D0000:0000 (SW1=0)and D4000:0000 (SW1=4).



Default memory address switch
SW1=0 (100000 - 100FFF) hex



Memory address switch
SW1=4 (104000 – D04FFF) hex

Communicating with the DCX Controller

Due to the problems of supporting the wide variety of processors (Motorola 68K and Intel X86) and operating systems (Unix, NT, OS9) used in VME applications, the DCX-VM200 motion controller does not include high level programming tools or utilities. The following sections describe the ASCII and Binary interfaces supported by the DCX controller. The ASCII command interface is intended primarily just for initial system integration, all controller operations are supported but the data transfer rate will be very slow. The Binary command interface is designed to support typical high performance VME based control systems. For further assistance with VME based application please contact PMC Technical Support.

ASCII Command Interface
The ASCII command interface allows the host to send PMC's Motion Control Command Language (MCCL) commands to the DCX and receive DCX replies.

In order for the host computer to communicate to the DCX in ASCII characters, two mailbox locations have been defined. The data input mailbox is used to send data to the DCX. This mailbox is the byte located at **800 hex** from the base address. For a DCX controller with the default configuration, this will be absolute address 100800 hex. The data output mailbox is used to receive data from the DCX. This mailbox is the byte located at **804 hex** from the base address. This will be absolute address 100804 hex.

A simple protocol allows ASCII characters to be transferred to and from DCX boards. In order to send an ASCII character to a board, first read the contents of its data input mailbox, if it is 0, write the data into the mailbox. If the contents are non-zero, the previous data put in the mailbox has not yet been processed by the DCX controller. The host should continue checking the mailbox until the contents become 0, and then write the data.

In order to receive ASCII characters from a DCX, read the contents of its data output mailbox. If the value read is non zero, it is a valid character and the host should save the character and then write a 0 to the data output mailbox to indicate it has received the data. If the contents of the mailbox is 0, the

DCX has not placed new data in the mailbox. In this case the host can continue checking the mailbox for new data.

In addition to the command interfaces, the host can read motor information directly from the DCX's memory. Please refer to the **VME Bus Dual Port RAM** description in the **Appendix**.

Binary Command Interface
In order to achieve the fastest VME host to DCX command throughput, the DCX supports a "Binary" command interface. This interface allows the host to write binary coded commands directly into a command buffer in the DCX's dual ported memory, thus bypassing the command interpreter. Using this method, any command replies the DCX generates will be returned in binary format to the host through a reply buffer in the DCX's dual ported memory. The procedure for using the DCX Binary Command Interface is described below:

1. Beginning at the start of the command buffer in the DCX's dual port memory, write up to 254 bytes of commands. The number of commands that can be placed in the buffer depends on the format of each command.

Description	***Memory address offset***
Command Count	900h
Command Buffer	902 – 9FFh
Reply Count	A00h
Reply Buffer	A02 - AFFh

A command that has no parameter will occupy 2 bytes in the buffer. Any commands that have a 32 bit integer or a 32 bit floating point parameter will occupy 6 bytes in the buffer. Any commands with a 64 bit double precision floating point parameter will occupy 10 bytes. A command with a string parameter can occupy whatever space is remaining in the command buffer. The host computer is free to use whichever number format is best suited to the command. The string format is reserved for specific commands that require a string parameter (see the command descriptions). Each command in the buffer must be in one of the following formats:

Command with no parameter
Byte 1:Command code (listed in appendix A), if code is greater than 255, place 8 least significant bits in this byte and set extended command flag
Byte 2: Command flags & axis number*

Command with 32 bit integer parameter
Byte 1: Command code (listed in appendix A), if code is greater than 255, place 8 least significant bits in this byte and set extended command flag
Byte 2: Command flags & axis number*
Bytes 3-6: Command parameter (binary integer, LSB first)

Command with 32 bit floating point parameter
Byte 1: Command code (listed in appendix A), if code is greater than 255, place 8 least significant bits in this byte and set extended command flag
Byte 2: Command flags & axis number*
Bytes 3-6: Command parameter (binary floating point, IEEE-754 format)

Command with 64 bit double precision floating point parameter
Byte 1:Command code (listed in appendix A), if code is greater than 255, place 8 least significant bits in this byte and set extended command flag
Byte 2: Command flags & axis number*
Bytes 3-10: Command parameter (binary double precision floating point, IEEE-754 format)

Command with string parameter
Byte 1: Command code (listed in appendix A), if code is greater than 255, place 8 least significant bits in this byte and set extended command flag
Byte 2: Command flags & axis number*
Bytes 3-4: String size in bytes including terminating null character
Bytes 5-249: String text followed by null (0 dec.) character

* The command flags & axis number byte has the following format:
bits 0 - 3 specify the axis number, set these bits to 0 for all axes
bit 4 set if extended command (ie. code is greater than 255)
bits 5,6 & 7 specify the type of parameter the command has (see table below)

Parameter type bit coding:

bit 7	*bit 6*	*bit 5*	*Parameter type*
0	0	0	32 bit integer parameter
0	0	1	64 bit floating point parameter
0	1	0	32 bit floating point parameter
0	1	1	No parameter (default value = 0)
1	0	0	32 bit integer specifying the user register that contains the parameter
1	0	1	string
1	1	0	Reserved
1	1	1	No parameter, use value in register 0

The format of the replies in the buffer is dependent on the command that generates the reply. The majority of the reporting commands use the command parameter to select the format of the reply. For these commands, a parameter value of 0 results in a 32 bit integer reply. A value of 1, results in a 64 bit floating point reply. And a value of 2 results in a 32 bit floating point reply. Commands such as Tell Register, Tell Channel, and Tell Analog which use the command parameter to select the item, have an implicit reply format. These formats are specified in the command description. In either case, a reply will have one of the following formats:

Reply with 32 bit integer value
Byte 1:Command code (listed in appendix A), if code is greater than 255, 8 least significant bits will be in this byte and extended command flag will be set
Byte 2: Reply flags & axis number*
Bytes 3-6: Reply value (binary integer, LSB first)

Reply with 32 bit floating point value
Byte 1:Command code (listed in appendix A), if code is greater than 255, 8 least significant bits will be in this byte and extended command flag will be set
Byte 2: Reply flags & axis number*
Bytes 3-6: Reply value (binary floating point, IEEE-754 format)

Reply with 64 bit double precision floating point value
Byte 1:Command code (listed in appendix A), if code is greater than 255, 8 least significant bits will be in this byte and extended command flag will be set
Byte 2: Reply flags & axis number*
Bytes 3-10: Reply value (binary double precision floating point, IEEE-754 format)

Reply with string text
Byte 1:Command code (listed in appendix A), if code is greater than 255, 8 least significant bits will be in this byte and extended command flag will be set
Byte 2: Reply flags & axis number*
Bytes 3-4: String size in bytes including terminating null character
Bytes 5-249: String text followed by null (0 dec.) character

* The reply flags & axis number byte will have the following format:
bits 0 - 3 specify the axis number, these bits will be 0 for no axis
bit 4 set if extended command (ie. code is greater than 255)
bit 5,6 & 7 indicate the type of reply value (see table below)

Reply value type bit coding:

bit 7	*bit 6*	*bit 5*	*Parameter type*
0	0	0	32 bit integer parameter
0	0	1	64 bit floating point parameter
0	1	0	32 bit floating point parameter
0	1	1	Reserved
1	0	0	Reserved
1	0	1	string
1	1	0	Reserved
1	1	1	Reserved

5. If it is necessary for the PC to terminate command execution before the DCX has set the command count back to 0, write a binary FFh into the command counter.

6. In order to clear the interrupt line, the PC can issue the next group of commands or write a binary FFh to the command counter location.

The address offsets of the command and reply buffers in the DCX's dual port memory are as follows:

Description	*Memory address offset*
Command Count	900h
Command Buffer	902 – 9FFh
Reply Count	A00h
Reply Buffer	A02 - AFFh

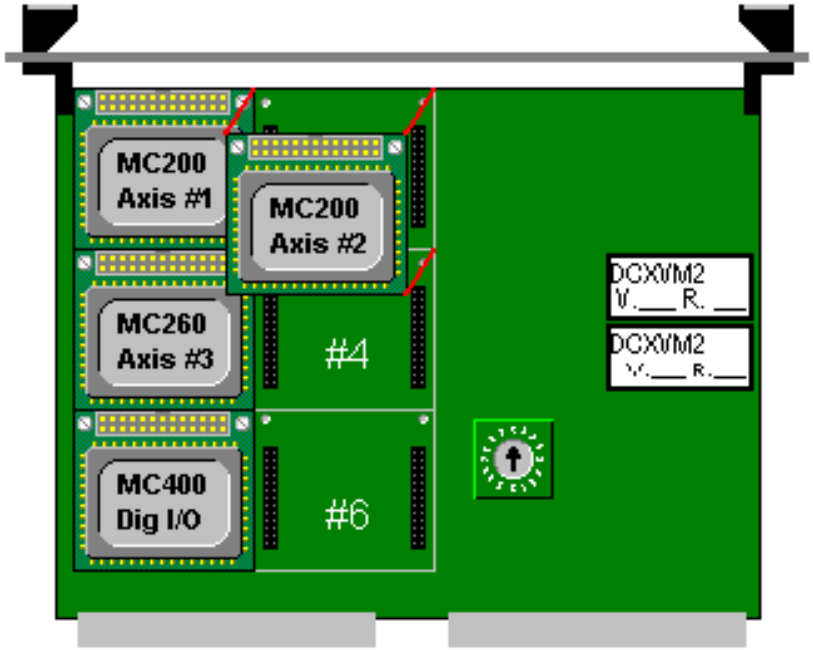
Installing DCX Motor Control and I/O Modules

DCX Modules can be placed in any open position on the DCX motherboard. If there are fewer than six modules to be installed on the DCX, spread them out as much as possible. This will allow easier installation and removal of the modules as well as mating cables.

If there are to be motor control modules installed on the DCX, and you want them to be numbered in a specific order, install them in module positions on the DCX in that order. For example, the module that is to control motor number 1 could be installed in module position number 1 (see numbers on DCX-VM200 motherboard). The module controlling motor number 2 could be installed in position number 2, and so on. Alternatively, the second module could be installed in any other module position and it will still be assigned number 2 since it is the second motor module on the DCX.

If a group of motors will be required to perform multi-axis contouring motion, one axis of the group should be assigned to axis 1. This will be the controlling axis for the group. Other groups of axes on the controller can also perform contouring motion, but will be more limited in the number of motion segments that can be stored on the board.

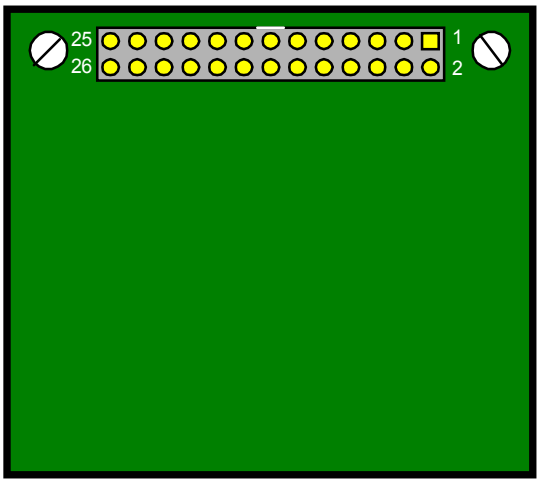
Once the position for each module has been determined, the modules can be installed on the DCX, one at a time. Lay the DCX on a flat surface in such a way that the numbers printed on the circuit board in each module position are right side up (the edge connector will be on the lower right corner of the board). Lay a module in its' target position on the DCX. The header pins of the module should be aligned with mating connectors on the DCX motherboard. If you look straight at the board, you should see that the standoffs in the upper corners of the module should be aligned with the mounting holes in the DCX. When you are satisfied that the module is properly aligned, carefully press the module into the DCX. The header pins of the module should seat completely into the mating connectors on the DCX motherboard. Two nylon mounting screws are supplied with each DCX module. These should be installed from the backside of the motherboard, into the standoffs on the modules. Repeat this process for installing modules on the DCX until all modules are in place.



Next the DCX should be installed in the system chassis and interfacing cables connected. Refer to the following sections in this chapter for specific information on the types of modules that are being used. When cabling has been completed, power can be applied to the system and initial checkout begun.

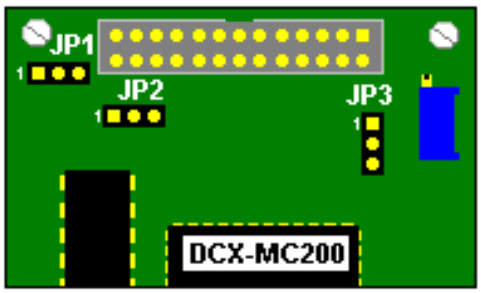
Please note that all DCX modules contain a 26 pin shrouded header for I/O. The pins of this connector are numbered from 1 to 26. The diagram below shows the location of pins 1, 2, 25 and 26. The other 22 pins are numbered and located respectively.

DCX MODULE CONNECTOR PIN NUMBERING (TOP SIDE VIEW)



DCX-MC200 – Servo Motor Module Installation

Installation of a DCX-MC200 Servo Motor Control Module includes setting three jumpers (JP1, JP2, and JP3). These jumpers are used to configure the module for the type of incremental encoder that will be used. These jumpers are configured by installing shorting blocks on the pins of the jumpers, or leaving them open Note that the pins of the three jumpers are numbered sequentially from 1 to 3, with pin 1 being shown as a square.



Jumper JP1 configures the module's encoder phase inputs for 'single ended' (A and B) or 'differential' (A+, A-, B+, and B-) signals. If a single ended outputs is used, install the 3 hole shorting block (supplied with the module) across all 3 pins of jumper JP1. Connect the A and B signals from the encoder to the A+ and B+ inputs of the module. If an encoder with differential phase outputs is to be connected to the module, jumper JP1 should be left open (no shorting block installed).

Jumper JP2 is used to configure the module's encoder index input. Either a single ended or differential index signal can be connected to the module. The table below lists the possible combinations.

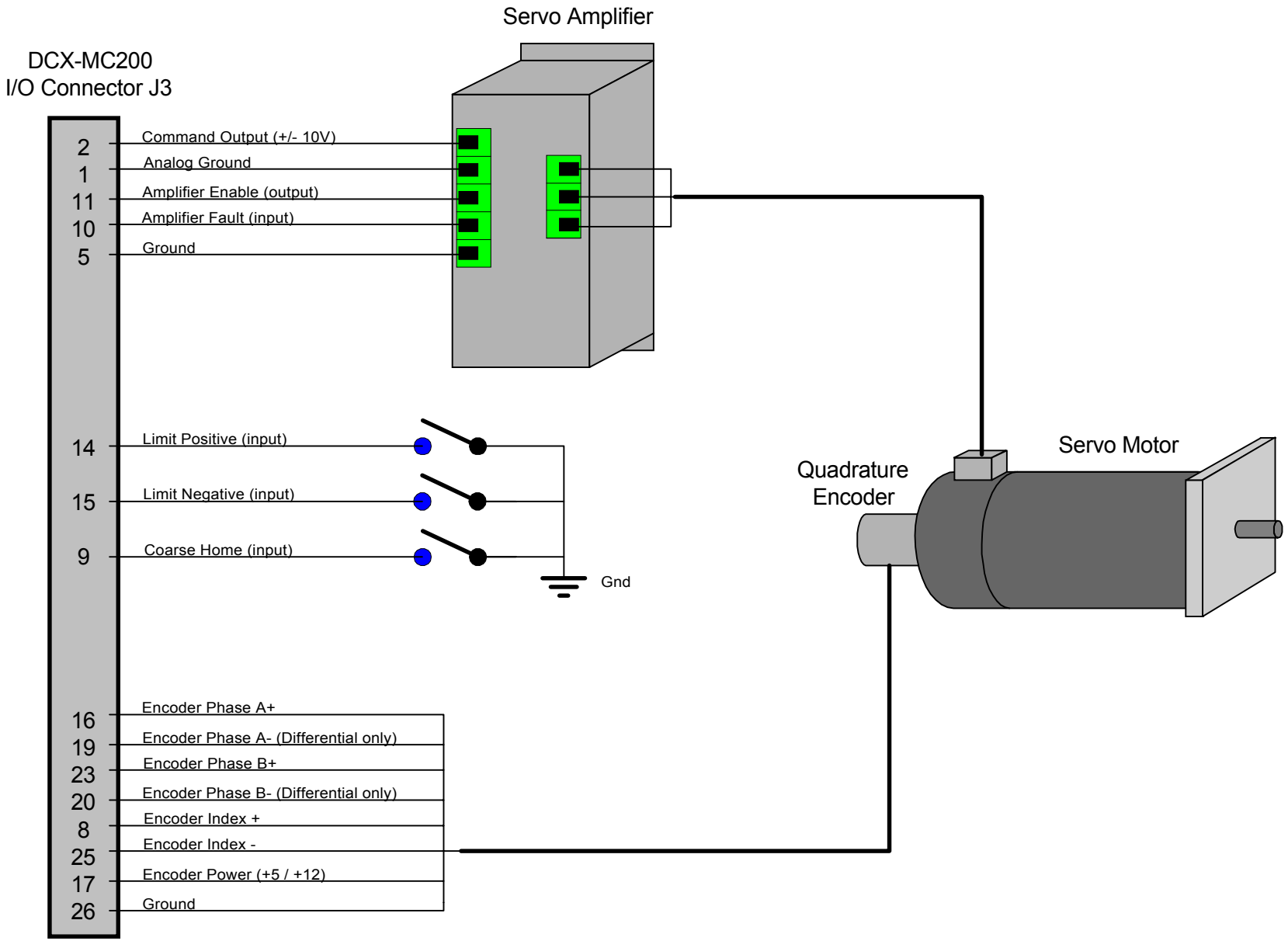
Table 1: MC200 Module Encoder Index Configuration

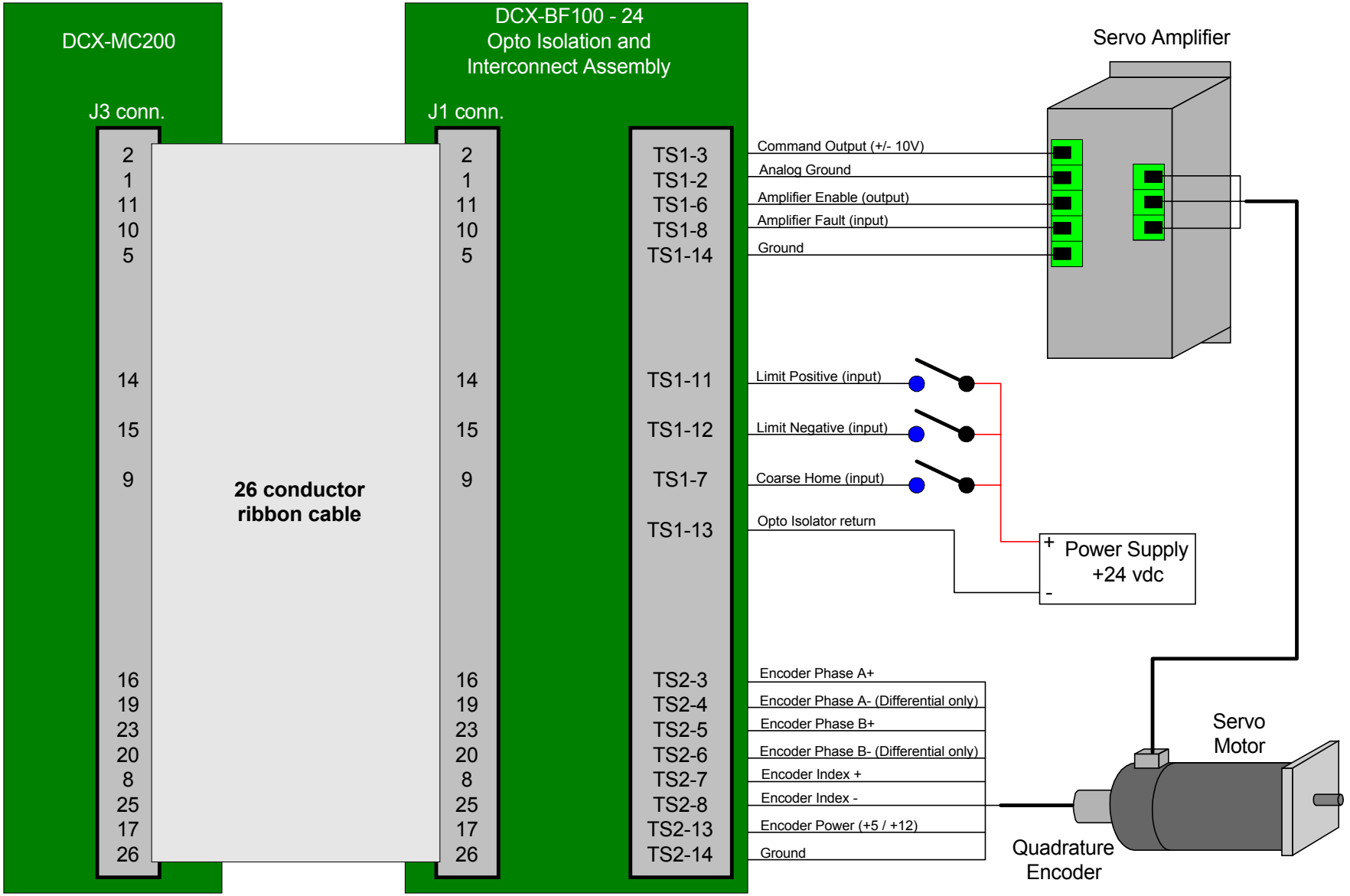
Signal Name	*Input Type*	*Active Level*	*Jumper JP2*	*J3 connections*
Index +	Single ended	High	1 to 2	Pin 8
Index -	Single ended	Low	2 to 3	Pin 25
Index + / -	Differential	N/A	None	Pins 8(+) & 25(-)

Jumper JP3 selects which of two supply voltages (from the PC computer power supply) will be available on the Encoder Power signal (connector J3 pin 17).
To select an encoder supply of +5 volts, connect pins 1 and 2 of jumper JP3 with a shorting block. To select +12 volts as the encoder power supply, connect pins 2 and 3 of JP3. The Encoder Power supply can provide up to 500ma of current.

Note: The DCX-MC200 provides the Encoder Power output as a convenience, It is **not required** that the Encoder Power supply output be used to power the encoder. If an external +5 volts or +12 volts supply is used to power the encoder, jumper JP3 **must still be configured** to match the voltage level (+5 volts or +12 volts) of the external supply.

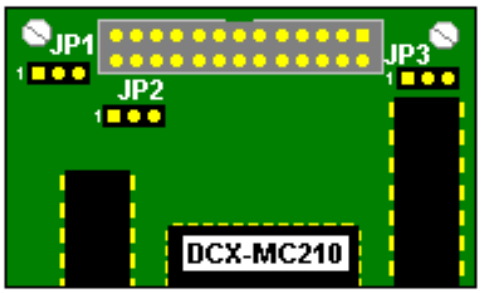
After configuring the jumpers of the module, the servo encoder, amplifier and limit switches can be connected to the module. Wiring diagrams on the next two pages depict typical installations. The first diagram details direct connection to the MC200. The second diagram details typical connections when the **DCX-BF100 Opto Isolation and Interconnect Assembly** is used.


Servo Amplifier
DCX-MC200
I/O Connector J3
2
1
11
10
5
Command Output (+/- 10V)
Analog Ground
Amplifier Enable (output)
Amplifier Fault (input)
Ground
14
15
9
Limit Positive (input)
Limit Negative (input)
Coarse Home (input)
Gnd
Servo Motor
Quadrature
Encoder
16
19
23
20
8
25
17
26
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


DCX-MC200
J3 conn.
2
1
11
10
5
14
15
9
16
19
23
20
8
25
17
26
26 conductor ribbon cable
DCX-BF100 - 24
Opto Isolation and
Interconnect Assembly
J1 conn.
2
1
11
10
5
14
15
9
16
19
23
20
8
25
17
26
TS1-3
TS1-2
TS1-6
TS1-8
TS1-14
TS1-11
TS1-12
TS1-7
TS1-13
TS2-3
TS2-4
TS2-5
TS2-6
TS2-7
TS2-8
TS2-13
TS2-14
Servo Amplifier
Command Output (+/- 10V)
Analog Ground
Amplifier Enable (output)
Amplifier Fault (input)
Ground
Limit Positive (input)
Limit Negative (input)
Coarse Home (input)
Opto Isolator return
+
Power Supply
+24 vdc
-
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
Servo
Motor
Quadrature
Encoder

DCX-MC210 – Servo Motor Module Installation

Installation of a DCX-MC210 Servo Motor Control Module includes setting three jumpers (JP1, JP2, and JP3). These jumpers are used to configure the module for the type of incremental encoder that will be used. These jumpers are configured by installing shorting blocks on the pins of the jumpers, or leaving them open Note that the pins of the three jumpers are numbered sequentially from 1 to 3, with pin 1 being shown as a square.



Jumper JP1 configures the module's encoder phase inputs for 'single ended' (A and B) or 'differential' (A+, A-, B+, and B-) signals. If a single ended encoder is to be used, install the 3 hole shorting block (supplied with the module) across all 3 pins of jumper JP1. Connect the A and B signals from the encoder to the A+ and B+ inputs of the module. If an encoder with differential phase outputs is to be connected to the module, jumper JP1 should be left open (no shorting block installed).

Jumper JP2 is used to configure the module's encoder index input. Either a single ended or differential index signal can be connected to the module. The table below lists the possible combinations.

Table 2: MC210 Module Encoder Index Configuration

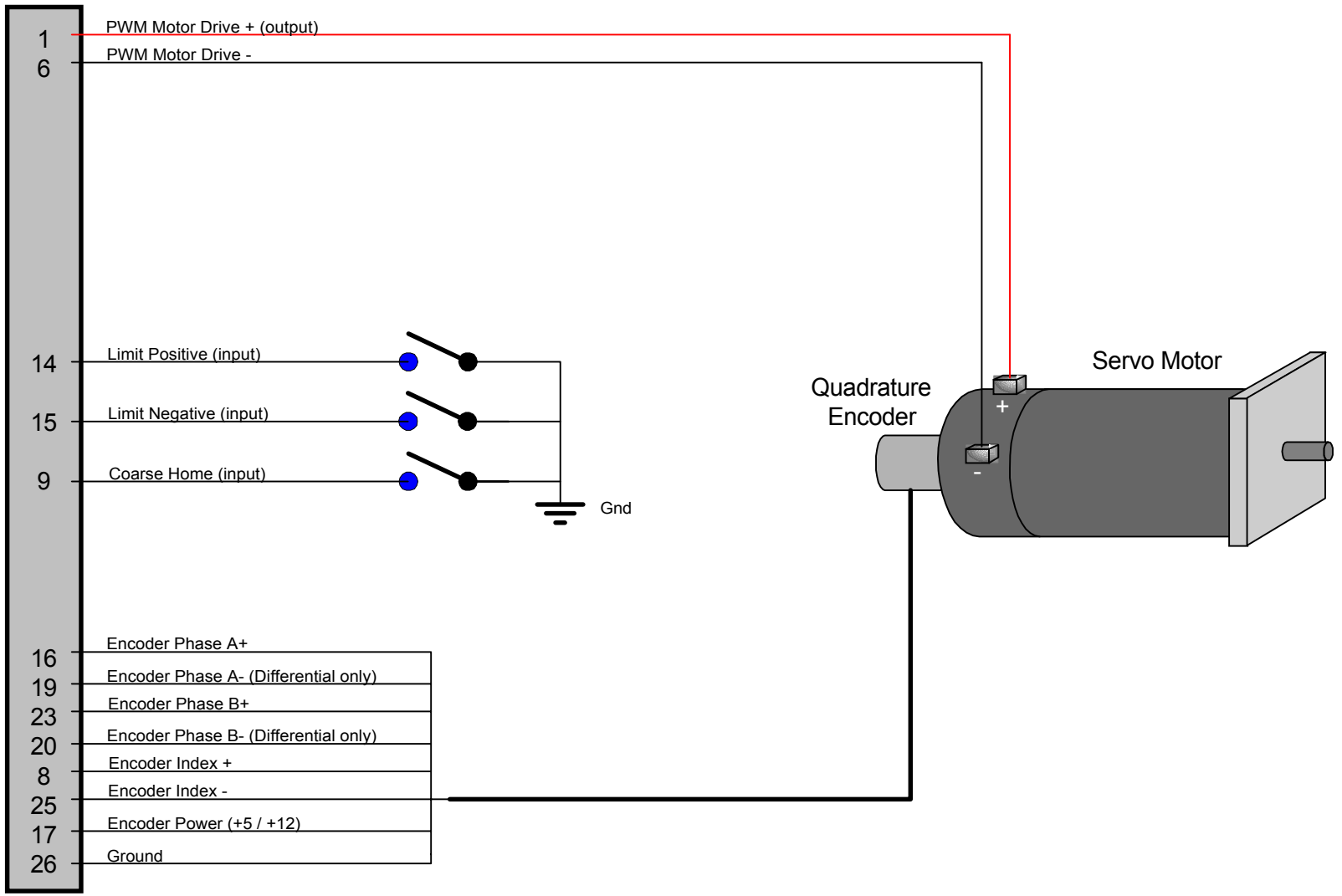
Signal Name	*Input Type*	*Active Level*	*Jumper JP2*	*J3 connections*
Index +	Single ended	High	1 to 2	Pin 8
Index -	Single ended	Low	2 to 3	Pin 25
Index + / -	Differential	N/A	None	Pins 8(+) & 25(-)

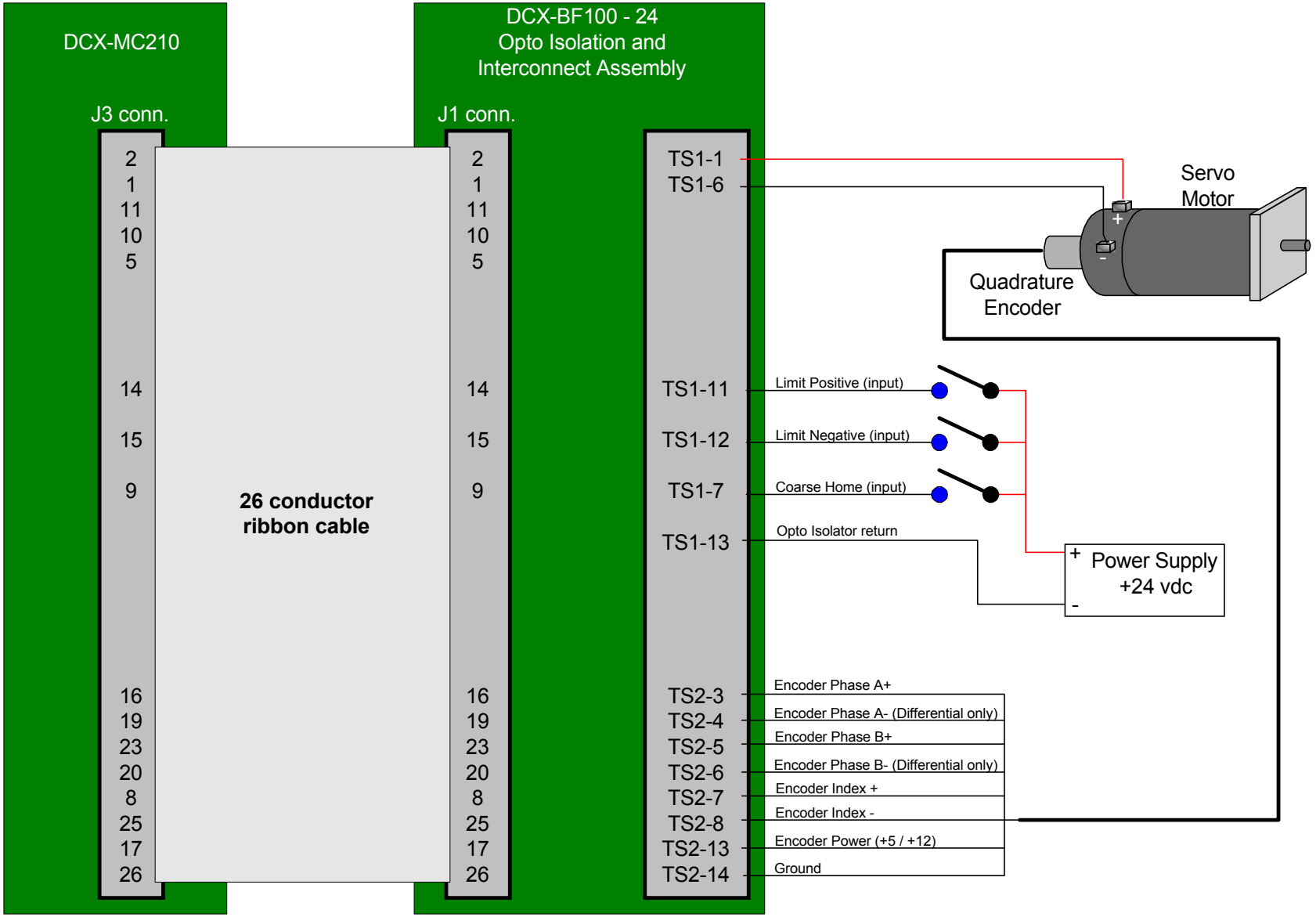
Jumper JP3 selects which of two supply voltages (from the PC computer power supply) will be available on the Encoder Power signal (connector J3 pin 17).
To select an encoder supply of +5 volts, connect pins 1 and 2 of jumper JP3 with a shorting block. To select +12 volts as the encoder power supply, connect pins 2 and 3 of JP3. The Encoder Power supply can provide up to 500ma of current.

Note: The DCX-MC210 provides the Encoder Power output as a convenience, It is **not required** that the Encoder Power supply output be used to power the encoder. If an external +5 volts or +12 volts supply is used to power the encoder, jumper JP3 **must still be configured** to match the voltage level (+5 volts or +12 volts) of the external supply.

After configuring the jumpers of the module, the servo encoder, motor and limit switches can be connected to the module. Wiring diagrams on the next two pages depict typical installations. The first diagram details direct connection to the MC210. The second diagram details typical connections when the **DCX-BF100 Opto Isolation and Interconnect Assembly** is used.

DCX-MC210
I/O Connector J3

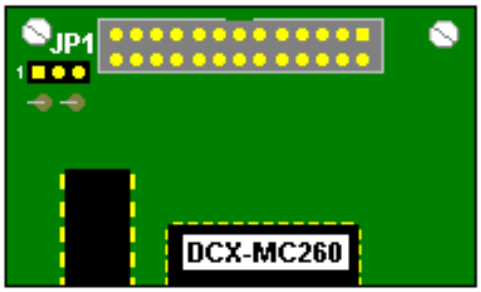



DCX-MC210
J3 conn.
2
1
11
10
5
14
15
9
16
19
23
20
8
25
17
26
26 conductor ribbon cable
DCX-BF100 - 24
Opto Isolation and Interconnect Assembly
J1 conn.
2
1
11
10
5
14
15
9
16
19
23
20
8
25
17
26
TS1-1
TS1-6
TS1-11
TS1-12
TS1-7
TS1-13
TS2-3
TS2-4
TS2-5
TS2-6
TS2-7
TS2-8
TS2-13
TS2-14
Servo Motor
+
-
Quadrature Encoder
Limit Positive (input)
Limit Negative (input)
Coarse Home (input)
Opto Isolator return
+
Power Supply
+24 vdc
-
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

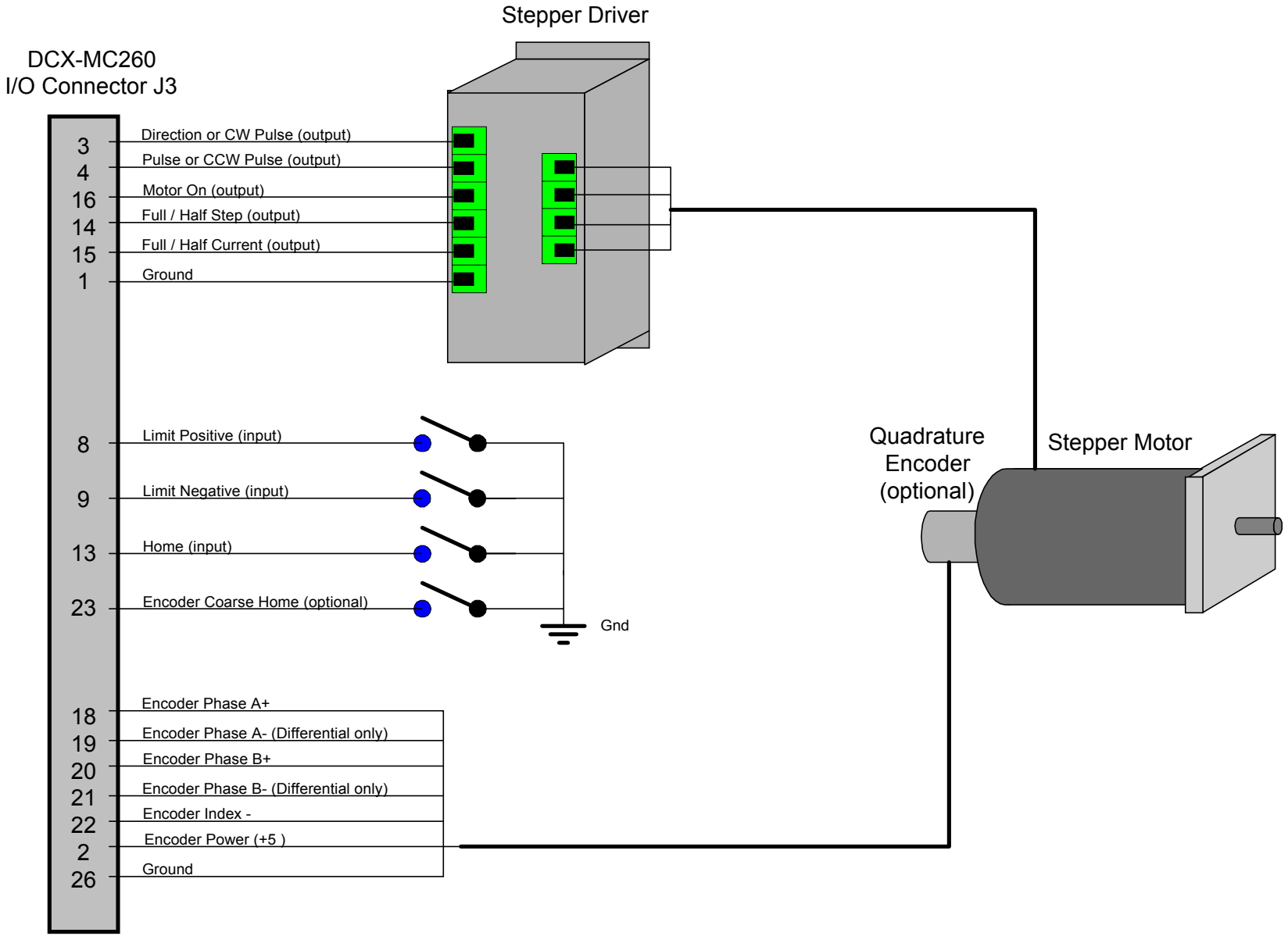
DCX-MC260 – Stepper Motor Module Installation

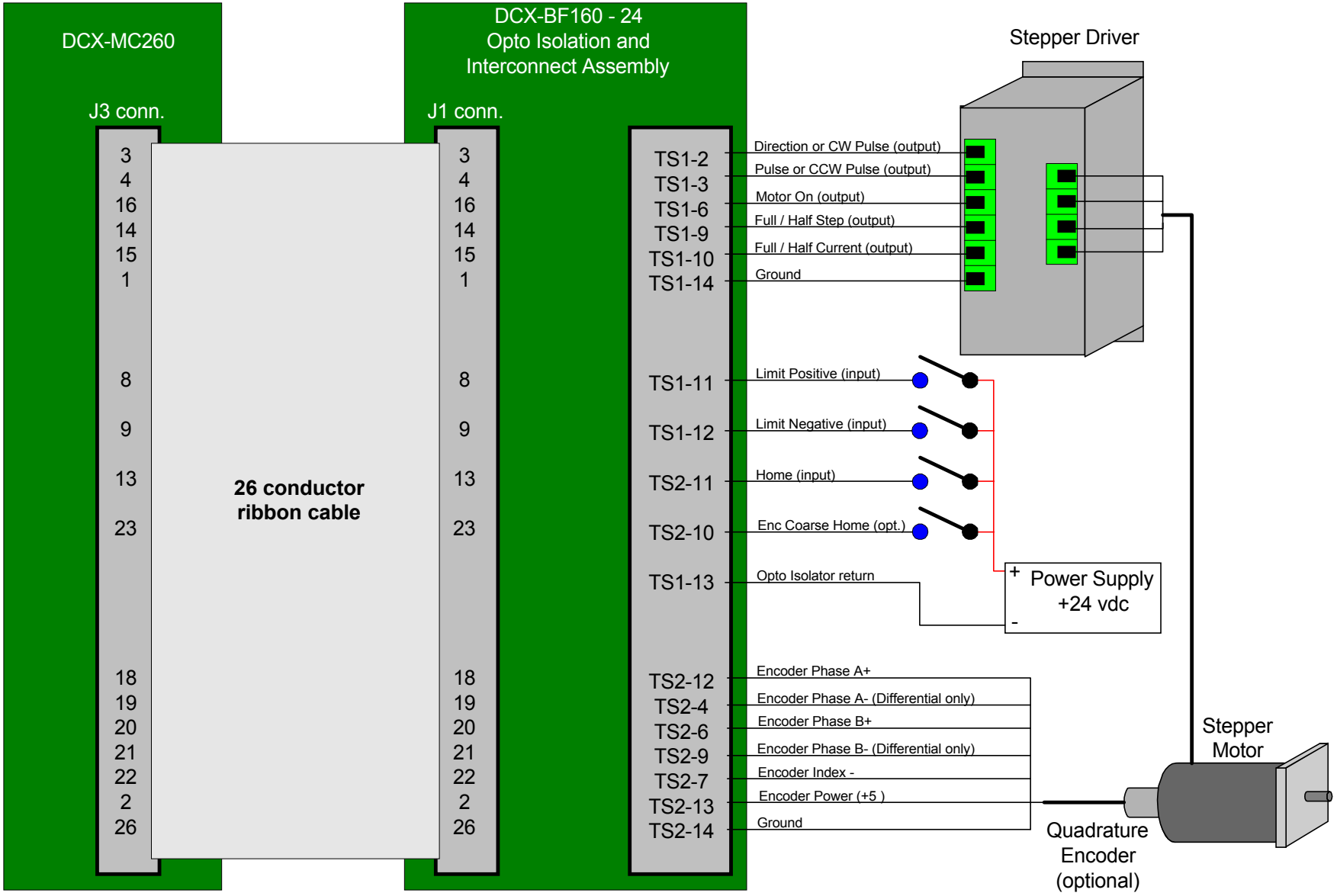
Installation of a DCX-MC260 Stepper Motor Control Module includes setting jumper JP1if an incremental encoder will be used for position feedback.

Jumper JP1 configures the module's encoder phase inputs for 'single ended' (A and B) or 'differential' (A+, A-, B+, and B-) signals. If an encoder with single ended outputs is to be used, a 3 hole shorting block (supplied with the module) should be installed across all 3 pins of jumper JP1. In this case, the A and B signals from the encoder should be connected to the A+ and B+ inputs of the module. If an encoder with differential phase outputs is to be connected to the module, jumper JP1 should be left open (no shorting block installed).



After configuring the jumper of the module, the stepper driver, limit switches and optional encoder can be connected to the module. Wiring diagrams on the next two pages depict typical installations. The first diagram details direct connection to the MC200. The second diagram details typical connections when the **DCX-BF160 Opto Isolation and Interconnect Assembly** is used.

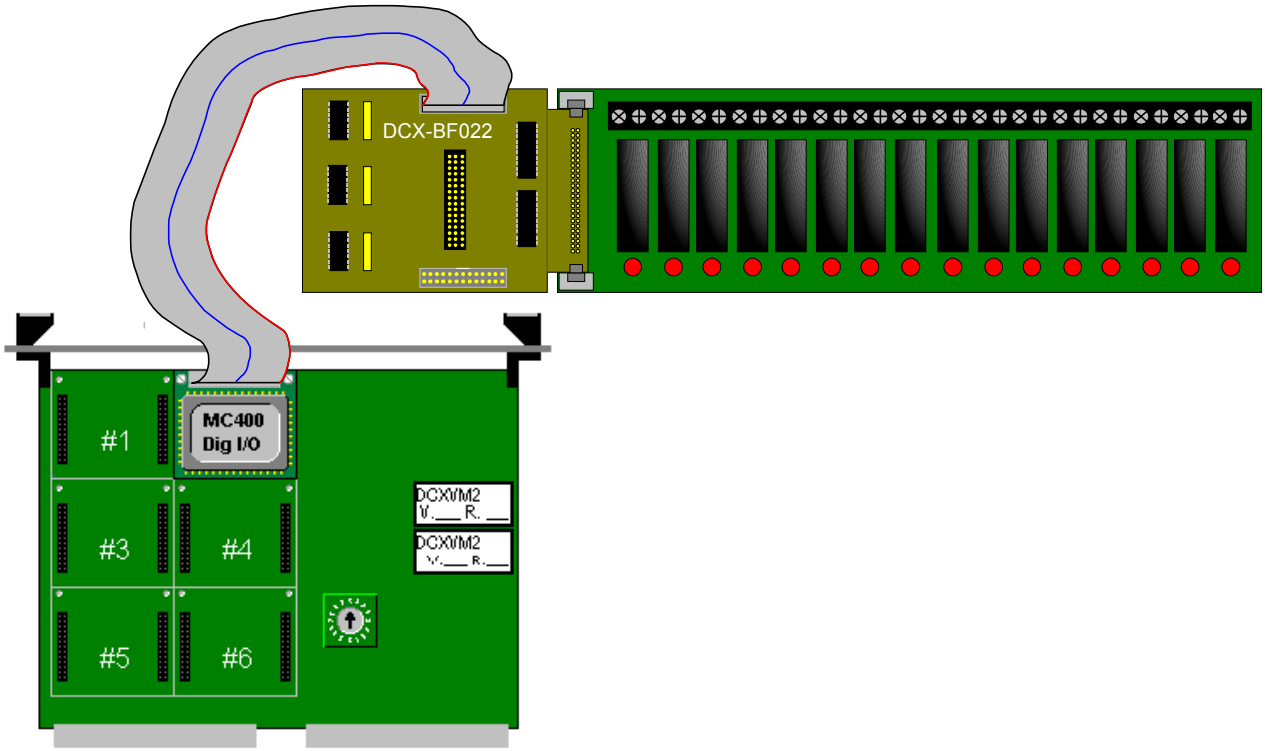

Stepper Driver
DCX-MC260
I/O Connector J3
3
4
16
14
15
1
Direction or CW Pulse (output)
Pulse or CCW Pulse (output)
Motor On (output)
Full / Half Step (output)
Full / Half Current (output)
Ground
8
9
13
23
Limit Positive (input)
Limit Negative (input)
Home (input)
Encoder Coarse Home (optional)
Gnd
Quadrature
Encoder
(optional)
Stepper Motor
18
19
20
21
22
2
26
Encoder Phase A+
Encoder Phase A- (Differential only)
Encoder Phase B+
Encoder Phase B- (Differential only)
Encoder Index -
Encoder Power (+5)
Ground


DCX-MC260
J3 conn.
3
4
16
14
15
1
8
9
13
23
18
19
20
21
22
2
26
26 conductor ribbon cable
DCX-BF160 - 24
Opto Isolation and
Interconnect Assembly
J1 conn.
3
4
16
14
15
1
8
9
13
23
18
19
20
21
22
2
26
TS1-2
TS1-3
TS1-6
TS1-9
TS1-10
TS1-14
TS1-11
TS1-12
TS2-11
TS2-10
TS1-13
TS2-12
TS2-4
TS2-6
TS2-9
TS2-7
TS2-13
TS2-14
Direction or CW Pulse (output)
Pulse or CCW Pulse (output)
Motor On (output)
Full / Half Step (output)
Full / Half Current (output)
Ground
Limit Positive (input)
Limit Negative (input)
Home (input)
Enc Coarse Home (opt.)
Opto Isolator return
Encoder Phase A+
Encoder Phase A- (Differential only)
Encoder Phase B+
Encoder Phase B- (Differential only)
Encoder Index -
Encoder Power (+5)
Ground
Stepper Driver
+
Power Supply
+24 vdc
-
Stepper
Motor
Quadrature
Encoder
(optional)

DCX-MC400 – Digital I/O Module Installation

One or more MC400 digital I/O modules can be installed on the DCX. There are no jumpers on this module to be configured. The module's TTL digital I/O signals can be connected directly to the external circuits if output loading (1ma maximum sink/source)and input voltages are within acceptable limits. Alternatively, a 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 BFO22 interface board provides a convenient means of connecting the MC400's 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 BFO22, an optically isolated I/O module can be connected to each of the MC400's digital I/O channels.



As shown above, the BFO22 plugs directly into the relay rack's 50 pin header connector and then connects to the MC400 via a 26 conductor ribbon. Note that the relays are numbered sequentially starting from 0, while the DCX digital I/O channels are numbered sequentially starting with 1.

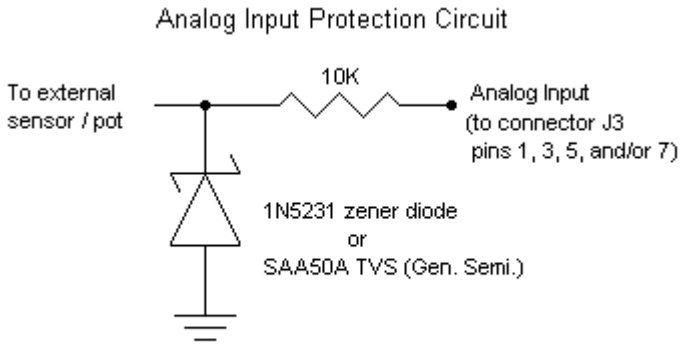
Although the relay rack has screw terminals for connecting a logic supply, it is not necessary to make this connection. By installing a shorting block on jumper JP17 of the BFO22, the 5 volt supply of the DCX will be supplied to the relay rack.

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.

DCX-MC500 – Analog I/O Module Installation

One or more MC500 analog I/O modules can be installed on the DCX as described in the first section of this chapter. There are no jumpers on this module to be configured. The module's I/O signals can be connected directly to the user's external circuits as long as output loading is not excessive and input voltages are maintained within the specified limits (see the MC500 appendix).

A voltage level greater than 5.6 volts will damage DCX-MC500 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.



DCX-MF300 – RS-232 Module Installation

A single MF300 RS-232 module can be installed in any module position on the DCX. There are several jumpers on the module that should be configured before connecting the module to an external device. These jumpers are configured by installing shorting blocks on the pins of the jumpers, or leaving them open. An appendix of this manual which covers the RS-232 module, includes a description of these jumpers and a diagram showing their locations. In addition, a portion of the module schematic is included in the appendix to help in configuring the module. Note that the pins of each jumper are numbered sequentially from 1, with pin 1 being shown as a square.

The RS-232 module connector J3 pin-out, is designed to allow a cable to be assembled from ribbon cable and Insulation Displacement Connectors (IDC). Such a cable, with a 25 pin D-subminiature connector on one end, can be directly connected to a terminal or host computer serial port. If the external device is considered to be Data Terminal Equipment (DTE), then the RS-232 module should be configured as Data Communications Equipment (DCE). This is the default configuration of the RS-232 module as shipped from the factory, and should have the jumpers set as follows:

Jumper number	Default setting
JP1	Pins 9 to 10
JP2	Pins 1 to 3, 2 to 4
JP3	Pins 1 to 2
JP4	Pins 1 to 2
JP5	Open
JP6	Pins 1 to 2
JP7	Open
JP8	Pins 1 to 2
JP9	Open
JP10	Pins 1 to 2
JP11	Open
JP12	Open
JP13	Open
JP14	Open

Jumper JP2 on the RS-232 module is used to select whether internal control signals are used for hardware handshaking, or for networking. When a single DCX is connected to the communicating device, this jumper should be configured for handshaking. This is done by installing a shorting block on pins 1 and 3, and another on pins 2 and 4 (shorting blocks will be perpendicular to 'JP2' label). For configuration of the module for use on a RS-232 network, a shorting block should be installed on pins 1 and 2, and another on pins 3 and 4.

DCX-MF310 – IEEE-488 Module Installation

A single MF-310 IEEE-488 module can be installed in any module position on the DCX. There are two jumpers on the module that should be configured before connecting the module to an external device. These jumpers are configured by installing a shorting block on the pins of the jumpers, or leaving them open. An appendix of this manual which covers the IEEE-488 module, includes a description of these jumpers and a diagram showing their locations.

The IEEE-488 module connector J3 pin-out, is designed to allow a cable to be assembled from ribbon cable and Insulation Displacement Connectors (IDC). Such a cable, with a IEEE-488 connector on one end, can be directly connected to a IEEE-488 controller.

Jumper JP1 should have a shorting block installed to provide open collector drivers on the module. Leaving the pins of jumper JP1 open will configure the module drivers as push-pull for high speed communications.

Installing a shorting block on jumper JP2, will connect the IEEE-488 connectors shield signal to the DCX's ground. Since the IEEE-488 controller should provide the grounding point for the cable shields, this jumper should be left open. The other jumpers on the IEEE-488 module are for production options, and should be left as shipped from the factory (JP3 hardwired, JP4 and JP5 open).

The DIP switch on IEEE-488 module is used to set the address of the DCX on the IEEE-488 bus. The appendix which covers the IEEE-488 module includes a list of all possible address settings. As an example, to set the DCX's address to 4 (talk address = 'D', listen address = '$'), switches 1, 2, 4, 5 and 6 should be in the off position, and switch 3 should be in the on position.

Chapter Contents

Chapter 3

Auxiliary Communication Interfaces

The DCX controller provides a high speed binary interface for communicating with the VME host. This interface is implemented using dual ported memory that is mapped into system memory using a jumper block (JP8) and an address switch. For a complete description of the binary communication interface please refer the **Communicating with the DCX Controller** section of the **Installation** chapter.

A VME bus ASCII communication interface is also provided. This communication port allows the user to quickly implement low level communication for system integration.

The DCX board supports two optional auxiliary communications interfaces. The optional RS-232 serial interface is supported by installing the DCX-MF300 RS-232 interface module available from PMC. For interfacing to the DCX board over the IEEE-488 Bus, the MF310 IEEE-488 module is available from PMC.

Commands sent to the DCX through any of the ASCII communication interfaces **must be followed by a carriage return (ASCII 13)**. A **linefeed (ASCII 10) is not required** at the end of command lines, and should not be sent.

RS-232 Communications Interface

The RS-232 interface has been designed to use either software or hardware handshaking for its operation. Hardware handshaking can be enabled or disabled by use of the HN and HF commands. Software handshaking can be enabled or disabled by use of the XN and XF commands.

Set the configuration of the RS-232 host to 8 data bits, one stop bit and no parity. This is the factory default configuration of the RS-232 interface. Also set the device to translate carriage returns received into carriage return and linefeed. The baud rate of the host should be set to match the baud rate of the RS-232 module (9600 baud is the default).

Note that input characters are only echoed through the RS-232 interface when enabled by the Echo oN (EN) command.

IEEE-488 Communications Interface

In order to send commands to the DCX via the IEEE-488 Bus interface it must first be addressed to listen. The command line is then sent in ASCII format, including the ending carriage return (no line feed). The DCX should then be unaddressed as a listener and addressed as a talker. If the DCX has a response to the command ready, it will send it at that time. Note that the DCX will not accept a new command line until the responses to the previous command have been accepted.

Chapter Contents

- Introduction to MCCL Commands
- Macro Commands
- Multi-Tasking

Chapter 4

DCX Operation

Introduction to MCCL commands

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

A command or sequence of commands, sent to the DCX via one of the communication interfaces (VME bus, RS-232, or IEEE-488), will be executed as soon as it is received. Alternatively, the command and/or sequence can be stored in the DCX's local memory as a ***macro***. A MCCL macro can be executed repeatedly based on user defined criteria. No matter where the command originates from, it will be executed by the DCX in the same way.

All MCCL commands are two character alphanumeric mnemonics built with two key characters from the description of the operation (eg. "MR" for Move Relative). When the command is received by the DCX it will be executed. An example DCX command description follows:

Move to relative position

MCCL command: aMR
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: CM, MA, PM

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

The operations described in this chapter assume that MCCL commands are being issued to the DCX with the VME bus ASCII command interface.

The following example shows the result of executing the VE command. This command causes the DCX to report firmware version and the amount of installed memory.

```
VE     <return>                          ;report the firmware version
DCX-VM200 Motion Controller              ;DCX reply
Hardware: 256K Dual Ported Ram, 1024K FLASH Memory
System Firmware Ver. PM1 Rev. 4.0a
Copyright (c) 1994-2000 Precision MicroControl Corporation
All Rights Reserved
```

Some MCCL commands will be preceded by an axis specifier, identifying to which axis the operation is intended. The following example will cause the DCX to report the position of axis one:

```
1tp    <return>                          ;report the position of axis #1
01     1500                              ;DCX reply
```

If an illegal character or an illegal series of valid characters are sent to the DCX the controller will reply with a question mark character, followed by an error code. The error code listing can be found in the **Appendix** at the end of this manual.

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 module is automatically set to the "motor off" state. In this state, there should be no drive current to the motors. For servos it is possible for a small offset voltage to be present. This is usually too small to cause any motion, but some systems have so little friction 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 module.

Before a motor can be controlled, certain parameters must be set by issuing commands to the DCX. These include , PID filter gains, maximum velocity, acceleration, deceleration, allowable following error, etc.. Depending on the type of motor being controlled (servo versus stepper), not all parameters will apply.

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

Assuming that the motor parameters have now been set, it can be change to the Motor oN (MN) state. All axes can be turned on with a single command, or a single axis can be enabled as desired. To enable all, enter the MN command without an axis number preceding it, or enter 0MN. To enable only axis 2, for example, enter 2MN.

After turning a particular axis on, it should hold steady at one position without moving. The Tell Target (TT) and Tell Position (TP) commands should report the same number. There are several commands which are used to cause motion, including the Move Absolute (MA), Move Relative (MR), Move to Point (MP), and Go Home (GH) commands. To move axis 2 by 1000 encoder counts:

```
2MR1000      <return>                    ;move axis #2 to 1000 encoder counts
```

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 its starting 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, pressing the return key only after the last command.

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

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

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

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

Entering the following command:

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

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

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

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

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

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

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

The position will be displayed continuously. This will repeat 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 the commands 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.

Macro Commands

A powerful feature of the DCX is the ability to define macro commands. This simply means giving a short name to a sequence of commands to form a new command defined by the user. For example:

```
1MR1000,WS0.25,MR-1000,WS0.25     <return>
```

may be entered as a command sequence. It causes the motor attached to axis 1 to move 1000 counts in the positive direction, waits one quarter second after it has reached the destination, then moves 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 Definition (MD) command as the first command in the command string. For example:

```
MD3,1MR1000,WS0.25,MR-1000,WS0.25        <return>
```

will define macro #3. Whenever it is desired to perform this motion sequence, just issue the command: MC3. To report all defined macro's issue the command Tell Macro (TMn) with parameter n = -1.

```
MD3,1MR1000,WS0.25,MR-1000,WS0.25<return>
TM-1   <return>                          ;report all stored macros
MC3,1MR1000,1WS0.25,1MR-1000,1WS0.25     ;DCX reply
```

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

The DCX can store up to 1100 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).

Macro numbers 0 through 9, and 256 through 1099 are stored in the on-board Flash memory. These macros will be retained when power to the board is turned off. However, once a macro that is stored in Flash memory is defined (MDn), it can't be redefined, only undefined (RM). Macro numbers 10 through 255 are stored in on-board RAM memory. These macros will not be retained when the power is off. The macros in RAM can be redefined or undefined as long as memory is available. The Reset Macros command can be used to erase the macros stored in RAM memory, in Flash memory, or in both, by using the appropriate command parameter.

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 Tell Macro command can be used to display the amount of Flash and RAM memory available for macros storage at any give time.

Another feature of the DCX is that macro 0 will be executed on power up or reset. A common use for this feature is to have the motors automatically configured when the board is powered up. To accomplish this: define a set of macros that contain commands to set the motor parameters. These macros should be located in flash memory so they are preserved when the power is turned off. Now define macro 0 to call these setup macros. Each time the DCX is powered up or reset, macro 0 will execute which will configure the motors. The following command example uses macro 0 to call macro's 300 (define PID parameters) and 301 (define trajectory parameters and turn on the motor).

```
MD0,MC300,MC301      <return>
MD300,1SQ.3,1SD1.1,1SI0.01,1IL5   <return>
1SV100000,1sa25000,1ds25000,1mn   <return>
```

Warning: If macro 0 contains a command sequence that runs indefinitely, the command interface will be busy while the macro is executing. This will cause the host **to be unable to communicate with the DCX**. If this happens and MCAPI is unable to communicate with the DCX:

1) Turn off power to the computer
2) Remove the DCX, set the memory offset rotary switch (SW1) to position F
3) Install the DCX in the VME system, turn on power for 10 seconds
4) Turn off power to the computer
5) Reset the memory offset rotary switch (SW1) to its original position
6) Reinstall the DCX into the computer, turn on the power
7) All macro's should have been deleted. Communication with DCX should be restored.

To terminate the execution of any macro, issue the ASCII Escape code.

Multi-Tasking

To start a macro that runs indefinitely without 'locking up' communication with the host, call the macro's with the Generate Task command instead of the Macro Call. This will allow the operations called by macro 0 to execute as a background task.

The DCX command interpreter is designed to accept commands from the host 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 DCX to execute continuous monitoring or control sequences while still communicating 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 Generate Task (GTn) 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.

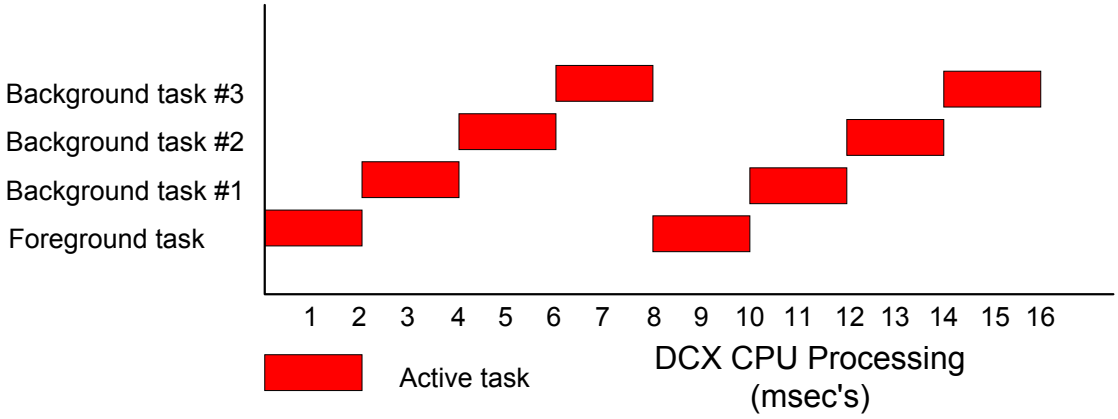
Note: Immediately after 'spawning' the background task (with the GTn command), the value in the accumulator (task identifier) should be stored in a user register. This value will be required to terminate execution of the background task.

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.

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 4. 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 2 msec's. 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 2 msec's of processor time every 8 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 BreaK command is executed. When a task is terminated, the resources it required are made available to run other background tasks. Alternatively, the Escape Task (ETn) command can be used to force a background task to terminate. 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.

The following example depicts a background task where the action of axis one will depend on the state of three digital inputs. If digital input #5 is on, axis one will move 1.5 inches in the positive direction and then dwell for 100 msec's. If digital input #6 is on, axis one will move 1.5 inches in the negative direction and then dwell for 100 msec's. In foreground, a command loop will monitor the state of digital input 2, if at anytime this channel goes true, the background task will be terminated.

```
MD100,IN5,MJ101,NO,IN6,MJ102,NO,JR-6     ;monitor digital inputs 5 & 6
MD101,1MR1.5,1WS.1,MJ100                 ;if channel 5 in on move relative
                                         ;+1.5"
MD101,1MR-1.5,1WS.1,MJ100                ;if channel 6in on move relative +1.5"
GT100                                    ;begin routine as a background task
1RL0, AR100                              ;store task ID in register 100

IN2,ET@100,no,JR-3                       ;terminate background task if channel
                                         ;2 is on
```

Chapter Contents

- Theory of DCX Motion Control
- DCX Servo Basics
- Tuning the Servo
- DCX Stepper Basics
- Moving Motors with PMC demo's
- Closed Loop Steppers
- Defining the Characteristics of a Move
- Velocity Profiles
- Point to Point Motion
- Constant Velocity Motion
- Contour Motion (arcs and lines)
- Electronic Gearing
- Jogging
- Defining Motion Limits
- Homing Axes
- Motion Complete Indicators
- On the Fly Changes
- Pause and Resume Motion
- Physical Assignment of Axes Numbers

Chapter 5

Motion Control

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

Theory of DCX Motion Control

The DCX motherboard (DCX-VM200) CPU is a Motorola 68020 which is programmed to perform motion control tasks. Specially designed servo or stepper motor control modules are installed on the motherboard to configure it for controlling from 1 to 6 servos or stepper motors. Each DCX motion control module (DCX-MC200, DCX-MC210, DCX-MC260) installed on the motherboard provides all the circuitry required to control one motor and its associated axis I/O (home, limits, amp/driver enable, fault, etc...).

The motherboard processor implements a trajectory generator (trapezoidal, S curve, and parabolic) which calculates the desired position and velocity of each servo or stepper motor at fixed time intervals. These values are sent to the respective servo (DCX-MC200 or DCX-MC210) or stepper module (DCX-MC260) installed on the motherboard. Each servo or stepper module has a 16 bit processor which is programmed to provide the appropriate control of the servo or stepper motor interfaced to the module.

Servo Motor Control

The servo modules uses a velocity feed-forward and a position feedback loop to control the servo. The **DCX-MC200** provides a 12 bit, +/-10 volt analog output signal to an external servo amplifier. The **DCX-MC210** provides a 23.4 KHz, 8 bit, PWM direct motor drive output capable of driving a 12 volt motor with up to 1A of current. An incremental encoder input to the module provides feedback information for closing the position loop. In operation, the servo module subtracts the actual position (feedback position) from the desired position (trajectory generator position), and the resulting position error is processed by the digital filter on the module. The output of the digital filter and the velocity feed-forward are combined to set the modules analog output level. The external amplifier uses this signal to drive the motor to the desired position.

The module processor monitors the motor's position via an incremental encoder. The two quadrature signals from the encoder are used to keep track of the absolute position of the motor. Each time a logic transition occurs at one of the quadrature inputs, the servo controller position counter is incremented or decremented accordingly. This provides four times the resolution over the number of lines provided by the encoder. The encoder inputs are buffered by a differential line receiver on the module. Configuration jumpers on the module allow this receiver to be configured for single ended input signals.

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

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

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

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

DCX Servo Basics

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

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

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

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

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

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

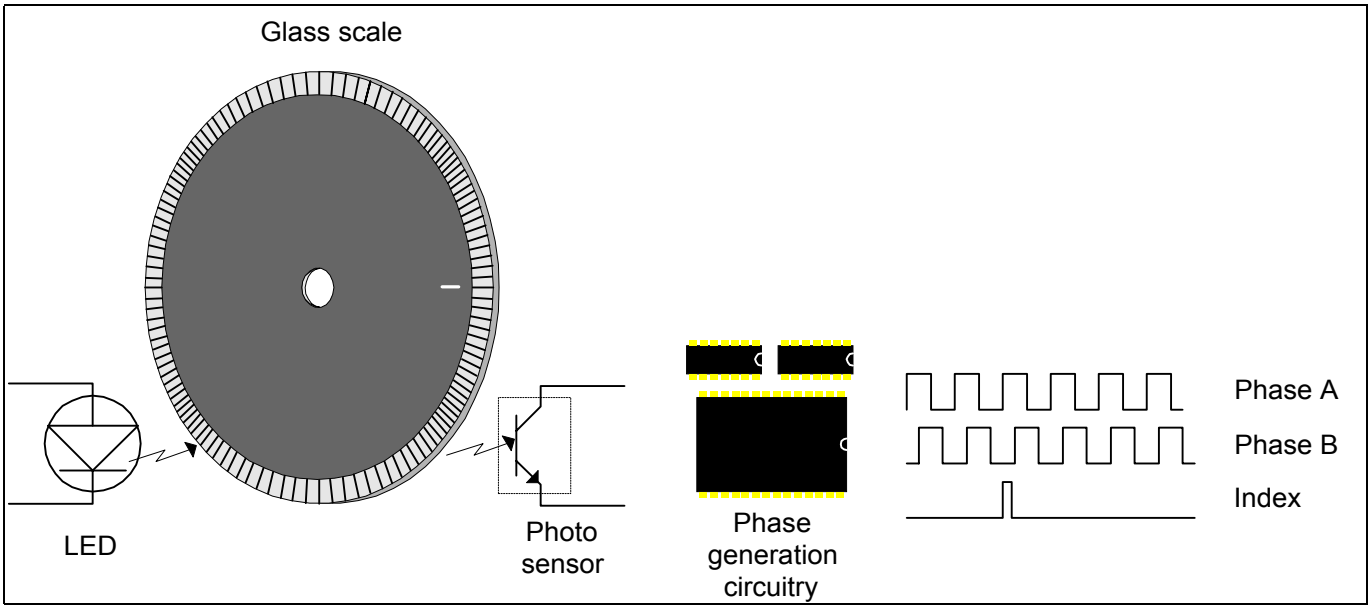
Some quadrature encoders include an additional 'mark' on the glass scale which is used to generate an index pulse. This signal, which 'goes active' once per rotation, is used by the motion controller to accurately home (re-define the position of an axis) the axis. Please refer to the **Homing Axes**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

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

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

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

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



Encoder Checkout
The DCX will report the position of the encoder each time the Tell Position (TP) command is issued. The position registers of the DCX will be reset to 0 when power is first applied or the DCX is reset. Manually rotate the motor / encoder in either direction, the position reported should increment or decrement accordingly. Refer to the Troubleshooting guide if the DCX does not report a change of position.

```
1TP    <return>                          ;report the position of the axis
01     0                                 ;DCX reply


;rotate the shaft of the motor / encoder
1TP    <return>
01     1327                              ;verify that the reported position has
                                         ;changed accordingly


;rotate the shaft of the motor / encoder the opposite direction
1TP    <return>
01     -967                              ;verify that the reported position has
                                         ;changed accordingly
```

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

While an axis is executing a move, the following error will typically be between 20 and 100 encoder counts. Very high performance systems can be 'tightly tuned' to maintain a following error within 5 to 10 encoder counts. Systems with low resolution encoders and/or high inertial loads will typically maintain a following error between 150 and 500 encoder counts during a move.

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

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

The DCX supports 'hard coded' following error checking. If at anytime the difference between the optimal position and the current position exceeds the user defined 'allowable following error' (set with the Stop on Error, aSEn) command, an error condition will be indicated. The axis will be disabled (Amplifier Enable output turned off, output command signal set to 0.0V) and the axis status word will indicate that an error has occurred. The Motor oN (aMN) command is used to clear a following error condition. To disable 'hard coded' following error checking set the allowable following error to zero (aSE0).

Selecting the Servo Loop Rate

The DCX support three ranges of servo loop rates:

Description	*MCCL command*	*Servo Loop Rate*
High Speed	HS	4 KHz ***(Integral term 'I' not used)***
Medium Speed	**MS**	**2 KHz (default)**
Low Speed	LS	1 KHz

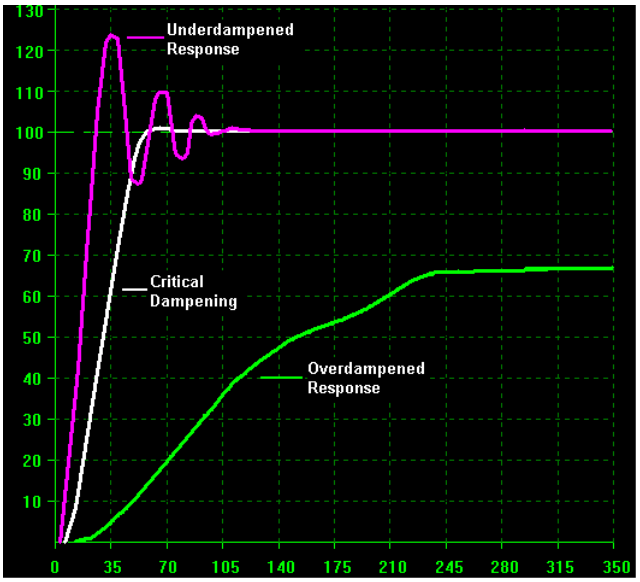
Tuning the Servo

A servo motor motion system is a closed loop system with negative feedback. Servo tuning is the process of adjusting the gains (proportional, derivative, and integral) of this axis controller to get the best possible performance from the system. The servo system is tuned by applying a command output or 'step response', plotting the resulting motion, then adjusting parameters of the digital PID filter until an acceptable system response is achieved.

A step response is an output command by the motion controller to a specific position, the magnitude of the output signal is determined only by the PID filter. The controller does not apply a maximum velocity or ramping (acceleration/deceleration). A typical step response would be 100 encoder counts. There is a 'loose' relationship between the step response and the following error of the system. The shorter the step response when tuning the servo, the lower the following error during application motion.

Note – Very short step response will very likely result in an unstable system that oscillates during and after a commanded move.

A servo motor and its load both have inertia, which the servo amplifier must accelerate and decelerate while attempting to follow a change in the input (from the motion controller). The presence of inertia will tend to result in over-correction, with the system oscillating or "ringing" beyond either side of its target (under-damped response). This ringing must be damped, but too much damping will cause the response to be sluggish (over-damped response). Proper balancing will result in an ideal or critically-damped system.



A closed loop servo system is 'tuned' by selecting settings for:

Proportional gain – Controls the responsiveness of a system
Derivative gain – A digital dampening factor
Integral gain – Position correction as a function of time

that meet the performance requirements of the application.

Setting the Phasing of the Axis

Axis Phasing - An axis is properly phased when a commanded move in the positive direction causes the encoder decode circuitry of the controller to increment the reported position of the axis. When an axis is reversed phased, the motor turns in the opposite direction of the commanded move. This will create an error condition, the DCX will turn the axis off (Amplifier Enable output and the digital PID filter will be disabled).

To verify the phasing of an axis execute the following command sequences:

```
1DH0                                ;define current position = 0
1SG1,1SD0,1SI0                      ;set proportional gain to 1, zero
                                    ;derivative and integral gain
1SE1024                             ;set following error = 1024
1MN                                 ;turn on axis (enable PID filter, turn
                                    ;on Amplifier Enable output)
1MR1000,WA1,1TP                     ;command move of +1000 encoder counts.
```

One of the following three results will occur:

1) If reported position is greater than position 100 the axis is properly phased, move to the section describing **Setting the Proportional Gain**.

2) If the reported position is between 99 and –99 the proportional gain is too low. Double the proportional gain (1SG2, 1SG4, ...) and repeat the move.

3) If the reported position is more negative than –100 the axis is reversed phased. Either:
   A) Reverse the connections of the encoder phase A & B (or A+, A-, B+, B-) to the DCX-MC200/210.
   B) or issue the Phase (*a*PH*n*) command with parameter *n* = 1(1PH1).

Setting the Proportional Gain

The first step in tuning the servo is to select a proportional gain setting that moves the axis to the target position. At this point the goal is not to find a proportional gain setting that perfectly positions the axis, just to get the axis to the target within a reasonable time frame.

From this point, until the axis has been tuned, all servo motion should occur with the Trajectory Generator disabled. This is accomplished by

changing the operating mode to Gain Mode (aGM). The DCX will no longer calculate a trajectory (ramping – velocity, acceleration, deceleration) for a commanded move. The only factor controlling the output of the controller to the servo system will be the digital PIG filter.

A typical step response distance used while tuning a servo is 100 encoder counts. If the system requires:

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

Then a step response of 50 encoder counts is recommended. If the servo system is moving a high inertial load then the step response should be increased to 200 – 300 encoder counts.

The following command sequences will select a proportional gain setting that positions the axis near the target position:

```
RM

AL0.0001,AR100              ;initialize the proportional gain register
AL100,AR110                 ;initialize the step response distance register
AL0,AR101                   ;initialize the derivative gain register
AL0,AR102                   ;initialize the integral gain register
AL0.0001,AR105              ;define proportional gain increment
AL0,AR111                   ;initialize the 'near target' register
AL0,AR112                   ;initialize the 'near zero' register



1SG@100,1SD@101,1SI@102
1DH0,1GM,1MN,1MA100,wa.3


MD100,1RL4,IG75,MJ101,BK,AL@100,AA@105,AR100,1SG@100,WA.3,1TP,RP
MD101,1MA-125,WA.2,1RL4,IB0,MJ105,BK,AL@100,AA@105,AR100,1SG@100,WA.3,1TP,JR-
10
MD105,1TG.5,WA2,MJ200



MD200,1MA0,1WS0.25,1DH0,1MN,MJ201
MD201,AL@110,AS2,AR111,MJ202
MD202,1MA@110,WA.05,1RL4,IB@111,MJ205,NO,MJ210
MD205,AL@100,AA@105,AR100,1SG@100,WA.2,1MA0,WA.3,MJ201
MD210,1MA0,WA.05,1RL4,IG3,MJ211,NO,MJ220
MD211,AL@100,AA@105,AR100,1SG@100,WA.2,1MA100,WA.3,MJ210
MD220,1MA100,WA.1,1TP,1MA0,WA.1,1TP,1TG.5


MC100                       ;start the 'coarse' adjustment of the proportional
                            ;gain setting
```

This MCCL program will select an initial setting for the proportional gain of the servo axis one. When this routine has completed execution the DCX will report:

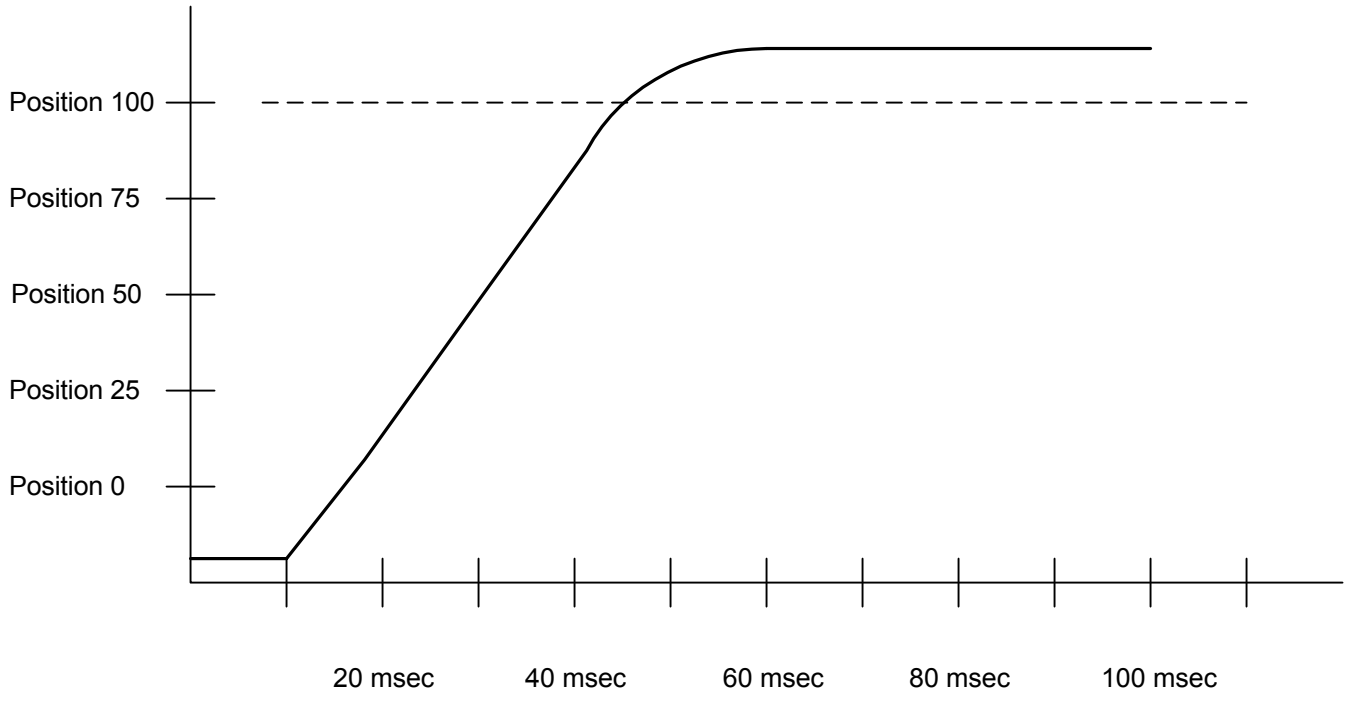
The actual position after a move to position 100
The actual position after a move to position 0
The selected value for the proportional gain

If this routine takes longer than 1 minute to execute, double the setting of user register 105 (proportional gain increment = 0.0001)

To plot the response of the axis use the Position Record (*a*PR*n*) command. Set the axis specifier *a* to the axis for which data is to be captured. Parameter *n* defines the quantity of points to be captured. The time interval between captured point is based on the current setting of the servo loop rate. If the servo loop rate is set to 2 KHz (default setting), the time between captured points will be 500 micro seconds. For additional information on the recording position data please refer to the **Record and Display Motion Data** section of the **Application Solutions** chapter. The following command example will issue a move to position 100, capture 150 data points, and then display the captured position data:

```
AL0,AR200                          ;initialize captured point register

1MA0,WA.5,1MA100,1PR150            ;move to position 100 and capture position
                                   ;points
1DR@200,AL@200,AA1,AR200,RP149     ;display captured position, increment point
                                   ;register, repeat 149 times
```

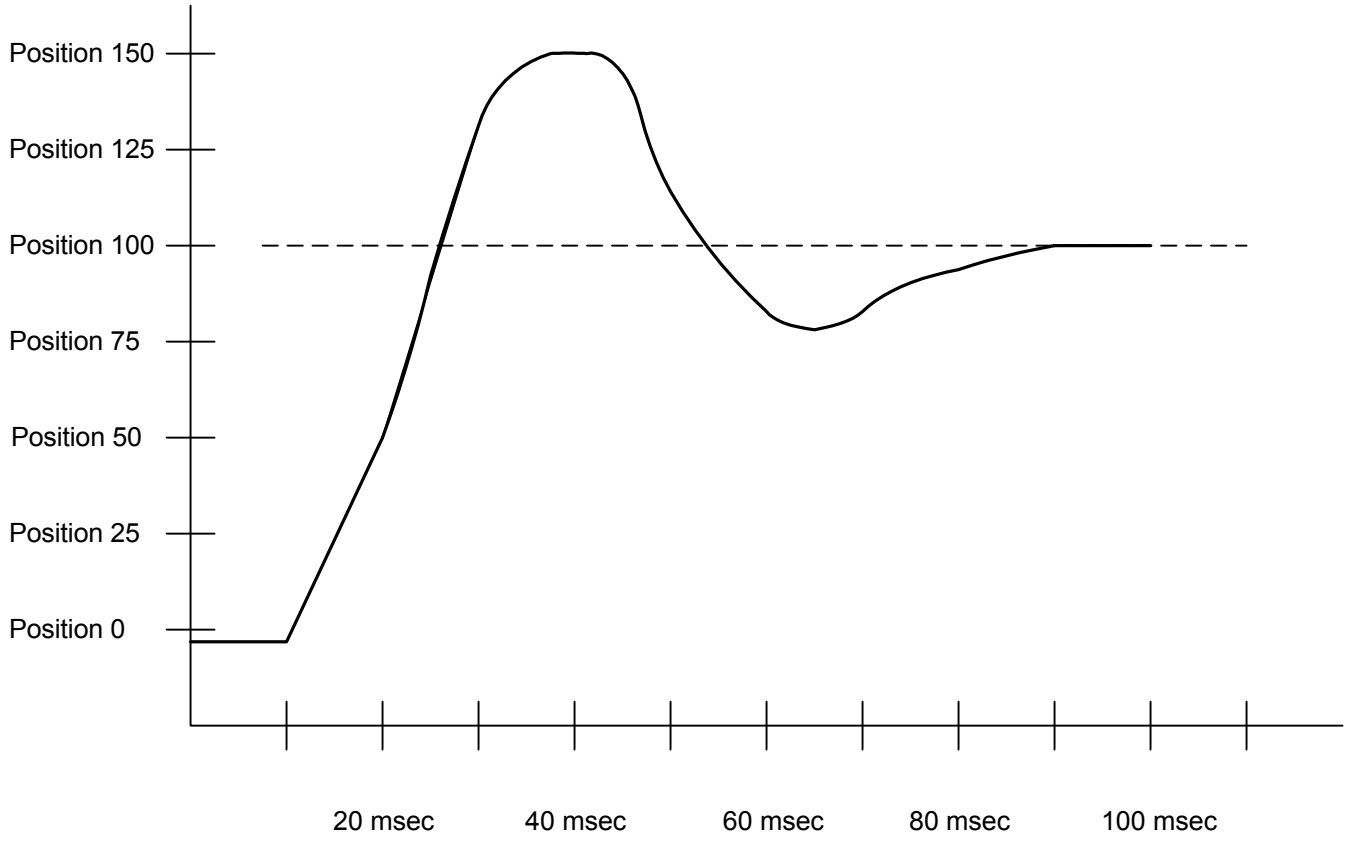
A typical plot of the captured data points would look similar to this:



The next step is to increase the proportional gain until the axis crosses the target three times. The MCCL command sequence that follows monitors the response of the system at fixed interval of time. Macro 301 will start a sub routine that increases the proportional gain until the axis crosses the target a second time. Macro 303 initiates a sub routine that increases the proportional gain until the axis crosses the target a third time.

```
MD300,1MA0,WA.1,1MA100,1WP102,MJ301
MD301,WA.01,1RL4,IG98,RP3,MJ302,MJ303
MD302,AL@100,AA.001,AR100,1SG@100,MJ300
MD303,WA.01,1RL4,IB100,RP4,MJ304,MJ305
MD304,AL@100,AA.001,AR100,1SG@100,MJ300
MD305,1TG.5

MC300                       ;start the 'fine adjustment of the proportional
                            ;gain setting
```

This MCCL routine will select an new setting for the proportional gain of the servo axis one. When this routine has completed execution the DCX will report proportional gain setting for the axis. A typical plot of the captured data points would look similar to this:



Setting the Derivative Gain

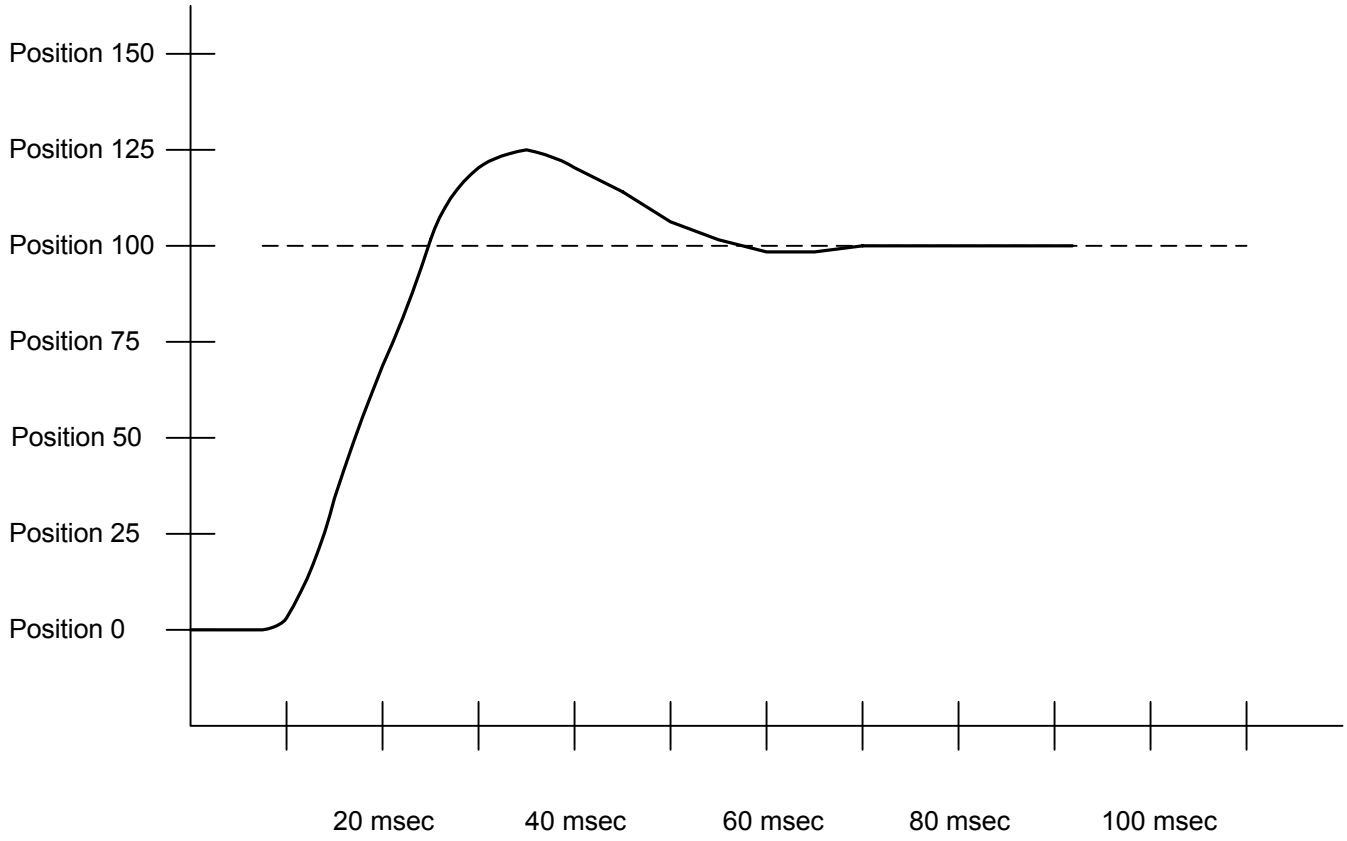
.Derivative gain is used to dampen the response of the servo. In the previous position plot the axis overshoots the target by 50 encoder counts. The next step is to reduce the overshoot of the system. The amount of acceptable overshoot will vary, for this example the overshoot will be limited to 25%.

Overshoot in Gain mode will not be present in Position and/or Velocity mode moves.

The following MCCL command routine will reduce the overshoot of the system to 25% of the target.

```
MD400,1MA0,WA.1,1MA100,MJ401
MD401,1RL4,IG125,MJ403,NO,RP200,MJ404
MD403,AL@101,AA.005,AR101,1SD@101,MJ400
MD404,1MA0,WA.1,1MA100,WA.2,1TD.5

MC400                       ;start the adjustment of the derivative gain
                            ;setting
```

This MCCL routine will select a setting for the derivative gain of the servo axis one. When this routine has completed execution the DCX will report derivative gain setting for the axis. A typical plot of the captured data points would look similar to this:

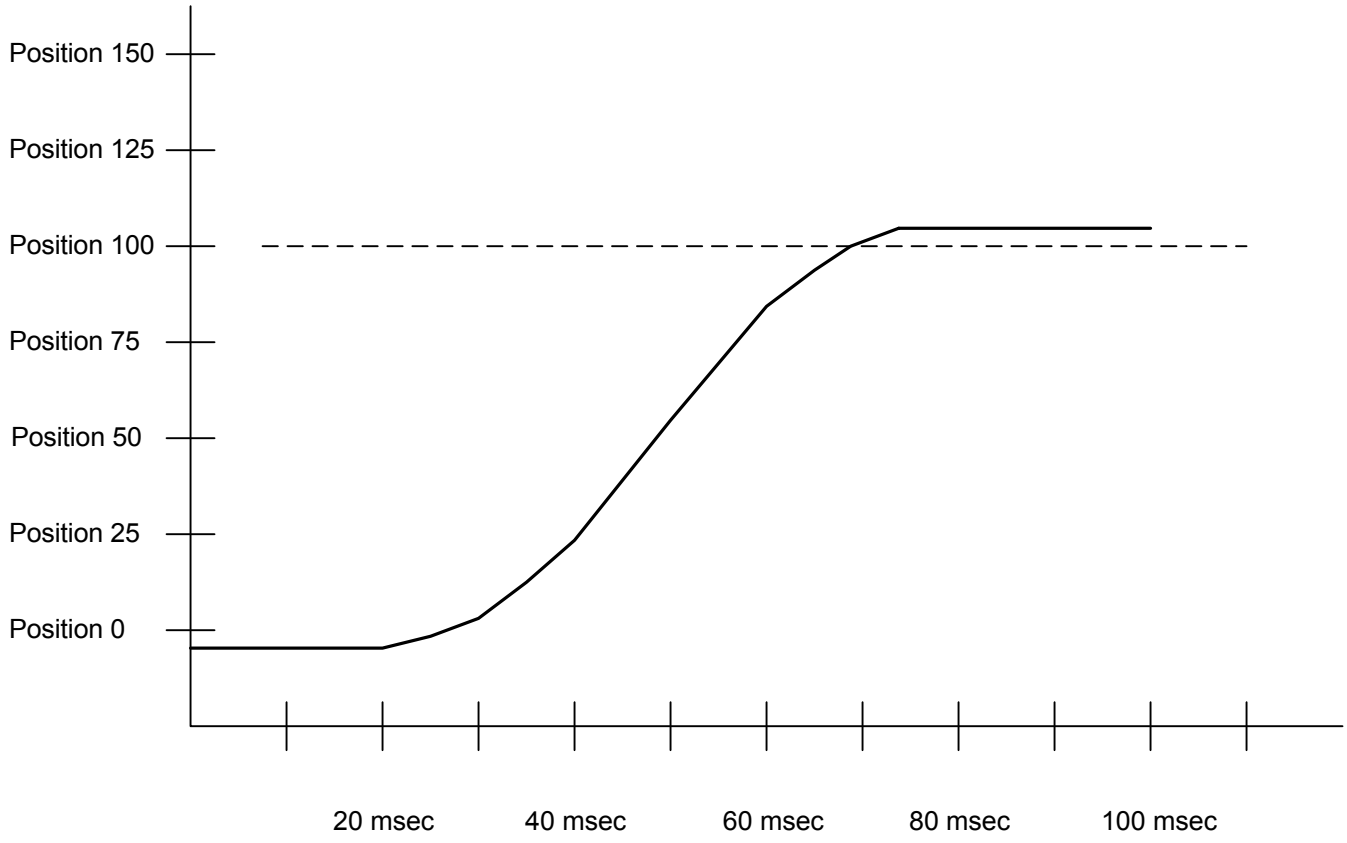


Setting the Integral Gain

.Due to friction, ‘sticktion’, amplifier offset, etc... some systems may have difficulty reaching and settling at the target. If the axis is not able to position and settle within +/- two encoder counts, Integral Gain will many times be able to ‘get the axis to the target’. Integral gain will appear to slowly ‘pull the axis’ to the target.

Integral gain is the only PID parameter that is selected with the trajectory generator enabled.

The previous plot showed a Gain mode step response with only proportional gain and derivative gain defined. The following position point plot displays the response of the **same axis** but with the **trajectory generator enabled** (velocity, acceleration and deceleration all set to 100,000).

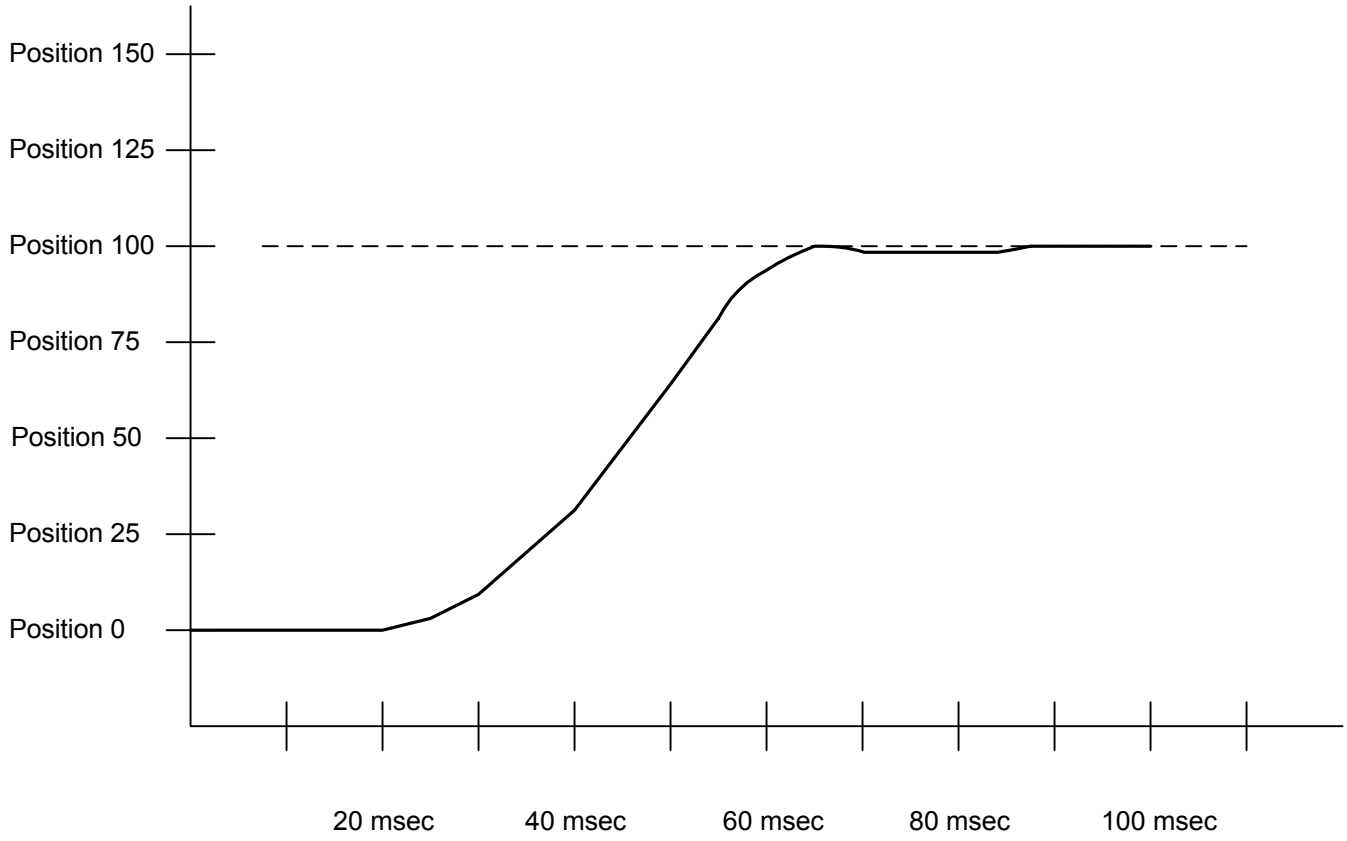


As show in the plot, the final position of the axis is 103, three counts from the target. Integral gain is used to command the axis to settle at the target position. The following MCCL command routine will increase the setting of the integral gain until the axis settles at the target position.

```
MD500,1PM,1SI@102,1MA0,1WS.1,1MA500,1WS.1,1TP,MJ501
MD501,1RL4,IG499,MJ503,NO,MJ502
MD502,AL@102,AA.001,AR102,MJ500
MD503,1RL4,IB501,MJ504,NO,MJ502
MD504,1SI@102,1MA0,1WS.1,1MA10000,1WS.1,1TP,MJ505
MD505,1RL4,IG9999,MJ507,NO,MJ506
MD506,AL@102,AA.001,AR102,MJ504
MD507,1RL4,IB10001,MJ508,NO,MJ502
MD508,VE,1TG.5,1TD.5,1TI.5,1TL.5,1tp

MC500                    ;start the macro sequence that will set the
                         ;integral gain
```

This MCCL command routine issues moves to two different position (1,500 and 10,000). These two targets represent the range of moves for the application. To change these target positions edit macros 500 (1MA500) and 504 (1MA10000). When this routine has completed execution the DCX will report

the integral gain setting for the axis. A typical plot of the captured data points would look similar to this:



For most systems servo tuning is completed by the setting of integral gain. The following sections address special cases that may need to be handled differently.

High Inertial Loads
In some high inertia applications., the default setting for the derivative sampling period (.0005 seconds) may be insufficient for achieving acceptable servo performance. If the derivative gain is to four to five times the proportional gain setting yet the axis is still **significantly** under dampened, increase the derivative sampling period (aFRn). A 'grinding' noise is another indication of a system that requires an increase in the derivative sampling period.

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

Systems of this type can be very are difficult to 'tune'. To over come the limitations of the system and get the axis moving, the proportional gain would need to be set very high. This will tend to make the system become unstable, causing the axis to 'oscillate'. The Output DeadBand (*a*OD*n*) command is used to compensate for the electrical and or mechanical dead band in a system. Parameter *n* of this command modifies the calculated output signal, allowing the module to simulate a 'frictionless' system. The value *n* will be added to a positive output and subtracted from a negative output.

The DCX Torque mode is used to test a system for deadband. This mode configures the analog output of a DCX-MC200 to simulate the operation of a programmable power supply, the user has direct control of the output DAC. To set the DAC output of a DCX-MC200:

```
1QM                                    ;enable torque mode
1SQ0                                   ;set output DAC to 0V
1MN                                    ;enable the axis
1SQ.5                                  ;output 0.5 volts
1SQ1.0                                 ;output 1.0 volts
1SQ1.5                                 ;output 1.5 volts
```

If the system deadband is 1.5 volts or greater it is **recommended that the Output Deadband be used**. If Output Deadband is to be defined, it must be done before tuning the PID parameters.

Programming an Output Offset

Both the MC200 and MC210 servo modules have output offset adjustment potentiometers for manually setting the zero point of the servo command output. The Output Offset command allows the user to enter a programmable output offset ranging from –10V to +10V (MC200) and 0 to 100% duty cycle (MC210).

Tuning Velocity Mode Amplifier Servo Systems

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

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

There are significant differences when tuning servo systems that close the velocity loop external to the DCX (position loop) controller. The PID filter becomes a secondary component in the generation

of the output signal that is applied to the velocity mode amplifier. The primary component that the DCX will use to generate the servo command signal is the Feed Forward term.

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

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

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

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

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

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

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

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

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

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

Tuning the Servo
After setting the velocity feed forward (velocity gain) as shown above:

1) Define the trajectory parameters (velocity, acceleration, and deceleration) to match the application requirements.

2) Zero the proportional, integral, and derivative gains.

```
1SG0,1SD0,1SI0
```

3) Turn the motor on

4) Execute a typical Position mode move. For this example the move distance is set to 2000 encoder counts. Repeatedly report the following error of the axis.

```
1MR2000
1TF,WA.01,RP29
```

5) Increase the proportional gain (aSGn) until the following error (reported by the TF command) does not increase during the move. Further increasing proportional gain will reduce the following error but may also result in the axis becoming unstable (oscillating at the end of a move.

6) Use integral gain (aSIn) to repeatedly position at the target (+/- 1 encoder count. Execute a typical Position mode move along with Wait for Stop and Tell Position commands.

```
1MR2000,1WS.1,1TP
```

Increase the integral setting until the reported position = the target +/- 1 encoder count. If the axis becomes unstable (oscillates):

1) Reduce the Integral Limit setting (aILn)
2) Reduce the proportional gain (aSGn)

DCX Stepper Basics

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

Open Loop Stepper Motion
Most stepper applications do not require feedback to accurately position an axis. This method of control is known as 'Open Loop'. To successfully complete the commanded move, the DCX controller counts each step pulse issued to the stepper motor driver. When the position of an axis is queried by issuing the Tell Position (TP) command, the number of pulses issued to the stepper driver is reported. Since there is no position (or velocity) feedback there is no need to 'tune' the of the axis. However, there are a few steps that must be performed in order to setup a stepper motor on the DCX.

The first step is to use the Output Mode (*a*OM*n*) command to configure the DCX-MC260 modules output signals for compatibility with the external stepper driver. The module defaults to defining the stepper control output signals as Step (J3 pin 4) and Direction (J3 pin 3). If the stepper driver requires Clockwise / Counter Clockwise output signals, the DCX module output signal mode can be changed by issuing the Output Mode command with parameter *n* set to 1.

```
OM1
```

The second step is to select the appropriate speed range for the axis. This step is necessary to provide high resolution of velocity increments. The three speed ranges are:

Description	*MCCL command*	*Step Pulse Output Range*
High Speed	*a*HS	1 Thousand to 1 Million steps per second
Medium Speed	*a*MS	125 to 156 Thousand steps per second
Low Speed	*a*LS	15 to 19.5 Thousand steps per second

Closed Loop Steppers

The advancements in stepper motor/micro stepping driver technology have allowed many machine builders to maintain ‘servo like’ performance while reducing costs by moving to closed loop stepper systems. While closed loop steppers are still be susceptible to ‘stalling’, they are not plagued with the familiar open loop stepper system problem of loosing steps due to 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 addressing the closed loop operation, the stepper axis must be installed and configured as described in the previous section (DCX Stepper Basics).

Connect the stepper's encoder to the module, the DCX should report the position of the encoder each time the Auxiliary encoder Tell position (AT) command is issued. If the stepper/encoder is manually turned in either direction, the position reported should increment or decrement accordingly.

```
1AT                                   ;report the aux. encoder position
       01     0                       ;DCX reply

1AT                                   ;move the axis (increment the position
                                      of the aux. encoder)
       01     915                     ;DCX reply

1AT                                   ;move the axis (decrement the position
                                      of the aux. encoder)
       01     -636                    ;DCX reply
```

To enable closed loop stepper motion issue the Input Mode (aIMn) command with parameter *n* equal to 1. Upon entering this mode of operation both the Tell Position (aTP) and the Auxiliary encoder Tell position (aAT) commands will report the number of encoder counts captured since the axis was last homed.

Configuring an axis for closed loop stepper motion requires the user to define the velocity gain of the axis. The algorithm for calculating the velocity gain of a closed loop stepper axis:

$$\text{Closed loop Velocity Gain} = \text{MC260 Calibration Constant X } \frac{\text{Step motor pulses per rotation}}{\text{Encoder counts per rotation}}$$

The MC260 calibration constant is a value generated by self test diagnostics upon power up/reset. This calibration value is defined as the default velocity gain of the axis and can be retrieved using the Tell Konstant (aTKp) command.

Closed loop stepper example
An incremental encoder is coupled to the shaft of a stepper motor. The encoder has 4000 counts per rotation (1000 lines). The stepper motor is connected to a micro stepping driver configured for 50,000 steps per motor rotation. The required maximum step rate of the axis is 625,000 or (750 RPM). This step rate requires the axis to be configured for High Speed (aHS) mode. After the step rate range is defined, the calibration constant is retrieved using the Tell Konstant (aTKn) command:

```
1HS
1TK.3                          ;report the calibration constant of
                               ;axis #1
```

the controller responds with the value:

```
01          0.746
```

The Velocity Gain is calculated as follows:

$$\text{Closed loop Velocity Gain} = \text{MC260 Calibration Constant X } \frac{\text{Step motor pulses per rotation}}{\text{Encoder counts per rotation}}$$

$$\text{Closed loop Velocity Gain} = 0.746 \text{ X } \frac{50{,}000}{4{,}000}$$

$$\text{Closed loop Velocity Gain} = 0.746 \text{ X } 12.5$$

$$\text{Closed loop Velocity Gain} = 9.325$$

```
1VG9.325
```

For some closed loop stepper applications, setting the velocity gain of the axis is the only 'tuning' the axis will require. Most applications will require that the proportional gain be used to:

Minimize the following error while moving
Eliminate slow speed slewing of the axis near the end of the move.

The following MCCL command sequences is used to determine the correct setting for proportional gain (1SGn).

```
1SG1                                ;set the proportional gain to a
                                    ;minimum value
1MR100000                           ;execute a 2 second move
WA.5,1TF,WA.25,1TF,WA.25,1TF,WA.25,1TF  ;report the following error
```

```
01    -62                           ;reported following error @ time .5 sec
01    -195                          ;reported following error @ time .75 sec
01    -369                          ;reported following error @ time 1.0 sec
01    -542                          ;reported following error @ time 1.25 sec
```

increase the proportional gain until the following error value stops increasing

```
1SG10                               ;increase proportional gain to 10
```

```
01    -57                           ;reported following error @ time .5 sec
01    -168                          ;reported following error @ time .75 sec
01    -287                          ;reported following error @ time 1.0 sec
01    -375                          ;reported following error @ time 1.25 sec
```

```
1SG50                                   ;increase proportional gain to 50

01    -39                               ;reported following error @ time .5 sec
01    -86                               ;reported following error @ time .75 sec
01    -117                              ;reported following error @ time 1.0 sec
01    -124                              ;reported following error @ time 1.25 sec


1SG100                                  ;increase proportional gain to 50

01    -24                               ;reported following error @ time .5 sec
01    -55                               ;reported following error @ time .75 sec
01    -62                               ;reported following error @ time 1.0 sec
01    -62                               ;reported following error @ time 1.25 sec
```

A proportional gain setting of 100 will result in acceptable stepper performance. The following command sequence will configure and operate stepper axis number one in closed loop mode:

```
1HS                                     ;Select Stepper speed range
1TK.3                                   ;Report the current calibrated
                                        ;velocity gain
01  0.746                               ;Controller will report the MC260
                                        ;calibrated gain
1VG9.25                                 ;Set the velocity gain (1VG9.25)
                                        ;Multiply the calibrated gain by the
                                        ;ratio of step pulses per rotation to
                                        ;encoder counts per rotation.

1IM1                                    ;Enable closed loop stepper mode
1AH                                     ;Zero the position of the encoder
1MN                                     ;Turn on the axis. The target and optimal
                                        ;positions will be initialized to the current
                                        ;position of the encoder.
1SV75000,1SA100000,1DS100000            ;Set maximum velocity, acceleration, and
                                        ;deceleration in units of encoder counts
1MV200                                  ;Set the minimum velocity so the motor
                                        ;reaches the target position with some
                                        ;non-zero velocity. Typically between 5% -
                                        ;10% of the maximum velocity.
                                        ;Warning: The minimum velocity must be less
                                        ;than the maximum velocity.
1SE500                                  ;Set the allowable maximum following error
                                        ;in encoder counts.
1SG100,1SD0,1SI0                        ;Set the proportional gain, derivative gain,
                                        ;integral gain, and integration limit for
                                        ;best motor speed stability.
1MR200000                               ;Issue moves in units of encoder counts.
                                        ;Watch the axis error LED's on the DCX
                                        ;motherboard for indication a motor
                                        ;error which can be caused by the axis
                                        ;exceeding the maximum following error
                                        ;or being reversed phased.
```

Be aware that if the 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 command (default = 1024) a motor error will occur and the axis will stop. If this happens, the phasing can be changed by issuing the PHase (aPHn) command to the

axis with a parameter of 1, or reverse the encoder inputs or the motor leads. If the motor is properly phased, it should resist movement away from its current position.

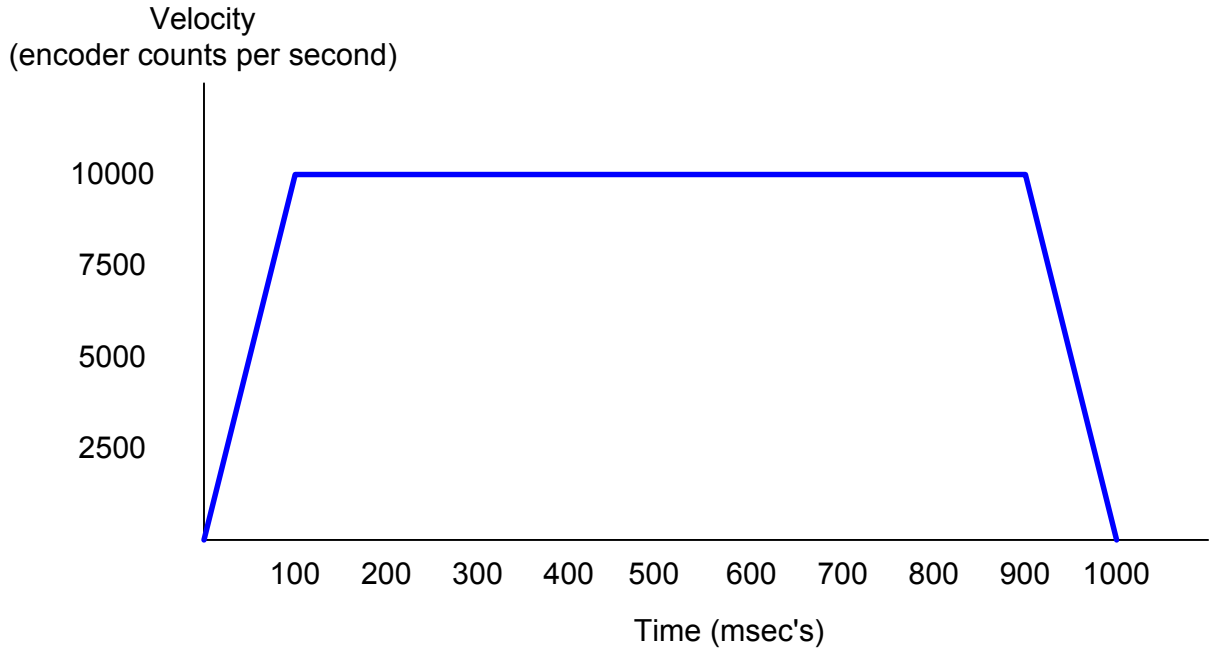
Defining the Characteristics of a Move

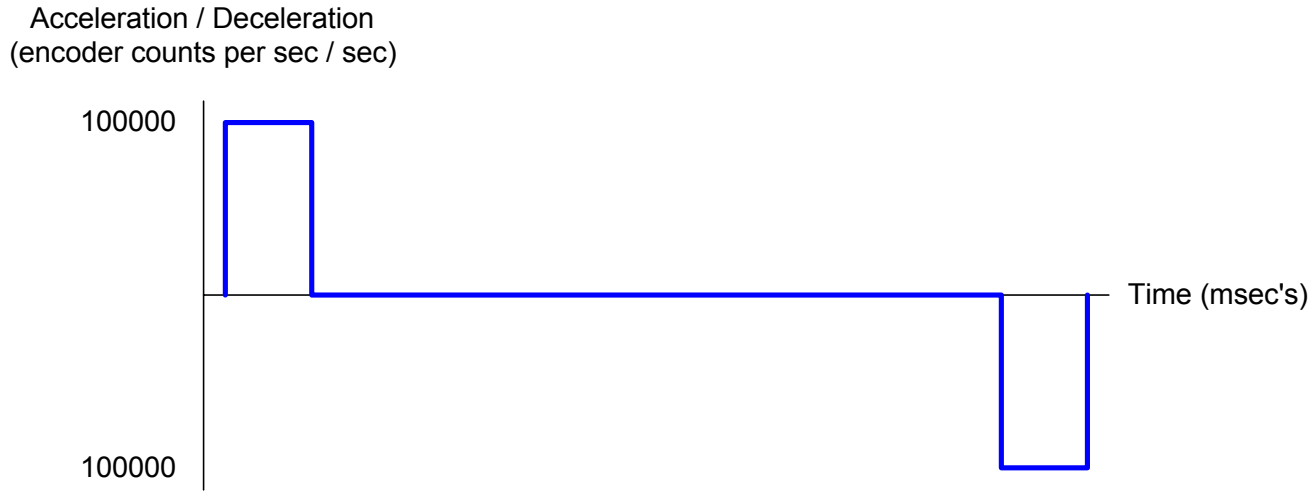
Prior to executing any move, the user should define the parameters of the move. The components that make up a move are:

- Maximum velocity during the move
- Acceleration - rate of change to achieve maximum velocity
- Deceleration - rate of change to target position
- Velocity profile (Trapezoidal, S curve, Parabolic)
- Target position (not applicable for velocity mode motion)

```
1MN                                  ;turn on the axis (enables the PID if
                                     ;axis is a servo)
1PT                                  ;trapezoidal velocity profile
1SV100000,1SA100000,1DS100000        ;define the trajectory parameters
1MA10000                             ;move to absolute position 10,000
```

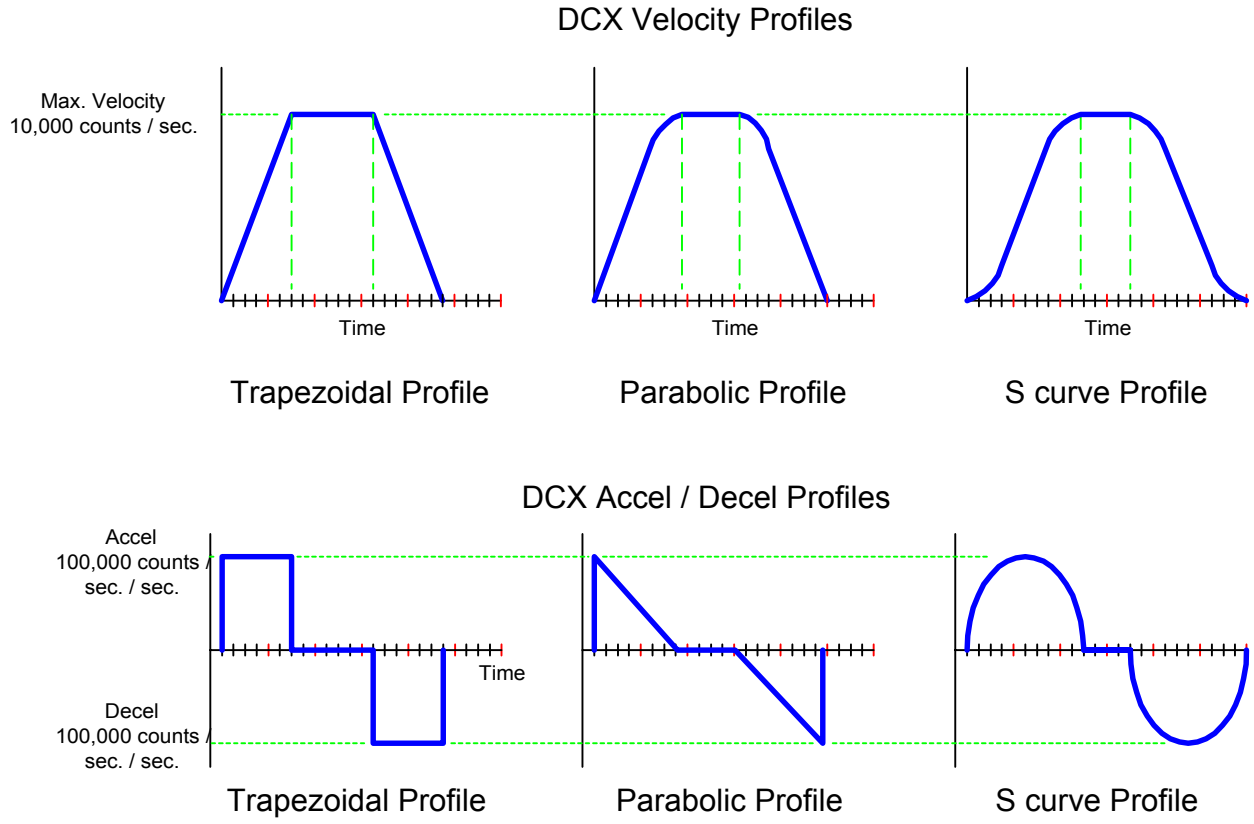
The parameters defined in the command example above specify a move to position 100,000. During the move the velocity will not exceed 10,000 encoder counts per second. A trapezoidal velocity profile will be calculated by the DCX. The rate of change (acceleration and deceleration) will be 100,000 encoder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




Acceleration / Deceleration
(encoder counts per sec / sec)
100000
100000
Time (msec's)

Velocity Profiles

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



Trapezoidal Profile – (servo's and/or steppers)
- 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 – (recommended for steppers only)
- Slow 'roll off' minimizes lost steps at high velocity
- Initial linear rate of change eliminates 'cogging'
- On the fly changes will cause the axis to first decelerate to a stop

S curve Profile – (recommended for servo's only)
- 'True sine' rate of change effectively eliminates 'jerk' and/or oscillation
- Longest time to target when using the same trajectory parameters
- On the fly changes will cause the axis to first decelerate to a stop

Point to Point Motion

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

Turn on the axis
Enable Position mode
Define the velocity profile (trapezoidal, S curve, or parabolic)
Define the velocity parameters (maximum velocity, acceleration, and deceleration)
Issue move command

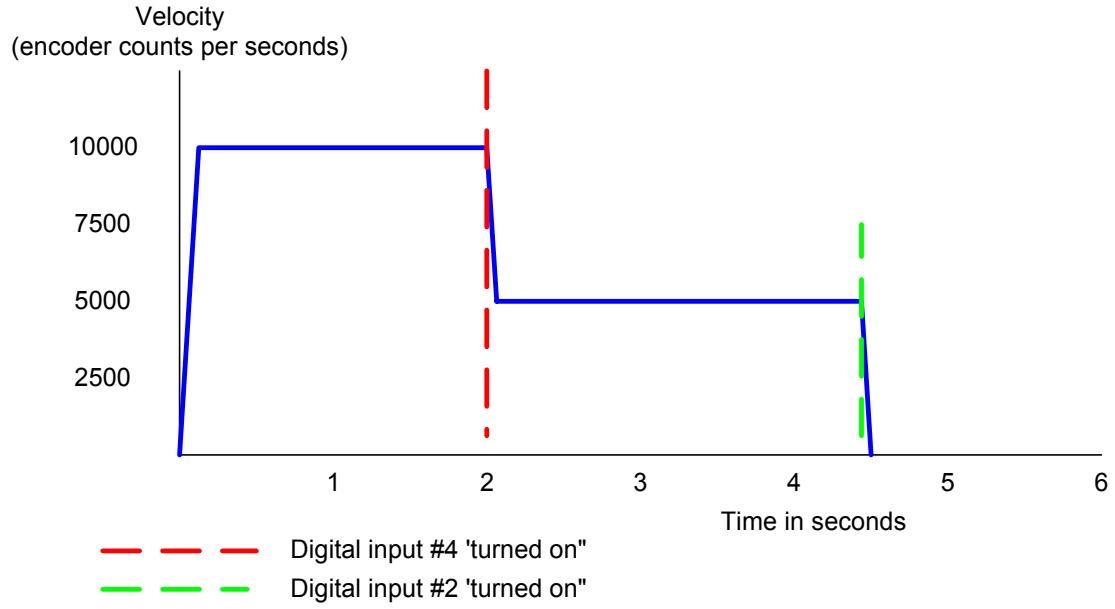
```
1MN                                  ;turn on the axis (enables the PID if
                                     ;axis is a servo)
1PM                                  ;position mode
1SV100000,1SA100000,1DS100000        ;define the trajectory parameters
1MA10000                             ;first move to absolute position 10,000
1WS0.01                              ;wait 10msec's after move complete
1MR5400,1WS0.01,1MA0                 ;move relative +5400, wait for move
                                     ;complete, move to absolute position 0.
```

Constant Velocity Motion

To move a servo or stepper at a continuous velocity until commanded to stop the following steps are required:

Turn on the axis
Enable Velocity mode
Define move as trapezoidal velocity profile (S curve and parabolic not supported)
Define the velocity parameters (maximum velocity, acceleration, and deceleration)
Define the direction of the move (positive = 0, negative = 1)
Issue move command

```
1MN                                  ;turn on the axis (enables the PID if
                                     ;axis is a servo)
1VM                                  ;velocity mode
1SV10000,1SA100000,1DS100000         ;define the trajectory parameters
1DI0,1GO                             ;define the direction, start motion
IN4,1SV5000,NO,JR-3                  ;if digital input #4 is on decrease
                                     ;velocity
IN2,1ST,NO,JR-3                      ;stop the axis when digital input #2 is
                                     ;on
```



Contour Motion (arcs and lines)

The DCX supports Linear Interpolated motion with any combination of two to six axes and Circular Contouring on as many as three 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 Contour Mode (aCMn) command is used to define the axes in a contour group. Issue the Contour Mode command to each of the axes in the contour group. The parameter *n* to the contour mode commands should be the lowest axis number of the servos or stepper motors that will be moving on the contour. This axis will then be defined as the 'controlling' axis for the contour group.

```
1CM1,2CM1,3CM1                          ;define the contour group (axes 1, 2,
                                        ;& 3), axis #1 is the controlling axis
```

Define the trajectory parameters

Issue the Vector Velocity (VV), Vector Acceleration (VA) and Vector Deceleration (VD) commands to the controlling axis of the contoured motion. The default units for the parameter to the vector velocity command are encoder counts or steps per second. The default units for the vector acceleration and deceleration commands are encoder counts or steps per second per second.

```
1VV10000,1VA100000,1VD100000            ;define the trajectory parameters
```

Define the type of contour move

Use the Contour Path (*a*CP*n*) command as the first command in a 'compound command' that specifies a motion. The controlling axis number *a* must precede the contour path command. The parameter *n* to the contour path command selects the type of motion. Please refer to the table that follows:

CPn where n =	*Contour move type*
0	User defined, 1 to 6 axes
1	Linear interpolated move, 1 – 6 axes
2	Clockwise arc, 2 axes
3	Counter Clockwise arc, 2 axes

Loading the Contour Buffer for Continuous Path Contouring

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

When axis 1 is the controlling axis, up to 256 linear or 128 arc motions (an arc move takes up twice as much buffer space) can be queued up in the Contouring Buffer. If one of the other five axes is the controlling axis, only 16 motions can be queued up. The Tell Contour (aTX) command (where a = controlling axis) will report how many contour moves have been executed since the Contour Mode command was last issued. The count reported by the Tell Contour command is stored as a 32 bit value.

To delay starting contour motion until the contour buffer has been loaded use the Synchronization oN (aSN) command. This command should be issued to the controlling axis a **before** issuing any contour moves. Contour moves issued after the Synchronization oN command will be queued into the contour buffer. To begin executing the moves in the buffer, issue the Go command to the controlling axis with parameter n = 0.

```
1SN                                    ;turn on synchronization
1GO0                                   ;start executing the moves queued in
                                       ;the contour buffer
```

To return to normal operation (immediate execution of contour move commands), issue the No Synchronization (aNS) command 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.

```
1CP1,1MA85000,2MR12000,3MA-33000       ;first linear move in X, Y, & Z
1CP1,1MA0,2MR0,3MA0                    ;first linear move in X, Y, & Z
1CP1,1MA5000,2MR23000,3MA-16000        ;first linear move in X, Y, & Z
```

Arc Motion
The DCX supports specifying arc motion in any of three different ways:

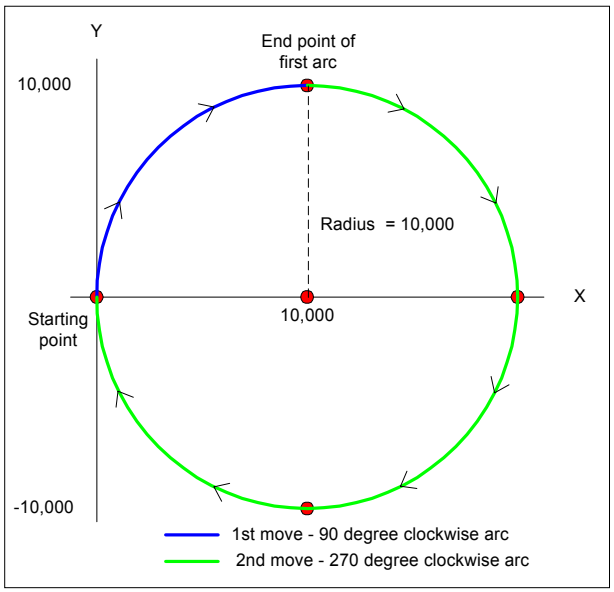
Specify center and end point
Specify radius and end point
Specify center and ending angle

Arc motions by specifying the center point and end point
The arc Center Absolute (CA) and arc Center Relative (CR) commands are used to specify the center position of the arc. These commands also serve to select which two axes will perform the arc motion. Either the Move Absolute (MA) or Move Relative (MR) 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.

```
1CP2,1CA10000,2CA0,1MA20000,2MA0        ;clockwise arc in X and Y
1CP3,1CR-10000,2CR0,1MR-20000,2MR0      ;counter clockwise arc in X and Y
```

Arc motions by specifying the radius and end point
The aRc Radius (*a*RRn) 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.

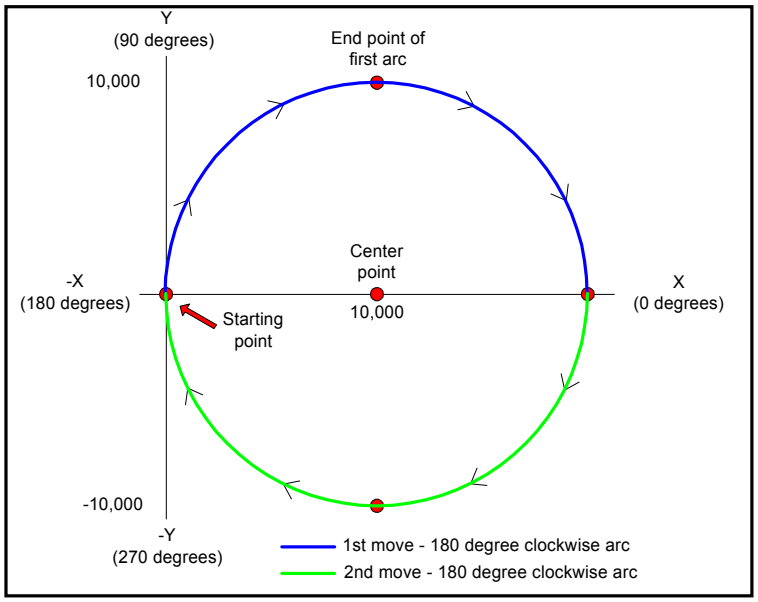
```
1CM1,2CM1                               ;define axis 1 as controlling axis
1CP2,1MR10000,2MR10000,1RR10000         ;90° clockwise arc, radius = 10000
1CP2,1MR-10000,2MR-10000,1RR-10000      ;270° degree arc, radius = 10000,
                                        ;negative radius parameter indicates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



Arc motions by specifying the center point and ending angle
The Ending arc angle Absolute (aEAn) and Ending arc angle Relative (aERn) commands can be used in conjunction with Center Absolute (CA) and arc Center Relative (CR) commands 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 (CA, CR) define the radius of the arc. The ending arc angle commands (EA, ER) 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 (aVOn) command. Issue the command (to the controlling axis) to scale the vector velocity of a linear or arc motion. The rate of change will be defined by the vector acceleration or vector deceleration settings. The command parameter n sets the scaling factor that is applied to the programmed vector velocity. A parameter value of 0.5 would reduce the vector velocity to half of the nominal value.

```
1VO0.5                                 ;reduce vector velocity to 50%
1VO1.0                                 ;reset vector velocity programmed
                                       ;value
```

Cubic Spline Interpolation of linear moves

To have the DCX perform 'curve fitting' (cubic spline interpolation) on a series of linear (aCP1) moves, issue the Synchronization oN (aSNn) command to the controlling axis. Next issue linear contour path commands to points on the curve. After loading the desired number of into the contour buffer, issue a GO command with a parameter of 1 to the controlling axis. Motion will continue from the first to the last point loaded. To return to normal operation, issue the No Synchronization (NS) command to the controlling axis.

Note that when performing cubic spline interpolation, only **128 motions** can be queued up if **axis 1 is the controlling axis**. For **all other axes, 16 motions** can be queued up in the controller.

User Defined Contour path
For applications where orthogonal geometry is not applicable, the DCX allows the user to define a custom contour distance. In a typical X, Y, and Z axis linear move (CP1), the DCX controller will calculate the total length of the contour distance as follows:

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

Calculated Contour distance = $\sqrt{(X^2 + Y^2 + Z^2)}$
$= \sqrt{(10{,}000^2 + 10{,}000^2 + 5^2)}$
$= \sqrt{(100{,}000{,}000 + 100{,}000{,}000 + 1{,}000{,}000)}$
$= \sqrt{201{,}000{,}000}$
$= 14177.44$

The DCX would then utilize the settings for Vector Velocity, Vector Acceleration, and Vector Deceleration to calculate the trajectory of the move. When a User Defined Contour Path is selected (issue the Contour Path command with the parameter set to 0), the Contour Distance command is used to enter the non-orthogonal contour distance.

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 the User Defined Contour Path command (CP0) the Contour Distance 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 command is **not** used to set the velocity of the contour group. The axis with the longest move time 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 other axes will move at velocities lower than their specified maximum. The velocity profiles of the other axes in the contour group will be set such that all axes start and stop at the same time. In the following example, axes one and two 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 also travel at matching velocities.

```
1SV300,1SA1000,1DS1000                ;set velocity profile for axis 1
2SV100,2SA1000,2DS1000                ;set velocity profile for axis 2
1CM1,2CM1                             ;enable contour mode
1CP0,1MA1000,2MA1000,1CD0             ;use contour path and contour distance
                                      ;commands to set the time of the move
```

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 the its maximum velocity (set by the command 1SV300).

Electronic Gearing

On the DCX it's possible to slave any axis to one or two master axes. When one master axis is specified for a slave axis, 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. This function will work with either servo or stepper axes, with the master axis operating in jogging, position, velocity or contouring mode.

Electronic gearing **does not support S-curve or Parabolic** velocity profiles

To configure the DCX for master/slave operation, the Set Slave ratio (aSS*n*) and Set Master (SM) commands must be issued to each slave axis. the parameter *n* to the Set Slave ratio command is the ratio that is to be maintained between the position of the slave and master axis (the default ratio is 1.0). The slave ratio can range from –65535 to +65536. If the ratio of the slave 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 ratio of the slave is a negative value, a move in the positive direction of the master will cause a move in the negative direction of the slave.

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

After setting the ratio, the Set Master (aSM*n*)command is issued to the slave using the axis number of the master as the parameter *n*. Once 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.

```
2SS0.5                          ;define axis #2 as slave,
                                ;ratio = 0.5:1
3SS-20.0                        ;define axis #3 as slave,
                                ;ratio = -20:1
2SM1,3SM1                       ;define axis #1 as the master
1MN,2MN,3MN                     ;turn axes 1, 2, & 3
1MR1000                         ;move axis #1 +1000 counts relative.
                                ;Axis 2 will move 500 counts, axis 3
                                ;will move -20,000 counts
```

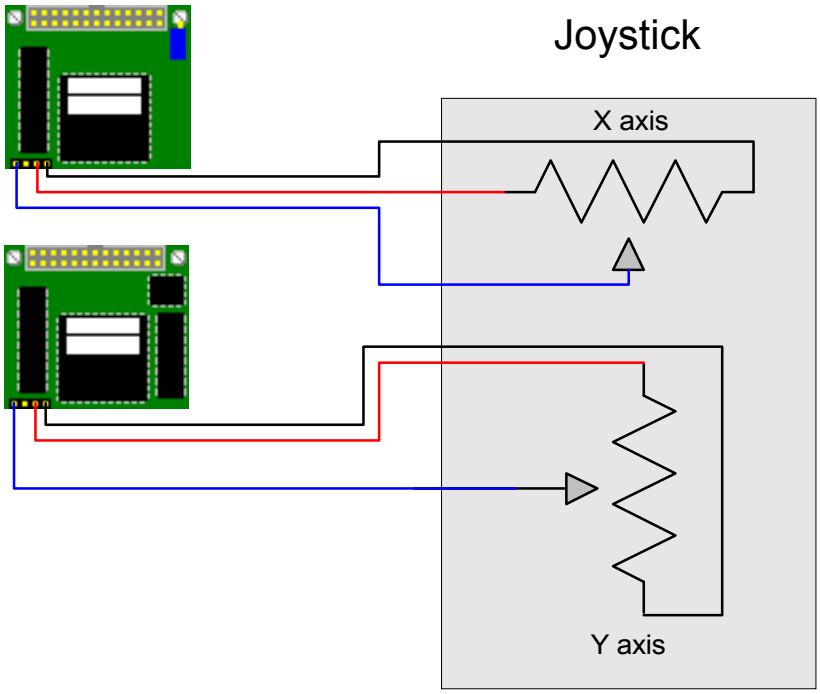
To terminate the master and slave connections between the axes, issue the Set Master (aSMn) command to the each of the slave axes with parameter n = 0, followed by either the Position Mode (PM) or the Velocity Mode (VM) command.

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

Jogging

In some applications it may be necessary to have a means of manually positioning the motors. Since the DCX is able to control the motion of servos and steppers with precision at both low and high speeds, all that is required is a manual command input. A joystick provides such an input with natural hand to motion coordination. It can be used for jogging both servo (DCX-MC200, DCX-MC210) and stepper (DCX-MC260) motors on the DCX. The rest of this section describes how to implement joystick control.

Typical joysticks have 2 potentiometers, one for each axis of motion. Each 'pot' of a joystick can be connected to one servo or stepper module. Pots with a total resistance between 10K and 100K ohms are suitable for use with the DCX motor modules. By connecting the end taps of one pot to pins 3 (+5V supply) and 4 (Ground) of a motor module's J4 connector, and the center tap to pin 1, the module is able to read the pot's position.



To test the connection of the joystick to a module's analog input, use the Read Float and Tell Register commands as in the following example:

```
1RV200,TR                              ;load the A/D reading of axis #1 into
                                       ;the accumulator, report the A/D value
```

The value that is reported (by the Tell Register command) is the voltage level at the module's analog input in units of volts. With the joystick at its center position, the reading should be 2.5 VDC. Most joysticks have centering adjustments to correct this reading if necessary. As the joystick is moved from one extreme to the other, the analog input reading should range from 0 to +5.0.

When jogging is enabled on an axis of the DCX, the readings from the analog input, multiplied by the Jog Gain, will set the servo or steppers maximum velocity. Given the desired maximum velocity when

the joystick is pushed to the extreme, the appropriate Jog Gain can be calculated by the following equation:

Jog Gain = Desired Velocity / 2.5

For example, if the desired maximum velocity is 1000 counts per second, the Jog Gain should be set to 1000 / 2.5 = 400.

```
1JG400
```

During jogging, the servo or stepper will accelerate and decelerate at the rate specified using the set Jog Acceleration command. For closed loop servos, this value is typically set high to provide quick response to the joystick. For stepper motors, the acceleration must be set to a value low enough that the motor will not stall during acceleration. For stepper motors, using the set Jog minimum Velocity command will enhance the response of the motor to the joystick. This command will have no effect on servo motors. The following command will set the jogging acceleration to 10000 counts per second per second.

```
1JA10000
```

Jogging of the servos or steppers is enabled using the Jog oN command. After issuing this command the servos should move with a velocity proportional to the amount the joystick is depressed from its center position.

To increase or decrease the maximum velocity, the Jog Gain can be adjusted up or down at any time. Depressing the joystick in the opposite direction should cause the servo or stepper to reverse direction. To change the direction that a servo or stepper moves in response to the joystick, set the Jog Gain to a negative value.

Because of mechanical and electrical variations, the joystick reading may dither around 2.5 when the joystick returns to its' center position. This may cause undesirable drifting of the servos or steppers when the joystick is enabled. To prevent this, another parameter called Jog Deadband can be adjusted. The parameter to the Jog Deadband command is in units of volts and specifies the amount that the joystick reading can very from the center offset before it has any effect. Once the reading exceeds the deadband level, it will increase linearly from zero. The following command sequence will provide a 0.06 volt deadband in the joystick operation:

```
1JB0.06
```

It was stated earlier that most joysticks have adjustments for setting the center positions. The DCX also has a parameter for adjusting the center position of the joystick. The Jog Offset command is used to set the center position in units of volts. This value defaults to 2.5 volts. Changing the Jog Offset to a different value will provide an increased velocity range in one direction, but a reduced range in the opposite direction. For example, the following command would set the center voltage to 2.0 volts:

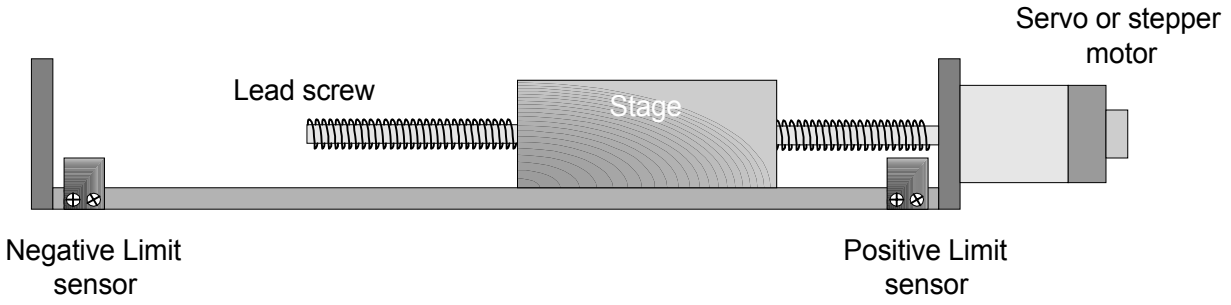
```
1JO2.0
```

To disable the joystick for an axis, issue the Jog OFf command:

```
1JF
```

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 default to TTL low true. When the input signal is pulled low (>.7V) by the limit switch/sensor the input is considered active. To configure the Limit switch inputs for high true (normally closed) operation, issue the Limit Mode command (see table on next page) with the parameter 'n' set to 128 (MSB set) plus the desired mode.

When properly connected to a DCX-MC2XX motor control module, if a limit switch input is 'active', the appropriate status bit (positive limit = bit 28, negative limit = bit 29) will be set. These two status bits will always reflect the ***current state*** of the two limit inputs. The example below will display the current state of the limit inputs:

Manually activate the positive limit sensor/switch

```
HM,1TS                          ;define hex mode, report the status of
                                ;axis #1
01     100100c                  ;bit 28 is set, the MC2XX properly
                                ;indicated Limit + is active

1LM1,1LN0                       ;define the limit mode (stop abrupt),
                                ;enable limit error checking



1MR10000                        ;move the axis then activate the limit
                                ;+ sensor. The motor should have
                                ;stopped
1TS                             ;report the status of the axis
01     1007008a                 ;bits 16 (limit + enabled) and 17
                                (limit + tripped) are set
```

To enable hard coded error checking of the hard limits use the Limit Mode (aLMn) and Limit oN (aLNn) commands. Issuing the Limit On command will cause the appropriate motor status bits (Hard Limit Positive Input Enabled – bit 16 and Hard Limit Negative Input Enabled – bit 18) to be set. If a Limit sensor goes active after it has been enabled by the motion Limits oN command, **and** the motor has been commanded to move **in the direction of that switch**, the Motor Error (motor status bit 7) and one of the Hard Limit Tripped Flags (Hard Limit Positive Tripped – bit 17 and Hard Limit Positive

Tripped – bit 19) 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 Limit Mode command.

The action that a motor takes when it encounters a hard or soft motion limit can be selected with the Limit Mode command. The default action is to turn the motor off. Alternatively, the motor can be stopped abruptly, or decelerated to a stop at the current deceleration rate. When a group of motors are to be operated in contour mode, their limit mode's should be set to turn the motors off.

Limit Mode (aLMn) command

Desired action	***Parameter n =***
Turn motor off (disable PID) when limit sensor 'goes' active	0,0 **
Stop the motor abruptly when limit sensor 'goes' active	1,4 **
Stop the motor smoothly when limit sensor 'goes' active. Use the current deceleration setting.	2,8 **

* Values in red are for defining the Limit Mode for soft limits. if both hard and soft limits are to be used, the parameter n should equal hard limit parameter n + soft limit parameter n.
*For ***high true (normally closed)*** Limit switch inputs, issue the Limit Mode command with the parameter 'n' set to 128 (MSB set) plus the desired mode (n = 0,1,2,4 or 8).

Limit oN (aLNn) command

Hard limit inputs enabled	***Parameter n =***
Positive and Negative	***0**,3,12 **
Positive	1,4 **
Negative	2,8 **

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

For both hard and soft motion limits, the error flags (bits 7, 17, 19) will remain set until the motor is enabled with the MN command. Once the motor is enabled, it must be moved out of the limit region with a move command (MA, MR, GO, etc…).

```
1LM1,2LM1                       ;stop axes 1 & 2 immediately when a
                                ;hard limit error occurs
3LM2                            ;stop axis 3  smoothly (decelerate to
                                ;a stop) when a hard or soft limit
                                ;error occurs

1LM3,2LM3                       ;enable hard limit error checking for
                                ;axis 1 & 2
3LM0                            ;enable hard and soft limit error
                                ;checking for axis 3
```

Soft Limits

If a soft motion limit is enabled, and the respective axis goes beyond the motion limits set by the High motion Limit (aHLn) and the Low motion Limit (LLn) commands, the Motor Error (motor status bit 7) and one of the Soft Limit Tripped Flags (Soft Motion Limit High Tripped – bit 21 and Soft Motion Limit Low Tripped – bit 23) will be set. The motor will be turned off or stopped based on the setting of parameter *n* of the Limit Mode command.

For both hard and soft motion limits, the error flags will remain set until the motor is turned back on with the MN command. Once the motor is turned back on, it must be moved out of the limit region with a move command (MA, MR, GO, etc…).

The action that a motor takes when it encounters a hard or soft motion limit can be selected with the Limit Mode command. The default action is to turn the motor off. Alternatively, the motor can be stopped abruptly, or decelerated to a stop at the current deceleration rate. When a group of motors are to be operated in contour mode, their limit mode's should be set to turn the motors off.

Limit Mode (aLMn) command

Desired action	***Parameter n =***
Turn motor off (disable PID) when limit sensor 'goes' active	0,0 *
Stop the motor abruptly when limit sensor 'goes' active	1,4 *
Stop the motor smoothly when limit sensor 'goes' active. Use the current deceleration setting.	2,8 *

* Values in red are for defining the Limit Mode 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.

Limit oN (aLNn) command

Soft Motion limit enabled	***Parameter n =***
High and Low	***0**,3,12 **
High	1,4 **
Low	2,8 **

* 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.

Homing Axes

When power is first applied to the DCX (or it is reset), all servo and stepper positions are initialized to zero. If they are subsequently moved, the controller will report their positions relative to the position where they were last initialized.

In most applications, there is some position of the motor (or mechanical apparatus) that is considered 'home'. When the motor is at this position, it is desirable to have the controller report its' position as zero. On the DCX there are several ways to achieve this re-initialization of position. The simple implementation would be to manually jog an axis to the home area and then issue a Define Home (aDHn) command to the axis. Typical automated systems utilize electro-mechanical devices (switches and sensors) to signal the DCX when a motor is at its home position. Using the DCX homing commands an axis can be programmed to find its home position.

Verifying the operation of the Home Sensor

The following command sequences will verify the connections to, and operation of, the Coarse Home (servo) and Home (stepper) sensor circuits

MC200 & MC210

```
1TS25                                   ;report the current state of the
                                        ;Coarse Home input
       01     0                         ;DCX reports that the input is off (0)

;activate the Coarse Home sensor
1TS25                                   ;report the current state of the
                                        ;Coarse Home input
       01     1                         ;DCX reports that the input is on (1)
```

MC360

```
1TS24                                   ;report the current state of the
                                        ;Coarse Home input
       01     0                         ;DCX reports that the input is off (0)

;activate the Stepper Home sensor
1TS24                                   ;report the current state of the
                                        Coarse Home input
       01     1                         ;DCX reports that the input is on (1)
```

Verifying the operation of the Encoder Index

The Find Index (aFIn) command is a conditional operation that will initialize the reported position of a servo to *n* when:

1) The encoder index pulse is asserted.
2) The Motor oN command has been issued.

The following command sequences will verify the connections to, and operation of, the encoder index.

```
1FI0                                    ;begin the capture of the encoder
                                        ;index mark. Find Index is a
                                        ;conditional command which will not
                                        ;complete execution until the encoder
                                        ;index has been captured.

:rotate the axis/encoder one full revolution, if the Find Index command does
not complete execution then the index mark was not captured by the DCX module.
To abort the Find Index command issue an escape

1MN                                     ;If the index mark is captured issue
                                        ;the Motor oN command to redefine that
                                        ;physical location as position n
```

The DCX is not shipped from the factory, programmed to perform a specific homing operation. Instead, it has been designed to allow the user to define a custom homing sequence that is specific to the system requirement. Using any of the built-in motion commands as part of a homing macro, the user can write a sequence of moves to perform the necessary homing operation. It is recommended that the homing sequence be defined as macros. The example homing sequences included in the descriptions that follow all begin with macro #1. This allows the programmer to call a complete homing sequence for axis #1 with one command:

```
MC1
```

The DCX also supports using the Home (aHO) command to begin a homing sequence. The following table shows how to associate a macro number with a homing sequence called by the home command:

Axis Number	*Macro Number*	*Home Command*
1	MD1	1HO
2	MD2	2HO
3	MD3	3HO
4	MD4	4HO
5	MD5	5HO
6	MD6	6HO

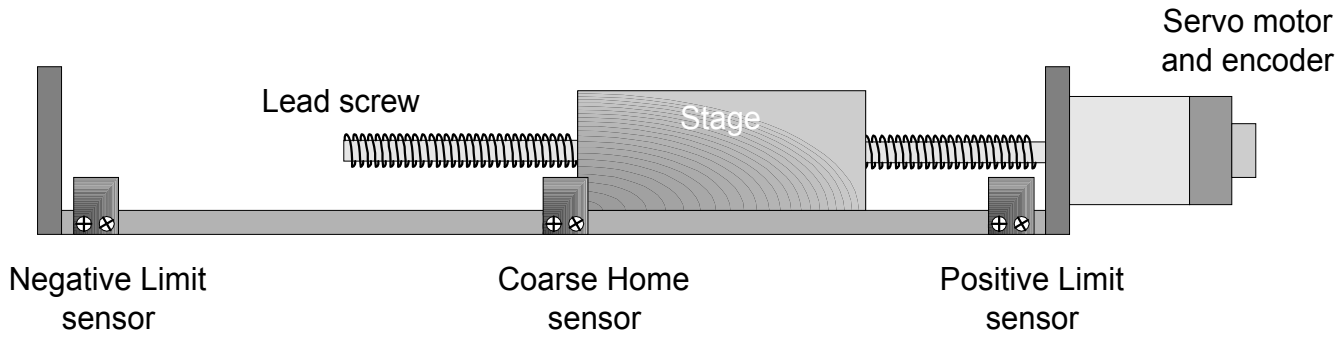
As with any other macro, a homing sequence can also be spawned as a background task by using the Generate Task command.

Homing Servo's

The encoder of a servo will 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. No additional sensors are required when an axis need only to be homed within 360 degrees. The following command sequence will home a multi turn rotary stage:

```
1VM,1DI0,1GO,1FI0,1ST,1WS.01,1PM,1MN,1MA0,1WS.01
                                        ;begin velocity mode move, capture the
                                        ;position of the index mark, stop the
                                        ;axis, issue position mode move to the
                                        location of the index mark
```

A typical linear axis will involve multiple rotations of the encoder over the full range of travel of the axis. These type of systems will typically utilize a coarse home sensor to qualify which of the index pulses are to be used in homing the axis. End of travel or Limit Switches will also be incorporated into the designed to protect against damage of the mechanical components. The diagram below depicts a typical single axis of linear stage.



When power is first applied to the DCX (or it is reset), the position of the stage is unknown. The following command sequence will move the stage in the positive direction. If the coarse home sensor 'goes active' before the positive limit sensor, the Find Index command 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.

```
MD1,1LM2,1LN3,MJ10                          ;call homing macro
MD10,1VM,1DI0,1GO,1RL0,IS25,MJ11,NO,IS17,MJ12,NO,JR-7
                                            ;test for sensors (home and +limit)
MD11,1ST,1WS.01,1DI1,1GO,1WE1,1ST,1DI0,1GO,1WE0,1FI0,1 ST,1WS.01,1PM,1MN,1MA0
                                            ;if home sensor true, initialize on
                                            ;index pulse
MD12,1MN,1DI1,1GO,1WE0,MJ11                 ;move negative until home true
```

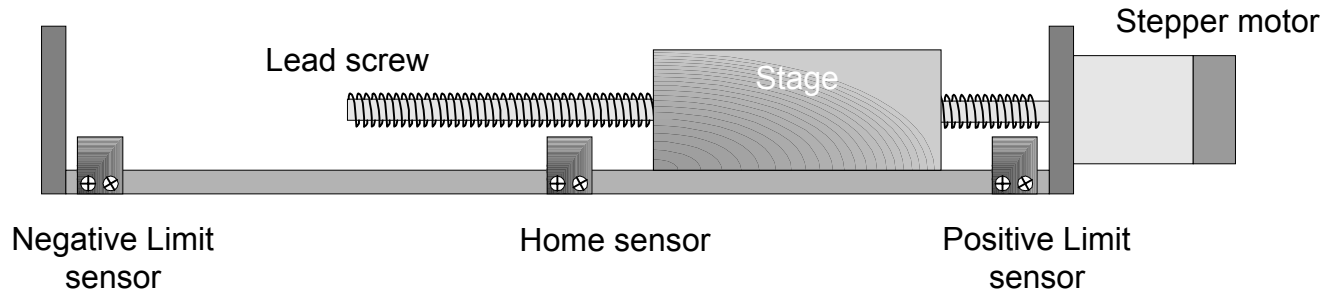
An axis can be homed even if no index mark or coarse home sensor is available. This method of homing utilizes one of the limit (end of travel) sensors to also serve as a home reference. Please note that this method **is not recommended** for applications that require **very high repeatability and accuracy**. To achieve the high possible accuracy when using this method, reduce the velocity of the axis while polling for the active state of the limit input.

The following command 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,1RL0,IS17,MJ11,NO,JR-4
                                            ;move and poll the Limit + sensor
MD11,1MN,1DI1,1SV5000,1GO,1RL0,IC17,MJ12,NO,RP-4
                                            ;move negative until limit + inactive
MD13,1ST,1WS.1,1DI0,1SV1000,1GO,1RL0,IC17,JR-2,NO,1 ST,1WS.1,1DH0,1PM,1MN
                                            ;move slowly in the positive direction
                                            ;until the limit + sensor is active.
                                            ;Stop and define the current position
                                            ;as home.
```

Homing open loop steppers

Open loop steppers are typically homed based on the position of a home sensor. Unlike servos that utilize a precision reference index mark, steppers are more prone to homing inaccuracies due the lower repeatability of the sensor output. 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 a 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 first applied to the DCX (or it is reset), the position of the stage is unknown. The following command sequence will move the stage in the positive direction. If the home sensor 'goes active' before the positive limit sensor, the Find Edge command will redefine the reported position of the axis. If the positive limit sensor 'goes active', the stage will change direction, until home sensor is located. The Find Edge command is then used to redefine the position of the axis. The command will remain in effect until the home input of the module goes active. At that time an internal position register of the MC360 module will be set to the current position. The position of the axis (where the index mark was captured) will not be defined to position *n* until after the Motor Off / Motor oN (aMNn) command sequence has been issued.

Note: The status bit associated with the Find Edge command is bit 24 (stepper Home). The Find Edge command causes this bit to be latched after the index mark has been captured. To clear the latched status bit, issue the Motor Off and Motor oN sequence.

```
MD1,1LM2,1LN3,MJ10                        ;call homing macro
MD10,1VM,1DI0,1SV10000,1GO,1RL0,IS24,MJ11,NO,IS17,MJ13,NO,JR-7
                                          ;test for sensors (home and +limit)
MD11,1ST,1WS.1,1DI0,1SV5000,1GO,1RL0,IC24,MJ12,NO,JR-4
                                          ;Move positive until home sensor off
MD12,1ST,1WS.1,1DI1,1SV5000,1GO,MJ15
                                          ;move back to the home sensor
MD13,1MN,1DI1,1SV5000,1GO,MJ15            ;move out of limit sensor range back
                                          ;toward the home sensor
MD14,1FE0,1ST,1WS.1,1MF,WA.1,1MN,1PM,1MA0
                                          ;find the active edge of the home
                                          ;sensor. Stop axis, initialize
                                          ;position, move to position 0.
```

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

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

```
MD1,1LM2,1LN3,MJ10                          ;call homing macro
MD10,1VM,1DI0,1GO,1RL0,IS17,MJ11,NO,JR-4
                                            ;move and poll the Limit + sensor
MD11,1MN,1DI1,1SV5000,1GO,1RL0,IC17,MJ12,NO,RP-4
                                            ;move negative until limit + inactive
MD13,1ST,1WS.1,1DI0,1SV1000,1GO,1RL0,IC17,JR-2,NO,1 ST,1WS.1,1DH0,1PM,1MN
                                            ;move slowly in the positive direction
                                            ;until the limit + sensor is active.
                                            ;Stop and define the current position
                                            ;as home.
```

Homing closed loop steppers

Homing a closed loop stepper is similar to homing an axis with an auxiliary encoder , see the description of **Auxiliary Encoder** in the **Application Solutions** chapter. The Index mark of the encoder is connected to the Auxiliary Encoder Index – input pin (connector J3 pin 22). A Coarse Home sensor is connected to the Auxiliary Encoder Coarse Home input (connector J3 pin 23). This device is used to qualify which index mark pulse is to be used for homing.

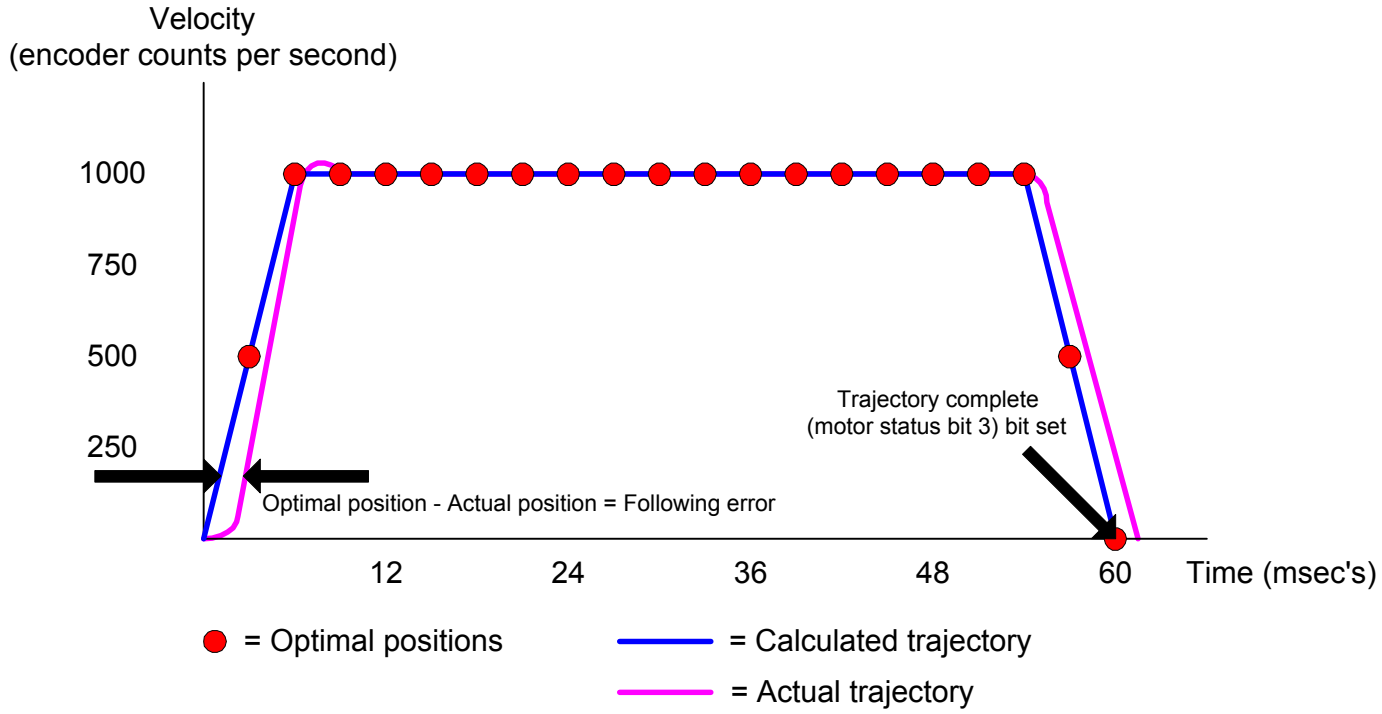
```
MD10,1MN,1VM,1SV10000,1SA100000,1DS100000,MJ11
                                            ;Enable axis and Velocity mode, set
                                            ;trajectory parameters
MD11,1DI0,1GO,MJ12                          ;Set direction (0=positive), start motion
MD12,1WE0,1SV1000,MJ13                      ;Wait for the auxiliary encoder Coarse Home
                                            ;input to be true, then reduce the velocity
                                            of the axis
MD13,1AF,1RL0,IS27,MJ14,NO,jr-3             ;loop until aux. encoder index is captured
MD14,1RL20,AR100,MJ15                       ;store the position of aux. encoder index in
                                            ;register 100
MJ15,1ST,1WS0.1,1PM,1MA@100,1WS0.1,MJ16     ;stop axis, move to location of aux. encoder
                                            ;index
MD16,1AH0                                   ;define the current position as aux. encoder
                                            ;home (position = 0)
```

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 4 msec's. These 4 msec optimal positions are passed to the DCX servo modules, which then performs a linear interpolation at the servo loop rate.



When the optimal position of an axis is the equal to the move target, the 'digital trajectory' of the move has been completed and the ***Trajectory Complete*** bit (motor status bit 3) will be set. This status bit is the conditional component of the ***Wait for Stop*** command (aWSn). As shown above, a following error can cause the trajectory complete bit to be set before the axis has reached its target. Issuing the Wait for Stop command with time value *n* allows the user to delay additional command execution until the move has been completed (following error = 0). In this example, issuing the Wait for Stop command with a delay of 3 msec's will delay the setting of the Trajectory Complete status bit until the move is complete.

```
1WS.003                          ;set trajectory complete bit 3 msec's
                                 ;after optimal position equals target
                                 ;position
```

Another method for indicating the end of a move is to use the ***At Target*** (motor status bit 2) status bit. The Position Deadband (aDBn) and Delay at Target (aDTn) commands are used to define an acceptable 'At Target range' for the axis. The Position Deadband command defines the 'At Target'

range (in encoder counts) of an axis. The Delay at Target command defines the amount of time that the axis must remain within the 'At Target' range before the "At Target' status bit will be set.

```
1DB5,1DT.005                    ;define 'At Target' parameters (+/- 5
                                ;encoder counts for 5 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 4 msec's.

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

Note – S-curve and Parabolic velocity profiles do not support 'on the fly' changes of target, velocity, acceleration and/or deceleration .

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).

While moving with Parabolic or S-curve profiles, on the fly changes will cause the axis to first decelerate to a stop, then resume motion.

Pause and Resume Motion

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

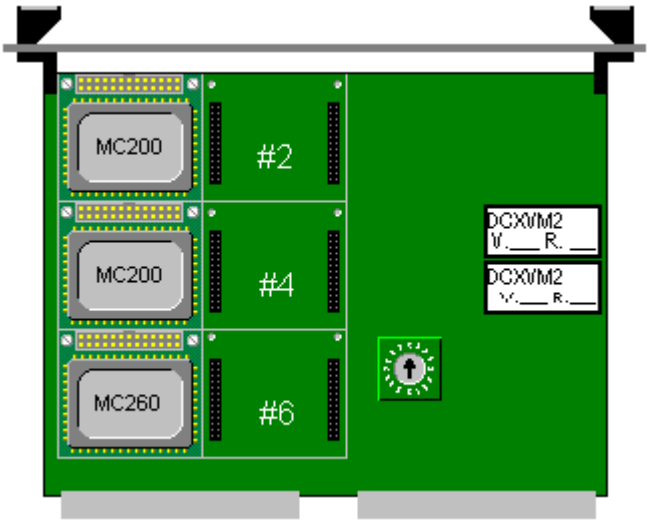
Each of these commands takes an axis specifier *a* and requires a file number as the command parameter n. These commands save and restore the entire motor table. This includes the public motor table in dual port memory and the private motor table in internal RAM.

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

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

Physical Assignment of Axes Numbers

The DCX defaults to assigning axis numbers logically, not based solely on a motor module's physical location. In the graphic below three modules are installed on a DCX-VM200. The MC200 in module location #1 would be defined as axis one. The MC200 in module location #3 would be defined as axis two. The MC260 in module location #5 would be defined as axis three.



The DCX does support the assigning of axis numbers based only on their physical location. The Use Physical command will redefine the logical addresses. To redefine axes 2 and 3 in the graphic above as axes 3 and 5:

```
3UP3                                    ;Define the module in location 3 as
                                        ;axis 3
5UP5                                    ;Define the module in location 5 as
                                        ;axis 5
```

Note – The reassignment of axes should be done before sending any other commands (setup, move, etc.)

Chapter Contents

Chapter 6

Application Solutions

Auxiliary Encoders

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

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

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

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

Typically an auxiliary encoder is added to a closed loop servo to allow the user to retrieve the position of the ‘end effector’ **at the end of a move**. The position of the auxiliary encoder is not a component of

the servo command output as calculated by the digital PID filter. The auxiliary encoder is used to determine whether or not the axis is properly positioned.

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

1MA1675.5,1WS.01                    ;move to absolute position 1675.5
1RD28,AR100                         ;load target position into register 100
1GX,AR101                           ;load auxiliary encoder position into
                                    ;register 101
AL@100,AS@101,AR102,IG.5,MJ100,NO,BK
                                    ;find the difference of target and aux.
                                    ;encoder positions. if more than 0.5"
                                    ;short of target jump to macro 100
MD100,1MR@102,1WS.01                ;move axis
```

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 because open loop systems do not require tuning

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

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

1MA122.5,1WS.01                     ;move to absolute position 1675.5
1RD28,AR100                         ;load target position into register
                                    ;100
1GX,AR101                           ;load auxiliary encoder position into
                                    ;register 101
AL@100,AS@101,AR102,IG.5,MJ100,NO,BK
                                    ;find the difference of target and
                                    ;aux. encoder positions. if more than
                                    ;0.5" short of target jump to macro
                                    ;100
MD100,1MR@102,1WS.01                ;move axis
```

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 a mechanical switch/electromechanical sensor. The following programming example will initialize the reported position of the auxiliary encoder
at the location of the Index mark:

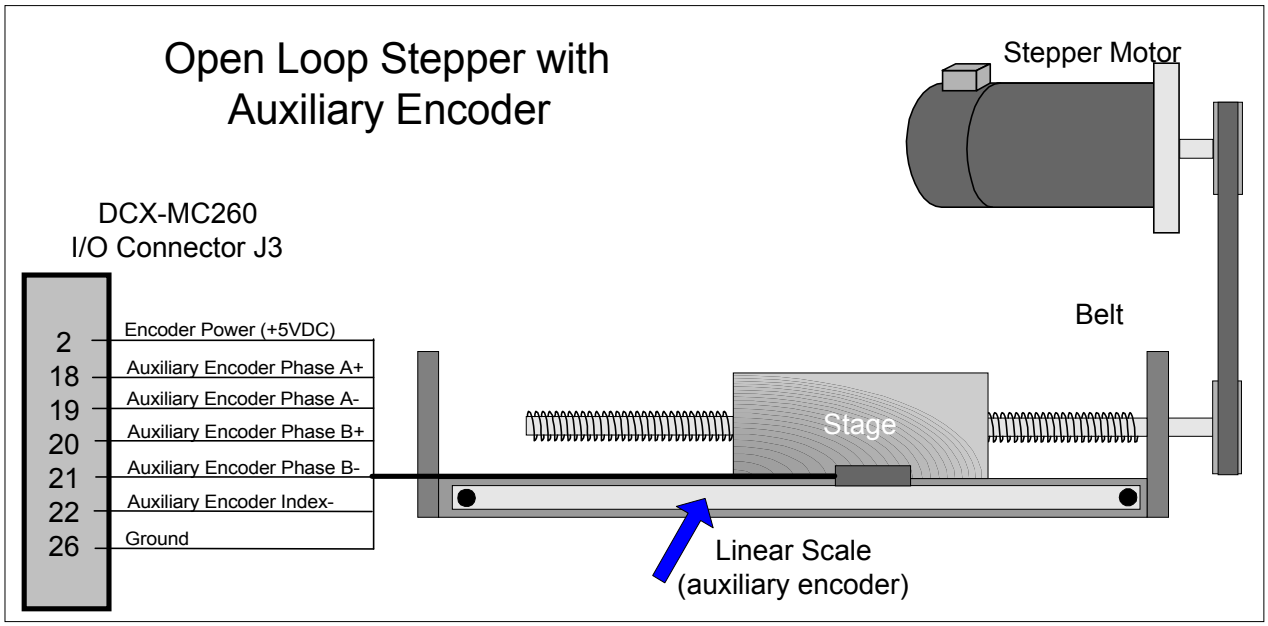
```
//  Enable the axis, place it in velocity mode, begin the move. After the Edge
// (Coarse Home Input), wait for the index  mark to be captured. Move to the
// location of the index mark, set the position of the auxiliary encoder
//
1MN,1VM,1SV1000,1DI0,1GO,1WE0           ;start velocity mode move, wait for
                                        ;aux. encoder coarse home = true
1AF,1RL0,IS27,MJ100,NO,JR-4             ;loop on aux. encoder index bit =
1MD100,1RL20,AR101,1ST,1WS.01           ;store location of index in
                                        ;register 101
1MA@101,1WS.01,1AH0                     ;move to location of index, define the
                                        ;position of aux. encoder to 0.
```

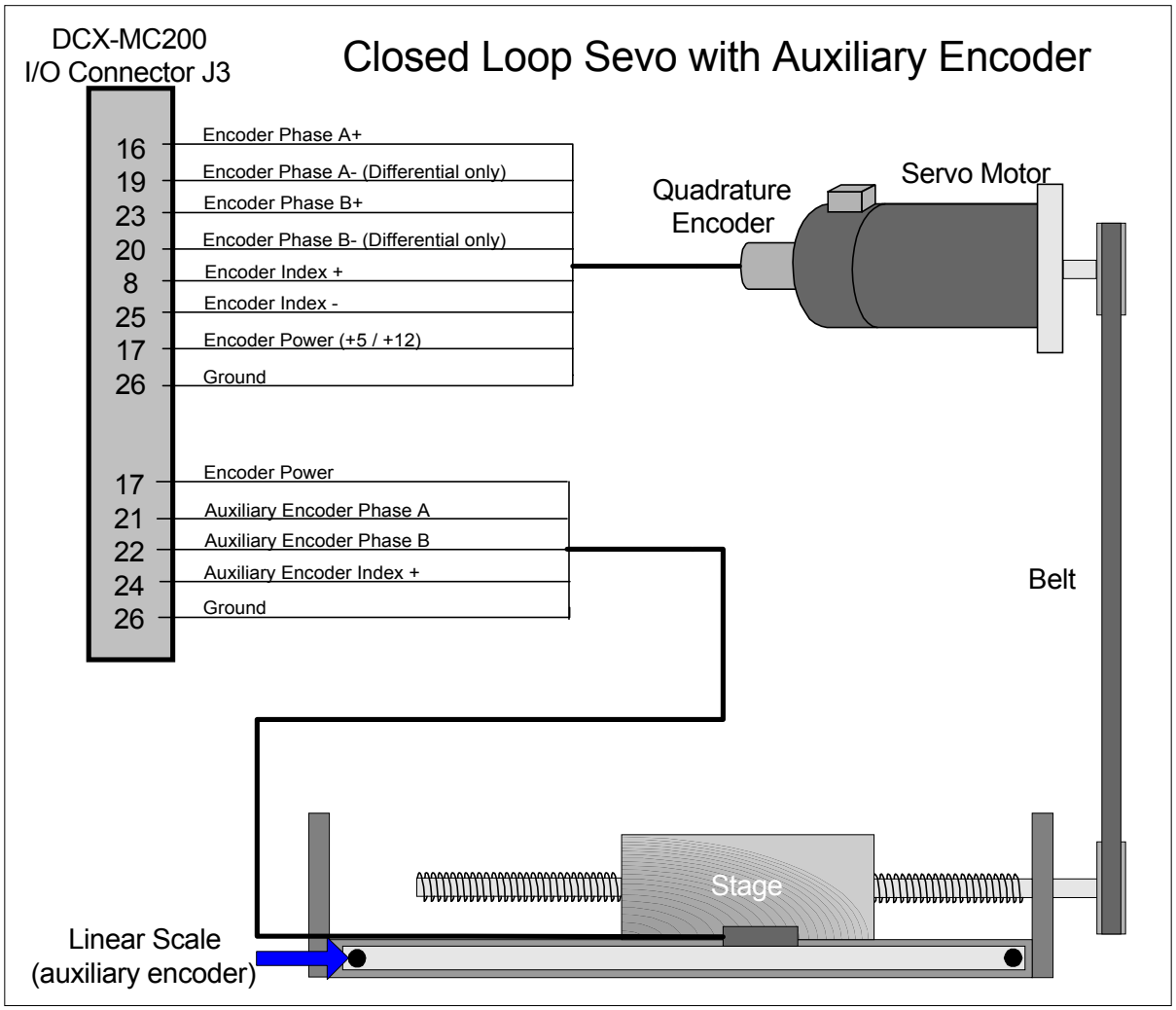
Unlike the Find Index (aFIn) command, which re-defines the position reported by a servos' encoder, the Auxiliary encoder Find index (aAF) command does not re-define the position of the auxiliary encoder. The Auxiliary encoder Find index command only arms the capture of the encoder index mark, which is then indicated by the status bit 27 (Auxiliary Encoder Index).

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

```
//  Home a stepper axis and re-define the position of the auxiliary encoder
//

1MN,1VM,1SV1000,1DI0,1GO,1FE0           ;start velocity mode move, wait for
                                        ;home sensor = true
1ST,1WS.01,1MA0,1WS.01                  ;stop the axis, move to position 0
1AH0                                    ;define the aux. encoder position = 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.


Open Loop Stepper with
Auxiliary Encoder
Stepper Motor
DCX-MC260
I/O Connector J3
2
18
19
20
21
22
26
Encoder Power (+5VDC)
Auxiliary Encoder Phase A+
Auxiliary Encoder Phase A-
Auxiliary Encoder Phase B+
Auxiliary Encoder Phase B-
Auxiliary Encoder Index-
Ground
Belt
Stage
Linear Scale
(auxiliary encoder)


DCX-MC200
I/O Connector J3
Closed Loop Sevo with Auxiliary Encoder
16
19
23
20
8
25
17
26
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
17
21
22
24
26
Encoder Power
Auxiliary Encoder Phase A
Auxiliary Encoder Phase B
Auxiliary Encoder Index +
Ground
Belt
Stage
Linear Scale
(auxiliary encoder)

Verifying the Operation of the Auxiliary Encoder
Use the Auxiliary encoder Tell position command to verify the operation of the auxiliary encoder. The following example 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                                 ;report the aux. encoder position
      01     0                      ;DCX reply

1AT                                 ;move the axis (increment the position
                                    of the aux. encoder)
      01     915                    ;DCX reply

1AT                                 ;move the axis (decrement the position
                                    of the aux. encoder)
      01     -636                   ;DCX reply
```

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 defined backlash distance. This feature is only available for servos (MC200 MC210) at this time.

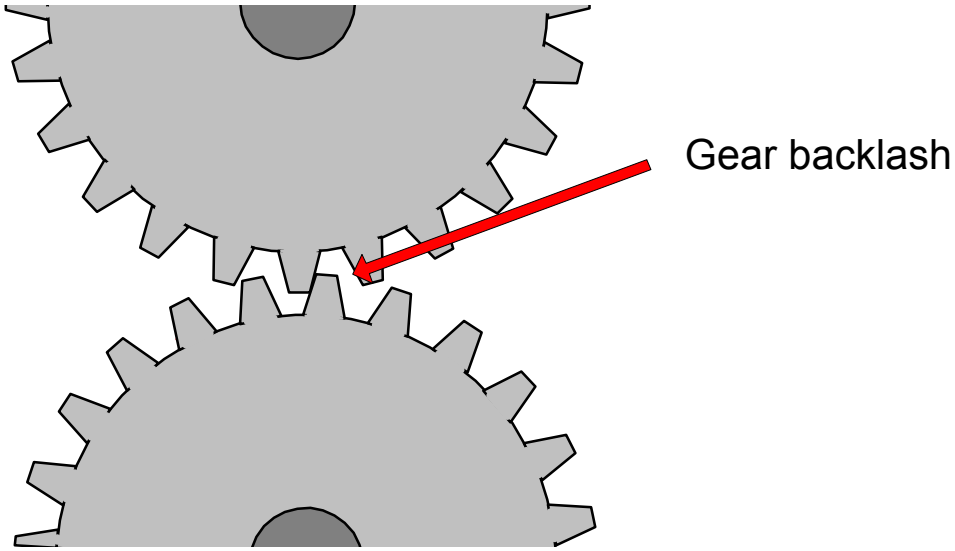
To setup this function, issue the Backlash compensation Distance (aBDn) command to the axis. The command parameter n should equal one half of the total system backlash. The units for this command parameter are encoder counts, or the units established by the User Scale command for this axis.

Once the backlash compensation distance is set, issuing the Backlash compensation oN (aBN) 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.

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.

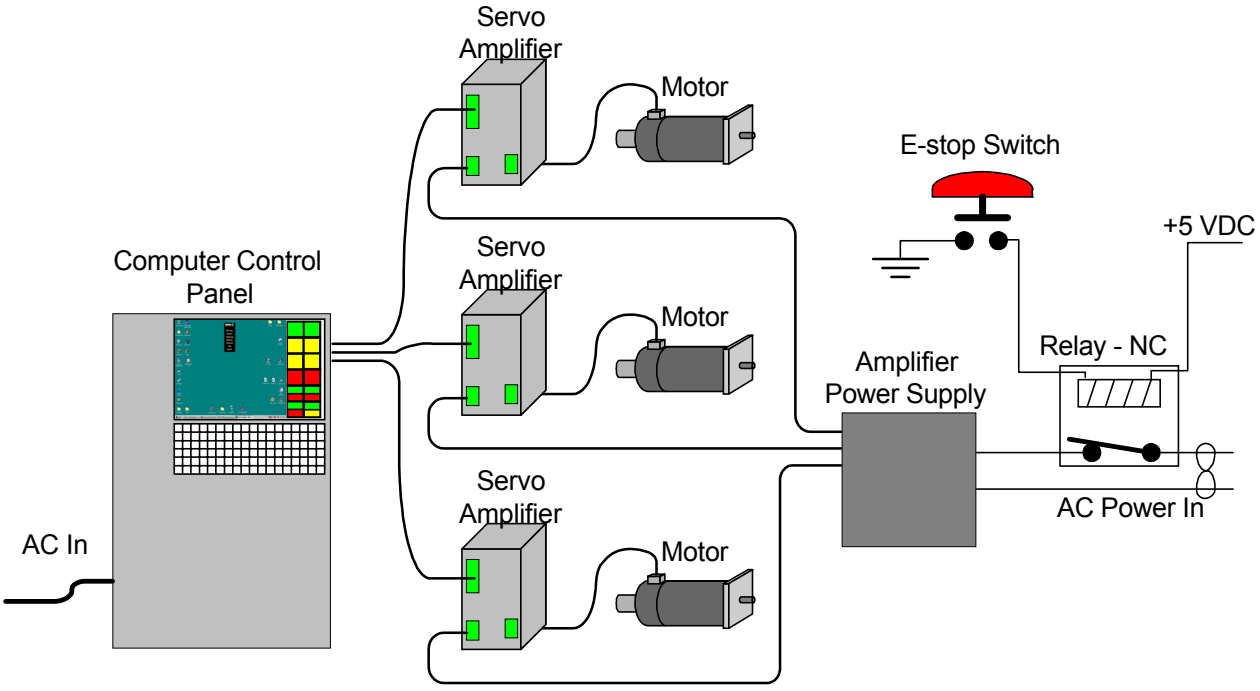
While backlash compensation is enabled, the response to the Tell Position, Tell Target and Tell Optimal commands will be adjusted to reflect the ideal positions (as if no mechanical backlash was present).

To disable backlash compensation, issue the Backlash compensation oFf (aBF) command. As soon as this command is executed, the motor will move to its' uncompensated position.



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.

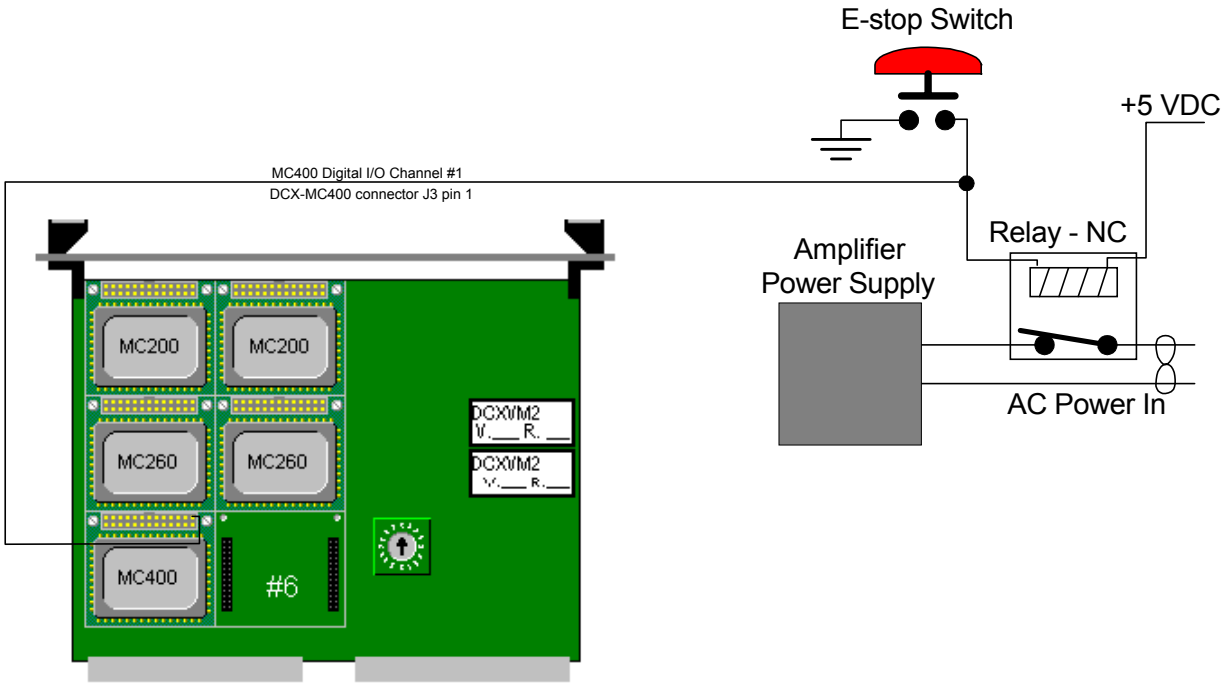


This 'hard wired' E-stop circuit uses a relay to disconnect power from the servo amplifiers. The motors and amplifiers would certainly be disabled, but the motion controller and the application program will have no indication that an error condition exists.

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 all of the Amplifier Fault (MC200 & MC210) inputs to the E-stop switch

E-stop switch connected to DCX General Purpose Digital Input
Wire the E-stop switch to a MC400 digital I/O (channel #1). Each DCX digital channel has a 4.7K resistor pulled up to +5 volts. A background task (macro #100 & 101) 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 Wait for channel oN (WNx) command will stay active in background until the input 'goes true'.



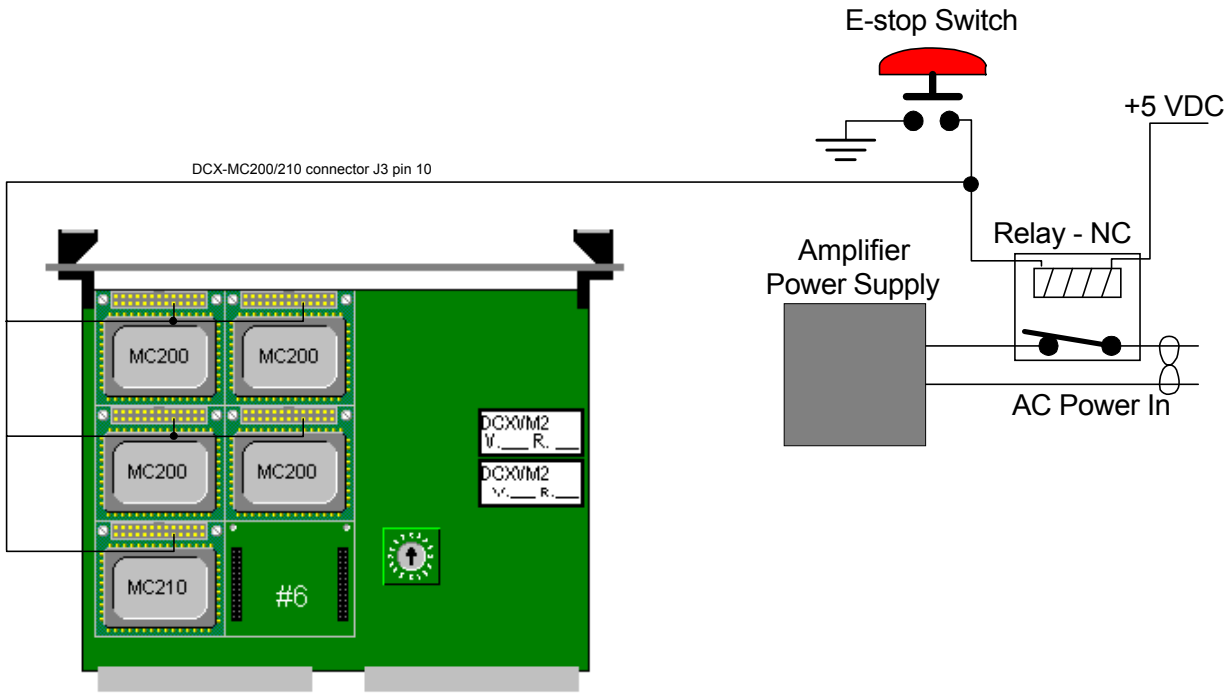

```
MD100,AL0,AR100,CI1,CL1,MJ101
MD101,WN1,AL1,AR100,1MF                 ;wait for E-stop = on

GT100                                   ;periodically poll user register #100
                                        ;for a value of 1.
```

E-stop switch connected to Amplifier Fault servo module input
The Amplifier Fault input of MC200 and MC210 servo modules can be used to disable motion with no user software action required. The E-stop switch is wired to the Amplifier Fault input (connector J3 pin

10) of **each servo module**. Auto shut down of motion upon activation of the Amplifier Fault input is enabled by the amplifier Fault input oN (aFN) command. When the E-stop switch is activated:

1) The axis is disabled (PID loop terminated, Amplifier Enable output turned off)
2) The status bit 26 (Amplifier fault) will be set for each axis
3) The status bit 7 (Motor Error) will be set for each axis

When the E-stop condition has been cleared, motion can be resumed after issuing the Motor oN (aMN) command to all axes.



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 35.8 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 (aVM) move.

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

Laser Cutting

For Laser cutting applications, The DCX can directly control the power of a laser. While executing contour move sequences, a MC200 generates a unipolar PWM output that is proportional to the vector velocity a contour move.

An additional MC200 module is used to generate the unipolar, TTL level, PWM output signal. In this mode of operation the Direction output of the MC200, available on connector J3 pin 7, is re-defined as PWM output. The signal define as Analog Output (J3 pin 2) is used for the direction signal.

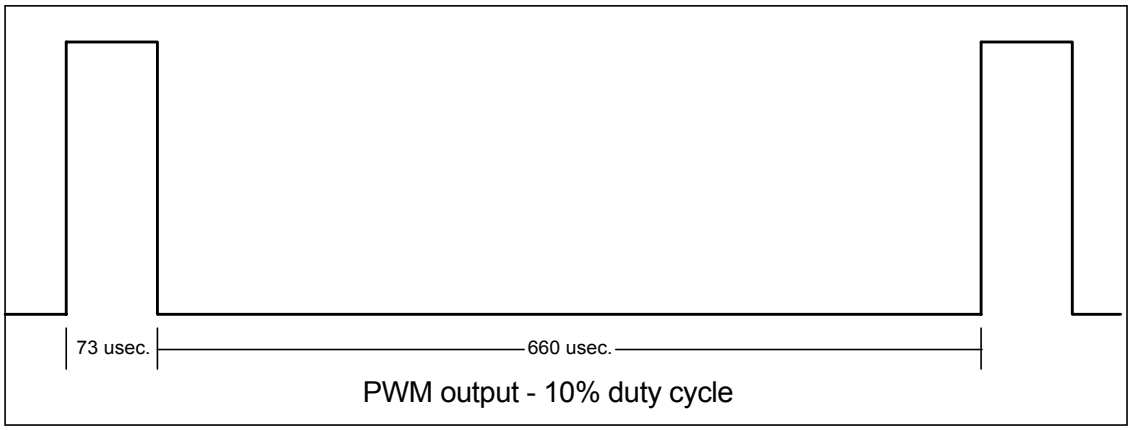
To enable the PWM output, the Output Mode command is issued with a parameter of n = 3. The motor module now outputs a PWM signal at a frequency of 1.4648 KHz. The PWM characteristics that must be defined by the user are Minimum Duty Cycle and Maximum Duty Cycle.

Minimum duty cycle defines the least amount of power that will be applied to the laser during a contour move. This value is typically specified as a percentage of the frequency of the PWM. The Minimum Duty Cycle (MinDC)is programmed using the Output Deadband command aODn where a is the axis number of the PWM module and n is the ‘scaled’ percentage of the PWM frequency. The following calculation is used to determine the value n for a minimum duty cycle of 10%:

n = PWM constant * desired minimum duty cycle
n = 10 * 10%
n = 10 * .1
n = 1
MinDC = aOD1

Based on the PWM frequency of 1.4648 KHz, the Minimum Duty Cycle Period (MDCP) will be:

MDCP = (PWM Frequency / 2) * 10%
= (1.4648 / 2) *10%
= 732.4 usec. * 10%
= 732.4 usec. * .1
= 73.24 usec.
MDCP = 73 usec. (rounded)



PWM output - 10% duty cycle

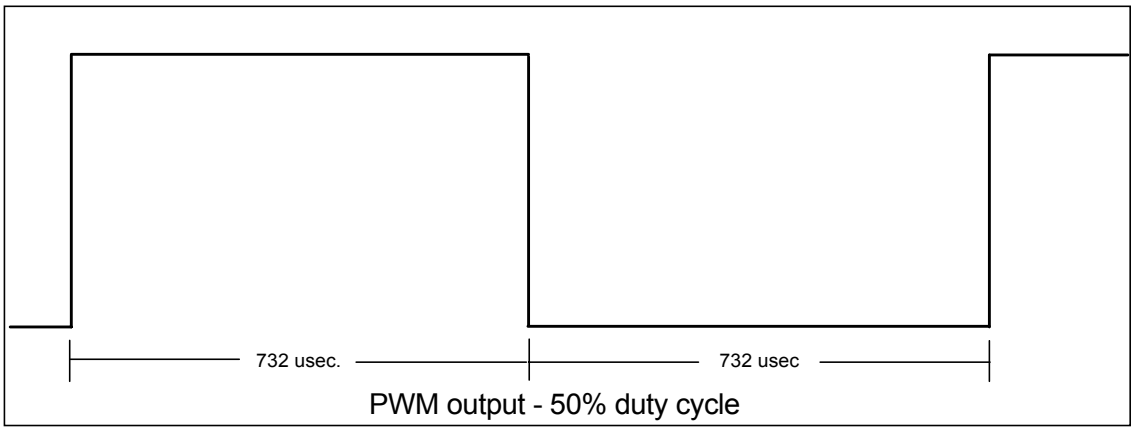
Selection of the Maximum Duty Cycle will be based on the specifications of the laser and the application specifics (motors, mechanics, thickness of material to be cut, etc…). This setting will determine the maximum power that will be applied to the laser. For this application example, the maximum duty cycle will be 50%. The Maximum Duty Cycle (MaxDC) is set by combining the Minimum Duty Cycle (MinDC) and the Remaining Duty Cycle (RDC).

MaxDC = MinDC + RDC
RDC = MaxDC – MinDC
RDC = 50% - 10%
RDC = 40%

Parameter n of the Velocity Gain command will be used to define the Remaining Pulse Period (RPP). For a 50% maximum duty cycle at a maximum vector velocity of 10,000 encoder counts per second the following calculation is used to determine the Velocity Gain parameter n :

n = (PWM Constant * RDC) / Maximum encoder Counts per second
n = (10 * 40%) / 10,000
n = (10 * .4) / 10,000
n = 4 /10000

```
aVGn = .0004
```



PWM output - 50% duty cycle

In the following example, axes 1, 2, and 3 are used to draw a triangle. Axis 4 is slaved to axis 1 (Contour move profiling axis)and will generate the PWM output signal with a frequency of 1.464 KHz. The resolution of the PWM is 16 bit. The PID loop gains (proportional, derivative, and integral)for axis

4 are set to 0. For this application the duty cycle of the PWM output must range from a minimum of 10% to 50%. The maximum vector velocity of the X, Y, & Z axes motion is 10,000 encoder counts per second.

```
1CM1,2CM1,3CM1                  ;Place X, Y and Z axes in contour mode
1VV10000,VA10000,VD10000        ;Define velocity profile (max 10000
                                ;counts/sec.)
4OM3                            ;Select PWM output for laser control
4SG0,SD0,SI0                    ;Turn off PID loop parameters
4SE0                            ;Turn off following error, axis will
                                ;be running open loop
4OB1.0                          ;Use the output deadband command to
                                ;define the minimum duty cycle. For
                                ;this calculation the value of Max.
                                ;Duty Cycle Constant = 10. 10% duty
                                ;cycle is 10*10%=1.
4VG0.0004                       ;Use velocity gain + minimum duty
                                ;cycle to define the maximum duty
                                ;cycle.
4MN                             ;Turn axis (laser) on, power = 0
4SS1.0                          ;Set slave ratio to 1.0
4SM1                            ;Define axis 4 as slave to axis 1
                                ;(which must already be in contour
                                ;mode as the profiling axis)
1CP1,1MR50000                   ;Draw Triangle
1CP1,2MR50000                   ;  "     "
1CP1,1MR-50000,2MR-50000        ;  "     "

4SM0                            ;Disconnect slave axis from contour

4MF                             ;Turn axis (laser) off
```

Learning / Teaching Points

As many as 256 points can be stored for ***each axis*** in the DCX's point memory by using the Learn Point (aLPn) and Learn Target (aLTn) commands.

The Learn Point command would typically be used in conjunction with jogging. The operator would jog the axes along the desired path, issuing the Learn Point command at regular intervals. The Move Point command would then be used to 'play back' the path traversed by the operator.

For applications where the target point data has previously recorded and stored in the 'PC', the Learn Target command would be used to load the target points into the DCX. For some applications, using the Learn Target 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 the points have been stored using the Learn Point or Learn Target commands, the axes can then be commanded to move to those positions by issuing Move to Point command.
The parameter n of the Move to Point command should be the same as was used by the 'learn' command.

```
MF                                  ;turn off the motors
1MA0,2MA0,0LT1                      ;learn target point #1
1MA1000,2MA2000,0LT2                ;learn target point #2
1MA3000,2MA3000,0LT3                ;learn target point #3
1MA2000,2MA1000,0LT4                ;learn target point #4
1MN,2MN,AL1                         ;turn motors on, load accumulator (1)
1MP@0,2MP@0,0WS0.1,AA1,RP3          ;move axes 1 & 2 to the first 'learned
                                    ;point'. The learned point 'pointer'
                                    ;is stored in the accumulator.
                                    ;Increment the pointer, repeat three
                                    ;more times (moving through
                                    ;points 2 - 4)
```

This example begins by issuing a motor off command to the DCX, otherwise the motor would actually move to the target position. Move commands are then issued to axes 1 and 2. No motion will occur because the motors are in the off state, but the target positions will be updated. After each move, a Learn Target command is issued with an axis specifier of 0. This will cause the respective motor target positions to be stored in the DCX's point memory. After teaching 4 position pairs, the motors are turned back on and a command sequence that will move axes 1 and 2 to positions 1,2,3 and 4 is issued. The Wait for Stop command will cause the axes to 'dwell' for 1 tenth second at each position before continuing to the next point. The axes will move with the accelerations and velocities that were set for those axes prior to the MP command being issued.

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 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.

```
1CM1,2CM1                           ;place axes 1 & 2 in contour mode
1VV100000,1VA100000,1VD100000       ;define the contouring trajectory
1MP1,1MP2,1MP3,1MP4                 ;execute linear contour move through
                                    ;points 1 - 4
```

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 and 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
```

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.

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

Record and Display Motion Data

The DCX supports recording and reporting of the following motion data for servo axes (MC200, MC210):

Function	*Frequency **	*Units*	*Command*	*Parameter range*
Record the motion data of an axis	2KHz, 1 KHz, .5 KHz	Encoder	aPRn	1 <= n <= 512
Display the actual position of an axis	2KHz, 1 KHz, .5 KHz	Encoder	aDRn	1 <= n <= 512
Display the optimal position of an axis	2KHz, 1 KHz, .5 KHz	Encoder	aDOn	1 <= n <= 512
Display the DAC output of an axis	2KHz, 1 KHz, .5 KHz	Volts	aDQn	1 <= n <= 512

* The frequency of the data recording is a function of the programmed servo loop rate (2KHz = HS, 1 KHz = MS, 0.5 KHz = LS).

To record the motion data for an axis, issue the Position Record (aPRn) command where a = the axis number for which data is to be acquired. Parameter n defines the quantity of data captures to execute:

```
1MR10000                                ;move axis one 10,000 encoder counts
1PR100                                  ;capture 100 samples of motion data
                                        ;for axis 1
```

To record motion data for all installed axes issue the Position Record command with an axis specifier of 0.

```
0MR10000                                ;move all axes 1000 counts
0PR256                                  ;capture 256 samples of motion data
                                        ;for all axes
```

The following commands will report captured data:

```
1DR0,1DR1,1DR2                          ;Display the first three recorded
                                        ;positions of axis 1
2DO10,2DO11,2DO12                       ;Display recorded optimal positions
                                        ;10, 11, & 12
```

The following command sequences will display the first 10 recorded positions of axis one:

```
Al0,ar100                               ;zero register 100 (recorded position
                                        ;pointer)
1MR1000,1PR100                          ;move 1000 counts, record 100 data
                                        ;points
1DR@100,AL@100,AA1,AR100,RP9            ;display first position, increment
                                        ;position pointer, repeat nine times
```

Single Stepping MCCL Programs

While the DCX is executing any Motion Control Command Language (MCCL) macro program, the user can enable single step mode via the ASCII interface 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.

To begin single step execution of the above example macro enter MC10 (call macro #10) then <ctrl> <B>, the DCX will respond with the following reply:

{C1,MC10} 1MR1000 <

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

```
MC10   <return>                         ;start the macro routine
                    <ctrl> <B>
{C1,M10}      1MR1000             <ctrl> <B>
(C2,M10}      1WS0.10000          <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                 <return>
```

To end single stepping 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.

Note: Firmware revision 3.5b or higher is required for single step mode

Tangential Knife Control

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 User Scaling (aUSn) 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 direction of the master axes 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.

```
3US2000                                 ;define scaling of axis 3, 2000
                                        ;encoder counts per revolution sets
                                        ;the scaling to 1 = 1 revolution
3SM33                                   ;define axes 1 & 2 as masters for axis
                                        ;3
1MN,2MN,3MN,4MN                         ;turn on axes 1, 2, 3, & 4
1CP1,1MA10000,2MA0                      ;Execute 1st linear move
1WS.1,4MR1000,4WS.1,3MR.25,3WS.1,4MR-1000
                                        ;wait for end of contour move, lift
                                        ;blade, rotate blade, lower blade
1CP1,1MA10000,2MA-10000                 ;Execute 2nd linear move
1WS.1,4MR1000,4WS.1,3MR.25,3WS.1,4MR-1000
                                        ;wait for end of contour move, lift
                                        ;blade, rotate blade, lower blade
1CP1,1MA0,2MA-10000                     ;Execute 3rd linear move
1WS.1,4MR1000,4WS.1,3MR.25,3WS.1,4MR-1000
                                        ;wait for end of contour move, lift
                                        ;blade, ;rotate blade, lower blade
1CP1,1MA0,2MA0                          ;Execute 4th linear move
1WS.1,4MR1000,4WS.1,3MR.25,3WS.1,4MR-1000
                                        ;wait for end of contour move, lift
                                        ;blade, rotate blade, lower blade
```

Threading Operations

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 implementation of threading utilizes 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 Set Master command has been issued to the slave axis. The slave will be synchronized to the master's position when its' encoder index pulse occurs. In the following example the spindle (master) is axis #2 and the thread cutting tool is positioned by axis #1 (slave).

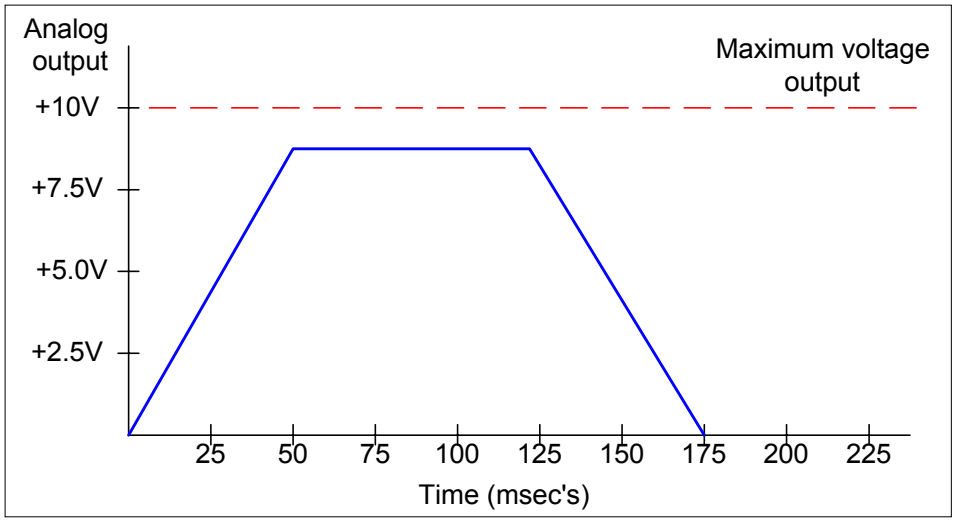
```
2US2000,MN                      ;Set scaling of master axis. For the
                                ;spindle, this would typically be set
                                ;to the number of encoder counts per
                                ;revolution.
1US4000,MN                      ;Set user scaling of slave axis.
1MA0,WS                         ;Position slave axis to the desired
                                ;starting point.
1SS0.1                          ;Set the slave ratio. This is the lead
                                ;or pitch when cutting a thread.
1SM258                          ;Define axis #1 as slave, axis #2 as
                                ;master. Enable threading by
                                ;n = 2 + 256
1MA1.0                          ;Set the target position. This is the
                                ;position at which slave mode is
                                ;terminated and axis #1 will stop.
2SQ3.0,QM                       ;Start master axis moving in torque
                                ;mode.
2IA                             ;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.

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

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

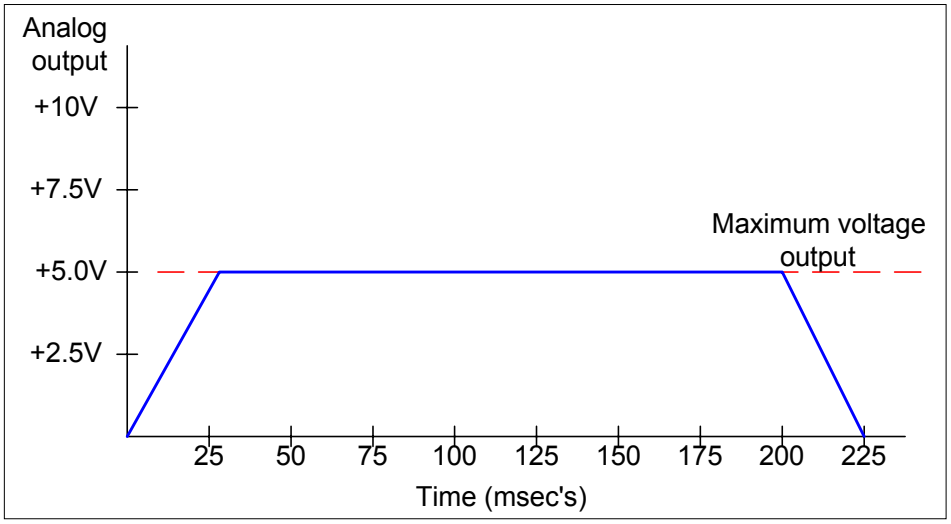
Bit 22 = Axis is slaved to master's encoder position
Bit 23 = Axis is slaved and waiting for master's index mark

Torque Mode Output Control

The DCX servo modules (MC200 & MC210) 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 Set torQue (aSQn) command allows the user to limit the output signal (or duty cycle) to a specific level. The following graph depicts a simple position mode move of 1000 encoder counts with the default torque setting of 10 volts (no limit).



The graphic below depicts the same 1000 encoder count move, but the maximum voltage output has been limited to 5.0 volts (1SQ5).



Servo Modules as simple D/A or PWM output with encoder reader
Torque mode (aQM) 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.

```
1QM,1SQ2.5                          ;set axis #1 analog output to 2.5
                                    ;volts (MC200)
2QM,2SQ7.5                          ;set duty cycle to 75% (MC210)
```

Defining User Units

When the DCX is initially turned on or 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 parameter of 1000, the DCX will move the servo 1000 encoder counts. If the user issues the same command to a stepper motor, the DCX will move it 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.

In order to setup the DCX to accept command parameters in user units, and to respond to reporting commands in user units, there are 2 variables that should be initialized for each servo or stepper axis. They are the User Scale and the User Rate Conversion. The commands for setting these variables are set User Scale (aUSn) and set User Rate Conversion (aURn) respectively. Each of these commands will accept integers, fixed point or floating point number as command parameters. Internally, these variables are stored as double precision (64 bit) floating point numbers.

Setting Move (Encoder/Step) Units
The parameter to the Set User Scale command is the number of encoder counts or steps per user unit. For example, if the servo encoder on axis 1 has 1000 quadrature counts per rotation, and the mechanics move 1 inch per rotation of the servo, then to setup the controller for user units of inches, the command 1US1000.0 should be issued.

Prior to issuing the Set User Scale command, the parameters to all motion commands for a particular axis are rounded to the nearest integer. After the Set User Scale and Motor oN commands are issued, motion command parameters are multiplied by the ratio prior to rounding to determine the correct encoder position. When the user issues a reporting command to an axis, if a User Scale has been set, the encoder position is divided by the ratio to calculate the position in user units to be displayed. Note – setting a user scale other than 1:1 will also scale trajectory settings (Velocity, acceleration, and deceleration) but not PID settings.

```
1US1000.00                          ;define the user scale of axis 1 to
                                    ;1000:1
1MN                                 ;turn the axis on to cause the new
                                    ;scaling to take effect
1MA2.5,1WS.1                        ;move to position 2.5 (2.5 X 1000 =
                                    ;2500 encoder counts
1TT.3                               ;report the actual position with 3
                                    ;digits to the right of the decimal
                                    ;point
```

Trajectory Time Base
The Set User Rate command can be used to change the default seconds time unit for velocity, acceleration and deceleration values, to a time unit selected by the user. If velocities are to be in units

of inches per minute, the user time unit is a minute. The parameter to the Set User Rate command is the number of seconds per 'user time unit'.

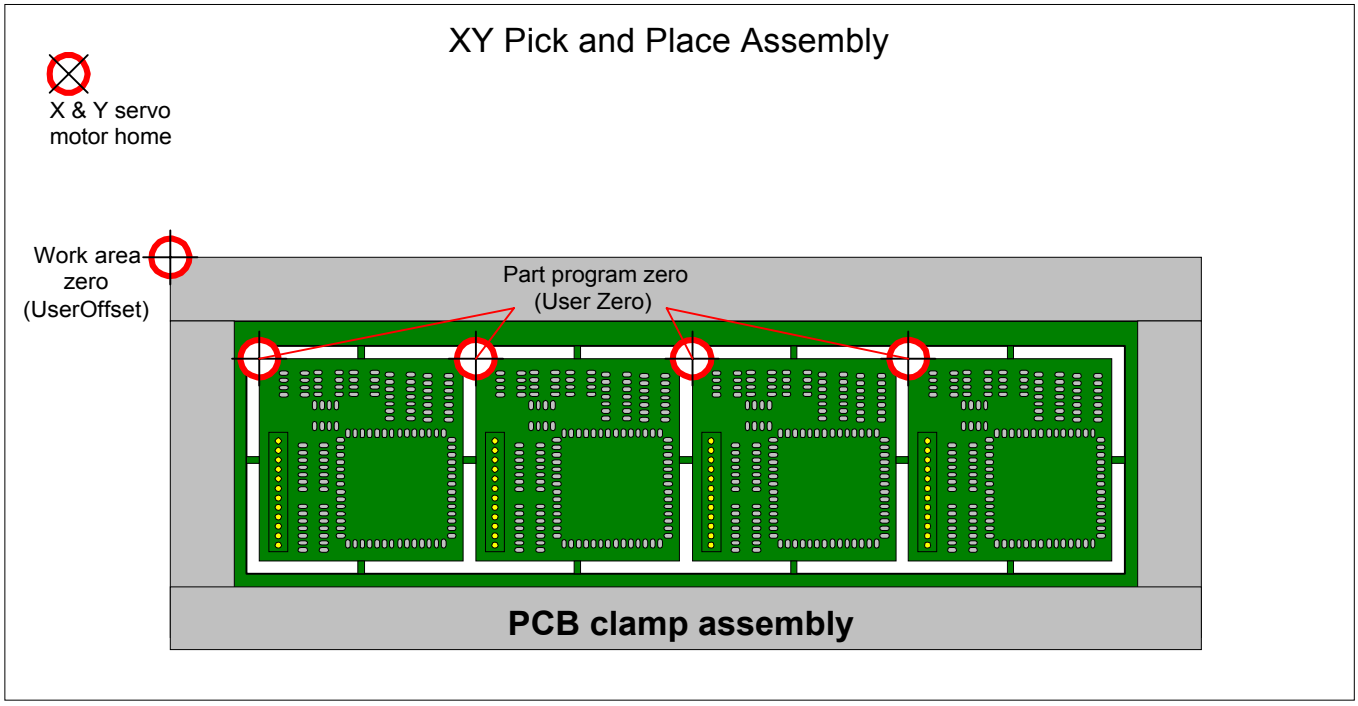
As an example of the User Rate command, if the velocity, acceleration and deceleration are to be specified in units of inches per minute and inches per minute per minute for axis 1, then the user rate conversion variable should be set to 60 seconds / 1 minute = 60 (1UR60). The Motor oN (aMN) command must be issued before the user rate will take effect.

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

Defining the Time Base for Wait commands
For the WAit, Wait Stop and Wait Target commands, the default units for their parameters are seconds. By setting the 'User Time' variable appropriately, these two commands can be issued with parameters in units of the user's preference. This variable can be set with the Set User Time command. The parameter to this command is the number of 1 second periods in the user's unit of time. If the user prefers time parameters in units of minutes, a UT60 command should be issued. Since there is only one User Time variable that is used for all axes, the set User Time command needs to be issued only once with no axis specified.

Defining a System/Machine zero
The set User Offset (aUOn) command allows the user to define a ‘work area’ zero position of the axis. The parameter to the set User Offset command should be the distance from the servo or stepper motor home position, to the machine zero position. If units of this command must be the same as currently defined by set User Scaling command. The set User Offset 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.

Defining a Part Zero
The set User Zero (aUZn) command would typically be used in conjunction with the set User Offset to define a ‘part zero’ position. A PCB (Printed Circuit Board) pick and place operation is a good example of how this command would be used. After a new PCB is loaded and clamped into place the X and Y axes would be homed. The set User Offset command would then be used to define the ‘work area’ zero of the PCB. The set User Zero command would then be used to define the ‘part program’ 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.



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

DCX Watchdog

The DCX incorporates a watchdog circuit to protect against improper CPU operation. If the DCX processor fails to properly execute firmware code for a period of 100 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, PWM, Step/Direction) will be disabled.

When the watchdog circuit has tripped, the two yellow LED's (L2 and L3) will be turned on. To clear the watchdog error either:

Cycle power to the computer ***(recommended)***
Reset the computer
Manually reset the DCX

Note: If the watchdog trips while a MCAPI based application program is running, manually resetting the DCX will **probably not** allow the application program to continue operation.

Chapter Contents

Chapter 7

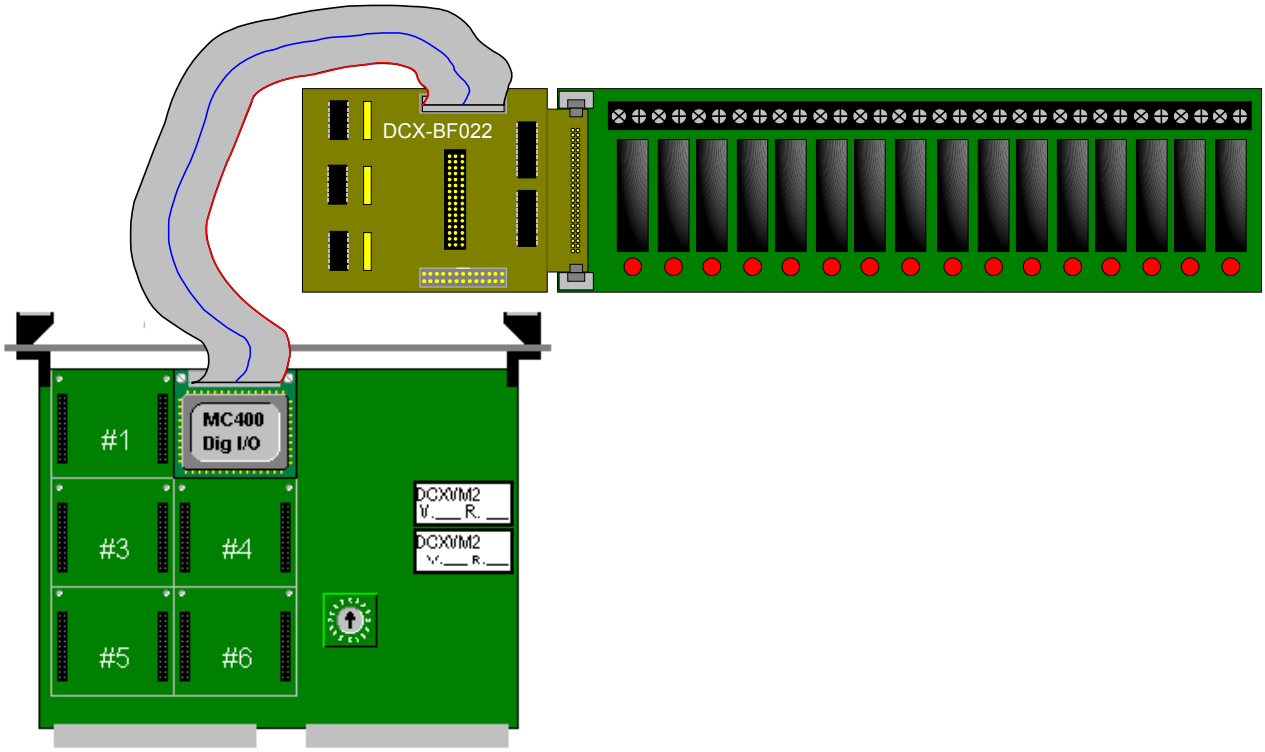
DCX General Purpose I/O

DCX Digital I/O

When installed on a DCX-VM200 motherboard, the DCX-MC400 Digital I/O provides 16 undedicated digital I/O channels. Each digital channel is user configured via software (input, output, high true, low true). The I/O channels can be accessed from the 26 pin header on the module or from the VME back plane (via connector P2). The **DCX-MC400** section of the **Connectors, Jumpers, and Schematics** chapter provides the connector pin-out for this module.

Interfacing to the 'Outside World'
The TTL digital I/O channels can be connected directly to the external circuits if output loading (1ma maximum sink/source) and input voltages (0.0V to +5.0V) are within acceptable limits. Alternatively, a DCX-BFO22 interface board can be used to connect the MC400'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-MC400 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.



As shown above, the DCX-BFO22 plugs directly into the relay rack's 50 pin header connector and then connects to the DCX-MC400 via a 26 conductor ribbon. 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 **DCX-BF022** section of the **Connectors, Jumpers, and Schematic** chapter.

Configuring the DCX Digital I/O

Configuring the Digital I/O channels
The configuration of the DCX-MC400 digital I/O channels is accomplished via software (MCCL) commands. Each channel is individually programmable as:

Input (CIx) or Output (CTx)
High true (CHx) or Low true (CLx)

Each of the digital channels commands require a channel number (1 – 16) as parameter *x*. If more than one MC400 is installed, additional I/O channel numbers (e.g. 17-32, 33-48, etc.) are assigned to

succeeding MC400's in blocks of 16. The MC400 installed in the lowest module location (1 – 6) will control channels 1 through 16.

```
CT1,CT2,CT3,CT4,CT5,CT6,CT6,CT8,CT9,CT10      ;define the output channels
CI11,CI12,CI13,CI14,CI15,CI16                 ;define the input channels
CH1,CH2,CH3,CH4,CH11,CH12,CH13,CH14           ;define all channels that are
                                              ;high true
CL5,CL6,CL7,CL8,CL9,CL10,CL15,CL16            ;define all channels that are
                                              ;low true
```

Note – If a BFO22 interface and relay rack are connected to the DCX Digital I/O, a global Channel Low **(CL0)** command 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.

For input modules, the Tell Channel command will return a "1" when the signal is present, and a "0" when it is not. By default, all I/O channels on the DCX are changed to inputs during power-up or reset. For input modules installed in the relay rack, no further configuration is necessary.

After configuring the Digital I/O channels, the following commands are used to report the state of a channel (input or output) and to turn the channel on or off (outputs only):

turn off the Digital output

MCCL command: CFx
parameter: integer (1 <= x)
compatibility: MC400
see also: CH, CI, CL, CN, CT, TC

explanation: Causes digital I/O 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 than 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).

turn on the Digital output

MCCL command: CNx
parameter: integer (1 <= x)
compatibility: MC400
see also: CF, CH, CI, CL, CT, TC

explanation: 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).

report the state of a digital Channel

MCCL command: aTCx

parameter: integer (0 < x <= 16 for motherboard channels)
compatibility: MC400
see also: CH, CI, CL, CT

explanation: 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".

DCX Module Analog I/O

The DCX-MC500 Analog I/O Module provides 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. When installed on the DCX-VM200, since there are already 4 analog input channels on the motherboard, the analog I/O channels on the DCX-MC500's start with number 5.

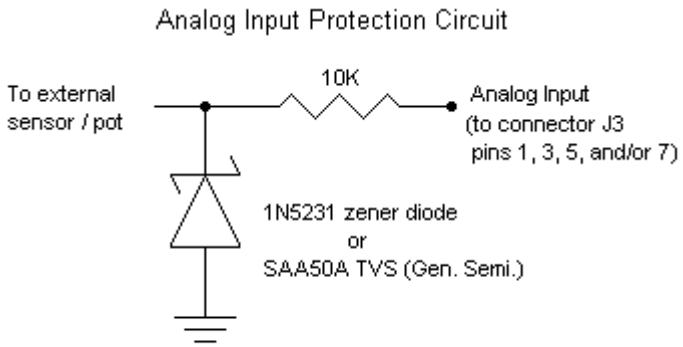
Because the DCX controller board is implemented in digital electronics, all analog input signals must be converted into a representative numerical value. This function is done by an Analog to Digital Converter (ADC) on the DCX-MC500. Similarly, analog output signals originate on the DCX board as numerical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



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

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

The outputs from the DAC are also connected to operational amplifiers on the module which offset and amplify them to provide a -10 to +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 ñ0.5 volt adjustment, and the gain pot provides a nominal ñ2% 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.

Calibration:
The analog input of the DCX-MC500 needs has no adjustments, 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.

Calibration of the analog outputs with the -10 to +10 volt range is accomplished using the 8 trim pots located on the module. 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. Refer to the module layout diagram at the end of the appendix describing the MC500 module for pot locations and assignments.

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.

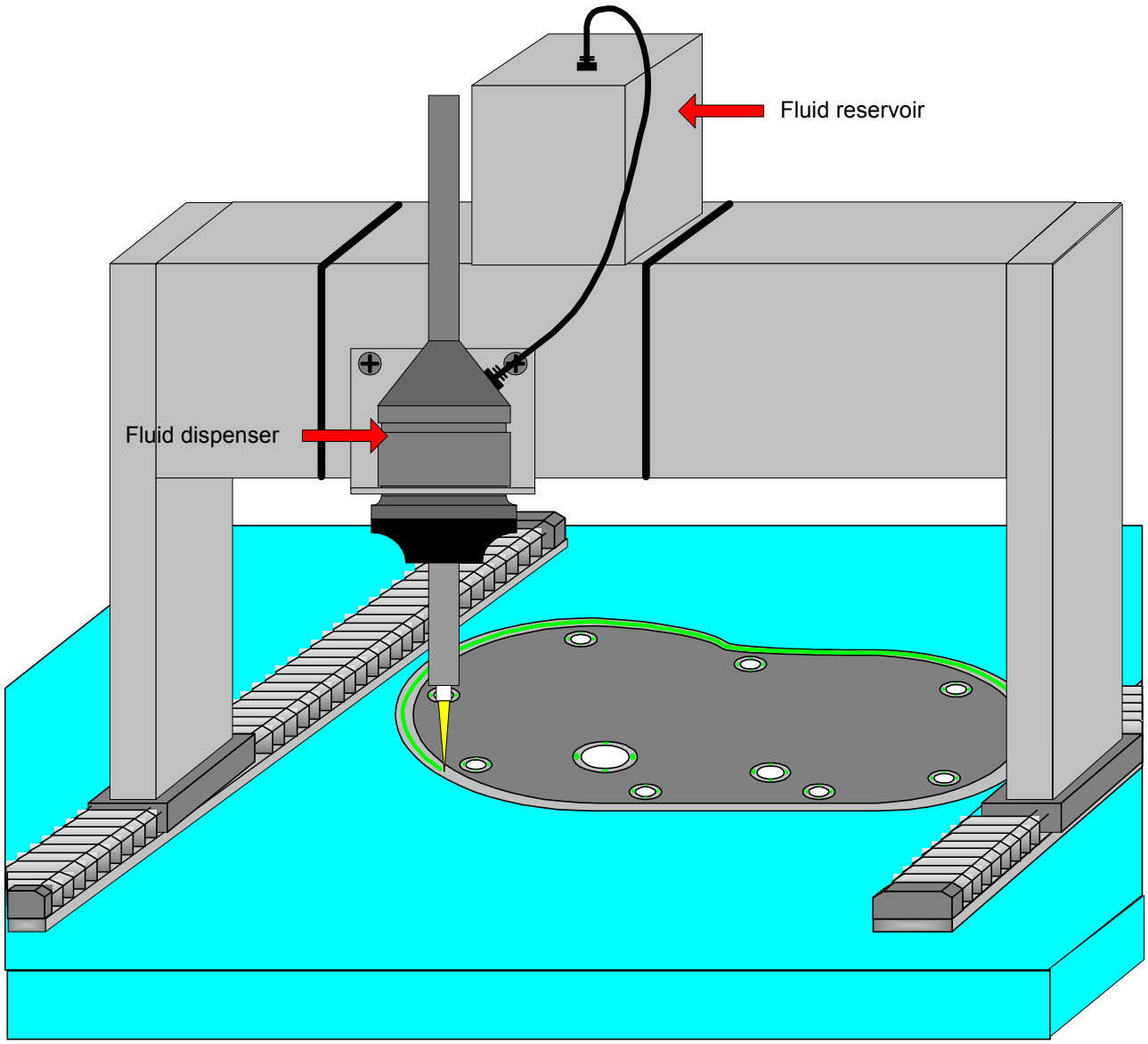
After the second step of the calibration procedure, the output swing should still be 20.00 volts. If not, repeat steps 1 and 2 again.

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 dispense 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:
Fluid reservoir empty (dig. I/O #1)
Dispense valve empty (dig. I/O #2)
Valve busy being primed (dig. I/O #3)

Outputs:
Turn on the dispense valve air supply (dig. I/O #4)
Dispense liquid – turn on Archimedes motor (dig I/O #5)
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 then 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
```

The following commands are used to program conditional (If / Then) command execution:

Execute remaining command sequence if Digital channel ‘x’ is off

MCCL command: DFx
parameter: integer (1 <= x)
compatibility: MC400
see also: DN, IF, IN

explanation: Used for conditional execution of commands. If the specified digital I/O channel 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.

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

Execute remaining command sequence if Digital channel ‘x’ is on

MCCL command: DNx
parameter: integer (1 <= x)
compatibility: MC400
see also: DF, IF, IN

explanation: Used for conditional execution of commands. If the specified digital I/O channel 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.

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

if Digital channel ‘*x*’ is off execute the next command, else skip 2 commands

MCCL command: IFx
parameter: (1 <= x)
compatibility: N/A
see also: DF, DN, IN

explanation: Used for conditional execution of commands. If the specified digital I/O channel 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.

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

if Digital channel ‘*x*’ is on execute the next command, else skip 2 commands

MCCL command: INx
parameter: (1 <= x)
compatibility: N/A
see also: DF, DN, IF

explanation: Used for conditional execution of commands. If the specified digital I/O channel 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.

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

Wait for digital channel off

MCCL command: WFn
parameter: (1 <= x)
compatibility: MC400
see also: WN

explanation: 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.

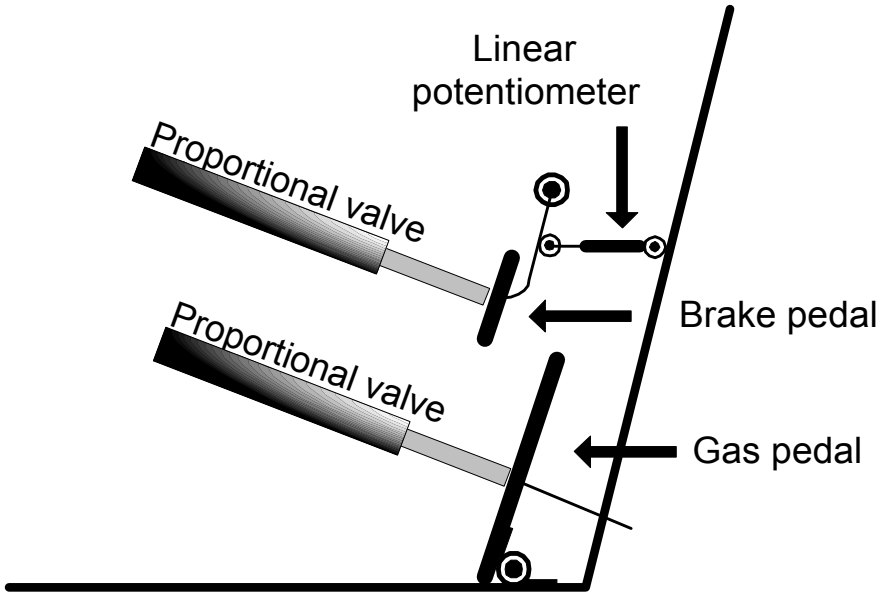
Wait for digital channel on

MCCL command: WNn
parameter: (1 <= x)
compatibility: MC400
see also: WF

explanation: 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.

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 ‘gas pedal’ to control the speed of the vehicle. The other is positioned to depress the ‘brake pedal’. An analog tachometer is connected to the drive train to provide the feedback of the speed of the vehicle. 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,AE3,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,AE3,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
```

Chapter Contents

Chapter 8

Working With DCX Data

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 in 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 Accumulator Load (ALn) 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 = ACCn)
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)

Reading Data From Memory

A group of read commands are available for accessing the DCX's internal memory. These commands provide an easy method of moving motor data in and out of the Accumulator (user register 0). To use a read command for this purpose, it should include an axis specifier 'a' and a parameter 'n' selected from the motor table offsets listed below. 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.

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

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

```
   2RL0                                ;load the status of axis #2 into the
accumulator
```

To load the position of axis 3 the accumulator, issue the following command:

```
   3RD20                          ;load the position of axis #3 into the
accumulator
```

Memory read/write commands:

aRBn Read Byte (8 bit) at memory location n into accumulator (ACC = (n))
aRDn Read Double at memory location n into accumulator (ACC = (n))
aRLn Read Long (32 bit) at memory location n into accumulator (ACC = (n))
aRVn Read float at memory location n into accumulator (ACC = (n))
aRWn Read Word (16 bit) at memory location n into accumulator (ACC = (n))

Motor Table Entries – 32 bit integer (long)

Motor Table Entry Description	*Offset (decimal)*
Motor Status	0
Position Count	4
Optimal Count	8
Index Count	12
Auxiliary Status	16
Module Base Address	252

Motor Table Entries – 64 bit floating point (double)

Motor Table Entry Description	*Offset (decimal)*
Position	20
Target	28
Optimal Position	36
Breakpoint Position	44
Position Deadband	52
Maximum Following Error	60
Soft Motion Limit Setting (low)	68
Soft Motion Limit Setting (high)	76
User Scale	84
User Zero	92
User Offset	100
User Rate Conversion	108
User Output Constant	116
Programmed Velocity	124
Programmed Acceleration	132
Programmed Deceleration	140
Minimum Velocity	148
Minimum Velocity	156
Jog Acceleration	220
Jog Minimum Velocity	228

Motor Table Entries – 32 bit floating point (float)

Motor Table Entry Description	*Offset (decimal)*
Velocity Gain	164
Acceleration Gain	168
Deceleration Gain	172
Velocity Override	176
Torque Limit	180
Proportional Gain	184
Derivative Gain	188
Integral Gain	192
Integration Limit	196

Module Analog Input 1	200
Module Analog Input 2	204
Jog Gain	208
Jog Offset	212
Jog Deadband	216

Motor Table Entries – 16 bit integer (word)

Motor Table Entry Description	***Offset (decimal)***
Wait Stop Timer	236
Wait Target Timer	238
Sampling Frequency	240
Master Axis	242
Module Status	244
Axis Number	246
Module Position	248
Module Type	250

Dual Ported Memory

When a DCX board is plugged in to an 'PC' compatible computer, depending on the setting of jumper JP8 and rotary switch SW1 on the DCX, a 4096 byte portion of the board's dual ported memory will be accessible somewhere in the PC's memory space. The location of this memory (defined by JP8), can be configured to be anywhere in the range 80000 hex to FFFFF hex.

JP8 - IBM-PC Interface base memory address select

Base address	***JP8 5 to 6***	***JP8 3 to 4***	***JP8 1 to 2***
80000 hex	connected	connected	connected
90000 hex	connected	connected	open
A0000 hex	connected	open	connected
B0000 hex	connected	open	open
C0000 hex	open	connected	connected
D0000 hex	**open**	**connected**	**open**
E0000 hex	open	open	connected
F0000 hex	open	open	open

In most PC configurations, the 64K area from D0000 hex to DFFFF hex is available and has been chosen as the factory default memory range for the DCX. Most other ranges that can be set with jumper JP8 will result in hardware conflicts within the PC when the DCX is installed. Unless the user determines that another range is acceptable, jumper JP8 should be left at its default setting as shown in appendix B.

Assuming that jumper JP8 is at its factory default setting, the DCX will occupy a 4096 bytes somewhere between D0000 hex and DFFFF hex in the PC's memory space. If the rotary switch SW1 is set to 0, the DCX will occupy D0000 hex through D0FFF hex. If it

is set to 1, it will occupy D1000 hex through D1FFF hex, and so on. In the PC's memory map, the lowest numbered memory location that a DCX board occupies is referred to as the 'base' address. If the board's rotary switch is set to 0, its base address will be D0000h. The base address should be added to all addresses listed in this appendix in order to calculate the physical address in the PC's memory map.

Motor Table Addresses:

Axis #	*Motor Table Base Address*
1	0h
2	100h
3	200h
4	300h
5	400h
6	500h

Example: If the rotary switch is to 0, the position for axis 3 will be at D0214 hex.

Motor Table Data:

Description	*Data type*	*Offset*
Motor Status	unsigned integer	0 (0 hex)
Position Count	integer	4 (4 hex)
Optimal Count	integer	8 (8 hex)
Index Count	integer	12 (C hex)
Auxiliary Status	unsigned integer	16 (10 hex)

Description	*Data type*	*Offset*
Position	double	20 (14 hex)
Target	double	28 (1C hex)
Optimal Position	double	36 (24 hex)
Breakpoint Position	double	44 (2C hex)
Position Deadband	double	52 (34 hex)
Maximum Following Error	double	60 (3C hex)
Low Limit of Movement	double	68 (44 hex)
High Limit of Movement	double	76 (4C hex)
User Scale	double	84 (54 hex)
User Zero	double	92 (5C hex)
User Offset	double	100 (64 hex)
User Rate Conversion	double	108 (6C hex)
User Output Constant	double	116 (74 hex)
Programmed Velocity	double	124 (7C hex)
Programmed Acceleration	double	132 (84 hex)
Programmed Deceleration	double	140 (8C hex)
Minimum Velocity	double	148 (94 hex)
Current Velocity	double	156 (9C hex)

Description	*Data type*	*Offset*
Velocity Gain	float	164 (A4 hex)
Acceleration Gain	float	168 (A8 hex)
Deceleration Gain	float	172 (AC hex)
Velocity Override	float	176 (B0 hex)
Torque Limit	float	180 (B4 hex)
Proportional Gain	float	184 (B8 hex)
Derivative Gain	float	188 (BC hex)
Integral Gain	float	192 (C0 hex)
Integration Limit	float	196 (C8 hex)
Module Analog Input 1	float	200 (C4 hex)
Module Analog Input 2	float	204 (CC hex)
Jog Gain	float	208 (D0 hex)
Jog Offset	float	212 (D4 hex)
Jog Deadband		216 (D8 hex)
Jog Acceleration		220 (DC hex)
Jog Minimum Velocity		228 (E4 hex)

Description	*Data type*	*Offset*
Wait Stop Timer	short	326 (EC hex)
Wait Target Timer	short	238 (EE hex)
Sampling Frequency	short	240 (F0 hex)
Master Axis	short	242 (F2 hex)
Module Status	short	244 (F4 hex)
Axis Number	short	246 (F6 hex)
Module Position	short	248 (F8 hex)
Module Type	short	250 (FA hex)
Module Base Address	unsigned integer	252 (FC hex)

where:

double = 64 bit floating point format
float = 32 bit floating point format
integer = 32 bit integer format (2's complement)
u.integer = 32 bit unsigned integer format
short = 16 bit integer format

Motor Status Bit Definitions:

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	Motor Jogging is Enabled
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	Unused
11	Unused
12	Breakpoint Reached (IP, IR, WP, WR)
13	Exceeded Max. Following Error *
14	Amplifier Fault Enabled *
15	Amplifier 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 / Stepper Home
25	Coarse home (current state)
26	Amplifier Fault *
27	Auxiliary Encoder Index
28	Limit Positive Input Active (current state)
29	Limit Negative Input Active
30	User Input 1 *
31	User Input 2 *

* not valid for stepper modules

General use variables:

Description	*Offset*
ASCII interface input mailbox	800h
ASCII interface output mailbox	804h
Command Interpreter status	808h
Position of Motor 1 Module	80Ah
Position of Motor 2 Module	80B4h
Position of Motor 3 Module	80Ch
Position of Motor 4 Module	80Dh
Position of Motor 5 Module	80Eh
Position of Motor 6 Module	80Fh
Type of Module in Position 1	812h
Type of Module in Position 2	813h
Type of Module in Position 3	814h
Type of Module in Position 4	815h
Type of Module in Position 5	816h
Type of Module in Position 6	817h
Digital I/O Channels 1-16	81Ah*
Analog Input Channel 1	81Ch*
Analog Input Channel 2	81Eh*
Analog Input Channel 3	820H*
Analog Input Channel 4	822H*

* Value updated every millisecond

Module type ID codes:

ID code	*Module type*
0	MC200 Advanced Servo
1	MC260 Advanced Stepper
8	MC400 Digital I/O Module
9	MF310 GPIB Communications
10	MF300 RS-232 Communications
12	MC500 Analog I/O Module
15	No module present
16	MC210 Advanced servo, motor output

Scratch Pad Memory

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 MEmory allocate (MEn) 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 Free Memory (FMn) 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 9

DCX Command Set

A powerful set of over 200 commands has been implemented on board the DCX motion controller. The following sections describe each command in detail. An example command description is shown below:

Move the axis a relative distance

parameter: integer or real
compatibility: MC200, MC210, MC260

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

```
1MR10000                                ;Move axis 1 motor 10000 units
                                        ;relative to its' current position
```

The "***parameter***" line in the description specifies the acceptable values, or range of values, for the command parameter. If no parameter is specified the controller defaults to using a parameter of 0. If a commands' syntax doesn't show a parameter value, including one when the command is issued won't cause an error, and the value will be ignored.

The "***compatibility***" line will list which DCX plug-in modules for which the command is valid. This is important since not all parameter setup and motion commands are supported on both servo and stepper axes. If a command isn't intended to be used for a specific axis or module, the letters 'N/A' for "Not Applicable" will appear on this line.

The "***see also***" line lists other commands that have an association or similar purpose to the command being described, their MCCL mnemonics will appear on this line .

The command "***explanation***" section provides information about the purpose of a command and how it is used. This section will optionally be followed by a command example. The characters that are sent to the controller in the examples are shown in a line printer typeface.

If there are any special notes about a command or how it is used, this will be included in a "***comments***" section at the end of the description.

MCCL command 'Quick' reference tables

The following is a list of the commands that the DCX will accept, they are grouped according to their

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
FC	40h	Full Current
FF	33h	amplifier Fault input ofF
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
JA	38h	Jog Acceleration
JB	DFh	Jog deadBand
JG	DCh	Jog proportional Gain
JO	DEh	Jog Offset
JV	37h	Jog Velocity
LF	36h	motion Limits ofF
LL	D2h	set motion Low Limit
LM	34h	Limit Mode
LN	35h	motion Limits oN
LS	36h	set Low Speed
MS	E7h	set Medium Speed
MV	C4h	set Minimum Velocity
OM	D8h	set Output Mode
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
UK	D7h	set User output constant
UA	9Ch	Use as default Axis
UO	B3h	set User Offset
UR	B1h	set User Rate conversion
UP	9Dh	Use Physical axis
US	AFh	set User Scale
UT	B2h	set User Time conversion
UZ	B0h	set User Zero
VA	ADh	set Vector Acceleration
VD	AEh	set Vector Deceleration
VG	77h	set Velocity Gain
VO	E0h	set Velocity Override
VV	ACh	set Vector Velocity

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
TQ	9H	enable TorQue mode
VM	18h	enable Velocity Mode

Motion Commands

MCCL	Code	Description
AB	Ah	ABort
AF	EBh	Auxiliary encoder arm/Find index
BF	CFh	Backlash compensation oFf
BN	CEh	Backlash compensation oN
CA	B4h	arc Center Absolute
CD	C1h	Contour Distance
CP	C0h	Contour Path
CR	B5h	arc Center Relative
EA		arc Ending Angle
FE	Bh	Find Edge
FI	Ch	Find Index
GH	Dh	Go Home
GO	Eh	GO
HO	Fh	HOme
IA	5Dh	Index Arm
JF	3Ah	Jogging ofF
JN	39h	Jogging oN
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
NS	ABh	No Synchronization
PP	EDh	Profile Parabolic
PR		Recode 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

Reporting Commands

MCCL	Code	Description
AT	E9h	Auxiliary encoder Tell position
AZ	ECh	Auxiliary encoder tell index
DO		Display recorded optimal position
DQ		Display recorded DAC output
DR		Display recorded actual position
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 position Gain
TI	4Fh	Tell Integral gain
TK	5Ch	Tell velocity constant
TL	50h	Tell integration Limit
TM	51h	Tell stored Macros
TO	59h	Tell Optimal
TP	52h	Tell Position
TR	57h	Tell Register n
TQ	D1h	Tell torQue
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

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 Stop

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
GA	F8h	Get Analog value
GD		Get module ID
GU	89h	Get the default axis
GX	F7h	Get auXiliary encoder position
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 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

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

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
TA	49h	Tell the value of Analog input
TC	4Ah	Tell state of digital Channel

Miscellaneous Commands

MCCL	Code	Description
BR	1Eh	Baud Rate
DM	3Ch	Decimal Mode
EF	25h	Echo ofF
EN	26h	Echo oN
FM	10Dh	Free Memory
HE	48h	HElp
HF	30h	Handshake ofF
HM	3Dh	Hexadecimal Mode
HN	31h	Handshake oN
ME	10Ch	MEmory allocate
NO	78h	No Operation
PC	80h	set Prompt Character
RT	2Ah	ReseT system
XF	7Eh	Xon/Xoff Protocol oFF
XN	7Dh	Xon/Xoff Protocol oN

File Commands

MCCL	Code	Description
DL	10Fh	Directory Listing
FO	10Eh	FOrmat file system
LO	112h	LOad file
RF	111h	Remove File
TY	110h	TYpe file

Plotting Commands

MCCL	Code	Description
PA	121h	Plotter Acceleration
PD	125h	Pen Down macro
PE	119h	Plotting Enable
PF	118h	Plot File
PI	123h	Plotter Initialize macro
PQ	122h	Plotter Quick velocity
PU	124h	Pen Up macro
PV	120h	Plotter Velocity
PX	11Ah	Plotter X axis
PY	11Bh	Plotter Y axis
SP	126h	Select Pen macro
XO	11Eh	plotter X Offset
XS	11Ch	plotter X Scale
YO	11Fh	plotter Y Offset
YS	11Dh	plotter Y Scale

Setup Commands

set Acceleration

parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: set Deceleration, set Velocity

explanation: 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.

disable Amplifier fault input monitoring

MCCL Command: aFFn
parameter: none
compatibility: MC200
see also: FN

explanation: Disables the Amplifier Fault input of a servo module. See description of amplifier Fault input oN command (FN), for further details.

enable Amplifier fault input monitoring

MCCL Command: aFNn
parameter: none
compatibility: MC200
see also: FF

explanation: Enables the Amplifier Fault input of a servo 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. This command has no effect on stepper motors.

define the position of the Auxiliary encoder

MCCL Command: aAHn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: DH

explanation: This command causes axis a auxiliary encoder position to be set to n. This encoder input is available on both the MC200 and MC260 modules, and is used for loop closure when a MC260 is controlling a closed loop stepper. The auxiliary encoder of a MC200 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.

set Backlash compensation distance

MCCL Command: aBDn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210
see also: BN, BF

explanation: Use this command to set the distance require to nullify the effects of mechanical backlash in the system. The command parameter should be equal to one half of 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.

define an output Deadband

MCCL Command: aODn
parameter: integer or real (0 <= n <= 10)
compatibility: MC200, MC210
see also: OO

explanation: This command can be used to simulate a ‘frictionless servo system’. The value n defines a voltage deadband range in the output of a MC200 and MC210 servo module. Parameter n modifies the commanded analog (MC200) or motor drive (MC210) output to a servo. The value n is added to a positive output and subtracted from a negative output.

set the position Deadband

MCCL Command: aDBn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210
see also: DT, TS

explanation: This command sets the position deadband 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 deadband of the current target position for a period of time specified with the Delay at Target (aDTn) command.

Define the position of an axis

MCCL Command: **aDHn**
parameter: integer or real (0 <= n)

compatibility: MC200, MC210, MC260
see also: FE, FI, IA, WE, WI

explanation: 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.

set Deceleration

MCCL Command: aDSn
integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: SA, SV

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

set the Direction of velocity mode motion

MCCL Command: aDIn
parameter: integer (n = 0 or 1))
compatibility: MC200, MC210, MC260
see also: GO, VM

explanation: 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 results in motion in the negative direction.

set the Feed forward acceleration gain

MCCL command: aASn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210
see also: DG, VG

explanation: 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
```

set the Feed forward deceleration gain

MCCL Command: aDGn
parameter: integer or real (0 <= n)

compatibility: MC200, MC210
see also: AG, VG

explanation: 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
```

set the Feed forward velocity gain

MCCL command: aVGn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: AG, DG

explanation: 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.

stop when defined Following error has been exceeded

MCCL Command: aSEn
parameter: integer (0 = n to disable, or 0 < n)
compatibility: MC200, MC210
see also: TS, aRD, IC, IS

explanation: 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 a parameter of 0 will disable errors on excessive following errors.

enable High speed mode

MCCL Command: aHS
parameter: None
compatibility: MC200, MC210, MC260
see also: LS, MS

explanation: 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 4 KHz update rate. For a stepper motor axis, it sets the maximum pulse rate to 1.25 Million Pulses / Sec.

set the Jog acceleration

MCCL Command: aJAn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: JB,JF,JG,JN,JO,JV

explanation: Sets jogging acceleration. See the section on Jogging for details.

set the Jog mode deadband

MCCL Command: aJBn
parameter: integer or real (0 <= n <= 5)
compatibility: MC200, MC210, MC260
see also: JA,JF,JG,JN,JO,JV

explanation: Sets jogging joystick input deadband. See the description of **Jogging** in the **Motion Control** chapter.

set the Jog mode proportional gain

MCCL Command: aJGn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: JA, JB, JF, JN, JO, JV

explanation: Sets jogging joystick proportional gain. See the description on **Jogging** in the **Motion Control** chapter.

set the Jog mode offset

MCCL Command: aJOn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: JA, JB, JF, JG, JN, JV

explanation: Sets jogging joystick input offset. See the description on **Jogging** in the **Motion Control** chapter.

set the Jog mode minimum velocity

MCCL Command: aJVn
parameter: integer or real (0 <= n)
compatibility: MC260

see also: JA, JB, JF, JG, JN, JO

explanation: Sets jogging minimum velocity for stepper motors. See the description on **Jogging** in the **Motion Control** chapter.

disable motion Limits error checking

MCCL Command: aLFn
parameter: (n = 0, 1, 2 or 3)
compatibility: MC200, MC210, MC260
see also: HL, LL, LM, LN

explanation: 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.

set the high Motion soft Limit

MCCL Command: aHLn
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: LF, LL, LM, LN

explanation: 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 it's 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.

set the low motion soft Limit

MCCL Command: aLLn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: HL, LF, LM, LN

explanation: 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 it's 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 on **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.

define the motion Limit type

MCCL Command: aLMn
parameter: integer (0 <= n)
compatibility: MC200, MC210, MC260
see also: HL, LF, LL, LN

explanation: This command is used to select how the DCX will react when a 'hard' limit switch or a 'soft' position limit is tripped on an axis. The command parameter should be formed by adding a value of 0, 1, or 2 for the hard limit switch mode, to a value of 0, 4 or 8 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. This will prevent the DCX from moving the motor until a Motor oN command is issued. See the description on **Motion Limits** in the **Motion Control** chapter.

enable motion Limits error checking

MCCL Command: aLNn
parameter: integer (n = 0, 1 or 2)
compatibility: MC200, MC210, MC260
see also: HL, LF, LL, LM

explanation: 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. If a soft motion limit is enabled, and the respective axis goes 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. 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 on **Motion Limits** in the **Motion Control** chapter.

enable Low speed mode

MCCL Command: aLS
parameter: None
compatibility: MC200, MC210, MC260
see also: HS, MS

explanation: 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 1 KHz update rate. For a stepper motor axis, it sets the maximum pulse rate to 19.5 Thousand Pulses / Sec.

enable Medium speed mode

MCCL Command: aMS
parameter: None
compatibility: MC200, MC210, MC260
see also: HS, LS

explanation: 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 156 Thousand Pulses / Sec.

set module Output mode

MCCL Command: aOMn
parameter: integer (n = 0, 1, 2, or 3)
compatibility: MC200, MC210, MC260
see also:
explanation: This command is used to set a servo or stepper module's output mode. The available modes are listed in the following tables.

n	*MC200 Output Mode*
0	Bipolar Analog output, -10V to +10V
1	Unipolar Analog output, 0V to +10V, direction J3 pin 7
2	Bipolar PWM signal output on J3 pin 7, 0 - 50% duty cycle
3	Unipolar PWM signal output on J3 pin 7, 0 – 100% duty cycle, Direction on Analog Output (J3 pin 2)

n	*MC260 Output Mode*
0	Pulse and Direction outputs (default)
1	CW and CCW Pulse Outputs

define an Output Offset

MCCL Command: aOOn
parameter: integer or real (-10 <= n <= 10)
compatibility: MC200, MC210
see also: OD

explanation: 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.

set the PID loop derivative gain

MCCL Command: aSDn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210
see also: IL, SG, SI,

explanation: 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.

set the PID loop derivative sampling period

MCCL Command: aFRn
parameter: integer (0 <= n <= 255)
compatibility: MC200, MC210
see also: SD

explanation: 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 zero. For a value of n, the sampling period will be (n +1) * sample period. See the High Speed command for a discussion of the sample period on servos.

set the PID loop integral gain

MCCL Command: aSIn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210
see also: IL, HS, LS, MS, SG, SD,

explanation: 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.

set the PID loop integration limit

MCCL Command: aILn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210
see also: FR, SI

explanation: 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 intervals.

set PID loop proportional gain of servo

MCCL Command: aSGn
parameter: integer or real (0.000153 <= n <= 10)
compatibility: MC200, MC210
see also: HS, LS, MS, SD, SI

explanation: 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.

set the Servo phasing

MCCL Command: aPHn
parameter: integer (n = 0 or 1)
compatibility: MC200, MC210
see also: OM

explanation: This command is used to set a servo module's output phasing. The phase of the output will determine whether the module drives the servo 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.

set Stepper to full current

MCCL Command: aFCn
parameter: none
compatibility: MC260
see also: HC

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

set Stepper to half current

MCCL Command: aHCn
parameter: none
compatibility: MC260
see also: FC

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

Stepper full step mode

MCCL Command: aSFn
parameter: none
compatibility: MC260

see also: SH

explanation: Causes Full/Half Step output signal of a stepper module to go low.

Stepper half step mode

MCCL Command: aSHn
parameter: none
compatibility: MC260
see also: SF

explanation: Causes Full/Half Step output signal of stepper module to go high.

set Stepper minimum velocity

MCCL Command: aMVn
parameter: integer or real (0 <= n)
compatibility: MC260
see also: DS,SA,SV

explanation: 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.

define the required delay at Target

MCCL Command: aDSn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210
see also: DB

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

set the Torque (output) of an axis

MCCL Command: aSQn
parameter: integer or real (-10.0 <= n <= 10.0)
compatibility: MC200, MC210
see also: QM

explanation: Sets maximum output level for servos. When an axis is placed in torque mode, this command sets the 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.

set the ratio of a Slave axis

MCCL Command: aSSn
parameter: integer or real (___ <= n <= ___)
compatibility: MC200, MC210
see also: SM

explanation: 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 **Master / Slave motion** in the **Motion Control** chapter.

set the maximum Velocity of an axis

MCCL command: aSVn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: DS, SA

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

set the Default axis

MCCL Command: aUAn
parameter: integer (0 <= n <= 6)
compatibility: MC200, MC210, MC260
see also: SM

explanation: This command is used to define a default axis. After issuing this command, any command move, setup, etc.. command that utilizes an axis designator (*a*) will execute the command to the axis specified by parameter n.

```
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
```

Use physical axis addressing

MCCL Command: aUPn
parameter: integer (1 <= n <= 6)

compatibility: MC200, MC210, MC260
see also:

explanation: This command is used to reassign the axis designator of a motor module. The value 'a' should equal the new axis designator. The parameter n should equal the current physical location of the motor module. See the description of **Physical Assignment of Axes Numbers** in the **Motion Control** chapter.

define the User offset

MCCL Command: aUOn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: UK, UR, US, UT, UZ

explanation: This command is used to configure an axis for commands in user units. The setting of the user output constant defaults to 1.0. See the description of **Setting User Units** in the **Application Solutions** chapter.

define the User output constant

MCCL Command: aUKn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210
see also: UO, UR, US, UT, UZ

explanation: This command is used to configure an axis for commands in user units. The setting of the user output constant defaults to 1.0. See the description of **Setting User Units** in the **Application Solutions** chapter.

set the User rate conversion factor

MCCL Command: aURn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: UK, UO, US, UT, UZ

explanation: This command is used to configure an axis for commands in user units. The setting of the user output constant defaults to 1.0. See the description of **Setting User Units** in the **Application Solutions** chapter.

set the User length conversion factor

MCCL Command: aUSn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: UK, UO, UR, UT, UZ

explanation: This command is used to configure an axis for commands in user units. The setting of the user output constant defaults to 1.0. See the description of **Setting User Units** in the **Application Solutions** chapter.

set the User time conversion factor

MCCL Command: aUTn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: UK, UO, UR, US, UZ

explanation: This command is used to configure an axis for commands in user units. The setting of the user output constant defaults to 1.0. See the description of **Setting User Units** in the **Application Solutions** chapter.

set the User zero position

MCCL Command: aUZn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: UK, UO, UR, US, UT

explanation: This command is used to configure an axis for commands in user units. The setting of the user output constant defaults to 1.0. See the description of **Setting User Units** in the **Application Solutions** chapter.

set contour move Vector acceleration

MCCL command: aVAn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: VD, VV

explanation: 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.

set contour move Vector deceleration

MCCL command: aVDn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: VA, VV

explanation: 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.

set contour move Vector Velocity

MCCL command: aVVn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: VA, VD

explanation: 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.

set contour move Velocity override

MCCL command: aVOn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: VA, VD, VV

explanation: Sets a multiplying factor that will be applied to the velocity of both servo and stepper motors. This command does not support jogging. For contour moves (linear, circular) the axis identified ‘a’ should the axis number of the ‘controlling’ axis.

Mode Commands

enable Closed loop stepper mode

MCCL command: aIMn
parameter: 0 or 1
compatibility: MC260
see also:

explanation: 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 **HDCX Stepper Basics** in the **Motion Control** chapter.

enable Contour mode

MCCL command: aCMn
parameter: integer (1<= n <= 6)
compatibility: MC200, MC210, MC260
see also: CP

explanation: This command places a servo or stepper motor in the Contour Mode of operation. In this mode, a motor can be included in a Contour Path motion command. The parameter 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. For example if axes 1,2 and 3 are to perform contour path commands as a group, the following command sequence should be used to place them in contour mode:

```
example:     1CM1,2CM1,3CM1           ;Place axes 1, 2 and 3 in contour mode
```

Contour Mode is terminated by issuing a Position (PM) or Velocity Mode (VM) command to all axes in the group. The controlling axis should be taken out of contour mode last.

enable Gain mode

MCCL command: aGM
parameter: none
compatibility: MC200, MC210
see also: QM

explanation: 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 (no maximum velocity, no acceleration, no deceleration) will be generated. The servo will be driven to the new target based only upon the output of the PID loop.

enable Master/Slave mode

MCCL command: aSM
parameter: integer (0 <= n <= 101)

compatibility: MC200, MC210, MC260
see also: SS

explanation: 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 description of **Master/Slave Motion** in the **Motion Control** chapter.

enable Position mode

MCCL command: aPM
parameter: none
compatibility: MC200, MC210, MC260
see also: VM

explanation: 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.

```
example:    1MN,1PM,1SV100,1SA10000,1DS10000,1MR1000
```

DCX Reset

MCCL command: RT
parameter: none
compatibility: MC200, MC210, MC260
see also:

explanation: 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.

enable Torque mode

MCCL command: aQM
parameter: none
compatibility: MC200, MC210

see also: SQ

explanation: This command places a servo (not valid for 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 Set torQue command. The parameter to this command has default units of volts. In this mode of operation, the servo motor control module (MC200, MC210) is 'turned into' 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.

enable Velocity mode

MCCL command: aVM
parameter: none
compatibility: MC200, MC210, MC260
see also: DI, GM, GO, PM, QM, ST

explanation: 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. When using Velocity Mode the user can specify the direction for the motor to move using the DIrection (DI) command. While a motor is moving in Velocity Mode, 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.

```
example:    1MN,SV10000,SA100,DI1,VM,1GO      ;start the motor moving
            WA5,1SV11000,WA5,SV12000          ;change the velocity
            1ST,WS0.1,PM,MN                   ;stop motor; wait for stop,
                                              ;back to position mode, motor
                                              ;on to reset target position
```

Motion Commands

Abort move

MCCL command: aAB
parameter: none
compatibility: MC200, MC210, MC260
see also: MF, ST

explanation: 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
```

Auxiliary encoder find index mark

MCCL command: aAFn
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: AH, DH, FI

explanation: This command is used to initialize a motor's auxiliary encoder at a given position. It will remain in effect until the auxiliary encoder's index pulse goes active. At that time the current position of the auxiliary encoder will be set to n. See the description of **Homing Axes** in the **Motion Control** chapter.

turn off Backlash compensation

MCCL command: aBF
parameter: none
compatibility: MC200, MC210
see also: BD, BN

explanation: 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.

turn on Backlash compensation

MCCL command: aBN
parameter: none
compatibility: MC200, MC210
see also: BD, BF

explanation: Use this command to enable backlash compensation. It should be issued after the backlash compensation distance has 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 was present). See the description of **Backlash Compensation** in the **Application Solutions** chapter.

define Center (absolute) of an arc

MCCL command: aCAn
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: CM, CP, CR

explanation: 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.

define Center (relative) of an arc

MCCL command: aCRn
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: CM, CP, CA

explanation: 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.

define the Ending angle (absolute) of an arc

MCCL command: aEAn
parameter: integer or real (0 >= n >= 360)
compatibility: MC200, MC210, MC260
see also: CA, CP, RR

explanation: 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.

define the radius of an Arc

MCCL command: aRRn
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: CM, CP

explanation: 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.

define the Contour distance of a user defined move

MCCL command: aCDn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: CP

explanation: 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.

define the Contour move type

MCCL command: aCPn
parameter: integer (n = 0, 1, 2, 3)
compatibility: MC200, MC210, MC260
see also: CM

explanation: 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.

```
example:    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

Find Edge

MCCL command: *a*FE*n* *a* = Axis number *n* = integer or real >= 0
compatibility: MC260
see also: FI

This command is used to initialize a stepper motor at a given position. The command will remain in effect until the home input of the module goes active. At that time an internal position register of the MC260 module will be set to the current position. The position of the axis (where the index mark was captured) will not be defined to position *n* until after the Motor Off / Motor oN (aMNn) command sequence has been issued. This command will not cause any motor motion to be started or stopped. It is up to the user to initiate motor motion before issuing the command, and to stop any motion after it completes. See the description of **Homing Axes** in the **Motion Control** chapter.

Note: The status bit associated with the Find Edge command is bit 24 (stepper Home). The Find Edge command causes this bit to be latched after the index mark has been captured. To clear the latched status bit, issue the Motor Off and Motor oN sequence.

```
MD1,1LM2,1LN3,MJ10                          ;call homing macro
MD10,1VM,1DI0,1SV10000,1GO,1RL0,IS24,MJ11,NO,IS17,MJ13,NO,JR-7
                                            ;test for sensors (home and +limit)
MD11,1ST,1WS.1,1DI0,1SV5000,1GO,1RL0,IC24,MJ12,NO,JR-4
                                            ;Move positive until home sensor off
MD12,1ST,1WS.1,1DI1,1SV5000,1GO,MJ15
                                            ;move back to the home sensor
MD13,1MN,1DI1,1SV5000,1GO,MJ15              ;move out of limit sensor range back
                                            ;toward the home sensor
MD14,1FE0,1ST,1WS.1,1MF,WA.1,1MN,1PM,1MA0
                                            ;find the active edge of the home
                                            ;sensor. Stop axis, initialize
                                            ;position, move to position 0.
```

Find the edge of the home sensor (stepper)

MCCL command: aFEn
parameter: integer or real
compatibility: MC260
see also: DH, FI

explanation: This command is used to initialize a stepper motor at a given position. It will remain in effect until the home input of the module goes active. At that time the current position of the motor will be defined to be n. This command will not cause any motor motion to be started or stopped. It is up to the user to initiate motor motion before issuing the command, and to stop any motion after it completes. See the description of **Homing Axes** in the **Motion Control** chapter.

```
MD1,1LM2,1LN3,MJ10                          ;call homing macro
MD10,1VM,1DI0,1GO,1RL0,IS24,MJ11,NO,IS17,MJ12,NO,JR-7
                                            ;test for sensors (home and +limit)
MD11,1ST,1DI1,1SV5000,1GO,1WE1,1ST,1DI0,1SV1000,1GO,1FE0,1ST,1PM,1MN,1MA0
                                            ;move 'back and forth' across the edge
                                            ;of the home sensor, then initialize
                                            ;on the leading edge
MD12,1MN,1DI1,1GO,1WE0,MJ11                 ;move negative until home true
```

Find the index mark of the encoder (servo)

MCCL command: aFIn
parameter: integer or real
compatibility: MC200, MC210
see also: DH, FE

explanation: This command is used to initialize a servo's encoder at a given position. It will remain in effect until the encoder index pulse goes active. At that time the current position of the servo will be defined to be n. This command will not start or stop any servo motions, 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 in rotary encoders), a typical servo application will require a coarse home signal to "qualify" the index pulse.

```
MD1,1LM2,1LN3,MJ10                          ;call homing macro
MD10,1VM,1DI0,1GO,1RL0,IS25,MJ11,NO,IS17,MJ12,NO,JR-7
                                            ;test for sensors (home and +limit)
MD11,1ST,1WS.01,1DI1,1GO,1WE1,1ST,1DI0,1GO,1WE0,1FI0,1 ST,1WS.01,1PM,1MN,1MA0
                                            ;if home sensor true, initialize on
                                            ;index
MD12,1MN,1DI1,1GO,1WE0,MJ11                 ;move negative until home true
```

See the description of **Homing Axes** in the **Motion Control** chapter.

Go home (move to position zero)

MCCL command: aGHn
parameter: none
compatibility: MC200, MC210, MC260
see also: MA

explanation: Causes the specified axis or axes to move to the offset position that was specified when the last DH, 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 (start velocity mode move / contour mode move)

MCCL command: aGOn
parameter: integer (n = 0 or 1)
compatibility: MC200, MC210, MC260
see also: DI, SN, VM

explanation: 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 linear interpolated (n = 0), or a cubic spline (n = 1).

Home the axis (call homing macro)

MCCL command: aHO
parameter: none
compatibility: MC200, MC210, MC260
see also: MC, MD

explanation: 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.

encoder Index mark arm (capture axis position)

MCCL command: aIAn
parameter: integer or real
compatibility: MC200, MC210
see also: FI, WI

explanation: This command is used to arm the index capture function of a servo axis. It has a similar function to the Find Index command, but does not wait for the index pulse to occur. After the Index Arm command is issued, and the index pulse occurs, the position where the index pulse occurred will be set to the command parameter n. The Wait for Index command and Motor oN commands should be issued to the axis following the Index Arm for the new position to be set.

disable Jogging

MCCL command: aJF
parameter: none
compatibility: MC200, MC210, MC260
see also: JN

explanation: Disables jogging of a servo or stepper motor. See the description of **Jogging** in the **Motion Control** chapter.

enable Jogging

MCCL command: aJN
parameter: none
compatibility: MC200, MC210, MC260
see also: JF

explanation: Enables jogging of a servo or stepper motor. See the description of **Jogging** in the **Motion Control** chapter.

Learn the position of one or all axes

MCCL command: aLP
parameter: integer (0 <= n < (1536 / # of axes on DCX board)
compatibility: MC200, MC210, MC260
see also: LT

explanation: 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.

Learn the target of one or all axes

MCCL command: aLT
parameter: integer (0 <= n < (1536 / # of axes on DCX board)
compatibility: MC200, MC210, MC260
see also: LP

explanation: Similar to the LP command, but stores the axes' current target (versus actual position). This command does not require actually moving to the position to store the point. 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.

turn the Motor off

MCCL command: aMF
parameter: none
compatibility: MC200, MC210, MC260
see also: MN

explanation: 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.

turn the Motor on

MCCL command: aMN
parameter: none
compatibility: MC200, MC210, MC260
see also: MF, US

explanation: 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.

Move to absolute position

MCCL command: aMA
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: CM, PM

explanation: 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.

Move to stored point

MCCL command: aMP
parameter: integer (0 <= n < (1536 / # of axes on DCX board))
compatibility: MC200, MC210, MC260
see also: LP, LT

explanation: 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 moved. 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.

Move to relative position

MCCL command: aMR
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: CM, PM

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

Record axis data

MCCL command: aPRn
parameter: integer (0 <= n < 512)
compatibility: MC200, MC210
see also: DO, DQ, DR, LS, HS, MS

explanation: 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.

use Parabolic velocity profile

MCCL command: aPP
parameter: none
compatibility: MC260
see also: PS, PT

explanation: 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 **Defining the Characteristics of a Move** in the **Motion Control** chapter.

use S-curve velocity profile

MCCL command: aPS
parameter: none
compatibility: MC200, MC210
see also: PP, PT

explanation: 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 **Defining the Characteristics of a Move** in the **Motion Control** chapter.

use Trapezoidal velocity profile

MCCL command: aPT
parameter: none
compatibility: MC200, MC210, MC260
see also: PS, PT

explanation: 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 **Defining the Characteristics of a Move** in the **Motion Control** chapter.

Restore the motor configuration

MCCL command: aRC
parameter: integer (1 <= n <= 127)
compatibility: MC200, MC210, MC260
see also: SC, VO

explanation: This command takes an axis specifier a and requires a file number as the command parameter n . This command restores the entire motor table. This includes the public motor table in dual port memory and the private motor table in internal RAM. When used with the Save Configuration command, motors can be stopped (aVO0) during a 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). See the description of **Pause and Resume Motion** in the **Motion Control** chapter.

Save the motor configuration

MCCL command: aSC
parameter: integer (1 <= n <= 127)
compatibility: MC200, MC210, MC260
see also: RC, VO

explanation: This command takes an axis specifier a and requires a file number as the command parameter n . This command saves the entire motor table. This includes the public motor table in dual port memory and the private motor table in internal RAM. When used with the Restore Configuration command, motors can be stopped (aVO0) during a 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). See the description of **Pause and Resume Motion** in the **Motion Control** chapter.

Stop a motor

MCCL command: aST
parameter: none
compatibility: MC200, MC210, MC260

see also: AB, MF

explanation: 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.

disable synchronization

MCCL command: aNS
parameter: none
compatibility: MC200, MC210, MC260
see also: SN

explanation: 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.

enable synchronization

MCCL command: aSN
parameter: none
compatibility: MC200, MC210, MC260
see also: GO, CP, CM, NS

explanation: 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.

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, this 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 the 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.

report the D/A conversion of an Analog input channel

MCCL command: aTAp
parameter: (x = 1, 2, 3 or 4 for motherboard input channels, x >= 5 for module channels)
compatibility: MC500, MC510
see also: GA

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

report the position of the Auxiliary encoder

MCCL command: aATp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210, MC260
see also: AZ, TP

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

report the position of the Auxiliary encoder index

MCCL command: aAZp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210, MC260
see also: AT

explanation: 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.

report the Breakpoint position of an axis

MCCL command: aTBp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210, MC260
see also: IP, IR, WP, WR

explanation: 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.

display the Captured actual position of an axis

MCCL command: aDRn
parameter: integer (0 <= n < 512)
compatibility: MC200, MC210
see also: DO, DQ, LS, HS, MS, PR

explanation: 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.

display the Captured DAC output of an axis

MCCL command: aDQn
parameter: integer (0 <= n < 512)
compatibility: MC200, MC210
see also: DO, DR, LS, HS, MS, PR

explanation: 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.

display the Captured optimal position of an axis

MCCL command: aDOn
parameter: integer (0 <= n < 512)
compatibility: MC200, MC210
see also: DQ, DR, LS, HS, MS, PR

explanation: 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.

report the Contouring count

MCCL command: aTXp
parameter: Integer)0 >= n >= 2,147,483,647)
compatibility: MC200, MC210, MC260
see also: CM, CP

explanation: 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.

report the Derivative gain setting

MCCL command: aTDp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210
see also: SG, TG, TI

explanation: Reports the derivative gain setting for a servo.

report the state of a Digital channel

MCCL command: aTCx
parameter: integer (0 < x <= 16 for motherboard channels)
compatibility: MC400
see also: CH, CI, CL, CT

explanation: 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".

report the position of the Encoder index

MCCL command: aTZp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210
see also: AZ, TP

explanation: 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.

report the current Following error of a servo axis

MCCL command: aTFp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210
see also: SE, TO, TP

explanation: 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.

report the state of a Digital channel

MCCL command: aTCx
parameter: integer (0 < x <= 16 for motherboard channels)
compatibility: MC400
see also: CH, CI, CL, CT

explanation: 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".

report the Integral gain setting

MCCL command: aTIp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210
see also: SD, SG, TG,

explanation: Reports the integral gain setting for a servo.

report the Integral limit setting

MCCL command: aTLp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210
see also: SD, SG, TG,

explanation: Reports the integral limit setting for a servo.

Report stored Macros

MCCL command: TMn
parameter: integer (-1 <= n <= 1099)
compatibility: N/A
see also: MD, RM

explanation: 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.

report the Optimal position of an axis

MCCL command: aTOp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210, MC260
see also: TP, TT

explanation: 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.

report the current Position of an axis

MCCL command: aTPp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210, MC260
see also: TO, TT

explanation: 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.

report the Proportional gain setting

MCCL command: aTGp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210
see also: SD, TG, TI

explanation: Reports the proportional gain setting for a servo.

report the maximum torque setting of a servo

MCCL command: aTQp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210
see also: SQ

explanation: Reports the maximum torque output for a servo. This is the value that was set with the Set torQue command. See the description of the **Torque Mode Output Control** in the **Application Solutions** chapter.

report the Status of an axis

MCCL command: aTSp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210
see also:

explanation: 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. Those status flags that are not valid for steppers have an "*" by them. 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	Motor Jogging is Enabled
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	Unused
11	Unused
12	Breakpoint Reached (IP, IR, WP, WR)
13	Exceeded Max. Following Error *
14	Amplifier Fault Enabled *
15	Amplifier 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 / Stepper Home
25	Coarse home (current state)
26	Amplifier Fault *
27	Auxiliary Encoder Index
28	Limit Positive Input Active (current state)
29	Limit Negative Input Active
30	User Input 1 *
31	User Input 2 *

* not valid for stepper modules

```
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 Hexidecimal 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

```
example:       1TS33
```

DCX returns:
MOTOR AUXILIARY STATUS:
Hard Motion Limit Mode = Turn Motor Off
Soft Motion Limit Mode = Turn Motor Off
Servo Loop Rate is Medium
Synchronization is Off
Servo Phasing is Standard
Backlash Compensation is Off

```
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
Deadband: 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: 0.019530 Input2: 0.000000

report the Target position of an axis

MCCL command: aTTp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210, MC260
see also: TO, TP

explanation: Reports target position. This is the absolute position to which the servo or stepper motor was last commanded to move to. It may be specified directly with the Move Absolute (MA) command

or indirectly with Move Relative (MR) command. The interpretation of the command parameter p is explained at the beginning of this section.

report the current Velocity of an axis

MCCL command: aTVp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210, MC260
see also: HS, LS, MS, US

explanation: Reports the current velocity of a servo or stepper motor. The value is reported in units of encoder counts per servo loop update.

report the Velocity constant setting

MCCL command: aTKp
parameter: ascii interface (0 <= p < 1), binary interface (p = 0, 1 or 2)
compatibility: MC200, MC210
see also: VG

explanation: 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.

Report the value in the user register '*n*'

MCCL command: TRn
parameter: integer (0 <= n <= 256)
compatibility: N/A
see also:

explanation: Displays the contents of a 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. This command will accept a decimal point and a tenths digit appended to the register number in the command parameter. This digit specifies the number of digits to the right of the decimal point that real number values will be rounded to for display.

report the firmware Version of the DCX controller

MCCL command: VE
parameter: none
compatibility: N/A
see also:

explanation: Reports the revision level of the firmware in the ROM of the DCX. This command also displays the amount of memory installed on the DCX motion controller motherboard.

example: VE

DCX returns:
DCX-VM200 Motion Controller
Hardware: 64K Dual Port RAM, 256K Flash Memory
System Firmware Ver. PM1 Rev. 3.6a

I/O Commands

report the D/A conversion of an Analog input channel

MCCL command: aTAp
parameter: (x = 1, 2, 3 or 4 for motherboard input channels, x >= 5 for module channels)
compatibility: MC500, MC510
see also: GA

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

configure Digital channel as an input

MCCL command: CIx
parameter: integer (1 <= x)
compatibility: MC400
see also: CF, CH, CL, CN, CT, TC

explanation: 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 view with the Tell Channel command.

configure Digital channel as high true

MCCL command: CHx
parameter: integer (1 <= x)
compatibility: MC400
see also: CF, CI, CL, CN, CT, TC

explanation: 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 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 "low true".

configure Digital channel as low true

MCCL command: CLx
parameter: integer (1 <= x)
compatibility: MC400

see also: CF, CH, CI, CN, CT, TC

explanation: 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".

configure Digital channel as an output

MCCL command: CTx
parameter: integer (1 <= x)
compatibility: MC400
see also: CF, CH, CI, CL, CN, TC

explanation: Used to configure digital I/O channel x as an output. The DCX will turn the channel "off" before changing it to an output.

turn off the Digital output

MCCL command: CFx
parameter: integer (1 <= x)
compatibility: MC400
see also: CH, CI, CL, CN, CT, TC

explanation: Causes digital I/O 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 than 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).

turn on the Digital output

MCCL command: CNx
parameter: integer (1 <= x)
compatibility: MC400
see also: CF, CH, CI, CL, CT, TC

explanation: 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).

report the state of a digital Channel

MCCL command: aTCx
parameter: integer (0 < x <= 16 for motherboard channels)
compatibility: MC400
see also: CH, CI, CL, CT

explanation: 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".

reset the counter (Digital input)

MCCL command: RC
parameter: none
compatibility: N/A
see also:

explanation: Initializes DCX hardware pulse counter to 0.
comment: ***Not implemented at this time.***

Register Commands

Add '*n*' to the value in the accumulator

MCCL command: AAn
parameter: integer or real

explanation: 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.

Copy the value in the accumulator to register '*n*'

MCCL command: ARn
parameter: integer or real

explanation: Copies the contents of the Accumulator (User Register 0) to the User Register specified by the command parameter. The contents of the Accumulator are unaffected by this command.

Copy value in register 'n' to the accumulator

MCCL command: RAn
parameter: integer (0 <= n <= 255)

explanation: Copies the contents of the User Register specified by the command parameter into the Accumulator (User Register 0). The original contents of the accumulator is overwritten, while the contents of the source User Register are unaffected.

Complement the accumulator, bit wise

MCCL command: AC
parameter: none

explanation: 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.

Divide the value in the accumulator by '*n*'

MCCL command: ADn
parameter: integer or real

explanation: 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.

Evaluate the value in the accumulator

MCCL command: AVn
parameter: integer or real (1<= n <= 24)

explanation: Performs a unary operation on the contents of the Accumulator (User Register 0), placing the result in the Accumulator, overwriting the original contents. The command parameter 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 <= 24) will be a 64 bit real in all cases except the Convert to ASCII operation which returns an integer.

Parameter n =	*Operation*
1	Convert to ASCII (Address placed in ACC)
2	Change Sign
3	Absolute Value
4	Ceiling
5	Floor
6	Fraction
7	Round
8	Square
9	Square Root
10	Sine
11	Cosine
12	Tangent
13	Arc Sine
14	Arc Cosine
15	Arc Tangent
16	Hyperbolic Sine
17	Hyperbolic Cosine
18	Hyperbolic Tangent
19	Exponent
20	Log
21	Log10
22	Load Pi
23	Load 2 * Pi
24	Load Pi / 2

Get the Analog value of channel '*n*'

MCCL command: GAx
parameter: (x = 1, 2, 3 or 4 for motherboard input channels, x >= 5 for module channels)

explanation: 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.

Get the Module ID

MCCL command: GDx
parameter: (1<= x <= 6)

explanation: Loads the accumulator with the type of motor module associated with an axis number

Module Type	ID code
MC200	0
MC210	16
MC260	1

Get the default axis number

MCCL command: GUx
parameter: (1<= x <= 6)

explanation: This command is used to place the current default axis number in the accumulator.

Get the position of the auxiliary encoder

MCCL command: aGX
parameter: none
compatibility: MC200, MC210, MC260

explanation: This command reads the auxiliary encoder associated with axis a and places the value into the Accumulator (User Register 0).

Load the accumulator with '*n*'

MCCL command: ALn
parameter: integer or real

explanation: Loads the Accumulator (User Register 0) with the value of the command parameter *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.

```
example:    AL1234567890            ;Load 1234567890 into the accumulator

            AL1234.56789            ;Load 1234.56789 into the accumulator

            AL0.123456789e4         ;Load 1234.56789 into the accumulator
```

Logical 'and' the value in the accumulator with '*n*' , bit wise

MCCL command: ANn
parameter: integer or real

explanation: 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.

Logical exclusive or the value in the accumulator with '*n*' , bit wise

MCCL command: AEn
parameter: integer or real

explanation: 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.

Logical 'or' the value in the accumulator with '*n*' , bit wise

MCCL command: AOn
parameter: integer or real

explanation: 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.

Multiply the value in the accumulator by '*n*'

MCCL command: AMn
parameter: integer or real

explanation: 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.

Output analog value '*n*'

MCCL command: OAn
parameter: integer or real
compatibility MC500, MC520

explanation: 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.

Subtract the value in the accumulator by '*n*'

MCCL command: ASn
parameter: integer or real

explanation: 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.

Read the byte at absolute memory location '*n*' into the accumulator

MCCL command: nRBn
parameter: integer

explanation: This command will copy the contents of the byte located at the absolute memory address specified by the command parameter 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 chapter on Reading and Writing Memory 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.

Read the double (64 bit real) value at absolute memory location '*n*' into the accumulator

MCCL command: nRDn
parameter: real

explanation: This command will copy the contents of the Double (64 bit real) located at the absolute memory address specified by the command parameter 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 chapter on Reading and Writing Memory lists the offsets of all data in the motor tables.

Read the float (32 bit real) value at absolute memory location '*n*' into the accumulator

MCCL command: nRVn
parameter: real

explanation: This command will copy the contents of the Float (32 bit real) located at the absolute memory address specified by the command parameter 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 chapter on Reading and Writing Memory lists the offsets of all data in the motor tables.

Read the long (32 bit integer) value at absolute memory location '*n*' into the accumulator

MCCL command: nRLn
parameter: integer

explanation: This command will copy the contents of the Long (32 bit integer) located at the absolute memory address specified by the command parameter 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 chapter on Reading and Writing Memory lists the offsets of all data in the motor tables.

Read the word (16 bit integer) value at absolute memory location '*n*' into the accumulator

MCCL command: nRWn
parameter: integer

explanation: This command will copy the contents of the Word (16 bit integer) located at the absolute memory address specified by the command parameter 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 chapter on Reading and Writing Memory lists the offsets of all data in the motor tables.

Shift the value in the accumulator left by '*n*' bits

MCCL command: SLn
parameter: integer (0 <= n <=31)

explanation: 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.

Shift the value in the accumulator right by '*n*' bits

MCCL command: SRn
parameter: integer (0 <= n <=31)

explanation: 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.

Report the value in the user register '*n*'

MCCL command: TRn
parameter: integer (0 <= n <= 256)

explanation: Displays the contents of a 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. This command will accept a decimal point and a tenths digit appended to the register number in the command parameter. This digit specifies the number of digits to the right of the decimal point that real number values will be rounded to for display.

Write the low byte in the accumulator to absolute memory location '*n*'

MCCL command: WBn
parameter: integer

explanation: This command will copy the low byte of the accumulator (User Register 0) to the byte located at the absolute memory address specified by the command parameter.

Write the double (64 bit real) value in the accumulator to absolute memory location '*n*'

MCCL command: WDn
parameter: real

explanation: This command will copy a Double (64 bit real) from the accumulator (User Register 0 and 1) to the absolute memory address specified by the command parameter.

Write the long (32 bit integer) value in the accumulator to absolute memory location '*n*'

MCCL command: WLn
parameter: integer

explanation: This command will copy a Long (32 bit integer) from the accumulator (User Register 0) to the absolute memory address specified by the command parameter.

Write the float (32 bit real) value in the accumulator to absolute memory location '*n*'

MCCL command: WVn
parameter: real

explanation: This command will copy a float (32 bit real) from the accumulator (User Register 0) to the absolute memory address specified by the command parameter.

Write the low word (16 bit integer) value in the accumulator to absolute memory location '*n*'

MCCL command: WWn
parameter: integer

explanation: This command will copy the low Word (16 bit integer) of the accumulator (User Register 0) to the Word located at the absolute memory address specified by the command parameter.

Macro and Multi-Tasking Commands

escape the Background task

MCCL command: ETn
parameter: integer (0 <= n)
compatibility: N/A
see also: GT

explanation: 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.

generate an Background task

MCCL command: GTn
parameter: integer (0 <= n <= 1099)
compatibility: N/A
see also: ET, MD, MC

explanation: 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.

terminate command execution (Break)

MCCL command: BK
parameter: none
compatibility: N/A
see also:

explanation: 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.

Call and execute macro command '*n*'

MCCL command: MCn
parameter: integer (0 <= n <= 1099)
compatibility: N/A

see also: ET, MD

explanation: 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.

define a MCCL command sequence as a Macro

MCCL command: MDn
parameter: integer (0 <= n <= 1099)
compatibility: N/A
see also: ET, GT, MD, TM

explanation: 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.

On the DCX, macros can be stored in one of two types of memory. Macro numbers 0 through 9, and 256 through 1099 are stored in 'Flash' memory which is preserved even if power to the board is turned off. Macro numbers 10 through 255 are stored in 'RAM', these macros will erased if power to the board is turned off (unless a user supplied battery is connected to the board). A macro in the Flash memory has the disadvantage that it can't be redefined unless all the macros are erased. A macro in Ram can be redefined without erasing the existing macros, 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, and n is between 10 and 255 inclusive, the previously defined macro will be replaced by the new macro definition. If n is outside this range, the previously defined macro will be "undefined", but not replaced.

Note that macro 0 will be executed when the board is powered up or reset.

No operation

MCCL command: NO
parameter: none
compatibility: N/A
see also: BK

explanation: This command does nothing. It can be used to cause short delays in command line executions or as a filler in sequence commands.

Jump to macro '*n*'

MCCL command: MJn
parameter: integer (0 <= n <= 1099)
compatibility: N/A
see also: MC, MD

explanation: 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.

Reset stored macros

MCCL command: RMn
parameter: integer (n = 0, 1 or 2)
compatibility: N/A
see also: MD, TM

explanation: 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. The parameter to this command selects which group of macros will be erased. A parameter value of 1, causes the macros in Ram to be erased, a value of 2 causes the macros in Flash memory to be erased, and a parameter value of 0 causes all macros to be erased. 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 Command** in the **DCX Operation** chapter.

Report stored Macros

MCCL command: TMn
parameter: integer (-1 <= n <= 1099)
compatibility: N/A
see also: MD, RM

explanation: 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.

Sequence (If / Then) Commands

if the value in the accumulator is Below ‘n’ execute the next command, else skip 2 commands

MCCL command: IBn
parameter: integer or real
compatibility: MC400
see also: IE, IG, IU

explanation: Used for conditional execution of commands. If the contents of the accumulator (User Register 0) is less than the value of the command parameter, 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.

```
example:     IB0,MJ10,NO,MJ11          ;If the accumulator contents is less
                                       ;than 10 jump to macro 10, otherwise
                                       ;jump to macro 11
```

if value in the accumulator Equals “n” execute the next command, else skip 2 commands

MCCL command: IEn
parameter: integer or real
compatibility: N/A
see also: IB, IG, IU

explanation: Used for conditional execution of commands. If the contents of the accumulator (User Register 0) equals the value of the command parameter, 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.

```
example:     IE0,MJ10,NO,MJ11          ;If accumulator contents equals 0 jump
                                       ;to macro 10, otherwise jump to macro
                                       ;11
```

if value in the accumulator is Unequal to “n” execute the next command, else skip 2 commands

MCCL command: IUn
parameter: integer or real
compatibility: N/A
see also: IB, IE, IG

explanation: Used for conditional execution of commands. If the contents of the accumulator (User Register 0) does not equals the value of the command parameter, 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.

```
example:    IU0,MJ10,NO,MJ11          ;If accumulator contents is unequal to
                                      ;0 jump to macro 10, otherwise jump to
                                      ;macro 11
```

if the value in the accumulator is Greater than ‘n’ execute the next command, else skip 2 commands

MCCL command: IGn
parameter: integer or real
compatibility: N/A
see also: IB, EG, IU

explanation: Used for conditional execution of commands. If the contents of the accumulator (User Register 0) is greater than the value of the command parameter, 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.

```
example:    IG0,MJ10,NO,MJ11          ;If the accumulator contents is
                                      ;greater than 0 jump to macro 10,
                                      ;otherwise jump to macro 11
```

if bit ‘n’ of the accumulator is Clear execute the next command, else skip 2 commands

MCCL command: ICn
parameter: (0 <= n <= 31)
compatibility: N/A
see also: IS

explanation: Used for conditional execution of commands. If the contents of the accumulator (User Register 0) has bit n (specified by the command parameter) 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.

```
example:    IC3,MJ10,NO,MJ11          ;If accumulator bit 3 is cleared jump
                                      ;to macro 10, otherwise jump to macro
                                      ;11
```

if bit ‘n’ of the accumulator is Set execute the next command, else skip 2 commands

MCCL command: ISn
parameter: (0 <= n <= 31)
compatibility: N/A
see also: IC

explanation: Used for conditional execution of commands. If the contents of the accumulator (User Register 0) has bit n (specified by the command parameter) set, 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.

```
example:     IS3,MJ10,NO,MJ11          ;If accumulator bit 3 is set jump to
                                       ;macro 10, otherwise jump to macro 11
```

Execute remaining command sequence if Digital channel ‘x’ is off

MCCL command: DFx
parameter: integer (1 <= x)
compatibility: MC400
see also: DN, IF, IN

explanation: Used for conditional execution of commands. If the specified digital I/O channel 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.

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

Execute remaining command sequence if Digital channel ‘x’ is on

MCCL command: DNx
parameter: integer (1 <= x)
compatibility: MC400
see also: DF, IF, IN

explanation: Used for conditional execution of commands. If the specified digital I/O channel 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.

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

if Digital channel ‘*x*’ is off execute the next command, else skip 2 commands

MCCL command: IFx
parameter: (1 <= x)
compatibility: N/A
see also: DF, DN, IN

explanation: Used for conditional execution of commands. If the specified digital I/O channel 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.

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

if Digital channel '*x*' is on execute the next command, else skip 2 commands

MCCL command: INx
parameter: (1 <= x)
compatibility: N/A
see also: DF, DN, IF

explanation: Used for conditional execution of commands. If the specified digital I/O channel 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.

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

Interrupt (set breakpoint reached flag) upon reaching absolute position

MCCL command: IPn
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: IR, WP, WR

explanation: This command is used to indicate when an axis has reached a specific position. The position is specified by the command parameter 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.

Interrupt (set breakpoint reached flag) upon reaching relative position

MCCL command: IRn
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: IP, WP, WR

explanation: This command is used to indicate when an axis 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. The IR command can be issued to an axis before or after it has been commanded to move.

Jump to command (absolute) within MCCL sequence

MCCL command: JPn
parameter: (0 <= n <= 127)
compatibility: N/A
see also: JR, RP

explanation: Jumps to the specified command in the current command string or macro. Commands are numbered consecutively starting with 0.

```
example:     IE0,JP5,NO,1MR1000,1WS,1MR2000,1WS
                                        ;If accumulator equals 0 jump to
                                        ;1MR2000
```

Jump to command (relative) within MCCL sequence

MCCL command: JRn
parameter: (0 <= n <= 127)
compatibility: N/A
see also: JP, RP

explanation: Jumps forward or backward by the specified number of 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.

```
example:     1MR1000,1WS.005,IE0,JR-3   ;If accumulator equals 0 jump to
                                        ;1MR1000
```

Repeat the preceding command sequence

MCCL command: RPn
parameter: (0 <= n <= 2,147,483,647)
compatibility: N/A
see also: JP, RP

explanation: This command causes all the commands preceding the RP command to 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.

```
example:     TP,RP999                   ;Display the position of axis #1, 1000
                                        ;times
```

Wait (a period of time)

MCCL command: WAn
parameter: (0 <= n)
compatibility: N/A

see also: WS, WT

explanation: 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.

```
example:    1TP,WA0.1,RP9              ;Display the position of axis #1, 10
                                       ;times with a delay of one tenth of a
                                       ;second between displays
```

Wait for the Coarse Home input

MCCL command: WEn
parameter: (x = 0 or 1)
compatibility: MC200, MC210, MC260
see also: FE, FI, WI

explanation: Wait until the coarse home input on axis a is at the specified logic level, and then continue operation. If x is not specified or is 0, wait for coarse home to go active. If x is 1 wait for coarse home to go inactive. If this command was issued from an ASCII interface, it can be aborted by sending an Escape character.

Wait for digital channel off

MCCL command: WFn
parameter: (1 <= x)
compatibility: MC400
see also: WN

explanation: 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.

Wait for digital channel on

MCCL command: WNn
parameter: (1 <= x)
compatibility: MC400
see also: WF

explanation: 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.

Wait for encoder index mark

MCCL command: WIn
parameter: integer or real
compatibility: MC200, MC210, MC260

see also: FI, IA, WE

explanation: Wait until the index pulse has been observed on servo axis a. This command should be used after a Index Arm command has been issued to the axis, even if it is known that the index pulse has occurred (this command performs internal operations). To complete the indexing function, a Motor On (aMN) command should also be issued to the axis to re-initialize the position registers. If this command was issued from an ASCII interface, it can be aborted by sending an Escape character.

Wait for position (absolute)

MCCL command: WPn
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: IP, IR, WR

explanation: This command is used to delay command execution until an axis has reached a specific position. The position is specified by the command parameter 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 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.

Wait for position (relative)

MCCL command: WRn
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: IP, IR, WP

explanation: This command is used to delay command execution until an axis 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.

Wait for trajectory complete

MCCL command: WSn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: WA, WT

explanation: Will delay execution of the next command in the sequence until the trajectory generator for axis a (or all axes if axis specifier a = 0) has completed the current motion. The command parameter ‘n’ specifies an additional time period (in seconds) that the controller will wait before continuing execution of the commands following WS.

```
example:     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.

Note: Trajectory complete is a digital event that occurs when the Optimal Position (calculated by the DCX trajectory generator) equals the Current Position. At this point the Trajectory Complete (bit 3) status bit will be set. Any following error present during the move will cause the Trajectory Complete status bit **to be set before the axis has stopped moving**. The time parameter 'n' of the WS command allows the user to define the time required for the following error to equal 0

Wait for target reached

MCCL command: WTn
parameter: integer or real (0 <= n)
compatibility: MC200, MC210
see also: DB, DT, WA, WS

explanation: This command will delay command execution until axis a (or all axes if axis specifier a = 0) has reached its' target position. The command parameter n specifies an additional time period (in seconds) that the controller will wait before continuing execution of the commands following WT. The conditions for a servo to have reached its' target, is that it remains within the position DeadBand for the time period specified by the Delay at Target parameter 'n'. The condition for a stepper motor to have reached its' target is that the controller has output the last step of the motion. The Wait for Target command should not be used for axes in contour mode.

```
example:     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.

Miscellaneous Commands

disable DCX character echo

MCCL command: EF
parameter: (0 = 0, 1, 2 or 3)
compatibility: N/A
see also: EN

explanation: Causes 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 the host or terminal in half duplex mode. The parameter to this command selects the terminating character or characters that will be transmitted with command replies. The table shown with the Echo oN command lists the available options.

enable DCX character echo

MCCL command: EN
parameter: (0 = 0, 1, 2 or 3)
compatibility: N/A
see also: EF

explanation: Causes all characters received through an ASCII command port to be echoed to that port as received. Normally used when operating with the host or terminal in full duplex mode. The parameter to this command selects the terminating character or characters that will be transmitted with command replies. The table below list 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)

enable Decimal mode

MCCL command: DM
parameter: none
compatibility: N/A
see also: HM

explanation: 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. Placing the two commands in the example in the same command string would be incorrect.

enable Hexadecimal mode

MCCL command: HM
parameter: none
compatibility: N/A
see also: DM

explanation: 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. Placing the two commands in the previous example in the same command string would be incorrect. 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).

display the supported MCCL commands

MCCL command: HE
parameter: none
compatibility: N/A
see also: VE

explanation: Reports the valid DCX command mnemonics for the installed software version.

allocate Memory

MCCL command: ME
parameter: integer (1 <= n <=8192)
compatibility: N/A
see also: FM, RB, RW, RL, RD, WB, WW, WL, WD

explanation: Formats and allocates scratch pad memory. The first allocated memory address will be loaded into the accumulator.

free Memory

MCCL command: FM
parameter: integer
compatibility: N/A
see also: ME

explanation: Returns the memory space allocated by the ME command and returns it to the ‘heap’

enable a Prompt character

MCCL command: PCn
parameter: integer (0 <= n <= 255)
compatibility: N/A
see also: EF, EN

explanation: 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).

No operation

MCCL command: NO
parameter: none
compatibility: N/A
see also: BK

explanation: This command does nothing. It can be used to cause short delays in command line executions or as a filler in sequence commands.

DCX Reset

MCCL command: RT
parameter: none
compatibility: MC200, MC210, MC260
see also:

explanation: 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.

set the RS-232 baud rate

MCCL command: BRn
parameter: none
compatibility: N/A
see also: EF, EN, PC, XF, XN

explanation: Used to change the programmed baud rate for the RS-232 interface. The actual baud rate is determined by using the value n in a countdown circuit. The values for parameter ‘n’ for standard baud rates is shown below:

Baud Rate	*Parameter n =*
19,200	1
9,600	2
4,800	4
2,400	8
1,200	16
600	32
300	64

The values given are decimal numbers. If you are in hex mode, be sure to enter the hexadecimal equivalent, or momentarily change to decimal mode.

```
example:     BR8                        ;sets the baud rate to 2,400 baud
```

comment: This command takes effect immediately, so use with caution. Any value entered will result in some baud rate but not necessarily a standard one.

disable RS-232 hardware handshaking

MCCL command: HF
parameter: none
compatibility: N/A
see also: HN

explanation: Disables hardware handshake of serial communications through the RS-232 module.

disable RS-232 hardware handshaking

MCCL command: HN
parameter: none
compatibility: N/A
see also: HF

explanation: Enables hardware handshake of serial communications through the RS-232 module.

disable RS-232 XON/XOFF protocol

MCCL command: XF
parameter: none
compatibility: N/A
see also: HN, HN, XN

explanation: This command disables the XON/OFF handshaking protocol of the RS-232 serial port. This command has no effect when issued through the host PC or IEEE-488 command interfaces.

enable RS-232 XON/XOFF protocol

MCCL command: XN
parameter: none
compatibility: N/A
see also: HF, HN, XN

explanation: This command enables the XON/XOFF handshaking protocol of the RS-232 serial port. This command has no effect when issued through the host PC or IEEE-488 command interfaces.

Chapter Contents

- Motherboard: DCX-VM200
- DCX-MC200 - +/- 10 Volt Analog Servo Motor Control Module
- DCX-MC210 - PWM Motor Drive Servo Control Module
- DCX-MC260 - Stepper Motor Control Module
- DCX-MC400 - 16 channel Digital I/O Module
- DCX-MC5X0 - Analog I/O Module
- DCX-MF300 - RS-232 Communications Interface Module
- DCX-MF310 - IEEE-488 Communications Interface Module

Chapter 10

DCX Specifications

Motherboard: DCX-VM200

Function	6 Axis Motion Controller
Installation	VME Bus Host Computer
Configuration	6 User Installed Modules
Main Processor	Motorola 68020
Processor Clock	16 MHz
Code Memory	128k x 16 bit Flash Memory
Data Memory	128k x 16 bit Dual Ported Ram (with battery backup option)
Processor Fault Detection	Watch Dog Circuit with Reset Relay
Status LED's	Power, Reset, Watch Dog, (6) Motor Error
Standard Communication Interface	VME bus 4 Kilobytes dual ported memory in Memory Address Space Jumper/rotary switch selection of base address
Optional Communication Interfaces	RS-232 Serial Port (Network capable) IEEE-488 Bus
Supply Voltages	+5,+12 and -12 vdc
Form Factor	6U VME card (6.8" x 10.3" including the front panel)
Operating Temperature range	0 degrees C to 60 degrees C
Weight	14 oz + 1.2 oz per module (approx.)

DCX-MC200 - +/- 10 Volt Analog Servo Motor Control Module

Function	Closed Loop Servo Controller with Dual Encoder Inputs
Installation	DCX-VM200 Motion Control Motherboard
Operating Modes	Position, Velocity, Contouring, Torque, Gain, and Joystick
Filter Algorithm	PID with Velocity and Acceleration Feed-Forwards
Filter Update Rate	1, 2 or 4 KHz (Software Selectable)
Trajectory Generator	Trapezoidal, Parabolic or S-Curve Independent Acceleration and Deceleration
Position Feedback	Incremental Encoder with Index
Position and Velocity Resolution	32 bit
Output	Analog Signal (+/- 10 vdc @ 10 ma, 12 bit)
Encoder and Index Inputs	Differential or single ended, -7 to +7 vdc max.
Encoder Count Rate	1,000,000 Quadrature Counts / Sec.
Encoder Supply Voltage	+5 or +12 vdc, jumper selectable
Axis Inputs	Limit+, Limit-, Coarse Home, Amplifier Fault (TTL compatible, optical isolation available on BF100 interconnect board)
Axis Outputs	Amplifier Enable (TTL compatible)
Jog Control Input	Analog (0 to 5 volts)
General purpose inputs	and 2 User Inputs
Operating Temperature range	0 degrees C to 60 degrees C

DCX-MC210 - PWM Motor Drive Servo Control Module

Function	Closed Loop Servo Controller with Dual Encoder Inputs
Installation	DCX-VM200 Motion Control Motherboard
Operating Modes	Position, Velocity, Contouring, Torque, Gain, and Joystick
Filter Algorithm	PID with Velocity and Acceleration Feed-Forwards
Filter Update Rate	1, 2 or 4 KHz (software selectable)
Trajectory Generator	Trapezoidal, Parabolic or S-Curve Independent Acceleration and Deceleration
Position Feedback	Incremental Encoder with Index
Position and Velocity Resolution	32 bit
Output	PWM (12 volt @ 1A), 23.4 KHz, 8 bit
Encoder and Index Inputs	Differential or single ended, -7 to +7 vdc max.
Encoder Count Rate	1,000,000 Quadrature Counts / Sec.
Encoder Supply Voltage	+5 or +12 vdc, jumper selectable
Axis Inputs	Limit+, Limit-, Coarse Home, Amplifier Fault (TTL compatible, optical isolation available on BF100 interconnect board)
Axis Outputs	Amplifier Enable (TTL compatible)
Jog Control Input	Analog (0 to 5 volts)
General purpose inputs	and 2 User Inputs
Operating Temperature range	0 degrees C to 60 degrees C

DCX-MC260 - Stepper Motor Control Module

Function	Open or Closed Loop Stepper Controller
Installation	DCX-VM200 Motion Control Motherboard
Operating Modes	Position, Velocity, Contouring, and Joystick
Trajectory Generator	Trapezoidal, Parabolic or S-Curve Independent Acceleration and Deceleration
Position Feedback	Incremental Encoder with Index (closed loop only)
Position and Velocity Resolution	32 bit
Step Outputs	Pulse/Direction – CW/CCW (software selectable), open collector drivers 50% duty cycle
Step Rates (Software Selectable)	High Speed - 1.0K Steps / Sec. - 1.0M Steps / Sec. Medium Speed - 125 Steps / Sec. - 156K Steps / Sec. Low Speed - 15 Steps / Sec. - 19.5K Steps / Sec.
Aux. Encoder and Index Inputs	Differential or single ended, -7 to +7 vdc max.
Aux. Encoder Count Rate	1,000,000 Quadrature Counts / Sec.
Aux. Encoder Supply Voltage	+5 or +12 vdc, jumper selectable
Axis Inputs	Home, Limit+, Limit-, Null (TTL compatible, optical isolation available on BF160 interconnect board)
Axis Outputs	Driver Enable, Full/Half Step, Full/Half Current, Stopped (TTL compatible)
Jog Control Input	Analog (0 to 5 volts)
General purpose inputs	and 2 User Inputs
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-VM200 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.
Relay rack interface	DCX-BF022
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-VM200 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
Operating Temperature range	0 degrees C to 60 degrees C

DCX-MF300 - RS-232 Communications Interface Module

Function	RS-232 Communications Interface module
Installation	DCX-VM200 Motion Control Motherboard
Baud Rates	300 - 19,200
Handshake Protocol	Hardware or XON-XOFF
Operating Temperature range	0 degrees C to 60 degrees C

DCX-MF310 - IEEE-488 Communications Interface Module

Function	IEEE-488 Communications Interface module
Installation	DCX-VM200 Motion Control Motherboard
Address Selection	DIP switch on module
Data Bus Drivers	Push-Pull or Open Collector
Operating Temperature range	0 degrees C to 60 degrees C

Chapter Contents

Chapter 11

Connectors, Jumpers, and Schematics

DCX-VM200 Motion Control Motherboard

(Refer to diagram at the end of this appendix)

LED Status Indicators

LED #	*Color*	*Description*
1	Green	+5V logic supply
2	Yellow	DCX Reset active
3	Yellow	Watchdog circuit tripped
4	Red	Module #1 motor error (following error or limit tripped)
5	Red	Module #2 motor error (following error or limit tripped)
6	Red	Module #3 motor error (following error or limit tripped)
7	Red	Module #4 motor error (following error or limit tripped)
8	Red	Module #5 motor error (following error or limit tripped)
9	Red	Module #6 motor error (following error or limit tripped)

DCX-VM200 Connectors

Battery Backup Input Connector – JP16

Pin #	*Description*
1	Battery voltage input (+3.0 VDC nominal)
2	Ground

DCX-VM200 Configuration Jumpers – configuration in **bold type** denotes default factory shipping configuration

JP1 – CPU Clock Select

Pins	*Description*
1 to 2	**16 MHz CPU clock**
2 to 3	8 MHz CPU clock

JP2 – RAM Access Delay Select

Pins	*Description*
1 to 2	160 nano second delay
3 to 4	**120 nano second delay**
5 to 6	80 nano second delay

JP3 – DCX Reset Source

Pins	*Description*
1 to 2	Enable software command ‘latched’ reset
3 to 4	**Enable front panel reset switch**
5 to 6	**Enable VME bus reset signal**

JP4 – Boot RAM enable

Pins	*Description*
1 to 2	Boot code from static memory
open	**Boot code from FLASH devices**

JP5 – FLASH memory select

Pins	*Description*
1 to 2	**1024K FLASH memory**
2 to 3	Reserved

JP6 – FLASH Write select

Pins	*Description*
2 to 4	**Enable FLASH Write (wire wrap)**

DCX-VM200 Configuration Jumpers – continued

JP7 – Reserved for factory use

JP8 – VME Bus Interface base memory address select

Base address	*JP8 7 to 8*	*JP8 5 to 6*	*JP8 3 to 4*	*JP8 1 to 2*
000000 hex	connected	connected	connected	connected
100000 hex	**connected**	**connected**	**connected**	**open**
200000 hex	connected	connected	open	connected
300000 hex	connected	connected	open	open
400000 hex	connected	open	connected	connected
500000 hex	connected	open	connected	open
600000 hex	connected	open	open	connected
700000 hex	connected	open	open	open
800000 hex	open	connected	connected	connected
900000 hex	open	connected	connected	open
A00000 hex	open	connected	open	connected
B00000 hex	open	connected	open	open
C00000 hex	open	open	connected	connected
D00000 hex	open	open	connected	open
E00000 hex	open	open	open	connected
F00000 hex	open	open	open	open

JP9 – VME Bus Interface Interrupt (select Interrupt level)

Interrupt Level	*JP9 5 to 6*	*JP9 3 to 4*	*JP9 1 to 2*
0	connected	connected	connected
1	**connected**	**connected**	**open**
2	connected	open	connected
3	connected	open	open
4	open	connected	connected
5	open	connected	open
6	open	open	connected
7	open	open	open

JP10 – VME Bus Interface Disable

Pins	*Description*
open	Disable VME bus Interface
1 to 2	**Enable VME Bus Interface**

JP11 – CPU Cache Disable

Pins	*Description*
1 to 2	**Enable CPU Cache**
open	Disable CPU cache

DCX-VM200 Configuration Jumpers – continued

JP12 – Watchdog Enable

Pins	*Description*
1 to 2	**Enable Watchdog circuit**
open	Disable Watchdog Circuit

JP13, JP14, JP15 – FLASH memory jumper configuration - settings for 1 Meg. FLASH memory devices:

Jumper	*Pins*	*Description*
JP13	**3 to 4**	**FLASH Memory, 1M**
JP14	**1 to 2**	**FLASH Memory, 1M**
JP15	**2 to 3**	**FLASH Memory, 1M**

JP17 & U18 Memory type configuration - Factory setting for 128K RAMS:

Pins	*Description*
1 to 2	**128K Static Memory**
2 to 3	Reserved

DCX Memory Address Rotary Switch Setting

The 16 bit position rotary switch selects where the dual port RAM appears in the host computer's memory map. In the table below, the X digit in the 'Host Address Range', is determined by the setting of jumper JP8. The default factory setting of the switch is 0. If multiple boards are used in a single host, no two boards should have the same switch setting.

Switch Setting	*Host Address Range*
0	**X0000h – X0FFFh**
1	X1000h – X1FFFh
2	X2000h – X2FFFh
3	X3000h – X3FFFh
4	X4000h – X4FFFh
5	X5000h – X5FFFh
6	X6000h – X6FFFh
7	X7000h – X7FFFh
8	X8000h – X8FFFh
9	X9000h – X9FFFh
A	XA000h – XAFFFh
B	XB000h – XBFFFh
C	XC000h – XCFFFh
D	XD000h – XDFFFh
E	XE000h – XEFFFh
F	Not a valid address – used to clear macro memory

DCX-VM200 Auxiliary Connectors

Asynchronous Serial I/O connector J1

Pin #	*Description*
1	Ground
2	Serial data input (TTL level)
3	Ground
4	Serial data output (TTL level)
5	Ground
6	Transmit enable (TTL level)
7	+5 VDC
8	+12 VDC
9	-12 VDC
10	NC

Mating Connector:10-pin dual-row IDC female, Circuit Assembly P/N CA-10IDS2-F-SPT or equivalent

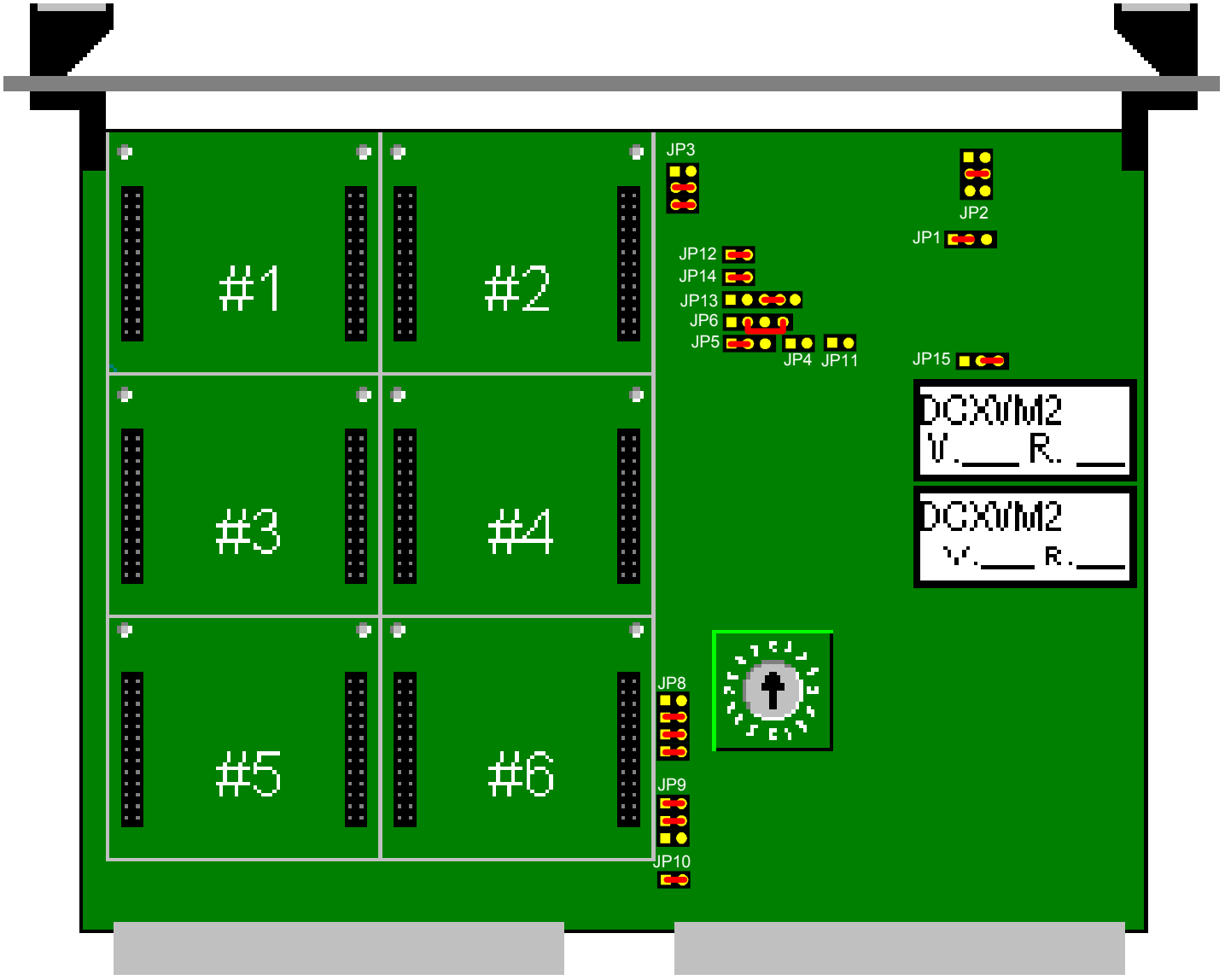
Auxiliary I/O connector J2

Pin #	*Description*
1	Reserved for factory use
2	Reserved for factory use
3	Ground
4	Reserved for factory use
5	Ground
6	Reserved for factory use
7	Ground
8	Reserved for factory use
9	+5 VDC
10	Reserved for factory use
11	Counter input
12	Reserved for factory use
13	Watchdog relay contact 1 (normally closed)
14	Watchdog relay contact 2 (normally closed

Mating Connector: 14-pin dual-row IDC female, Circuit Assembly P/N CA-14IDS2-F-SPT or equivalent

VME Bus Connector P1

Pin Number	Row A	Row B	Row C
1	D00	-----	D08
2	D01	-----	D09
3	D02	-----	D10
4	D03	-----	D11
5	D04	-----	D12
6	D05	-----	D13
7	D06	-----	D14
8	D07	-----	D15
9	Gnd	-----	Gnd
10	-----	-----	-----
11	Gnd	-----	-----
12	DS1	-----	SYSREST
13	DS0	-----	-----
14	Write	-----	-----
15	Gnd	-----	A23
16	DTACK	-----	A22
17	Gnd	-----	A21
18	as	-----	A20
19	Gnd	-----	A19
20	IACK	Gnd	A18
21	IACKIN	-----	A17
22	IACKOUT	-----	A16
23	-----	Gnd	A15
24	A07	1RQ7	A14
25	A06	1RQ6	A13
26	A05	1RQ5	A12
27	A04	1RQ4	A11
28	A03	1RQ3	A10
29	A02	1RQ2	A09
30	A01	1RQ1	A08
31	-12V	-----	+12V
32	+5V	+5V	+5V



= Default jumpering

DCX-MC200 +/- 10V Servo Motor Control Module

SIGNAL DESCRIPTIONS:

Analog Command Return
connection point: 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 return where this signal should be connected.

Analog Command Output
connection point: J3 - pin 2
signal type: +/- 10V analog, 12 bit
notes: connects to servo amplifier motor command input
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.

Direction / PWM Output
connection point: J3 - pin 7
signal type: TTL output
notes:
explanation:
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. The voltage on this output is TTL compatible. This means that it will be between 0 and 0.4 volts (low) to indicate one direction, and between 2.4 and 5.0 volts (high) for the opposite direction. The maximum sink current for this signal when it is low is 4.0 mA, the maximum source current when it is high is 1.0 mA.

PWM Output – For servo drives requiring a TTL level PWM command signal. The Analog Command Output (J3 pin 2) is used as the direction signal. The frequency of the PWM is 1.4648 KHz. For a description see the description of **Laser Cutting Application Solutions** chapter.

Coarse Home Input
connection point: J3 - pin 9
signal type: TTL input
notes: 4.7K pull up resistor is connected to the +5V logic supply

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.

Amplifier Fault Input
connection point: J3 - pin 10
signal type: TTL input
notes: 4.7K pull up resistor is connected to the +5V logic supply
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. Using the Fault oN command, this signal can be enabled to shut the axis off if the input goes active low. In this condition, no further servo motion will occur until the fail signal is deactivated and the Motor oN command is issued. The Fault oFf command can be used to disable this signal.

Amplifier Enable Output
connection point: J3 - pin 11
signal type: TTL output
notes: 2ma sink / source
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 Motor oN command is issued to the DCX, 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.

User Input 1 and User Input 2
connection point: J3 - pin 12 (User Input #1), J3 - pin 13 (User Input #2)
signal type: TTL input
notes: 4.7K pull up resistor is connected to the +5V logic supply
explanation: These module inputs can be connected to any digital logic signals that the DCX needs to monitor. The state of these inputs (high or low) is recorded in the associated bits of the motor status. These signals have no built in control function in the DCX. It is suggested that these inputs be reserved for monitoring signals related to the respective servo axis.

Limit Positive and Limit Negative Inputs
connection point: J3 - pin 14 (Limit Positive), J3 - pin 15 (Limit Negative)
signal type: TTL input
notes: 4.7K pull up resistor is connected to the +5V logic supply
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 the can be configured by the Limit Mode command. The limit switch inputs can be enabled and disabled with the Limits oN and Limits oFf commands respectively. See the description of **Motion Limits** in the **Motion Control** chapter.

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 that supplies position feedback information for the servo. The plus (+) and minus (-) signs refer to the two sides of differential inputs. By setting jumpers JP1 and JP2 appropriately, the plus signal inputs can be configured for single ended inputs.

Auxiliary Encoder Inputs (Phase A, Phase B, Index+, Index-)
connection point: see pin out table
signal type: TTL or Differential driver output (-7V to +7V)
notes:
explanation: - These input signals can be used for an auxiliary encoder.

Encoder Power Output
connection point: J3 pin 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
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.

Joystick Input (A/D Channel, 8 bit)
connection point: J4 pin 1
signal type: 0.0 to +5V analog input
notes: used for joystick control
explanation: This input is used to implement manual jogging of the axis. See the description of **Jogging** in the **Motion Control** chapter.

A/D +5 Volt Reference Output
connection point: J4 pin 3
signal type: precision +5V reference voltage
notes: used for joystick control
explanation: This input is used to implement manual jogging of the axis. See the description of **Jogging** in the **Motion Control** chapter.

Analog Ground
connection point: J4 pin 4
signal type: analog ground for A/D conversion
notes: used for joystick control
explanation: This input is used to implement manual jogging of the axis. See the description of **Jogging** in the **Motion Control** chapter.

DCX-MC200 Module connectors

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
4	-12 VDC
5	Ground
6	+5 VDC
7	Direction / PWM Output (TTL level)**
8	Primary Encoder Index + (input, active high)
9	Coarse Home (input, active low, with 4.7K ohm pull-up to +5V)
10	Amplifier Fault (input, active low, with 4.7K ohm pull-up to +5V)
11	Amplifier Enable (output, active low, TTL level)**
12	User input 1 (input, active low, with 4.7K ohm pull-up to +5V)
13	User input 2 (input, active low, with 4.7K ohm pull-up to +5V)
14	Limit Positive (input, active low, with 4.7K ohm pull-up to +5V)
15	Limit Negative (input, active low, with 4.7K ohm pull-up to +5V)
16	Primary Encoder Phase A+ (input)*
17	Encoder Power (+5VDC or +12VDC, see jumper JP3)
18	Auxiliary Encoder Index - (input, active low)
19	Primary Encoder Phase A- (input)
20	Primary Encoder Phase B- (input)
21	Auxiliary Encoder Phase A
22	Auxiliary Encoder Phase B
23	Primary Encoder Phase B+ (input)*
24	Auxiliary Encoder Index+ (input, active high)
25	Primary Encoder Index- (input, active low)
26	Ground

* Use A+ and B+ for single-ended ENCODER INPUTS
** These signals are not suitable for directly driving optically isolated inputs.

Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N CA-26IDS2-F-SPT or equivalent

J4 connector pin out (A/D channels for jogging)

Pin #	Description
1	Analog Input #1
2	Analog Input #2
3	+5 volt reference output
4	Analog ground

DCX-MC200V P2 connector pin-out – Installed in module position 1, 3, and/or 5

P2 Connector Pin Number	Row A (module #1)	Row B (module #3)	Row C (module #5)
17	Analog Return	Analog Return	Analog Return
18	Analog Com. +/-10 V	Analog Com. +/-10 V	Analog Com. +/-10 V
19	Primary Encoder A-	Primary Encoder A-	Primary Encoder A-
20	Primary Encoder B-	Primary Encoder B-	Primary Encoder B-
21	Primary Encoder Index +	Primary Encoder Index +	Primary Encoder Index +
22	Primary Encoder A+	Primary Encoder A+	Primary Encoder A+
23	Reserved	Reserved	Reserved
24	Primary Encoder Index-	Primary Encoder Index-	Primary Encoder Index-
25	Coarse Home	Coarse Home	Coarse Home
26	Amplifier Fault	Amplifier Fault	Amplifier Fault
27	Amplifier Enable	Amplifier Enable	Amplifier Enable
28	Reserved	Reserved	Reserved
29	Reserved	Reserved	Reserved
30	Limit Positive	Limit Positive	Limit Positive
31	Limit Negative	Limit Negative	Limit Negative
32	Primary Encoder B+	Primary Encoder B+	Primary Encoder B+

DCX-MC200V P2 connector pin-out – Installed in module position 2, 4, and/or 6

P2 Connector Pin Number	Row A (module #2	Row B (module #4)	Row C (module #6)
1	Analog Return	Analog Return	Analog Return
2	Analog Com. +/-10 V	Analog Com. +/-10 V	Analog Com. +/-10 V
3	Primary Encoder A-	Primary Encoder A-	Primary Encoder A-
4	Primary Encoder B-	Primary Encoder B-	Primary Encoder B-
5	Primary Encoder Index +	Primary Encoder Index +	Primary Encoder Index +
6	Primary Encoder A+	Primary Encoder A+	Primary Encoder A+
7	Reserved	Reserved	Reserved
8	Primary Encoder Index-	Primary Encoder Index-	Primary Encoder Index-
9	Coarse Home	Coarse Home	Coarse Home
10	Amplifier Fault	Amplifier Fault	Amplifier Fault
11	Amplifier Enable	Amplifier Enable	Amplifier Enable
12	Reserved	Reserved	Reserved
13	Reserved	Reserved	Reserved
14	Limit Positive	Limit Positive	Limit Positive
15	Limit Negative	Limit Negative	Limit Negative
16	Primary Encoder B+	Primary Encoder B+	Primary Encoder B+

DCX-MC200 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-

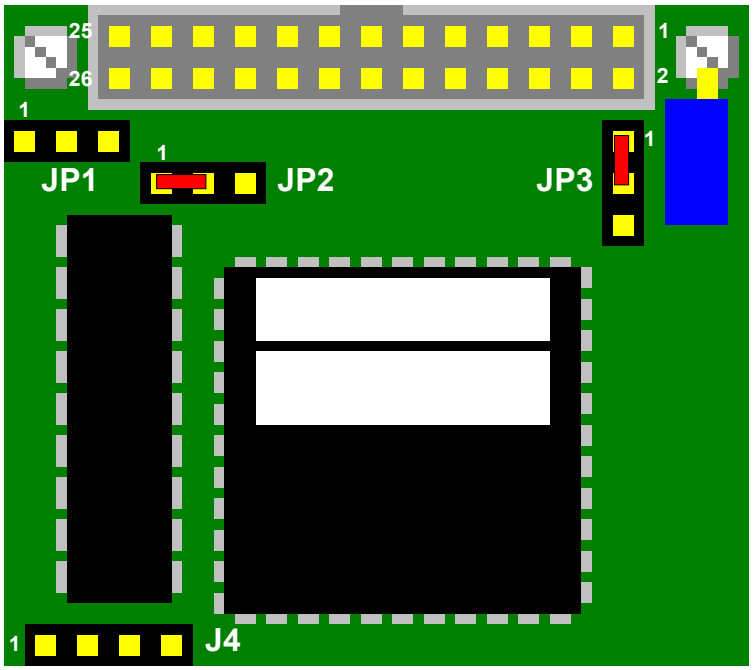
JP3 – Encoder Power Select (+5VDC or +12 VDC)

Pins	*Description*
1 to 2	**+5 VDC encoder supply on J3 pin 17**
2 to 3	+12 VDC encoder supply on J3 pin 17

DCX-MC200 Module Output Offset Potentiometers

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-MC200 Module Layout



DCX-MC210 PWM Motor Drive Servo Control Module

SIGNAL DESCRIPTIONS:

Motor Drive Outputs
connection point: J3 - pin 1 (Motor Drive +), J3 – pin 6 (Motor Drive -)
signal type: TTL output
notes:
explanation: These module outputs provide the PWM drive signal for a DC servo motor. The PWM frequency is 31.25 KHz. The resolution of the PWM is a full eight bits, resulting in .0390625 volts per DAC unit. Rotational direction is determined by connecting the Motor Drive signals (Motor - and Motor +) to the appropriate terminals on the DC servo motor.

Coarse Home Input
connection point: J3 - pin 9
signal type: TTL input
notes: 4.7K pull up resistor is connected to the +5V logic supply
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.

Amplifier Fault Input
connection point: J3 - pin 10
signal type: TTL input
notes: 4.7K pull up resistor is connected to the +5V logic supply
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. Using the Fault oN command, this signal can be enabled to shut the axis off if the input goes active low. In this condition, no further servo motion will occur until the fail signal is deactivated and the Motor oN command is issued. The Fault oFf command can be used to disable this signal.

Amplifier Enable Output
connection point: J3 - pin 11
signal type: TTL output
notes: 2ma sink / source
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 Motor oN command is issued to the DCX, 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.

Limit Positive and Limit Negative Inputs
connection point: J3 - pin 14 (Limit Positive), J3 - pin 15 (Limit Negative)
signal type: TTL input

notes: 4.7K pull up resistor is connected to the +5V logic supply
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 the can be configured by the Limit Mode command. The limit switch inputs can be enabled and disabled with the Limits oN and Limits oFf commands respectively. See the description of **Motion Limits** in the **Motion Control** chapter.

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 that supplies position feedback information for the servo. The plus (+) and minus (-) signs refer to the two sides of differential inputs. By setting jumpers JP1 and JP2 appropriately, the plus signal inputs can be configured for single ended inputs.

Auxiliary Encoder Inputs (Phase A, Phase B, Index+, Index-)
connection point: see pin out table
signal type: TTL or Differential driver output (-7V to +7V)
notes:
explanation: - These input signals can be used for an auxiliary encoder.

Encoder Power Output
connection point: J3 pin 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
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.

Joystick Input (A/D Channel, 8 bit)
connection point: J4 pin 1
signal type: 0.0 to +5V analog input
notes: used for joystick control
explanation: This input is used to implement manual jogging of the axis. See the description of **Jogging** in the **Motion Control** chapter.

A/D +5 Volt Reference Output
connection point: J4 pin 3
signal type: precision +5V reference voltage
notes: used for joystick control

explanation: This input is used to implement manual jogging of the axis. See the description of **Jogging** in the **Motion Control** chapter.

Analog Ground
connection point: J4 pin 4
signal type: analog ground for A/D conversion
notes: used for joystick control
explanation: This input is used to implement manual jogging of the axis. See the description of **Jogging** in the **Motion Control** chapter.

DCX-MC210 Module connectors

J3 connector pin out (Motor command, encoders, and axis I/O)

Pin #	*Description*
1	PWM Motor Drive + (output, 500ma max.)
2	Encoder Power (+5VDC or +12VDC, see jumper JP3)
3	Primary Encoder Phase B+ (input)*
4	Primary Encoder Phase A+ (input)*
5	Ground
6	PWM Motor Drive - (output, 500ma max.)
7	Ext. Motor Power + (optional) ****
8	Primary Encoder Index + (input, active high) / *** Ext. Motor Power - (optional) ****
9	Coarse Home (input, active low, with 4.7K ohm pull-up to +5V)
10	Amplifier Fault (input, active low, with 4.7K ohm pull-up to +5V)
11	Amplifier Enable (output, active low, TTL level)**
12	Reserved
13	Reserved
14	Limit Positive (input, active low, with 4.7K ohm pull-up to +5V)
15	Limit Negative (input, active low, with 4.7K ohm pull-up to +5V)
16	Primary Encoder Phase A+ (input)*
17	Encoder Power (+5VDC or +12VDC, see jumper JP3)
18	Auxiliary Encoder Index - (input, active low)
19	Primary Encoder Phase A- (input)
20	Primary Encoder Phase B- (input)
21	Auxiliary Encoder Phase A
22	Auxiliary Encoder Phase B
23	Primary Encoder Phase B+ (input)*
24	Auxiliary Encoder Index+ (input, active high)
25	Primary Encoder Index- (input, active low)
26	Ground

* Use A+ and B+ for single-ended Encoder inputs
** These signals are not suitable for directly driving optically isolated inputs.
*** Selected by JP5
**** For use when +12VDC supplied to DCX motherboard is not used for motor supply

Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N CA-26IDS2-F-SPT or equivalent

J4 connector pin out (A/D channels for jogging)

Pin #	*Description*
1	Analog Input #1
2	Analog Input #2
3	+5 volt reference output
4	Analog ground

DCX-MC210V P2 connector pin-out – Installed in module position 1, 3, and/or 5

P2 Connector Pin Number	Row A (module #1)	Row B (module #3)	Row C (module #5)
17	PWM Motor Drive +	PWM Motor Drive +	PWM Motor Drive +
18	Encoder Power	Encoder Power	Encoder Power
19	Primary Encoder B+	Primary Encoder B+	Primary Encoder B+
20	Primary Encoder A+	Primary Encoder A+	Primary Encoder A+
21	Primary Encoder Index-	Primary Encoder Index-	Primary Encoder Index-
22	PWM Motor Drive -	PWM Motor Drive -	PWM Motor Drive -
23	Reserved	Reserved	Reserved
24	Primary Encoder Index +	Primary Encoder Index +	Primary Encoder Index +
25	Coarse Home	Coarse Home	Coarse Home
26	Amplifier Fault	Amplifier Fault	Amplifier Fault
27	Amplifier Enable	Amplifier Enable	Amplifier Enable
28	Reserved	Reserved	Reserved
29	Reserved	Reserved	Reserved
30	Limit Positive	Limit Positive	Limit Positive
31	Limit Negative	Limit Negative	Limit Negative
32	Reserved	Reserved	Reserved

DCX-MC210V P2 connector pin-out – Installed in module position 2, 4 and/or 6

P2 Connector Pin Number	Row A (module #2	Row B (module #4)	Row C (module #6)
1	PWM Motor Drive +	PWM Motor Drive +	PWM Motor Drive +
2	Encoder Power	Encoder Power	Encoder Power
3	Primary Encoder B+	Primary Encoder B+	Primary Encoder B+
4	Primary Encoder A+	Primary Encoder A+	Primary Encoder A+
5	Primary Encoder Index-	Primary Encoder Index-	Primary Encoder Index-
6	PWM Motor Drive -	PWM Motor Drive -	PWM Motor Drive -
7	Reserved	Reserved	Reserved
8	Primary Encoder Index +	Primary Encoder Index +	Primary Encoder Index +
9	Coarse Home	Coarse Home	Coarse Home
10	Amplifier Fault	Amplifier Fault	Amplifier Fault
11	Amplifier Enable	Amplifier Enable	Amplifier Enable
12	Reserved	Reserved	Reserved
13	Reserved	Reserved	Reserved
14	Limit Positive	Limit Positive	Limit Positive
15	Limit Negative	Limit Negative	Limit Negative
16	Reserved	Reserved	Reserved

DCX-MC210 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 17**
2 to 3	+12 VDC encoder supply on J3 pin 17

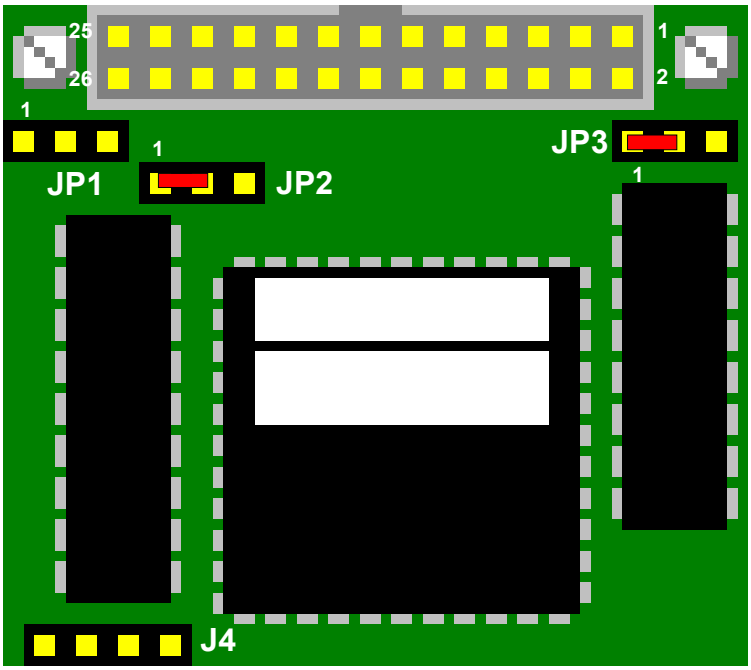
JP4 – Motor Supply + (+12 VDC PC Power supply or external + supply)

Pins	*Description*
1 to 2	**+5 VDC encoder supply on J3 pin 17**
2 to 3 cut trace JP4 1 to 2 (bottom side)	Use external supply voltage (connected to J3 pin 7)

JP5 – Motor Supply - (PC Ground or external supply -)

Pins	*Description*
1 to 2	**PC ground**
2 to 3 cut traces JP5 1 to 2 JP5 3 to 4 (bottom side)	Use external supply voltage – (connected to J3 pin 8)

DCX-MC210 Module Layout



DCX-MC260 Stepper Motor Control Module

SIGNAL DESCRIPTIONS:

Pulse and Direction Outputs
connection point: J3 - pin 3 (Direction/CW), J3 – pin 4 (Pulse/CCW)
signal type: open collector driver
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.

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 goes low at the beginning of a step. It stays low for one half the step period (ie. the time before the next pulse), and then goes back high. When it is time for the next step, the signal will go low again. The Direction signal selects which direction the motor will move. When this signal is high, the stepper controller will decrement the current position for every step taken, and when the signal is low, it will increment the current position for every step taken. Both of these signals have high current open collector drivers on the module 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.

The Output Mode command 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.

Stopped (motion complete) Output
connection point: J3 - pin 7
signal type: TTL output
notes:
explanation: The Stopped signal is a status output from the stepper module, indicating when the motor is stepping. At the beginning of a motion, the Stopped signal is brought high. It will continue to stay high for the duration of that motion. At the end of the motion, the Stopped signal is brought low again. The high to low transition on Stopped indicates when the motion is over, and the low level indicates that the motor is no longer moving.

The Stopped signal may be used as a motion complete indication, it may also control the power selection for the stepper motor power driver, switching automatically between a high current while stepping, and low current while stopped. This will reduce power dissipation of the motor when it isn't moving.

Limit Positive and Limit Negative Inputs
connection point: J3 - pin 8 (Limit Positive), J3 - pin 9 (Limit Negative)
signal type: TTL input
notes: 4.7K pull up resistor is connected to the +5V logic supply
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 the can be configured by the Limit Mode command. The limit switch inputs can be enabled and disabled with the Limits oN and Limits oFf commands respectively. See the description of **Motion Limits** in the **Motion Control** chapter.

Home Input
connection point: J3 - pin 13
signal type: TTL input
notes: 4.7K pull up resistor is connected to the +5V logic supply
explanation: This input is used to set the stepper motors zero position. It is typically connected to a switch that is activated at a fixed position in the motors motion range.

Full/Half Step Output
connection point: J3 - pin 14
signal type: TTL output
notes: 2ma sink / source
explanation: If the motor driver that this module is controlling has a digital input to select between full and half step modes, this module output is used to select between the modes. This output will be set low by the Step Full (SF) command or high by the Step Half (SH) commands.

Full/Half Current Output
connection point: J3 - pin 15
signal type: TTL output
notes: 2ma sink / source
explanation: If the motor driver that the module is controlling has a digital current control signal input, this module output can be used to control the motor drivers output current. This output will be set low by the Full Current (FC) command or high by the Half Current (HC) commands. Normally a stepper motor driver will be set for full current while it is moving and half current while holding to reduce motor heating.

Motor On Output
connection point: J3 - pin 16
signal type: TTL output
notes: 2ma sink / source
explanation: This module output will go low when the Motor oN (MN) command is issued. ***It will go high when the Motor oFf (MF) command is issued, the controller is reset, or a limit switch input is activated***. This signal can be connected to the enable signal input of the stepper driver so that it can be disabled by issuing commands to the DCX.

Null Input
connection point: J3 - pin 17
signal type: TTL input
notes: 4.7K pull up resistor is connected to the +5V logic supply
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.

Auxiliary Encoder Inputs (Phase A & A+, Phase B & B+, 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 Coarse Home Input
connection point: J3 - pin 23
signal type: TTL input
notes: 4.7K pull up resistor is connected to the +5V logic supply
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.

SUPPLY CONNECTIONS (+5, +12, -12, GROUND) - These module pins provide access to the DCX supply voltages.

Joystick Input (A/D Channel, 8 bit)
connection point: J4 pin 1
signal type: 0.0 to +5V analog input
notes: used for joystick control
explanation: This input is used to implement manual jogging of the axis. This signal is also available on connector J3 pin 10. See the description of **Jogging** in the **Motion Control** chapter.

A/D +5 Volt Reference Output
connection point: J4 pin 3
signal type: precision +5V reference voltage
notes: used for joystick control
explanation: This input is used to implement manual jogging of the axis. See the description of **Jogging** in the **Motion Control** chapter.

Analog Ground
connection point: J4 pin 4
signal type: analog ground for A/D conversion
notes: used for joystick control
explanation: This input is used to implement manual jogging of the axis. See the description of **Jogging** in the **Motion Control** chapter.

DCX-MC260 Module connectors

J3 connector pin out (Motor command, encoders, and axis I/O)

Pin #	Description
1	Ground
2	+5 VDC
3	Direction or CW Pulse (output, active low, open collector driver, 100 ma max.)*
4	Pulse or CCW Pulse (output, active low, open collector driver, 100 ma max.)*
5	Reserved
6	Reserved
7	Stopped (output, high while moving, TTL level)**
8	Limit Positive (input, active low, with 4.7K ohm pull-up to +5V)
9	Limit Negative (input, active low, with 4.7K ohm pull-up to +5V)
10	Jog Input (connected to J4 pin 1)
11	Reserved
12	Reserved
13	Home (input, active low, with 4.7K ohm pull-up to +5V)
14	Full/Half Step (output, low when full stepping, TTL level)**
15	Full/Half Current (output, low when moving, TTL level)**
16	Motor On (output, low when motor on, TTL level)**
17	Null Position (input, active low, with 4.7K ohm pull-up to +5V)
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	Auxiliary Encoder Index- (input, active low)
23	Auxiliary Encoder Coarse Home (input, active low, with 4.7K ohm pull-up to +5V)
24	+12 VDC
25	-12 VDC
26	Ground

* These signals default to DIRECTION and PULSE, use Output Mode command to change to CW and CCW PULSE.

** These signals are not suitable for directly driving optically isolated inputs.

Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N CA-26IDS2-F-SPT or equivalent

J4 connector pin out (A/D channels for jogging)

Pin #	Description
1	Analog Input #1
2	Analog Input #2
3	+5 volt reference output
4	Analog ground

DCX-MC260V P2 connector pin-out – Installed in module position 1, 3, and/or 5

P2 Connector Pin Number	Row A (module #1)	Row B (module #3)	Row C (module #5)
17	Ground	Ground	Ground
18	+5 VDC	+5 VDC	+5 VDC
19	Direction / CW Pulse	Direction / CW Pulse	Direction / CW Pulse
20	Pulse / CCW Pulse	Pulse / CCW Pulse	Pulse / CCW Pulse
21	Reserved	Reserved	Reserved
22	Reserved	Reserved	Reserved
23	Limit Positive	Limit Positive	Limit Positive
24	Limit Negative	Limit Negative	Limit Negative
25	Reserved	Reserved	Reserved
26	Reserved	Reserved	Reserved
27	Reserved	Reserved	Reserved
28	Reserved	Reserved	Reserved
29	Home	Home	Home
30	Full / Half Step	Full / Half Step	Full / Half Step
31	Full / Half Current	Full / Half Current	Full / Half Current
32	Motor On	Motor On	Motor On

DCX-MC260V P2 connector pin-out – Installed in module position 2, 4, and/or 6

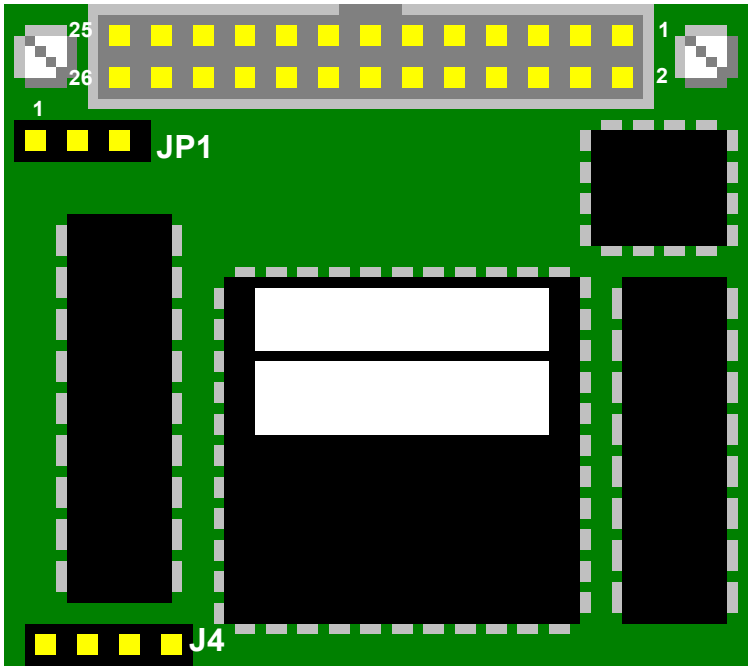
P2 Connector Pin Number	Row A (module #2)	Row B (module #4)	Row C (module #6)
1	Ground	Ground	Ground
2	+5 VDC	+5 VDC	+5 VDC
3	Direction / CW Pulse	Direction / CW Pulse	Direction / CW Pulse
4	Pulse / CCW Pulse	Pulse / CCW Pulse	Pulse / CCW Pulse
5	Reserved	Reserved	Reserved
6	Reserved	Reserved	Reserved
7	Limit Positive	Limit Positive	Limit Positive
8	Limit Negative	Limit Negative	Limit Negative
9	Reserved	Reserved	Reserved
10	Reserved	Reserved	Reserved
11	Reserved	Reserved	Reserved
12	Reserved	Reserved	Reserved
13	Home	Home	Home
14	Full / Half Step	Full / Half Step	Full / Half Step
15	Full / Half Current	Full / Half Current	Full / Half Current
16	Motor On	Motor On	Motor On

DCX-MC260 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-**

DCX-MC260 Module Layout



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

J3 connector pin out

Pin #	*Description*
1	Digital I/O channel #1
2	Digital I/O channel #1
3	Digital I/O channel #1
4	Digital I/O channel #1
5	Digital I/O channel #1
6	Digital I/O channel #1
7	Digital I/O channel #1
8	Digital I/O channel #1
9	Digital I/O channel #1
10	Digital I/O channel #1
11	Digital I/O channel #1
12	Digital I/O channel #1
13	Digital I/O channel #1
14	Digital I/O channel #1
15	Digital I/O channel #1
16	Digital I/O channel #1
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

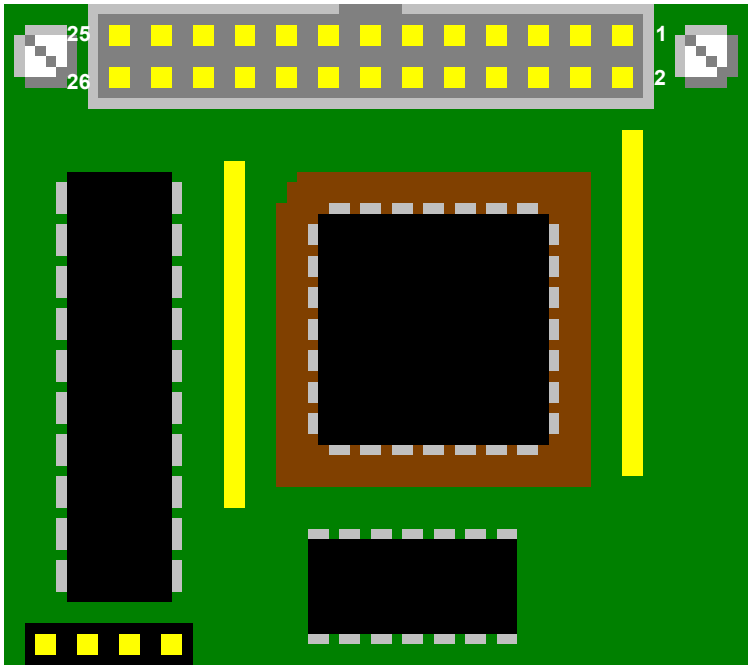
DCX-MC400V P2 connector pin-out – Installed in module position 1, 3, and/or 5

P2 Connector Pin Number	Row A (module #1)	Row B (module #3)	Row C (module #5)
17	Channel 1	Channel 1	Channel 1
18	Channel 2	Channel 2	Channel 2
19	Channel 3	Channel 3	Channel 3
20	Channel 4	Channel 4	Channel 4
21	Channel 5	Channel 5	Channel 5
22	Channel 6	Channel 6	Channel 6
23	Channel 7	Channel 7	Channel 7
24	Channel 8	Channel 8	Channel 8
25	Channel 9	Channel 9	Channel 9
26	Channel 10	Channel 10	Channel 10
27	Channel 11	Channel 11	Channel 11
28	Channel 12	Channel 12	Channel 12
29	Channel 13	Channel 13	Channel 13
30	Channel 14	Channel 14	Channel 14
31	Channel 15	Channel 15	Channel 15
32	Channel 16	Channel 16	Channel 16

DCX-MC400V P2 connector pin-out – Installed in module position 2, 4, and/or 6

P2 Connector Pin Number	Row A (module #2)	Row B (module #4)	Row C (module #6)
1	Channel 1	Channel 1	Channel 1
2	Channel 2	Channel 2	Channel 2
3	Channel 3	Channel 3	Channel 3
4	Channel 4	Channel 4	Channel 4
5	Channel 5	Channel 5	Channel 5
6	Channel 6	Channel 6	Channel 6
7	Channel 7	Channel 7	Channel 7
8	Channel 8	Channel 8	Channel 8
9	Channel 9	Channel 9	Channel 9
10	Channel 10	Channel 10	Channel 10
11	Channel 11	Channel 11	Channel 11
12	Channel 12	Channel 12	Channel 12
13	Channel 13	Channel 13	Channel 13
14	Channel 14	Channel 14	Channel 14
15	Channel 15	Channel 15	Channel 15
16	Channel 16	Channel 16	Channel 16

DCX-MC400 Module layout



DCX-MC500/510/520 Analog I/O Module

DCX-MC500 Electrical Specifications

Parameter	*Min.*	*Max*	*Unit*
Input Resolution	12		Bits
Input Conversion Rate		10	KHz
Input Zero Error			
Using Internal Reference		+/- 3	LSB
Using External Reference		+/- 1/2	LSB
Input Full-Scale Error			
Using Internal Reference		+/- 15	LSB
Using External Reference		+/- 1/2	LSB
Input Zero Temp. Coefficient		0.5	ppm/C
Input Differential Nonlinearity		+/- 1	LSB
Input Total Unadjusted Error			
Using Internal Reference		+/- 15	
Using External Reference		+/- 1	
Input Voltage Range			
Using Internal Reference	0.0	5.0	
Using External Reference	0.0	Vref	
Input Capacitance		8	
Input Leakage Current		100	
External Reference Voltage	4.0	6.0	

Parameter	*Min.*	*Max*	*Unit*
Output Resolution	12		Bits
Output Zero Code Error *			LSB
Output Full Scale Error *			LSB
Output Nonlinearity *			LSB
Output Total Unadjusted Error *			LSB
Output Voltage Range	0.0	5.0	V
	-10.0	+10.0	V

* These values are for 0 to +5.0 volt outputs

J3 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**

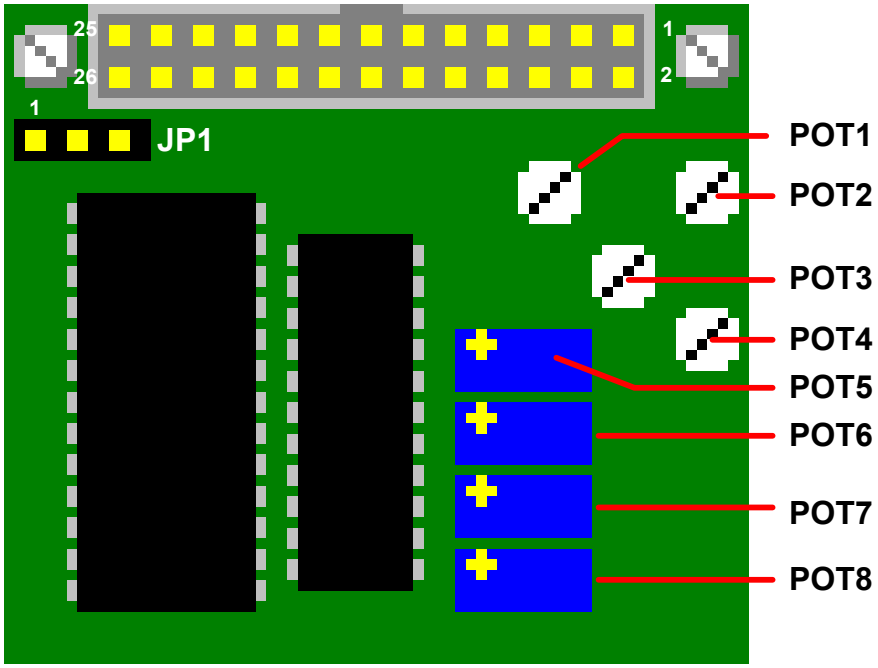
DCX-MC500V P2 connector pin-out – Installed in module position 1, 3, and/or 5

P2 Connector Pin Number	Row A (module #1)	Row B (module #3)	Row C (module #5)
17	Channel 1 In (0-+5)	Channel 1 In (0-+5)	Channel 1 In (0-+5)
18	Channel 1 Out (-10-+10)	Channel 1 Out (-10-+10)	Channel 1 Out (-10-+10)
19	Channel 2 In (0 - +5)	Channel 2 In (0 - +5)	Channel 2 In (0 - +5)
20	Channel 2 Out (-10-+10)	Channel 2 Out (-10-+10)	Channel 2 Out (-10-+10)
21	Channel 3 In (0 - +5)	Channel 3 In (0 - +5)	Channel 3 In (0 - +5)
22	Channel 3 Out (-10-+10)	Channel 3 Out (-10-+10)	Channel 3 Out (-10-+10)
23	Channel 4 In (0 - +5)	Channel 4 In (0 - +5)	Channel 4 In (0 - +5)
24	Channel 4 Out (-10-+10)	Channel 4 Out (-10-+10)	Channel 4 Out (-10-+10)
25	Reserved	Reserved	Reserved
26	Channel 1 Out (0-+5)	Channel 1 Out (0-+5)	Channel 1 Out (0-+5)
27	Reserved	Reserved	Reserved
28	Channel 2 Out (0-+5)	Channel 2 Out (0-+5)	Channel 2 Out (0-+5)
29	Reserved	Reserved	Reserved
30	Channel 3 Out (0-+5)	Channel 3 Out (0-+5)	Channel 3 Out (0-+5)
31	Reserved	Reserved	Reserved
32	Channel 4 Out (0-+5)	Channel 4 Out (0-+5)	Channel 4 Out (0-+5)

DCX-MC500V P2 connector pin-out – Installed in module position 2, 4, and/or 6

P2 Connector Pin Number	Row A (module #2)	Row B (module #4)	Row C (module #6)
1	Channel 1 In (0-+5)	Channel 1 In (0-+5)	Channel 1 In (0-+5)
2	Channel 1 Out (-10-+10)	Channel 1 Out (-10-+10)	Channel 1 Out (-10-+10)
3	Channel 2 In (0 - +5)	Channel 2 In (0 - +5)	Channel 2 In (0 - +5)
4	Channel 2 Out (-10-+10)	Channel 2 Out (-10-+10)	Channel 2 Out (-10-+10)
5	Channel 3 In (0 - +5)	Channel 3 In (0 - +5)	Channel 3 In (0 - +5)
6	Channel 3 Out (-10-+10)	Channel 3 Out (-10-+10)	Channel 3 Out (-10-+10)
7	Channel 4 In (0 - +5)	Channel 4 In (0 - +5)	Channel 4 In (0 - +5)
8	Channel 4 Out (-10-+10)	Channel 4 Out (-10-+10)	Channel 4 Out (-10-+10)
9	Reserved	Reserved	Reserved
10	Channel 1 Out (0-+5)	Channel 1 Out (0-+5)	Channel 1 Out (0-+5)
11	Reserved	Reserved	Reserved
12	Channel 2 Out (0-+5)	Channel 2 Out (0-+5)	Channel 2 Out (0-+5)
13	Reserved	Reserved	Reserved
14	Channel 3 Out (0-+5)	Channel 3 Out (0-+5)	Channel 3 Out (0-+5)
15	Reserved	Reserved	Reserved
16	Channel 4 Out (0-+5)	Channel 4 Out (0-+5)	Channel 4 Out (0-+5)

DCX-MC500 Module layout



DCX-MF300 – RS-232 Interface Module

J3 connector pin out

Pin #	*Description*
1	*
2	*
3	Receive (Maps to DB25 pin 2)
4	*
5	Transmit (Maps to DB25 pin 3)
6	*
7	Clear to Send (Maps to DB25 pin 4)
8	*
9	Request to Send (Maps to DB25 pin 5)
10	*
11	Data Set Ready (Maps to DB25 pin 6)
12	*
13	Ground (Maps to DB25 pin 7)
14	Data Terminal Ready (Maps to DB25 pin 20)
15	Data Carrier Detect (Maps to DB25 pin 8)
16	*
17	*
18	*
19	*
20	*
21	*
22	*
23	*
24	*
25	*
26	*

* No connect
Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N CA-26IDS2-F-SPT or equivalent

DCE / DTE (jumpers JP3 – JP10) configuration

DCE (factory default)	**DTE**
JP3 - 1 to 2	JP3 - 2 to 3
JP4 - 1 to 2	JP4 - 2 to 3
JP6 - 1 to 2	JP5 - 2 to 3
JP7 - 2 to 3	JP8 - 2 to 3
JP9 - 1 to 2	JP10 - 1 to 2
JP10 - 1 to 2	
JP5 - 1 to 2 or	JP6 - 2 to 3 or
JP8 - 1 to 2	JP7 - 2 to 3 to
	JP9 - 2 to 3

(See the layout diagram at end of this appendix)

DCX-MF300 Module Configuration Jumpers - configuration in **bold type** denotes default factory shipping configuration

JP1 – Baud Rate select

Pins	*Description*
1 to 2	300 baud
3 to 4	1200 baud
5 to 6	2400 baud
7 to 8	4800 baud
9 to 10	**9600 baud**
11 to 12	19200 baud
13 to 14	Enable network

JP2 – Handshake / Network Select

Pins	*Description*
1 to 3, 2 to 4	**Hardware handshaking**
1 to 2, 3 to 4	Networking (multiple devices on a serial line)

JP3 – Receive pin select

Pins	*Description*
1 to 2	**Data input on J3 pin 3**
2 to 3	Data input on J3 pin 3

JP4 – Transmit pin select

Pins	*Description*
1 to 2	**Data output on J3 pin 5**
2 to 3	Data input on J3 pin 3

JP5 – Pin 7 select

Pins	*Description*
1 to 2	**DCE**
2 to 3	DTE

JP6 – Pin 9 select

Pins	*Description*
1 to 2	**DCE**
2 to 3	DTE

JP7 – Pin 11 select

Pins	*Description*
1 to 2	DTE
2 to 3	**DCE**

DCX-MF300 Module Configuration Jumpers (cont.) - configuration in **bold type** denotes default factory shipping configuration

JP8 – Pin 14 select

Pins	*Description*
1 to 2	**DCE**
2 to 3	DTE

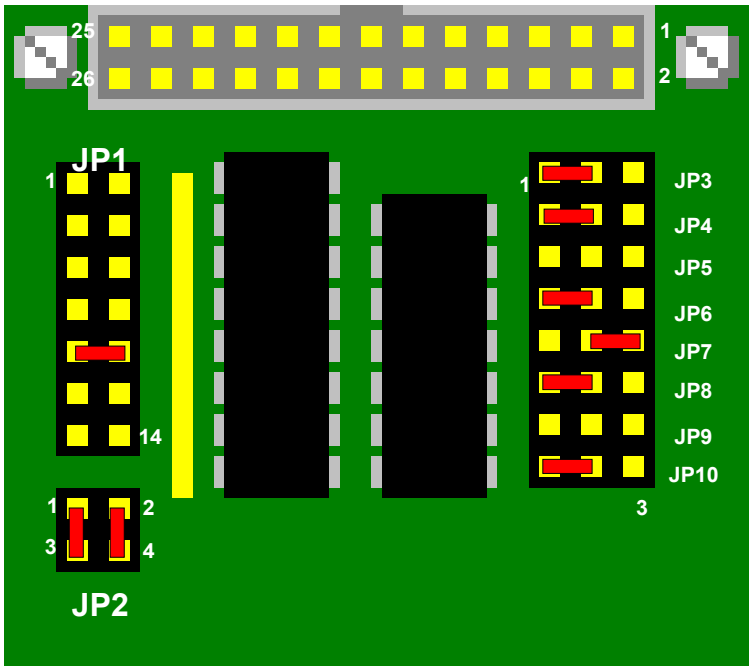
JP9 – Pin 15 select

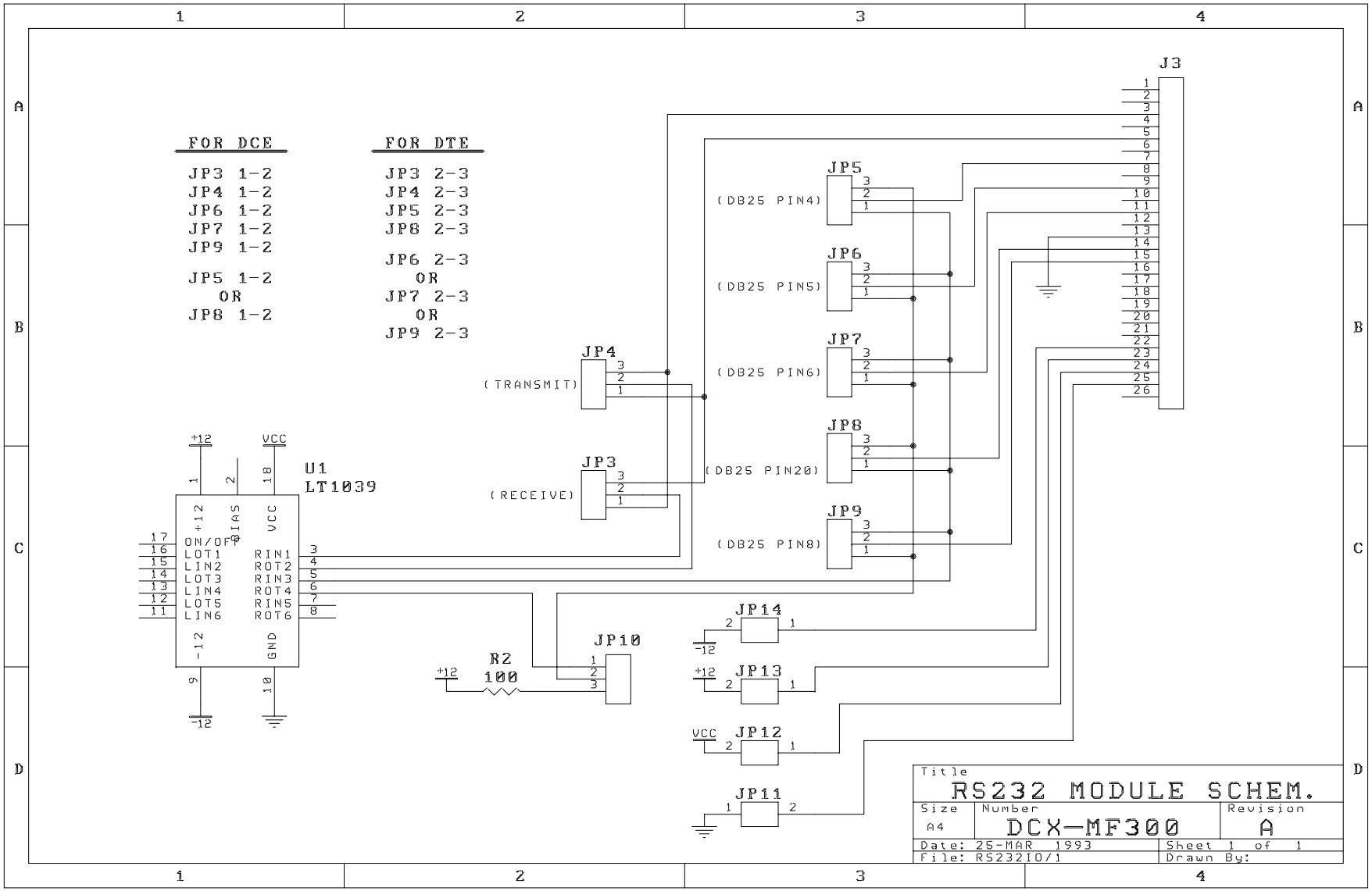
Pins	*Description*
1 to 2	DCE
2 to 3	DTE

JP10 – Ready select

Pins	*Description*
1 to 2	**DCX Ready signal**
2 to 3	100 ohm pull-up to +12V (This signal goes to JP5-3, JP6-1, JP7-1, JP8-3, JP9-1)

DCX-MF300 Module layout




FOR DCE
JP3 1-2
JP4 1-2
JP6 1-2
JP7 1-2
JP9 1-2
JP5 1-2
OR
JP8 1-2
FOR DTE
JP3 2-3
JP4 2-3
JP5 2-3
JP8 2-3
JP6 2-3
OR
JP7 2-3
OR
JP9 2-3
J3
JP4
(TRANSMIT)
JP3
(RECEIVE)
JP5
(DB25 PIN4)
JP6
(DB25 PIN5)
JP7
(DB25 PIN6)
JP8
(DB25 PIN20)
JP9
(DB25 PIN8)
U1
LT1039
+12
VCC
-12
GND
BIAS
ON/OFF
LOT1
LIN2
LOT3
LIN4
LOT5
LIN6
RIN1
ROT2
RIN3
ROT4
RIN5
ROT6
R2
100
JP10
JP14
JP13
JP12
JP11
Title
RS232 MODULE SCHEM.
Size
A4
Number
DCX-MF300
Revision
A
Date: 25-MAR 1993
Sheet 1 of 1
File: RS232IO/1
Drawn By:

DCX-MF310 IEEE-488 Interface Module

J3 connector pin out - The signals are arranged so that the connection to a standard IEEE-488 connector will be straight through a ribbon cable.

Pin #	*Description*
1	IEEE-488: D101
2	IEEE-488: D105
3	IEEE-488: D102
4	IEEE-488: D106
5	IEEE-488: D103
6	IEEE-488: D107
7	IEEE-488: D104
8	IEEE-488: D108
9	IEEE-488: E01
10	IEEE-488: REN
11	IEEE-488: DAV
12	IEEE-488: Ground
13	IEEE-488: NRFD
14	IEEE-488: Ground
15	IEEE-488: NDAC
16	IEEE-488: Ground
17	IEEE-488: IFC
18	IEEE-488: Ground
19	IEEE-488: SRQ
20	IEEE-488: Ground
21	IEEE-488: ATN
22	IEEE-488: Ground
23	IEEE-488: Shield
24	IEEE-488: GND
25	IEEE-488: Not used
26	IEEE-488: Not used

Mating Connector:26-pin dual-row IDC female, Circuit Assembly P/N CA-26IDS2-F-SPT or equivalent

DCX-MF310 Module Configuration Jumpers - configuration in **bold type** denotes default factory shipping configuration

JP1 – Baud Rate select

Pins	***Description***
1 to 2	Open collector drivers
open	**Push –Pull drivers**

Note: Push-pull used for high speed bus operation

JP2 – Select cable shield to ground

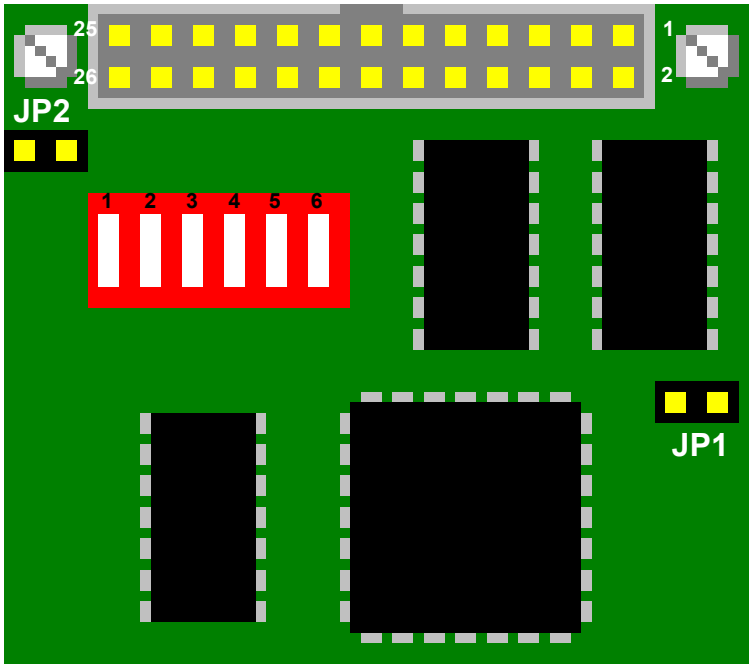
Pins	***Description***
1 to 2	Tie cable shield to DCX ground
open	

Note: Cable shield should only be grounded at one point

IEEE-488 Bus Address Selection

SW6	*SW5*	*SW4*	*SW3*	*SW2*	*SW1*		*Listen*	*Talk*
Off	Off	Off	Off	Off	Off		" " (20h)	" " (40h)
Off	Off	Off	Off	Off	On		"!" (21h)	"A" (41h)
Off	Off	Off	Off	On	Off		""" (22h)	"B" (42h)
Off	Off	Off	Off	On	On		"#" (23h)	"C" (43h)
Off	Off	Off	On	Off	Off		"$" (24h)	"D" (44h)
Off	Off	Off	On	Off	On		"%" (25h)	"E" (45h)
Off	Off	Off	On	On	Off		"&" (26h)	"F" (46h)
Off	Off	Off	On	On	On		"'" (27h)	"G" (47h)
Off	Off	On	Off	Off	Off		"(" (28h)	"H" (48h)
Off	Off	On	Off	Off	On		")" (29h)	"I" (49h)
Off	Off	On	Off	On	Off		"*" (2Ah)	"J" (4Ah)
Off	Off	On	Off	On	On		"+" (2Bh)	"K" (4Bh)
Off	Off	On	On	Off	Off		"," (2Ch)	"L" (4Ch)
Off	Off	On	On	Off	On		"-" (2Dh)	"M" (4Dh)
Off	Off	On	On	On	Off		"." (2Eh)	"N" (4Eh)
Off	Off	On	On	On	On		"/" (2Fh)	"O" (4Fh)
Off	On	Off	Off	Off	Off		"0" (30h)	"P" (50h)
Off	On	Off	Off	Off	On		"1" (31h)	"Q" (51h)
Off	On	Off	Off	On	Off		"2" (32h)	"R" (52h)
Off	On	Off	Off	On	On		"3" (33h)	"S" (53h)
Off	On	Off	On	Off	Off		"4" (34h)	"T" (54h)
Off	On	Off	On	Off	On		"5" (35h)	"U" (55h)
Off	On	Off	On	On	Off		"6" (36h)	"V" (56h)
Off	On	Off	On	On	On		"7" (37h)	"W" (57h)
Off	On	On	Off	Off	Off		"8" (38h)	"X" (58h)
Off	On	On	Off	Off	On		"9" (39h)	"Y" (59h)
Off	On	On	Off	On	Off		":" (3Ah)	"Z" (5Ah)
Off	On	On	Off	On	On		"; " (3Bh)	"[" (5Bh)
Off	On	On	On	Off	Off		"<" (3Ch)	"\" (5Ch)
Off	On	On	On	Off	On		"=" (3Dh)	"]" (5Dh)
Off	On	On	On	On	Off		">" (3Eh)	" " (5Eh)
Off	On	On	On	On	On		"?" (3Fh)	" " (5Fh)

DCX-MF310 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

DCX-BF022 Configuration Jumpers - configuration in **bold type** denotes default factory shipping configuration

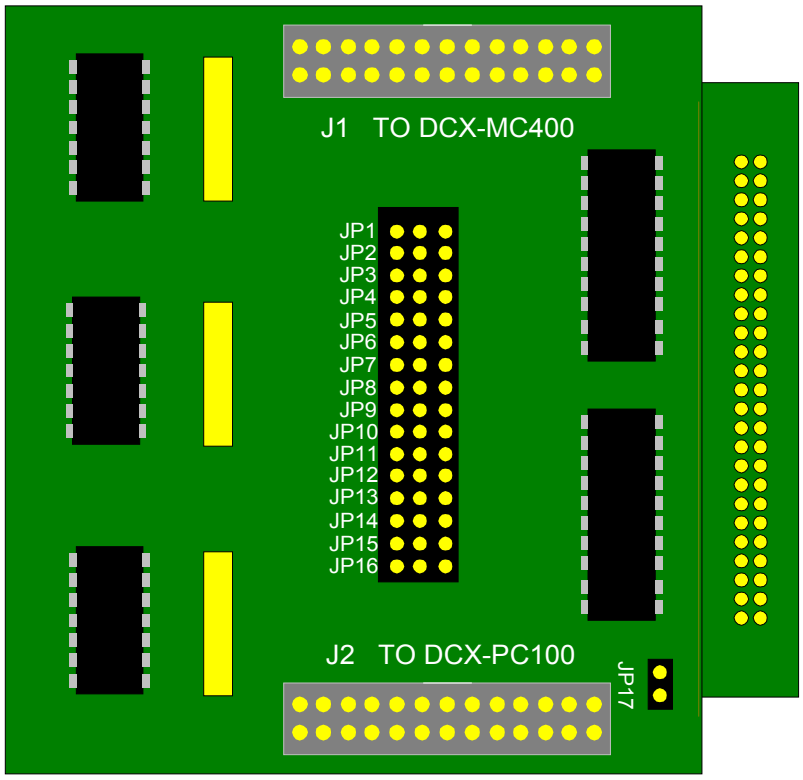
JP1 – JP16 Configure Digital channel as Input or Output

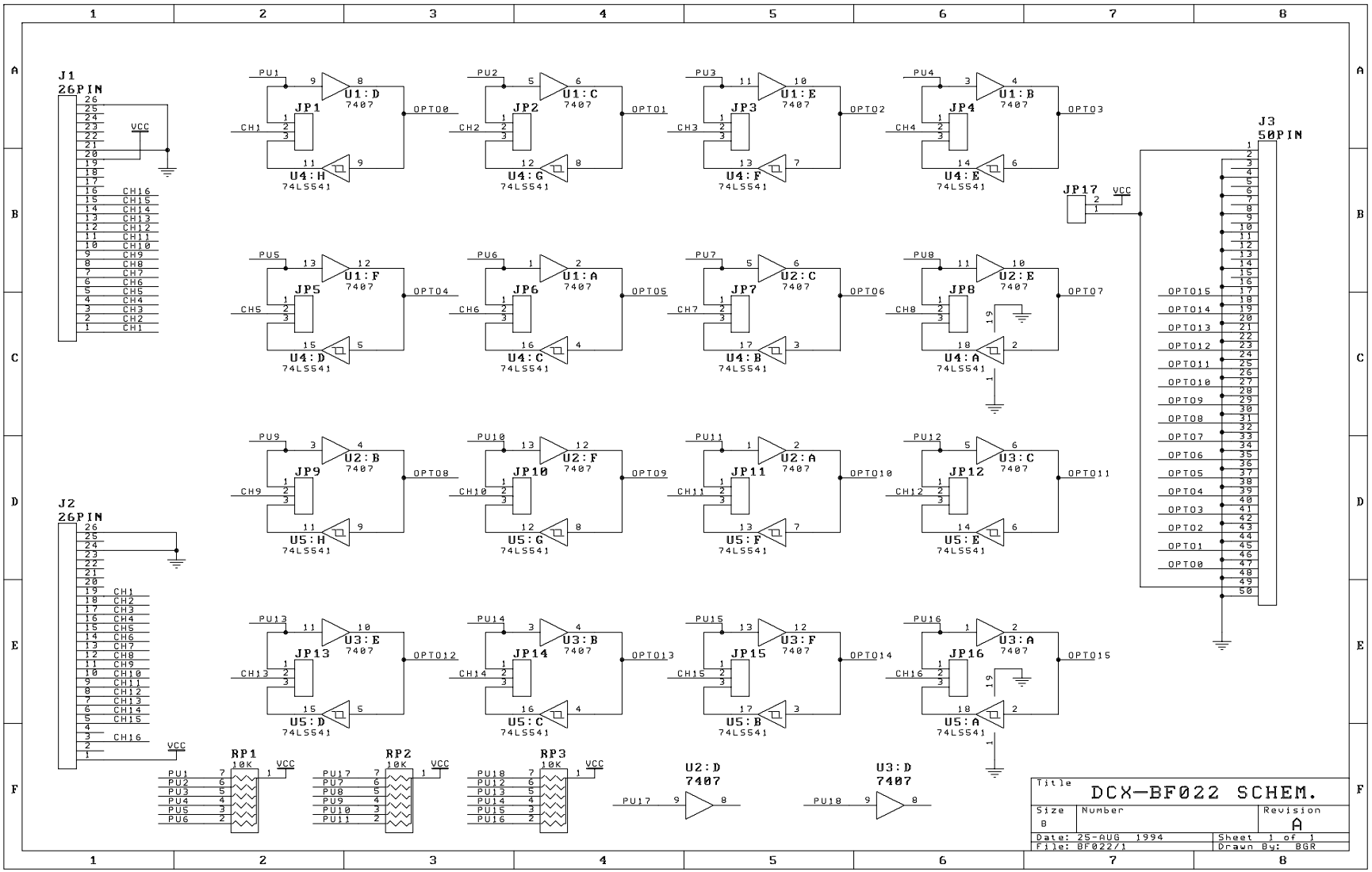
Pins	*Description*
1 to 2	**Configure channel as Output**
2 to 3	Configure channel as an Input

JP17 – Select Relay Rack supply source

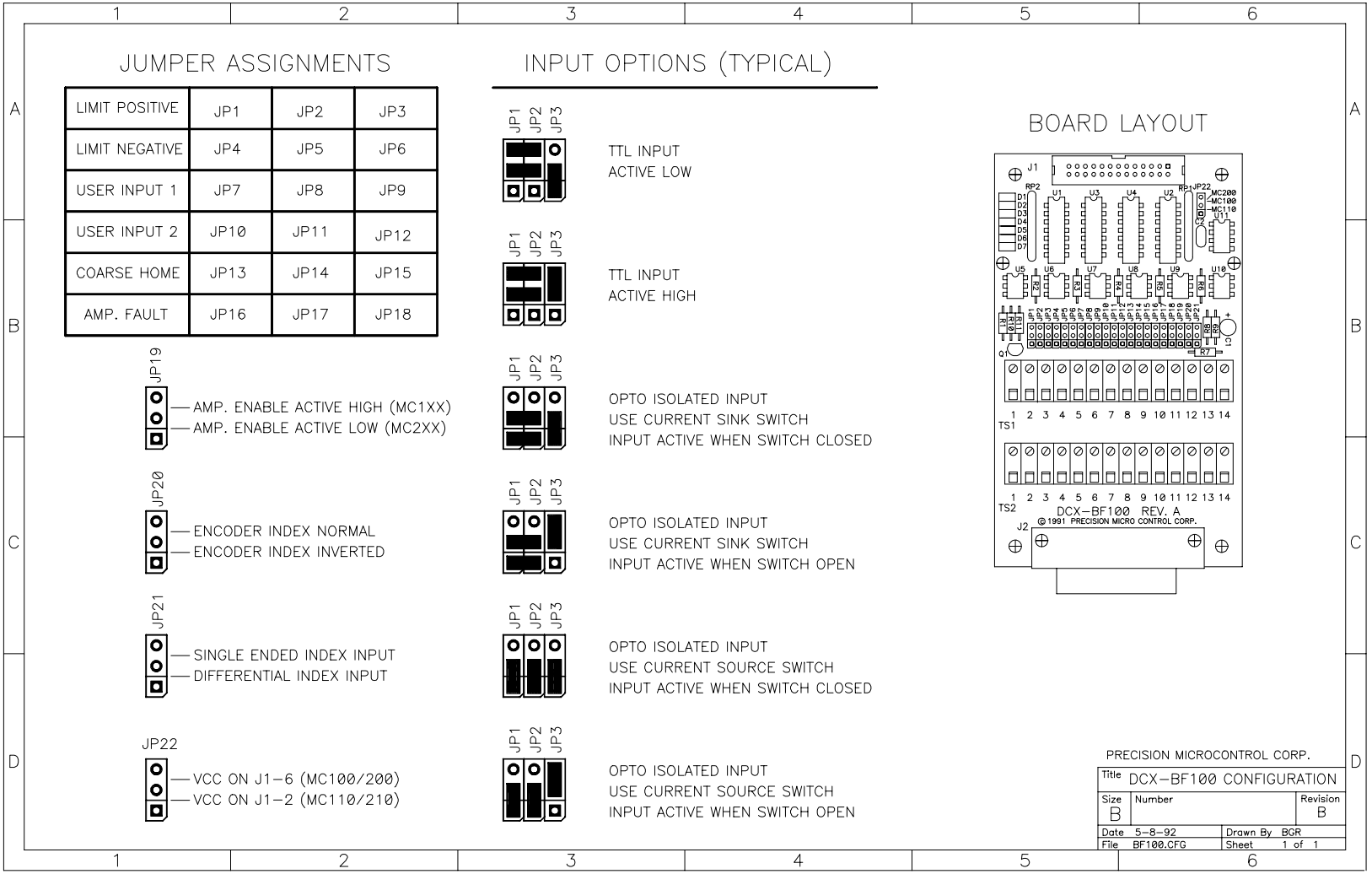
Pins	*Description*
1 to 2	**DCX provides +5 VDC Relay Rack supply**
2 to 3	Relay Rack has separate +5 VDC supply

DCX-BF022 Interface layout




J1
26PIN
J2
26PIN
J3
50PIN
JP1
JP2
JP3
JP4
JP5
JP6
JP7
JP8
JP9
JP10
JP11
JP12
JP13
JP14
JP15
JP16
JP17
U1:D
U1:C
U1:E
U1:B
U1:F
U1:A
U2:C
U2:E
U2:B
U2:F
U2:A
U3:C
U3:E
U3:B
U3:F
U3:A
U4:H
U4:G
U4:F
U4:E
U4:D
U4:C
U4:B
U4:A
U5:H
U5:G
U5:F
U5:E
U5:D
U5:C
U5:B
U5:A
74LS541
7407
RP1
RP2
RP3
10K
VCC
OPTO0
OPTO1
OPTO2
OPTO3
OPTO4
OPTO5
OPTO6
OPTO7
OPTO8
OPTO9
OPTO10
OPTO11
OPTO12
OPTO13
OPTO14
OPTO15
Title
DCX-BF022 SCHEM.
Size
B
Number
Revision
A
Date: 25-AUG 1994
Sheet 1 of 1
File: BF022/1
Drawn By: BGR

DCX-BF100 Servo Module Interconnect Board



DCX-BF100 to DCX-MC200 Connections:

Connector J1

Pin	Description
1	Analog Ground
2	Analog Command output
3	+12 VDC
4	-12 VDC
5	Ground
6	+5 VDC
7	No connect
8	No connect
9	Coarse Home
10	Amplifier Fault
11	Amplifier Enable
12	User Input #1
13	User Input #2
14	Limit +
15	Limit -
16	Prim. Encoder Phase A+
17	Encoder Power
18	Aux. Encoder Index-
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

Connector J2

Pin	Description
1	Analog Ground
2	+12 VDC
3	Ground
4	Opto Supply
5	Coarse Home
6	Amplifier Enable
7	User Input #2
8	Limit -
9	Encoder Power
10	Prim. Encoder Phase A-
11	Aux. Encoder Phase A
12	Prim. Encoder Phase B+
13	Prim. Encoder Index-
14	Analog Command output
15	-12 VDC
16	+5 VDC
17	Prim. Encoder Index+
18	Amplifier Fault
19	User Input #1
20	Limit +
21	Prim. Encoder Phase A+
22	Aux. Encoder Index-
23	Prim. Encoder Phase B-
24	Aux. Encoder Phase B
25	Aux. Encoder Index+

Note: Set MC200 module for single ended encoder index input (ie. connect jumper JP2 pins 2 and 3)

Terminal strip TS1

Pin	Description
1	Shield
2	Analog Ground
3	Analog Command output
4	+5 VDC
5	+5 VDC
6	Amplifier Enable
7	Coarse Home
8	Amplifier Fault
9	User Input #1
10	User Input #2
11	Limit +
12	Limit -
13	Opto Supply
14	Ground

Terminal strip TS2

Pin	Description
1	Shield
2	Encoder Power
3	Prim. Encoder Phase A+
4	Prim. Encoder Phase A-
5	Prim. Encoder Phase B+
6	Prim. Encoder Phase B-
7	Prim. Encoder Index+
8	Prim. Encoder Index-
9	Aux. Encoder Phase A
10	Aux. Encoder Phase A
11	Aux. Encoder Index+
12	Aux. Encoder Index-
13	+5 VDC
14	Ground

DCX-BF100 to DCX-MC210 Connections:

Connector J1

Pin	Description
1	PWM Motor Drive +
2	Encoder Power
3	Prim. Encoder Phase B+
4	Prim. Encoder Phase A+
5	Ground
6	PWM Motor Drive -
7	Ext. Motor Power +
8	Prim. Enc. Index+ / Ext -
9	Coarse Home
10	Amplifier Fault
11	Amplifier Enable
12	Reserved
13	Reserved
14	Limit +
15	Limit -
16	Prim. Encoder Phase A+
17	Encoder Power
18	Aux. Encoder Index-
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

Connector J2

Pin	Description
1	PWM Motor Drive +
2	Prim. Encoder Phase B+
3	Ground
4	Opto Supply
5	Coarse Home
6	Amplifier Enable
7	No connect
8	Limit -
9	Encoder Power
10	Prim. Encoder Phase A-
11	Aux. Encoder Phase A
12	Prim. Encoder Phase B+
13	Prim. Encoder Index-
14	Encoder Power
15	Prim. Encoder Phase A+
16	PWM Motor Drive +
17	Prim. Encoder Index+
18	Amplifier Fault
19	No connect
20	Limit +
21	Prim. Encoder Phase A+
22	Aux. Encoder Index-
23	Prim. Encoder Phase B-
24	Aux. Encoder Phase B
25	Aux. Encoder Index+

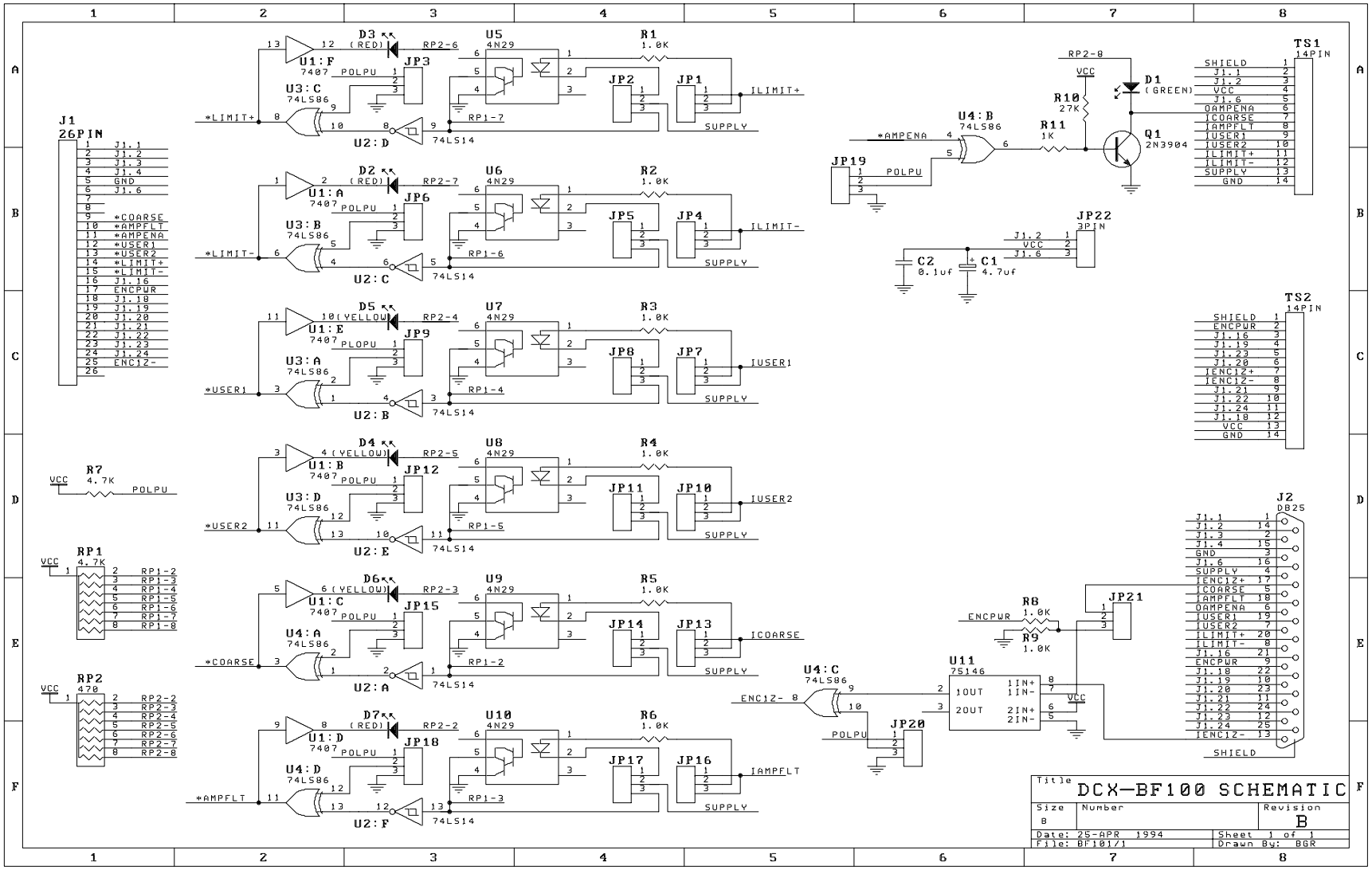
Note: Set MC210 module for single ended encoder index input (ie. connect jumper JP2 pins 2 and 3)

Terminal strip TS1

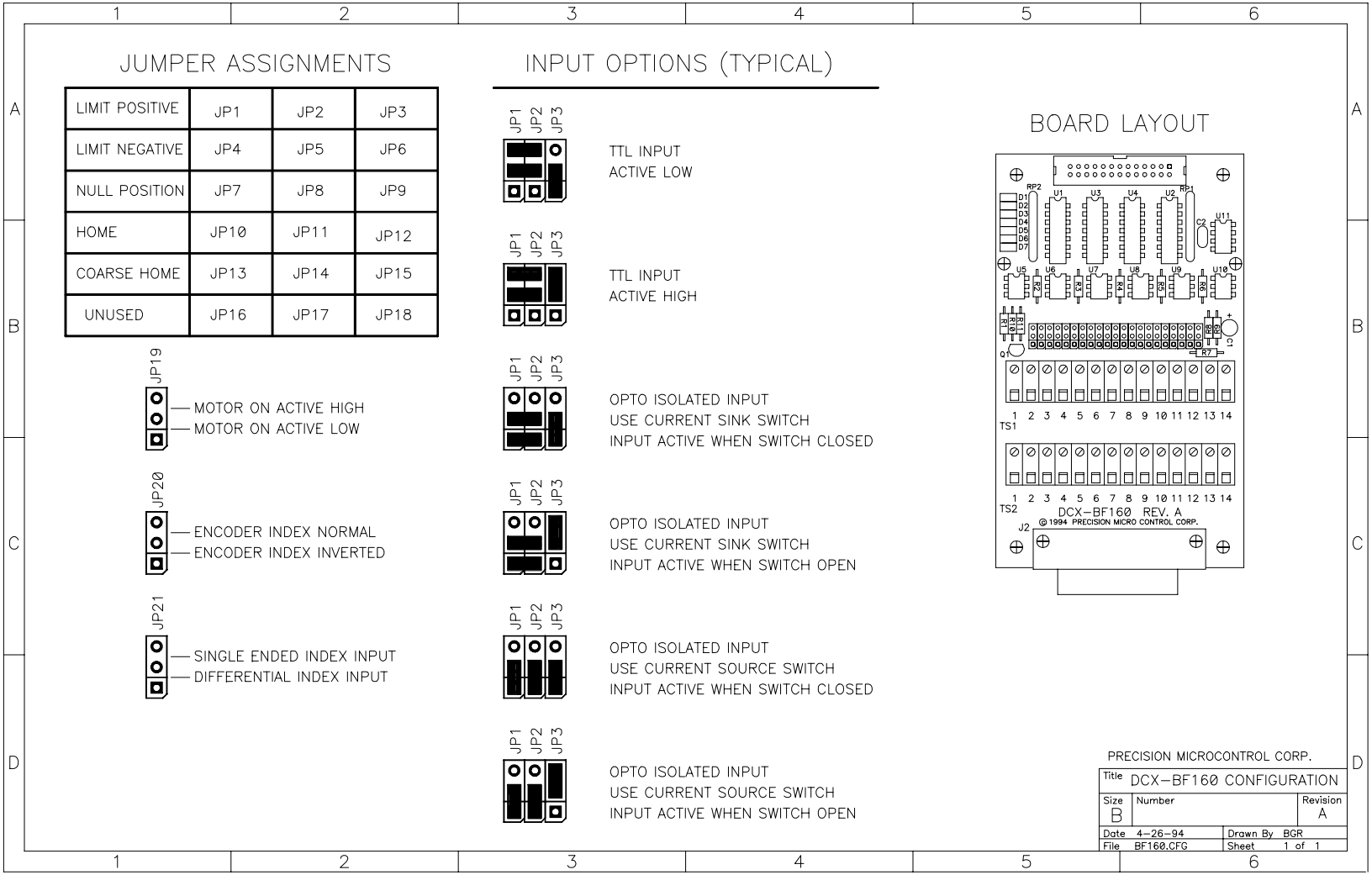
Pin	Description
1	Shield
2	PWM Motor Drive +
3	Encoder Power
4	+5 VDC
5	PWM Motor Drive -
6	Amplifier Enable
7	Coarse Home
8	Amplifier Fault
9	No connect
10	No connect
11	Limit +
12	Limit -
13	Opto Supply
14	Ground

Terminal strip TS2

Pin	Description
1	Shield
2	Encoder Power
3	Prim. Encoder Phase A+
4	Prim. Encoder Phase A-
5	Prim. Encoder Phase B+
6	Prim. Encoder Phase B-
7	Prim. Encoder Index+
8	Prim. Encoder Index-
9	Aux. Encoder Phase A
10	Aux. Encoder Phase A
11	Aux. Encoder Index+
12	Aux. Encoder Index-
13	+5 VDC
14	Ground


J1
26PIN
*LIMIT+
*LIMIT-
*USER1
*USER2
*COARSE
*AMPFLT
*AMPENA
ENCPWR
ENC1Z-
POLPU
VCC
U1:F
U1:A
U1:E
U1:B
U1:C
U1:D
U2:D
U2:C
U2:B
U2:E
U2:A
U2:F
U3:C
U3:B
U3:A
U3:D
U4:A
U4:B
U4:C
U4:D
74LS86
74LS14
7407
4N29
U5
U6
U7
U8
U9
U10
U11
75146
D1 (GREEN)
D2 (RED)
D3 (RED)
D4 (YELLOW)
D5 (YELLOW)
D6 (YELLOW)
D7 (RED)
Q1
2N3904
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
1.0K
4.7K
27K
1K
RP1
RP2
JP1
JP2
JP3
JP4
JP5
JP6
JP7
JP8
JP9
JP10
JP11
JP12
JP13
JP14
JP15
JP16
JP17
JP18
JP19
JP20
JP21
JP22
3PIN
C1
4.7uf
C2
0.1uf
TS1
TS2
14PIN
J2
DB25
SHIELD
Title
DCX-BF100 SCHEMATIC
Size
B
Number
Revision
B
Date: 25-APR 1994
Sheet 1 of 1
File: BF101/1
Drawn By: BGR

DCX-BF160 Stepper Module Interconnect Board



DCX-BF100 to DCX-MC260 Connections:

Connector J1

Pin	Description
1	Ground
2	+5 VDC
3	Direction / CW Pulse
4	Pulse / CCW Pulse
5	Reserved
6	Reserved
7	Stopped
8	Limit +
9	Limit -
10	Jog
11	No connect
12	No connect
13	Home
14	Full / Half Step
15	Full / Half Current
16	Motor On
17	Null Position
18	Aux. Encoder Phase A+
19	Aux. Encoder Phase A-
20	Aux. Encoder Phase B+
21	Aux. Encoder Phase B-
22	Aux. Encoder Index
23	Aux. Enc. Coarse Home
24	+12 VDC
25	-12 VDC
26	Ground

Connector J2

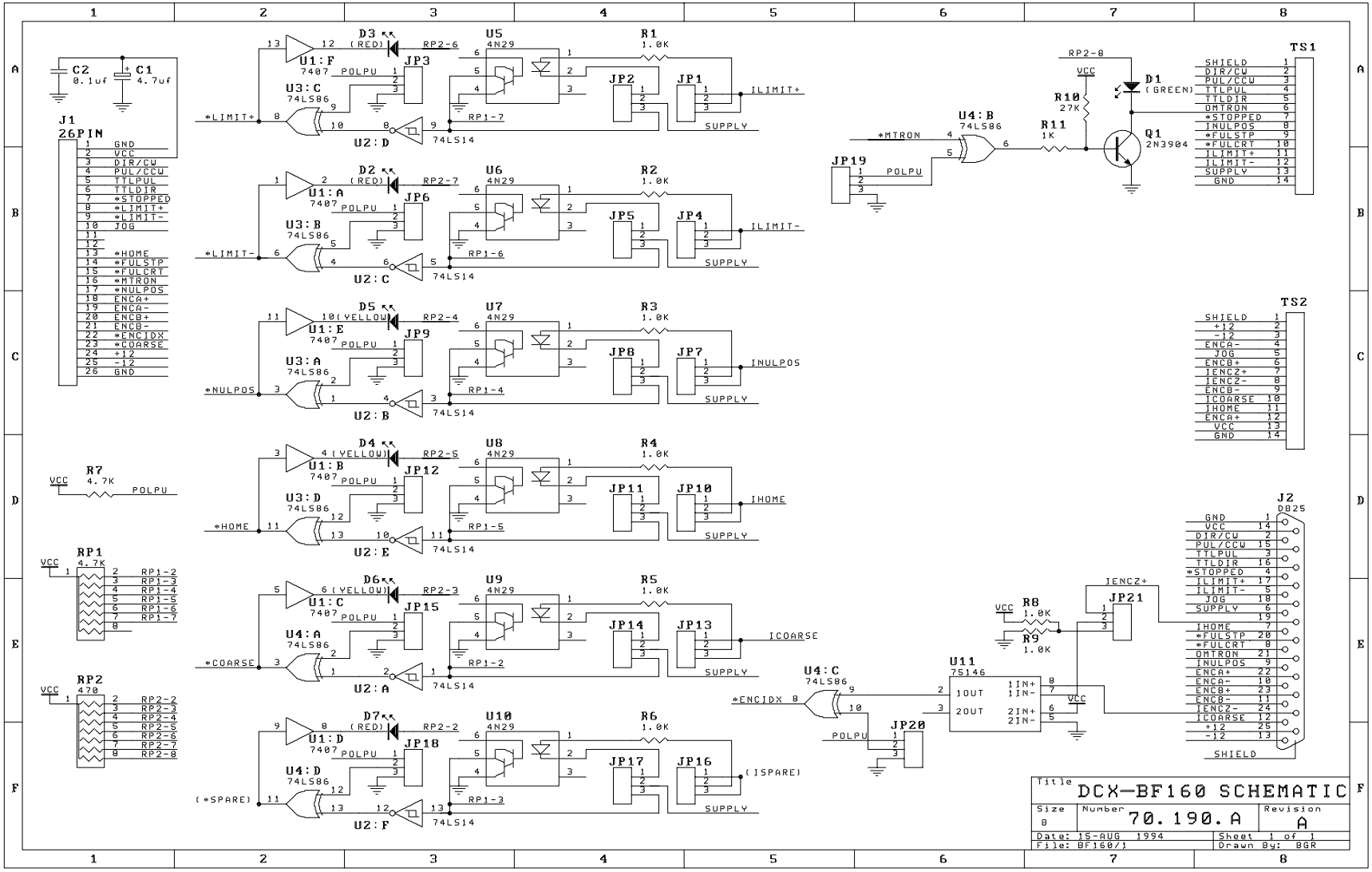
Pin	Description
1	Ground
2	Direction / CW Pulse
3	Reserved
4	Stopped
5	Limit -
6	Opto Supply
7	Home
8	Full / Half Current
9	Null Position
10	Aux. Encoder Phase A-
11	Aux. Encoder Phase B-
12	Aux. Enc. Coarse Home
13	-12 VDC
14	+5 VDC
15	Pulse / CCW Pulse
16	Reserved
17	Limit +
18	Jog
19	Aux. Encoder Index+
20	Full / Half Step
21	Motor On
22	Aux. Encoder Phase A+
23	Aux. Encoder Phase B+
24	Aux. Encoder Index-
25	+12 VDC

Terminal strip TS1

Pin	Description
1	Shield
2	Direction / CW Pulse
3	Pulse / CCW Pulse
4	Reserved
5	Reserved
6	Motor On
7	Stopped
8	Null Position
9	Full / Half Step
10	Full / Half current
11	Limit +
12	Limit -
13	Opto Supply
14	Ground

Terminal strip TS2

Pin	Description
1	Shield
2	+12 VDC
3	-12 VDC
4	Aux. Encoder Phase A-
5	Jog
6	Aux. Encoder Phase B+
7	Aux. Encoder Index+
8	Aux. Encoder Index-
9	Aux. Encoder Phase B-
10	Aux. Enc. Coarse Home
11	Home
12	Aux. Encoder Phase A+
13	+5 VDC
14	Ground


C2 0.1uf
C1 4.7uf
J1 26PIN
1 GND
2 VCC
3 DIR/CW
4 PUL/CCW
5 TTLPUL
6 TTLDIR
7 *STOPPED
8 *LIMIT+
9 *LIMIT-
10 JOG
13 *HOME
14 *FULSTP
15 *FULCRT
16 *MTRON
17 *NULPOS
18 ENCA+
19 ENCA-
20 ENCB+
21 ENCB-
22 *ENCIDX
23 *COARSE
24 +12
25 -12
26 GND
D3 (RED) RP2-6
D2 (RED) RP2-7
D5 10(YELLOW) RP2-4
D4 4(YELLOW) RP2-5
D6 6(YELLOW) RP2-3
D7 (RED) RP2-2
U1:F 7407
U1:A 7407
U1:E 7407
U1:B 7407
U1:C 7407
U1:D 7407
U3:C 74LS86
U3:B 74LS86
U3:A 74LS86
U3:D 74LS86
U4:A 74LS86
U4:D 74LS86
U2:D 74LS14
U2:C 74LS14
U2:B 74LS14
U2:E 74LS14
U2:A 74LS14
U2:F 74LS14
U5 4N29
U6 4N29
U7 4N29
U8 4N29
U9 4N29
U10 4N29
JP3 JP6 JP9 JP12 JP15 JP18
JP2 JP5 JP8 JP11 JP14 JP17
JP1 JP4 JP7 JP10 JP13 JP16
R1 1.0K
R2 1.0K
R3 1.0K
R4 1.0K
R5 1.0K
R6 1.0K
RP1-7
RP1-6
RP1-4
RP1-5
RP1-2
RP1-3
*LIMIT+
*LIMIT-
*NULPOS
*HOME
*COARSE
(*SPARE)
ILIMIT+
ILIMIT-
INULPOS
IHOME
ICOARSE
(ISPARE)
SUPPLY
POLPU
R7 4.7K
RP1 4.7K
RP2 470
U4:B 74LS86
*MTRON
JP19
R10 27K
R11 1K
D1 (GREEN)
Q1 2N3904
TS1
SHIELD 1
DIR/CW 2
PUL/CCW 3
TTLPUL 4
TTLDIR 5
OMTRON 6
*STOPPED 7
INULPOS 8
*FULSTP 9
*FULCRT 10
ILIMIT+ 11
ILIMIT- 12
SUPPLY 13
GND 14
TS2
SHIELD 1
+12 2
-12 3
ENCA- 4
JOG 5
ENCB+ 6
IENCZ+ 7
IENCZ- 8
ENCB- 9
ICOARSE 10
IHOME 11
ENCA+ 12
VCC 13
GND 14
J2 DB25
GND 1
VCC 14
DIR/CW 2
PUL/CCW 15
TTLPUL 3
TTLDIR 16
*STOPPED 4
ILIMIT+ 17
ILIMIT- 5
JOG 18
SUPPLY 6
19
IHOME 7
*FULSTP 20
*FULCRT 8
OMTRON 21
INULPOS 9
ENCA+ 22
ENCA- 16
ENCB+ 23
ENCB- 11
IENCZ- 24
ICOARSE 12
+12 25
-12 13
SHIELD
U4:C 74LS86
*ENCIDX
JP20
U11 75146
1OUT 2OUT
1IN+ 1IN- 2IN+ 2IN-
R8 1.0K
R9 1.0K
IENCZ+
JP21
Title DCX-BF160 SCHEMATIC
Size B
Number 70.190.A
Revision A
Date: 15-AUG 1994
File: BF160/1
Sheet 1 of 1
Drawn By: BGR

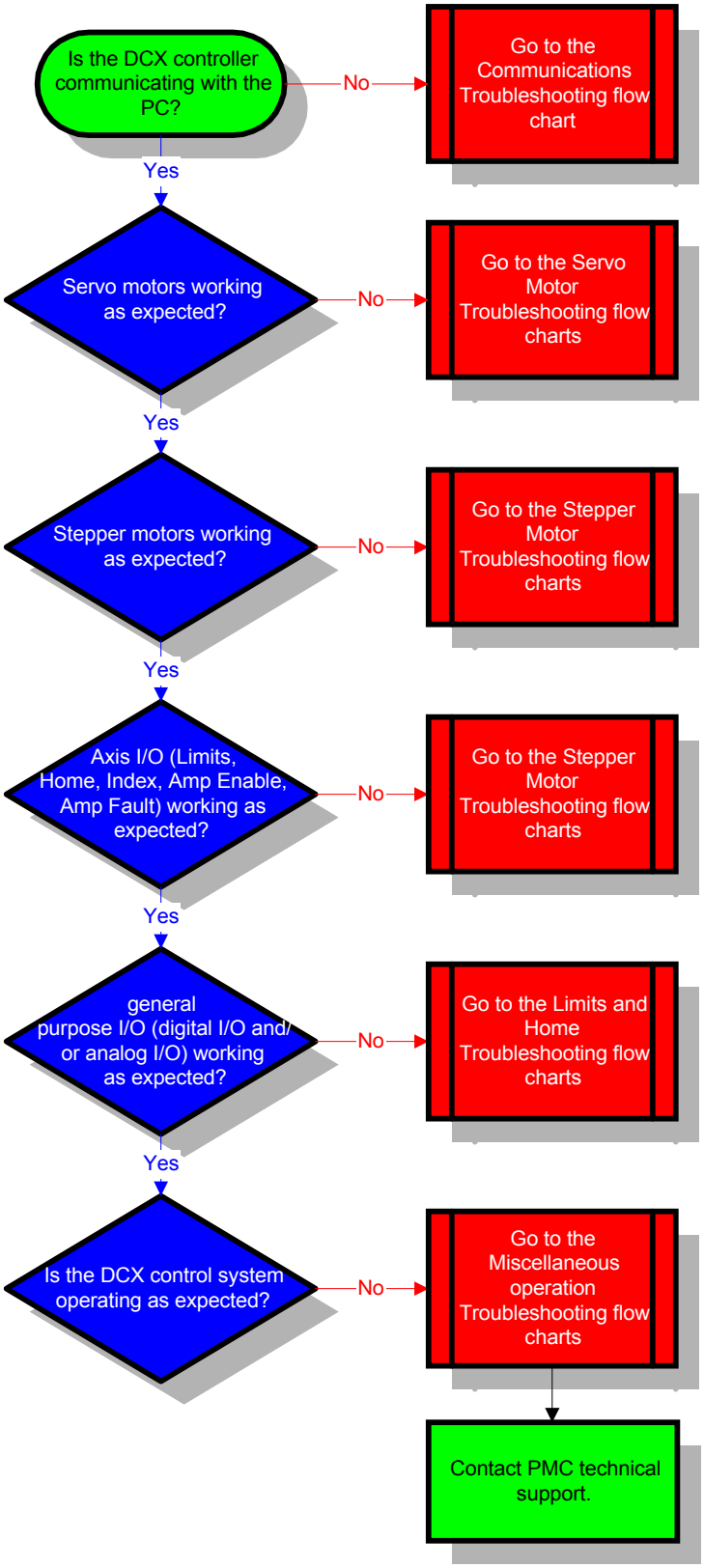
Chapter Contents

Chapter

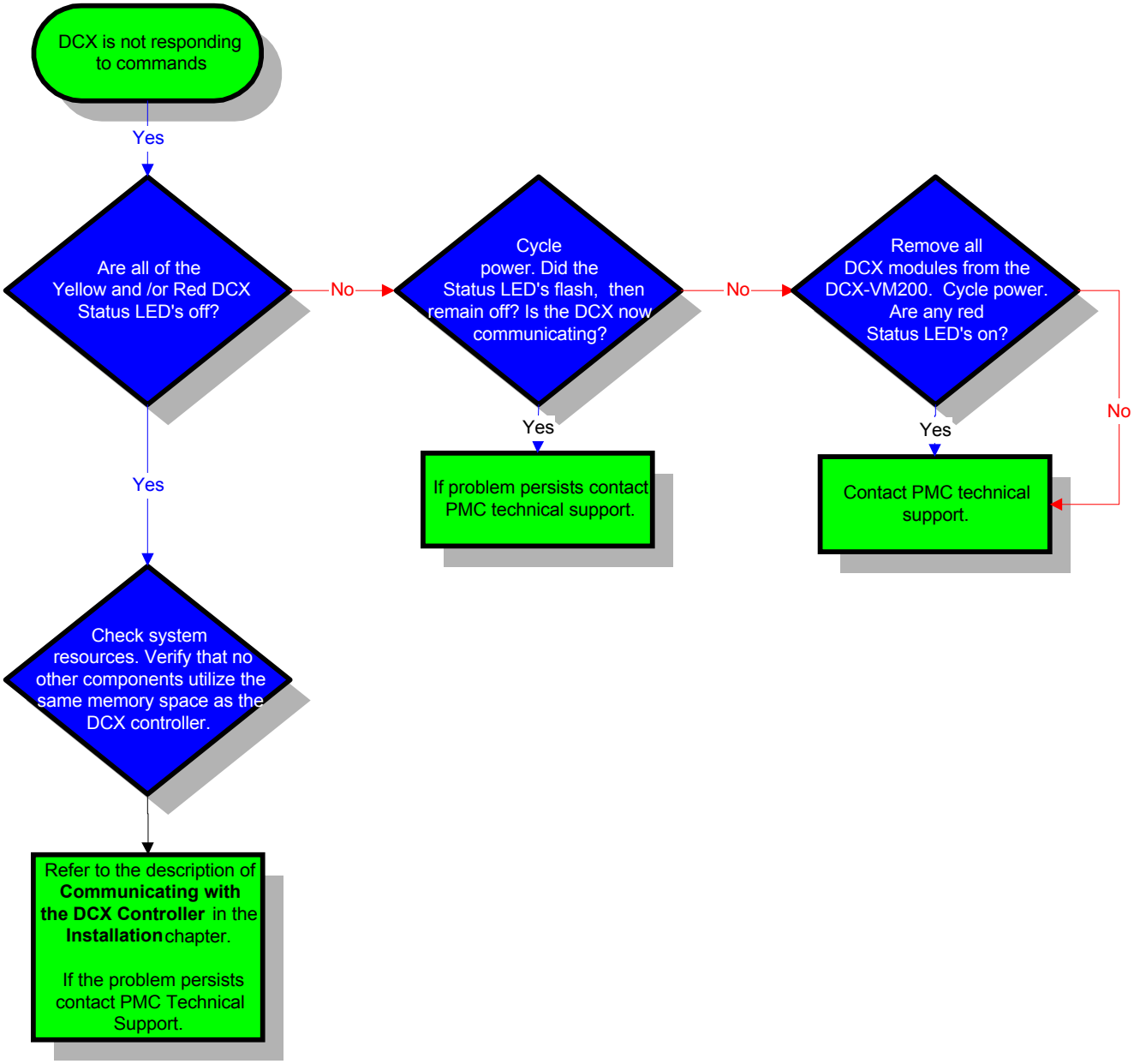
12

Troubleshooting

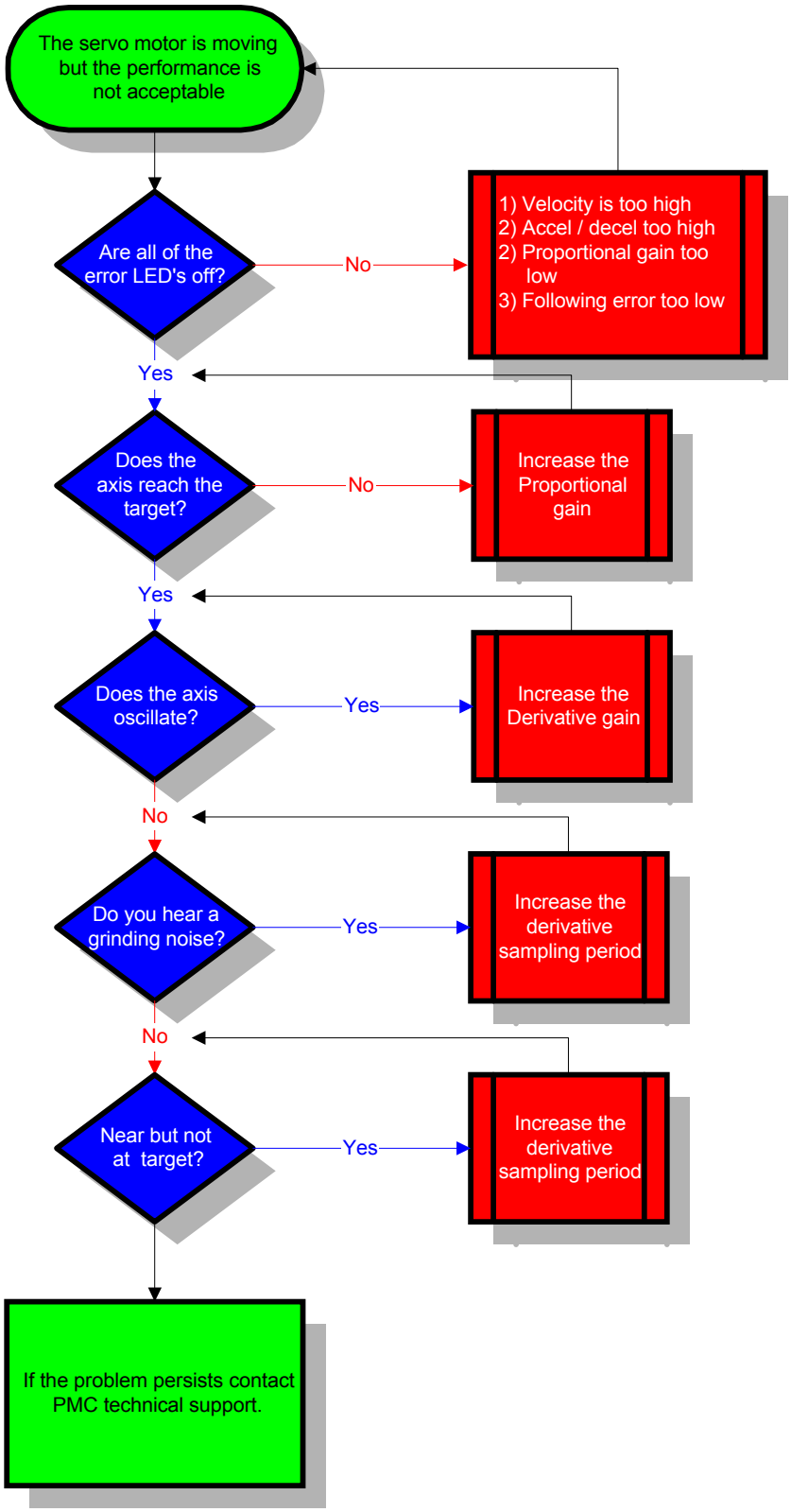
DCX System Troubleshooting



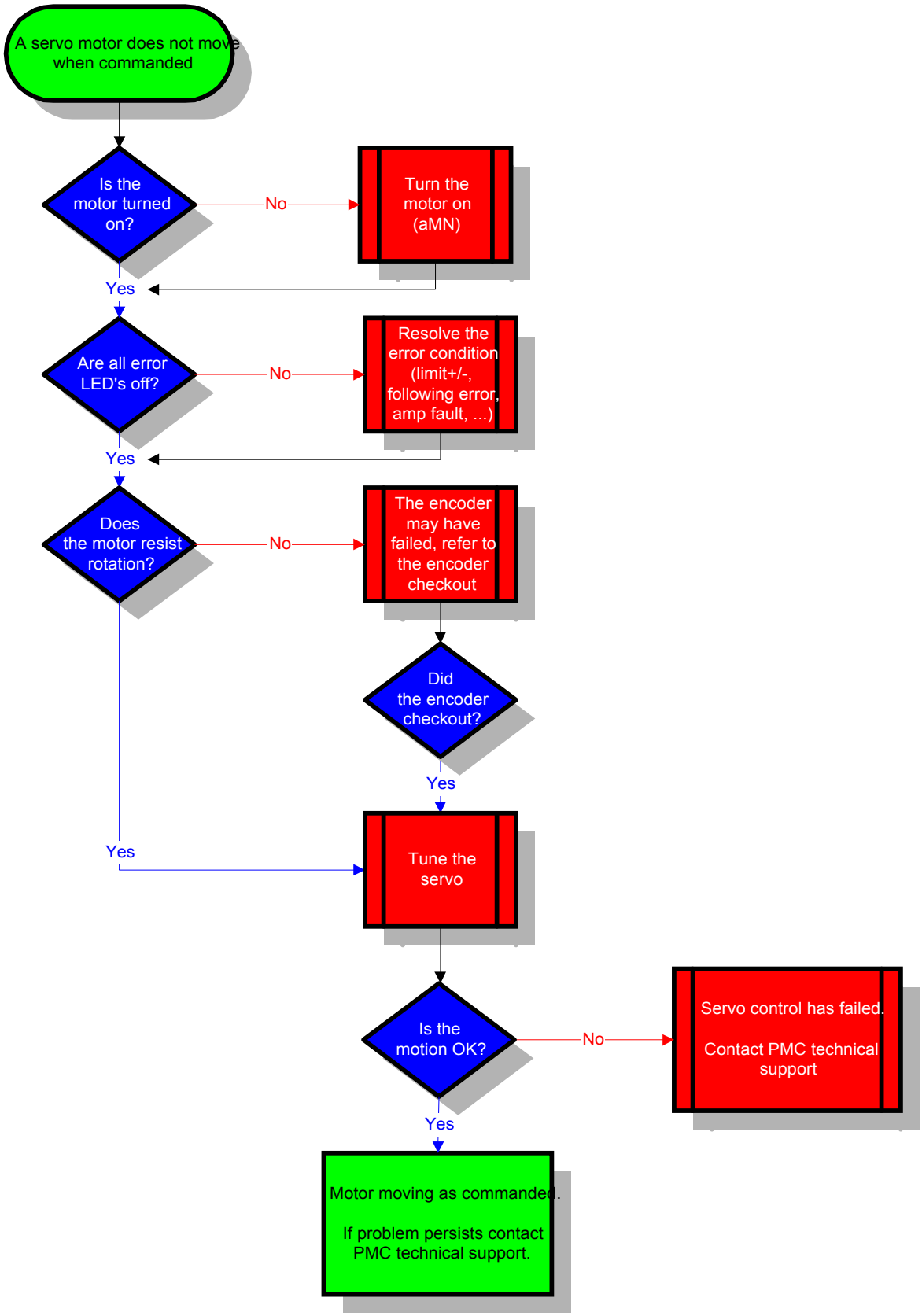
Communications Troubleshooting



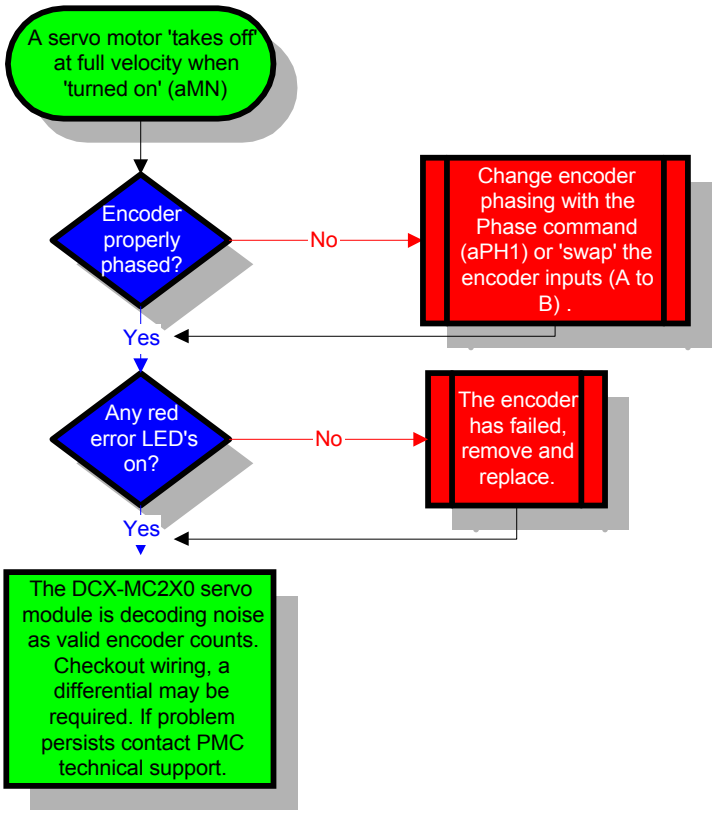
Troubleshooting - Tuning a Servo Motor



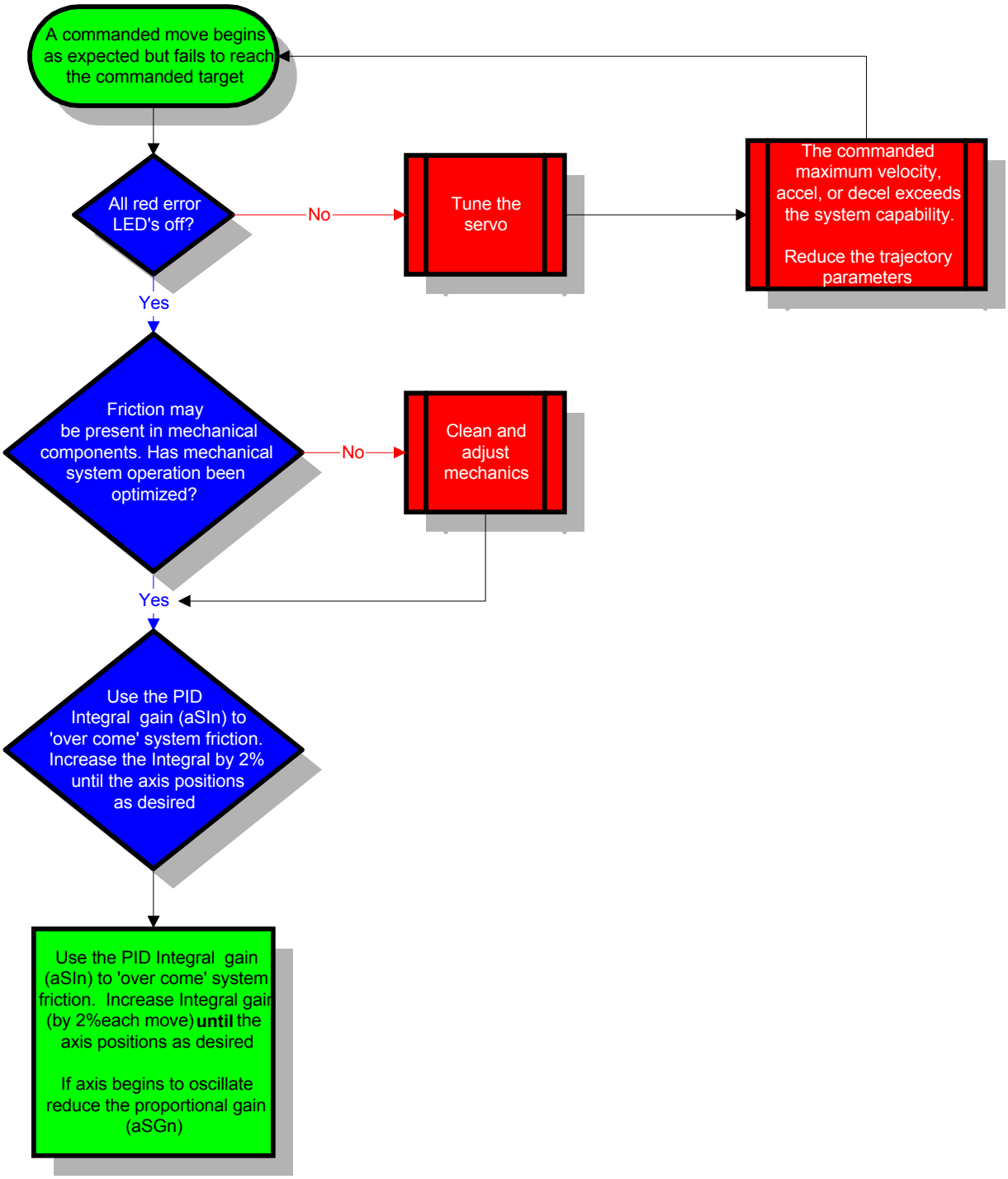
Troubleshooting - Servo Motion chart #1



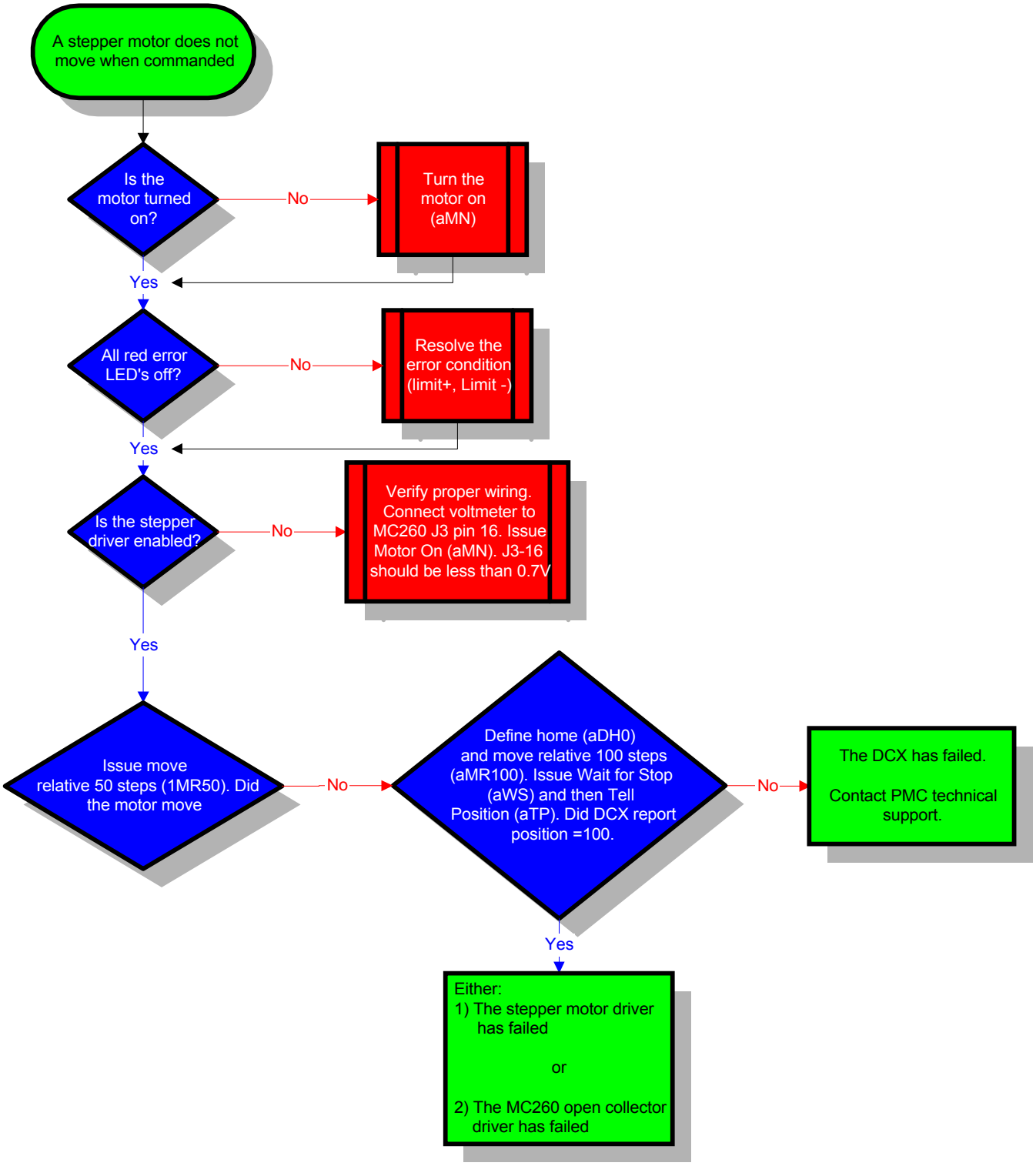
Troubleshooting - Servo Motion chart #2



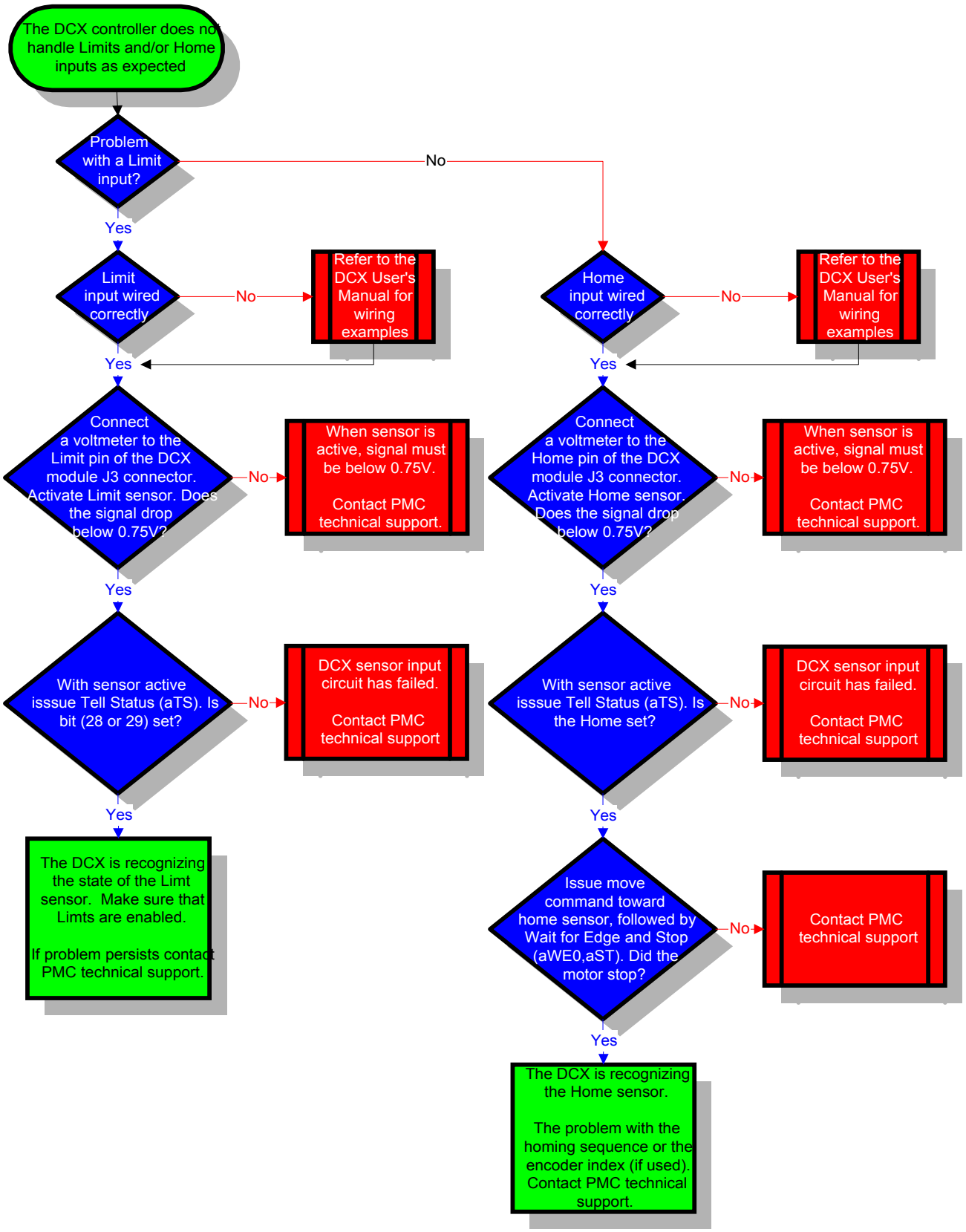
Troubleshooting - Servo Motion chart #3



Troubleshooting - Stepper Motion chart #1



Troubleshooting - Limits and Home



Chapter Contents

- Introduction to PDF
- Printing a complete PDF document
- Printing selected pages of a PDF document
- Paper
- Binding
- Pricing
- Obtaining a Word 2000 version of this user manual

Chapter 13

Printing a PDF Document

Introduction to PDF
PDF stands for Portable Document Format. It is the defacto standard for transporting electronic documents. PDF files are based on the PostScript language imaging model. This enables sharp, color-precise printing on almost all printers.

Printing a complete PDF document
It is **not recommended** that large PDF documents be printed on personal computer printers. The 'wear and tear' incurred by these units, coupled with the difficulties of two sided printing, typically resulting in degraded performance of the printer and a whole lot of wasted paper. PMC recommends that PDF document be printer by a full service print shop that uses digital (computer controlled) copy systems with paper collating/sorting capability.

Printing selected pages of a PDF document
While viewing a PDF document with Adobe Reader (or Adobe Acrobat), any page or range of pages can be printed by a personal computer printer by:

Selecting the printer icon on the tool bar
Selecting **Print** from the Adobe **File** menu

Paper
The selection of the paper type to be used for printing a PDF document should be based on the target market for the document. For a user's manual with extensive graphics that is printed on both sides of a page the minimum recommended paper type is 24 pound. A heavier paper stock (26 – 30 pound) will reduce the 'bleed through' inherent with printed graphics. Typically the front and back cover pages are printed on heavy paper stock (50 to 60 pound).

Binding
Unlike the binding of a book or catalog, a user's manual distributed in as a PDF file will typically use 'comb' or 'coil' binding. This service is provided by most full service print shops. Coil binding is

suitable for documents with no more than 100 pieces of paper (24 pound). Comb binding is acceptable for documents with as many as 300 pieces of paper (24 pound). Most print shops stock a wide variety of 'combs'. The print shop can recommend the appropriate 'comb' based on the number of pages.

Pricing
The final cost for printing and binding a PDF document is based on:

- Quantity per print run
- Number of pages
- Paper type

The price range for printing and binding a PDF document similar to this user manual will be $15 to $30 (printed in Black & White) in quantities of 1 to 10 pieces.

Obtaining a Word 2000 version of this user manual
This user document was written using Microsoft's Word 2000. Qualified OEM's, Distributors, and Value Added Reps (VAR's) can obtain a copy of this document for

- Editing
- Customization
- Language translation.

Please contact Precision MicroControl to obtain a Word 2000 version of this document.

Chapter 13

Glossary

Accuracy - A measure of the difference between the expected position and actual position of a motion system.

Actuator - Device which creates mechanical motion by converting energy to mechanical energy.

Axis Phasing - An axis is properly phased when a commanded move in the positive direction causes the encoder decode circuitry of the controller to increment the reported position of the axis.

Back EMF - The voltage generated when a permanent magnet motor is rotated. This voltage is proportional to motor speed and is present regardless of whether the motor windings are energized or de-energized.

Closed Loop - A broadly applied term, relating to any system in which the output is measured and compared to the input. The output is then adjusted to reach the desired condition. In motion control, the term typically describes a system utilizing a velocity and/or position transducer to generate correction signals in relation to desired parameters.

Commutation - The action of applying currents or voltages to the proper motor phases in order to produce optimum motor torque.

Critical Damping - A system is critically damped when the response to a step change in desired velocity or position is achieved in the minimum possible time with little or no overshoot.

DAC - The digital-to-analog converter (DAC) is the electrical interface between the motion controller and the motor amplifier. It converts the digital voltage value computed by the motion controller into an

analog voltage. The more DAC bits, the finer the analog voltage resolution. DAC's are available in three common sizes: 8, 12, and 16 bit. The bit count partitions the total peak-to-peak output voltage swing into 256, 4096, or 65536 DAC steps, respectively.

Dead Band - A range of input signals for which there is no system response.

Driver - Electronics which convert step and direction inputs to high power currents and voltages to drive a step motor. The step motor driver is analogous to the servo motor amplifier.

Dual Loop Servo – A servo system that combines a velocity mode amplifier/tachometer with a position loop controller/encoder. It is recommended that the encoder not be directly coupled to the motor. The linear scale encoder should be mounted on the external mechanics, as closely coupled as possible to the 'end effector'

Duty Cycle - For a repetitive cycle, the ratio of on time to total time:

Efficiency - The ratio of power output to power input.

Encoder - A type of feedback device which converts mechanical motion into electrical signals to indicate actuator position or velocity.

End Effector – The point of focus of a motion system. The tools with which a motion system will work. Example: The leading edge of the knife is the *end effector* of a three axis (XYZ) system designed to cut patterns from vinyl.

Following Error - The difference between the calculated desired trajectory position and the actual position.

Friction - A resistance to motion caused by contacting surfaces. Friction can be constant with varying speed (Coulomb friction) or proportional to speed (viscous friction).

Holding Torque - Sometimes called static torque, holding torque specifies the maximum external torque that can be applied to a stopped, energized motor without causing the rotor to rotate continuously.

Inertia - The measure of an object's resistance to a change in its current velocity. Inertia is a function of the object's mass and shape.

Kd - K is a generally accepted variable used to represent gain, an arbitrary multiplier, or a constant. The lower case 'd' designates derivative gain.

Ki - K is a generally accepted variable used to represent gain, an arbitrary multiplier, or a constant. The lower case 'i' designates integral gain.

Kp - K is a generally accepted variable used to represent gain, an arbitrary multiplier, or a constant. The lower case 'p' designates proportional gain.

Limits - Motion system sensors (hard limits) or user programmable range (soft limits) that alert the motion controller that the physical end of travel is being approached and that motion should stop.

MCAPI - The Motion Control Application Programming Interface - this is the programming interface used by Windows programmers to control PMC's family of motion control cards.

MCCL - Motion Control Command Language - this is the command language used to program PMC's family of motion control cards.

Micro-Stepping - Stepper drive systems have a fixed number of electromechanical detents or steps. Micro stepping is an electronic technique to break each detent or step into smaller parts. This results in higher positional resolution and smoother operation.

Open Loop – A control system in which the control output is not referenced or scaled to an external feedback.

Position Error - see following error.

Position Move - Unlike a velocity move, a position move includes a predefined stopping position. The trajectory generator will determine when to begin deceleration in order to ensure the actual stopping point is at the desired target position.

PWM - Pulse Width Modulation is a method of controlling the average current in a motor’s phase windings by varying the duty cycle of transistor switches.

Repeatability - The degree to which the positioning accuracy for a given move performed repetitively can be duplicated.

Resonance - A condition resulting from energizing a motor at a frequency at or close to the motor's natural frequency.

Resolution - The smallest positioning increment that can be achieved.

Resolver - A type of feedback device which converts mechanical position into an electrical signal. A resolver is a variable transformer that divides the impressed AC signal into sine and cosine output signals. The amplitude of these signals represents the absolute position of the resolver shaft.

Slew - That portion of a move made at constant, non-zero velocity.

Step Response - An instantaneous command to a new position. Typically used for tuning a closed loop system, ramping (velocity, acceleration, and deceleration) is not applied nor calculated for the move.

Tachometer - A device attached to a moving shaft that generates a voltage signal directly proportional to rotational speed.

Torque -

Velocity Mode Amplifier – An amplifier that requires a tachometer to provide the feedback used to close the velocity loop within the amplifier.

Velocity Move - A move where no final stopping position is given to the motion controller. When a start command is issued the motor will rotate indefinitely until it is commanded to stop.

Chapter Contents

Appendix

Power Supply Requirements

Part Number	*+5 VDC*	*+12 VDC*	*-12 VDC*	*Unit*
DCX-VM200	0.8	-----	-----	A
DCX-MC200	.2	.01	.01	A
DCX-MC210	.2	*	*	A
DCX-MC260	.2	-----	-----	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	0
Current Velocity	0
Velocity Gain	0
Acceleration Gain	0
Deceleration Gain	0
Velocity Override	1
Torque Limit	10
Proportional Gain	.2
Derivative Gain	.1
Integral Gain	.01
Integration Limit	50
Maximum Following Error	1024
Motion Limits	disabled
Low Limit of Movement	0
High Limit of Movement	0
Servo Loop Rate	MS
Stepper Pulse Range	HS
Position Count	0
Optimal Count	0
Index Count	0
Auxiliary Status	0
Position	0
Target	0
Optimal Position	0
Breakpoint Position	0
Position Deadband	0
User Scale	1
User Zero	0
User Offset	0
User Rate Conversion	1
User Output Constant	1
Jog Gain	1000
Jog Offset	2.5
Jog Deadband	.1
Jog Acceleration	1000
Jog Minimum Velocity	0
Sampling Frequency	0
Slave Ratio	1

MCCL Error Codes

When executing MCCL (Motion Control Command Language) command sequences the command interpreter will report the following error code when appropriate:

Description	*Error code*
No error	0
Unrecognized command	1
Bad command format	2
I/O error	3
Command string to long	4
Command Parameter Error	-1
Command Code Invalid	-2
Negative Repeat Count	-3
Macro Define Command Not First	-4
Macro Number Out of Range	-5
Macro Doesn't Exist	-6
Command Canceled by User	-7
Contour Path Command Not First	-8
Contour Path Command Parameter Invalid	-9
Contour Path Command Doesn't Specify an AXIS	-10
No axis specified	-14
Axis not assigned	-15
Axis already assigned	-16
Axis duplicate assigned	-17

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 macro's in memory error code –6 is displayed.

```
RM     <return>                          ;reset all DCX macros
1MN,2MN,3MN,MC10    <return>             ;turn on axes 1, 2, & 3, call macro 10

    ? -6                                 ;error -6 = macro doesn't exist
    {C3} MC10                            ;error report;
                                         ;     C3 = command number
                                         ;     MC10 = offending command
```

Stand Alone Applications

When used in the stand-alone configuration, power must be provided to the DCX through the P1 connector. In this configuration, one of the available communication interface modules (RS-232 or IEE-488) can be used to send commands to the controller.

Multiple User Communication Interfaces

The DCX supports multiple user interfaces. The interfaces supported are:

VME bus
Serial ASCII (requires installation of one MC300 RS-232 module)
GPIB ASCII (Requires installation of one MF310 IEEE-488 module)

Each of the four available command interfaces are handled by a separate CPU task controller. Any or all user interface tasks can run simultaneously and independently.

Multi user interfaces offer the machine programmer great flexibility in supporting messaging. A typical example of this capability would be a Vertical Form Fill and Seal (VFFS) packaging application. After an empty bag is filled with coffee, a 'hot seal bar' is used to close the bag. Most machine operations would typically be programmed using PMC's Motion Control Application Programming Interface (MCAPI) which communicates via the binary interface. But the monitoring of the temperature of the hot seal bar could be implemented via a DCX macro running as a background task. This would allow the operator interface to display a preprogrammed ASCII text message issued from the DCX if the seal bar is not within the proper temperature range. For additional information on messaging please refer to the section **titled Outputting Formatted Messages**.

The Command interpreter status word in dual port memory (808h offset) has the following format:

Bit	*Flag*
0	Host Binary Interface - Busy
1	Unused
2	Unused
3	Unused
4	Host Ascii Interface - Busy
5	Unused
6	Unused
7	Host Ascii Interface - Loading File
8	Serial Interface - Busy
9	Unused
10	Unused
11	Serial Interface - Loading File
12	GPIB Interface - Busy
13	Unused
14	Unused
15	GPIB Interface - Loading File

File Operations

The DCX has the ability to store text files in its' on-board memory. The primary purpose for this capability is for support of the Plotting function described in a later section. This section describes the commands used to load and manage files in the controller's file system.

The standard memory on the DCX provides 8 Kbytes (8,192 characters) of memory for file storage. With the expanded memory option, an additional 191 Kbytes (195,584 characters) of file storage is available. The file system organizes the memory using a directory that can maintain up to 127 separate files. Each file is identified by an integer number from 1 to 127. A file can be any size, up to the available memory in the file system. A file can contain any number of ASCII characters, but the total space required by all files cannot exceed the file system memory. Note that file storage space is allocated in 1024 byte blocks, so a file that is 100 bytes will occupy 1024 bytes of file system memory.

Prior to loading any files onto the DCX controller, the file memory must be initialized, similar to formatting a floppy disk of a personal computer. This is accomplished by issuing the Format command to the DCX.

Example:FO

Initializing the file system memory will delete all existing files from memory and make all space available for new files.

Files can be loaded into the controller using any of the ASCII interfaces. Files can't be loaded using the binary interface. To load a file, issue the LOad command with a parameter from 1 to 127 specifying the file to be loaded. After issuing this command the board will accept all characters received at the interface and place them in the file. To terminate loading of the file, send the End-Of-File ASCII character (1A hex) to the interface. This code can be generated by pressing the control-Z key combination on the PC keyboard. When a file is loaded in this manner, all previous contents of the specified file will be lost.

A program on the Utility Disk, DCX2LOAD can be used to copy a file on the host computer's disk drive into the DCX's file storage. This DOS program takes the name of the file as a command line parameter. Include a command line parameter with the form /C"LOn", where n is a number between 1 and 127 specifying the file number on the DCX that the file will be copied to. The program will terminate immediately after copying the file.

Example:DCX2LOAD /C"LO10" FILE.EXT

This DOS command line will copy a file named 'FILE.EXT' to the DCX and store it as file number 10.

To display what has been loaded into a file, issue the TYpe command to the controller from on of the ASCII interfaces. The controller will respond by displaying the entire contents of the file. To pause the file display, press the control-S key combination. To resume the display, press the control-Q key combination.

The Directory Listing command is used to display a list of what files are loaded and their respective sizes.

The Remove File command issued with a parameter value from 1 to 127 will delete the respective file from the controller's memory.

The following DCX commands are available for creating and maintaining files in the controller's memory:

DL Directory Listing
syntax: DL
code: 271d, 10Fh
parameter: none
compatibility: N/A
see also: TY
explanation: This command causes the DCX to display information about the files that are currently loaded in its' memory, and the amount of file storage space remaining for new files.

example: DL

```
FILE   1:  SIZE = 8377
FILE  10:  SIZE = 6129
FILE 100:  SIZE = 26574
FREE SPACE = 162696 BYTES
```

EC Execute Command file
syntax: EC
code: 279d, 117h
parameter: 1 <= n <= 127
compatibility: N/A
see also:
explanation: Implemented Execute Command file (EC, code=279) command. This command assumes the on-board file specified by the EC command parameter contains DCX commands. If single stepping is enabled while executing the command file, the file and line numbers will be displayed with the command index.

FO FOrmat file system
syntax: FO
code: 270d, 10Eh
parameter: none
compatibility: N/A
see also: RF
explanation: This command initializes the DCX's file storage memory and makes the full amount of memory allocated for file storage available. Any files that are stored in the file system will be erased when this command is executed.

LO LOad file
syntax: LOn
code: 274d, 112h
parameter: integer (1 <= n <= 127)
compatibility: N/A
see also: TY

explanation: When this command is issued over one of the DCX's ASCII interfaces, the controller will begin accepting characters from that interface and storing them sequentially in the file specified by the command parameter. Loading of the file will be terminated when the End-Of-File ASCII character (1A hex) is received.

RF Remove File
syntax: RFn
code: 273d, 111h
parameter: integer (1 <= n <= 127)
compatibility: N/A
see also: DL
explanation: This command will delete the specified file from the DCX's storage memory. Functionally this command will make the memory occupied by the specified file available for storage of new files, and set the directory entry for this file to 'empty'.

TY TYpe file
syntax: aCMn
code: 272d, 110h
parameter: integer (0 <= n <= 127)
compatibility: N/A
see also: DL
explanation: This command will cause the contents of the specified file to be sent to the ASCII interface that the command was issued to.

example: TY10

```
IN;SP1;
PA1000,0;
PA1000,1000;
PA0,1000;
PA0,0;
```

HPGL Plotting

The DCX has the ability to execute Hewlett Packard Graphic Language (HPGL) commands. This language is a popular standard used for controlling X-Y tables in plotting and engraving applications. The DCX can accept HPGL commands sent to it over an ASCII interface port, or it can read them from an on-board file (see the section on file commands). Prior to executing HPGL commands from either source, the plotting environment must be setup by issuing DCX plotter configuration commands.

After the configuring the plotting environment, the Plotting Enable command will cause the DCX to accept HPGL commands from the ASCII interface port that the PE command was issued to. Once plotting is enabled, the DCX will accept only HPGL formatted commands until plotting is disabled by sending it an End-Of-File ASCII character (1A hex). Plotting can be re-enabled without issuing the configuration commands if no changes are required.

Alternatively, HPGL commands can be stored in an on-board file and the Plot File command issued to the DCX using the file number as the command parameter. In this mode, the DCX will execute the HPGL commands in the file without host computer assistance. Similar to the Plot Enable command, the Plot File command can be issued multiple times without issuing the configuration commands if no changes are required.

A program on the Utility Disk, DCX2LOAD can be used to plot a file on the host computer's disk drive. This DOS program takes the name of the file as a command line parameter. Include a command line parameter with the form /C"PE". The program will terminate immediately after plotting the file.

Example:DCX2LOAD /C"PE" FILE.EXT

This DOS command line will plot a file named 'FILE.EXT' using the DCX.

DCX Plotter configuration commands:

PA Plotter Acceleration
syntax: PAn
code: 289d, 121h
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: PV
explanation: This command sets the acceleration and deceleration for contoured motion of the X and Y plotting axes. The default value for the plotting acceleration is 1.0.

PD Pen Down
syntax: PDn
code: 293d, 125h
parameter: integer or real (0 <= n <= 1099)
compatibility: N/A
see also: PU,SP
explanation: Specifies a macro containing DCX commands that will be called whenever the HPGL Pen Down (PD) command is executed.

PE Plotting Enable
syntax: PE
code: 281d, 119h
parameter: none
compatibility: MC200, MC210, MC260
see also: PF
explanation: This command places the DCX in plotting mode. Once this command is executed, the board will only accept HPGL commands from the command port that the PE command was issued to. To terminate plotting, the End-Of-File ASCII character (1A hex) must be sent to the same port.

PF Plot File
syntax: PFn
code: 280d, 118h
parameter: integer (1 <= n <= 127)
compatibility: MC200, MC210, MC260
see also: PE
explanation: This command will cause the specified file to be plotted. See the section in this manual describing file commands for an explanation of how to load a file prior to issuing this command. This command can be executed as either a foreground or background task.

PI Plotter Initialize macro
syntax: PIn
code: 291d, 123h
parameter: integer (0 <= n)
compatibility: N/A
see also: PU,PD,SP
explanation: Specifies a macro containing DCX commands that will be called whenever the HPGL Initialize (IN) command is executed.

PQ Plotter Quick velocity
syntax: PQn
code: 290d, 122h
parameter: integer or real (0 <= n)
compatibility: MC200, MC210, MC260
see also: PV
explanation: This command sets the velocity for 'pen up' motion of the X and Y plotting axes. The parameter to this command will be in the plotting units that were set by the Plotter X and Y Scaling commands.

PU Pen Up macro
syntax: PUn
code: 292d, 124h
parameter: integer (0 <= n <= 1099)
compatibility: N/A
see also: PD,SP
explanation: Specifies a macro containing DCX commands that will be called whenever the HPGL Pen Up (PU) command is executed.

PV Plotter Velocity
syntax: PVn
code: 288d, 120h
parameter: integer or real (0 <= n)
compatibility: MC200, MC260
see also: PA,PQ
explanation: This command sets the velocity for the 'pen down' contoured motion of the X and Y plotting axes. The parameter to this command will be in the plotting units that were set by the Plotter X and Y Scaling commands. Alternatively, the parameter to this command will be used as a scaling factor if a HPGL Velocity Select command is executed during plotting. The default value for the plotting velocity is 1.0.

PX Plotter X axis
syntax: PXn
code: 282d, 11Ah
parameter: integer (0 <= n <= 6)
compatibility: MC200, MC210, MC260
see also: PY
explanation: Specifies the DCX controller axis that will be used for the X axis motion in executing the HPGL commands.

PY Plotter Y axis
syntax: PYn
code: 283d, 11Bh
parameter: integer (0 <= n <=6)
compatibility: MC200, MC210, MC260
see also: PX
explanation: Specifies the DCX controller axis that will be used for the Y axis motion in executing the HPGL commands.

SP Select Pen macro
syntax: SPn
code: 294d, 126h
parameter: integer (0 <= n <=1099)
compatibility: N/A
see also: PU,PD
explanation: Specifies a macro containing DCX commands that will be called whenever the HPGL Select Pen (SP) command is executed. Prior to calling the macro, the pen number that was specified with the HPGL Select Pen command will be placed in User Register 0. The macro can use the pen number stored in User Register 0 to determine which pen or tool to select.

XO plotter X Offset
syntax: XOn
code: 286d, 11Eh
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: XS,YO

explanation: Specifies an offset in the origin for the plotting X axis. The parameter to this command should be in the same plotter units that are established with the Plotter X Scale (XS) command. The value set with this command will be used as the axes' User Offset when the Plotting Enable (PE) command is issued. This offset is independent of the offset that can be set with the HPGL Input P1 and P2 (IP), and Scale (SC) Commands. The default offset is 0.

XS plotter X Scale
syntax: XSn
code: 284d, 11Ch
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: XO,YS
explanation: Specifies the scaling factor for the plotting X axis. The parameter to this command should be the number of encoder counts or steps per plotter unit. The value set with this command will be used as the axes' User Scale when the Plotting Enable (PE) command is issued. This scaling is independent of the scaling that can be set with the HPGL Input P1 and P2 (IP), and Scale (SC) Commands. The default scale factor is 1.0, or 1 plotter unit per encoder count or step.

YO plotter Y Offset
syntax: YOn
code: 287d, 11Fh
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: XO,YS
explanation: Specifies an offset in the origin for the plotting Y axis. The parameter to this command should be in the same plotter units that are established with the Plotter Y Scale (YS) command. The value set with this command will be used as the axes' User Offset when the Plotting Enable (PE) command is issued. This offset is independent of the offset that can be set with the HPGL Input P1 and P2 (IP), and Scale (SC) Commands. The default offset is 0.

YS plotter Y Scale
syntax: PAn
code: 285d, 11Dh
parameter: integer or real
compatibility: MC200, MC210, MC260
see also: XS,YO
explanation: Specifies the scaling factor for the plotting Y axis. The parameter to this command should be the number of encoder counts or steps per plotter unit. The value set with this command will be used as the axes' User Scale when the Plotting Enable (PE) command is issued. This scaling is independent of the scaling that can be set with the HPGL Input P1 and P2 (IP), and Scale (SC) Commands. The default scale factor is 1.0, or 1 plotter unit per encoder count or step.

Supported HPGL PLOTTING commands

When plotting is configured (using the DCX plotting configuration commands) and then enabled on the DCX, it will accept and execute Hewlett Packard Graphic Language (HPGL) commands. This appendix describes the HPGL commands that have been implemented on the DCX.

In the syntax descriptions, upper case letters are required, lower case letter represent numerical values, and parenthesis delimit optional values.

Arc Absolute
syntax: AA x_abs,y_abs,angle (,chord)

Arc Relative
syntax: AA x_rel,y_rel,angle (,chord)

CIrcle
syntax: CI radius

INInitialize
syntax: IN

IP Input P1 and P2
syntax: IP p1x,p1y(,p2x,p2y)

PA Plot Absolute
syntax: PA x,y(,...)

PD Pen Down
syntax: PD (x,y(,...))

PR Plot Relative
syntax: PR x,y(,...)

PU Pen Up
syntax: PD (x,y(,...))

SC Scale
syntax: SC xmin,xmax,ymin,ymax

SP Select Pen
syntax: SP n

VS Velocity Select
syntax: VS v

Index

D

E

F

G

H

I

J

L

M

O

P

R

S

T

U

V

W