**USER MANUAL**

# DMC-40x0

**Manual Rev. 1.0d**

**By Galil Motion Control, Inc.**

*Galil Motion Control, Inc.*
*270 Technology Way*
*Rocklin, California 95765*
*Phone: (916) 626-0101*
*Fax: (916) 626-0102*
*E-mail Address: support@galilmc.com*
*URL: www.galilmc.com*

*Rev 2/09*

# Using This Manual

This user manual provides information for proper operation of the DMC-40x0 controller. A separate supplemental manual, the Command Reference, contains a description of the commands available for use with this controller.

Your DMC-40x0 motion controller has been designed to work with both servo and stepper type motors. Installation and system setup will vary depending upon whether the controller will be used with stepper motors or servo motors. To make finding the appropriate instructions faster and easier, icons will be next to any information that applies exclusively to one type of system. Otherwise, assume that the instructions apply to all types of systems. The icon legend is shown below.

Attention: Pertains to servo motor use.

Attention: Pertains to stepper motor use.


Attention: Pertains to controllers with more than 4 axes.

Please note that many examples are written for the DMC-4040 four-axes controller or the DMC-4080 eight axes controller. Users of the DMC-4030 3-axis controller, DMC-4020 2-axes controller or DMC-4010 1-axis controller should note that the DMC-4030 uses the axes denoted as XYZ, the DMC-4020 uses the axes denoted as XY, and the DMC-4010 uses the X-axis only.

Examples for the DMC-4080 denote the axes as A,B,C,D,E,F,G,H. Users of the DMC-4050 5-axes controller. DMC-4060 6-axes controller or DMC-4070, 7-axes controller should note that the DMC-4050 denotes the axes as A,B,C,D,E, the DMC-4060 denotes the axes as A,B,C,D,E,F and the DMC-4070 denotes the axes as A,B,C,D,E,F,G. The axes A,B,C,D may be used interchangeably with A,B,C,D.

**WARNING: Machinery in motion can be dangerous! It is the responsibility of the user to design effective error handling and safety protection as part of the machinery. Galil shall not be liable or responsible for any incidental or consequential damages.**

# Contents

# Chapter 1 Overview

## Introduction

The DMC-40x0 Series are Galil's highest performance stand-alone controller. The controller series offers many enhanced features including high speed communications, non-volatile program memory, faster encoder speeds, and improved cabling for EMI reduction.

Each DMC-40x0 provides two communication channels: high speed RS-232 (2 channels up to 115K Baud) and 10BaseT Ethernet. The controllers allow for high-speed servo control up to 22 million encoder counts/sec and step motor control up to 6 million steps per second. Sample rates as low as 31.25 μsec per axis are available.

A Flash EEPROM provides non-volatile memory for storing application programs, parameters, arrays and firmware. New firmware revisions are easily upgraded in the field.

The DMC-40x0 is available with up to eight axes in a single stand alone unit. The DMC-4010, 4020, 4030, 4040 are one thru four axes controllers and the DMC-4050, 4060, 4070, 4080 are five thru eight axes controllers. All eight axes have the ability to use Galil's integrated amplifiers or drivers and connections for integrating external devices.

Designed to solve complex motion problems, the DMC-40x0 can be used for applications involving jogging, point-to-point positioning, vector positioning, electronic gearing, multiple move sequences, and contouring. The controller eliminates jerk by programmable acceleration and deceleration with profile smoothing. For smooth following of complex contours, the DMC-40x0 provides continuous vector feed of an infinite number of linear and arc segments. The controller also features electronic gearing with multiple master axes as well as gantry mode operation.

For synchronization with outside events, the DMC-40x0 provides uncommitted I/O, including 8 opto-isolated digital inputs (16 inputs for DMC-4050 thru DMC-4080), 8 high power optically isolated outputs (16 outputs for DMC-4050 thru DMC-4080), and 8 analog inputs for interface to joysticks, sensors, and pressure transducers. The DMC-40x0 also has an additional 32 I/O at 3.3V logic. Further I/O is available if the auxiliary encoders are not being used (2 inputs / each axis). Dedicated optoisolated inputs are provided for forward and reverse limits, abort, home, and definable input interrupts.

Commands can be sent in either Binary or ASCII. Additional software is available for automatic-tuning, trajectory viewing on a PC screen, CAD translation, and program development using many environments such as Visual Basic, C, C++ etc. Drivers for Windows XP (32 & 64 bit).

# Overview of Motor Types

The DMC-40x0 can provide the following types of motor control:

1. Standard servo motors with +/- 10 volt command signals
2. Brushless servo motors with sinusoidal commutation
3. Step motors with step and direction signals
4. Other actuators such as hydraulics - For more information, contact Galil.

The user can configure each axis for any combination of motor types, providing maximum flexibility.

## Standard Servo Motor with +/- 10 Volt Command Signal

The DMC-40x0 achieves superior precision through use of a 16-Bit motor command output DAC and a sophisticated PID filter that features velocity and acceleration feed-forward, an extra pole filter and integration limits.

The controller is configured by the factory for standard servo motor operation. In this configuration, the controller provides an analog signal (+/- 10 volts) to connect to a servo amplifier. This connection is described in Chapter 2.

## Brushless Servo Motor with Sinusoidal Commutation

The DMC-40x0 can provide sinusoidal commutation for brushless motors (BLM). In this configuration, the controller generates two sinusoidal signals for connection with amplifiers specifically designed for this purpose.

**Note:** The task of generating sinusoidal commutation may be accomplished in the brushless motor amplifier. If the amplifier generates the sinusoidal commutation signals, only a single command signal is required and the controller should be configured for a standard servo motor (described above).

Sinusoidal commutation in the controller can be used with linear and rotary BLMs. However, the motor velocity should be limited such that a magnetic cycle lasts at least 6 milliseconds with a standard update rate of 1 millisecond. For faster motors, please contact the factory.

To simplify the wiring, the controller provides a one-time, automatic set-up procedure. When the controller has been properly configured, the brushless motor parameters may be saved in non-volatile memory.

The DMC-40x0 can control BLMs equipped with Hall sensors as well as without Hall sensors. If Hall sensors are available, once the controller has been setup, the brushless motor parameters may be saved in non-volatile memory. In this case, the controller will automatically estimate the commutation phase upon reset. This allows the motor to function immediately upon power up. The Hall effect sensors also provide a method for setting the precise commutation phase. Chapter 2 describes the proper connection and procedure for using sinusoidal commutation of brushless motors.

## Stepper Motor with Step and Direction Signals

The DMC-40x0 can control stepper motors. In this mode, the controller provides two signals to connect to the stepper motor: Step and Direction. For stepper motor operation, the controller does not require an encoder and operates the stepper motor in an open loop fashion. Chapter 2 describes the proper connection and procedure for using stepper motors.

If encoders are available on the stepper motor, Galil's Stepper Position Maintenance Mode may be used for automatic monitoring and correction of the stepper position. See Stepper Position Maintenance Mode (SPM) in Chapter 6 for more information.

# Overview of External Amplifiers

The amplifiers should be suitable for the motor and may be linear or pulse-width-modulated. An amplifier may have current feedback, voltage feedback or velocity feedback.

## Amplifiers in Current Mode

Amplifiers in current mode should accept an analog command signal in the +/-10 volt range. The amplifier gain should be set such that a +10V command will generate the maximum required current. For example, if the motor peak current is 10A, the amplifier gain should be 1 A/V.

## Amplifiers in Velocity Mode

For velocity mode amplifiers, a command signal of 10 volts should run the motor at the maximum required speed. The velocity gain should be set such that an input signal of 10V runs the motor at the maximum required speed.

## Stepper Motor Amplifiers

For step motors, the amplifiers should accept step and direction signals.

# Overview of Galil Amplifiers and Drivers

With the DMC-40x0 Galil offers a variety of Servo Amplifiers and Stepper Drivers that are integrated into the same enclosure as the controller. Using the Galil Amplifiers and Drivers provides a simple straightforward motion control solution in one box.

## A1 – AMP-43040 (-D30x0)

The AMP-43040 (four-axis) and AMP-43020 (two-axis) are multi-axis brush/brushless amplifiers that are capable of handling 500 watts of continuous power per axis. The AMP-43040/43020 Brushless drive modules are connected to a DMC-40x0. The standard amplifier accepts DC supply voltages from 18-80 VDC.

## A2 – AMP-43140 (-D3140)

The AMP-43140 contains four linear drives for operating small brush-type servo motors. The AMP-43140 requires a ± 12–30 DC Volt input. Output power is 20 W per amplifier or 60 W total. The gain of each transconductance linear amplifier is 0.1 A/V at 1 A maximum current. The typical current loop bandwidth is 4 kHz.

## A3 – SDM-44040 (-D4040)

The SDM-44040 is a stepper driver module capable of driving up to four bipolar two-phase stepper motors. The current is selectable with options of 0.5, 0.75, 1.0, and 1.4 Amps/Phase. The step resolution is selectable with options of full, half, 1/4 and 1/16.

## A4 – SDM-44140 (-D4140)

The SDM-44140 microstepper module drives four bipolar two-phase stepper motors with 1/64 microstep resolution (the SDM-44140 drives two). The current is selectable with options of 0.5, 1.0, 2.0, & 3.0 Amps per axis.

# DMC-40x0 Functional Elements

The DMC-40x0 circuitry can be divided into the following functional groups as shown in Figure 1.1 and discussed below.

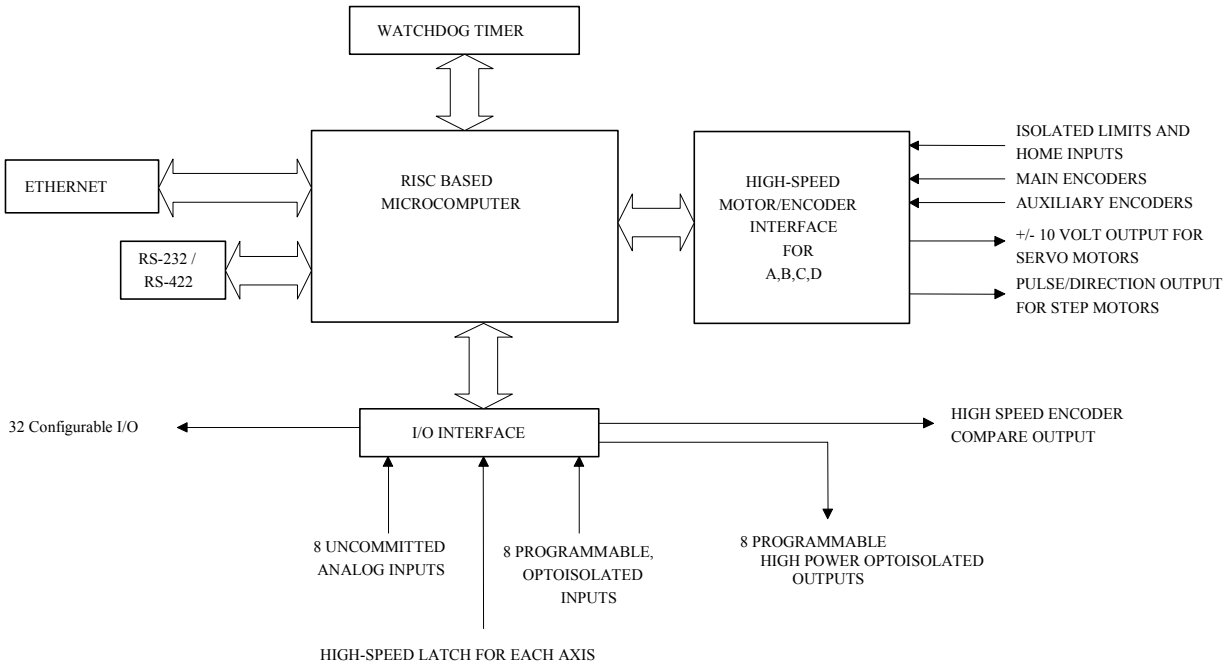



*Figure 1.1 - DMC-40x0 Functional Elements*

## Microcomputer Section

The main processing unit of the controller is a specialized Microcomputer with RAM and Flash EEPROM. The RAM provides memory for variables, array elements, and application programs. The flash EEPROM provides non-volatile storage of variables, programs, and arrays. The Flash also contains the firmware of the controller, which is field upgradeable.

## Motor Interface

Galil's GL-1800 custom, sub-micron gate array performs quadrature decoding of each encoder at up to 12 MHz. For standard servo operation, the controller generates a +/-10 volt analog signal (16 Bit DAC). For sinusoidal commutation operation, the controller uses two DACs to generate two +/-10 volt analog signals. For stepper motor operation, the controller generates a step and direction signal.

## Communication

The communication interface with the DMC-40x0 consists of high speed RS-232 and Ethernet. The Ethernet is 10/100Bt and the two RS-232 channels can generate up to 115K.

## General I/O

The DMC-40x0 provides interface circuitry for 8 bi-directional, optoisolated inputs, 8 high power optoisolated outputs and 8 analog inputs with 12-Bit ADC (16-Bit optional). The DMC-40x0 also has an additional 32 I/O (3.3V

logic) and unused auxiliary encoder inputs may also be used as additional inputs (2 inputs / each axis). The general inputs can also be used as high speed latches for each axis. A high speed encoder compare output is also provided.

**4080** The DMC-4050 through DMC-4080 controller provides an additional 8 optoisolated inputs and 8 high power optoisolated outputs.

## System Elements

As shown in Fig. 1.2, the DMC-40x0 is part of a motion control system which includes amplifiers, motors and encoders. These elements are described below.

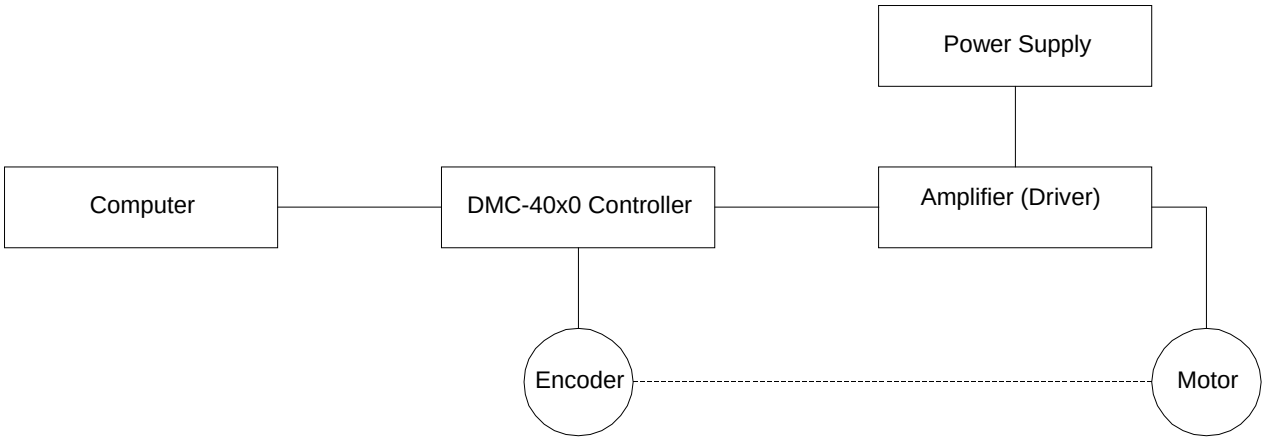



*Figure 1-.2 - Elements of Servo systems*

## Motor

A motor converts current into torque which produces motion. Each axis of motion requires a motor sized properly to move the load at the required speed and acceleration. (Galil's MotorSizer Web tool can help you with motor sizing: www.galilmc.com/support/motorsizer)

The motor may be a step or servo motor and can be brush-type or brushless, rotary or linear. For step motors, the controller can be configured to control full-step, half-step, or microstep drives. An encoder is not required when step motors are used.

Other motors and devices such as Ultrasonic Ceramic motors and voice coils can be controlled with the DMC-40x0.

## Amplifier (Driver)

For each axis, the power amplifier converts a +/-10 volt signal from the controller into current to drive the motor. For stepper motors, the amplifier converts step and direction signals into current. The amplifier should be sized properly to meet the power requirements of the motor. For brushless motors, an amplifier that provides electronic commutation is required or the controller must be configured to provide sinusoidal commutation. The amplifiers may be either pulse-width-modulated (PWM) or linear. They may also be configured for operation with or without a tachometer. For current amplifiers, the amplifier gain should be set such that a 10 volt command generates the maximum required current. For example, if the motor peak current is 10A, the amplifier gain should be 1 A/V. For velocity mode amplifiers, 10 volts should run the motor at the maximum speed.

Galil offers amplifiers that are integrated into the same enclosure as the DMC-40x0. See the Integrated Amplifiers and Drivers section in the

Appendices or http://galilmc.com/products/accelera/dmc40x0.html for more information.

## Encoder

An encoder translates motion into electrical pulses which are fed back into the controller. The DMC-40x0 accepts feedback from either a rotary or linear encoder. Typical encoders provide two channels in quadrature, known as CHA and CHB. This type of encoder is known as a quadrature encoder. Quadrature encoders may be either single-ended (CHA and CHB) or differential (CHA,CHA- and CHB,CHB-). The DMC-40x0 decodes either type into quadrature states or four times the number of cycles. Encoders may also have a third channel (or index) for synchronization.

The DMC-40x0 can also interface to encoders with pulse and direction signals. Refer to the "CE" command in the command reference for details.

There is no limit on encoder line density; however, the input frequency to the controller must not exceed 5,500,000 full encoder cycles/second (22,000,000 quadrature counts/sec). For example, if the encoder line density is 10,000 cycles per inch, the maximum speed is 300 inches/second. If higher encoder frequency is required, please consult the factory.

The standard encoder voltage level is TTL (0-5v), however, voltage levels up to 12 Volts are acceptable. (If using differential signals, 12 Volts can be input directly to the DMC-40x0. Single-ended 12 Volt signals require a bias voltage input to the complementary inputs).

The DMC-40x0 can accept analog feedback (+/-10v) instead of an encoder for any axis. For more information see the command AF in the command reference.

To interface with other types of position sensors such as absolute encoders, Galil can customize the controller and command set. Please contact Galil to talk to one of our applications engineers about your particular system requirements.

## Watch Dog Timer

The DMC-40x0 provides an internal watch dog timer which checks for proper microprocessor operation. The timer toggles the Amplifier Enable Output (AMPEN) which can be used to switch the amplifiers off in the event of a serious DMC-40x0 failure. The AMPEN output is normally high. During power-up and if the microprocessor ceases to function properly, the AMPEN output will go low. The error light will also turn on at this stage. A reset is required to restore the DMC-40x0 to normal operation. Consult the factory for a Return Materials Authorization (RMA) Number if your DMC-40x0 is damaged.

# Chapter 2 Getting Started

## DMC-4040 Layout

The following layouts assume either an ICM-42000(I000) or ICM-42100(I100) interconnect modules are installed. For layouts of systems with ICM-42200's(I200) installed please contact Galil. Overall dimensions and footprint are identical, the only differences are in connector type and location.

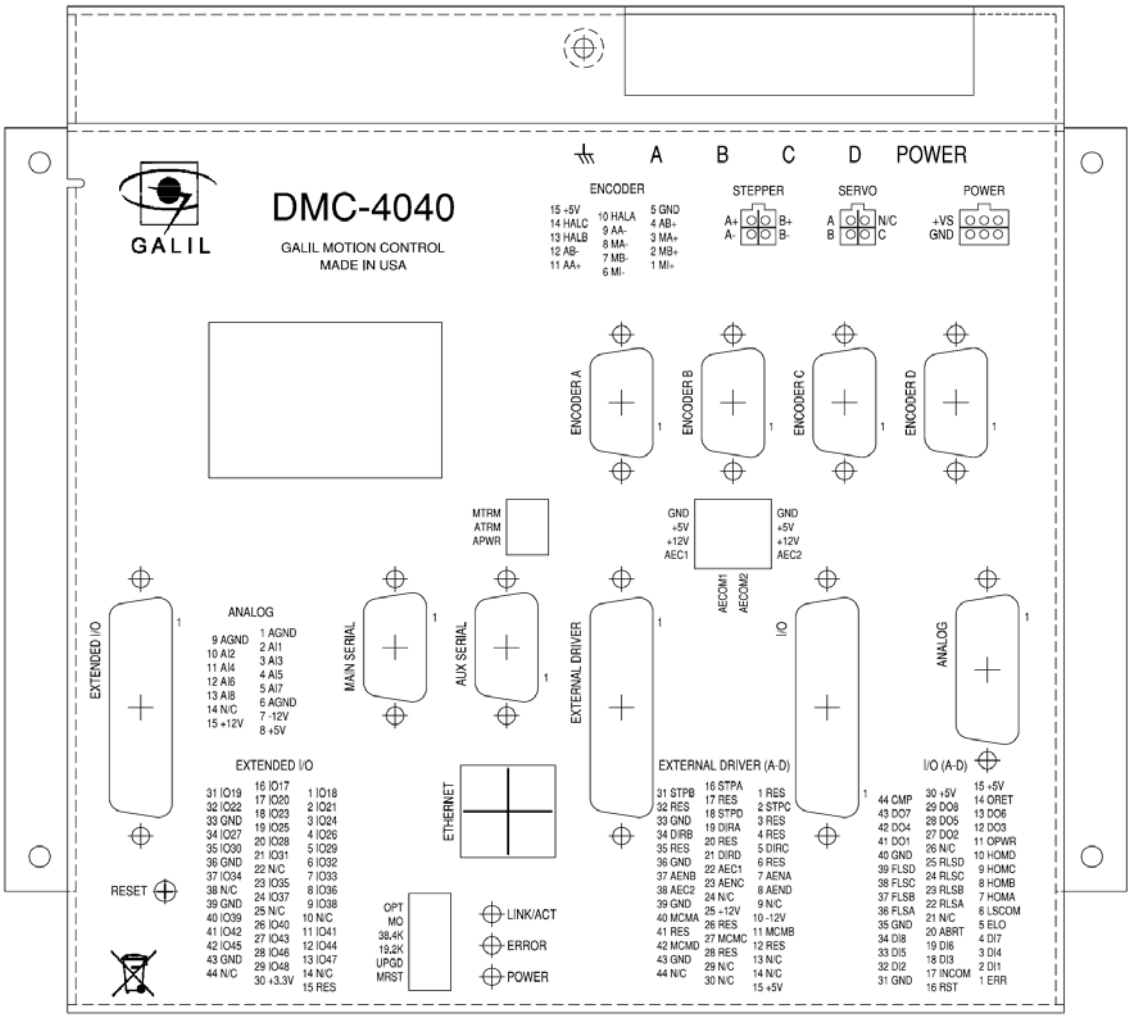



*Figure 2-1 - Outline of the of the DMC-4040*

# DMC-4080 Layout

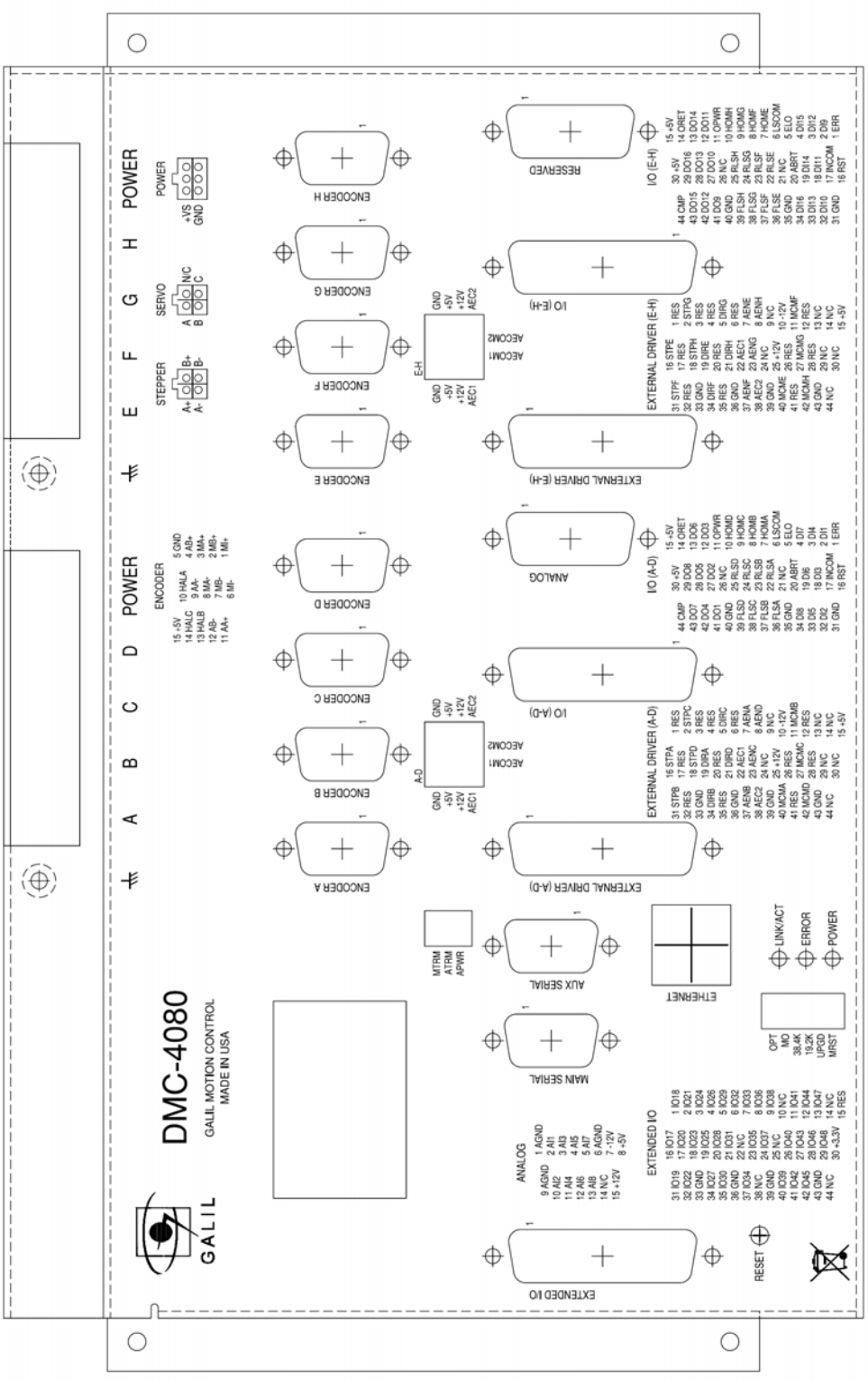



*Figure 2-2 - Outline of the of the DMC-4080*

# DMC-40x0 Power Connections

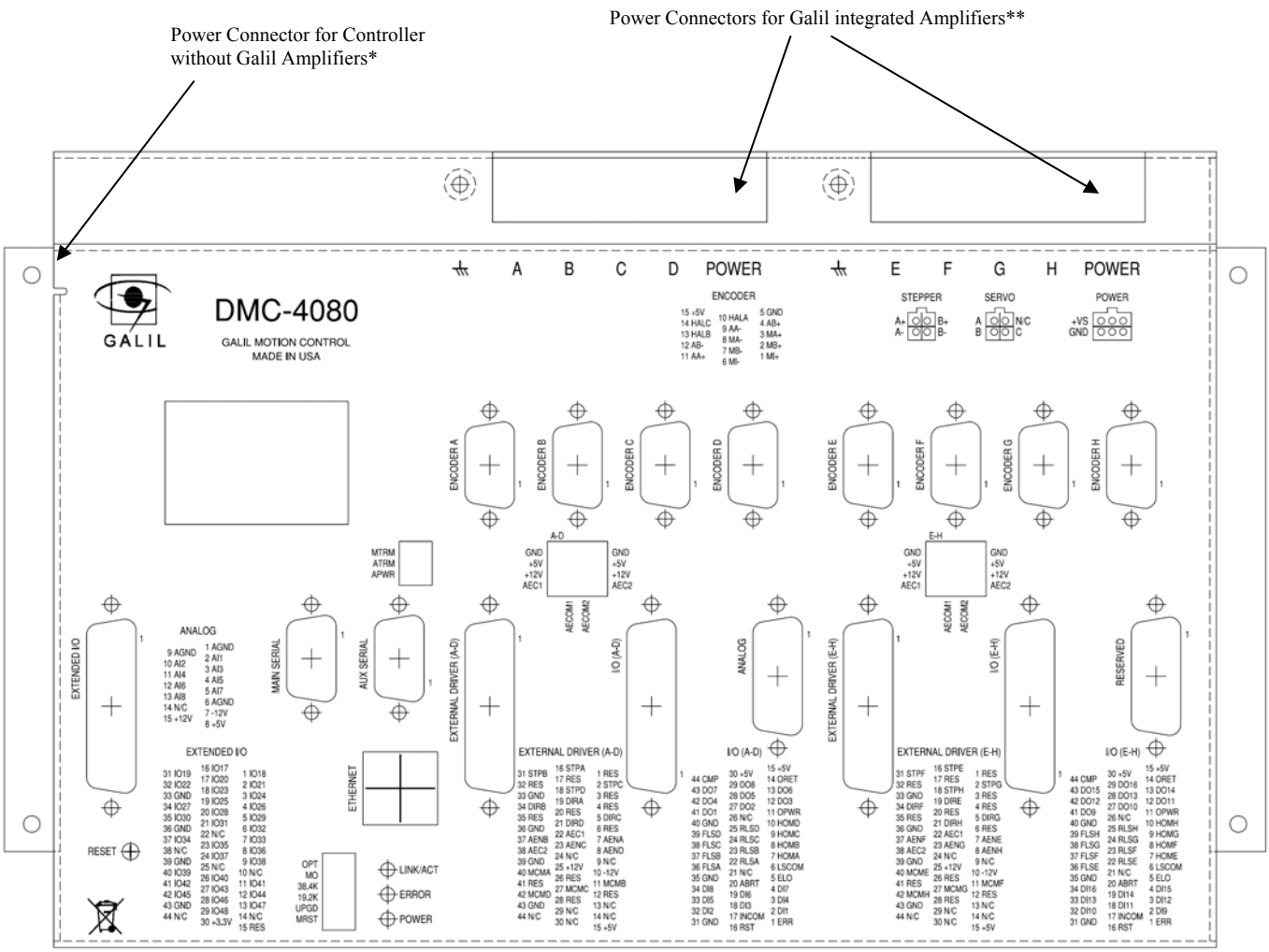



*Figure 2-3 – Connector locations for the DMC-40x0*

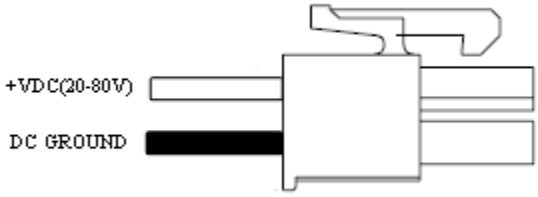



*Figure 2-4 – Power Connector used when controller is ordered without Galil Amplifiers*

**Also used for powering the controller when using the (Linear Amplifier).*

***See Power connector information for specific amplifiers in the Integrated Amplifiers and Drivers section of the Appendices.*

For more information on Connectors (mfg PN's and diagrams) see the Power Connector Section in the Appendix.

# DMC-4040 Dimensions

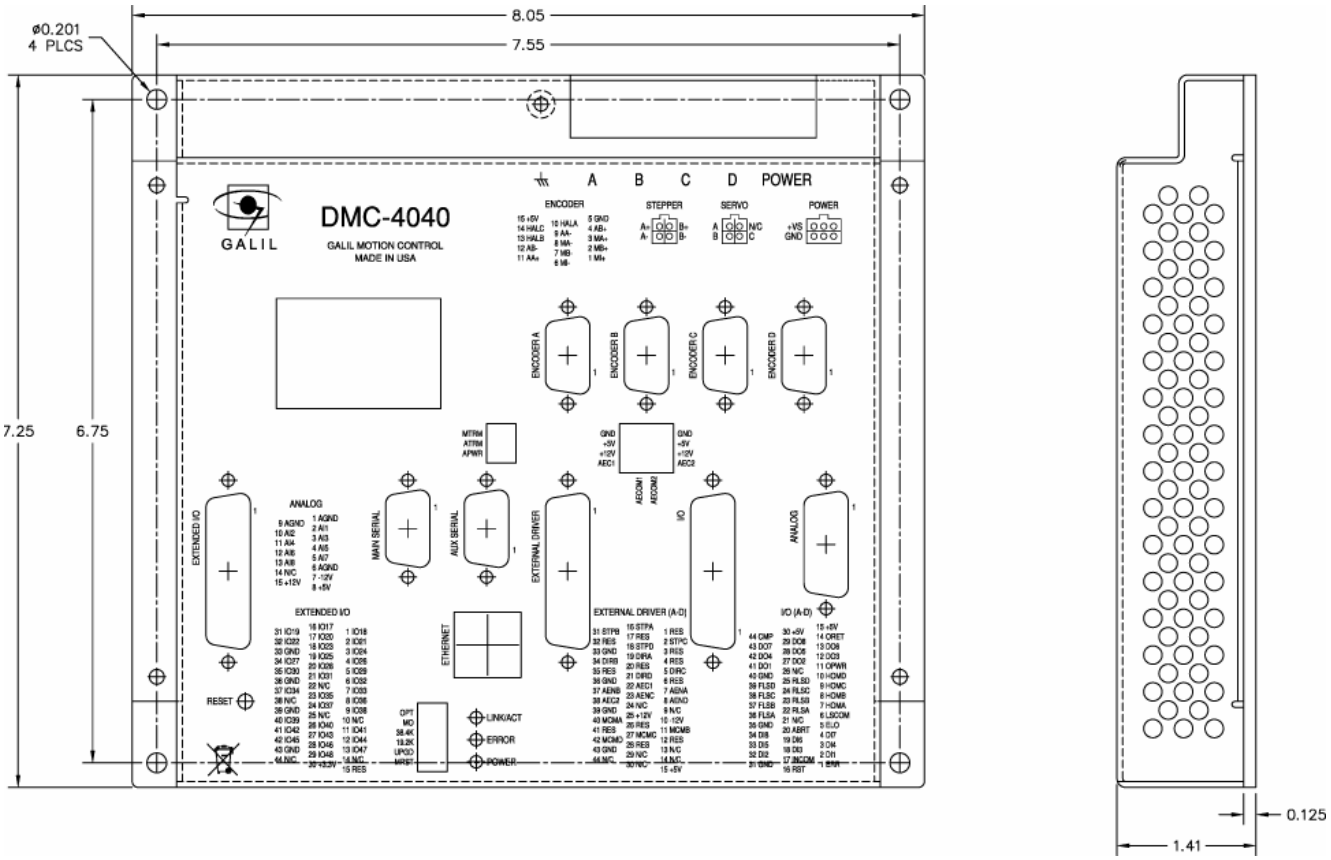



*Figure 2-5 – Dimensions of DMC-4040*

# DMC-4080 Dimensions

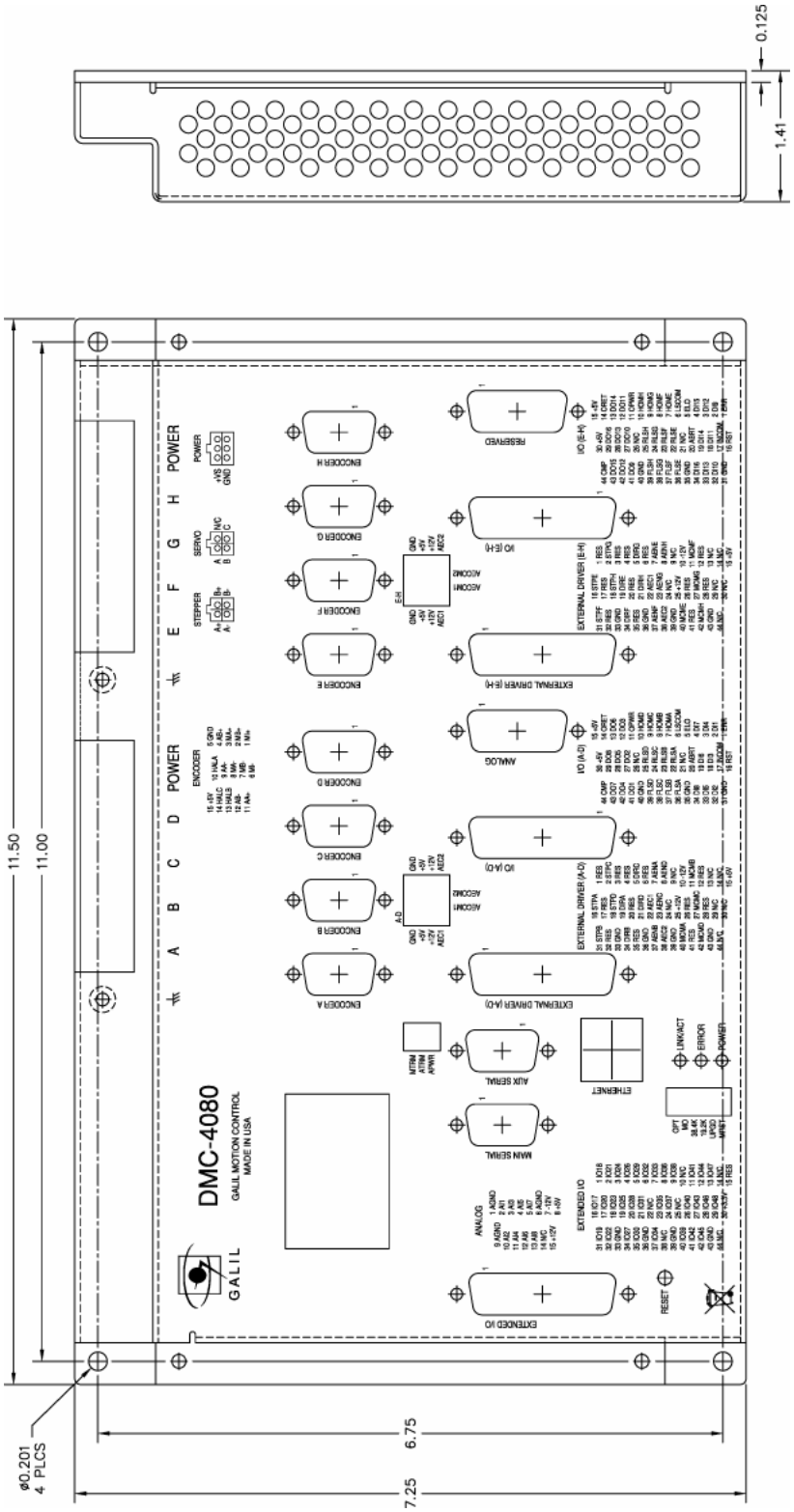



*Figure 2-6 Dimensions of DMC-4080*

# Elements You Need

For a complete system, Galil recommends the following elements:

1. DMC-4010, 4020, 4030, or DMC-4040 Motion Controller

   or

   DMC-4050, 4060, 4070 or DMC-4080
2. Motor Amplifiers (Integrated when using Galil amplifiers and drivers)
3. Power Supply for Amplifiers and controller
4. Brush or Brushless Servo motors with Optical Encoders or stepper motors.
   a. Cables for connecting to the DMC-40x0's integrated ICM's.
5. PC (Personal Computer - RS232 or Ethernet for DMC-40x0)
6. GalilTools, or GalilTools-Lite Software package

GalilTools is highly recommended for first time users of the DMC-40x0. It provides step-by-step instructions for system connection, tuning and analysis.

# Installing the DMC-40x0

Installation of a complete, operational DMC-40x0 system consists of 9 steps.

**Step 1.** Determine overall motor configuration.

**Step 2.** Install Jumpers on the DMC-40x0.

**Step 3.** Install the communications software.

**Step 4.** Connect DC power to controller.

**Step 5.** Establish communications with the Galil Communication Software.

**Step 6.** Determine the Axes to be used for sinusoidal commutation.

**Step 7.** Make connections to amplifier and encoder.

**Step 8a.** Connect standard servo motors.

**Step 8b.** Connect sinusoidal commutation motors

**Step 8c**. Connect step motors.

**Step 9**. Tune the servo system

## Step 1. Determine Overall Motor Configuration

Before setting up the motion control system, the user must determine the desired motor configuration. The DMC-40x0 can control any combination of standard servo motors, sinusoidally commutated brushless motors, and stepper motors. Other types of actuators, such as hydraulics can also be controlled, please consult Galil.

The following configuration information is necessary to determine the proper motor configuration:

### Standard Servo Motor Operation:

Unless ordered with stepper motor drivers or in a non-standard configuration, the DMC-40x0 has been setup by the factory for standard servo motor operation providing an analog command signal of +/- 10V. No hardware or software configuration is required for standard servo motor operation.

### Sinusoidal Commutation:

Sinusoidal commutation is configured through a single software command, BA. This configuration causes the controller to reconfigure the number of available control axes.

Each sinusoidally commutated motor requires two DACs. In standard servo operation, the DMC-40x0 has one DAC per axis. In order to have the additional DAC for sinusoidal commutation, the controller must be designated as having one additional axis for each sinusoidal commutation axis. For example, to control two standard servo axes and one axis of sinusoidal commutation, the controller will require a total of four DACs and the controller must be a DMC-4040.

Sinusoidal commutation is configured with the command, BA. For example, BAA sets the A axis to be sinusoidally commutated. The second DAC for the sinusoidal signal will be the highest available DAC on the controller. For example: Using a DMC-4040, the command BAA will configure the A axis to be the main sinusoidal signal and the 'D' axis to be the second sinusoidal signal.

The BA command also reconfigures the controller to indicate that the controller has one less axis of 'standard' control for each axis of sinusoidal commutation. For example, if the command BAA is given to a DMC-4040 controller, the controller will be re-configured to a DMC-4030 controller. By definition, a DMC-4030 controls 3 axes: A,B and C. The 'D' axis is no longer available since the output DAC is being used for sinusoidal commutation.

Further instruction for sinusoidal commutation connections are discussed in Step 6.

### Stepper Motor Operation

To configure the DMC-40x0 for stepper motor operation, the controller requires that the command, MT, must be given. Further instruction for stepper motor connections are discussed in Step 8c.

## Step 2. Install Jumpers on the DMC-40x0

### Master Reset and Upgrade Jumpers

JP1 on the main board contains two jumpers, MRST and UPGRD. The MRST jumper is the Master Reset jumper. When MRST is connected, the controller will perform a master reset upon PC power up or upon the reset input going low. Whenever the controller has a master reset, all programs, arrays, variables, and motion control parameters stored in EEPROM will be ERASED.

The UPGRD jumper enables the user to unconditionally update the controller's firmware. This jumper is not necessary for firmware updates when the controller is operating normally, but may be necessary in cases of corrupted EEPROM. EEPROM corruption should never occur, however, it is possible if there is a power fault during a firmware update. If EEPROM corruption occurs, your controller may not operate properly. In this case, install the UPGRD Jumper and use the update firmware function on the Galil Terminal to re-load the system firmware.

### Motor Off Jumpers

The state of the motor upon power up may be selected with the placement of a hardware jumper on the controller. With a jumper installed at the MO location, the controller will be powered up in the "motor off" state. The SH command will need to be issued in order for the motor to be enabled. With no jumper installed, the controller will immediately enable the motor upon power up. The MO command will need to be issued to turn the motor off, unless an error occurs that will turn the motors off. The MO jumper is located on JP1, the same block as the Master Reset and Upgrade jumpers.

### Communications Jumpers for DMC-40x0

The baud rate for RS232 communication can be set with jumpers found on JP1 of the communication board (same set of jumpers where MO, MRST and UPGD can be found). To set the baud rate to the desired value, see Table 2-below.

| 19.2 | 38.4 | BAUD RATE |
|---|---|---|
| ON | ON | 9600 |
| ON | OFF | 19200 |
| OFF | ON | 38400 |
| OFF | OFF | 115200 |

*Table 2-1 : Baud Rate Jumper Settings*

Other serial communication protocols, such as RS-485, can be implemented as a special - consult Galil.

## Step 3. Install the Communications Software

After applying power to the computer, you should install the Galil software that enables communication between the controller and PC.

### Using Windows XP (32 & 64 bit):

Install the Galil Software Products CD-ROM into your CD drive. A Galil .htm page should automatically appear with links to the software products. Select the correct version of GalilTools software for your particular operating system and click "Install…" Follow the installation procedure as outlined.

The most recent copy of the GalilTools software can be downloaded from the Galil website.

http://www.galilmc.com/products/software/galiltools.html

All other Galil software is also available for download at the Galil software downloads page.

http://www.galilmc.com/support/download.html

### Using Linux (32 & 64 bit):

The GalilTools software package is fully compatible with a number of Linux distributions. See the GalilTools webpage and user manual for downloads and installation instructions.

http://www.galilmc.com/products/software/galiltools.html

## Step 4. Connect 20-80VDC Power to the Controller

If the controller was ordered with Galil Amplifiers or Drivers, then power to the controller will be supplied through those power connectors. Otherwise the power will come through the connector on the side of the controller. See DMC-40x0 Power Connections.

**WARNING: Dangerous voltages, current, temperatures and energy levels exist in this product and the associated amplifiers and servo motor(s). Extreme caution should be exercised in the application of this equipment. Only qualified individuals should attempt to install, set up and operate this equipment. Never open the controller box when DC power is applied to it.**

The green power light indicator should go on when power is applied.

## Step 5. Establish Communications with Galil Software

### Communicating through an Ethernet connection

The DMC-40x0 motion controller is equipped with DHCP. If the controller is connected to a DHCP enabled network, an IP address will automatically be assigned to the controller. See Ethernet Configuration in Chapter 4 for more information.

### Using GalilTools Software for Windows

Registering controllers in the Windows registry is no longer required when using the GalilTools software package. A simple connection dialog box appears when the software is opened that shows all available controllers.

Any available controllers with assigned IP addresses can be found under the 'Available' tab in the Connections Dialog Box. If the controller is not connected to a DHCP enabled network, or the DH command is set to 0, and the controller has not been assigned an IP address, the controller can be found under the 'No IP Address' tab.

For more information on establishing communication to the controller via the GalilTools software, see the GalilTools user manual.

http://www.galilmc.com/support/manuals/galiltools/index.html

### Using DMC-SmartTerminal or WSDK Software for Windows

**NOTE:** For new applications, Galil recommends using the GalilTools software package.

The controller must be registered in the Windows registry for the host computer to communicate with it. The registry may be accessed via Galil software, such as WSDK or GALIL Smart Terminal.

A dedicated network card with a static IP address is recommended. To set your NIC card to a static IP, go to the Control Panel → Network Connections → Local Area Connection → Properties → TCP/IP and choose "use the

following IP address". If a "Dynamic" IP address is used, make sure there is a DHCP Server on your network or you will encounter an error.

Use the **"New Controller"** button to add a new entry in the registry or alternatively click on the "**Find Ethernet Controller"** to have the software search for controllers connected to the network. When adding a new controller, choose DMC-40x0 as the controller type. Enter the IP address obtained from your system administrator. Select the button corresponding to the UDP or TCP protocol in which you wish to communicate with the controller. If the IP address has not been already assigned to the controller, click on **ASSIGN IP ADDRESS.**

**ASSIGN IP ADDRESS** will check the controllers that are linked to the network to see which ones do not have an IP address. The program will then ask you whether you would like to assign the IP address you entered to the controller with the specified serial number. Click on **YES** to assign it, **NO** to move to next controller, or **CANCEL** to not save the changes. If there are no controllers on the network that do not have an IP address assigned, the program will state this.

When done registering, click on **OK**. If you do not wish to save the changes, click on **CANCEL**.

Once the controller has been registered, select the correct controller from the list and click on **OK**. If the software successfully established communications with the controller, the registry entry will be displayed at the bottom of the screen in the Status window.

**NOTE:** The controller must be registered via an Ethernet connection.

### Communicating through the Main Serial Communications Port

Connect the DMC-40x0 MAIN serial port to your computer via the Galil CABLE-9PIN-D (RS-232 Cable). This is a straight through serial cable – **NOT** a NULL modem.

### Using GalilTools Software for Windows

Registering controllers in the Windows registry is no longer required when using the GalilTools software package. A simple connection dialog box appears when the software is opened that shows all available controllers.

The serial ports are listed as COMn 'communication speed'. (ex COM1 115200). The default serial communication speed on the DMC-40x0 is 115200Bps.

For more information on establishing communication to the controller via the GalilTools software, see the GalilTools user manual.

http://www.galilmc.com/support/manuals/galiltools/index.html

### Using DMC-SmartTerminal or WSDK Software for Windows

**NOTE:** For new applications, Galil recommends using the GalilTools software package.

In order for the windows software to communicate with a Galil controller, the controller must be registered in the Windows Registry. To register a controller, you must specify the model of the controller, the communication parameters, and other information. The registry is accessed through the Galil software under the "File" menu in WSDK or under the "Tools" menu in the Galil Smart Terminal.

Use the **"New Controller"** button to add a new entry to the Registry. You will need to supply the Galil Controller model (eg: DMC-40x0). Pressing the down arrow to the right of this field will reveal a menu of valid controller types. You then need to choose serial or Ethernet connection. The registry information will show a default Comm. Port of 1 and a default Comm. Speed of 115200 appears. This information can be changed as necessary to reflect the computers Comm. Port and the baud rate set by the jumpers found on the communications board. The registry entry also displays timeout and delay information. These are advanced parameters which should only be modified by advanced users (see software documentation for more information).

Once you have set the appropriate Registry information for your controller, Select OK and close the registry window. You will now be able to communicate with the controller.

To establish communication to the controller, open up the Terminal and hit the "Enter" key. You should receive a colon prompt. Communicating with the controller is described in later sections.

If you are not properly communicating with the controller, the program will pause for 3-15 seconds and an error message will be displayed. In this case, there is most likely an incorrect setting of the serial communications port or the serial cable is not connected properly. The user must ensure that the correct communication port and baud rate are specified when attempting to communicate with the controller. Please note that the serial port on the controller must be set for handshake mode for proper communication with Galil software. The user must also insure that a "straight-through" serial cable is being used (**NOT** a Null Modem cable), see appendix for pin-out of serial cable.

### Using Non-Galil Communication Software

The DMC-40x0 main serial port is configured as DATASET. Your computer or terminal must be configured as a DATATERM for full duplex, no parity, 8 data bits, one start bit and one stop bit.

Check to insure that the baud rate jumpers have been set to the desired baud rate as described above.

Your computer needs to be configured as a "dumb" terminal which sends ASCII characters as they are typed to the DMC-40x0.

### Sending Test Commands to the Terminal:

After you connect your terminal, press <return> or the <enter> key on your keyboard. In response to carriage return <return>, the controller responds with a colon, :

Now type

```
TPA <return>
```

This command directs the controller to return the current position of the A axis. The controller should respond with a number such as

```
:0
```

## Step 6. Determine the Axes to be Used for Sinusoidal Commutation

* This step is only required when the controller will be used to control a brushless motor(s) with sinusoidal commutation.

The command, BA is used to select the axes of sinusoidal commutation. For example, BAAC sets A and C as axes with sinusoidal commutation.

### Notes on Configuring Sinusoidal Commutation:

The command, BA, reconfigures the controller such that it has one less axis of 'standard' control for each axis of sinusoidal commutation. For example, if the command BAA is given to a DMC-4040 controller, the controller will be re-configured to be a DMC-4030 controller. In this case the highest axis is no longer available except to be used for the 2nd phase of the sinusoidal commutation. Note that the highest axis on a controller can never be configured for sinusoidal commutation.

The DAC associated with the selected axis represents the first phase. The second phase uses the highest available DAC. When more than one axis is configured for sinusoidal commutation, the controller will assign the second phases to the DACs which have been made available through the axes reconfiguration. The highest sinusoidal commutation axis will be assigned to the highest available DAC and the lowest sinusoidal commutation axis will be assigned to the lowest available DAC. Note that the lowest axis is the A axis and the highest axis is the highest available axis for which the controller has been configured.

### Example: Sinusoidal Commutation Configuration using a DMC-4070

```
BAAC
```

This command causes the controller to be reconfigured as a DMC-4050 controller. The A and C axes are configured for sinusoidal commutation. The first phase of the A axis will be the motor command A signal. The second phase of the A axis will be F signal. The first phase of the C axis will be the motor command C signal. The second phase of the C axis will be the motor command G signal.

## Step 7. Make Connections to Amplifier and Encoder.

If the system is run solely by Galil's integrated amplifiers or drivers, skip this section, the amplifier is already connected to the controller.

Once you have established communications between the software and the DMC-40x0, you are ready to connect the rest of the motion control system. The motion control system typically consists of the controller with interconnect module, an amplifier for each axis of motion, and a motor to transform the current from the amplifier into torque for motion.

System connection procedures will depend on system components and motor types. Any combination of motor types can be used with the DMC-40x0. There can also be a combination of axes running from Galil integrated amplifiers and drivers and external amplifiers or drivers. If sinusoidal commutation is to be used, special attention must be paid to the reconfiguration of axes (see above section for more information).

### Connecting to External Amplifiers

Here are the first steps for connecting a motion control system:

**Step A.** Connect the motor to the amplifier *with no connection to the controller*. Consult the amplifier documentation for instructions regarding proper connections. Connect and turn-on the amplifier power supply. If the amplifiers are operating properly, the motor should stand still even when the amplifiers are powered up.

**Step B.** Connect the amplifier enable signal.

Before making any connections from the amplifier to the controller, you need to verify that the ground level of the amplifier is either floating or at the same potential as earth.

**WARNING: When the amplifier ground is not isolated from the power line or when it has a different potential than that of the computer ground, serious damage may result to the computer controller and amplifier.**

If you are not sure about the potential of the ground levels, connect the two ground signals (amplifier ground and earth) by a 10 kΩ resistor and measure the voltage across the resistor. Only if the voltage is zero, connect the two ground signals directly.

The amplifier enable signal is used by the controller to disable the motor. When configured with the ICM-42000 or ICM-42100, this signal is labeled AENA for the A axis and is found on the 15 pin Dsub connector associated with the A axis (if configured with the ICM-42200 the AENA signal is located on the 26 pin Dsub associated with the A axis). Note that many amplifiers designate this signal as the INHIBIT signal. Use the command, MO, to disable the motor amplifiers - check to insure that the motor amplifiers have been disabled (often this is indicated by an LED on the amplifier).

This signal changes under the following conditions: the watchdog timer activates, the motor-off command, MO, is given, or the OE3 command (Enable Off-On-Error) is given and the position error exceeds the error limit. AMPEN can be used to disable the amplifier for these conditions.

The AMPEN signal from the DMC-40x0 is shipped as a default of 5V active high or high amp enable. In other words, the AMPEN signal will be high when the controller expects the amplifier to be enabled.

**If your amplifier requires a different configuration it is highly recommended that the DMC-40x0 is ordered with the desired configuration.** See the DMC-40x0 ordering information in the catalog (http://www.galilmc.com/catalog/cat40x0.pdf) or contact Galil for more information on ordering different configurations. If the amplifier enable needs to be changed, see the ICM-42000 and ICM-42100 Amplifier Enable Circuit section in Chapter 3 Connecting Hardware.

**4080**

When ordered with ICM-42000's or ICM-42100's the AEN signal is configurable for axes 1-4 and axes 5-8. Ex – axes 1-4 could be ordered as 5V high amp enable, and axes 5-8 could be ordered as 12V low amp enable. When ordered with ICM-42200's each axis is individually configurable.

**Step C.** Connect the encoders

For stepper motor operation, an encoder is optional.

For servo motor operation, if you have a preferred definition of the forward and reverse directions, make sure that the encoder wiring is consistent with that definition.

The DMC-40x0 accepts single-ended or differential encoder feedback with or without an index pulse. The encoder signals are wired to that axis associated 15pin DSub connector found on top of the controller. The signal leads are labeled MA+ (channel A), MB+ (channel B), and MI+. For differential encoders, the complement signals are labeled MA-, MB-, and MI-. For complete pin-out information see Connectors for ICM-42000 Interconnect Board in the Appendices.

**NOTE:** When using pulse and direction encoders, the pulse signal is connected to CHA and the direction signal is connected to CHB. The controller must be configured for pulse and direction with the command CE. See the command summary for further information on the command CE.

**Step D.** Verify proper encoder operation.

Start with the A encoder first. Once it is connected, turn the motor shaft and interrogate the position with the instruction TPA <return>. The controller response will vary as the motor is turned.

At this point, if TPA does not vary with encoder rotation, there are three possibilities:

1. The encoder connections are incorrect - check the wiring as necessary.
2. The encoder has failed - using an oscilloscope, observe the encoder signals. Verify that both channels A and B have a peak magnitude between 5 and 12 volts. Note that if only one encoder channel fails, the position reporting varies by one count only. If the encoder failed, replace the encoder. If you cannot observe the encoder signals, try a different encoder.
3. There is a hardware failure in the controller - connect the same encoder to a different axis. If the problem disappears, you may have a hardware failure. Consult the factory for help.

**Step E.** Connect Hall Sensors if available.

Hall sensors are only used with sinusoidal commutation and are not necessary for proper operation. The use of Hall sensors allows the controller to automatically estimate the commutation phase upon reset and also provides the controller the ability to set a more precise commutation phase. Without Hall sensors, the commutation phase must be determined manually.

The Hall Effect sensors are connected to the digital inputs of the controller. These inputs can be used with the general use inputs (bits 1-8), the auxiliary encoder inputs (bits 81-96), or the extended I/O inputs of the DMC-40x0 controller (bits 17-80).

**NOTE:** The general use inputs are optoisolated and require a voltage connection at the INCOM point - for more information regarding the digital inputs, see Chapter 3, Connecting Hardware.

Each set of sensors must use inputs that are in consecutive order. The input lines are specified with the command, BI. For example, if the Hall sensors of the C axis are connected to inputs 6, 7 and 8, use the instruction:

```
BI ,, 6   or
BIC = 6
```

## Step 8a. Connect Standard Servo Motors

The following discussion applies to connecting the DMC-40x0 controller to standard servo motors:

The motor and the amplifier may be configured in the torque or the velocity mode. In the torque mode, the amplifier gain should be such that a 10 volt signal generates the maximum required current. In the velocity mode, a command signal of 10 volts should run the motor at the maximum required speed. For Galil amplifiers, see Integrated Amplifiers and Drivers.

**Step A.** Check the Polarity of the Feedback Loop

*It is assumed that the motor and amplifier are connected together and that the encoder is operating correct (Step 7).* Before connecting the motor amplifiers to the controller, read the following discussion on setting Error Limits and Torque Limits. Note that this discussion only uses the A axis as an examples.

**Step B.** Set the Error Limit as a Safety Precaution

Usually, there is uncertainty about the correct polarity of the feedback. The wrong polarity causes the motor to run away from the starting position. Using a terminal program, such as Galil Tools, the following parameters can be given to avoid system damage:

Input the commands:

`ER 2000 <return>` Sets error limit on the A axis to be 2000 encoder counts

`OE 1 <return>` Disables A axis amplifier when excess position error exists

If the motor runs away and creates a position error of 2000 counts, the motor amplifier will be disabled.

**NOTE:** This function requires the AMPEN signal to be connected from the controller to the amplifier.

**Step C.** Set Torque Limit as a Safety Precaution

To limit the maximum voltage signal to your amplifier, the DMC-40x0 controller has a torque limit command, TL. This command sets the maximum voltage output of the controller and can be used to avoid excessive torque or speed when initially setting up a servo system.

When operating an amplifier in torque mode, the voltage output of the controller will be directly related to the torque output of the motor. The user is responsible for determining this relationship using the documentation of the motor and amplifier. The torque limit can be set to a value that will limit the motors output torque.

When operating an amplifier in velocity or voltage mode, the voltage output of the controller will be directly related to the velocity of the motor. The user is responsible for determining this relationship using the documentation of the motor and amplifier. The torque limit can be set to a value that will limit the speed of the motor.

For example, the following command will limit the output of the controller to 1 volt on the X axis:

`TL 1 <return>`

**NOTE:** Once the correct polarity of the feedback loop has been determined, the torque limit should, in general, be increased to the default value of 9.99. The servo will not operate properly if the torque limit is below the normal operating range. See description of TL in the command reference.

**Step D.** Connect the Motor

Once the parameters have been set, connect the analog motor command signal (MCM*n* where *n* is A-H) to the amplifier input.

To test the polarity of the feedback, command a move with the instruction:

`PR 1000 <CR>` Position relative 1000 counts

`BGA <CR>` Begin motion on A axis

When the polarity of the feedback is wrong, the motor will attempt to run away. The controller should disable the motor when the position error exceeds 2000 counts. If the motor runs away, the polarity of the loop must be inverted.

### Inverting the Loop Polarity

When the polarity of the feedback is incorrect, the user must invert the loop polarity and this may be accomplished by several methods. If you are driving a brush-type DC motor, the simplest way is to invert the two motor wires (typically red and black). For example, switch the M1 and M2 connections going from your amplifier to the motor. When driving a brushless motor, the polarity reversal may be done with the encoder. If you are using a single-ended encoder, interchange the signal MA+ and MB+. If, on the other hand, you are using a differential encoder, interchange only MA+ and MA-. The loop polarity and encoder polarity can also be affected through software with the MT, and CE commands. For more details on the MT command or the CE command, see the Command Reference section.

To Invert Polarity using Hall-Commutated brushless motors, invert motor phases B & C, exchange Hall A with Hall B, and invert encoder polarity as described above.

Sometimes the feedback polarity is correct (the motor does not attempt to run away) but the direction of motion is reversed with respect to the commanded motion. If this is the case, reverse the motor leads AND the encoder signals.

If the motor moves in the required direction but stops short of the target, it is most likely due to insufficient torque output from the motor command signal MCM*n*. This can be alleviated by reducing system friction on the motors. The instruction:

```
TTA  <return>   Tell torque on A
```

reports the level of the output signal. It will show a non-zero value that is below the friction level.

Once you have established that you have closed the loop with the correct polarity, you can move on to the compensation phase (servo system tuning) to adjust the PID filter parameters, KP, KD and KI. It is necessary to accurately tune your servo system to ensure fidelity of position and minimize motion oscillation as described in the next section.

## Step 8b. Connect Sinusoidal Commutation Motors

When using sinusoidal commutation, the parameters for the commutation must be determined and saved in the controller's non-volatile memory. The setup for sinusoidal commutation is different when using Hall Sensors. Each step which is affected by Hall Sensor Operation is divided into two parts, part 1 and part 2. After connecting sinusoidal commutation motors, the servos must be tuned as described in Step 9.

**Step A.** Disable the motor amplifier

Use the command, MO, to disable the motor amplifiers. For example, MOA will turn the A axis motor off.

**Step B.** Connect the motor amplifier to the controller.

The sinusoidal commutation amplifier requires 2 signals, usually denoted as Phase A & Phase B. These inputs should be connected to the two sinusoidal signals generated by the controller. The first signal is the axis specified with the command, BA (Step 6). The second signal is associated with the highest analog command signal available on the controller - note that this axis was made unavailable for standard servo operation by the command BA.

When more than one axis is configured for sinusoidal commutation, the controller will assign the second phase to the command output which has been made available through the axes reconfiguration. The 2nd phase of the highest sinusoidal commutation axis will be the highest command output and the 2nd phase of the lowest sinusoidal commutation axis will be the lowest command output.

It is not necessary to be concerned with cross-wiring the 1st and 2nd signals. If this wiring is incorrect, the setup procedure will alert the user (Step D).

### Example: Sinusoidal Commutation Configuration using a DMC-4070

```
BAAC
```

This command causes the controller to be reconfigured as a DMC-4050 controller. The A and C axes are configured for sinusoidal commutation. The first phase of the A axis will be the motor command A signal. The second phase of the A axis will be the motor command F signal. The first phase of the C axis will be the motor command C signal. The second phase of the C axis will be the motor command G signal.

**Step C.** Specify the Size of the Magnetic Cycle.

Use the command, BM, to specify the size of the brushless motors magnetic cycle in encoder counts. For example, if the X axis is a linear motor where the magnetic cycle length is 62 mm, and the encoder resolution is 1 micron, the cycle equals 62,000 counts. This can be commanded with the command:

```
BM 62000
```

On the other hand, if the C axis is a rotary motor with 4000 counts per revolution and 3 magnetic cycles per revolution (three pole pairs) the command is:

```
BM,, 1333.333
```

**Step D - part 1 (Systems with or without Hall Sensors).** Test the Polarity of the DACs

Use the brushless motor setup command, BS, to test the polarity of the output DACs. This command applies a certain voltage, V, to each phase for some time T, and checks to see if the motion is in the correct direction.

The user must specify the value for V and T. For example, the command:

```
BSA = 2,700
```

will test the A axis with a voltage of 2 volts, applying it for 700 millisecond for each phase. In response, this test indicates whether the DAC wiring is correct and will indicate an approximate value of BM. If the wiring is correct, the approximate value for BM will agree with the value used in the previous step.

**NOTE:** In order to properly conduct the brushless setup, the motor must be allowed to move a minimum of one magnetic cycle in both directions.

**NOTE:** When using Galil Windows software, the timeout must be set to a minimum of 10 seconds (time-out = 10000) when executing the BS command. This allows the software to retrieve all messages returned from the controller.

**Step D - part 2 (Systems with Hall Sensors Only).** Test the Hall Sensor Configuration.

Since the Hall sensors are connected randomly, it is very likely that they are wired in the incorrect order. The brushless setup command indicates the correct wiring of the Hall sensors. The Hall sensor wires should be re-configured to reflect the results of this test.

The setup command also reports the position offset of the Hall transition point and the zero phase of the motor commutation. The zero transition of the Hall sensors typically occur at 0°, 30° or 90° of the phase commutation. It is necessary to inform the controller about the offset of the Hall sensor and this is done with the instruction, BB.

**Step E.** Save Brushless Motor Configuration

It is very important to save the brushless motor configuration in non-volatile memory. After the motor wiring and setup parameters have been properly configured, the burn command, BN, should be given.

**NOTE:** Without Hall sensors, the controller will not be able to estimate the commutation phase of the brushless motor. In this case, the controller could become unstable until the commutation phase has been set using the BZ command (see next step). It is highly recommended that the motor off command be given before executing the BN command. In this case, the motor will be disabled upon power up or reset and the commutation phase can be set before enabling the motor.

**Step F - part 1 (Systems with or without Hall Sensors).** Set Zero Commutation Phase

When an axis has been defined as sinusoidally commutated, the controller must have an estimate for commutation phase. When Hall sensors are used, the controller automatically estimates this value upon

reset of the controller. If no Hall sensors are used, the controller will not be able to make this estimate and the commutation phase must be set before enabling the motor.

To initialize the commutation without Hall effect sensor use the command, BZ. This function drives the motor to a position where the commutation phase is zero, and sets the phase to zero.

The BZ command is followed by real numbers in the fields corresponding to the driven axes. The number represents the voltage to be applied to the amplifier during the initialization. When the voltage is specified by a positive number, the initialization process ends up in the motor off (MO) state. A negative number causes the process to end in the Servo Here (SH) state.

**WARNING: This command must move the motor to find the zero commutation phase. This movement is instantaneous and will cause the system to jerk. Larger applied voltages will cause more severe motor jerk. The applied voltage will typically be sufficient for proper operation of the BZ command. For systems with significant friction, this voltage may need to be increased and for systems with very small motors, this value should be decreased. For example:**

**BZ –2, 0,1**

**will drive both A and C axes to zero, will apply 2V and 1V respectively to A and C and will end up with A in SH and C in MO.**

**Step F - part 2 (Systems with Hall Sensors Only).** Set Zero Commutation Phase

With Hall sensors, the estimated value of the commutation phase is good to within 30°. This estimate can be used to drive the motor but a more accurate estimate is needed for efficient motor operation. There are 3 possible methods for commutation phase initialization:

**Method 1.** Use the BZ command as described above.

**Method 2.** Drive the motor close to commutation phase of zero and then use BZ command. This method decreases the amount of system jerk by moving the motor close to zero commutation phase before executing the BZ command. The controller makes an estimate for the number of encoder counts between the current position and the position of zero commutation phase. This value is stored in the operand _BZn. Using this operand the controller can be commanded to move the motor. The BZ command is then issued as described above. For example, to initialize the A axis motor upon power or reset, the following commands may be given:

```
SHA           ;Enable A axis motor
PRA=-1*(_BZA) ;Move A motor close to zero commutation phase
BGA           ;Begin motion on A axis
AMA           ;Wait for motion to complete on A axis
BZA=-1        ;Drive motor to commutation phase zero and leave motor on
```

**Method 3.** Use the command, BC. This command uses the Hall transitions to determine the commutation phase. Ideally, the Hall sensor transitions will be separated by exactly 60° and any deviation from 60° will affect the accuracy of this method. If the Hall sensors are accurate, this method is recommended. The BC command monitors the Hall sensors during a move and monitors the Hall sensors for a transition point. When that occurs, the controller computes the commutation phase and sets it. For example, to initialize the A axis motor upon power or reset, the following commands may be given:

```
SHA           ;Enable A axis motor
BCA           ;Enable the brushless calibration command
PRA=50000     ;Command a relative position movement on A axis
BGA           ;Begin motion on A axis.  When the Hall sensors detect a
              ;phase transition, the commutation phase is reset
```

## Step 8c. Connect Step Motors

In Stepper Motor operation, the pulse output signal has a 50% duty cycle. Step motors operate open loop and do not require encoder feedback. When a stepper is used, the auxiliary encoder for the corresponding axis is unavailable for an external connection. If an encoder is used for position feedback, connect the encoder to the main encoder input corresponding to that axis. The commanded position of the stepper can be interrogated with RP or TD. The encoder position can be interrogated with TP.

If encoders are available on the stepper motor, Galil's Stepper Position Maintenance Mode may be used for automatic monitoring and correction of the stepper position. See Stepper Position Maintenance Mode (SPM) in Chapter 6 Programming Motion for more information.

The frequency of the step motor pulses can be smoothed with the filter parameter, KS. The KS parameter has a range between 0.25 and 64, where 64 implies the largest amount of smoothing. *See Command Reference regarding KS.*

The DMC-40x0 profiler commands the step motor amplifier. All DMC-40x0 motion commands apply such as PR, PA, VP, CR and JG. The acceleration, deceleration, slew speed and smoothing are also used. Since step motors run open-loop, the PID filter does not function and the position error is not generated.

To connect step motors with the DMC-40x0 you must follow this procedure – If you have a Galil integrated stepper driver skip Step A, the step and direction lines are already connected to the driver:

**Step A.** Connect step and direction signals from controller to motor amplifier

From the controller to respective signals on your step motor amplifier. (These signals are labeled STPA and DIRA for the A-axis on the EXTERNAL DRIVER (A-D) D-Sub connector top of the controller). Consult the documentation for connecting these signals to your step motor amplifier.

**Step B.** Configure DMC-40x0 for motor type using MT command. You can configure the DMC-40x0 for active high or active low pulses. Use the command MT 2 or 2.5 for active low step motor pulses and MT -2 or -2.5 for active high step motor pulses. *See description of the MT command in the Command Reference.*

## Step 9. Tune the Servo System

Adjusting the tuning parameters is required when using servo motors (standard or sinusoidal commutation). The system compensation provides fast and accurate response and the following section suggests a simple and easy way for compensation. More advanced design methods are available with software design tools from Galil, such as the GalilTools.

The filter has three parameters: the damping, KD; the proportional gain, KP; and the integrator, KI. The parameters should be selected in this order.

To start, set the integrator to zero with the instruction

```
KI 0  <return>      Integrator gain
```

and set the proportional gain to a low value, such as

```
KP 1 <return>       Proportional gain
```

```
KD 100 <return>     Derivative gain
```

For more damping, you can increase KD (maximum is 4095.875). Increase gradually and stop after the motor vibrates. A vibration is noticed by audible sound or by interrogation. If you send the command

```
TE A  <return>      Tell error
```

a few times, and get varying responses, especially with reversing polarity, it indicates system vibration. When this happens, simply reduce KD by about 20%.

Next you need to increase the value of KP gradually (maximum allowed is 1023.875). You can monitor the improvement in the response with the Tell Error instruction

```
KP 10 <return>      Proportion gain
TE A <return>       Tell error
```

As the proportional gain is increased, the error decreases.

Again, the system may vibrate if the gain is too high. In this case, reduce KP by about 20%. Typically, KP should not be greater than KD/4 (only when the amplifier is configured in the current mode).

Finally, to select KI, start with zero value and increase it gradually. The integrator eliminates the position error, resulting in improved accuracy. Therefore, the response to the instruction

```
TE A  <return>
```

becomes zero. As KI is increased, its effect is amplified and it may lead to vibrations. If this occurs, simply reduce KI. Repeat tuning for the B, C and D axes.

*Note:* For a more detailed description of the operation of the PID filter and/or servo system theory, see Chapter 10 Theory of Operation

# Design Examples

Here are a few examples for tuning and using your controller. These examples have remarks next to each command - these remarks must not be included in the actual program.

## Example 1 - System Set-up

This example assigns the system filter parameters, error limits and enables the automatic error shut-off.

| Instruction | Interpretation |
|---|---|
| KP10,10,10,10 | Set gains for a,b,c,d (or A,B,C,D axes) |
| KP*=10 | Alternate method for setting gain on all axes |
| KPA=10 | Method for setting only A (or X) axis gain |
| KPX=10 | Method for setting only X (or A) axis gain |
| KP, 20 | Set B axis gain only |

| Instruction | Interpretation |
|---|---|
| OE 1,1,1,1,1,1,1,1 | Enable automatic Off on Error function for all axes |
| ER*=1000 | Set error limit for all axes to 1000 counts |
| KP10,10,10,10,10,10,10,10 | Set gains for a,b,c,d,e,f,g,and h axes |
| KP*=10 | Alternate method for setting gain on all axes |
| KPA=10 | Alternate method for setting A axis gain |
| KP,,10 | Set C axis gain only |
| KPD=10 | Alternate method for setting D axis gain |
| KPH=10 | Alternate method for setting H axis gain |

## Example 2 - Profiled Move

Rotate the A axis a distance of 10,000 counts at a slew speed of 20,000 counts/sec and an acceleration and deceleration rates of 100,000 counts/s2. In this example, the motor turns and stops:

| Instruction | Interpretation |
|---|---|
| PR1000 | Distance |

```
SP20000                 Speed
DC 100000               Deceleration
AC 100000               Acceleration
BG A                    Start Motion
```

## Example 3 - Multiple Axes

Objective: Move the four axes independently.

| Instruction | Interpretation |
|---|---|
| PR 500,1000,600,-400 | Distances of A,B,C,D |
| SP 10000,12000,20000,10000 | Slew speeds of A,B,C,D |
| AC 10000,10000,10000,10000 | Accelerations of A,B,C,D |
| DC 80000,40000,30000,50000 | Decelerations of A,B,C,D |
| BG AC | Start A and C motion |
| BG BD | Start B and D motion |

## Example 4 - Independent Moves

The motion parameters may be specified independently as illustrated below.

| Instruction | Interpretation |
|---|---|
| PR ,300,-600 | Distances of B and C |
| SP ,2000 | Slew speed of B |
| DC ,80000 | Deceleration of B |
| AC ,100000 | Acceleration of B |
| AC ,,100000 | Acceleration of C |
| DC,,150000 | Deceleration of C |
| BG C | Start C motion |
| BG B | Start B motion |

## Example 5 - Position Interrogation

The position of the four axes may be interrogated with the instruction, TP.

| Instruction | Interpretation |
|---|---|
| TP | Tell position all four axes |
| TP A | Tell position – A axis only |
| TP B | Tell position – B axis only |
| TP C | Tell position – C axis only |
| TP D | Tell position – D axis only |

The position error, which is the difference between the commanded position and the actual position can be interrogated with the instruction TE.

| Instruction | Interpretation |
|---|---|
| TE | Tell error – all axes |
| TE A | Tell error – A axis only |
| TE B | Tell error – B axis only |
| TE C | Tell error – C axis only |
| TE D | Tell error – D axis only |

## Example 6 - Absolute Position

Objective: Command motion by specifying the absolute position.

| Instruction | Interpretation |
|---|---|

```
DP 0,2000           Define the current positions of A,B as 0 and 2000
PA 7000,4000        Sets the desired absolute positions
BG A                Start A motion
BG B                Start B motion
```

After both motions are complete, the A and B axes can be command back to zero:

```
PA 0,0              Move to 0,0
BG AB               Start both motions
```

## Example 7 - Velocity Control

Objective: Drive the A and B motors at specified speeds.

| Instruction | Interpretation |
|---|---|
| JG 10000,-20000 | Set Jog Speeds and Directions |
| AC 100000, 40000 | Set accelerations |
| DC 50000,50000 | Set decelerations |
| BG AB | Start motion |

after a few seconds, command:

```
JG -40000           New A speed and Direction
TV A                Returns A speed
```

and then

```
JG ,20000           New B speed
TV B                Returns B speed
```

These cause velocity changes including direction reversal. The motion can be stopped with the instruction

```
ST                  Stop
```

## Example 8 - Operation Under Torque Limit

The magnitude of the motor command may be limited independently by the instruction TL.

| Instruction | Interpretation |
|---|---|
| TL 0.2 | Set output limit of A axis to 0.2 volts |
| JG 10000 | Set A speed |
| BG A | Start A motion |

In this example, the A motor will probably not move since the output signal will not be sufficient to overcome the friction. If the motion starts, it can be stopped easily by a touch of a finger.

Increase the torque level gradually by instructions such as

| Instruction | Interpretation |
|---|---|
| TL 1.0 | Increase torque limit to 1 volt. |
| TL 9.998 | Increase torque limit to maximum, 9.998 volts. |

The maximum level of 9.998 volts provides the full output torque.

## Example 9 - Interrogation

The values of the parameters may be interrogated. Some examples …

| Instruction | Interpretation |
|---|---|

```
KP?                         Return gain of A axis
KP ,,?                      Return gain of C axis.
KP ?,?,?,?                  Return gains of all axes.
```

Many other parameters such as KI, KD, FA, can also be interrogated. The command reference denotes all commands which can be interrogated.

## Example 10 - Operation in the Buffer Mode

The instructions may be buffered before execution as shown below.

| Instruction | Interpretation |
|---|---|
| PR 600000 | Distance |
| SP 10000 | Speed |
| WT 10000 | Wait 10000 milliseconds before reading the next instruction |
| BG A | Start the motion |

## Example 11 - Using the On-Board Editor

Motion programs may be edited and stored in the controller's on-board memory. When the command, ED is given from the Galil DOS terminal (such as DMCTERM), the controllers editor will be started.

The instruction

```
ED                  Edit mode
```

moves the operation to the editor mode where the program may be written and edited. The editor provides the line number. For example, in response to the first ED command, the first line is zero.

| Line # | Instruction | Interpretation |
|---|---|---|
| 000 | #A | Define label |
| 001 | PR 700 | Distance |
| 002 | SP 2000 | Speed |
| 003 | BGA | Start A motion |
| 004 | EN | End program |

To exit the editor mode, input <cntrl>Q. The program may be executed with the command.

```
XQ #A               Start the program running
```

If the ED command is issued from the Galil Windows terminal software (such as SmartTERM), the software will open a Windows based editor. From this editor a program can be entered, edited, downloaded and uploaded to the controller.

## Example 12 - Motion Programs with Loops

Motion programs may include conditional jumps as shown below.

| Instruction | Interpretation |
|---|---|
| #A | Label |
| DP 0 | Define current position as zero |
| V1=1000 | Set initial value of V1 |
| #LOOP | Label for loop |
| PA V1 | Move A motor V1 counts |
| BG A | Start A motion |
| AM A | After A motion is complete |

```
WT 500                    Wait 500 ms
TP A                      Tell position A
V1=V1+1000                Increase the value of V1
JP #LOOP,V1<10001         Repeat if V1<10001
EN                        End
```

After the above program is entered, quit the Editor Mode, <cntrl>Q. To start the motion, command:

```
XQ #A                     Execute Program #A
```

## Example 13 - Motion Programs with Trippoints

The motion programs may include trippoints as shown below.

| Instruction | Interpretation |
|---|---|
| #B | Label |
| DP 0,0 | Define initial positions |
| PR 30000,60000 | Set targets |
| SP 5000,5000 | Set speeds |
| BGA | Start A motion |
| AD 4000 | Wait until A moved 4000 |
| BGB | Start B motion |
| AP 6000 | Wait until position A=6000 |
| SP 2000,50000 | Change speeds |
| AP ,50000 | Wait until position B=50000 |
| SP ,10000 | Change speed of B |
| EN | End program |

To start the program, command:

```
XQ #B                     Execute Program #B
```

## Example 14 - Control Variables

Objective: To show how control variables may be utilized.

| Instruction | Interpretation |
|---|---|
| #A;DP0 | Label; Define current position as zero |
| PR 4000 | Initial position |
| SP 2000 | Set speed |
| BGA | Move A |
| AMA | Wait until move is complete |
| WT 500 | Wait 500 ms |
| #B | |
| V1 = _TPA | Determine distance to zero |
| PR -V1/2 | Command A move 1/2 the distance |
| BGA | Start A motion |
| AMA | After A moved |
| WT 500 | Wait 500 ms |
| V1= | Report the value of V1 |
| JP #C, V1=0 | Exit if position=0 |
| JP #B | Repeat otherwise |
| #C | Label #C |

```
EN                  End of Program
```

To start the program, command

```
XQ #A               Execute Program #A
```

This program moves A to an initial position of 1000 and returns it to zero on increments of half the distance. Note, _TPA is an internal variable which returns the value of the A position. Internal variables may be created by preceding a DMC-40x0 instruction with an underscore, _.

## Example 15 - Linear Interpolation

Objective: Move A,B,C motors distance of 7000,3000,6000, respectively, along linear trajectory. Namely, motors start and stop together.

| Instruction | Interpretation |
|---|---|
| LM ABC | Specify linear interpolation axes |
| LI 7000,3000,6000 | Relative distances for linear interpolation |
| LE | Linear End |
| VS 6000 | Vector speed |
| VA 20000 | Vector acceleration |
| VD 20000 | Vector deceleration |
| BGS | Start motion |

## Example 16 - Circular Interpolation

Objective: Move the AB axes in circular mode to form the path shown on Fig. 2-8. Note that the vector motion starts at a local position (0,0) which is defined at the beginning of any vector motion sequence. See application programming for further information.

| Instruction | Interpretation |
|---|---|
| VM AB | Select AB axes for circular interpolation |
| VP -4000,0 | Linear segment |
| CR 2000,270,-180 | Circular segment |
| VP 0,4000 | Linear segment |
| CR 2000,90,-180 | Circular segment |
| VS 1000 | Vector speed |
| VA 50000 | Vector acceleration |
| VD 50000 | Vector deceleration |
| VE | End vector sequence |
| BGS | Start motion |

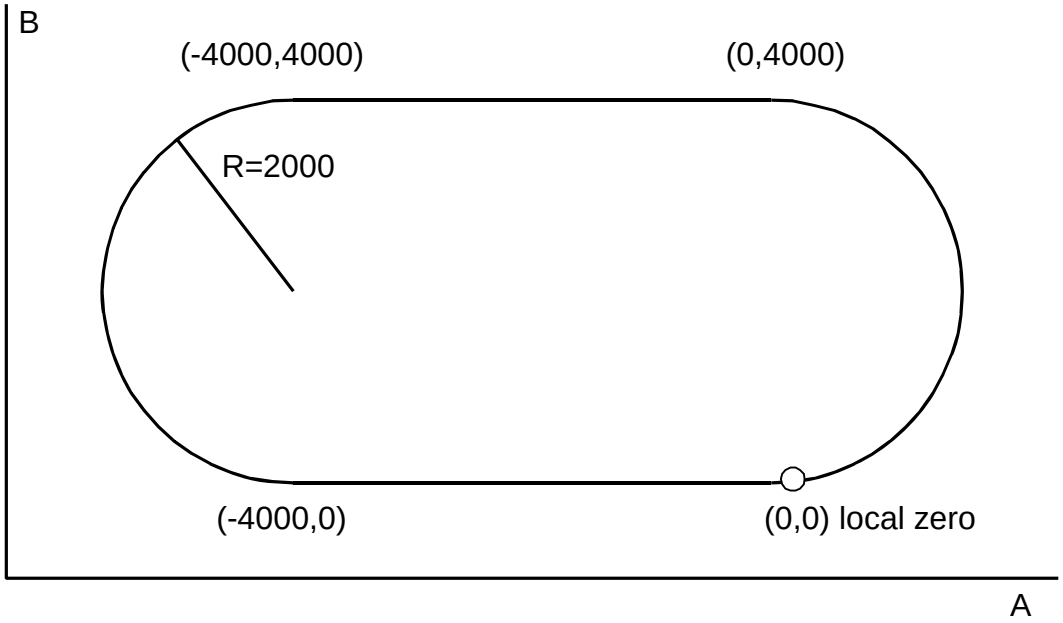



*Figure 2-7 Motion Path for Circular Interpolation Example*

# Chapter 3 Connecting Hardware

## Overview

The DMC-40x0 provides opto-isolated digital inputs for **forward limit**, **reverse limit, home**, and **abort** signals. The controller also has **8 opto-isolated, uncommitted inputs** (for general use) as well as **8 high power opto-isolated outputs** and **8 analog inputs** configured for voltages between +/- 10 volts.

**4080** Controllers with 5 or more axes have an additional 8 opto-isolated inputs and an additional 8 high power opto-isolated outputs.

This chapter describes the inputs and outputs and their proper connection.

## Using Optoisolated Inputs

### Limit Switch Input

The forward limit switch (FLSx) inhibits motion in the forward direction immediately upon activation of the switch. The reverse limit switch (RLSx) inhibits motion in the reverse direction immediately upon activation of the switch. If a limit switch is activated during motion, the controller will make a decelerated stop using the deceleration rate previously set with the SD command. The motor will remain on (in a servo state) after the limit switch has been activated and will hold motor position.

When a forward or reverse limit switch is activated, the current application program that is running in thread zero will be interrupted and the controller will automatically jump to the #LIMSWI subroutine if one exists. This is a subroutine which the user can include in any motion control program and is useful for executing specific instructions upon activation of a limit switch. Automatic Subroutines for Monitoring Conditions are discussed in Chapter 7 Application Programming.

After a limit switch has been activated, further motion in the direction of the limit switch will not be possible until the logic state of the switch returns back to an inactive state. This usually involves physically opening the tripped switch. Any attempt at further motion before the logic state has been reset will result in the following error: "022 - Begin not possible due to limit switch" error.

The operands, _LFx and _LRx, contain the state of the forward and reverse limit switches, respectively (x represents the axis, X, Y, Z, W etc.). The value of the operand is either a '0' or '1' corresponding to the logic state of the limit switch. Using a terminal program, the state of a limit switch can be printed to the screen with the command, MG_LFx or MG_LRx. This prints the value of the limit switch operands for the 'x' axis. The logic state of the limit switches can also be interrogated with the TS command. For more details on TS see the Command Reference.

## Home Switch Input

Homing inputs are designed to provide mechanical reference points for a motion control application. A transition in the state of a Home input alerts the controller that a particular reference point has been reached by a moving part in the motion control system. A reference point can be a point in space or an encoder index pulse.

The Home input detects any transition in the state of the switch and toggles between logic states 0 and 1 at every transition. A transition in the logic state of the Home input will cause the controller to execute a homing routine specified by the user.

There are three homing routines supported by the DMC-40x0: Find Edge (FE), Find Index (FI), and Standard Home (HM).

The Find Edge routine is initiated by the command sequence: FEX <return>, BGX <return>. The Find Edge routine will cause the motor to accelerate, and then slew at constant speed until a transition is detected in the logic state of the Home input. The direction of the FE motion is dependent on the state of the home switch. High level causes forward motion. The motor will then decelerate to a stop. The acceleration rate, deceleration rate and slew speed are specified by the user, prior to the movement, using the commands AC, DC, and SP. *When using the FE command, it is recommended that a high deceleration value be used so the motor will decelerate rapidly after sensing the Home switch.*

The Find Index routine is initiated by the command sequence: FIX <return>, BGX <return>. Find Index will cause the motor to accelerate to the user-defined slew speed (SP) at a rate specified by the user with the AC command and slew until the controller senses a change in the index pulse signal from low to high. The motor then decelerates to a stop at the rate previously specified by the user with the DC command and then moves back to the index pulse and speed HV. Although Find Index is an option for homing, it is not dependent upon a transition in the logic state of the Home input, but instead is dependent upon a transition in the level of the index pulse signal.

The Standard Homing routine is initiated by the sequence of commands HMX <return>, BGX <return>. Standard Homing is a combination of Find Edge and Find Index homing. Initiating the standard homing routine will cause the motor to slew until a transition is detected in the logic state of the Home input. The motor will accelerate at the rate specified by the command, AC, up to the slew speed. After detecting the transition in the logic state on the Home Input, the motor will decelerate to a stop at the rate specified by the command, DC. After the motor has decelerated to a stop, it switches direction and approaches the transition point at the speed of HV counts/sec. When the logic state changes again, the motor moves forward (in the direction of increasing encoder count) at the same speed, until the controller senses the index pulse. After detection, it decelerates to a stop, moves back to the index, and defines this position as 0. The logic state of the Home input can be interrogated with the command MG_HMX. This command returns a 0 or 1 if the logic state is low or high, respectively. The state of the Home input can also be interrogated indirectly with the TS command.

For examples and further information about Homing, see command HM, FI, FE of the Command Reference and the section entitled Homing in the Programming Motion Section of this manual.

## Abort Input

The function of the Abort input is to immediately stop the controller upon transition of the logic state.

**NOTE:** The response of the abort input is significantly different from the response of an activated limit switch. When the abort input is activated, the controller stops generating motion commands immediately, whereas the limit switch response causes the controller to make a decelerated stop.

**NOTE:** The effect of an Abort input is dependent on the state of the off-on-error function for each axis. If the Off-On-Error function is enabled for any given axis, the motor for that axis will be turned off when the abort signal is generated. This could cause the motor to 'coast' to a stop since it is no longer under servo control. If the Off-On-Error function is disabled, the motor will decelerate to a stop as fast as mechanically possible and the motor will remain in a servo state.

All motion programs that are currently running are terminated when a transition in the Abort input is detected. This can be configured with the CN command. For information see the Command Reference, OE and CN.

## ELO (Electronic Lock-Out) Input

Used in conjunction with Galil amplifiers, this input allows the user the shutdown the amplifier at a hardware level. For more detailed information on how specific Galil amplifiers behave when the ELO is triggered, see Integrated Amplifiers and Drivers in the Appendices.

## Reset Input

When this input is driven low, the controller will reset. This is the same as pressing the "RESET" button on the controller.

## Uncommitted Digital Inputs

The DMC-40x0 has 8 opto-isolated inputs. These inputs can be read individually using the function @ IN[x] where x specifies the input number (1 thru 8). These inputs are uncommitted and can allow the user to create conditional statements related to events external to the controller. For example, the user may wish to have the x-axis motor move 1000 counts in the positive direction when the logic state of DI1 goes high.

This can be accomplished by connecting a voltage in the range of +5V to +28V into INCOM of the input circuitry from a separate power supply.

**4080**

Controllers with more than 4 axes have an additional 8 general opto-isolated inputs (inputs 9-16). The INCOM for these inputs is found on the I/O (E-H) D-Sub connector.

An additional 32 I/O are provided at 3.3V (5V option) through the extended I/O. These are not opto-isolated.

**NOTE:** INCOM and LSCOM for Inputs 9-16 and Limit and Home Switches for axes 5-8 are found on the connectors for the E-H axes. **These are NOT the same INCOM and LSCOM for axes 1-4.**

| | |
|---|---|
| DI9-DI16 | INCOM (I/O (E-H) D-Sub connectors) |
| FLSE,RLSE,HOME<br>FLSF,RLSF,HOMF<br>FLSG,RLSG,HOMG<br>FLSH,RLSH,HOMH | LSCOM (I/O (E-H) D-Sub connector) |

# Wiring the Optoisolated Inputs

## Electrical Specifications

| | |
|---|---|
| Input Common (INCOM) Max Voltage | 28 VDC |
| Limit Common (LSCOM) Max Voltage | 28 VDC |
| Minimum Current to turn on Inputs | 1 mA |

## Bi-Directional Capability

All inputs can be used as active high or low - If you are using an isolated power supply you can connect the positive voltage of the supply (+Vs) to INCOM or supply the isolated ground to INCOM. Connecting +Vs to INCOM will configure the inputs for active low. Connecting the isolated ground to INCOM will configure the inputs for active high. If there is not an isolated available, the Galil 5V or 12V and GND may be used. It is recommended to use an isolated supply for the optoisolated inputs.

The optoisolated inputs are configured into groups. For example, the general inputs, DI1-DI8 (inputs 1-8), the ABRT (abort) input and RST (reset) and ELO (electronic lock-out) inputs are one group. Figure 3.1 illustrates the internal circuitry. The INCOM signal is a common connection for all of the inputs in each group.

**4080** The ELO, ABRT and RST pins are found on the I/O (A-D) D-Sub and are duplicated on the I/O (E-H) D-Sub. I.e. There is only one ELO, ABRT and RST input for an 8 axis controller. The common is the INCOM found on the I/O (A-D) D-Sub connector.

The optoisolated inputs are connected in the following groups:

| Group (Controllers with 1- 4 Axes) | Common Signal |
|---|---|
| DI1-DI8, ABRT, RST, ELO | INCOM (I/O (A-D) D-Sub Connectors) |
| FLSA,RLSA,HOMA<br>FLSB,RLSB,HOMB<br>FLSC,RLSC,HOMC<br>FLSD,RLSD,HOMD | LSCOM (I/O (A-D) D-Sub Connectors) |
| **Group (Controllers with 5-8 Axes)** | |
| DI1-DI8, ABRT, RST, ELO | INCOM (I/O (A-D) D-Sub Connectors) |
| FLSA,RLSA,HOMA<br>FLSB,RLSB,HOMB<br>FLSC,RLSC,HOMC<br>FLSD,RLSD,HOMD | LSCOM (I/O (A-D) D-Sub Connectors) |
| DI9-DI16 | INCOM (I/O (E-H) D-Sub Connectors) |
| FLSE,RLSE,HOME<br>FLSF,RLSF,HOMF<br>FLSG,RLSG,HOMG<br>FLSH,RLSH,HOMH | LSCOM (I/O (E-H) D-Sub Connectors) |

*Table 3-1: INCOM and LSCOM information*

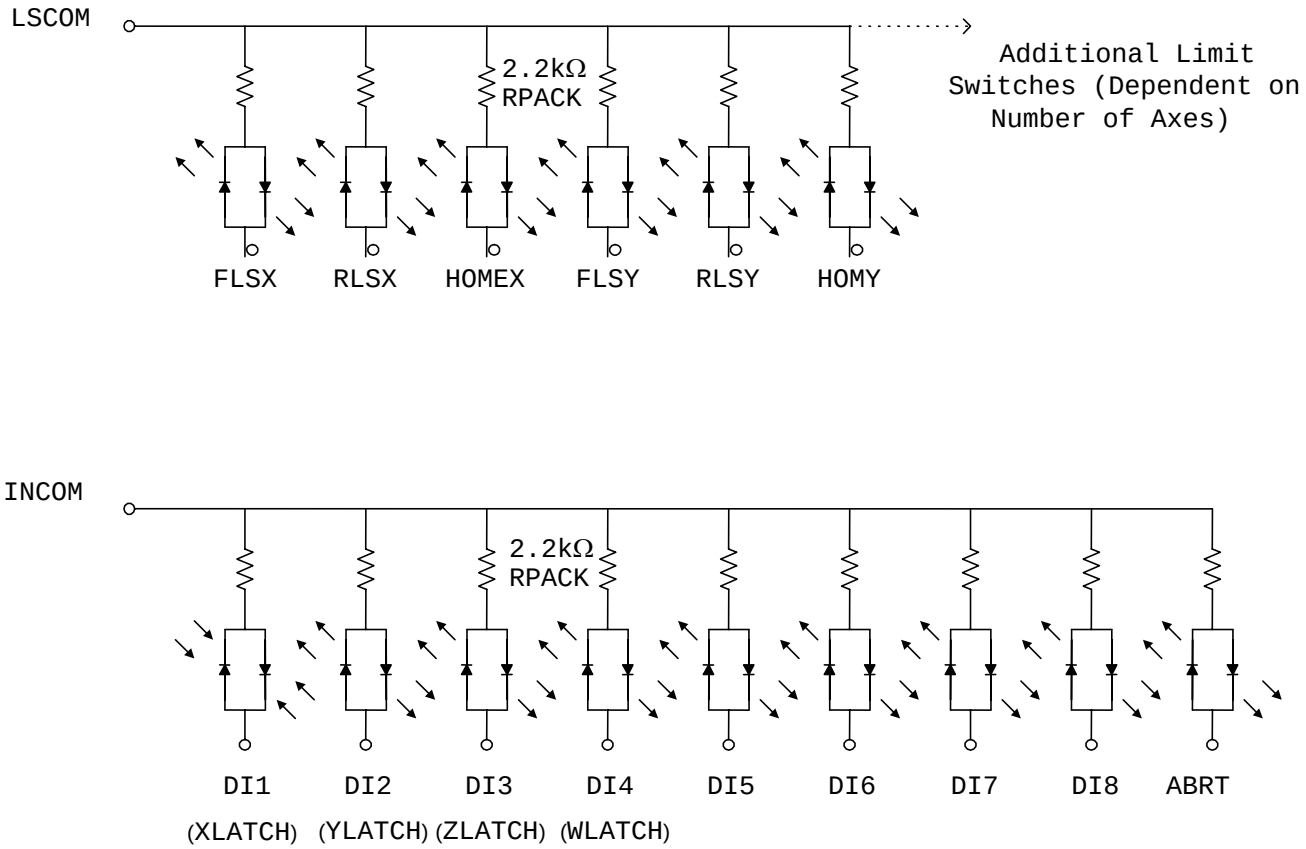



*Figure 3-1: The Optoisolated Inputs.*

## Using an Isolated Power Supply

To take full advantage of opto-isolation, an isolated power supply should be used to provide the voltage at the input common connection. When using an isolated power supply, do not connect the ground of the isolated power to the ground of the controller. A power supply in the voltage range between 5 to 28 Volts may be applied directly (see Figure 3-2). For voltages greater than 28 Volts, a resistor, R, is needed in series with the input such that:

$1\ mA < V\ supply/(R + 2.2K\Omega) < 11\ mA$

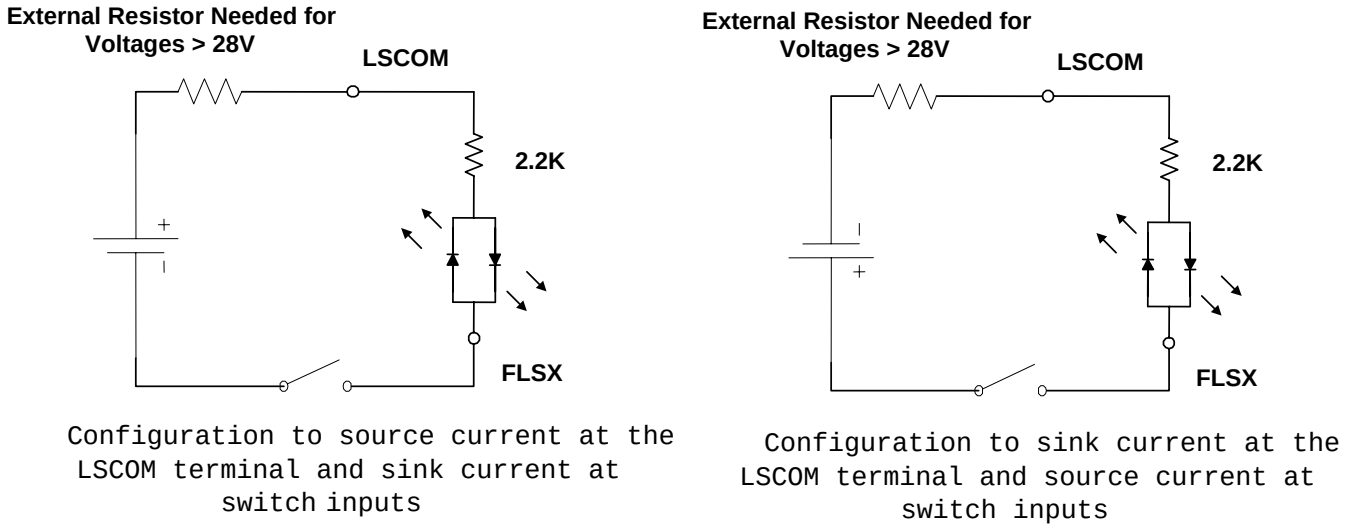



*Figure 3-2. Connecting a single Limit or Home Switch to an Isolated Supply. This diagram only shows the connection for the forward limit switch of the X axis.*

### Bypassing the Opto-Isolation:

If no isolation is needed, the internal 5 Volt supply may be used to power the switches. This can be done by connecting LSCOM or INCOM to 5V.

To close the circuit, wire the desired input to any ground (GND) pin on the controller.

# TTL Inputs

## The Auxiliary Encoder Inputs

The auxiliary encoder inputs can be used for general use. For each axis, the controller has one auxiliary encoder and each auxiliary encoder consists of two inputs, channel A and channel B. The auxiliary encoder inputs are mapped to the inputs 81-96.

Each input from the auxiliary encoder is a differential line receiver and can accept voltage levels between +/- 12 volts. The inputs have been configured to accept TTL level signals. To connect TTL signals, simply connect the signal to the + input and leave the - input disconnected. For other signal levels, the - input should be connected to a voltage that is ½ of the full voltage range (for example, connect the - input to 6 volts if the signal is a 0 - 12 volt logic).

**Example:**

A DMC-4010 has one auxiliary encoder. This encoder has two inputs (channel A and channel B). Channel A input is mapped to input 81 and Channel B input is mapped to input 82. To use this input for 2 TTL signals, the first signal will be connected to AA+ and the second to AB+. AA- and AB- will be left unconnected. To access this input, use the function @IN[81] and @IN[82].

**NOTE:** The auxiliary encoder inputs are not available for any axis that is configured for stepper motor.

# High Power Opto-Isolated Outputs

The DMC-40x0 has different interconnect module options, this section will describe the 500mA optically isolated outputs that are used on the ICM-42x00.

## Electrical Specifications

| | |
|---|---|
| Output Common Max Voltage | 30 VDC |
| Output Common Min Voltage | 12 VDC |
| Max Drive Current per Output | 0.5 A (not to exceed 3A for all 8 outputs) |

## Wiring the Opto-Isolated Outputs

The ICM-42x00 module allows for opto-isolation on all of the digital inputs and outputs. The digital outputs are optically isolated and are capable of sourcing up to 0.5 A per pin with a 3 A limit for the group of 8 outputs. The outputs are configured for hi-side drive only. The supply voltage must be connected to o*utput supply voltage* (OPWR), and the supply return must be connected to *output return* (ORET).

Figure 3-3 shows the manner in which the load should be connected. The output will be at the voltage that is supplied to the OPWR pin. Up to 30 VDC may be supplied to OPWR.

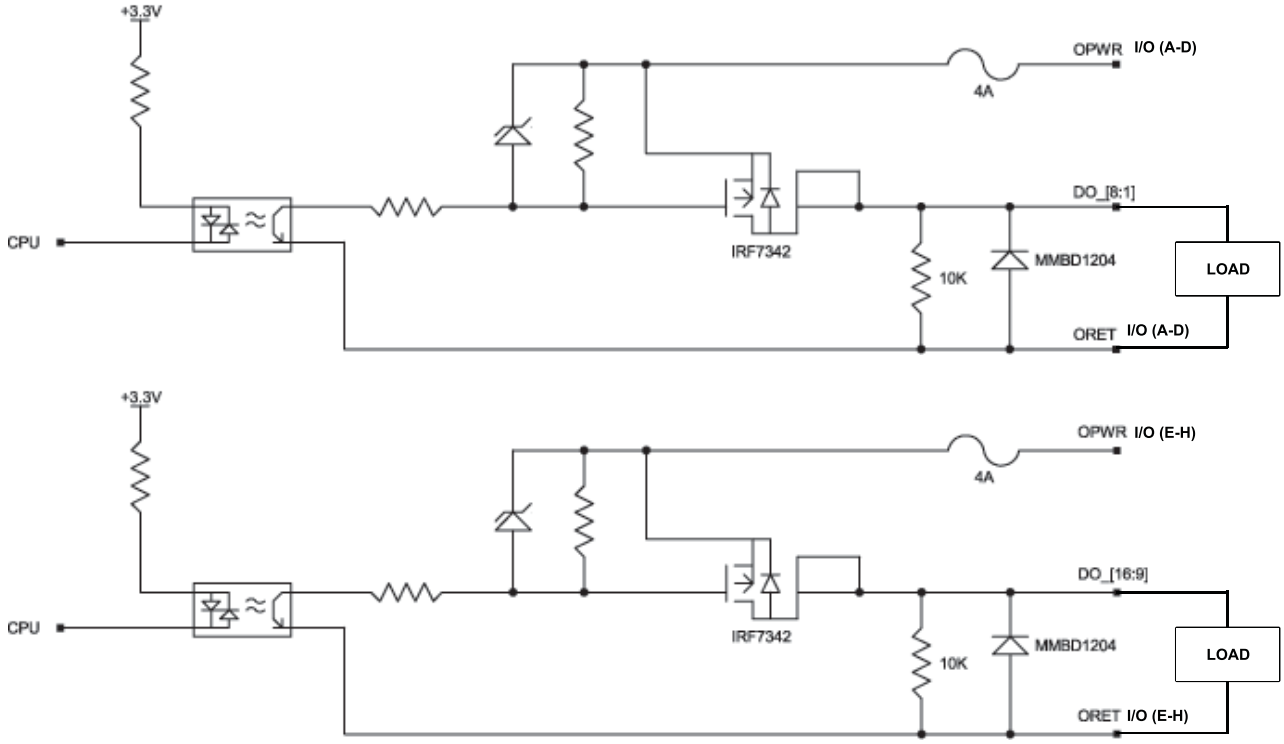



*Figure 3-3 ICM-42x00 General-Purpose Digital Output Opto-Isolation*

**4080** For controllers with 5-8 axes, outputs 9-16 are located on the I/O (E-H) D-Sub connector. The OPWR and ORET for these outputs are also found on the I/O (E-H) D-Sub connector. Connections to the OPWR and ORET on the I/O (E-H) as described above are required for operation of outputs 9-16.

# Analog Inputs

The DMC-40x0 has eight analog inputs configured for the range between -10V and 10V. The inputs are decoded by a 12-bit A/D decoder giving a voltage resolution of approximately .005V. A 16-bit ADC is available as an option (Ex. *DMC-4020(-16bit)-C012-I000*). The analog inputs are specified as AN[x] where x is a number 1 thru 8.

## AQ settings

The analog inputs can be set to a range of +/-10V, +/-5V, 0-5V or 0-10V. The inputs can also be set into a differential mode where analog inputs 2,4,6 and 8 can be set to the negative differential inputs for analog inputs 1,3,5 and 7 respectivally. See the AQ command in the command reference for more information.

## Electrical Specifications

Input Impedance (12 and 16 bit) –

| | |
|---|---|
| Single Ended (Unipolar) | 42kΩ |
| Differential (Bipolar) | 31kΩ |

# TTL Outputs

## Output Compare

The output compare signal is TTL and is available on the I/O (A-D) D-Sub connector as CMP. Output compare is controlled by the position of any of the main encoders on the controller. The output can be programmed to produce an active low pulse (250 nsec) based on an incremental encoder value or to activate once when an axis position has been passed. For further information, see the command OC in the Command Reference.

**4080** For controllers with 5-8 axes, a second output compare signal is available on the I/O (E-H) D-Sub connector.

## Error Output

The controller provides a TTL signal, ERR, to indicate a controller error condition. When an error condition occurs, the ERR signal will go low and the controller LED will go on. An error occurs because of one of the following conditions:

1. At least one axis has a position error greater than the error limit. The error limit is set by using the command ER.
2. The reset line on the controller is held low or is being affected by noise.
3. There is a failure on the controller and the processor is resetting itself.
4. There is a failure with the output IC which drives the error signal.

The ERR signal is found on the I/O (A-D) D-Sub connector.

**4080** For controllers with 5-8 axes, the ERR signal is duplicated on the I/O (E-H) D-Sub connector.

# Extended I/O of the DMC-40x0 Controller

The DMC-40x0 controller offers 32 extended TTL I/O points which can be configured as inputs or outputs in 8 bit increments. Configuration is accomplished with command CO – see Extended I/O of the DMC-40x0 Controller The I/O points are accessed through the 44 pin D-Sub connector labeled EXTENDED I/O. See the appendix for a complete pin out of CMB-41012 Extended I/O 44 pin HD D-Sub Connector.

## Electrical Specifications (3.3V – Standard)

### Inputs

| | |
|---|---|
| Max Input Voltage | 3.4 VDC |
| Guarantee High Voltage | 2.0 VDC |
| Guarantee Low Voltage | 0.8 VDC |

Inputs are internally pulled up to 3.3V through a 4.7kΩ resistor

### Outputs

| | |
|---|---|
| Sink/Source | 4mA per output |

## Electrical Specifications (5V – Option)

### Inputs

| | |
|---|---|
| Max Input Voltage | 5.25 VDC |
| Guarantee High Voltage | 2.0 VDC |
| Guarantee Low Voltage | 0.8 VDC |

Inputs are internally pulled up to 5V through a 4.7kΩ resistor

### Outputs

| | |
|---|---|
| Sink/Source | 20mA |

# Amplifier Interface

## Electrical Specifications

| | | |
|---|---|---|
| Max Amplifier Enable Voltage | 24V | |
| Max Amplifier Enable Current @24V | sink/source | 25 mA |
| Motor Command Output Impedance | 500 Ω | |

## Overview

The DMC-40x0 command voltage ranges between +/-10V and is output on the motor command line - MCMn (where n is A-H). This signal, along with GND, provides the input to the motor amplifiers. The amplifiers must be sized to drive the motors and load. For best performance, the amplifiers should be configured for a torque (current) mode of operation with no additional compensation. The gain should be set such that a 10 volt input results in the maximum required current.

**Note:** The DMC-40x0 controller has an option for differential motor command outputs. For more information contact Galil.

The DMC-40x0 also provides an amplifier enable signal - AENn (where n is A-H). This signal changes under the following conditions: the motor-off command, MO, is given, the watchdog timer activates, or the OE command (Enable Off-On-Error) is set and the position error exceeds the error limit or a limit switch is reached (see OE command in the Command Reference for more information).

For all versions of the ICM-42x00, the standard configuration of the amplifier enable signal is 5V active high amp enable (HAEN) sinking. In other words, the AEN signal will be high when the controller expects the amplifier to be enabled. The polarity and the amplitude can be changed by configuring the Amplifier Enable Circuit on the ICM-42x00.

**If your amplifier requires a different configuration than the default 5V HAEN sinking it is highly recommended that the DMC-40x0 is ordered with the desired configuration.** See the DMC-40x0 ordering information in the catalog (http://www.galilmc.com/catalog/cat40x0.pdf) or contact Galil for more information on ordering different configurations.

**Note1:** Many amplifiers designate the enable input as 'inhibit'.

## ICM-42000 and ICM-42100 Amplifier Enable Circuit

This section describes how to configure the ICM-42000 and ICM-42100 for different Amplifier Enable configurations. It is advised that the user order the DMC-40x0 with the proper Amplifier enable configuration.

The ICM-42000 and ICM-42100 gives the user a broad range of options with regards to the voltage levels present on the enable signal. The user can choose between High-Amp-Enable (HAEN), Low-Amp-Enable (LAEN), 5V logic, 12V logic, external voltage supplies up to 24V, sinking, or sourcing. Tables 3-2 and 3-3 found below illustrate the settings for jumpers, resistor packs, and the socketed optocoupler IC. Refer to Figures 3-4 and 3-5 for precise physical locations of all components. Note that the resistor pack located at RP2 may be reversed to change the active state of the amplifier enable output. However, the polarity of RP6 must not be changed; a different resistor value may be needed to limit the current to 6 mA. The default value for RP6 is 820 ohms, which works at 5V. When using 24 V, RP6 should be replaced with a 4.7 kΩ resistor pack.

**NOTE:** For detailed step-by-step instructions on changing the Amplifier Enable configuration on the ICM-42000 or ICM-42100 see the Configuring the Amplifier Enable Circuit section in the Appendices.

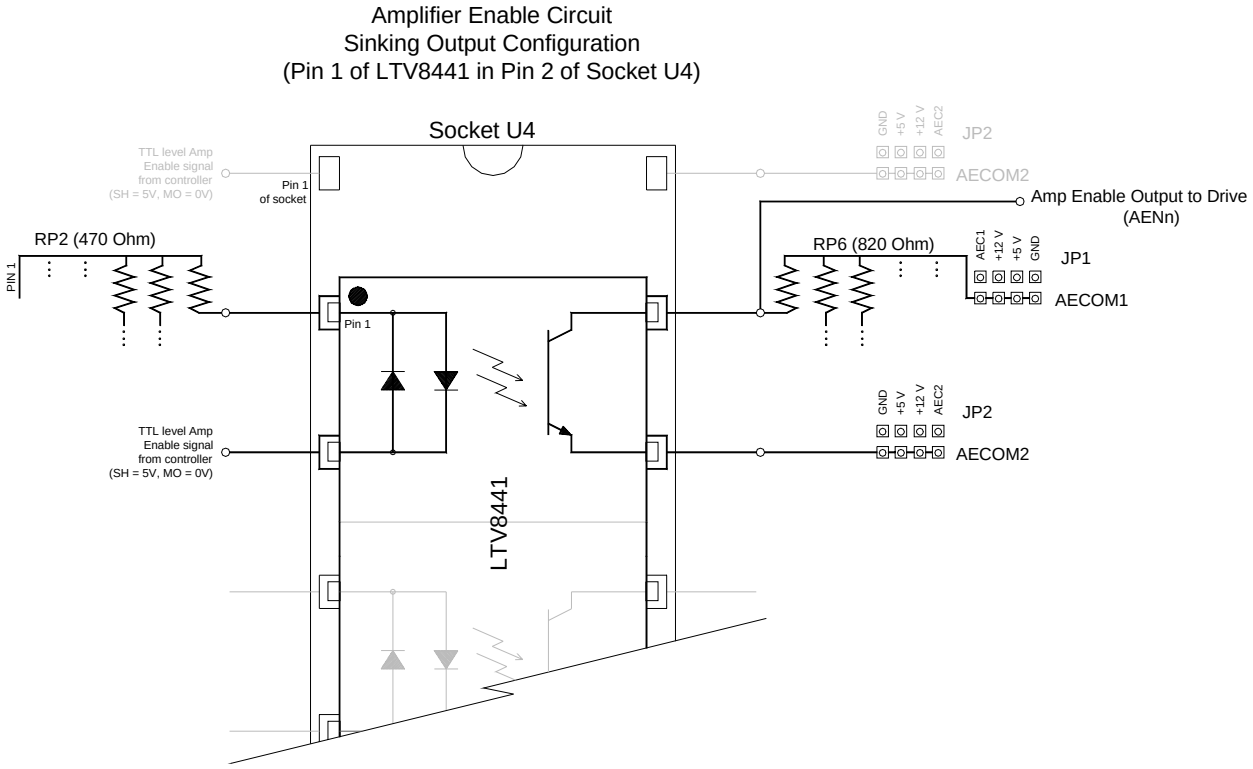



*Figure 3-4: Amplifier Enable Circuit Sinking Output Configuration*

| **Sinking Configuration (pin1 of LTV8441 chip in pin2 of socket U4)** | | | |
|---|---|---|---|
| **Logic State** | **JP1** | **JP2** | **RP2** (square pin next to RP2 label is 5V) |
| 5V, HAEN (Default Configuration) | 5V - AECOM1 | GND – AECOM2 | Dot on R-pack next to RP2 label |
| 5V, LAEN | 5V – AECOM1 | GND – AECOM2 | Dot on R-pack opposite RP2 label |
| 12V, HAEN | +12V – AECOM1 | GND – AECOM2 | Dot on R-pack next to RP2 label |
| 12V, LAEN | +12V – AECOM1 | GND – AECOM2 | Dot on R-pack opposite RP2 label |
| Isolated 24V, HAEN | AEC1 – AECOM1 | AEC2 – AECOM2 | Dot on R-pack next to RP2 label |
| Isolated 24V, LAEN | AEC1 - AECOM1 | AEC2 - AECOM2 | Dot on R-pack opposite RP2 label |
| For 24V isolated enable, tie +24V of external power supply to AEC1 at the D-sub, tie common return to AEC2. Replace RP6 with a 4.7 kΩ resistor pack. For Axes A-D, AEC1 and AEC2 are located on the EXTERNAL DRIVER (A-D) D-Sub connector. For Axes E-H, AEC1 and AEC2 are located on the EXTERNAL DRIVER (E-H) D-Sub connector. **Note:** AEC1 and AEC2 for axes A-D are NOT connected to AEC1 and AEC2 for axes E-H. | | | |

*Table 3-2: Sinking Configuration*

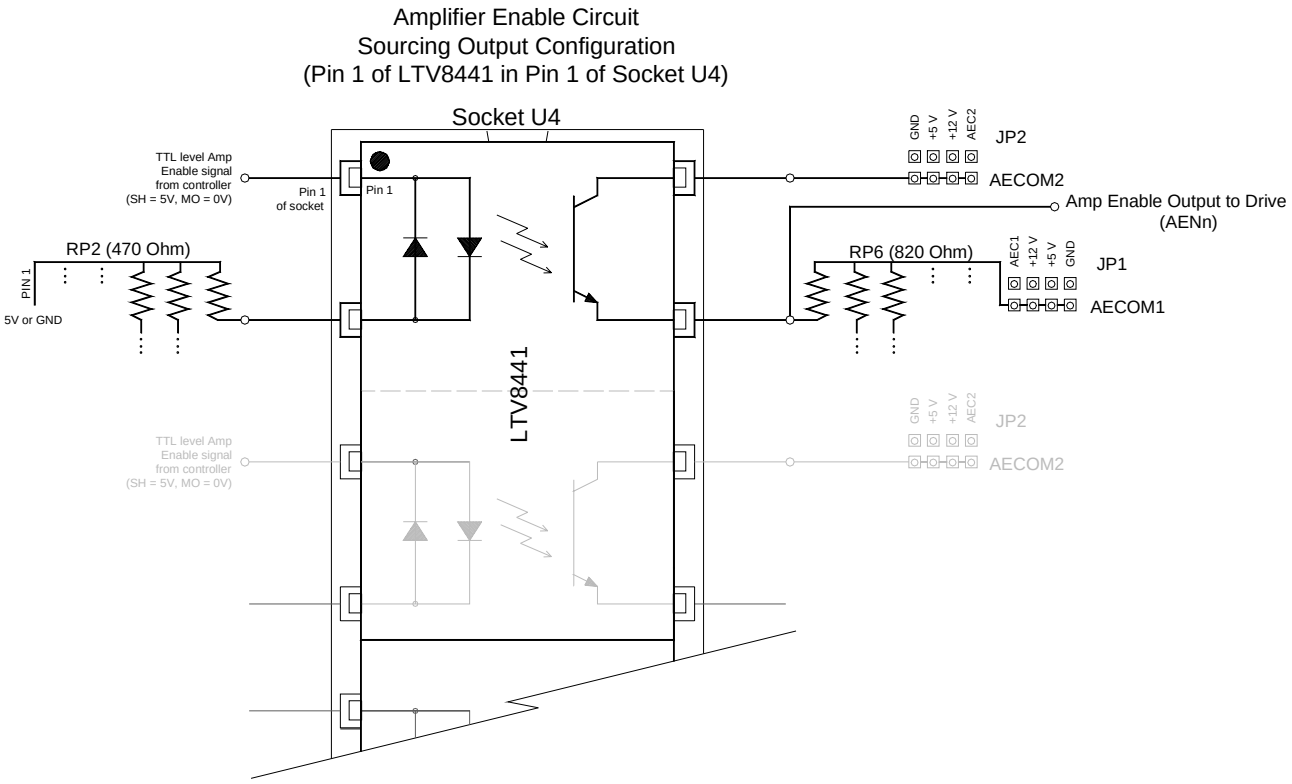



*Figure 3-5: Amplifier Enable Circuit Sourcing Output Configuration*

| **Sourcing Configuration (pin1 of LTV8441 chip in pin1 of socket U4)** | | | |
|---|---|---|---|
| **Logic State** | **JP1** | **JP2** | **RP2** (square pin next to RP2 label is 5V) |
| 5V, HAEN | GND – AECOM1 | 5V – AECOM2 | Dot on R-pack opposite RP2 label |
| 5V, LAEN | GND – AECOM1 | 5V – AECOM2 | Dot on R-pack next to RP2 label |
| 12V, HAEN | GND – AECOM1 | +12V – AECOM2 | Dot on R-pack opposite RP2 label |
| 12V, LAEN | GND – AECOM1 | +12V – AECOM2 | Dot on R-pack next to RP2 label |
| Isolated 24V, HAEN | AEC1 – AECOM1 | AEC2 - AECOM2 | Dot on R-pack opposite RP2 label |
| Isolated 24V, LAEN | AEC1 - AECOM1 | AEC2 – AECOM2 | Dot on R-pack next to RP2 label |
| For 24V isolated enable, tie +24V of external power supply to AEC2 at the D-sub, tie common return to AEC1. Replace RP6 with a 4.7 kΩ resistor pack. For Axes A-D, AEC1 and AEC2 are located on the EXTERNAL DRIVER (A-D) D-Sub connector. For Axes E-H, AEC1 and AEC2 are located on the EXTERNAL DRIVER (E-H) D-Sub connector. **Note:** AEC1 and AEC2 for axes A-D are NOT connected to AEC1 and AEC2 for axes E-H. | | | |

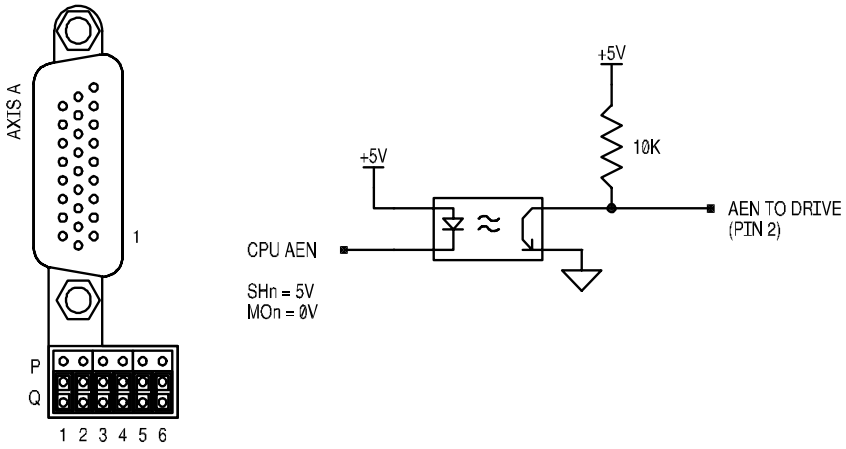
*Table3-3: Sourcing Configuration*

## ICM-42200 Amplifier Enable Circuit

This section describes how to configure the ICM-42200 for different Amplifier Enable outputs. The ICM-42200 is designed to be used with external amplifiers. As a result, the amplifier enable circuit for each axis is individually configurable through jumper settings. The user can choose between High-Amp-Enable (HAEN), Low-Amp-Enable (LAEN), 5V logic, 12V logic, external voltage supplies up to 24V, sinking, or sourcing. Every different configuration is described below with jumper settings and a schematic of the circuit.

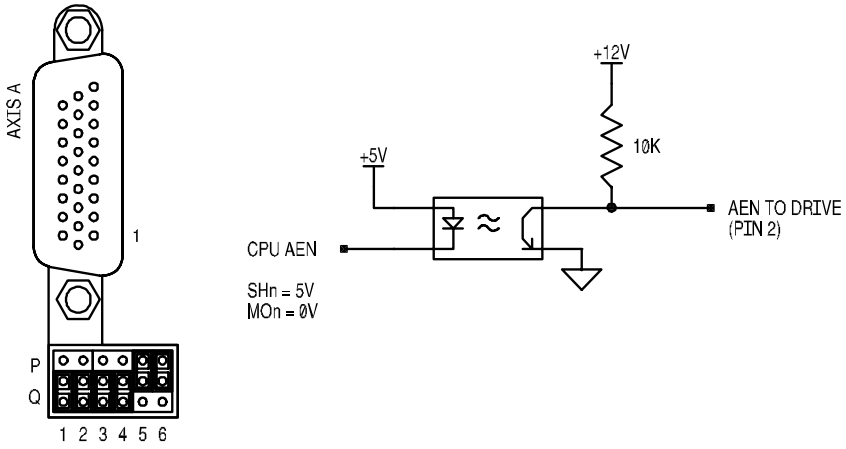
---

+5V
HIGH AMP ENABLE
SINKING



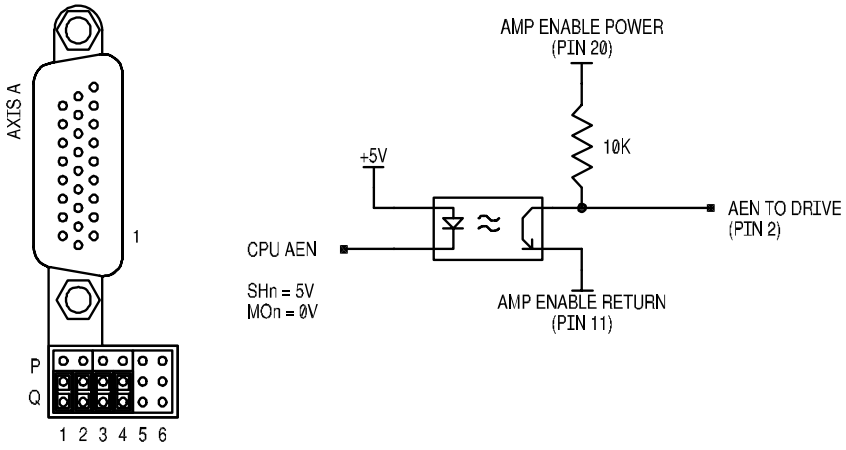

---

+12V
HIGH AMP ENABLE
SINKING



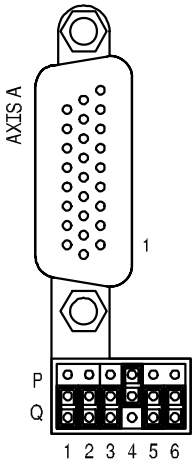

---

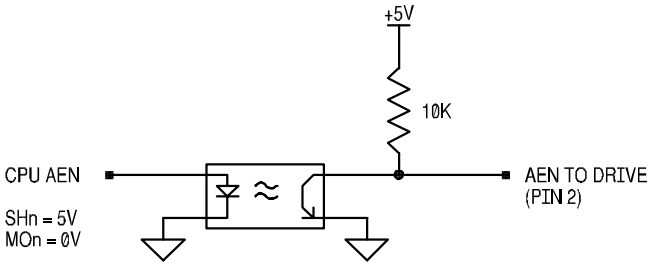
ISOLATED SUPPLY
HIGH AMP ENABLE
SINKING



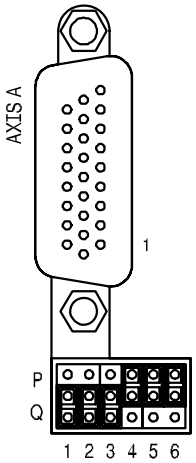

---

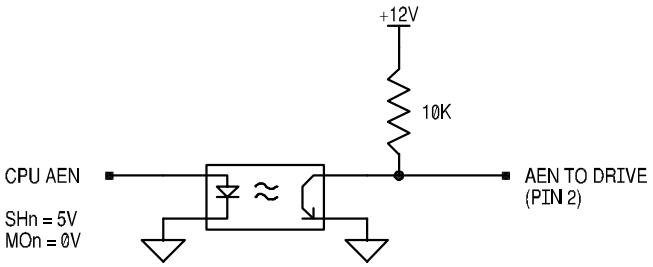
+5V
LOW AMP ENABLE
SINKING



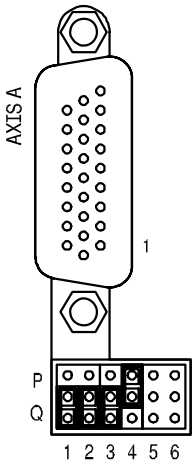



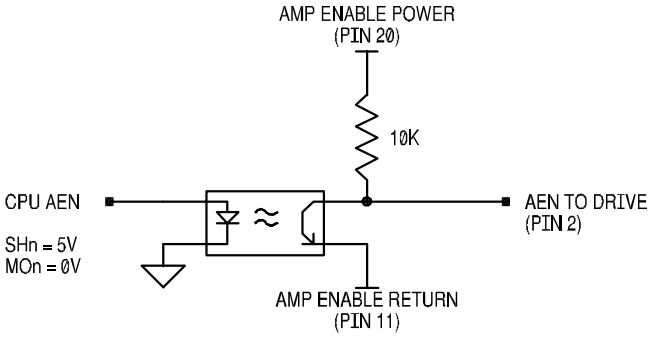

+12V
LOW AMP ENABLE
SINKING



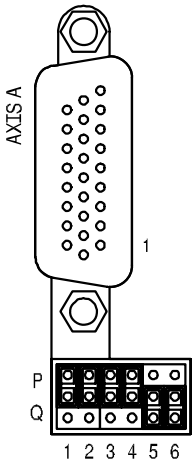



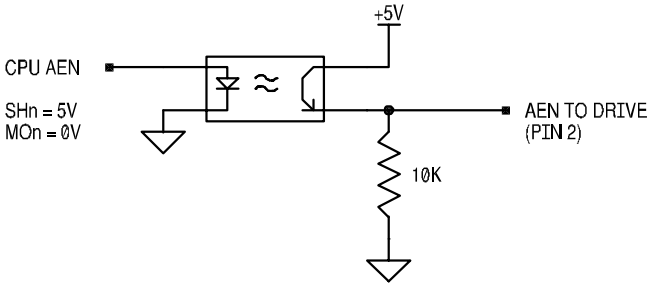

ISOLATED SUPPLY
LOW AMP ENABLE
SINKING



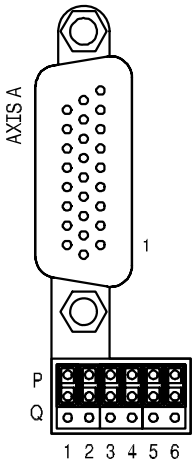



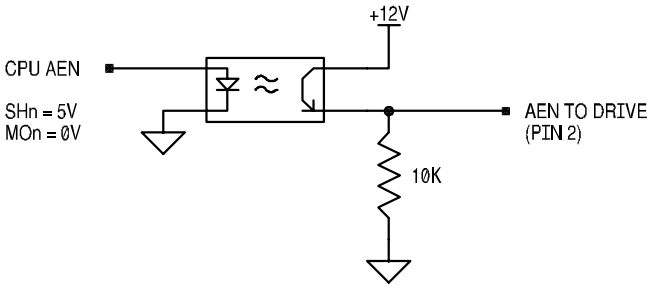
+5V
HIGH AMP ENABLE
SOURCING



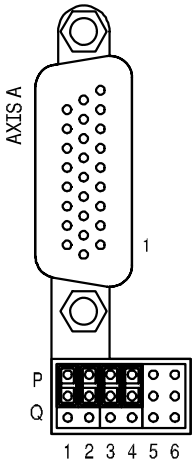



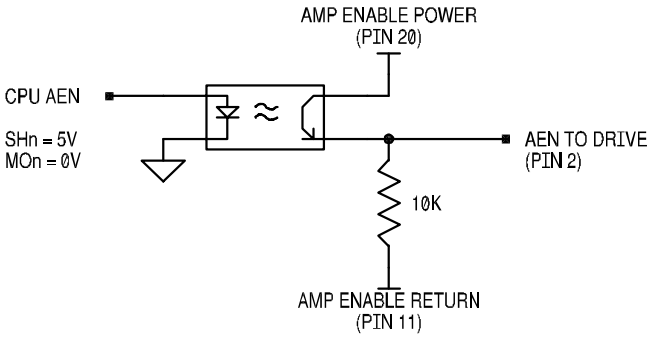

+12V
HIGH AMP ENABLE
SOURCING



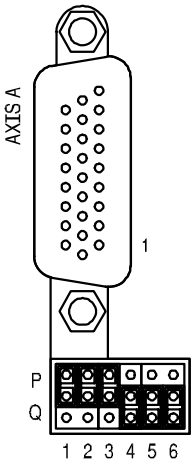



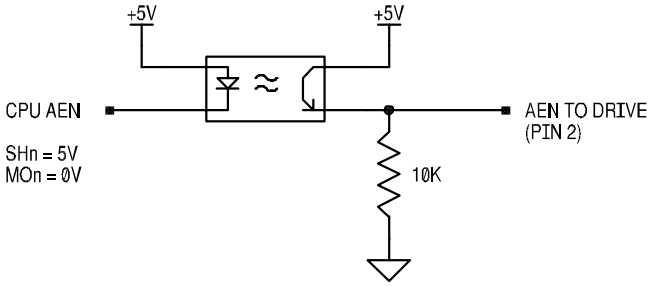

ISOLATED SUPPLY
HIGH AMP ENABLE
SOURCING



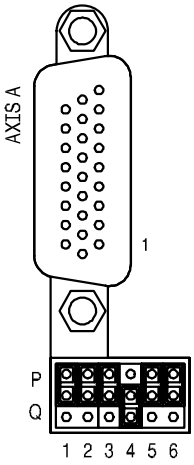



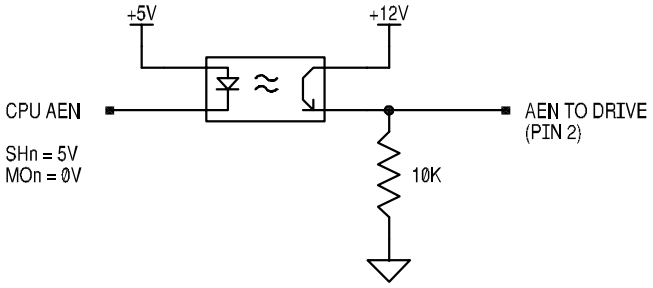
+5V
LOW AMP ENABLE
SOURCING



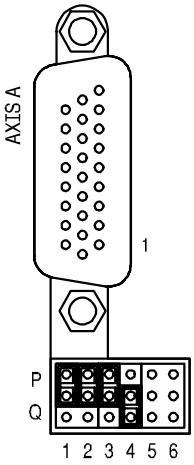



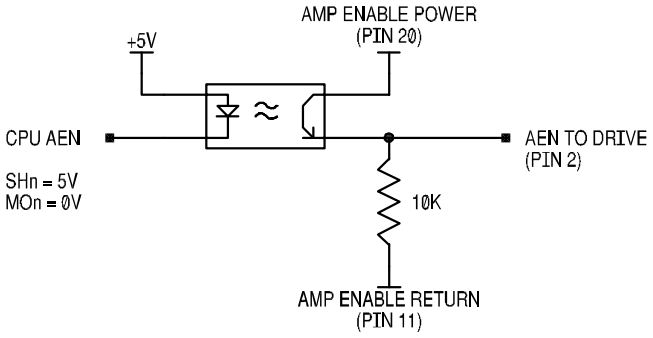

+12V
LOW AMP ENABLE
SOURCING




ISOLATED SUPPLY
LOW AMP ENABLE
SOURCING



# Chapter 4 Software Tools and Communication

## Introduction

The default configuration DMC-40x0 has two RS232 ports and 1 Ethernet port. The main RS-232 port is the data set and can be configured through the jumpers on the top of the controller. The auxiliary RS-232 port is the data term and can be configured with the software command CC. The auxiliary RS-232 port can be configured either for daisy chain operation or as a general port. This configuration can be saved using the Burn (BN) instruction. The RS232 ports also have a clock synchronizing line that allows synchronization of motion on more than one controller.

Galil software is available for PC computers running Microsoft Windows® to communicate with the DMC-40x0 controller. Standard Galil communications software utilities are available for Windows operating systems, which includes **GalilTools**. This software package has been developed to operate under Windows and Linix, and include all the necessary drivers to communicate to the controller. In addition, GalilTools includes a software development communication library which allows users to create their own application interfaces using programming environments such as C, C++, Visual Basic, and LabVIEW.

The following sections in this chapter are a description of the communications protocol, and a brief introduction to the software tools and communication techniques used by Galil. At the application level, GalilTools is the basic programs that the majority of users will need to communicate with the controller, to perform basic setup, and to develop application code (.DMC programs) that is downloaded to the controller. At the Galil API level, the GalilTools Communication Library is available for users, who wish to develop their own custom application programs to communicate to the controller. Custom application programs can utilize API function calls directly to our DLL's. At the driver level, we provide fundamental hardware interface information for users who desire to create their own drivers.

## RS232 and RS422 Ports

The RS232 pin-out description for the main and auxiliary port is given below. Note that the auxiliary port is essentially the same as the main port except inputs and outputs are reversed. The DMC-40x0 may also be configured by the factory for RS422. These pin-outs are also listed below.

### RS-232 Configuration

**NOTE:** If you are connecting the RS232 auxiliary port to a terminal or any device which is a DATASET, it is necessary to use a connector adapter, which changes a dataset to a dataterm. This cable is also known as a 'null' modem cable.

### RS232 - Main Port {P1} DATATERM

1 No Connect
2 Transmit Data - output
3 Receive Data - input
4 No Connect
5 Ground
6 No Connect
7 Clear To Send - input
8 Request To Send - output
9 No connect

### RS232 - Auxiliary Port {P2} DATASET

1 No Connect
2 Receive Data - input
3 Transmit Data - output
4 No Connect
5 Ground
6 No Connect
7 Request To Send - output
8 Clear To Send - input
9 No Connect (Can be connected to 5V with APWR jumper)

## Configuration

Configure your PC for 8-bit data, one start-bit, one stop-bit, full duplex and no parity. The baud rate for the RS232 communication can be selected by setting the proper switch configuration on the front panel according to the table below.

### Baud Rate Selection

| SWITCH SETTINGS | | |
|---|---|---|
| **19.2** | **38.4** | **BAUD RATE** |
| ON | ON | 9600 |
| ON | OFF | 19200 |
| OFF | ON | 38400 |
| OFF | OFF | 115200 |

## Handshaking

The RS232 main port is set for hardware handshaking. Hardware Handshaking uses the RTS and CTS lines. The CTS line will go high whenever the DMC-40x0 is not ready to receive additional characters. The RTS line will inhibit the DMC-40x0 from sending additional characters. Note, the RTS line goes high for inhibit.

The auxiliary port of the DMC-40x0 can be configured either as a general port or for the daisy-chain. When configured as a general port, the port can be commanded to send ASCII messages to another DMC-40x0 controller or to a display terminal or panel.

**CC Command:** (Configure Communication) at port 2. The command is in the format of (See CC in the Command Reference for more information):

```
CC m,n,r,p
```

where ‘m’ sets the baud rate, ‘n’ sets for either handshake or non-handshake mode, ‘r’ sets for general port or the auxiliary port, and ‘p’ turns echo on or off.

m - Baud Rate – 9600,19200,38400,115200
n - Handshake - 0=No; 1=Yes
r - Mode - 0=Disabled; 1=enabled
p - Echo - 0=Off; 1=On; Valid only if r=0

**NOTE:** for the handshake of the auxiliary port, the roles for the RTS and CTS lines are reversed.

**Example:**

```
CC 19200,0,1,1
```

Configure auxiliary communication port for 19200 baud, no handshake, general port mode and echo turned on.

## RS-422 Configuration

The DMC-40x0 can be ordered with the main and/or auxiliary port configured for RS-422 communication. RS-422 communication is a differentially driven serial communication protocol that should be used when long distance serial communication is required in an application.

### RS-422-Main Port (Non-Standard Option)

Standard connector and cable when DMC-40x0 is ordered with RS-422 Option.

| Pin | Signal |
|---|---|
| 1 | RTS- |
| 2 | TXD- |
| 3 | RXD- |
| 4 | CTS- |
| 5 | GND |
| 6 | RTS+ |
| 7 | TXD+ |
| 8 | RXD+ |
| 9 | CTS+ |

### RS-422-Auxiliary Port (Non-Standard Option)

Standard connector and cable when DMC-40x0 is ordered with RS-422 Option.

| Pin | Signal |
|---|---|
| 1 | CTS- |
| 2 | RXD- |
| 3 | TXD- |
| 4 | RTS- |
| 5 | GND |
| 6 | CTS+ |
| 7 | RXD+ |
| 8 | TXD+ |
| 9 | RTS+ |

# Ethernet Configuration

## Communication Protocols

The Ethernet is a local area network through which information is transferred in units known as packets. Communication protocols are necessary to dictate how these packets are sent and received. The DMC-40x0 supports two industry standard protocols, TCP/IP and UDP/IP. The controller will automatically respond in the format in which it is contacted.

TCP/IP is a "connection" protocol. The master must be connected to the slave in order to begin communicating. Each packet sent is acknowledged when received. If no acknowledgement is received, the information is assumed lost and is resent.

Unlike TCP/IP, UDP/IP does not require a "connection". This protocol is similar to communicating via RS232. If information is lost, the controller does not return a colon or question mark. Because the protocol does not provide for lost information, the sender must re-send the packet.

Although UDP/IP is more efficient and simple, Galil recommends using the TCP/IP protocol. TCP/IP insures that if a packet is lost or destroyed while in transit, it will be resent.

Ethernet communication transfers information in 'packets'. The packets must be limited to 512 data bytes (including UDP/TCP IP Header) or less. Larger packets could cause the controller to lose communication.

**NOTE:** In order not to lose information in transit, Galil recommends that the user wait for an acknowledgement of receipt of a packet before sending the next packet.

## Addressing

There are three levels of addresses that define Ethernet devices. The first is the MAC or hardware address. This is a unique and permanent 6 byte number. No other device will have the same MAC address. The DMC-40x0 MAC address is set by the factory and the last two bytes of the address are the serial number of the board. To find the Ethernet MAC address for a DMC-40x0 unit, use the TH command. A sample is shown here with a unit that has a serial number of 3:

Sample MAC Ethernet Address: 00-50-4C-20-04-AF

The second level of addressing is the IP address. This is a 32-bit (or 4 byte) number that usually looks like this: 192.168.15.1. The IP address is constrained by each local network and must be assigned locally. Assigning an IP address to the DMC-40x0 controller can be done in a number of ways.

The first method for setting the IP address is using a DHCP server. The DH command controls whether the DMC-40x0 controller will get an IP address from the DHCP server. If the unit is set to DH1 (default) and there is a DHCP server on the network, the controller will be dynamically assigned an IP address from the server. Setting the board to DH0 will prevent the controller from being assigned an IP address from the server.

The second method to assign an IP address is to use the BOOT-P utility via the Ethernet connection. The BOOT-P functionality is only enabled when DH is set to 0. Either a BOOT-P server on the internal network or the Galil software may be used. When opening the Galil Software, it will respond with a list of all DMC-40x0's and other controllers on the network that do not currently have IP addresses. The user must select the board and the software will assign the specified IP address to it. This address will be burned into the controller (BN) internally to save the IP address to the non-volatile memory.

**NOTE**: if multiple boards are on the network – use the serial numbers to differentiate them.

**CAUTION: Be sure that there is only one BOOT-P or DHCP server running. If your network has DHCP or BOOT-P running, it may automatically assign an IP address to the DMC-40x0 controller upon linking it to the network. In order to ensure that the IP address is correct, please contact your system administrator before connecting the I/O board to the Ethernet network.**

The third method for setting an IP address is to send the IA command through the RS-232 port. (Note: The IA command is only valid if DH0 is set). The IP address may be entered as a 4 byte number delimited by commas (industry standard uses periods) or a signed 32 bit number (e.g. IA 124,51,29,31 or IA 2083724575). Type in BN to save the IP address to the DMC-40x0 non-volatile memory.

**NOTE:** Galil strongly recommends that the IP address selected is not one that can be accessed across the Gateway. The Gateway is an application that controls communication between an internal network and the outside world.

The third level of Ethernet addressing is the UDP or TCP port number. The Galil board does not require a specific port number. The port number is established by the client or master each time it connects to the DMC-40x0 board. Typical port numbers for applications are:

Port 23: Telnet
Port 502: Modbus

## Communicating with Multiple Devices

The DMC-40x0 is capable of supporting multiple masters and slaves. The masters may be multiple PC's that send commands to the controller. The slaves are typically peripheral I/O devices that receive commands from the controller.

**NOTE:** The term "Master" is equivalent to the internet "client". The term "Slave" is equivalent to the internet "server".

An Ethernet handle is a communication resource within a device. The DMC-40x0 can have a maximum of 8 Ethernet handles open at any time. When using TCP/IP, each master or slave uses an individual Ethernet handle. In UDP/IP, one handle may be used for all the masters, but each slave uses one. (Pings and ARPs do not occupy handles.) If all 8 handles are in use and a 9$^{th}$ master tries to connect, it will be sent a "reset packet" that generates the appropriate error in its windows application.

**NOTE:** There are a number of ways to reset the controller. Hardware reset (push reset button or power down controller) and software resets (through Ethernet or RS232 by entering RS). The only reset that will not cause the controller to disconnect is a software reset via the Ethernet.

When the Galil controller acts as the master, the IH command is used to assign handles and connect to its slaves. The IP address may be entered as a 4 byte number separated with commas (industry standard uses periods) or as a signed 32 bit number. A port number may also be specified, but if it is not, it will default to 1000. The protocol (TCP/IP or UDP/IP) to use must also be designated at this time. Otherwise, the controller will not connect to the slave. (Ex. IHB=151,25,255,9<179>2 This will open handle #2 and connect to the IP address 151.25.255.9, port 179, using TCP/IP)

Which devices receive what information from the controller depends on a number of things. If a device queries the controller, it will receive the response unless it explicitly tells the controller to send it to another device. If the command that generates a response is part of a downloaded program, the response will route to whichever port is specified as the default (unless explicitly told to go to another port with the CF command). To designate a specific destination for the information, add {Eh} to the end of the command. (Ex. MG{EC}"Hello" will send the message "Hello" to handle #3. TP,,?{EF} will send the z axis position to handle #6.)

## Multicasting

A multicast may only be used in UDP/IP and is similar to a broadcast (where everyone on the network gets the information) but specific to a group. In other words, all devices within a specified group will receive the information that is sent in a multicast. There can be many multicast groups on a network and are differentiated by their multicast IP address. To communicate with all the devices in a specific multicast group, the information can be sent to the multicast IP address rather than to each individual device IP address. All Galil controllers belong to a default multicast address of 239.255.19.56. The controller's multicast IP address can be changed by using the IA> u command.

## Using Third Party Software

Galil supports DHCP, ARP, BOOT-P, and Ping which are utilities for establishing Ethernet connections. DHCP is a protocol used by networked devices (clients) to obtain the parameters necessary for operation in an Internet Protocol network. ARP is an application that determines the Ethernet (hardware) address of a device at a specific IP address. BOOT-P is an application that determines which devices on the network do not have an IP address and assigns the IP address you have chosen to it. Ping is used to check the communication between the device at a specific IP address and the host computer.

The DMC-40x0 can communicate with a host computer through any application that can send TCP/IP or UDP/IP packets. A good example of this is Telnet, a utility that comes with most Windows systems.

# Modbus

An additional protocol layer is available for speaking to I/O devices. Modbus is an RS-485 protocol that packages information in binary packets that are sent as part of a TCP/IP packet. In this protocol, each slave has a 1 byte slave address. The DMC-40x0 can use a specific slave address or default to the handle number. The port number for Modbus is 502.

The Modbus protocol has a set of commands called function codes. The DMC-40x0 supports the 10 major function codes:

| Function Code | Definition |
|---|---|
| 01 | Read Coil Status (Read Bits) |
| 02 | Read Input Status (Read Bits) |
| 03 | Read Holding Registers (Read Words) |
| 04 | Read Input Registers (Read Words) |
| 05 | Force Single Coil (Write One Bit) |
| 06 | Preset Single Register (Write One Word) |
| 07 | Read Exception Status (Read Error Code) |
| 15 | Force Multiple Coils (Write Multiple Bits) |
| 16 | Preset Multiple Registers (Write Words) |
| 17 | Report Slave ID |

The DMC-40x0 provides three levels of Modbus communication. The first level allows the user to create a raw packet and receive raw data. It uses the MBh command with a function code of –1. The format of the command is

MBh = -1,len,array[] where len is the number of bytes

array[] is the array with the data

The second level incorporates the Modbus structure. This is necessary for sending configuration and special commands to an I/O device. The formats vary depending on the function code that is called. For more information refer to the Command Reference.

The third level of Modbus communication uses standard Galil commands. Once the slave has been configured, the commands that may be used are @IN[], @AN[], SB, CB, OB, and AO. For example, AO 2020,8.2 would tell I/O number 2020 to output 8.2 volts.

If a specific slave address is not necessary, the I/O number to be used can be calculated with the following:

I/O Number = (HandleNum*1000) + ((Module-1)*4) + (BitNum-1)

Where HandleNum is the handle number from 1 (A) to 6 (F). Module is the position of the module in the rack from 1 to 16. BitNum is the I/O point in the module from 1 to 4.

If an explicit slave address is to be used, the equation becomes:

I/O Number = (SlaveAddress*10000) + (HandleNum*1000) +((Module-1)*4) + (Bitnum-1)

# Modbus Examples

## Example #1

DMC-4040 connected as a Modbus master to a RIO-47120 via Modbus. The DMC-4040 will set or clear all 16 of the RIO's digital outputs

1. Begin by opening a connection to the RIO which in our example has IP address 192.168.1.120

```
IHB=192,168,1,120<502          (Issued to DMC-4040)
```

2. Dimension an array to store the commanded values. Set array element 0 equal to 170 and array element 1 equal to 85. (array element 1 configures digital outputs 15-8 and array element 0 configures digital outputs 7-0)

```
DM myarray[2]
myarray[0] = 170               (which is 10101010 in binary)
myarray[1] = 85                (which is 01010101in binary)
```

3. a) Send the appropriate MB command. Use function code 15. Start at output 0 and set/clear all 16 outputs based on the data in myarray[]

```
MBB=,15,0,16,myarray[]
```

3. b) Set the outputs using the SB command.

```
SB2001;SB2003;SB2005;SB2007;SB2008;SB2010;SB2012;SB2014;
```

### Results:

Both steps 3a and 3b will result in outputs being activated as below. The only difference being that step 3a will set and clear all 16 bits where as step 3b will only set the specified bits and will have no affect on the others.

| Bit Number | Status |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 0 |
| 3 | 1 |
| 4 | 0 |
| 5 | 1 |
| 6 | 0 |
| 7 | 1 |

| Bit Number | Status |
|---|---|
| 8 | 1 |
| 9 | 0 |
| 10 | 1 |
| 11 | 0 |
| 12 | 1 |
| 13 | 0 |
| 14 | 1 |
| 15 | 0 |

## Example #2

DMC-4040 connected as a Modbus master to a 3rd party PLC. The DMC-4040 will read the value of analog inputs 3 and 4 on the PLC located at addresses 40006 and 40008 respectively. The PLC stores values as 32-bit floating point numbers which is common.

1. Begin by opening a connection to the PLC which has an IP address of 192.168.1.10 in our example

```
IHB=192,168,1,10<502
```

2. Dimension an array to store the results

```
DM myanalog[4]
```

3. Send the appropriate MB command. Use function code 4 (as specified per the PLC). Start at address 40006. Retrieve 4 modbus registers (2 modbus registers per 1 analog input, as specified by the PLC)

```
MBB=,4,40006,4,myanalog[]
```

### Results:

Array elements 0 and 1 will make up the 32 bit floating point value for analog input 3 on the PLC and array elements 2 and 3 will combine for the value of analog input 4.

```
myanalog[0]=16412=0x401C
myanalog[1]=52429=0xCCCD
myanalog[2]=49347=0xC0C3
myanalog[3]=13107=0x3333
```

Analog input 3 = 0x401CCCCD = 2.45V

Analog input 4 = 0xC0C33333 = -6.1V

### Example #3

DMC-4040 connected as a Modbus master to a hydraulic pump. The DMC-4040 will set the pump pressure by writing to an analog output on the pump located at Modbus address 30000 and consisting of 2 Modbus registers forming a 32 bit floating point value.

1. Begin by opening a connection to the pump which has an IP address of 192.168.1.100 in our example

```
IHB=192,168,1,100<502
```

2. Dimension and fill an array with values that will be written to the PLC

```
DM pump[2]
pump[0]=16531=0x4093
pump[1]=13107=0x3333
```

3. Send the appropriate MB command. Use function code 16. Start at address 30000 and write to 2 registers using the data in the array pump[]

```
MBB=,16,30000,2,pump[]
```

### Results:

Analog output will be set to 0x40933333 which is 4.6V

To view an example procedure for communicating with an OPTO-22 rack, refer to Example- Communicating with OPTO-22 SNAP-B3000-ENET in the Appendices.

# Data Record

The DMC-40x0 can provide a block of status information with the use of a single command, QR. This command, along with the QZ command can be very useful for accessing complete controller status. The QR command will return 4 bytes of header information and specific blocks of information as specified by the command arguments:

```
QR ABCDEFGHST
```

Each argument corresponds to a block of information according to the Data Record Map below. If no argument is given, the entire data record map will be returned. Note that the data record size will depend on the number of axes.

***Note:*** *UB = Unsigned Byte (1), UW = Unsigned Word (2), SW = Signed Word (2), SL = Signed Long Word (4), UL = Unsigned Long Word (4)*

| ADDR | TYPE | ITEM |
|---|---|---|
| 00 | UB | 1st Byte of Header |
| 01 | UB | 2nd Byte of Header |
| 02 | UB | 3rd Byte of Header |
| 03 | UB | 4th Byte of Header |
| 04-05 | UW | sample number |
| 06 | UB | general input block 0 (inputs 1-8) |
| 07 | UB | general input block 1 (inputs 9-16) |
| 08 | UB | general input block 2 (inputs 17-24) |
| 09 | UB | general input block 3 (inputs 25-32) |
| 10 | UB | general input block 4 (inputs 33-40) |
| 11 | UB | general input block 5 (inputs 41-48) |
| 12 | UB | general input block 6 (inputs 49-56) |
| 13 | UB | general input block 7 (inputs 57-64) |
| 14 | UB | general input block 8 (inputs 65-72) |
| 15 | UB | general input block 9 (inputs 73-80) |
| 16 | UB | general output block 0 (outputs 1-8) |
| 17 | UB | general output block 1 (outputs 9-16) |
| 18 | UB | general output block 2 (outputs 17-24) |
| 19 | UB | general output block 3 (outputs 25-32) |
| 20 | UB | general output block 4 (outputs 33-40) |
| 21 | UB | general output block 5 (outputs 41-48) |
| 22 | UB | general output block 6 (outputs 49-56) |
| 23 | UB | general output block 7 (outputs 57-64) |
| 24 | UB | general output block 8 (outputs 65-72) |
| 25 | UB | general output block 9 (outputs 73-80) |
| 26-27 | SW (new) | Reserved |
| 28-29 | SW (new) | Reserved |
| 30-31 | SW (new) | Reserved |
| 32-33 | SW (new) | Reserved |
| 34-35 | SW (new) | Reserved |
| 36-37 | SW (new) | Reserved |
| 38-39 | SW (new) | Reserved |
| 40-41 | SW (new) | Reserved |
| 42 | UB | Ethernet Handle A Status |
| 43 | UB | Ethernet Handle B Status |
| 44 | UB | Ethernet Handle C Status |
| 45 | UB | Ethernet Handle D Status |
| 46 | UB | Ethernet Handle E Status |
| 47 | UB | Ethernet Handle F Status |
| 48 | UB | Ethernet Handle G Status |
| 49 | UB | Ethernet Handle H Status |
| 50 | UB | error code |
| 51 | UB | thread status – see bit field map below |

| | | |
|---|---|---|
| 52-55 | UL (new) | Amplifier Status |
| 56-59 | UL (new) | Segment Count for Contour Mode |
| 60-61 | UW (new) | Buffer space remaining – Contour Mode |
| 62-63 | UW | segment count of coordinated move for S plane |
| 64-65 | UW | coordinated move status for S plane – see bit field map below |
| 66-69 | SL | distance traveled in coordinated move for S plane |
| 70-71 | UW (new) | Buffer space remaining – S Plane |
| 72-73 | UW | segment count of coordinated move for T plane |
| 74-75 | UW | Coordinated move status for T plane – see bit field map below |
| 76-79 | SL | distance traveled in coordinated move for T plane |
| 80-81 | UW (new) | Buffer space remaining – T Plane |
| | | **Axis information:** |
| 82-83 | UW | A axis status – see bit field map below |
| 84 | UB | A axis switches – see bit field map below |
| 85 | UB | A axis stop code |
| 86-89 | SL | A axis reference position |
| 90-93 | SL | A axis motor position |
| 94-97 | SL | A axis position error |
| 98-101 | SL | A axis auxiliary position |
| 102-105 | SL | A axis velocity |
| 106-109 | SL (new size) | A axis torque |
| 110-111 | SW | A axis analog input |
| 112 | UB(new) | A Hall Input Status |
| 113 | UB | Reserved |
| 114-117 | SL (new) | A User defined variable (ZA) |
| 118-119 | UW | B axis status – see bit field map below |
| 120 | UB | B axis switches – see bit field map below |
| 121 | UB | B axis stop code |
| 122-125 | SL | B axis reference position |
| 126-129 | SL | B axis motor position |
| 130-133 | SL | B axis position error |
| 134-137 | SL | B axis auxiliary position |
| 138-141 | SL | B axis velocity |
| 142-145 | SL (new size) | B axis torque |
| 146-147 | SW | B axis analog input |
| 148 | UB (new) | B Hall Input Status |
| 149 | UB | Reserved |
| 150-153 | SL (new) | B User defined variable (ZA) |
| 154-155 | UW | C axis status – see bit field map below |
| 156 | UB | C axis switches – see bit field map below |
| 157 | UB | C axis stop code |
| 158-161 | SL | C axis reference position |
| 162-165 | SL | C axis motor position |
| 166-169 | SL | C axis position error |

| | | |
|---|---|---|
| 170-173 | SL | C axis auxiliary position |
| 174-177 | SL | C axis velocity |
| 178-181 | SL (new size) | C axis torque |
| 182-183 | SW | C axis analog input |
| 184 | UB (new) | C Hall Input Status |
| 185 | UB | Reserved |
| 186-189 | SL (new) | C User defined variable (ZA) |
| 190-191 | UW | D axis status – see bit field map below |
| 192 | UB | D axis switches – see bit field map below |
| 193 | UB | D axis stop code |
| 194-197 | SL | D axis reference position |
| 198-201 | SL | D axis motor position |
| 202-205 | SL | D axis position error |
| 206-209 | SL | D axis auxiliary position |
| 210-213 | SL | D axis velocity |
| 214-217 | SL (new size) | D axis torque |
| 218-219 | SW | D axis analog input |
| 220 | UB (new) | D Hall Input Status |
| 221 | UB | Reserved |
| 222-225 | SL (new) | D User defined variable (ZA) |
| 226-227 | UW | E axis status – see bit field map below |
| 228 | UB | E axis switches – see bit field map below |
| 229 | UB | E axis stop code |
| 230-233 | SL | E axis reference position |
| 234-237 | SL | E axis motor position |
| 238-241 | SL | E axis position error |
| 242-245 | SL | E axis auxiliary position |
| 246-249 | SL | E axis velocity |
| 250-253 | SL (new size) | E axis torque |
| 254-255 | SW | E axis analog input |
| 256 | UB (new) | E Hall Input Status |
| 257 | UB | Reserved |
| 258-261 | SL (new) | E User defined variable (ZA) |
| 262-263 | UW | F axis status – see bit field map below |
| 264 | UB | F axis switches – see bit field map below |
| 265 | UB | F axis stop code |
| 266-269 | SL | F axis reference position |
| 270-273 | SL | F axis motor position |
| 274-277 | SL | F axis position error |
| 278-281 | SL | F axis auxiliary position |
| 282-285 | SL | F axis velocity |
| 286-289 | SL (new size) | F axis torque |
| 290-291 | SW | F axis analog input |
| 292 | UB (new) | F Hall Input Status |
| 293 | UB | Reserved |

| | | |
|---|---|---|
| 294-297 | SL (new) | F User defined variable (ZA) |
| 298-299 | UW | G axis status – see bit field map below |
| 300 | UB | G axis switches – see bit field map below |
| 301 | UB | G axis stop code |
| 302-305 | SL | G axis reference position |
| 306-309 | SL | G axis motor position |
| 310-313 | SL | G axis position error |
| 314-317 | SL | G axis auxiliary position |
| 318-321 | SL | G axis velocity |
| 322-325 | SL (new size) | G axis torque |
| 326-327 | SW | G axis analog input |
| 328 | UB (new) | G Hall Input Status |
| 329 | UB | Reserved |
| 330-333 | SL (new) | G User defined variable (ZA) |
| 334-335 | UW | H axis status – see bit field map below |
| 336 | UB | H axis switches – see bit field map below |
| 337 | UB | H axis stop code |
| 338-341 | SL | H axis reference position |
| 342-345 | SL | H axis motor position |
| 346-349 | SL | H axis position error |
| 350-353 | SL | H axis auxiliary position |
| 354-357 | SL | H axis velocity |
| 358-361 | SL (new size) | H axis torque |
| 362-363 | SW | H axis analog input |
| 364 | UB (new) | H Hall Input Status |
| 365 | UB | Reserved |
| 366-369 | SL (new) | H User defined variable (ZA) |

## Explanation Data Record Bit Fields

### Header Information - Byte 0, 1 of Header:

| BIT 15 | BIT 14 | BIT 13 | BIT 12 | BIT 11 | BIT 10 | BIT 9 | BIT 8 |
|---|---|---|---|---|---|---|---|
| 1 | N/A | N/A | N/A | N/A | I Block Present in Data Record | T Block Present in Data Record | S Block Present in Data Record |
| BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 |
| H Block Present in Data Record | G Block Present in Data Record | F Block Present in Data Record | E Block Present in Data Record | D Block Present in Data Record | C Block Present in Data Record | B Block Present in Data Record | A Block Present in Data Record |

### Bytes 2, 3 of Header:

Bytes 2 and 3 make a word which represents the Number of bytes in the data record, including the header.

Byte 2 is the low byte and byte 3 is the high byte

**NOTE:** The header information of the data records is formatted in little endian.

### Thread Status (1 Byte)

| BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 |
|---|---|---|---|---|---|---|---|
| Thread 7 Running | Thread 6 Running | Thread 5 Running | Thread 4 Running | Thread 3 Running | Thread 2 Running | Thread 1 Running | Thread 0 Running |

### Coordinated Motion Status for S or T Plane (2 Byte)

| BIT 15 | BIT 14 | BIT 13 | BIT 12 | BIT 11 | BIT 10 | BIT 9 | BIT 8 |
|---|---|---|---|---|---|---|---|
| Move in Progress | N/A | N/A | N/A | N/A | N/A | N/A | N/A |

| BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 |
|---|---|---|---|---|---|---|---|
| N/A | N/A | Motion is slewing | Motion is stopping due to ST or Limit Switch | Motion is making final decel. | N/A | N/A | N/A |

### Axis Status (1 Word)

| BIT 15 | BIT 14 | BIT 13 | BIT 12 | BIT 11 | BIT 10 | BIT 9 | BIT 8 |
|---|---|---|---|---|---|---|---|
| Move in Progress | Mode of Motion PA or PR | Mode of Motion PA only | (FE) Find Edge in Progress | Home (HM) in Progress | 1st Phase of HM complete | 2nd Phase of HM complete or FI command issued | Mode of Motion Coord. Motion |

| BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 |
|---|---|---|---|---|---|---|---|
| Negative Direction Move | Mode of Motion<br><br>Contour | Motion is slewing | Motion is stopping due to ST of Limit Switch | Motion is making final decel. | Latch is armed | 3rd Phase of HM in Progress | Motor Off |

### Axis Switches (1 Byte)

| BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 |
|---|---|---|---|---|---|---|---|
| Latch Occurred | State of Latch Input | N/A | N/A | State of Forward Limit | State of Reverse Limit | State of Home Input | Stepper Mode |

## Notes Regarding Velocity and Torque Information

The velocity information that is returned in the data record is 64 times larger than the value returned when using the command TV (Tell Velocity). See command reference for more information about TV.

The Torque information is represented as a number in the range of +/-32767. Maximum negative torque is -32767. Maximum positive torque is 32767. Zero torque is 0.

## QZ Command

The QZ command can be very useful when using the QR command, since it provides information about the controller and the data record. The QZ command returns the following 4 bytes of information.

| BYTE # | INFORMATION |
|---|---|
| 0 | Number of axes present |
| 1 | number of bytes in general block of data record |
| 2 | number of bytes in coordinate plane block of data record |
| 3 | Number of Bytes in each axis block of data record |

# Controller Response to Commands

Most DMC-40x0 instructions are represented by two characters followed by the appropriate parameters. Each instruction must be terminated by a carriage return or semicolon.

Instructions are sent in ASCII, and the DMC-40x0 decodes each ASCII character (one byte) one at a time. It takes approximately 0.05 msec for the controller to decode each command.

After the instruction is decoded, the DMC-40x0 returns a response to the port from which the command was generated. If the instruction was valid, the controller returns a colon (:) or a question mark (?) if the instruction was not valid. For example, the controller will respond to commands which are sent via the main RS-232 port back through the RS-232 port, and to commands which are sent via the Ethernet port back through the Ethernet port.

For instructions that return data, such as Tell Position (TP), the DMC-40x0 will return the data followed by a carriage return, line feed and : .

It is good practice to check for : after each command is sent to prevent errors. An echo function is provided to enable associating the DMC-40x0 response with the data sent. The echo is enabled by sending the command EO 1 to the controller.

# Unsolicited Messages Generated by Controller

When the controller is executing a program, it may generate responses which will be sent via the main RS-232 port or Ethernet ports. This response could be generated as a result of messages using the MG command OR as a result of a command error. These responses are known as unsolicited messages since they are not generated as the direct response to a command.

Messages can be directed to a specific port using the specific Port arguments – see the MG and CF commands in the Command Reference. If the port is not explicitly given or the default is not changed with the CF command, unsolicited messages will be sent to the default port. The default port is the main serial port.

The controller has a special command, CW, which can affect the format of unsolicited messages. This command is used by Galil Software to differentiate response from the command line and unsolicited messages. The command, CW1 causes the controller to set the high bit of ASCII characters to 1 of all unsolicited characters. This may cause characters to appear garbled to some terminals. This function can be disabled by issuing the command, CW2. For more information, see the CW command in the Command Reference.

When handshaking is used (hardware and/or software handshaking) characters which are generated by the controller are placed in a FIFO buffer before they are sent out of the controller. The size of the RS-232 buffer is 512 bytes. When this buffer becomes full, the controller must either stop executing commands or ignore additional characters generated for output. The command CW,1 causes the controller to ignore all output from the controller while the FIFO is full. The command, CW ,0 causes the controller to stop executing new commands until more room is made available in the FIFO. This command can be very useful when hardware handshaking is being used and the communication line between controller and terminal will be disconnected. In this case, characters will continue to build up in the controller until the FIFO is full. For more information, see the CW command in the Command Reference.

# GalilTools (Windows and Linux)

GalilTools is Galil's set of software tools for current Galil controllers. It is highly recommended for all first-time purchases of Galil controllers as it provides easy set-up, tuning and analysis. GalilTools replaces the WSDK Tuning software with an improved user-interface, real-time scopes and communications utilities.

The Galil Tools set contains the following tools: Scope, Editor, Terminal, Watch and Tuner, and a Communication Library for development with Galil Controllers.

The powerful Scope Tool is ideal for system analysis as it captures numerous types of data for each axis in real-time. Up to eight channels of data can be displayed at once, and additional real-time data can be viewed by changing the scope settings. This allows literally hundreds of parameters to be analyzed during a single data capture sequence. A rising or falling edge trigger feature is also included for precise synchronization of data.

The Program Editor Tool allows for easy writing of application programs and multiple editors to be open simultaneously.

The Terminal Tool provides a window for sending and receiving Galil commands and responses.

The Watch Tool displays controller parameters in a tabular format and includes units and scale factors for easy viewing.

The Tuning Tool helps select PID parameters for optimal servo performance.

The Communication Library provides function calls for communicating to Galil Controllers with C++ (Windows and Linux) and COM enabled languages such as VB C#, and Labview (Windows only).

GalilTools runs on Windows and Linux platforms as standard with other platforms available on request.

GalilTools-Lite is available at no charge and contains the Editor, Terminal, Watch and Communcition Library tools only.

The latest version of GalilTools can be downloaded from the Galil website at:

http://www.galilmc.com/products/software/galiltools.html

For information on using GalilTools see the help menu in GalilTools, or the GalilTools user manual.

http://www.galilmc.com/support/manuals/galiltools/index.html

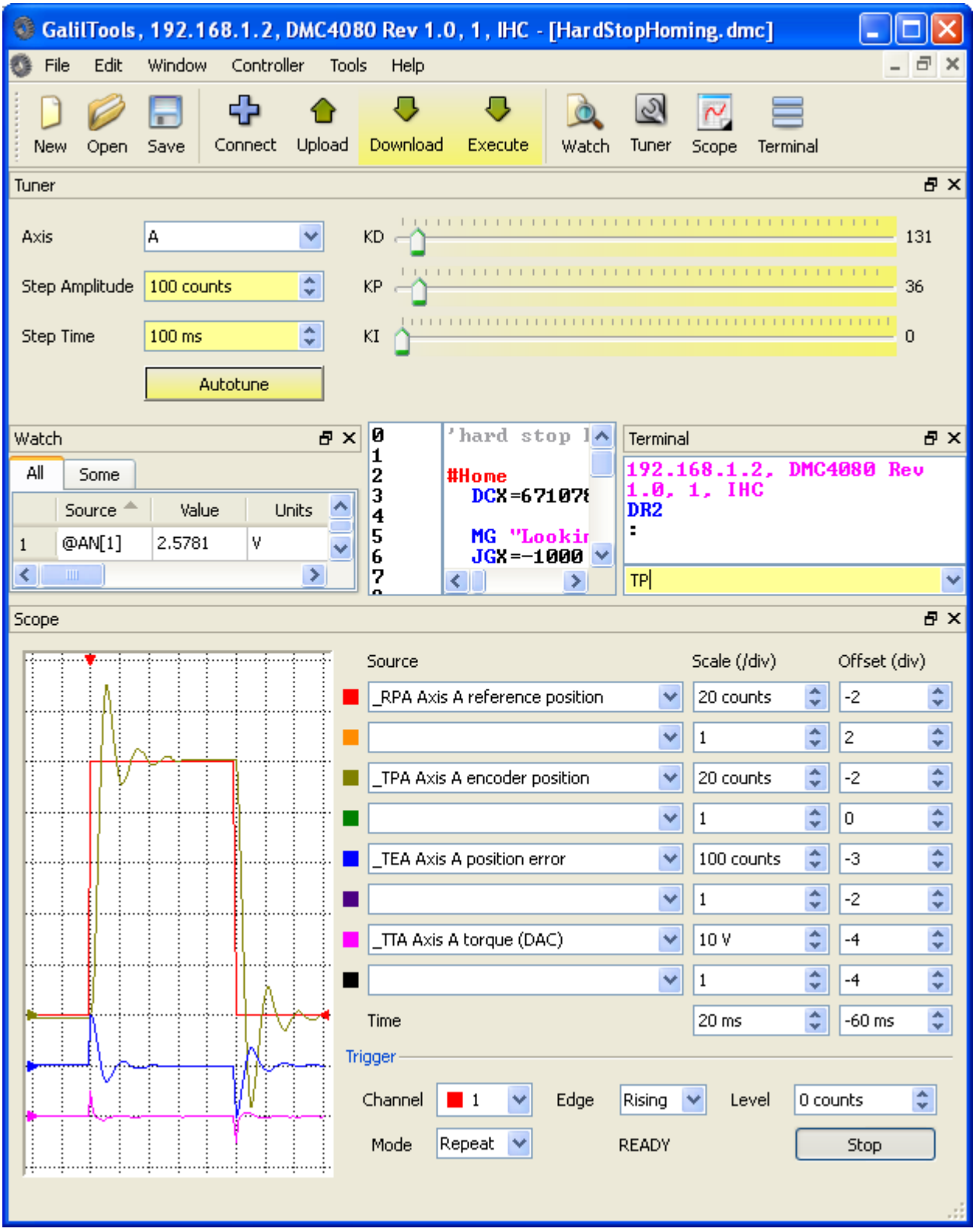



*Figure 4.1 - GalilTools*

# Creating Custom Software Interfaces

Galil provides programming tools so that users can develop their own custom software interfaces to a Galil controller. These tools include the GalilTools Communication Library, ActiveX Toolkit, .NET API and DMCWin.

For new applications, Galil recommends the GalilTools Communication Libraries.

## HelloGalil – Quick Start to PC programming

For programmers developing Windows applications that communicate with a Galil controller, the HelloGalil library of quick start projects immediately gets you communicating with the controller from the programming language of your choice. In the "Hello World" tradition, each project contains the bare minimum code to demonstrate communication to the controller and simply prints the controller's model and serial numbers to the screen:

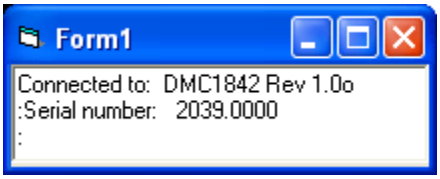



*Figure 4.2. Sample program output*

http://www.galilmc.com/support/hello_galil.html

## GalilTools Communication Libraries

The GalilTools Communication Library (Galil class) provides methods for communication with a Galil motion controller over Ethernet, RS-232 or PCI buses. It consists of a native C++ Library and a similar COM interface which extends compatibility to Windows programming languages (e.g. VB, C#, etc).

A Galil object (usually referred to in sample code as "g") represents a single connection to a Galil controller.

For Ethernet controllers, which support more than one connection, multiple objects may be used to communicate with the controller. An example of multiple objects is one Galil object containing a TCP handle to a DMC-40x0 for commands and responses, and one Galil object containing a UDP handle for unsolicited messages from the controller. If recordsStart() is used to begin the automatic data record function, the library will open an additional UDP handle to the controller (transparent to the user).

The library is conceptually divided into six categories:

1. Connecting and Disconnecting - functions to establish and discontinue communication with a controller.
2. Basic Communication - The most heavily used functions for command-and-response and unsolicited messages.
3. Programs - Downloading and uploading embedded programs.
4. Arrays - Downloading and uploading array data.
5. Advanced - Lesser-used calls.
6. Data Record - Access to the data record in both synchronous and asynchronous modes.

### C++ Library (Windows and Linux)

Both Full and Lite versions of GalilTools ship with a native C++ communication library. The Linux version (libGalil.so) is compatible with g++ and the Windows version (Galil1.dll) with Visual C++ 2008. Contact Galil if another version of the C++ library is required. See the getting started guide and the hello.cpp example in /lib.

### COM (Windows)

To further extend the language compatibility on Windows, a COM (Component Object Model) class built on top of the C++ library is also provided with Windows releases. This COM wrapper can be used in any language and IDE supporting COM (Visual Studio 2005, 2008, etc). The COM wrapper includes all of the functionality of the base C++ class. See the getting started guide and the hello.* examples in \lib for more info.

For more information on the GalilTools Communications Library, see the online user manual.

http://www.galilmc.com/support/manuals/galiltools/library.html

## ActiveX Toolkit

**Galil recommends the GalilTools Communication Library for all new applications.**

Galil's ActiveX Toolkit is useful for the programmer who wants to easily create a custom operator interface to a Galil controller. The ActiveX Toolkit includes a collection of ready-made ActiveX COM controls for use with Visual Basic, Visual C++, Delphi, LabVIEW and other ActiveX compatible programming tools. The most common environment is Visual Basic 6, but Visual Basic.NET, Visual C++, Wonderware, LabVIEW and HPVEE have all been tested by Galil to work with the .OCX controls.

The ActiveX Toolkit can be purchased from Galil at:

http://www.galilmc.com/buy/index.html

The ActiveX toolkit can save many hours of programming time. Built-in dialog boxes are provided for quick parameter setup, selection of color, size, location and text. The toolkit controls are easy to use and provide context sensitive help, making it ideal for even the novice programmer.

ActiveX Toolkit Includes:

- a terminal control for sending commands and editing programs
- a polling window for displaying responses from the controller such as position and speed
- a storage scope control for plotting real time trajectories such as position versus time or X versus Y
- a send file control for sending contour data or vector DMC files
- a continuous array capture control for data collection, and for teach and playback
- a graphical display control for monitoring a 2-D motion path
- a diagnostics control for capturing current configurations
- a display control for input and output status
- a vector motion control for tool offsets and corner speed control

For more detailed information on the ActiveX Toolkit, please refer to the user manual at

http://www.galilmc.com/support/manuals/activex.pdf.

## DMCWin Programmers Toolkit

**Galil recommends the GalilTools Communication Library for all new applications.**

DMCWin is a programmer's toolkit for C/C++ and Visual Basic users. The toolkit includes header files for the Galil communications API, as well as source code and examples for developing Windows® programs that communicate to Galil Controllers. The Galil communications API includes functions to send commands, download programs, download/upload arrays, access the data record, etc. For a complete list of all the functions, refer to the DMCWin user manual at:

http://www.galilmc.com/support/manuals/dmcwin.pdf.

This software package is free for download and is available at:

http://www.galilmc.com/support/download.html

## Galil Communications API with C/C++

**Galil recommends the GalilTools Communication Library for all new applications.**

When programming in C/C++, the communications API can be used as included functions or through a class library. All Galil communications programs written in C must include the DMCCOM.H file and access the API functions through the declared routine calls. C++ programs can use the DMCCOM.H routines or use the class library defined in DMCWIN.H.

After installing DMCWin into the default directory, the DMCCOM.H header file is located in C:\Program Files\Galil\DMCWIN\INCLUDE. C++ programs that use the class library need the files DMCWIN.H and DMCWIN.CPP, which contain the class definitions and implementations respectively. These can be found in the C:\ProgramFiles\Galil\DMCWIN\CPP directory.

To link the application with the DLL's, the DMC32.lib file must be included in the project and is located at C:\Program Files\Galil\DMCWIN\LIB

**Example: A simple console application that sends commands to the controller**

To initiate communication, declare a variable of type HANDLEDMC (a long integer) and pass the address of that variable in the DMCOpen() function. If the DMCOpen() function is successful, the variable will contain the handle to the Galil controller, which is required for all subsequent function calls. The following simple example program written as a Visual C console application tells the controller to move the X axis 1000 encoder counts. Remember to add DMC32.LIB to your project prior to compiling.

```
#include <windows.h>
#include <dmccom.h>
long lRetCode;
HANDLEDMC hDmc;
HWND hWnd;
int main(void)
{
      // Connect to controller number 1
      lRetCode= DMCOpen(1, hWnd, &hDmc);
      if (rc == DMCNOERROR)
      {
      char szBuffer[64];
      // Move the X axis 1000 counts
      lRetCode = DMCCommand(hDmc, "PR1000;BGX;", szBuffer,
     sizeof(szBuffer));
     // Disconnect from controller number 1 as the last action
      lRetCode = DMCClose(hDmc);
      }
     return 0;
}
```

## Galil Communications API with Visual Basic

### Declare Functions

**Galil recommends the GalilTools Communication Library for all new applications.**

To use the Galil communications API functions, add the module file included in the C:\ProgramFiles\Galil\DMCWIN\VB directory named DMCCOM40.BAS. This module declares the routines making them available for the VB project. To add this file, select 'Add Module' from the 'Project' menu in VB5/6.

### Sending Commands in VB

Most commands are sent to the controller with the DMCCommand() function. This function allows any Galil command to be sent from VB to the controller. The DMCCommand() function will return the response from the controller as a string. Before sending any commands the DMCOpen() function must be called. This function establishes communication with the controller and is called only once.

This example code illustrates the use of DMCOpen() and DMCCommand(). A connection is made to controller #1 in the Galil registry upon launching the application. Then, the controller is sent the command 'TPX' whenever a command button is pressed. The response is then placed in a text box. When the application is closed, the controller is disconnected.

To use this example, start a new Visual Basic project, place a Text Box and a Command Button on a Form, add the DMCCOM40.BAS module, and type the following code:

```
Dim m_nController As Integer
Dim m_hDmc As Long
Dim m_nRetCode As Long
Dim m_nResponseLength As Long
Dim m_sResponse As String * 256

Private Sub Command1_Click()
   m_nRetCode = DMCCommand(m_hDmc, "TPX", m_sResponse, m_nResponseLength)
   Text1.Text = Val(m_sResponse)
End Sub

Private Sub Form_Load()
   m_nResponseLength = 256
   m_nController = 1
   m_nRetCode = DMCOpen(m_nController, 0, m_hDmc)
End Sub

Private Sub Form_Unload(Cancel As Integer)
   m_nRetCode = DMCClose(m_hDmc)
End Sub
```

Where:
'm_nController' is the number for the controller in the Galil registry.
'm_hDmc' is the DMC handle used to identify the controller. It is returned by DMCOpen.
'm_nRetCode' is the return code for the routine.
'm_nResponseLength' is the response string length which must be set to the size of the response string.
'm_sResponse' is the string containing the controller response to the command.

# DOS, and QNX tools

Galil offers unsupported code examples that demonstrate communications to the controller using the following operating systems.

### DOS

DOS based utilities & Programming Libraries for Galil controllers, which includes a terminal, utilities to upload and download programs, and source code for BASIC and C programs. Download DMCDOS at :

http://www.galilmc.com/support/download.html#dos.

### QNX

Galil offers sample drivers for ISA and PCI cards for the QNX 4.24 operating system. We also offer drivers and utilities for QNX 6.2 for PCI only. Download at:

http://www.galilmc.com/support/download.html#linux.

### Linux

Galil now offers full support to Linux users through the GalilTools software package. For more information see the previous section on GalilTools, or visit the Galil website.

http://www.galilmc.com/products/software/galiltools.html

# Chapter 5 Command Basics

## Introduction

The DMC-40x0 provides over 100 commands for specifying motion and machine parameters. Commands are included to initiate action, interrogate status and configure the digital filter. These commands can be sent in ASCII or binary.

In ASCII, the DMC-40x0 instruction set is BASIC-like and easy to use. Instructions consist of two uppercase letters that correspond phonetically with the appropriate function. For example, the instruction BG begins motion, and ST stops the motion. In binary, commands are represented by a binary code ranging from 80 to FF.

ASCII commands can be sent "live" over the communications port for immediate execution by the DMC-40x0, or an entire group of commands can be downloaded into the DMC-40x0 memory for execution at a later time. Combining commands into groups for later execution is referred to as Applications Programming and is discussed in the following chapter. Binary commands cannot be used in Applications programming.

This section describes the DMC-40x0 instruction set and syntax. A summary of commands as well as a complete listing of all DMC-40x0 instructions is included in the *Command Reference*.

## Command Syntax - ASCII

DMC-40x0 instructions are represented by two ASCII upper case characters followed by applicable arguments. A space may be inserted between the instruction and arguments. A semicolon or <return> is used to terminate the instruction for processing by the DMC-40x0 command interpreter.

**NOTE:** If you are using a Galil terminal program, commands will not be processed until an <return> command is given. This allows the user to separate many commands on a single line and not begin execution until the user gives the <return> command.

**IMPORTANT: All DMC-40x0 commands are sent in upper case.**

*For example, the command*

PR 4000 <return> Position relative

PR is the two character instruction for position relative. 4000 is the argument which represents the required position value in counts. The <return> terminates the instruction. The space between PR and 4000 is optional.

For specifying data for the A,B,C and D axes, commas are used to separate the axes. If no data is specified for an axis, a comma is still needed as shown in the examples below. If no data is specified for an axis, the previous value is maintained.

To view the current values for each command, type the command followed by a ? for each axis requested.

```
PR 1000                              Specify A only as 1000
PR ,2000                             Specify B only as 2000
PR ,,3000                            Specify C only as 3000
PR ,,,4000                           Specify D only as 4000
PR 2000, 4000,6000, 8000             Specify A,B,C and D
PR ,8000,,9000                       Specify B and D only
PR ?,?,?,?                           Request A,B,C,D values
PR ,?                                Request B value only
```

The DMC-40x0 provides an alternative method for specifying data. Here data is specified individually using a single axis specifier such as A, B, C or D. An equals sign is used to assign data to that axis. For example:

```
PRA=1000            Specify a position relative movement for the A axis of
                    1000
ACB=200000          Specify acceleration for the B axis as 200000
```

Instead of data, some commands request action to occur on an axis or group of axes. For example, ST AB stops motion on both the A and B axes. Commas are not required in this case since the particular axis is specified by the appropriate letter A, B, C or D. If no parameters follow the instruction, action will take place on all axes. Here are some examples of syntax for requesting action:

```
BG A                Begin A only
BG B                Begin B only
BG ABCD             Begin all axes
BG BD               Begin B and D only
BG                  Begin all axes
```

**4080**

For controllers with 5 or more axes, the axes are referred to as A,B,C,D,E,F,G,H. The specifiers X,Y,Z,W and A,B,C,D may be used interchangeably.

```
BG ABCDEFGH         Begin all axes
BG D                Begin D only
```

## Coordinated Motion with more than 1 axis

When requesting action for coordinated motion, the letter S or T is used to specify the coordinated motion. This allows for coordinated motion to be setup for two separate coordinate systems. Refer to the CA command in the Command Reference for more information on specifying a coordinate system. For example:

```
BG S                Begin coordinated sequence, S
BG TW               Begin coordinated sequence, T, and D axis
```

# Command Syntax – Binary (advanced)

Some commands have an equivalent binary value. Binary communication mode can be executed about 20% faster than ASCII commands. Binary format can only be used when commands are sent from the PC and cannot be embedded in an application program.

## Binary Command Format

All binary commands have a 4 byte header and is followed by data fields. The 4 bytes are specified in hexadecimal format.

### Header Format:

**Byte 1** specifies the command number between 80 to FF. The complete binary command number table is listed below.

**Byte 2** specifies the # of bytes in each field as 0,1,2,4 or 6 as follows:

| | |
|---|---|
| 00 | No data fields (i.e. SH or BG) |
| 01 | One byte per field |
| 02 | One word (2 bytes per field) |
| 04 | One long word (4 bytes) per field |
| 06 | Galil real format (4 bytes integer and 2 bytes fraction) |

**Byte 3** specifies whether the command applies to a coordinated move as follows:

| | |
|---|---|
| 00 | No coordinated motion movement |
| 01 | Coordinated motion movement |

For example, the command STS designates motion to stop on a vector move, S coordinate system. The third byte for the equivalent binary command would be 01.

**Byte 4** specifies the axis # or data field as follows

Bit 7 = H axis or 8th data field

Bit 6 = G axis or 7th data field

Bit 5 = F axis or 6th data field

Bit 4 = E axis or 5th data field

Bit 3 = D axis or 4th data field

Bit 2 = C axis or 3rd data field

Bit 1 = B axis or 2nd data field

Bit 0 = A axis or 1st data field

### Data fields Format

Data fields must be consistent with the format byte and the axes byte. For example, the command PR 1000,, -500 would be :

```
A7 02 00 05 03 E8 FE 0C
```

where A7 is the command number for PR

02 specifies 2 bytes for each data field

00 S is not active for PR

05 specifies bit 0 is active for A axis and bit 2 is active for C axis ($2^0 + 2^2=5$)

03 E8 represents 1000

FE OC represents -500

### Example

The command ST XYZS would be :

```
A1 00 01 07
```

where A1 is the command number for ST

00 specifies 0 data fields

01 specifies stop the coordinated axes S

07 specifies stop X (bit 0), Y (bit 1) and Z (bit 2) $2^0+2^1+2^3 =7$

## Binary command table

| Command | No. | Command | No. | Command | No. |
|---|---|---|---|---|---|
| | | | | | |
| reserved | 80 | reserved | ab | reserved | d6 |
| KP | 81 | reserved | ac | reserved | d7 |
| KI | 82 | reserved | ad | RP | d8 |
| KD | 83 | reserved | ae | TP | d9 |
| DV | 84 | reserved | af | TE | da |
| AF | 85 | LM | b0 | TD | db |
| KS | 86 | LI | b1 | TV | dc |
| PL | 87 | VP | b2 | RL | dd |
| ER | 88 | CR | b3 | TT | de |
| IL | 89 | TN | b4 | TS | df |
| TL | 8a | LE, VE | b5 | TI | e0 |
| MT | 8b | reserved | b6 | SC | e1 |
| CE | 8c | VA | b7 | reserved | e2 |
| OE | 8d | VD | b8 | reserved | e3 |
| FL | 8e | VS | b9 | reserved | e4 |
| BL | 8f | VR | ba | TM | e5 |
| AC | 90 | reserved | bb | CN | e6 |
| DC | 91 | reserved | bc | LZ | e7 |
| SP | 92 | CM | bd | OP | e8 |
| IT | 93 | CD | be | OB | e9 |
| FA | 94 | DT | bf | SB | ea |
| FV | 95 | ET | c0 | CB | eb |
| GR | 96 | EM | c1 | II | ec |
| DP | 97 | EP | c2 | EI | ed |

| | | | | | |
|---|---|---|---|---|---|
| DE | 98 | EG | c3 | AL | ee |
| OF | 99 | EB | c4 | reserved | ef |
| GM | 9a | EQ | c5 | reserved | f0 |
| reserved | 9b | EC | c6 | reserved | f1 |
| reserved | 9c | reserved | c7 | reserved | f2 |
| reserved | 9d | AM | c8 | reserved | f3 |
| reserved | 9e | MC | c9 | reserved | f4 |
| reserved | 9f | TW | ca | reserved | f5 |
| BG | a0 | MF | cb | reserved | f6 |
| ST | a1 | MR | cc | reserved | f7 |
| AB | a2 | AD | cd | reserved | f8 |
| HM | a3 | AP | ce | reserved | f9 |
| FE | a4 | AR | cf | reserved | fa |
| FI | a5 | AS | d0 | reserved | fb |
| PA | a6 | AI | d1 | reserved | fc |
| PR | a7 | AT | d2 | reserved | fd |
| JG | a8 | WT | d3 | reserved | fe |
| MO | a9 | reserved | d4 | reserved | ff |
| SH | aa | reserved | d5 | | |

# Controller Response to DATA

The DMC-40x0 returns a : for valid commands and a ? for invalid commands.

For example, if the command BG is sent in lower case, the DMC-40x0 will return a ?.

| | |
|---|---|
| :bg <return> | invalid command, lower case |
| ? | DMC-40x0 returns a ? |

When the controller receives an invalid command the user can request the error code. The error code will specify the reason for the invalid command response. To request the error code type the command TC1. For example:

| | |
|---|---|
| ?TC1 <return> | Tell Code command |
| 1 Unrecognized | Returned response |

There are many reasons for receiving an invalid command response. The most common reasons are: unrecognized command (such as typographical entry or lower case), command given at improper time (such as during motion), or a command out of range (such as exceeding maximum speed). A complete listing of all codes is listed in the TC command in the Command Reference section.

# Interrogating the Controller

## Interrogation Commands

The DMC-40x0 has a set of commands that directly interrogate the controller. When the command is entered, the requested data is returned in decimal format on the next line followed by a carriage return and line feed. The format of the returned data can be changed using the Position Format (PF), Variable Format (VF) and Leading Zeros (LZ) command. See Chapter 7 and the Command Reference.

## Summary of Interrogation Commands

| | |
|---|---|
| RP | Report Command Position |
| RL | Report Latch |
| ^R ^V | Firmware Revision Information |
| SC | Stop Code |
| TB | Tell Status |
| TC | Tell Error Code |
| TD | Tell Dual Encoder |
| TE | Tell Error |
| TI | Tell Input |
| TP | Tell Position |
| TR | Trace |
| TS | Tell Switches |
| TT | Tell Torque |
| TV | Tell Velocity |

For example, the following example illustrates how to display the current position of the X axis:

```
TP A <return>              Tell position A
0                          Controllers Response
TP AB <return>             Tell position A and B
0,0                        Controllers Response
```

## Interrogating Current Commanded Values.

Most commands can be interrogated by using a question mark (?) as the axis specifier. Type the command followed by a ? for each axis requested.

```
PR ?,?,?,?                 Request A,B,C,D values
PR ,?                      Request B value only
```

The controller can also be interrogated with operands.

## Operands

Most DMC-40x0 commands have corresponding operands that can be used for interrogation. Operands must be used inside of valid DMC expressions. For example, to display the value of an operand, the user could use the command:

MG ‘operand’ where ‘operand’ is a valid DMC operand

All of the command operands begin with the underscore character (_). For example, the value of the current position on the A axis can be assigned to the variable ‘V’ with the command:

V=_TPA

The Command Reference denotes all commands which have an equivalent operand as "Used as an Operand". Also, see description of operands in Chapter 7.

## Command Summary

For a complete command summary, see *Command Reference* manual.

# Chapter 6 Programming Motion

## Overview

The DMC-40x0 provides several modes of motion, including independent positioning and jogging, coordinated motion, electronic cam motion, and electronic gearing. Each one of these modes is discussed in the following sections.

The DMC-4010 are single axis controllers and use X-axis motion only. Likewise, the DMC-4020 use X and Y, the DMC-4030 use X,Y, and Z, and the DMC-4040 use X,Y,Z, and W. The DMC-4050 use A,B,C,D, and E. The DMC-4060 use A,B,C,D,E, and F. The DMC-4070 use A,B,C,D,E,F, and G. The DMC-4080 use the axes A,B,C,D,E,F,G, and H.

The example applications described below will help guide you to the appropriate mode of motion.

**4080** For controllers with 5 or more axes, the specifiers, ABCDEFGH, are used. XYZ and W may be interchanged with ABCD.

| EXAMPLE APPLICATION | MODE OF MOTION | COMMANDS |
|---|---|---|
| Absolute or relative positioning where each axis is independent and follows prescribed velocity profile. | Independent Axis Positioning | PA,PR<br>SP,AC,DC |
| Velocity control where no final endpoint is prescribed. Motion stops on Stop command. | Independent Jogging | JG<br>AC,DC<br>ST |
| Absolute positioning mode where absolute position targets may be sent to the controller while the axis is in motion. | Position Tracking | PA,<br>PT<br>SP<br>AC, DC |
| Motion Path described as incremental position points versus time. | Contour Mode | CM<br>CD<br>DT |
| 2,3 or 4 axis coordinated motion where path is described by linear segments. | Linear Interpolation | LM<br>LI, LE<br>VS,VR<br>VA,VD |

| | | |
|---|---|---|
| 2-D motion path consisting of arc segments and linear segments, such as engraving or quilting. | Coordinated Motion | VM<br>VP<br>CR<br>VS,VR<br>VA,VD<br>VE |
| Third axis must remain tangent to 2-D motion path, such as knife cutting. | Coordinated motion with tangent axis specified | VM<br>VP<br>CR<br>VS,VA,VD<br>TN<br>VE |
| Electronic gearing where slave axes are scaled to master axis which can move in both directions. | Electronic Gearing | GA<br>GD<br>_GP<br>GR<br>GM (if gantry) |
| Master/slave where slave axes must follow a master such as conveyer speed. | Electronic Gearing | GA<br>GD<br>_GP<br>GR |
| Moving along arbitrary profiles or mathematically prescribed profiles such as sine or cosine trajectories. | Contour Mode | CM<br>CD<br>DT |
| Teaching or Record and Play Back | Contour Mode with Automatic Array Capture | CM<br>CD<br>DT<br>RA<br>RD<br>RC |
| Backlash Correction | Dual Loop | DV |
| Following a trajectory based on a master encoder position | Electronic Cam | EA<br>EM<br>EP<br>ET<br>EB<br>EG<br>EQ |
| Smooth motion while operating in independent axis positioning | Independent Motion Smoothing | IT |
| Smooth motion while operating in vector or linear interpolation positioning | Vector Smoothing | IT |
| Smooth motion while operating with stepper motors | Stepper Motor Smoothing | KS |
| Gantry - two axes are coupled by gantry | Gantry Mode | GR<br>GM |

# Independent Axis Positioning

In this mode, motion between the specified axes is independent, and each axis follows its own profile. The user specifies the desired absolute position (PA) or relative position (PR), slew speed (SP), acceleration ramp (AC), and deceleration ramp (DC), for each axis. On begin (BG), the DMC-40x0 profiler generates the corresponding trapezoidal or triangular velocity profile and position trajectory. The controller determines a new command position along the trajectory every sample period until the specified profile is complete. Motion is complete when the last position command is sent by the DMC-40x0 profiler. Note: The actual motor motion may not be complete when the profile has been completed, however, the next motion command may be specified.

The Begin (BG) command can be issued for all axes either simultaneously or independently. XYZ or W axis specifiers are required to select the axes for motion. When no axes are specified, this causes motion to begin on all axes.

The speed (SP) and the acceleration (AC) can be changed at any time during motion, however, the deceleration (DC) and position (PR or PA) cannot be changed until motion is complete. Remember, motion is complete when the profiler is finished, not when the actual motor is in position. The Stop command (ST) can be issued at any time to decelerate the motor to a stop before it reaches its final position.

An incremental position movement (IP) may be specified during motion as long as the additional move is in the same direction. Here, the user specifies the desired position increment, n. The new target is equal to the old target plus the increment, n. Upon receiving the IP command, a revised profile will be generated for motion towards the new end position. The IP command does not require a begin. Note: If the motor is not moving, the IP command is equivalent to the PR and BG command combination.

## Command Summary - Independent Axis

| COMMAND | DESCRIPTION |
|---|---|
| PR x,y,z,w | Specifies relative distance |
| PA x,y,z,w | Specifies absolute position |
| SP x,y,z,w | Specifies slew speed |
| AC x,y,z,w | Specifies acceleration rate |
| DC x,y,z,w | Specifies deceleration rate |
| BG XYZW | Starts motion |
| ST XYZW | Stops motion before end of move |
| IP x,y,z,w | Changes position target |
| IT x,y,z,w | Time constant for independent motion smoothing |
| AM XYZW | Trippoint for profiler complete |
| MC XYZW | Trippoint for “in position” |

The lower case specifiers (x,y,z,w) represent position values for each axis.

The DMC-40x0 also allows use of single axis specifiers such as PRY=2000

## Operand Summary - Independent Axis

| OPERAND | DESCRIPTION |
|---|---|
| _ACx | Return acceleration rate for the axis specified by ‘x’ |
| _DCx | Return deceleration rate for the axis specified by ‘x’ |

| | |
|---|---|
| _SPx | Returns the speed for the axis specified by 'x' |
| _PAx | Returns current destination if 'x' axis is moving, otherwise returns the current commanded position if in a move. |
| _PRx | Returns current incremental distance specified for the 'x' axis |

## Example - Absolute Position Movement

```
PA 10000,20000          Specify absolute X,Y position
AC 1000000,1000000      Acceleration for X,Y
DC 1000000,1000000      Deceleration for X,Y
SP 50000,30000          Speeds for X,Y
BG XY                   Begin motion
```

## Example - Multiple Move Sequence

Required Motion Profiles:

| | | |
|---|---|---|
| X-Axis | 500 counts | Position |
| | 20000 count/sec | Speed |
| | 500000 counts/sec$^2$ | Acceleration |
| Y-Axis | 1000 counts | Position |
| | 10000 count/sec | Speed |
| | 500000 counts/sec$^2$ | Acceleration |
| Z-Axis | 100 counts | Position |
| | 5000 counts/sec | Speed |
| | 500000 counts/sec | Acceleration |

This example will specify a relative position movement on X, Y and Z axes. The movement on each axis will be separated by 20 msec. Fig. 6.0 shows the velocity profiles for the X,Y and Z axis.

```
#A                      Begin Program
PR 2000,500,100         Specify relative position movement of 1000, 500 and 100 counts for X,Y and Z axes.
SP 20000,10000,5000     Specify speed of 20000, 10000, and 5000 counts / sec
AC 500000,500000,500000 Specify acceleration of 500000 counts / sec^2 for all axes
DC 500000,500000,500000 Specify deceleration of 500000 counts / sec^2 for all axes
BG X                    Begin motion on the X axis
WT 20                   Wait 20 msec
BG Y                    Begin motion on the Y axis
WT 20                   Wait 20 msec
BG Z                    Begin motion on Z axis
EN                      End Program
```

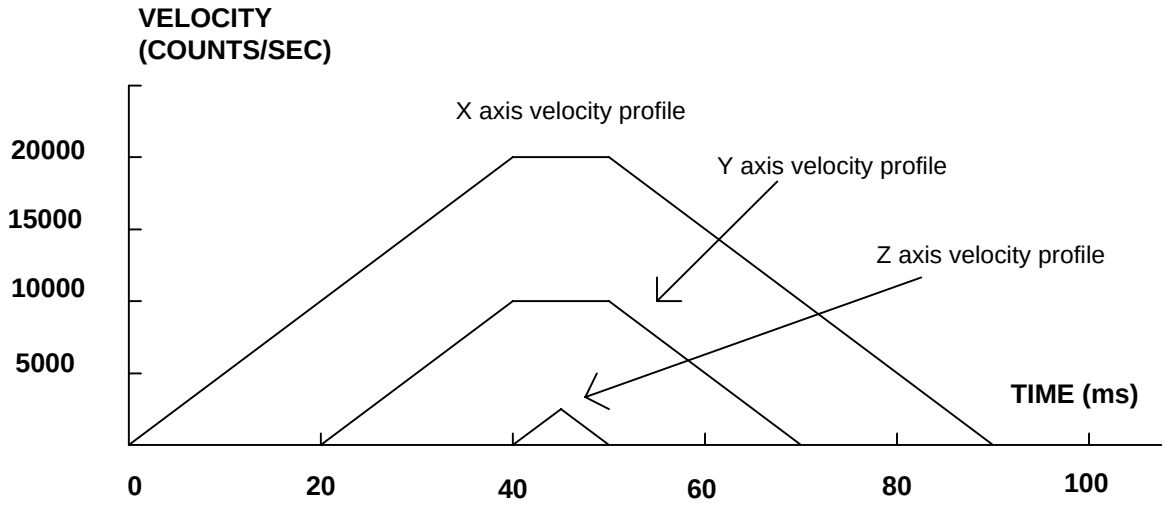



*Figure 6.0 - Velocity Profiles of XYZ*

**Notes on figure 6.0:** The X and Y axis have a 'trapezoidal' velocity profile, while the Z axis has a 'triangular' velocity profile. The X and Y axes accelerate to the specified speed, move at this constant speed, and then decelerate such that the final position agrees with the command position, PR. The Z axis accelerates, but before the specified speed is achieved, must begin deceleration such that the axis will stop at the commanded position. All 3 axes have the same acceleration and deceleration rate, hence, the slope of the rising and falling edges of all 3 velocity profiles are the same.

# Independent Jogging

The jog mode of motion is very flexible because speed, direction and acceleration can be changed during motion. The user specifies the jog speed (JG), acceleration (AC), and the deceleration (DC) rate for each axis. The direction of motion is specified by the sign of the JG parameters. When the begin command is given (BG), the motor accelerates up to speed and continues to jog at that speed until a new speed or stop (ST) command is issued. If the jog speed is changed during motion, the controller will make a accelerated (or decelerated) change to the new speed.

An instant change to the motor position can be made with the use of the IP command. Upon receiving this command, the controller commands the motor to a position which is equal to the specified increment plus the current position. This command is useful when trying to synchronize the position of two motors while they are moving.

Note that the controller operates as a closed-loop position controller while in the jog mode. The DMC-40x0 converts the velocity profile into a position trajectory and a new position target is generated every sample period. This method of control results in precise speed regulation with phase lock accuracy.

## Command Summary - Jogging

| COMMAND | DESCRIPTION |
|---|---|
| AC x,y,z,w | Specifies acceleration rate |
| BG XYZW | Begins motion |
| DC x,y,z,w | Specifies deceleration rate |
| IP x,y,z,w | Increments position instantly |
| IT x,y,z,w | Time constant for independent motion smoothing |
| JG +/-x,y,z,w | Specifies jog speed and direction |
| ST XYZW | Stops motion |

Parameters can be set with individual axes specifiers such as JGY=2000 (set jog speed for Y axis to 2000).

## Operand Summary - Independent Axis

| OPERAND | DESCRIPTION |
|---|---|
| _ACx | Return acceleration rate for the axis specified by 'x' |
| _DCx | Return deceleration rate for the axis specified by 'x' |
| _SPx | Returns the jog speed for the axis specified by 'x' |
| _TVx | Returns the actual velocity of the axis specified by 'x' (averaged over 0.25 sec) |

### Example - Jog in X only

Jog X motor at 50000 count/s. After X motor is at its jog speed, begin jogging Z in reverse direction at 25000 count/s.

```
#A
AC 20000,,20000        Specify X,Z acceleration of 20000 cts / sec
DC 20000,,20000        Specify X,Z deceleration of 20000 cts / sec
JG 50000,,-25000       Specify jog speed and direction for X and Z axis
BG X                   Begin X motion
AS X                   Wait until X is at speed
BG Z                   Begin Z motion
EN
```

### Example - Joystick Jogging

The jog speed can also be changed using an analog input such as a joystick. Assume that for a 10 Volt input the speed must be 50000 counts/sec.

```
#JOY                   Label
JG0                    Set in Jog Mode
BGX                    Begin motion
#B                     Label for loop
V1 =@AN[1]             Read analog input
VEL=V1*50000/10        Compute speed
JG VEL                 Change JG speed
JP #B                  Loop
```

# Position Tracking

The Galil controller may be placed in the position tracking mode to support changing the target of an absolute position move on the fly. New targets may be given in the same direction or the opposite direction of the current position target. The controller will then calculate a new trajectory based upon the new target and the acceleration, deceleration, and speed parameters that have been set. The motion profile in this mode is trapezoidal. There is not a set limit governing the rate at which the end point may be changed, however at the standard TM rate, the controller updates the position information at the rate of 1msec. The controller generates a profiled point every other sample, and linearly interpolates one sample between each profiled point. Some examples of applications that may use this mode are satellite tracking, missile tracking, random pattern polishing of mirrors or lenses, or any application that requires the ability to change the endpoint without completing the previous move.

The PA command is typically used to command an axis or multiple axes to a specific absolute position. For some applications such as tracking an object, the controller must proceed towards a target and have the ability to change the target during the move. In a tracking application, this could occur at any time during the move or at regularly scheduled intervals. For example if a robot was designed to follow a moving object at a specified distance and the path of the object wasn't known the robot would be required to constantly monitor the motion of the object that it was following. To remain within a specified distance it would also need to constantly update the position target it is moving towards. Galil motion controllers support this type of motion with the position tracking mode. This mode will allow scheduled or random updates to the current position target on the fly. Based on the new target the controller will either continue in the direction it is heading, change the direction it is moving, or decelerate to a stop.

The position tracking mode shouldn't be confused with the contour mode. The contour mode allows the user to generate custom profiles by updating the reference position at a specific time rate. In this mode, the position can be updated randomly or at a fixed time rate, but the velocity profile will always be trapezoidal with the parameters specified by AC, DC, and SP. Updating the position target at a specific rate will not allow the user to create a custom profile.

The following example will demonstrate the possible different motions that may be commanded by the controller in the position tracking mode. In this example, there is a host program that will generate the absolute position targets. The absolute target is determined based on the current information the host program has gathered on the object that it is tracking. The position tracking mode does allow for all of the axes on the controller to be in this mode, but for the sake of discussion, it is assumed that the robot is tracking only in the X dimension.

The controller must be placed in the position tracking mode to allow on the fly absolute position changes. This is performed with the PT command. To place the X axis in this mode, the host would issue PT1 to the controller if both X and Y axes were desired the command would be PT 1,1. The next step is to begin issuing PA command to the controller. The BG command isn't required in this mode, the SP, AC, and DC commands determine the shape of the trapezoidal velocity profile that the controller will use.

Example Motion 1: The host program determines that the first target for the controller to move to is located at 5000 encoder counts. The acceleration and deceleration should be set to 150,000 cts/sec2 and the velocity is set to 50,000 cts/sec. The command sequence to perform this is listed below.

| Command | Description |
|---|---|
| PT1 | Place the X axis in Position tracking mode |
| AC150000 | Set the X axis acceleration to 150000 cts/sec$^2$ |
| DC150000 | Set the X axis deceleration to 150000 cts/sec$^2$ |
| SP50000 | Set the X axis speed to 50000 cts/sec |
| PA5000 | Command the X axis to absolute position 5000 encoder counts |

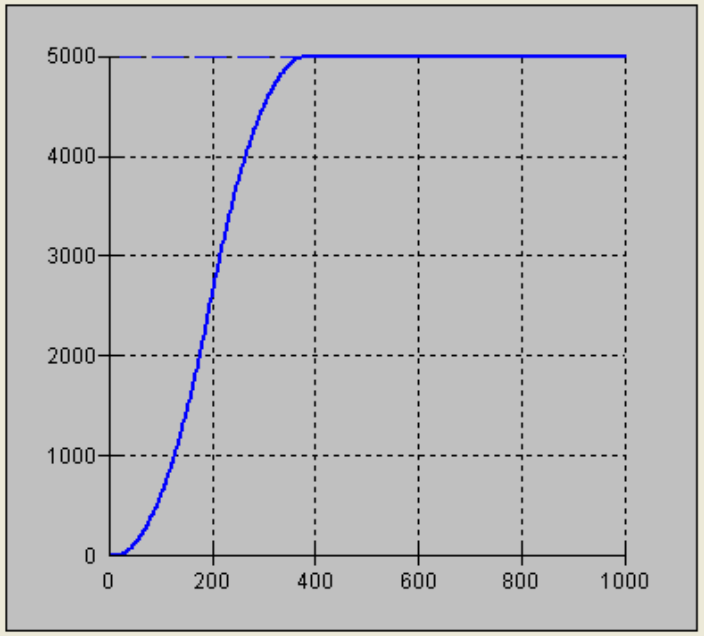



*Figure 6.1 Position vs. Time (msec) Motion 1*

## Example - Motion 2:

The previous step showed the plot if the motion continued all the way to 5000, however partway through the motion, the object that was being tracked changed direction, so the host program determined that the actual target position should be 2000 cts at that time. Figure 6.1 shows what the position profile would look like if the move was allowed to complete to 5000 cts. The position was modified when the robot was at a position of 4200 cts. Note that the robot actually travels to a distance of almost 5000 cts before it turns around. This is a function of the deceleration rate set by the DC command. When a direction change is commanded, the controller decelerates at the rate specified by the DC command. The controller then ramps the velocity in up to the value set with SP in the opposite direction traveling to the new specified absolute position. In Figure 6.3 the velocity profile is triangular because the controller doesn't have sufficient time to reach the set speed of 50000 cts/sec before it is commanded to change direction.

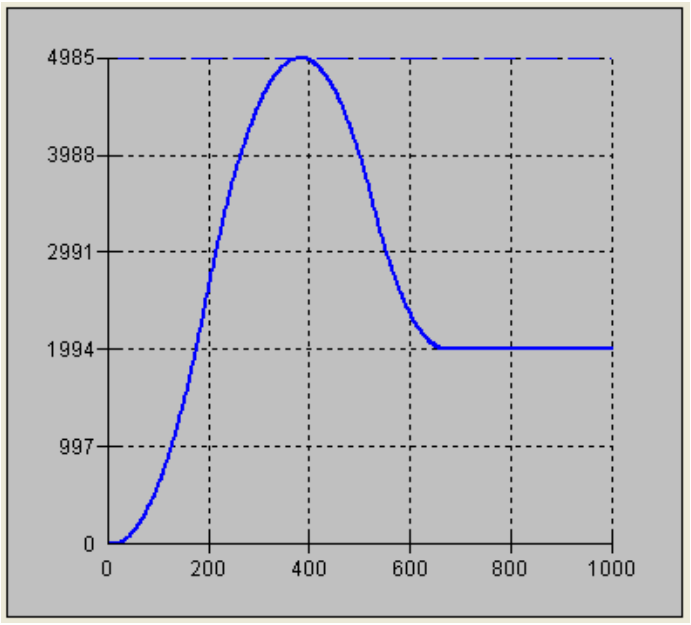



*Figure 6.2: Position vs. Time (msec) Motion 2*

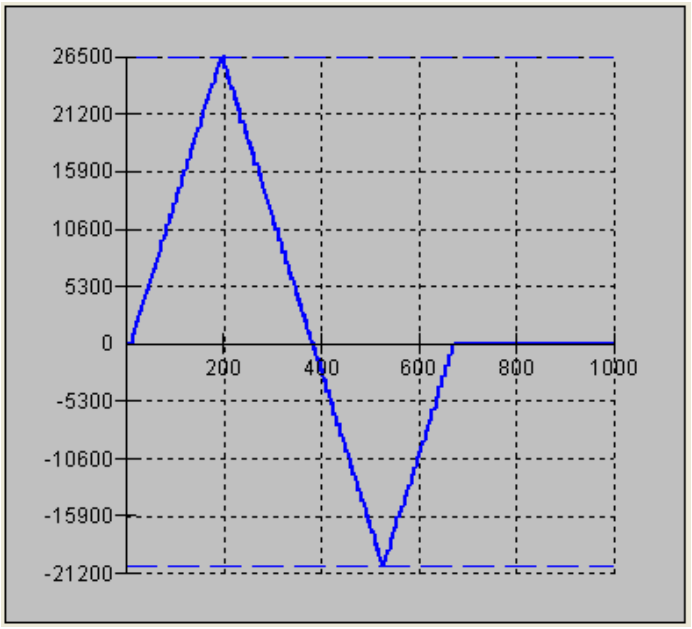



*Figure 6.3 Velocity vs. Time (msec) Motion 2*

## Example Motion 4

In this motion, the host program commands the controller to begin motion towards position 5000, changes the target to -2000, and then changes it again to 8000. Figure 6.4 shows the plot of position vs. time, Figure 6.5 plots velocity vs. time, and Figure 6.6 demonstrates the use of motion smoothing (IT) on the velocity profile in this mode. The jerk in the system is also affected by the values set for AC and DC.

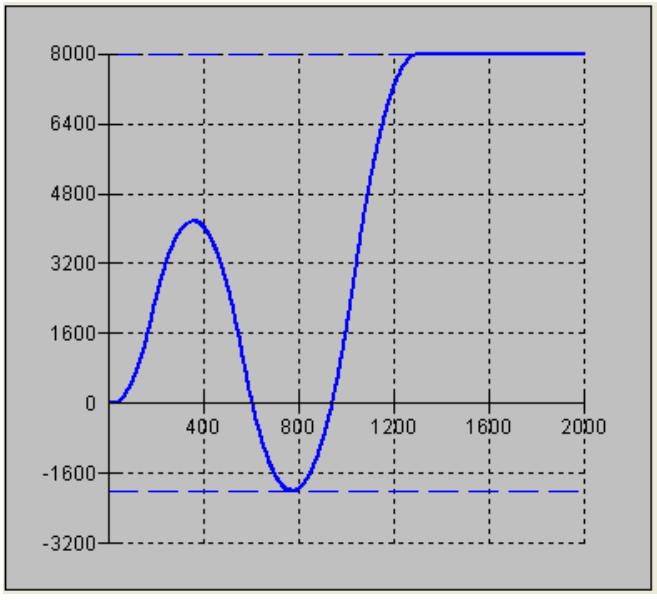



*Figure 6.4 Position vs. Time (msec) Motion 4*

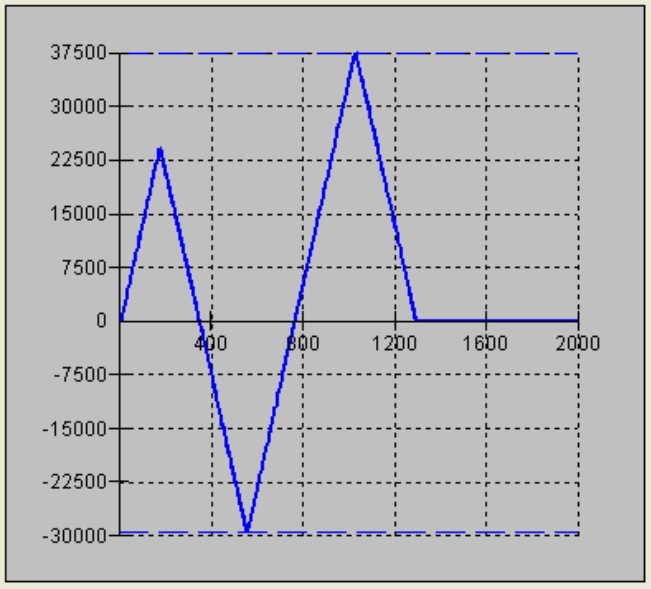



*Figure 6.5 Velocity vs. Time Motion 4*

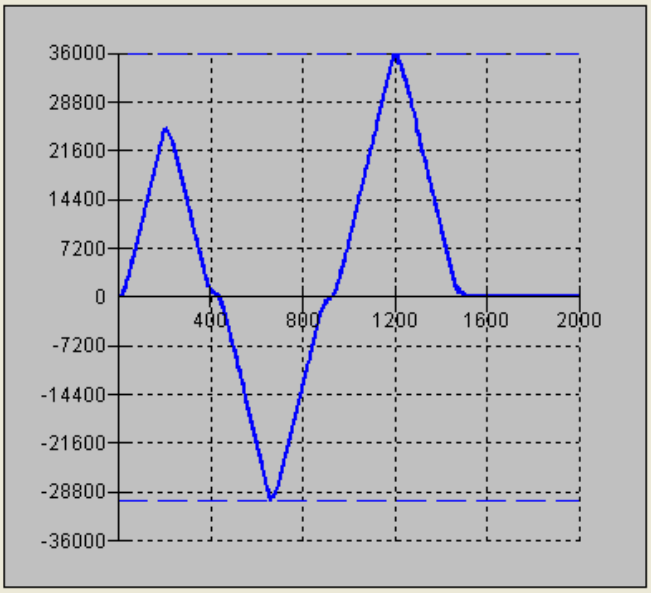



*Figure 6.6 Velocity cts/sec vs. Time (msec) with IT*

Note the controller treats the point where the velocity passes through zero as the end of one move, and the beginning of another move. IT is allowed, however it will introduce some time delay.

## Trip Points

Most trip points are valid for use while in the position tracking mode. There are a few exceptions to this; the AM and MC commands may not be used while in this mode. It is recommended that MF, MR, or AP be used, as they involve motion in a specified direction, or the passing of a specific absolute position.

## Command Summary – Position Tracking Mode

| COMMAND | DESCRIPTION |
|---|---|
| AC n,n,n,n,n,n,n,n | Acceleration settings for the specified axes |
| AP n,n,n,n,n,n,n,n | Trip point that holds up program execution until an absolute position has been reached |
| DC n,n,n,n,n,n,n,n | Deceleration settings for the specified axes |
| MF n,n,n,n,n,n,n,n | Trip point to hold up program execution until n number of counts have passed in the forward direction. Only one axis at a time may be specified. |
| MR n,n,n,n,n,n,n,n | Trip point to hold up program execution until n number of counts have passed in the reverse direction. Only one axis at a time may be specified. |
| PT n,n,n,n,n,n,n,n | Command used to enter and exit the Trajectory Modification Mode |
| PA n,n,n,n,n,n,n,n | Command Used to specify the absolute position target |
| SP n,n,n,n,n,n,n,n | Speed settings for the specified axes |

# Linear Interpolation Mode

The DMC-40x0 provides a linear interpolation mode for 2 or more axes. In linear interpolation mode, motion between the axes is coordinated to maintain the prescribed vector speed, acceleration, and deceleration along the specified path. The motion path is described in terms of incremental distances for each axis. An unlimited number of incremental segments may be given in a continuous move sequence, making the linear interpolation mode ideal for following a piece-wise linear path. There is no limit to the total move length.

The LM command selects the Linear Interpolation mode and axes for interpolation. For example, LM YZ selects only the Y and Z axes for linear interpolation.

When using the linear interpolation mode, the LM command only needs to be specified once unless the axes for linear interpolation change.

## Specifying Linear Segments

The command LI x,y,z,w or LI a,b,c,d,e,f,g,h specifies the incremental move distance for each axis. This means motion is prescribed with respect to the current axis position. Up to 511 incremental move segments may be given prior to the Begin Sequence (BGS) command. Once motion has begun, additional LI segments may be sent to the controller.

The clear sequence (CS) command can be used to remove LI segments stored in the buffer prior to the start of the motion. To stop the motion, use the instructions STS or AB. The command, ST, causes a decelerated stop. The command, AB, causes an instantaneous stop and aborts the program, and the command AB1 aborts the motion only.

The Linear End (LE) command must be used to specify the end of a linear move sequence. This command tells the controller to decelerate to a stop following the last LI command. If an LE command is not given, an Abort AB1 must be used to abort the motion sequence.

It is the responsibility of the user to keep enough LI segments in the DMC-40x0 sequence buffer to ensure continuous motion. If the controller receives no additional LI segments and no LE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for LI segments that can be sent to the buffer. 511 returned means the buffer is empty and 511 LI segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional LI segments can be sent at PC bus speeds.

The instruction _CS returns the segment counter. As the segments are processed, _CS increases, starting at zero. This function allows the host computer to determine which segment is being processed.

## Additional Commands

The commands VS n, VA n, and VD n are used to specify the vector speed, acceleration and deceleration. The DMC-40x0 computes the vector speed based on the axes specified in the LM mode. For example, LM XYZ designates linear interpolation for the X,Y and Z axes. The vector speed for this example would be computed using the equation:

VS2=XS2+YS2+ZS2, where XS, YS and ZS are the speed of the X,Y and Z axes.

The controller always uses the axis specifications from LM, not LI, to compute the speed.

IT is used to set the S-curve smoothing constant for coordinated moves. The command AV n is the 'After Vector' trippoint, which halts program execution until the vector distance of n has been reached.

## An Example of Linear Interpolation Motion:

```
#LMOVE          label
DP 0,0          Define position of X and Y axes to be 0
LMXY            Define linear mode between X and Y axes.
LI 5000,0       Specify first linear segment
LI 0,5000       Specify second linear segment
LE              End linear segments
VS 4000         Specify vector speed
BGS             Begin motion sequence
AV 4000         Set trippoint to wait until vector distance of 4000 is reached
VS 1000         Change vector speed
AV 5000         Set trippoint to wait until vector distance of 5000 is reached
VS 4000         Change vector speed
EN              Program end
```

In this example, the XY system is required to perform a 90° turn. In order to slow the speed around the corner, we use the AV 4000 trippoint, which slows the speed to 1000 count/s. Once the motors reach the corner, the speed is increased back to 4000 cts / s.

## Specifying Vector Speed for Each Segment

The instruction VS has an immediate effect and, therefore, must be given at the required time. In some applications, such as CNC, it is necessary to attach various speeds to different motion segments. This can be done by two functions: < n and > m

For example: LI x,y,z,w < n >m

The first command, < n, is equivalent to commanding VSn at the start of the given segment and will cause an acceleration toward the new commanded speeds, subjects to the other constraints.

The second function, > m, requires the vector speed to reach the value m at the end of the segment. Note that the function > m may start the deceleration within the given segment or during previous segments, as needed to meet the final speed requirement, under the given values of VA and VD.

Note, however, that the controller works with one > m command at a time. As a consequence, one function may be masked by another. For example, if the function >100000 is followed by >5000, and the distance for deceleration is not sufficient, the second condition will not be met. The controller will attempt to lower the speed to 5000, but will reach that at a different point.

As an example, consider the following program.

```
#ALT                    Label for alternative program
DP 0,0                  Define Position of X and Y axis to be 0
LMXY                    Define linear mode between X and Y axes.
LI 4000,0 <4000 >1000   Specify first linear segment with a vector speed of 4000 and end speed 1000
```

```
LI 1000,1000 < 4000 >1000      Specify second linear segment with a vector speed of 4000 and end speed 1000
LI 0,5000 < 4000 >1000         Specify third linear segment with a vector speed of 4000 and end speed 1000
LE                             End linear segments
BGS                            Begin motion sequence
EN                             Program end
```

### Changing Feed Rate:

The command VR n allows the feed rate, VS, to be scaled between 0 and 10 with a resolution of .0001. This command takes effect immediately and causes VS to be scaled. VR also applies when the vector speed is specified with the '<' operator. This is a useful feature for feed rate override. VR does not ratio the accelerations. For example, VR .5 results in the specification VS 2000 to be divided in half.

## Command Summary - Linear Interpolation

| COMMAND | DESCRIPTION |
|---|---|
| LM xyzw<br>LM abcdefgh | Specify axes for linear interpolation<br>(same) controllers with 5 or more axes |
| LM? | Returns number of available spaces for linear segments in DMC-40x0 sequence buffer. Zero means buffer full. 511 means buffer empty. |
| LI x,y,z,w < n<br>LI a,b,c,d,e,f,g,h < n | Specify incremental distances relative to current position, and assign vector speed n. |
| VS n | Specify vector speed |
| VA n | Specify vector acceleration |
| VD n | Specify vector deceleration |
| VR n | Specify the vector speed ratio |
| BGS | Begin Linear Sequence |
| CS | Clear sequence |
| LE | Linear End- Required at end of LI command sequence |
| LE? | Returns the length of the vector (resets after 2147483647) |
| AMS | Trippoint for After Sequence complete |
| AV n | Trippoint for After Relative Vector distance, n |
| IT | S curve smoothing constant for vector moves |

## Operand Summary - Linear Interpolation

| OPERAND | DESCRIPTION |
|---|---|
| _AV | Return distance traveled |
| _CS | Segment counter - returns number of the segment in the sequence, starting at zero. |
| _LE | Returns length of vector (resets after 2147483647) |
| _LM | Returns number of available spaces for linear segments in DMC-40x0 sequence buffer. Zero means buffer full. 511 means buffer empty. |

| | |
|---|---|
| _VPm | Return the absolute coordinate of the last data point along the trajectory. (m=X,Y,Z or W or A,B,C,D,E,F,G or H) |

To illustrate the ability to interrogate the motion status, consider the first motion segment of our example, #LMOVE, where the X axis moves toward the point X=5000. Suppose that when X=3000, the controller is interrogated using the command 'MG _AV'. The returned value will be 3000. The value of _CS, _VPX and _VPY will be zero.

Now suppose that the interrogation is repeated at the second segment when Y=2000. The value of _AV at this point is 7000, _CS equals 1, _VPX=5000 and _VPY=0.

## Example - Linear Move

Make a coordinated linear move in the ZW plane. Move to coordinates 40000,30000 counts at a vector speed of 100000 counts/sec and vector acceleration of 1000000 counts/sec2.

```
LM ZW                  Specify axes for linear interpolation
LI,,40000,30000        Specify ZW distances
LE                     Specify end move
VS 100000              Specify vector speed
VA 1000000             Specify vector acceleration
VD 1000000             Specify vector deceleration
BGS                    Begin sequence
```

Note that the above program specifies the vector speed, VS, and not the actual axis speeds VZ and VW. The axis speeds are determined by the controller from:

$$VS = \sqrt{VZ^2 + VW^2}$$

The result is shown in Figure 6.7

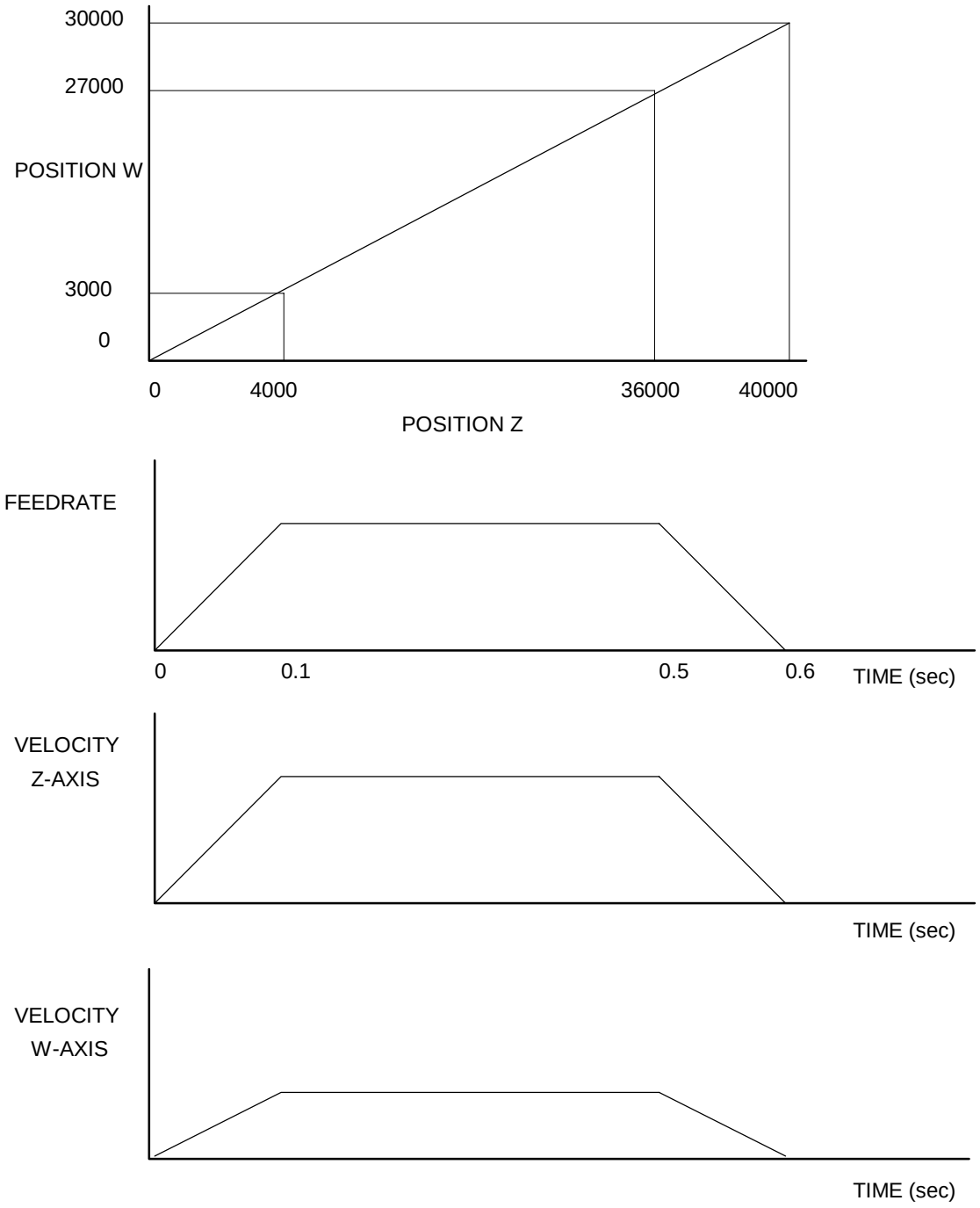



*Figure 6.7 - Linear Interpolation*

## Example - Multiple Moves

This example makes a coordinated linear move in the XY plane. The Arrays VX and VY are used to store 750 incremental distances which are filled by the program #LOAD.

```
#LOAD                          Load Program
DM VX [750],VY [750]           Define Array
COUNT=0                        Initialize Counter
```

```
N=0                              Initialize position increment
#LOOP                            LOOP
VX [COUNT]=N                     Fill Array VX
VY [COUNT]=N                     Fill Array VY
N=N+10                           Increment position
COUNT=COUNT+1                    Increment counter
JP #LOOP,COUNT<750               Loop if array not full
#A                               Label
LM XY                            Specify linear mode for XY
COUNT=0                          Initialize array counter
#LOOP2;JP#LOOP2,_LM=0            If sequence buffer full, wait
JS#C,COUNT=500                   Begin motion on 500th segment
LI VX[COUNT],VY[COUNT]           Specify linear segment
COUNT=COUNT+1                    Increment array counter
JP #LOOP2,COUNT<750              Repeat until array done
LE                               End Linear Move
AMS                              After Move sequence done
MG "DONE"                        Send Message
EN                               End program
#C;BGS;EN                        Begin Motion Subroutine
```

# Vector Mode: Linear and Circular Interpolation Motion

The DMC-40x0 allows a long 2-D path consisting of linear and arc segments to be prescribed. Motion along the path is continuous at the prescribed vector speed even at transitions between linear and circular segments. The DMC-40x0 performs all the complex computations of linear and circular interpolation, freeing the host PC from this time intensive task.

The coordinated motion mode is similar to the linear interpolation mode. Any pair of two axes may be selected for coordinated motion consisting of linear and circular segments. In addition, a third axis can be controlled such that it remains tangent to the motion of the selected pair of axes. Note that only one pair of axes can be specified for coordinated motion at any given time.

The command VM m,n,p where 'm' and 'n' are the coordinated pair and p is the tangent axis (Note: the commas which separate m,n and p are not necessary). For example, VM XWZ selects the XW axes for coordinated motion and the Z-axis as the tangent.

## Specifying the Coordinate Plane

The DMC-40x0 allows for 2 separate sets of coordinate axes for linear interpolation mode or vector mode. These two sets are identified by the letters S and T.

To specify vector commands the coordinate plane must first be identified. This is done by issuing the command CAS to identify the S plane or CAT to identify the T plane. All vector commands will be applied to the active coordinate system until changed with the CA command.

## Specifying Vector Segments

The motion segments are described by two commands; VP for linear segments and CR for circular segments. Once a set of linear segments and/or circular segments have been specified, the sequence is ended with the command VE. This defines a sequence of commands for coordinated motion. Immediately prior to the execution of the first coordinated movement, the controller defines the current position to be zero for all movements in a sequence. Note:

This 'local' definition of zero does not affect the absolute coordinate system or subsequent coordinated motion sequences.

The command, VP x,y specifies the coordinates of the end points of the vector movement with respect to the starting point. Non-sequential axis do not require comma delimitation. The command, CR r,q,d define a circular arc with a radius r, starting angle of q, and a traversed angle d. The notation for q is that zero corresponds to the positive horizontal direction, and for both q and d, the counter-clockwise (CCW) rotation is positive.

Up to 511 segments of CR or VP may be specified in a single sequence and must be ended with the command VE. The motion can be initiated with a Begin Sequence (BGS) command. Once motion starts, additional segments may be added.

The Clear Sequence (CS) command can be used to remove previous VP and CR commands which were stored in the buffer prior to the start of the motion. To stop the motion, use the instructions STS or AB1. ST stops motion at the specified deceleration. AB1 aborts the motion instantaneously.

The Vector End (VE) command must be used to specify the end of the coordinated motion. This command requires the controller to decelerate to a stop following the last motion requirement. If a VE command is not given, an Abort (AB1) must be used to abort the coordinated motion sequence.

It is the responsibility of the user to keep enough motion segments in the DMC-40x0 sequence buffer to ensure continuous motion. If the controller receives no additional motion segments and no VE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for motion segments that can be sent to the buffer. 511 returned means the buffer is empty and 511 segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional segments can be sent at PC bus speeds.

The operand _CS can be used to determine the value of the segment counter.

## Additional commands

The commands VS n, VA n and VD n are used for specifying the vector speed, acceleration, and deceleration.

IT is the s curve smoothing constant used with coordinated motion.

### Specifying Vector Speed for Each Segment:

The vector speed may be specified by the immediate command VS. It can also be attached to a motion segment with the instructions

VP x,y < n >m

CR r,θ,δ < n >m

The first command, <n, is equivalent to commanding VSn at the start of the given segment and will cause an acceleration toward the new commanded speeds, subjects to the other constraints.

The second function, > m, requires the vector speed to reach the value m at the end of the segment. Note that the function > m may start the deceleration within the given segment or during previous segments, as needed to meet the final speed requirement, under the given values of VA and VD.

Note, however, that the controller works with one > m command at a time. As a consequence, one function may be masked by another. For example, if the function >100000 is followed by >5000, and the distance for deceleration is not sufficient, the second condition will not be met. The controller will attempt to lower the speed to 5000, but will reach that at a different point.

### Changing Feed Rate:

The command VR n allows the feed rate, VS, to be scaled between 0 and 10 with a resolution of .0001. This command takes effect immediately and causes VS scaled. VR also applies when the vector speed is specified with the '<' operator. This is a useful feature for feed rate override. VR does not ratio the accelerations. For example, VR 0.5 results in the specification VS 2000 to be divided by two.

### Compensating for Differences in Encoder Resolution:

By default, the DMC-40x0 uses a scale factor of 1:1 for the encoder resolution when used in vector mode. If this is not the case, the command, ES can be used to scale the encoder counts. The ES command accepts two arguments which represent the number of counts for the two encoders used for vector motion. The smaller ratio of the two numbers will be multiplied by the higher resolution encoder. For more information, see ES command in the Command Reference.

### Trippoints:

The AV n command is the After Vector trippoint, which waits for the vector relative distance of n to occur before executing the next command in a program.

### Tangent Motion:

Several applications, such as cutting, require a third axis (i.e. a knife blade), to remain tangent to the coordinated motion path. To handle these applications, the DMC-40x0 allows one axis to be specified as the tangent axis. The VM command provides parameter specifications for describing the coordinated axes and the tangent axis.

| | |
|---|---|
| VM m,n,p | m,n specifies coordinated axes p specifies tangent axis such as X,Y,Z,W p=N turns off tangent axis |

Before the tangent mode can operate, it is necessary to assign an axis via the VM command and define its offset and scale factor via the TN m,n command. m defines the scale factor in counts/degree and n defines the tangent position that equals zero degrees in the coordinated motion plane. The operand _TN can be used to return the initial position of the tangent axis.

### Example:

Assume an XY table with the Z-axis controlling a knife. The Z-axis has a 2000 quad counts/rev encoder and has been initialized after power-up to point the knife in the +Y direction. A 180° circular cut is desired, with a radius of 3000, center at the origin and a starting point at (3000,0). The motion is CCW, ending at (-3000,0). Note that the 0° position in the XY plane is in the +X direction. This corresponds to the position -500 in the Z-axis, and defines the offset. The motion has two parts. First, X,Y and Z are driven to the starting point, and later, the cut is performed. Assume that the knife is engaged with output bit 0.

```
#EXAMPLE           Example program
VM XYZ             XY coordinate with Z as tangent
TN 2000/360,-500   2000/360 counts/degree, position -500 is 0 degrees in XY plane
CR 3000,0,180      3000 count radius, start at 0 and go to 180 CCW
VE                 End vector
CB0                Disengage knife
PA 3000,0,_TN      Move X and Y to starting position, move Z to initial tangent position
BG XYZ             Start the move to get into position
AM XYZ             When the move is complete
SB0                Engage knife
WT50               Wait 50 msec for the knife to engage
BGS                Do the circular cut
AMS                After the coordinated move is complete
CB0                Disengage knife
MG "ALL DONE"
EN                 End program
```

## Command Summary - Coordinated Motion Sequence

| COMMAND | DESCRIPTION. |
|---|---|
| VM m,n | Specifies the axes for the planar motion where m and n represent the planar axes and p is the tangent axis. |
| VP m,n | Return coordinate of last point, where m=X,Y,Z or W. |
| CR r,Θ, ±ΔΘ | Specifies arc segment where r is the radius, Θ is the starting angle and ΔΘ is the travel angle. Positive direction is CCW. |
| VS s,t | Specify vector speed or feed rate of sequence. |
| VA s,t | Specify vector acceleration along the sequence. |
| VD s,t | Specify vector deceleration along the sequence. |
| VR s,t | Specify vector speed ratio |
| BGST | Begin motion sequence, S or T |
| CSST | Clear sequence, S or T |
| AV s,t | Trippoint for After Relative Vector distance. |
| AMST | Holds execution of next command until Motion Sequence is complete. |
| TN m,n | Tangent scale and offset. |
| ES m,n | Ellipse scale factor. |
| IT s,t | S curve smoothing constant for coordinated moves |
| LM? | Return number of available spaces for linear and circular segments in DMC-40x0 sequence buffer. Zero means buffer is full. 511 means buffer is empty. |
| CAS or CAT | Specifies which coordinate system is to be active (S or T) |

## Operand Summary - Coordinated Motion Sequence

| OPERAND | DESCRIPTION |
|---|---|
| _VPM | The absolute coordinate of the axes at the last intersection along the sequence. |
| _AV | Distance traveled. |
| _LM | Number of available spaces for linear and circular segments in DMC-40x0 sequence buffer. Zero means buffer is full. 511 means buffer is empty. |
| _CS | Segment counter - Number of the segment in the sequence, starting at zero. |
| _VE | Vector length of coordinated move sequence. |

When AV is used as an operand, _AV returns the distance traveled along the sequence.

The operands _VPX and _VPY can be used to return the coordinates of the last point specified along the path.

### Example:

Traverse the path shown in Fig. 6.8. Feed rate is 20000 counts/sec. Plane of motion is XY

```
VM XY                Specify motion plane
VS 20000             Specify vector speed
VA 1000000           Specify vector acceleration
VD 1000000           Specify vector deceleration
VP -4000,0           Segment AB
```

```
CR 1500,270,-180        Segment BC
VP 0,3000               Segment CD
CR 1500,90,-180         Segment DA
VE                      End of sequence
BGS                     Begin Sequence
```

The resulting motion starts at the point A and moves toward points B, C, D, A. Suppose that we interrogate the controller when the motion is halfway between the points A and B.

The value of _AV is 2000

The value of _CS is 0

_VPX and _VPY contain the absolute coordinate of the point A

Suppose that the interrogation is repeated at a point, halfway between the points C and D.

The value of _AV is 4000+1500π+2000=10,712

The value of _CS is 2

_VPX,_VPY contain the coordinates of the point C

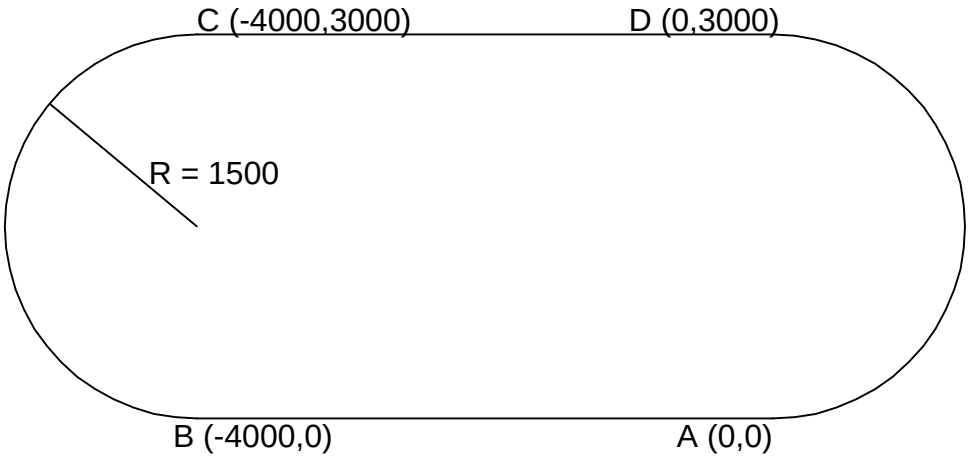



*Figure 6.8 - The Required Path*

# Electronic Gearing

This mode allows up to 8 axes to be electronically geared to some master axes. The masters may rotate in both directions and the geared axes will follow at the specified gear ratio. The gear ratio may be different for each axis and changed during motion.

The command GAX yzw or GA ABCDEFGH specifies the master axes. GR x,y,z,w specifies the gear ratios for the slaves where the ratio may be a number between +/-127.9999 with a fractional resolution of .0001. There are two modes: standard gearing and gantry mode. The gantry mode (enabled with the command GM) allows the gearing to stay enabled even if a limit is hit or an ST command is issued. GR 0,0,0,0 turns off gearing in both modes.

The command GM x,y,z,w select the axes to be controlled under the gantry mode. The parameter 1 enables gantry mode, and 0 disables it.

GR causes the specified axes to be geared to the actual position of the master. The master axis is commanded with motion commands such as PR, PA or JG.

When the master axis is driven by the controller in the jog mode or an independent motion mode, it is possible to define the master as the command position of that axis, rather than the actual position. The designation of the

commanded position master is by the letter, C. For example, GACX indicates that the gearing is the commanded position of X.

An alternative gearing method is to synchronize the slave motor to the commanded vector motion of several axes performed by GAS. For example, if the X and Y motor form a circular motion, the Z axis may move in proportion to the vector move. Similarly, if X,Y and Z perform a linear interpolation move, W can be geared to the vector move.

Electronic gearing allows the geared motor to perform a second independent or coordinated move in addition to the gearing. For example, when a geared motor follows a master at a ratio of 1:1, it may be advanced an additional distance with PR, or JG, commands, or VP, or LI.

# Ramped Gearing

In some applications, especially when the master is traveling at high speeds, it is desirable to have the gear ratio ramp gradually to minimize large changes in velocity on the slave axis when the gearing is engaged. For example if the master axis is already traveling at 1,000,000 cts/sec and the slave will be geared at a ratio of 1:1 when the gearing is engaged, the slave will instantly develop following error, and command maximum current to the motor. This can be a large shock to the system. For many applications it is acceptable to slowly ramp the engagement of gearing over a greater time frame. Galil allows the user to specify an interval of the master axis over which the gearing will be engaged. For example, the same master X axis in this case travels at 1,000,000 counts/sec, and the gear ratio is 1:1, but the gearing is slowly engaged over 30,000 cts of the master axis, greatly diminishing the initial shock to the slave axis. Figure 1 below shows the velocity vs. time profile for instantaneous gearing. Figure 6.9 shows the velocity vs. time profile for the gradual gearing engagement.

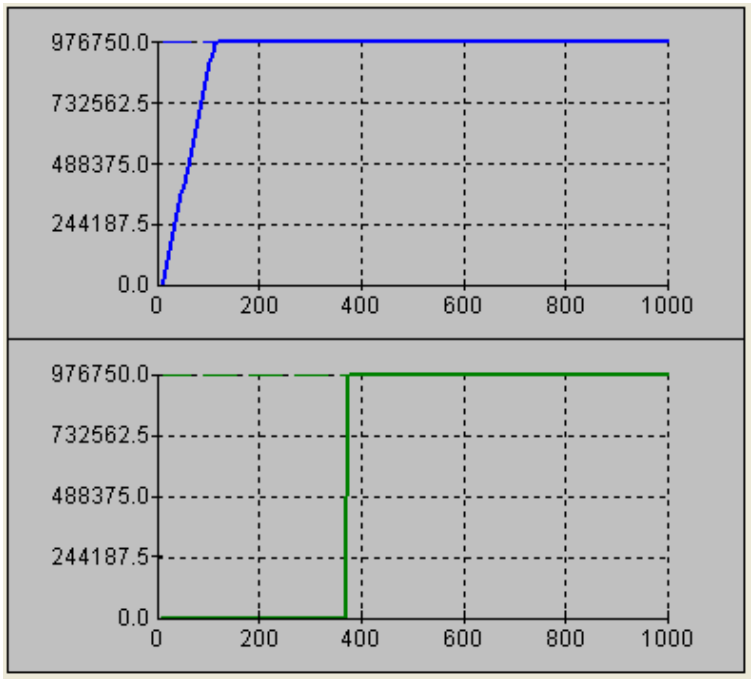



*Figure 6.9 Velocity cts/sec vs. Time (msec) Instantaneous Gearing Engagement*

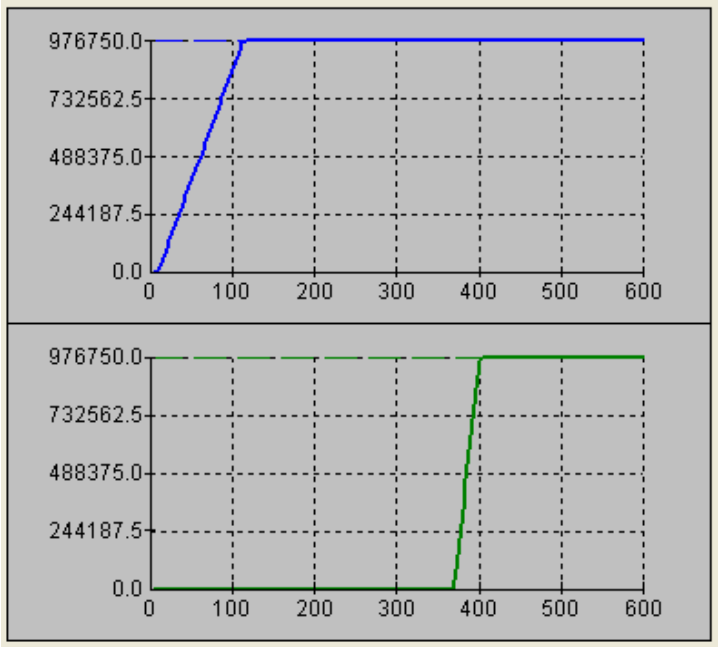



*Figure 6.10 Velocity (cts/sec) vs. Time (msec) Ramped Gearing*

The slave axis for each figure is shown on the bottom portion of the figure; the master axis is shown on the top portion. The shock to the slave axis will be significantly less in figure 6.10 than in figure 6.9. The ramped gearing does have one consequence. There isn't a true synchronization of the two axes, until the gearing ramp is complete. The slave will lag behind the true ratio during the ramp period. If exact position synchronization is required from the point gearing is initiated, then the position must be commanded in addition to the gearing. The controller keeps track of this position phase lag with the _GP operand. The following example will demonstrate how the command is used.

## Example – Electronic Gearing Over a Specified Interval

Objective Run two geared motors at speeds of 1.132 and -.045 times the speed of an external master. Because the master is traveling at high speeds, it is desirable for the speeds to change slowly.

Solution: Use a DMC-4030 controller where the Z-axis is the master and X and Y are the geared axes. We will implement the gearing change over 6000 counts (3 revolutions) of the master axis.

```
MO Z                    Turn Z off, for external master
GA Z, Z                 Specify Z as the master axis for both X and Y.
GD6000,6000            Specify ramped gearing over 6000 counts of the master axis.
GR 1.132,-.045          Specify gear ratios
```

Question: What is the effect of the ramped gearing?

Answer: Below, in the example titled Electronic Gearing, gearing would take effect immediately. From the start of gearing if the master traveled 6000 counts, the slaves would travel 6792 counts and 270 counts.

Using the ramped gearing, the slave will engage gearing gradually. Since the gearing is engaged over the interval of 6000 counts of the master, the slave will only travel ~3396 counts and ~135 counts respectively. The difference between these two values is stored in the _GPn operand. If exact position synchronization is required, the IP command is used to adjust for the difference.

## Command Summary - Electronic Gearing

| COMMAND | DESCRIPTION |
|---|---|
| GA n | Specifies master axes for gearing where:<br>n = X,Y,Z or W or A,B,C,D,E,F,G,H for main encoder as master |
| | n = CX,CY,CZ, CW **or** CA, CB,CC,CD,CE,CF,CG,CH for commanded position. |
| | n = DX,DY,DZ or DW or DA, DB, DC, DD, DE, DF,DG,DH for auxiliary encoders |
| | n = S or T for gearing to coordinated motion. |
| GD a,b,c,d,e,f,g,h | Sets the distance the master will travel for the gearing change to take full effect. |
| _GPn | This operand keeps track of the difference between the theoretical distance traveled if gearing changes took effect immediately, and the distance traveled since gearing changes take effect over a specified interval. |
| GR a,b,c,d,e,f,g,h | Sets gear ratio for slave axes. 0 disables electronic gearing for specified axis. |
| GM a,b,c,d,e,f,g,h | X = 1 sets gantry mode, 0 disables gantry mode |
| MR x,y,z,w | Trippoint for reverse motion past specified value. Only one field may be used. |
| MF x,y,z,w | Trippoint for forward motion past specified value. Only one field may be used. |

## Example - Simple Master Slave

Master axis moves 10000 counts at slew speed of 100000 counts/sec. Y is defined as the master. X,Z,W are geared to master at ratios of 5,-.5 and 10 respectively.

```
GA Y,,Y,Y            Specify master axes as Y
GR 5,,-.5,10         Set gear ratios
PR ,10000            Specify Y position
SP ,100000           Specify Y speed
BGY                  Begin motion
```

## Example - Electronic Gearing

Objective: Run two geared motors at speeds of 1.132 and -0.045 times the speed of an external master. The master is driven at speeds between 0 and 1800 RPM (2000 counts/rev encoder).

Solution: Use a DMC-4030 controller, where the Z-axis is the master and X and Y are the geared axes.

```
MO Z                 Turn Z off, for external master
GA Z, Z              Specify Z as the master axis for both X and Y.
GR 1.132,-.045       Specify gear ratios
```

Now suppose the gear ratio of the X-axis is to change on-the-fly to 2. This can be achieved by commanding:

```
GR 2                 Specify gear ratio for X axis to be 2
```

## Example - Gantry Mode

In applications where both the master and the follower are controlled by the DMC-40x0 controller, it may be desired to synchronize the follower with the commanded position of the master, rather than the actual position. This eliminates the coupling between the axes which may lead to oscillations.

For example, assume that a gantry is driven by two axes, X,Y, on both sides. This requires the gantry mode for strong coupling between the motors. The X-axis is the master and the Y-axis is the follower. To synchronize Y with the commanded position of X, use the instructions:

```
GA, CX              Specify the commanded position of X as master for Y.
GR,1                Set gear ratio for Y as 1:1
GM,1                Set gantry mode
PR 3000             Command X motion
BG X                Start motion on X axis
```

You may also perform profiled position corrections in the electronic gearing mode. Suppose, for example, that you need to advance the slave 10 counts. Simply command

```
IP ,10              Specify an incremental position movement of 10 on Y axis.
```

Under these conditions, this IP command is equivalent to:

```
PR,10               Specify position relative movement of 10 on Y axis
BGY                 Begin motion on Y axis
```

Often the correction is quite large. Such requirements are common when synchronizing cutting knives or conveyor belts.

### Example - Synchronize two conveyor belts with trapezoidal velocity correction

```
GA,X                Define X as the master axis for Y.
GR,2                Set gear ratio 2:1 for Y
PR,300              Specify correction distance
SP,5000             Specify correction speed
AC,100000           Specify correction acceleration
DC,100000           Specify correction deceleration
BGY                 Start correction
```

# Electronic Cam

The electronic cam is a motion control mode which enables the periodic synchronization of several axes of motion. Up to 7 axes can be slaved to one master axis. The master axis encoder must be input through a main encoder port.

The electronic cam is a more general type of electronic gearing which allows a table-based relationship between the axes. It allows synchronizing all the controller axes. For example, the DMC-4080 controllers may have one master and up to seven slaves.

To illustrate the procedure of setting the cam mode, consider the cam relationship for the slave axis Y, when the master is X. Such a graphic relationship is shown in Figure 6.11.

### Step 1. Selecting the master axis

The first step in the electronic cam mode is to select the master axis. This is done with the instruction

EAp where p = X,Y,Z,W,E,F,G,H

p is the selected master axis

For the given example, since the master is x, we specify EAX

### Step 2. Specify the master cycle and the change in the slave axis (or axes).

In the electronic cam mode, the position of the master is always expressed modulo one cycle. In this example, the position of x is always expressed in the range between 0 and 6000. Similarly, the slave position is also redefined such that it starts at zero and ends at 1500. At the end of a cycle when the master is 6000 and the slave is 1500, the positions of both x and y are redefined as zero. To specify the master cycle and the slave cycle change, we use the instruction EM.

```
EM x,y,z,w
```

where x,y,z,w specify the cycle of the master and the total change of the slaves over one cycle.

The cycle of the master is limited to 8,388,607 whereas the slave change per cycle is limited to 2,147,483,647. If the change is a negative number, the absolute value is specified. For the given example, the cycle of the master is 6000 counts and the change in the slave is 1500. Therefore, we use the instruction:

```
EM 6000,1500
```

### Step 3. Specify the master interval and starting point.

Next we need to construct the ECAM table. The table is specified at uniform intervals of master positions. Up to 256 intervals are allowed. The size of the master interval and the starting point are specified by the instruction:

```
EP m,n
```

where m is the interval width in counts, and n is the starting point.

For the given example, we can specify the table by specifying the position at the master points of 0, 2000, 4000 and 6000. We can specify that by

```
EP 2000,0
```

### Step 4. Specify the slave positions.

Next, we specify the slave positions with the instruction

```
ET[n]=x,y,z,w
```

where n indicates the order of the point.

The value, n, starts at zero and may go up to 256. The parameters x,y,z,w indicate the corresponding slave position. For this example, the table may be specified by

```
ET[0]=,0
ET[1]=,3000
ET[2]=,2250
ET[3]=,1500
```

This specifies the ECAM table.

### Step 5. Enable the ECAM

To enable the ECAM mode, use the command

```
EB n
```

where n=1 enables ECAM mode and n=0 disables ECAM mode.

### Step 6. Engage the slave motion

To engage the slave motion, use the instruction

```
EG x,y,z,w
```

where x,y,z,w are the master positions at which the corresponding slaves must be engaged.

If the value of any parameter is outside the range of one cycle, the cam engages immediately. When the cam is engaged, the slave position is redefined, modulo one cycle.

### Step 7. Disengage the slave motion

To disengage the cam, use the command

```
EQ x,y,z,w
```

where x,y,z,w are the master positions at which the corresponding slave axes are disengaged.

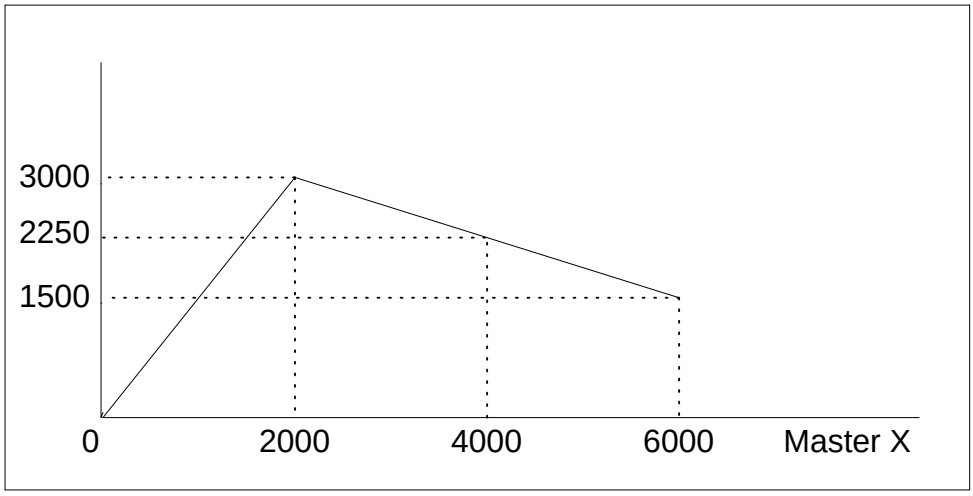



*Figure 6.11: Electronic Cam Example*

This disengages the slave axis at a specified master position. If the parameter is outside the master cycle, the stopping is instantaneous.

To illustrate the complete process, consider the cam relationship described by

the equation:

$$Y = 0.5 * X + 100 \sin (0.18*X)$$

where X is the master, with a cycle of 2000 counts.

The cam table can be constructed manually, point by point, or automatically by a program. The following program includes the set-up.

The instruction EAX defines X as the master axis. The cycle of the master is

2000. Over that cycle, Y varies by 1000. This leads to the instruction EM 2000,1000.

Suppose we want to define a table with 100 segments. This implies increments of 20 counts each. If the master points are to start at zero, the required instruction is EP 20,0.

The following routine computes the table points. As the phase equals 0.18X and X varies in increments of 20, the phase varies by increments of 3.6□. The program then computes the values of Y according to the equation and assigns the values to the table with the instruction ET[N] = ,Y.

| INSTRUCTION | INTERPRETATION |
|---|---|
| #SETUP | Label |
| EAX | Select X as master |
| EM 2000,1000 | Cam cycles |
| EP 20,0 | Master position increments |
| N = 0 | Index |
| #LOOP | Loop to construct table from equation |
| P = N*3.6 | Note 3.6 = 0.18 * 20 |
| S = @SIN [P]*100 | Define sine position |
| Y = N*10+S | Define slave position |
| ET [N] =, Y | Define table |
| N = N+1 | |

```
JP #LOOP, N<=100      Repeat the process
EN
```

Now suppose that the slave axis is engaged with a start signal, input 1, but that both the engagement and disengagement points must be done at the center of the cycle: X = 1000 and Y = 500. This implies that Y must be driven to that point to avoid a jump.

This is done with the program:

| INSTRUCTION | INTERPRETATION |
|---|---|
| #RUN | Label |
| EB1 | Enable cam |
| PA,500 | starting position |
| SP,5000 | Y speed |
| BGY | Move Y motor |
| AM | After Y moved |
| AI1 | Wait for start signal |
| EG,1000 | Engage slave |
| AI - 1 | Wait for stop signal |
| EQ,1000 | Disengage slave |
| EN | End |

## Command Summary - Electronic CAM

| Command | Description |
|---|---|
| EA p | Specifies master axes for electronic cam where:<br>p = X,Y,Z or W or A,B,C,D,E,F,G,H for main encoder as master or M or N a for virtual axis master |
| EB n | Enables the ECAM |
| EC n | ECAM counter - sets the index into the ECAM table |
| EG x,y,z,w | Engages ECAM |
| EM x,y,z,w | Specifies the change in position for each axis of the CAM cycle |
| EP m,n | Defines CAM table entry size and offset |
| EQ m,n | Disengages ECAM at specified position |
| ET[n] | Defines the ECAM table entries |
| EW | Widen Segment (see Application Note #2444) |
| EY | Set ECAM cycle count |

## Operand Summary - Electronic CAM

| Command | Description |
|---|---|
| _EB | Contains State of ECAM |
| _EC | Contains current ECAM index |
| _EGx | Contains ECAM status for each axis |
| _EM | Contains size of cycle for each axis |

| | |
|---|---|
| _EP | Contains value of the ECAM table interval |
| _EQx | Contains ECAM status for each axis |
| _EY | Set ECAM cycle count |

## Example - Electronic CAM

The following example illustrates a cam program with a master axis, Z, and two slaves, X and Y.

| INSTRUCTION | INTERPRETATION |
|---|---|
| #A;V1=0 | Label;  Initialize variable |
| PA 0,0;BGXY;AMXY | Go to position 0,0 on X and Y axes |
| EA Z | Z axis as the Master for ECAM |
| EM 0,0,4000 | Change for Z is 4000, zero for X, Y |
| EP400,0 | ECAM interval is 400 counts with zero start |
| ET[0]=0,0 | When master is at 0 position; 1st point. |
| ET[1]=40,20 | 2nd point in the ECAM table |
| ET[2]=120,60 | 3rd point in the ECAM table |
| ET[3]=240,120 | 4th point in the ECAM table |
| ET[4]=280,140 | 5th point in the ECAM table |
| ET[5]=280,140 | 6th point in the ECAM table |
| ET[6]=280,140 | 7th point in the ECAM table |
| ET[7]=240,120 | 8th point in the ECAM table |
| ET[8]=120,60 | 9th point in the ECAM table |
| ET[9]=40,20 | 10th point in the ECAM table |
| ET[10]=0,0 | Starting point for next cycle |
| EB 1 | Enable ECAM mode |
| JGZ=4000 | Set Z to jog at 4000 |
| EG 0,0 | Engage both X and Y when Master = 0 |
| BGZ | Begin jog on Z axis |
| #LOOP;JP#LOOP,V1=0 | Loop until the variable is set |
| EQ2000,2000 | Disengage X and Y when Master = 2000 |
| MF,, 2000 | Wait until the Master goes to 2000 |
| ST Z | Stop the Z axis motion |
| EB 0 | Exit the ECAM mode |
| EN | End of the program |

The above example shows how the ECAM program is structured and how the commands can be given to the controller. The next page provides the results captured by the WSDK program. This shows how the motion will be seen during the ECAM cycles. The first graph is for the X axis, the second graph shows the cycle on the Y axis and the third graph shows the cycle of the Z axis.

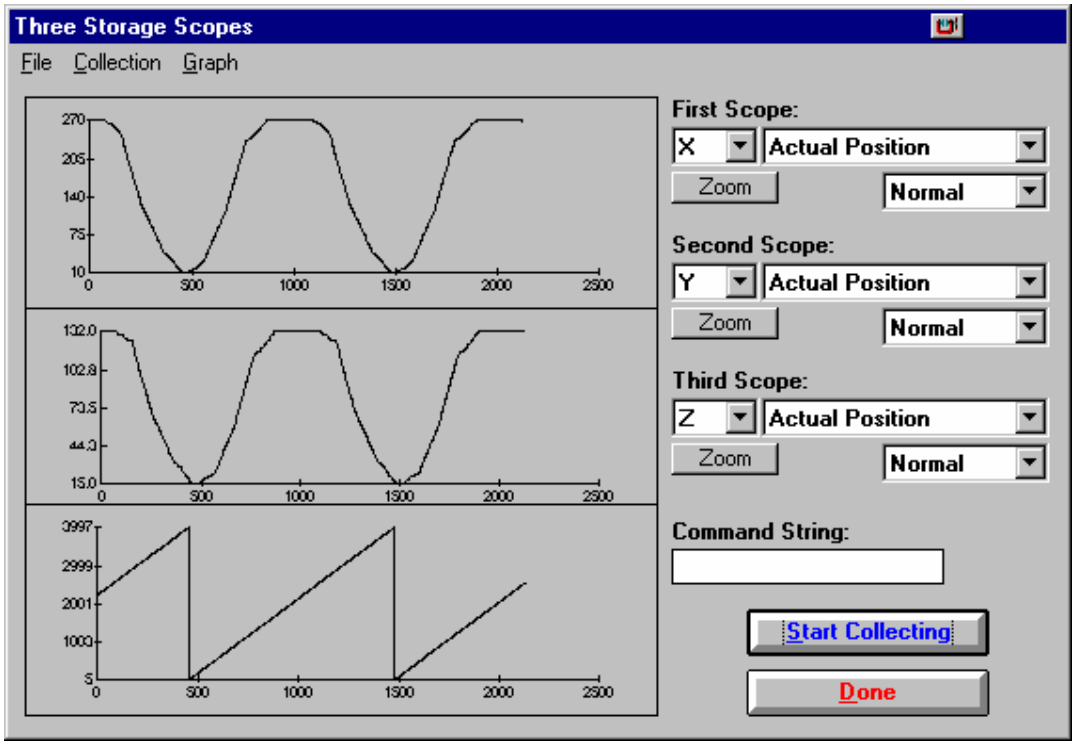



*Figure 6.12 – Three Storage Scopes*

# Contour Mode

The DMC-40x0 also provides a contouring mode. This mode allows any arbitrary position curve to be prescribed for 1 to 8 axes. This is ideal for following computer generated paths such as parabolic, spherical or user-defined profiles. The path is not limited to straight line and arc segments and the path length may be infinite.

## Specifying Contour Segments

The Contour Mode is specified with the command, CM. For example, CMXZ specifies contouring on the X and Z axes. Any axes that are not being used in the contouring mode may be operated in other modes.

A contour is described by position increments which are described with the command, CD x,y,z,w over a time interval, DT n. The parameter, n, specifies the time interval. The time interval is defined as 2n sample period (1 ms for TM1000), where n is a number between 1 and 8. The controller performs linear interpolation between the specified increments, where one point is generated for each sample. If the time interval changes for each segment, use CD x,y,z,w=n where n is the new DT value.

Consider, for example, the trajectory shown in Fig. 6.13. The position X may be described by the points:

| | |
|---|---|
| Point 1 | X=0 at T=0ms |
| Point 2 | X=48 at T=4ms |
| Point 3 | X=288 at T=12ms |
| Point 4 | X=336 at T=28ms |

The same trajectory may be represented by the increments

| | | | |
|---|---|---|---|
| Increment 1 | DX=48 | Time=4 | DT=2 |
| Increment 2 | DX=240 | Time=8 | DT=3 |
| Increment 3 | DX=48 | Time=16 | DT=4 |

When the controller receives the command to generate a trajectory along these points, it interpolates linearly between the points. The resulting interpolated points include the position 12 at 1 msec, position 24 at 2 msec, etc.

The programmed commands to specify the above example are:

```
#A
CMX                     Specifies X axis for contour mode
CD 48=2                 Specifies first position increment and time interval, 2^2 ms
CD 240=3                Specifies second position increment and time interval, 2^3 ms
CD 48=4                 Specifies the third position increment and time interval, 2^4 ms
CD 0=0                  End Contour buffer
#Wait;JP#Wait,_CM<>511  Wait until path is done
EN
```

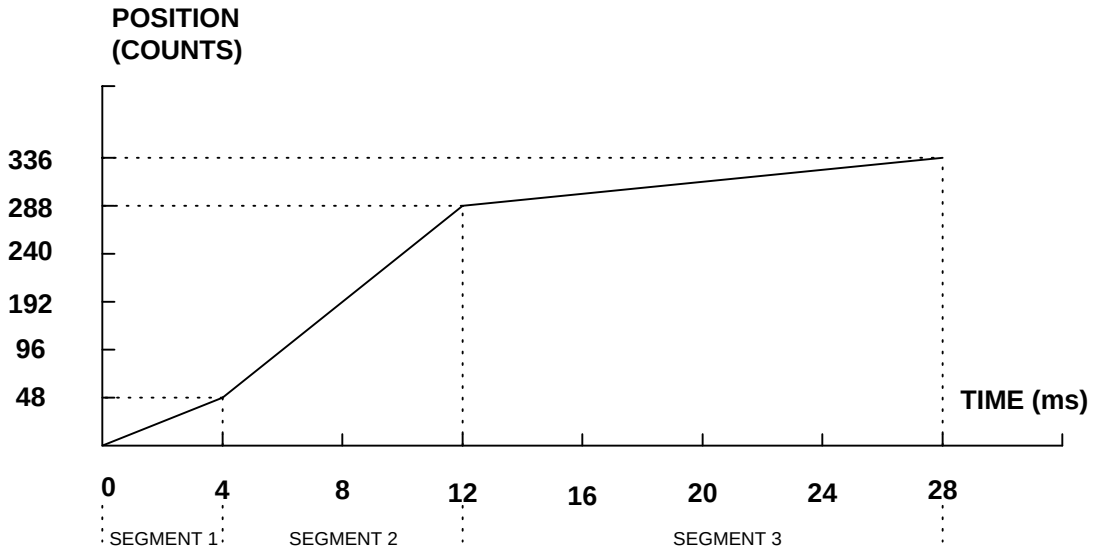



*Figure 6.13 - The Required Trajectory*

## Additional Commands

_CM gives the amount of space available in the contour buffer (511 maximum). Zero parameters for DT followed by zero parameters for CD exit the contour mode.

If no new data record is found and the controller is still in the contour mode, the controller waits for new data. No new motion commands are generated while waiting. If bad data is received, the controller responds with a ?.

## Command Summary - Contour Mode

| COMMAND | DESCRIPTION |
|---|---|
| CM XYZW | Specifies which axes for contouring mode. Any non-contouring axes may be operated in other modes. |
| CM ABCDEFGH | Contour axes for DMC-4080 |
| CD x,y,z,w | Specifies position increment over time interval. Range is +/-32,000. CD 0,0,0.. .=0 ends the contour buffer. This is much like the LE or VE commands. |

| | |
|---|---|
| CD a,b,c,d,e,f,g,h | Position increment data for DMC-4080 |
| DT n | Specifies time interval $2^n$ sample periods (1 ms for TM1000) for position increment, where n is an integer between 1 and 8. Zero ends contour mode. If n does not change, it does not need to be specified with each CD. |
| _CM | Amount of space left in contour buffer (511 maximum) |

## General Velocity Profiles

The Contour Mode is ideal for generating any arbitrary velocity profiles. The velocity profile can be specified as a mathematical function or as a collection of points.

The design includes two parts: Generating an array with data points and running the program.

## Generating an Array - An Example

Consider the velocity and position profiles shown in Fig. 6.14. The objective is to rotate a motor a distance of 6000 counts in 120 ms. The velocity profile is sinusoidal to reduce the jerk and the system vibration. If we describe the position displacement in terms of A counts in B milliseconds, we can describe the motion in the following manner:

$$\omega = \frac{A}{B}\left(1 - \cos(2\pi/B)\right)$$

$$X = \frac{AT}{B} - \frac{A}{2\pi}\sin(2\pi/B)$$

Note: ω is the angular velocity; X is the position; and T is the variable, time, in milliseconds.

In the given example, A=6000 and B=120, the position and velocity profiles are:

$X = 50T - (6000/2\pi) \sin (2\pi T/120)$

Note that the velocity, ω, in count/ms, is

$\omega = 50 [1 - \cos 2\pi T/120]$

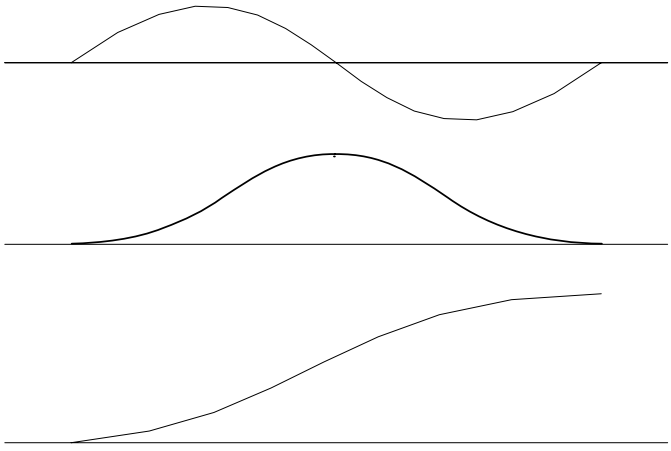


*Figure 6.14 - Velocity Profile with Sinusoidal Acceleration*

The DMC-40x0 can compute trigonometric functions. However, the argument must be expressed in degrees. Using our example, the equation for X is written as:

$X = 50T - 955 \sin 3T$

A complete program to generate the contour movement in this example is given below. To generate an array, we compute the position value at intervals of 8 ms. This is stored at the array POS. Then, the difference between the positions is computed and is stored in the array DIF. Finally the motors are run in the contour mode.

## Contour Mode Example

| INSTRUCTION | INTERPRETATION |
|---|---|
| #POINTS | Program defines X points |
| DM POS[16] | Allocate memory |
| DM DIF[15] | |
| C=0 | Set initial conditions, C is index |
| T=0 | T is time in ms |
| #A | |
| V1=50*T | |
| V2=3*T | Argument in degrees |
| V3=-955*@SIN[V2]+V1 | Compute position |
| V4=@INT[V3] | Integer value of V3 |
| POS[C]=V4 | Store in array POS |
| T=T+8 | |
| C=C+1 | |
| JP #A,C<16 | |
| #B | Program to find position differences |
| C=0 | |
| #C | |
| D=C+1 | |
| DIF[C]=POS[D]-POS[C] | Compute the difference and store |
| C=C+1 | |
| JP #C,C<15 | |
| | |
| #RUN | Program to run motor |
| CMX | Contour Mode |
| DT3 | 8 millisecond intervals |
| C=0 | |
| #E | |
| CD DIF[C] | Contour Distance is in DIF |
| C=C+1 | |
| JP #E,C<15 | |
| CD 0=0 | End contour buffer |
| #Wait;JP#Wait,_CM<>511 | Wait until path is done |
| EN | End the program |

## Teach (Record and Play-Back)

Several applications require teaching the machine a motion trajectory. Teaching can be accomplished using the DMC-40x0 automatic array capture feature to capture position data. The captured data may then be played back in the contour mode. The following array commands are used:

| | |
|---|---|
| DM C[n] | Dimension array |
| RA C[] | Specify array for automatic record (up to 4 for DMC-4040) |
| RD _TPX | Specify data for capturing (such as _TPX or _TPZ) |
| RC n,m | Specify capture time interval where n is $2^n$ sample periods (1 ms for TM1000), m is number of records to be captured |
| RC? or _RC | Returns a 1 if recording |

### Record and Playback Example:

```
#RECORD                 Begin Program
DM XPOS[501]            Dimension array with 501 elements
RA XPOS[]               Specify automatic record
RD _TPX                 Specify X position to be captured
MOX                     Turn X motor off
RC2                     Begin recording; 4 msec interval (at TM1000)
#A;JP#A,_RC=1           Continue until done recording
#COMPUTE                Compute DX
DM DX[500]              Dimension Array for DX
C=0                     Initialize counter
#L                      Label
D=C+1
DELTA=XPOS[D]-XPOS[C]   Compute the difference
DX[C]=DELTA             Store difference in array
C=C+1                   Increment index
JP #L,C<500             Repeat until done
#PLAYBCK                Begin Playback
CMX                     Specify contour mode
DT2                     Specify time increment
I=0                     Initialize array counter
#B                      Loop counter
CD DX[I]; I=I+1         Specify contour data I=I+1 Increment array counter
JP #B,I<500             Loop until done
CD 0=0                  End countour buffer
#Wait;JP#Wait,_CM<>511  Wait until path is done
EN                      End program
```

For additional information about automatic array capture, see Chapter 7, Arrays.

## Virtual Axis

The DMC-40x0 controller has two additional virtual axes designated as the M and N axes. These axes have no encoder and no DAC. However, they can be commanded by the commands:

```
AC, DC, JG, SP, PR, PA, BG, IT, GA, VM, VP, CR, ST, DP, RP
```

The main use of the virtual axes is to serve as a virtual master in ECAM modes, and to perform an unnecessary part of a vector mode. These applications are illustrated by the following examples.

## ECAM Master Example

Suppose that the motion of the XY axes is constrained along a path that can be described by an electronic cam table. Further assume that the ecam master is not an external encoder but has to be a controlled variable.

This can be achieved by defining the N axis as the master with the command EAN and setting the modulo of the master with a command such as EMN= 4000. Next, the table is constructed. To move the constrained axes, simply command the N axis in the jog mode or with the PR and PA commands.

For example,

```
PAN = 2000
BGN
```

will cause the XY axes to move to the corresponding points on the motion cycle.

### Sinusoidal Motion Example

The x axis must perform a sinusoidal motion of 10 cycles with an amplitude of 1000 counts and a frequency of 20 Hz.

This can be performed by commanding the X and N axes to perform circular motion. Note that the value of VS must be

$$VS=2\pi * R * F$$

where R is the radius, or amplitude and F is the frequency in Hz.

Set VA and VD to maximum values for the fastest acceleration.

| INSTRUCTION | INTERPRETATION |
|---|---|
| VMXN | Select Axes |
| VA 68000000 | Maximum Acceleration |
| VD 68000000 | Maximum Deceleration |
| VS 125664 | VS for 20 Hz |
| CR 1000, -90, 3600 | Ten Cycles |
| VE | |
| BGS | |

# Stepper Motor Operation

When configured for stepper motor operation, several commands are interpreted differently than from servo mode. The following describes operation with stepper motors.

## Specifying Stepper Motor Operation

Stepper motor operation is specified by the command MT. The argument for MT is as follows:

2 specifies a stepper motor with active low step output pulses
-2 specifies a stepper motor with active high step output pulses
2.5 specifies a stepper motor with active low step output pulses and reversed direction
-2.5 specifies a stepper motor with active high step output pulse and reversed direction

### Stepper Motor Smoothing

The command, KS, provides stepper motor smoothing. The effect of the smoothing can be thought of as a simple Resistor-Capacitor (single pole) filter. The filter occurs after the motion profiler and has the effect of smoothing out the spacing of pulses for a more smooth operation of the stepper motor. Use of KS is most applicable when operating in full step or half step operation. KS will cause the step pulses to be delayed in accordance with the time constant specified.

When operating with stepper motors, you will always have some amount of stepper motor smoothing, KS. Since this filtering effect occurs after the profiler, the profiler may be ready for additional moves before all of the step pulses have gone through the filter. It is important to consider this effect since steps may be lost if the controller is commanded to generate an additional move before the previous move has been completed. See the discussion below, Monitoring Generated Pulses vs. Commanded Pulses.

The general motion smoothing command, IT, can also be used. The purpose of the command, IT, is to smooth out the motion profile and decrease 'jerk' due to acceleration.

### Monitoring Generated Pulses vs. Commanded Pulses

For proper controller operation, it is necessary to make sure that the controller has completed generating all step pulses before making additional moves. This is most particularly important if you are moving back and forth. For example, when operating with servo motors, the trippoint AM (After Motion) is used to determine when the motion

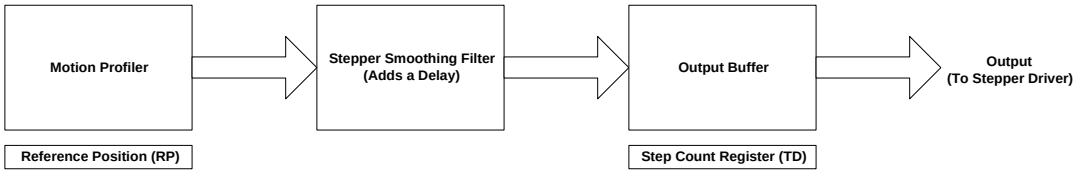
profiler is complete and is prepared to execute a new motion command. However when operating in stepper mode, the controller may still be generating step pulses when the motion profiler is complete. This is caused by the stepper motor smoothing filter, KS. To understand this, consider the steps the controller executes to generate step pulses:

First, the controller generates a motion profile in accordance with the motion commands.

Second, the profiler generates pulses as prescribed by the motion profile. The pulses that are generated by the motion profiler can be monitored by the command, RP (Reference Position). RP gives the absolute value of the position as determined by the motion profiler. The command, DP, can be used to set the value of the reference position. For example, DP 0, defines the reference position of the X axis to be zero.

Third, the output of the motion profiler is filtered by the stepper smoothing filter. This filter adds a delay in the output of the stepper motor pulses. The amount of delay depends on the parameter which is specified by the command, KS. As mentioned earlier, there will always be some amount of stepper motor smoothing. The default value for KS is 1.313 which corresponds to a time constant of 3.939 sample periods.

Fourth, the output of the stepper smoothing filter is buffered and is available for input to the stepper motor driver. The pulses which are generated by the smoothing filter can be monitored by the command, TD (Tell Dual). TD gives the absolute value of the position as determined by actual output of the buffer. The command, DP sets the value of the step count register as well as the value of the reference position. For example, DP 0, defines the reference position of the X axis to be zero.




### Motion Complete Trippoint

When used in stepper mode, the MC command will hold up execution of the proceeding commands until the controller has generated the same number of steps out of the step count register as specified in the commanded position. The MC trippoint (Motion Complete) is generally more useful than AM trippoint (After Motion) since the step pulses can be delayed from the commanded position due to stepper motor smoothing.

## Using an Encoder with Stepper Motors

An encoder may be used on a stepper motor to check the actual motor position with the commanded position. If an encoder is used, it must be connected to the main encoder input. Note: The auxiliary encoder is not available while operating with stepper motors. The position of the encoder can be interrogated by using the command, TP. The position value can be defined by using the command, DE.

**Note:** Closed loop operation with a stepper motor is not possible.

## Command Summary - Stepper Motor Operation

| COMMAND | DESCRIPTION |
|---|---|
| DE | Define Encoder Position (When using an encoder) |
| DP | Define Reference Position and Step Count Register |
| IT | Motion Profile Smoothing - Independent Time Constant |
| KS | Stepper Motor Smoothing |
| MT | Motor Type (2,-2,2.5 or -2.5 for stepper motors) |
| RP | Report Commanded Position |

| | |
|---|---|
| TD | Report number of step pulses generated by controller |
| TP | Tell Position of Encoder |

## Operand Summary - Stepper Motor Operation

| OPERAND | DESCRIPTION |
|---|---|
| _DEx | Contains the value of the step count register for the 'x' axis |
| _DPx | Contains the value of the main encoder for the 'x' axis |
| _ITx | Contains the value of the Independent Time constant for the 'x' axis |
| _KSx | Contains the value of the Stepper Motor Smoothing Constant for the 'x' axis |
| _MTx | Contains the motor type value for the 'x' axis |
| _RPx | Contains the commanded position generated by the profiler for the 'x' axis |
| _TDx | Contains the value of the step count register for the 'x' axis |
| _TPx | Contains the value of the main encoder for the 'x' axis |

# Stepper Position Maintenance Mode (SPM)

The Galil controller can be set into the Stepper Position Maintenance (SPM) mode to handle the event of stepper motor position error. The mode looks at position feedback from the main encoder and compares it to the commanded step pulses. The position information is used to determine if there is any significant difference between the commanded and the actual motor positions. If such error is detected, it is updated into a command value for operator use. In addition, the SPM mode can be used as a method to correct for friction at the end of a microstepping move. This capability provides closed-loop control at the application program level. SPM mode can be used with Galil and non-Galil step drives.

SPM mode is configured, executed, and managed with seven commands. This mode also utilizes the #POSERR automatic subroutine allowing for automatic user-defined handling of an error event.

## Internal Controller Commands (user can query):

QS Error Magnitude (pulses)

## User Configurable Commands (user can query & change):

OE Profiler Off-On Error
YA Step Drive Resolution (pulses / full motor step)
YB Step Motor Resolution (full motor steps / revolution)
YC Encoder Resolution (counts / revolution)
YR Error Correction (pulses)
YS Stepper Position Maintenance enable, status

A pulse is defined by the resolution of the step drive being used. Therefore, one pulse could be a full step, a half step or a microstep.

When a Galil controller is configured for step motor operation, the step pulse output by the controller is internally fed back to the auxiliary encoder register. For SPM the feedback encoder on the stepper will connect to the main encoder port. Enabling the SPM mode on a controller with YS=1 executes an internal monitoring of the auxiliary and main encoder registers for that axis or axes. Position error is then tracked in step pulses between these two registers (QS command).

$$QS = TD - \frac{TP \times YA \times YB}{YC}$$

Where TD is the auxiliary encoder register(step pulses) and TP is the main encoder register(feedback encoder). Additionally, YA defines the step drive resolution where YA = 1 for full stepping or YA = 2 for half stepping. The full range of YA is up to YA = 9999 for microstepping drives.

## Error Limit

The value of QS is internally monitored to determine if it exceeds a preset limit of three full motor steps. Once the value of QS exceeds this limit, the controller then performs the following actions:

1. The motion is maintained or is stopped, depending on the setting of the OE command. If OE=0 the axis stays in motion, if OE=1 the axis is stopped.
2. YS is set to 2, which causes the automatic subroutine labeled #POSERR to be executed.

## Correction

A correction move can be commanded by assigning the value of QS to the YR correction move command. The correction move is issued only after the axis has been stopped. After an error correction move has completed and QS is less than three full motor steps, the YS error status bit is automatically reset back to 1; indicating a cleared error.

### Example: SPM Mode Setup

The following code demonstrates what is necessary to set up SPM mode for a full step drive, a half step drive, and a 1/64th microstepping drive for an axis with a 1.8° step motor and 4000 count/rev encoder. Note the necessary difference is with the YA command.

**Full-Stepping Drive, X axis:**

```
#SETUP
OE1;          Set the profiler to stop axis upon error
KS16;         Set step smoothing
MT-2;         Motor type set to stepper
YA1;          Step resolution of the full-step drive
YB200;        Motor resolution (full steps per revolution)
YC4000;       Encoder resolution (counts per revolution)
SHX;          Enable axis
WT50;         Allow slight settle time
YS1;          Enable SPM mode
```

**Half-Stepping Drive, X axis:**

```
#SETUP
OE1;          Set the profiler to stop axis upon error
KS16;         Set step smoothing
MT-2;         Motor type set to stepper
YA2;          Step resolution of the half-step drive
YB200;        Motor resolution (full steps per revolution)
YC4000;       Encoder resolution (counts per revolution)
SHX;          Enable axis
WT50;         Allow slight settle time
```

```
YS1;          Enable SPM mode
```

**1/64th Step Microstepping Drive, X axis:**

```
#SETUP
OE1;          Set the profiler to stop axis upon error
KS16;         Set step smoothing
MT-2;         Motor type set to stepper
YA64;         Step resolution of the microstepping drive
YB200;        Motor resolution (full steps per revolution)
YC4000;       Encoder resolution (counts per revolution)
SHX;          Enable axis
WT50;         Allow slight settle time
YS1;          Enable SPM mode
```

## Example: Error Correction

The following code demonstrates what is necessary to set up SPM mode for the X axis, detect error, stop the motor, correct the error, and return to the main code. The drive is a full step drive, with a 1.8° step motor and 4000 count/rev encoder.

```
#SETUP
OE1;                  Set the profiler to stop axis upon error
KS16;                 Set step smoothing
MT-2,-2,-2,-2;        Motor type set to stepper
YA2;                  Step resolution of the drive
YB200;                Motor resolution (full steps per revolution)
YC4000;               Encoder resolution (counts per revolution)
SHX;                  Enable axis
WT100;                Allow slight settle time


#MOTION               Perform motion
SP512;                Set the speed
PR1000;               Prepare mode of motion
BGX;                  Begin motion
#LOOP;JP#LOOP;        Keep thread zero alive for #POSERR to run in

REM  When error occurs, the axis will stop due to OE1.  In
REM #POSERR, query the status YS and the error QS, correct,
REM and return to the main code.

#POSERR;              Automatic subroutine is called when YS=2
WT100;                Wait helps user see the correction
spsave=_SPX;          Save current speed setting
JP#RETURN,_YSX<>2;    Return to thread zero if invalid error
SP64;                 Set slow speed setting for correction
MG"ERROR= ",_QSX
YRX=_QSX;             Else, error is valid, use QS for correction
MCX;                  Wait for motion to complete
MG"CORRECTED, ERROR NOW= ",_QSX
```

```
WT100;              Wait helps user see the correction

#RETURN
SPX=spsave;         Return the speed to previous setting
REO;                Return from #POSERR
```

### Example: Friction Correction

The following example illustrates how the SPM mode can be useful in correcting for X axis friction after each move when conducting a reciprocating motion. The drive is a 1/64th microstepping drive with a 1.8° step motor and 4000 count/rev encoder.

```
#SETUP;             Set the profiler to continue upon error
KS16;               Set step smoothing
MT-2,-2,-2;         Motor type set to stepper
YA64;               Step resolution of the microstepping drive
YB200;              Motor resolution (full steps per revolution)
YC4000;             Encoder resolution (counts per revolution)
SHX;                Enable axis
WT50;               Allow slight settle time
YS1;                Enable SPM mode

#MOTION;              Perform motion
SP16384;              Set the speed
PR10000;              Prepare mode of motion
BGX;                  Begin motion
MCX
JS#CORRECT;           Move to correction
#MOTION2
SP16384;              Set the speed
PR-10000;             Prepare mode of motion
BGX;                  Begin motion
MCX
JS#CORRECT;           Move to correction
JP#MOTION
#CORRECT;             Correction code
spx=_SPX
#LOOP;                Save speed value
SP2048;               Set a new slow correction speed
WT100;                Stabilize
JP#END,@ABS[_QSX]<10; End correction if error is within defined tolerance
YRX=_QSX;             Correction move
MCX
WT100;                Stabilize
JP#LOOP;              Keep correcting until error is within tolerance
#END;                 End #CORRECT subroutine, returning to code
SPX=spx
EN
```

# Dual Loop (Auxiliary Encoder)

The DMC-40x0 provides an interface for a second encoder for each axis except for axes configured for stepper motor operation and axis used in circular compare. When used, the second encoder is typically mounted on the motor or the load, but may be mounted in any position. The most common use for the second encoder is backlash compensation, described below.

The second encoder may be a standard quadrature type, or it may provide pulse and direction. The controller also offers the provision for inverting the direction of the encoder rotation. The main and the auxiliary encoders are configured with the CE command. The command form is CE x,y,z,w (or a,b,c,d,e,f,g,h for controllers with more than 4 axes) where the parameters x,y,z,w each equal the sum of two integers m and n. m configures the main encoder and n configures the auxiliary encoder.

### Using the CE Command

| m= | Main Encoder | n= | Second Encoder |
|---|---|---|---|
| 0 | Normal quadrature | 0 | Normal quadrature |
| 1 | Pulse & direction | 4 | Pulse & direction |
| 2 | Reverse quadrature | 8 | Reversed quadrature |
| 3 | Reverse pulse & direction | 12 | Reversed pulse & direction |

For example, to configure the main encoder for reversed quadrature, m=2, and a second encoder of pulse and direction, n=4, the total is 6, and the command for the X axis is:

```
CE 6
```

### Additional Commands for the Auxiliary Encoder

The command, DE x,y,z,w, can be used to define the position of the auxiliary encoders. For example,

```
DE 0,500,-30,300
```

sets their initial values. The positions of the auxiliary encoders may be interrogated with the command, DE?. For example:

```
DE ?,,?
```

returns the value of the X and Z auxiliary encoders.

The auxiliary encoder position may be assigned to variables with the instructions

```
V1= _DEX
```

The command, TD XYZW, returns the current position of the auxiliary encoder.

The command, DV 1,1,1,1, configures the auxiliary encoder to be used for backlash compensation.

## Backlash Compensation

There are two methods for backlash compensation using the auxiliary encoders:

1. Continuous dual loop
2. Sampled dual loop

To illustrate the problem, consider a situation in which the coupling between the motor and the load has a backlash. To compensate for the backlash, position encoders are mounted on both the motor and the load.

The continuous dual loop combines the two feedback signals to achieve stability. This method requires careful system tuning, and depends on the magnitude of the backlash. However, once successful, this method compensates for the backlash continuously.

The second method, the sampled dual loop, reads the load encoder only at the end point and performs a correction. This method is independent of the size of the backlash. However, it is effective only in point-to-point motion systems which require position accuracy only at the endpoint.

### Continuous Dual Loop - Example

Connect the load encoder to the main encoder port and connect the motor encoder to the dual encoder port. The dual loop method splits the filter function between the two encoders. It applies the KP (proportional) and KI (integral) terms to the position error, based on the load encoder, and applies the KD (derivative) term to the motor encoder. This method results in a stable system.

The dual loop method is activated with the instruction DV (Dual Velocity), where

```
DV      1,1,1,1
```

activates the dual loop for the four axes and

```
DV      0,0,0,0
```

disables the dual loop.

**Note**: that the dual loop compensation depends on the backlash magnitude, and in extreme cases will not stabilize the loop. The proposed compensation procedure is to start with KP=0, KI=0 and to maximize the value of KD under the condition DV1. Once KD is found, increase KP gradually to a maximum value, and finally, increase KI, if necessary.

### Sampled Dual Loop - Example

In this example, we consider a linear slide which is run by a rotary motor via a lead screw. Since the lead screw has a backlash, it is necessary to use a linear encoder to monitor the position of the slide. For stability reasons, it is best to use a rotary encoder on the motor.

Connect the rotary encoder to the X-axis and connect the linear encoder to the auxiliary encoder of X. Assume that the required motion distance is one inch, and that this corresponds to 40,000 counts of the rotary encoder and 10,000 counts of the linear encoder.

The design approach is to drive the motor a distance, which corresponds to 40,000 rotary counts. Once the motion is complete, the controller monitors the position of the linear encoder and performs position corrections.

This is done by the following program.

| INSTRUCTION | INTERPRETATION |
|---|---|
| #DUALOOP | Label |
| CE 0 | Configure encoder |
| DE0 | Set initial value |
| PR 40000 | Main move |
| BGX | Start motion |
| #Correct | Correction loop |
| AMX | Wait for motion completion |
| V1=10000-_DEX | Find linear encoder error |
| V2=-_TEX/4+V1 | Compensate for motor error |
| JP#END,@ABS[V2]<2 | Exit if error is small |
| PR V2*4 | Correction move |
| BGX | Start correction |
| JP#CORRECT | Repeat |
| #END | |
| EN | |

# Motion Smoothing

The DMC-40x0 controller allows the smoothing of the velocity profile to reduce the mechanical vibration of the system.

Trapezoidal velocity profiles have acceleration rates which change abruptly from zero to maximum value. The discontinuous acceleration results in jerk which causes vibration. The smoothing of the acceleration profile leads to a continuous acceleration profile and reduces the mechanical shock and vibration.

## Using the IT Command:

When operating with servo motors, motion smoothing can be accomplished with the IT command. This command filters the acceleration and deceleration functions to produce a smooth velocity profile. The resulting velocity profile, has continuous acceleration and results in reduced mechanical vibrations.

The smoothing function is specified by the following commands:

| | |
|---|---|
| IT x,y,z,w | Independent time constant |

The command, IT, is used for smoothing independent moves of the type JG, PR, PA and to smooth vector moves of the type VM and LM.

The smoothing parameters, x,y,z,w and n are numbers between 0 and 1 and determine the degree of filtering. The maximum value of 1 implies no filtering, resulting in trapezoidal velocity profiles. Smaller values of the smoothing parameters imply heavier filtering and smoother moves.

The following example illustrates the effect of smoothing. Fig. 6.15 shows the trapezoidal velocity profile and the modified acceleration and velocity.

Note that the smoothing process results in longer motion time.

### Example - Smoothing

```
PR 20000        Position
AC 100000       Acceleration
DC 100000       Deceleration
SP 5000         Speed
IT .5           Filter for smoothing
BG X            Begin
```

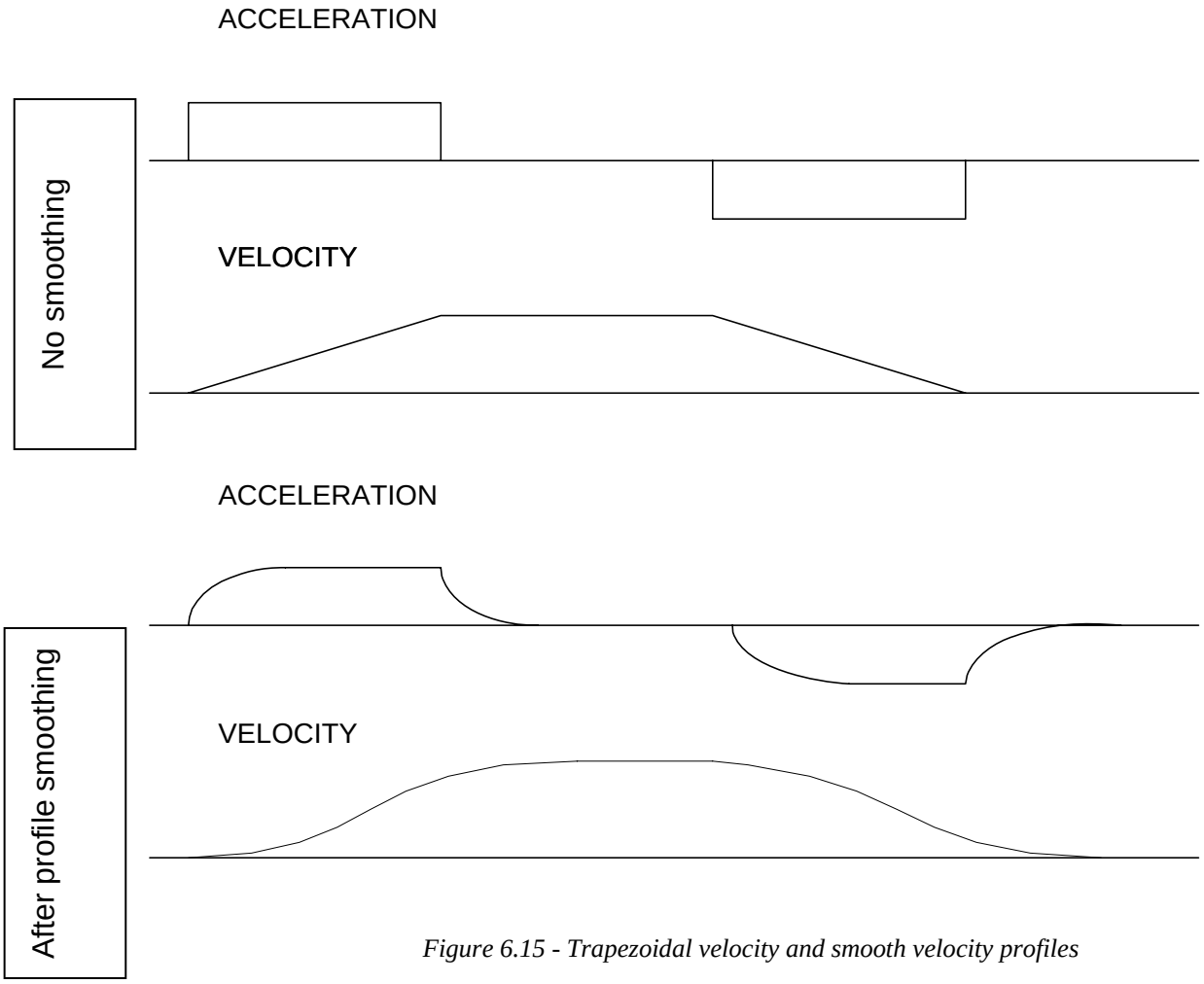



*Figure 6.15 - Trapezoidal velocity and smooth velocity profiles*

### Using the KS Command (Step Motor Smoothing):

When operating with step motors, motion smoothing can be accomplished with the command, KS. The KS command smoothes the frequency of step motor pulses. Similar to the command IT, this produces a smooth velocity profile.

The step motor smoothing is specified by the following command:

| | |
|---|---|
| KS x,y,z,w | where x,y,z,w is an integer from 0.25 to 64 and represents the amount of smoothing |

The smoothing parameters, x,y,z,w and n are numbers between 0.25 and 64 and determine the degree of filtering. The minimum value of 0.25 implies no filtering, resulting in trapezoidal velocity profiles. Larger values of the smoothing parameters imply heavier filtering and smoother moves.

Note that KS is valid only for step motors.

## Homing

The Find Edge (FE) and Home (HM) instructions may be used to home the motor to a mechanical reference. This reference is connected to the Home input line. The HM command initializes the motor to the encoder index pulse in addition to the Home input. The configure command (CN) is used to define the polarity of the home input.

The Find Edge (FE) instruction is useful for initializing the motor to a home switch. The home switch is connected to the Homing Input. When the Find Edge command and Begin is used, the motor will accelerate up to the slew speed and slew until a transition is detected on the Homing line. The motor will then decelerate to a stop. A high deceleration value must be input before the find edge command is issued for the motor to decelerate rapidly after sensing the home switch. The Home (HM) command can be used to position the motor on the index pulse after the home switch is detected. This allows for finer positioning on initialization. The HM command and BG command causes the following sequence of events to occur.

## Stage 1:

Upon begin, the motor accelerates to the slew speed specified by the JG or SP commands. The direction of its motion is determined by the state of the homing input. If _HMX reads 1 initially, the motor will go in the reverse direction first (direction of decreasing encoder counts). If _HMX reads 0 initially, the motor will go in the forward direction first. CN is the command used to define the polarity of the home input. With CN,-1 (the default value) a normally open switch will make _HMX read 1 initially, and a normally closed switch will make _HMX read zero. Furthermore, with CN,1 a normally open switch will make _HMX read 0 initially, and a normally closed switch will make _HMX read 1. Therefore, the CN command will need to be configured properly to ensure the correct direction of motion in the home sequence.

Upon detecting the home switch changing state, the motor begins decelerating to a stop.

**Note:** The direction of motion for the FE command also follows these rules for the state of the home input.

## Stage 2:

The motor then traverses at HV counts/sec in the opposite direction of Stage 1 until the home switch toggles again. If Stage 3 is in the opposite direction of Stage 2, the motor will stop immediately at this point and change direction. If Stage 2 is in the same direction as Stage 3, the motor will never stop, but will smoothly continue into Stage 3.

## Stage 3:

The motor traverses forward at HV counts/sec until the encoder index pulse is detected. The motor then decelerates to a stop and goes back to the index.

The DMC-40x0 defines the home position as the position at which the index was detected and sets the encoder reading at this point to zero.

The 4 different motion possibilities for the home sequence are shown in the following table.

| | | | Direction of Motion | | |
|---|---|---|---|---|---|
| Switch Type | CN Setting | Initial _HMX state | Stage 1 | Stage 2 | Stage 3 |
| Normally Open | CN,-1 | 1 | Reverse | Forward | Forward |
| Normally Open | CN,1 | 0 | Forward | Reverse | Forward |
| Normally Closed | CN,-1 | 0 | Forward | Reverse | Forward |
| Normally Closed | CN,1 | 1 | Reverse | Forward | Forward |

### Example: Homing

| Instruction | Interpretation |
|---|---|
| `#HOME` | `Label` |
| `CN,-1` | `Configure the polarity of the home input` |
| `AC 1000000` | `Acceleration Rate` |

```
DC 1000000            Deceleration Rate
SP 5000               Speed for Home Search
HM                    Home
BG                    Begin Motion
AM                     After Complete
MG "AT HOME"          Send Message
EN                    End
```

Figure 6.16 shows the velocity profile from the homing sequence of the example program above. For this profile, the switch is normally closed and CN,-1.

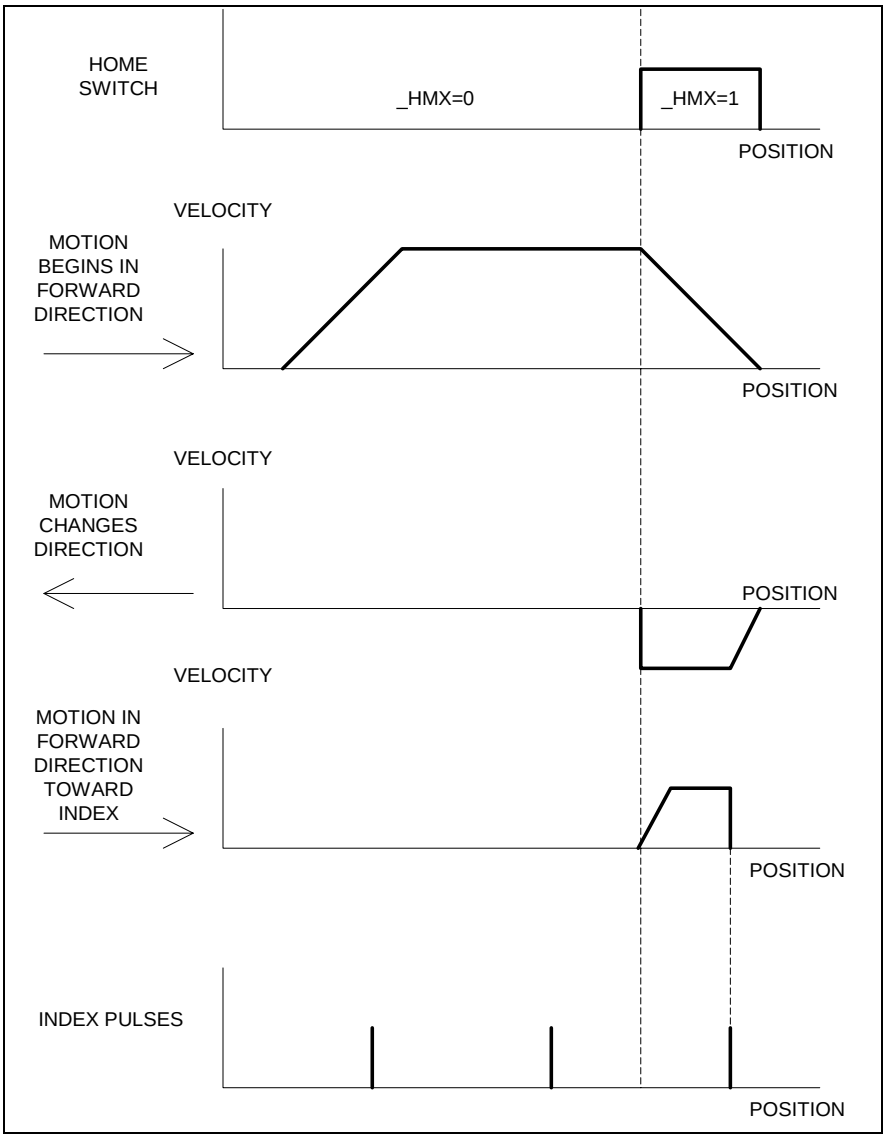



*Figure 6.16 – Homing Sequence for Normally Closed Switch and CN,-1*

### Example: Find Edge

```
#EDGE                Label
AC 2000000           Acceleration rate
DC 2000000           Deceleration rate
SP 8000              Speed
FE                   Find edge command
BG                   Begin motion
AM                   After complete
MG "FOUND HOME"      Send message
DP 0                 Define position as 0
EN                   End
```

## Command Summary - Homing Operation

| command | description |
|---|---|
| FE XYZW | Find Edge Routine. This routine monitors the Home Input |
| FI XYZW | Find Index Routine - This routine monitors the Index Input |
| HM XYZW | Home Routine - This routine combines FE and FI as Described Above |
| SC XYZW | Stop Code |
| TS XYZW | Tell Status of Switches and Inputs |

## Operand Summary - Homing Operation

| operand | Description |
|---|---|
| _HMx | Contains the value of the state of the Home Input |
| _SCx | Contains stop code |
| _TSx | Contains status of switches and inputs |

# High Speed Position Capture (The Latch Function)

Often it is desirable to capture the position precisely for registration applications. The DMC-40x0 provides a position latch feature. This feature allows the position of the main or auxiliary encoders of X,Y,Z or W to be captured within 25 microseconds of an external low input signal (or index pulse). The general inputs 1 through 4 and 9 thru 12 correspond to each axis.

| 1 through 4: | | 9 through 12 | |
|---|---|---|---|
| IN1 | X-axis latch | IN9 | E-axis latch |
| IN2 | Y-axis latch | IN10 | F-axis latch |
| IN3 | Z-axis latch | IN11 | G-axis latch |
| IN4 | W-axis latch | IN12 | H-axis latch |

**Note:** To insure a position capture within 25 microseconds, the input signal must be a transition from high to low.

The DMC-40x0 software commands, AL and RL, are used to arm the latch and report the latched position. The steps to use the latch are as follows:

1. Give the AL XYZW command or ABCDEFGH for DMC-4080, to arm the latch for the main encoder and ALSXSYSZSW for the auxiliary encoders.
2. Test to see if the latch has occurred (Input goes low) by using the _AL X or Y or Z or W command. Example, V1=_ALX returns the state of the X latch into V1. V1 is 1 if the latch has not occurred.
3. After the latch has occurred, read the captured position with the RL XYZW command or _RL XYZW.

**Note:** The latch must be re-armed after each latching event.

**Example:**

```
#Latch          Latch program
JG,5000         Jog Y
BG Y            Begin motion on Y axis
AL Y            Arm Latch for Y axis
#Wait           #Wait label for loop
JP #Wait,_ALY=1 Jump to #Wait label if latch has not occurred
Result=_RLY     Set value of variable 'Result' equal to the report position of y axis
Result=         Print result
EN              End
```

# Fast Update Rate Mode

The DMC-40x0 can operate with much faster servo update rates than the default of every millisecond. This mode is known as 'fast mode' and allows the controller to operate with the following update rates:

| | |
|---|---|
| DMC-4010 | 31.25 μsec |
| DMC-4020 | 31.25 μsec |
| DMC-4030 | 62.5 μsec |
| DMC-4040 | 62.5 μsec |
| DMC-4050 | 93.75 μsec |
| DMC-4060 | 93.75 μsec |
| DMC-4070 | 125 μsec |
| DMC-4080 | 125 μsec |

In order to run the DMC-40x0 motion controller in fast mode, the fast firmware must be uploaded. This can be done through the GalilTools communication software. The fast firmware is included with the original DMC-40x0 utilities.

In order to set the desired update rates, use the command TM.

When the controller is operating with the fast firmware, the following functions are **disabled**:

Gearing mode

Ecam mode

Pole (PL)

Analog Feedback (AF)

Stepper Motor Operation (MT 2,-2,2.5,-2.5)

Trippoints in thread 2-8

Tell Velocity Interrogation Command (TV)

Aux Encoders (TD)

Dual Velocity (DV)

Peak Torque Limit (TK)

Notch Filter (NB, NF, NZ )

# Chapter 7 Application Programming

## Overview

The DMC-40x0 provides a powerful programming language that allows users to customize the controller for their particular application. Programs can be downloaded into the DMC-40x0 memory freeing the host computer for other tasks. However, the host computer can send commands to the controller at any time, even while a program is being executed. Only ASCII commands can be used for application programming.

In addition to standard motion commands, the DMC-40x0 provides commands that allow the DMC-40x0 to make its own decisions. These commands include conditional jumps, event triggers and subroutines. For example, the command JP#LOOP, n<10 causes a jump to the label #LOOP if the variable n is less than 10.

For greater programming flexibility, the DMC-40x0 provides user-defined variables, arrays and arithmetic functions. For example, with a cut-to-length operation, the length can be specified as a variable in a program which the operator can change as necessary.

The following sections in this chapter discuss all aspects of creating applications programs. The program memory size is 80 characters x 2000 lines.

## Using the DMC-40x0 Editor to Enter Programs

The GalilTools software package provides an editor and utilities that allow the upload and download of DMC programs to the motion controller.

Application programs for the DMC-40x0 may also be created and edited locally using the DMC-40x0.

The DMC-40x0 provides a line Editor for entering and modifying programs. The Edit mode is entered with the ED instruction. (Note: The ED command can only be given when the controller is in the non-edit mode, which is signified by a colon prompt).

In the Edit Mode, each program line is automatically numbered sequentially starting with 000. If no parameter follows the ED command, the editor prompter will default to the last line of the last program in memory. If desired, the user can edit a specific line number or label by specifying a line number or label following ED.

```
ED                  Puts Editor at end of last program
:ED 5               Puts Editor at line 5
:ED #BEGIN          Puts Editor at label #BEGIN
```

Line numbers appear as 000,001,002 and so on. Program commands are entered following the line numbers. Multiple commands may be given on a single line as long as the total number of characters doesn't exceed 80 characters per line.

While in the Edit Mode, the programmer has access to special instructions for saving, inserting and deleting program lines. These special instructions are listed below:

## Edit Mode Commands

<RETURN>

Typing the return key causes the current line of entered instructions to be saved. The editor will automatically advance to the next line. Thus, hitting a series of <RETURN> will cause the editor to advance a series of lines. Note, changes on a program line will not be saved unless a <return> is given.

<cntrl>P

The <cntrl>P command moves the editor to the previous line.

<cntrl>I

The <cntrl>I command inserts a line above the current line. For example, if the editor is at line number 2 and <cntrl>I is applied, a new line will be inserted between lines 1 and 2. This new line will be labeled line 2. The old line number 2 is renumbered as line 3.

<cntrl>D

The <cntrl>D command deletes the line currently being edited. For example, if the editor is at line number 2 and <cntrl>D is applied, line 2 will be deleted. The previous line number 3 is now renumbered as line number 2.

<cntrl>Q

The <cntrl>Q quits the editor mode. In response, the DMC-40x0 will return a colon.

After the Edit session is over, the user may list the entered program using the LS command. If no operand follows the LS command, the entire program will be listed. The user can start listing at a specific line or label using the operand n. A command and new line number or label following the start listing operand specifies the location at which listing is to stop.

Example:

| Instruction | Interpretation |
|---|---|
| `:LS` | `List entire program` |
| `:LS 5` | `Begin listing at line 5` |
| `:LS 5,9` | `List lines 5 thru 9` |
| `:LS #A,9` | `List line label #A thru line 9` |
| `:LS #A, #A +5` | `List line label #A and additional 5 lines` |

# Program Format

A DMC-40x0 program consists of DMC instructions combined to solve a machine control application. Action instructions, such as starting and stopping motion, are combined with Program Flow instructions to form the complete program. Program Flow instructions evaluate real-time conditions, such as elapsed time or motion complete, and alter program flow accordingly.

Each DMC-40x0 instruction in a program must be separated by a delimiter. Valid delimiters are the semicolon (;) or carriage return. The semicolon is used to separate multiple instructions on a single program line where the maximum number of instructions on a line is limited by 80 characters. A carriage return enters the final command on a program line.

## Using Labels in Programs

All DMC-40x0 programs must begin with a label and end with an End (EN) statement. Labels start with the pound (#) sign followed by a maximum of seven characters. The first character must be a letter; after that, numbers are permitted. Spaces are not permitted.

The maximum number of labels which may be defined is 510.

**Valid labels**

```
#BEGIN
#SQUARE
#X1
#BEGIN1
```

**Invalid labels**

```
#1Square
#123
```

A Simple Example Program:

```
#START                  Beginning of the Program
PR 10000,20000          Specify relative distances on X and Y axes
BG XY                   Begin Motion
AM                      Wait for motion complete
WT 2000                 Wait 2 sec
JP #START               Jump to label START
EN                      End of Program
```

The above program moves X and Y 10000 and 20000 units. After the motion is complete, the motors rest for 2 seconds. The cycle repeats indefinitely until the stop command is issued.

## Special Labels

The DMC-40x0 have some special labels, which are used to define input interrupt subroutines, limit switch subroutines, error handling subroutines, and command error subroutines. See section on Auto-Start Routine

The DMC-40x0 has a special label for automatic program execution. A program which has been saved into the controller's non-volatile memory can be automatically executed upon power up or reset by beginning the program with the label #AUTO. The program must be saved into non-volatile memory using the command, BP.

Automatic Subroutines for Monitoring Conditions

| | |
|---|---|
| #ININT | Label for Input Interrupt subroutine |
| #LIMSWI | Label for Limit Switch subroutine |
| #POSERR | Label for excess Position Error subroutine |
| #MCTIME | Label for timeout on Motion Complete trip point |
| #CMDERR | Label for incorrect command subroutine |

## Commenting Programs

### Using the command, NO or Apostrophe (‘)

The DMC-40x0 provides a command, NO, for commenting programs or single apostrophe. This command allows the user to include up to 78 characters on a single line after the NO command and can be used to include comments from the programmer as in the following example:

```
#PATH
‘ 2-D CIRCULAR PATH
VMXY
‘  VECTOR MOTION ON X AND Y
VS 10000
‘  VECTOR SPEED IS 10000
VP -4000,0
‘  BOTTOM LINE
CR 1500,270,-180
```

```
‘  HALF CIRCLE MOTION
VP 0,3000
‘  TOP LINE
CR 1500,90,-180
‘  HALF CIRCLE MOTION
VE
‘  END VECTOR SEQUENCE
BGS
‘  BEGIN SEQUENCE MOTION
EN
‘  END OF PROGRAM
```

**Note:** The NO command is an actual controller command. Therefore, inclusion of the NO commands will require process time by the controller.

# Executing Programs - Multitasking

The DMC-40x0 can run up to 8 independent programs simultaneously. These programs are called threads and are numbered 0 through 7, where 0 is the main thread. Multitasking is useful for executing independent operations such as PLC functions that occur independently of motion.

The main thread differs from the others in the following ways:

1. Only the main thread, thread 0, may use the input command, IN.
2. When input interrupts are implemented for limit switches, position errors or command errors, the subroutines are executed as thread 0.

To begin execution of the various programs, use the following instruction:

```
XQ #A, n
```

Where n indicates the thread number. To halt the execution of any thread, use the instruction

```
HX n
```

where n is the thread number.

Note that both the XQ and HX commands can be performed by an executing program.

The example below produces a waveform on Output 1 independent of a move.

```
#TASK1              Task1 label
AT0                 Initialize reference time
CB1                 Clear Output 1
#LOOP1              Loop1 label
AT 10               Wait 10 msec from reference time
SB1                 Set Output 1
AT -40              Wait 40 msec from reference time, then initialize reference
CB1                 Clear Output 1
JP #LOOP1           Repeat Loop1
#TASK2              Task2 label
XQ #TASK1,1         Execute Task1
#LOOP2              Loop2 label
PR 1000             Define relative distance
BGX                 Begin motion
```

```
AMX                  After motion done
WT 10                Wait 10 msec
JP #LOOP2,@IN[2]=1   Repeat motion unless Input 2 is low
HX                   Halt all tasks
```

The program above is executed with the instruction XQ #TASK2,0 which designates TASK2 as the main thread (i.e. Thread 0). #TASK1 is executed within TASK2.

# Debugging Programs

The DMC-40x0 provides commands and operands which are useful in debugging application programs. These commands include interrogation commands to monitor program execution, determine the state of the controller and the contents of the controllers program, array, and variable space. Operands also contain important status information which can help to debug a program.

## Trace Commands

The trace command causes the controller to send each line in a program to the host computer immediately prior to execution. Tracing is enabled with the command, TR1. TR0 turns the trace function off. Note: When the trace function is enabled, the line numbers as well as the command line will be displayed as each command line is executed.

**NOTE**: When the trace function is enabled, the line numbers as well as the command line will be displayed as each command line is executed.

Data which is output from the controller is stored in the output UART. The UART buffer can store up to 512 characters of information. In normal operation, the controller places output into the FIFO buffer. When the trace mode is enabled, the controller will send information to the UART buffer at a very high rate. In general, the UART will become full because the hardware handshake line will halt serial data until the correct data is read. When the UART becomes full, program execution will be delayed until it is cleared. If the user wants to avoid this delay, the command CW,1 can be given. This command causes the controller to throw away the data which can not be placed into the FIFO. In this case, the controller does not delay program execution.

## Error Code Command

When there is a program error, the DMC-40x0 halts the program execution at the point where the error occurs. To display the last line number of program execution, issue the command, MG _ED.

The user can obtain information about the type of error condition that occurred by using the command, TC1. This command reports back a number and a text message which describes the error condition. The command, TC0 or TC, will return the error code without the text message. For more information about the command, TC, see the Command Reference.

## Stop Code Command

The status of motion for each axis can be determined by using the stop code command, SC. This can be useful when motion on an axis has stopped unexpectedly. The command SC will return a number representing the motion status. See the command reference for further information.

## RAM Memory Interrogation Commands

For debugging the status of the program memory, array memory, or variable memory, the DMC-40x0 has several useful commands. The command, DM ?, will return the number of array elements currently available. The command, DA ?, will return the number of arrays which can be currently defined. For example, a standard DMC-14010 will have a maximum of 16000 array elements in up to 30 arrays. If an array of 100 elements is defined, the command DM ? will return the value 15900 and the command DA ? will return 29.

To list the contents of the variable space, use the interrogation command LV (List Variables). To list the contents of array space, use the interrogation command, LA (List Arrays). To list the contents of the Program space, use the interrogation command, LS (List). To list the application program labels only, use the interrogation command, LL (List Labels).

### Operands

In general, all operands provide information which may be useful in debugging an application program. Below is a list of operands which are particularly valuable for program debugging. To display the value of an operand, the message command may be used. For example, since the operand, _ED contains the last line of program execution, the command MG _ED will display this line number.

_ED contains the last line of program execution. Useful to determine where program stopped.

_DL contains the number of available labels.

_UL contains the number of available variables.

_DA contains the number of available arrays.

_DM contains the number of available array elements.

_AB contains the state of the Abort Input

_LFx contains the state of the forward limit switch for the 'x' axis

_LRx contains the state of the reverse limit switch for the 'x' axis

### Debugging Example:

The following program has an error. It attempts to specify a relative movement while the X-axis is already in motion. When the program is executed, the controller stops at line 003. The user can then query the controller using the command, TC1. The controller responds with the corresponding explanation:

```
:ED                    Edit Mode
000 #A                 Program Label
001 PR1000             Position Relative 1000
002 BGX                Begin
003 PR5000             Position Relative 5000
004 EN                 End
<cntrl> Q              Quit Edit Mode
:XQ #A                 Execute #A
?003 PR5000            Error on Line 3
:TC1                   Tell Error Code
?7 Command not valid   Command not valid while running
while running.
:ED 3                  Edit Line 3
003 AMX;PR5000;BGX     Add After Motion Done
<cntrl> Q              Quit Edit Mode
:XQ #A                 Execute #A
```

# Program Flow Commands

The DMC-40x0 provides instructions to control program flow. The controller program sequencer normally executes program instructions sequentially. The program flow can be altered with the use of event triggers, trippoints, and conditional jump statements.

## Event Triggers & Trippoints

To function independently from the host computer, the DMC-40x0 can be programmed to make decisions based on the occurrence of an event. Such events include waiting for motion to be complete, waiting for a specified amount of time to elapse, or waiting for an input to change logic levels.

The DMC-40x0 provides several event triggers that cause the program sequencer to halt until the specified event occurs. Normally, a program is automatically executed sequentially one line at a time. When an event trigger instruction is decoded, however, the actual program sequence is halted. The program sequence does not continue until the event trigger is "tripped". For example, the motion complete trigger can be used to separate two move sequences in a program. The commands for the second move sequence will not be executed until the motion is complete on the first motion sequence. In this way, the controller can make decisions based on its own status or external events without intervention from a host computer.

### DMC-40x0 Event Triggers

| Command | Function |
|---|---|
| AM X Y Z W or S<br>(A B C D E F G H) | Halts program execution until motion is complete on the specified axes or motion sequence(s). AM with no parameter tests for motion complete on all axes. This command is useful for separating motion sequences in a program. |
| AD X or Y or Z or W<br>(A or B or C or D or E or F or G or H) | Halts program execution until position command has reached the specified relative distance from the start of the move. Only one axis may be specified at a time. |
| AR X or Y or Z or W<br>(A or B or C or D or E or F or G or H) | Halts program execution until after specified distance from the last AR or AD command has elapsed. Only one axis may be specified at a time. |
| AP X or Y or Z or W<br>(A or B or C or D or E or F or G or H) | Halts program execution until after absolute position occurs. Only one axis may be specified at a time. |
| MF X or Y or Z or W<br>(A or B or C or D or E or F or G or H) | Halt program execution until after forward motion reached absolute position. Only one axis may be specified. If position is already past the point, then MF will trip immediately. Will function on geared axis or aux. inputs. |
| MR X or Y or Z or W<br>(A or B or C or D or E or F or G or H) | Halt program execution until after reverse motion reached absolute position. Only one axis may be specified. If position is already past the point, then MR will trip immediately. Will function on geared axis or aux. inputs. |
| MC X or Y or Z or W<br>(A or B or C or D or E or F or G or H) | Halt program execution until after the motion profile has been completed and the encoder has entered or passed the specified position. TW x,y,z,w sets timeout to declare an error if not in position. If timeout occurs, then the trippoint will clear and the stop code will be set to 99. An application program will jump to label #MCTIME. |
| AI +/- n | Halts program execution until after specified input is at specified logic level. n specifies input line. Positive is high logic level, negative is low level. n=1 through 8 for DMC-4010, 4020, 4030, 4040. n=1 through 16 for DMC-4050, 4060, 4070, 4080<br>Also n= 17-48 |

| | |
|---|---|
| AS X Y Z W S<br>(A B C D E F G H) | Halts program execution until specified axis has reached its slew speed. |
| AT +/-n | Halts program execution until n msec from reference time. AT 0 sets reference. AT n waits n msec from reference. AT -n waits n msec from reference and sets new reference after elapsed time. |
| AV n | Halts program execution until specified distance along a coordinated path has occurred. |
| WT n | Halts program execution until specified time in msec has elapsed. |

# Event Trigger Examples:

## Event Trigger - Multiple Move Sequence

The AM trippoint is used to separate the two PR moves. If AM is not used, the controller returns a ? for the second PR command because a new PR cannot be given until motion is complete.

```
#TWOMOVE            Label
PR 2000             Position Command
BGX                 Begin Motion
AMX                 Wait for Motion Complete
PR 4000             Next Position Move
BGX                 Begin 2nd move
EN                  End program
```

## Event Trigger - Set Output after Distance

Set output bit 1 after a distance of 1000 counts from the start of the move. The accuracy of the trippoint is the speed multiplied by the sample period.

```
#SETBIT             Label
SP 10000            Speed is 10000
PA 20000            Specify Absolute position
BGX                 Begin motion
AD 1000             Wait until 1000 counts
SB1                 Set output bit 1
EN                  End program
```

## Event Trigger - Repetitive Position Trigger

To set the output bit every 10000 counts during a move, the AR trippoint is used as shown in the next example.

```
#TRIP               Label
JG 50000            Specify Jog Speed
BGX;n=0             Begin Motion
#REPEAT             # Repeat Loop
AR 10000            Wait 10000 counts
TPX                 Tell Position
SB1                 Set output 1
WT50                Wait 50 msec
CB1                 Clear output 1
n=n+1               Increment counter
JP #REPEAT,n<5      Repeat 5 times
```

```
STX                 Stop
EN                  End
```

### Event Trigger - Start Motion on Input

This example waits for input 1 to go low and then starts motion. Note: The AI command actually halts execution of the program until the input occurs. If you do not want to halt the program sequences, you can use the Input Interrupt function (II) or use a conditional jump on an input, such as `JP#GO,@IN[1] = 1`.

```
#INPUT              Program Label
AI-1                Wait for input 1 low
PR 10000            Position command
BGX                 Begin motion
EN                  End program
```

### Event Trigger - Set output when At speed

```
#ATSPEED            Program Label
JG 50000            Specify jog speed
AC 10000            Acceleration rate
BGX                 Begin motion
ASX                 Wait for at slew speed 50000
SB1                 Set output 1
 EN                 End program
```

### Event Trigger - Change Speed along Vector Path

The following program changes the feed rate or vector speed at the specified distance along the vector. The vector distance is measured from the start of the move or from the last AV command.

```
#VECTOR             Label
VMXY;VS 5000        Coordinated path
VP 10000,20000      Vector position
VP 20000,30000      Vector position
VE                  End vector
BGS                 Begin sequence
AV 5000             After vector distance
VS 1000             Reduce speed
EN                  End
```

### Event Trigger - Multiple Move with Wait

This example makes multiple relative distance moves by waiting for each to be complete before executing new moves.

```
#MOVES              Label
PR 12000            Distance
SP 20000            Speed
AC 100000           Acceleration
BGX                 Start Motion
AD 10000            Wait a distance of 10,000 counts
SP 5000             New Speed
AMX                 Wait until motion is completed
WT 200              Wait 200 ms
PR -10000           New Position
```

```
SP 30000              New Speed
AC 150000             New Acceleration
BGX                   Start Motion
EN                    End
```

### Define Output Waveform Using AT

The following program causes Output 1 to be high for 10 msec and low for 40 msec. The cycle repeats every 50 msec.

```
#OUTPUT               Program label
AT0                   Initialize time reference
SB1                   Set Output 1
#LOOP                 Loop
AT 10                 After 10 msec from reference,
CB1                   Clear Output 1
AT -40                Wait 40 msec from reference and reset reference
SB1                   Set Output 1
JP #LOOP              Loop
EN
```

## Conditional Jumps

The DMC-40x0 provides Conditional Jump (JP) and Conditional Jump to Subroutine (JS) instructions for branching to a new program location based on a specified condition. The conditional jump determines if a condition is satisfied and then branches to a new location or subroutine. Unlike event triggers, the conditional jump instruction does not halt the program sequence. Conditional jumps are useful for testing events in real-time. They allow the controller to make decisions without a host computer. For example, the DMC-40x0 can decide between two motion profiles based on the state of an input line.

### Command Format - JP and JS

| FORMAT: | DESCRIPTION |
|---|---|
| JS destination, logical condition | Jump to subroutine if logical condition is satisfied |
| JP destination, logical condition | Jump to location if logical condition is satisfied |

The destination is a program line number or label where the program sequencer will jump if the specified condition is satisfied. Note that the line number of the first line of program memory is 0. The comma designates "IF". The logical condition tests two operands with logical operators.

### Logical operators:

| OPERATOR | DESCRIPTION |
|---|---|
| < | less than |
| > | greater than |
| = | equal to |
| <= | less than or equal to |
| >= | greater than or equal to |
| <> | not equal |

## Conditional Statements

The conditional statement is satisfied if it evaluates to any value other than zero. The conditional statement can be any valid DMC-40x0 numeric operand, including variables, array elements, numeric values, functions, keywords, and arithmetic expressions. If no conditional statement is given, the jump will always occur.

**Examples:**

| | |
|---|---|
| Number | `V1=6` |
| Numeric Expression | `V1=V7*6` |
| | `@ABS[V1]>10` |
| Array Element | `V1<Count[2]` |
| Variable | `V1<V2` |
| Internal Variable | `_TPX=0` |
| | `_TVX>500` |
| I/O | `V1>@AN[2]` |
| | `@IN[1]=0` |

## Multiple Conditional Statements

The DMC-40x0 will accept multiple conditions in a single jump statement. The conditional statements are combined in pairs using the operands "&" and "|". The "&" operand between any two conditions, requires that both statements must be true for the combined statement to be true. The "|" operand between any two conditions, requires that only one statement be true for the combined statement to be true.

**Note:** Each condition must be placed in parentheses for proper evaluation by the controller. In addition, the DMC-40x0 executes operations from left to right. See Mathematical and Functional Expressions for more information.

For example, using variables named V1, V2, V3 and V4:

```
JP #TEST, (V1<V2) & (V3<V4)
```

In this example, this statement will cause the program to jump to the label #TEST if V1 is less than V2 and V3 is less than V4. To illustrate this further, consider this same example with an additional condition:

```
JP #TEST, ((V1<V2) & (V3<V4)) | (V5<V6)
```

This statement will cause the program to jump to the label #TEST under two conditions; 1. If V1 is less than V2 and V3 is less than V4. OR 2. If V5 is less than V6.

## Using the JP Command:

If the condition for the JP command is satisfied, the controller branches to the specified label or line number and continues executing commands from this point. If the condition is not satisfied, the controller continues to execute the next commands in sequence.

| Conditional | Meaning |
|---|---|
| `JP #Loop,COUNT<10` | Jump to #Loop if the variable, COUNT, is less than 10 |
| `JS #MOVE2,@IN[1]=1` | Jump to subroutine #MOVE2 if input 1 is logic level high. After the subroutine MOVE2 is executed, the program sequencer returns to the main program location where the subroutine was called. |
| `JP #BLUE,@ABS[V2]>2` | Jump to #BLUE if the absolute value of variable, V2, is greater than 2 |
| `JP #C,V1*V7<=V8*V2` | Jump to #C if the value of V1 times V7 is less than or equal to the value of V8*V2 |
| `JP#A` | Jump to #A |

## Example Using JP command:

Move the X motor to absolute position 1000 counts and back to zero ten times. Wait 100 msec between moves.

```
#BEGIN              Begin Program
```

```
COUNT=10            Initialize loop counter
#LOOP               Begin loop
PA 1000             Position absolute 1000
BGX                 Begin move
AMX                 Wait for motion complete
WT 100              Wait 100 msec
PA 0                Position absolute 0
BGX                 Begin move
AMX                 Wait for motion complete
WT 100              Wait 100 msec
COUNT=COUNT-1       Decrement loop counter
JP #LOOP,COUNT>0    Test for 10 times thru loop
EN                  End Program
```

## Using If, Else, and Endif Commands

The DMC-40x0 provides a structured approach to conditional statements using IF, ELSE and ENDIF commands.

### Using the IF and ENDIF Commands

An IF conditional statement is formed by the combination of an IF and ENDIF command. The IF command has as it's arguments one or more conditional statements. If the conditional statement(s) evaluates true, the command interpreter will continue executing commands which follow the IF command. If the conditional statement evaluates false, the controller will ignore commands until the associated ENDIF command is executed OR an ELSE command occurs in the program (see discussion of ELSE command below).

**Note**: An ENDIF command must always be executed for every IF command that has been executed. It is recommended that the user not include jump commands inside IF conditional statements since this causes re-direction of command execution. In this case, the command interpreter may not execute an ENDIF command.

### Using the ELSE Command

The ELSE command is an optional part of an IF conditional statement and allows for the execution of command only when the argument of the IF command evaluates False. The ELSE command must occur after an IF command and has no arguments. If the argument of the IF command evaluates false, the controller will skip commands until the ELSE command. If the argument for the IF command evaluates true, the controller will execute the commands between the IF and ELSE command.

### Nesting IF Conditional Statements

The DMC-40x0 allows for IF conditional statements to be included within other IF conditional statements. This technique is known as 'nesting' and the DMC-40x0 allows up to 255 IF conditional statements to be nested. This is a very powerful technique allowing the user to specify a variety of different cases for branching.

### Command Format - IF, ELSE and ENDIF

| Format: | Description |
|---|---|
| IF conditional statement(s) | Execute commands proceeding IF command (up to ELSE command) if conditional statement(s) is true, otherwise continue executing at ENDIF command or optional ELSE command. |
| ELSE | Optional command. Allows for commands to be executed when argument of IF command evaluates not true. Can only be used with IF command. |
| ENDIF | Command to end IF conditional statement. Program must have an ENDIF command for every IF command. |

### Example using IF, ELSE and ENDIF:

```
#TEST                                Begin Main Program "TEST"
II,,3                                Enable input interrupts on input 1 and input 2
MG "WAITING FOR INPUT 1, INPUT 2"    Output message
#LOOP                                Label to be used for endless loop
JP #LOOP                             Endless loop
EN                                   End of main program
#ININT                               Input Interrupt Subroutine
IF (@IN[1]=0)                        IF conditional statement based on input 1
IF (@IN[2]=0)                        2nd IF conditional statement executed if 1st IF conditional true
MG "INPUT 1 AND INPUT 2 ARE ACTIVE"  Message to be executed if 2nd IF conditional is true
ELSE                                 ELSE command for 2nd IF conditional statement
MG "ONLY INPUT 1 IS ACTIVE           Message to be executed if 2nd IF conditional is false
ENDIF                                End of 2nd conditional statement
ELSE                                 ELSE command for 1st IF conditional statement
MG"ONLY INPUT 2 IS ACTIVE"           Message to be executed if 1st IF conditional statement is false
ENDIF                                End of 1st conditional statement
#WAIT                                Label to be used for a loop
JP#WAIT,(@IN[1]=0) | (@IN[2]=0)      Loop until both input 1 and input 2 are not active
RI0                                  End Input Interrupt Routine without restoring trippoints
```

## Subroutines

A subroutine is a group of instructions beginning with a label and ending with an end command (EN). Subroutines are called from the main program with the jump subroutine instruction JS, followed by a label or line number, and conditional statement. Up to 8 subroutines can be nested. After the subroutine is executed, the program sequencer returns to the program location where the subroutine was called unless the subroutine stack is manipulated as described in the following section.

**Example:**

An example of a subroutine to draw a square 500 counts per side is given below. The square is drawn at vector position 1000,1000.

```
#M                    Begin Main Program
CB1                   Clear Output Bit 1 (pick up pen)
VP 1000,1000;LE;BGS   Define vector position; move pen
AMS                   Wait for after motion trippoint
SB1                   Set Output Bit 1 (put down pen)
JS #Square;CB1        Jump to square subroutine
EN                    End Main Program
#Square               Square subroutine
V1=500;JS #L          Define length of side
V1=-V1;JS #L          Switch direction
EN                    End subroutine
#L;PR V1,V1;BGX       Define X,Y; Begin X
AMX;BGY;AMY           After motion on X, Begin Y
EN                    End subroutine
```

## Stack Manipulation

It is possible to manipulate the subroutine stack by using the ZS command. Every time a JS instruction, interrupt or automatic routine (such as #POSERR or #LIMSWI) is executed, the subroutine stack is incremented by 1. Normally the stack is restored with an EN instruction. Occasionally it is desirable not to return back to the program line where the subroutine or interrupt was called. The ZS1 command clears 1 level of the stack. This allows the program sequencer to continue to the next line. The ZS0 command resets the stack to its initial value. For example, if a limit occurs and the #LIMSWI routine is executed, it is often desirable to restart the program sequence instead of returning to the location where the limit occurred. To do this, give a ZS command at the end of the #LIMSWI routine.

## Auto-Start Routine

The DMC-40x0 has a special label for automatic program execution. A program which has been saved into the controller's non-volatile memory can be automatically executed upon power up or reset by beginning the program with the label #AUTO. The program must be saved into non-volatile memory using the command, BP.

## Automatic Subroutines for Monitoring Conditions

Often it is desirable to monitor certain conditions continuously without tying up the host or DMC-40x0 program sequences. The controller can monitor several important conditions in the background. These conditions include checking for the occurrence of a limit switch, a defined input, position error, or a command error. Automatic monitoring is enabled by inserting a special, predefined label in the applications program. The pre-defined labels are:

| SUBROUTINE | DESCRIPTION |
|---|---|
| #LIMSWI | Limit switch on any axis goes low |
| #ININT | Input specified by II goes low |
| #POSERR | Position error exceeds limit specified by ER |
| #MCTIME | Motion Complete timeout occurred. Timeout period set by TW command |
| #CMDERR | Bad command given |
| #AUTO | Automatically executes on power up |
| #AUTOERR | Automatically executes when a checksum is encountered during #AUTO start-up. Check error condition with _RS.<br>bit 0 for variable checksum error<br>bit 1 for parameter checksum error<br>bit 2 for program checksum error<br>bit 3 for master reset error (there should be no program ) |

For example, the #POSERR subroutine will automatically be executed when any axis exceeds its position error limit. The commands in the #POSERR subroutine could decode which axis is in error and take the appropriate action. In another example, the #ININT label could be used to designate an input interrupt subroutine. When the specified input occurs, the program will be executed automatically.

**NOTE**: An application program must be running for #CMDERR to function.

### Example - Limit Switch:

This program prints a message upon the occurrence of a limit switch. Note, for the #LIMSWI routine to function, the DMC-40x0 must be executing an applications program from memory. This can be a very simple program that does nothing but loop on a statement, such as #LOOP;JP #LOOP;EN. Motion commands, such as JG 5000 can still be sent from the PC even while the "dummy" applications program is being executed.

```
:ED                         Edit Mode
```

```
000 #LOOP                    Dummy Program
001 JP #LOOP;EN              Jump to Loop
002 #LIMSWI                  Limit Switch Label
003 MG "LIMIT OCCURRED"      Print Message
004 RE                       Return to main program
<control> Q                  Quit Edit Mode
:XQ #LOOP                    Execute Dummy Program
:JG 5000                     Jog
:BGX                         Begin Motion
```

Now, when a forward limit switch occurs on the X axis, the #LIMSWI subroutine will be executed.

Notes regarding the #LIMSWI Routine:

1) The RE command is used to return from the #LIMSWI subroutine.
2) The #LIMSWI subroutine will be re-executed if the limit switch remains active.

The #LIMSWI routine is only executed when the motor is being commanded to move.

### Example - Position Error

```
:ED                                Edit Mode
000 #LOOP                          Dummy Program
001 JP #LOOP;EN                    Loop
002 #POSERR                        Position Error Routine
003 V1=_TEX                        Read Position Error
004 MG "EXCESS POSITION ERROR"     Print Message
005 MG "ERROR=",V1=                Print Error
006 RE                             Return from Error
<control> Q                        Quit Edit Mode
:XQ #LOOP                          Execute Dummy Program
:JG 100000                         Jog at High Speed
:BGX                               Begin Motion
```

### Example - Input Interrupt

```
#A                          Label
II1                         Input Interrupt on 1
JG 30000,,,60000            Jog
BGXW                        Begin Motion
#LOOP;JP#LOOP;EN            Loop
#ININT                      Input Interrupt
STXW;AM                     Stop Motion
#TEST;JP #TEST, @IN[1]=0    Test for Input 1 still low
JG 30000,,,6000             Restore Velocities
BGXW                        Begin motion
RI0                         Return from interrupt routine to Main Program and do not re-
                            enable trippoints
```

### Example - Motion Complete Timeout

```
#BEGIN              Begin main program
TW 1000             Set the time out to 1000 ms
PA 10000            Position Absolute command
```

```
BGX                     Begin motion
MCX                     Motion Complete trip point
EN                      End main program
#MCTIME                 Motion Complete Subroutine
MG "X fell short"       Send out a message
EN                      End subroutine
```

This simple program will issue the message "X fell short" if the X axis does not reach the commanded position within 1 second of the end of the profiled move.

### Example - Command Error

```
#BEGIN                          Begin main program
IN "ENTER SPEED", SPEED         Prompt for speed
JG SPEED;BGX;                   Begin motion
JP #BEGIN                       Repeat
EN                              End main program
#CMDERR                         Command error utility
JP#DONE,_ED<>2                  Check if error on line 2
JP#DONE,_TC<>6                  Check if out of range
MG "SPEED TOO HIGH"             Send message
MG "TRY AGAIN"                  Send message
ZS1                             Adjust stack
JP #BEGIN                       Return to main program
#DONE                           End program if other error
ZS0                             Zero stack
EN                              End program
```

The above program prompts the operator to enter a jog speed. If the operator enters a number out of range (greater than 8 million), the #CMDERR routine will be executed prompting the operator to enter a new number.

In multitasking applications, there is an alternate method for handling command errors from different threads. Using the XQ command along with the special operands described below allows the controller to either skip or retry invalid commands.

| OPERAND | FUNCTION |
|---|---|
| _ED1 | Returns the number of the thread that generated an error |
| _ED2 | Retry failed command (operand contains the location of the failed command) |
| _ED3 | Skip failed command (operand contains the location of the command after the failed command) |

The operands are used with the XQ command in the following format:

```
XQ _ED2 (or _ED3),_ED1,1
```

Where the ",1" at the end of the command line indicates a restart; therefore, the existing program stack will not be removed when the above format executes.

The following example shows an error correction routine which uses the operands.

### Example - Command Error w/Multitasking

```
#A                              Begin thread 0 (continuous loop)
JP#A
EN                              End of thread 0
```

```
#B                          Begin thread 1
N=-1                        Create new variable
KP N                        Set KP to value of N, an invalid value
TY                          Issue invalid command
EN                          End of thread 1

#CMDERR                     Begin command error subroutine
IF _TC=6                    If error is out of range (KP -1)
N=1                         Set N to a valid number
XQ _ED2,_ED1,1              Retry KP N command
ENDIF
IF _TC=1                    If error is invalid command (TY)
XQ _ED3,_ED1,1              Skip invalid command
ENDIF
EN                          End of command error routine
```

### Example - Communication Interrupt

A DMC-4010 is used to move the A axis back and forth from 0 to 10000. This motion can be paused, resumed and stopped via input from an auxiliary port terminal.

| Instruction | Interpretation |
|---|---|
| #BEGIN | Label for beginning of program |
| CC 9600,0,1,0 | Setup communication configuration for auxiliary serial port |
| CI 2 | Setup communication interrupt for auxiliary serial port |
| MG {P2}"Type 0 to stop motion" | Message out of auxiliary port |
| MG {P2}"Type 1 to pause motion" | Message out of auxiliary port |
| MG {P2}"Type 2 to resume motion" | Message out of auxiliary port |
| rate=2000 | Variable to remember speed |
| SPA=rate | Set speed of A axis motion |
| #LOOP | Label for Loop |
| PAA=10000 | Move to absolute position 10000 |
| BGA | Begin Motion on A axis |
| AMA | Wait for motion to be complete |
| PAA=0 | Move to absolute position 0 |
| BGA | Begin Motion on A axis |
| AMA | Wait for motion to be complete |
| JP #LOOP | Continually loop to make back and forth motion |
| EN | End main program |
| #COMINT | Interrupt Routine |
| JP #STOP,P2CH="0" | Check for S (stop motion) |
| JP #PAUSE,P2CH="1" | Check for P (pause motion) |
| JP #RESUME,P2CH="2" | Check for R (resume motion) |
| EN1,1 | Do nothing |
| #STOP | Routine for stopping motion |
| STA;ZS;EN | Stop motion on A axis; Zero program stack; End Program |
| #PAUSE | Routine for pausing motion |
| rate=_SPA | Save current speed setting of A axis motion |
| SPA=0 | Set speed of A axis to zero (allows for pause) |
| EN1,1 | Re-enable trip-point and communication interrupt |

```
#RESUME                         Routine for resuming motion
SPA=rate                        Set speed on A axis to original speed
EN1,1                           Re-enable trip-point and communication interrupt
```

For additional information, see section on Using Communication Interrupt.

### Example – Ethernet Communication Error

This simple program executes in the DMC-40x0 and indicates (via the serial port) when a communication handle fails. By monitoring the serial port, the user can re-establish communication if needed.

| Instruction | Interpretation |
|---|---|
| #LOOP | Simple program loop |
| JP#LOOP | |
| EN | |
| #TCPERR | Ethernet communication error auto routine |
| MG {P1}_IA4 | Send message to serial port indicating which handle did not receive proper acknowledgment. |
| RE | |

## JS Subroutine Stack Variables (^a, ^b, ^c, ^d, ^e, ^f, ^g, ^h)

There are 8 variables that may be passed on the subroutine stack when using the JS command. Passing values on the stack is advanced DMC programming, and is recommended for experienced DMC programmers familiar with the concept of passing arguments by value and by reference.

**Notes:**

1. Passing parameters has no type checking, so it is important to exercise good programming style when passing parameters. See examples below for recommended syntax.
2. Do not use spaces in expressions containing ^.
3. Global variables MUST be assigned prior to any use in subroutines where variables are passed by reference.

### Example: A Simple Adding Function

```
#Add
JS#SUM(1,2,3,4,5,6,7,8)             ;' call subroutine, pass values
MG_JS                               ;' print return value
EN
'
#SUM                                ;NO(^a,^b,^c,^d,^e,^f,^g,^h) syntax note for use
EN,,(^a+^b+^c+^d+^e+^f+^g+^h)       ;' return sum

:Executed program from program1.dmc
36.0000
```

### Example: Variable, and an Important Note about Creating Global Variables

```
#Var
value=5                             ;'a value to be passed by reference
global=8                            ;'a global variable
JS#SUM(&value,1,2,3,4,5,6,7)        ;'note first arg passed by reference
MG value                            ;'message out value after subroutine.
MG _JS                              ;'message out returned value
EN
```

```
'
#SUM                                ;NO(* ^a,^b,^c,^d,^e,^f,^g)
^a=^b+^c+^d+^e+^f+^g+^h+global
EN,,^a
'notes:
'do not use spaces when working with ^
'If using global variables, they MUST be created before the subroutine is run

Executed program from program2.dmc
36.0000
36.0000
```

## Example: Working with Arrays

```
#Array
DM speeds[8]
DM other[256]
JS#zeroAry("speeds",0)              ;'zero out all buckets in speeds[]
JS#zeroAry("other",0)               ;'zero out all buckers in other[]
EN
'
#zeroAry                            ;NO(array ^a, ^b) zeros array starting at index ^b
^a[^b]=0
^b=^b+1
JP#zeroAry,(^b<^a[-1])              ;'[-1] returns the length of an array
EN
```

## Example: Abstracting Axes

```
#Axes
JS#runMove(0,10000,1000,100000,100000)
MG "Position:",_JS
EN
'
#runMove            ;NO(axis ^a, PR ^b, SP ^c, AC ^d, DC ^e) Profile movement for axis
~a=^a               ;'~a is global, so use carefully in subroutines
                     'try one variable axis a-h for each thread A-H
PR~a=^b
SP~a=^c
AC~a=^d
DC~a=^e
BG~a
MC~a
EN,,_TP~a
```

## Example: Local Scope

```
#Local
JS#POWER(2,2)
MG_JS
JS#POWER(2,16)
MG_JS
JS#POWER(2,-8)
MG_JS
'
#POWER        ;NO(base ^a,exponent^b) Returns base^exponent power. +/- integer only
```

```
^c=1          ;'unpassed variable space (^c-^h) can be used as local scope variables
IF ^b=0       ;'special case, exponent = 0
 EN,,1
ENDIF
IF ^b<0       ;'special case, exponent < 0, invert result
 ^d=1
 ^b=@ABS[^b]
ELSE
 ^d=0
ENDIF
#PWRHLPR
^c=^c*^a
^b=^b-1
JP#PWRHLPR,^b>0
IF ^d=1       ;'if inversion required
 ^c=(1/^c)
ENDIF
EN,,^c

Executed program from program1.dmc
4.0000
65536.0000
0.0039
```

## Example: Recursion

```
'although the stack depth is only 16, Galil DMC code does support recursion
JS#AxsInfo(0)
MG{Z2.0}"Recursed through ",_JS," stacks"
EN
'
#AxsInfo                          ;NO(axis ^a) List info for axes
~h=^a
^b=(^a+$41)*$1000000              ;'convert to Galil String
MG^b{S1}, " Axis: "{N}
MG{F8.0}"Position: ",_TP~h," Error:",_TE~h," Torque:",_TT~h{F1.4}
IF ^a=7                           ;'recursion exit condition
EN,,1
ENDIF
JS#AxsInfo(^a + 1)                ;'stack up recursion
EN,,_JS+1                         ;' as recursion closes, add up stack depths

Executed program from program1.dmc
A Axis: Position: 00029319 Error: 00001312 Torque: 9.9982
B Axis: Position: -00001612 Error: 00000936 Torque: 1.7253
C Axis: Position: 00001696 Error:-00001076 Torque:-1.9834
D Axis: Position: -00002020 Error: 00001156 Torque: 2.1309
E Axis: Position: 00000700 Error:-00001300 Torque:-2.3963
F Axis: Position: 00000156 Error:-00000792 Torque:-1.4599
G Axis: Position: -00002212 Error: 00001732 Torque: 3.1926
H Axis: Position: 00002665 Error:-00001721 Torque:-3.1723
Recursed through 8 stacks
```

# Mathematical and Functional Expressions

## Mathematical Operators

For manipulation of data, the DMC-40x0 provides the use of the following mathematical operators:

| Operator | Function |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | Modulus |
| & | Logical And (Bit-wise) |
| \| | Logical Or (On some computers, a solid vertical line appears as a broken line) |
| () | Parenthesis |

The numeric range for addition, subtraction and multiplication operations is +/-2,147,483,647.9999. The precision for division is 1/65,000.

Mathematical operations are executed from left to right. Calculations within parentheses have precedence.

**Examples:**

```
speed = 7.5*V1/2            The variable, speed, is equal to 7.5 multiplied by V1 and divided by 2
count = count+2             The variable, count, is equal to the current value plus 2.
result =_TPX-(@COS[45]*40)  Puts the position of X - 28.28 in result.  40 * cosine of 45° is 28.28
temp = @IN[1]&@IN[2]        temp is equal to 1 only if Input 1 and Input 2 are high
```

## Bit-Wise Operators

The mathematical operators & and | are bit-wise operators. The operator, &, is a Logical And. The operator, |, is a Logical Or. These operators allow for bit-wise operations on any valid DMC-40x0 numeric operand, including variables, array elements, numeric values, functions, keywords, and arithmetic expressions. The bit-wise operators may also be used with strings. This is useful for separating characters from an input string. When using the input command for string input, the input variable will hold up to 6 characters. These characters are combined into a single value which is represented as 32 bits of integer and 16 bits of fraction. Each ASCII character is represented as one byte (8 bits), therefore the input variable can hold up to six characters. The first character of the string will be placed in the top byte of the variable and the last character will be placed in the lowest significant byte of the fraction. The characters can be individually separated by using bit-wise operations as illustrated in the following example:

```
#TEST                          Begin main program
IN "ENTER",len{S6}             Input character string of up to 6 characters into variable 'len'
Flen=@FRAC[len]                Define variable 'Flen' as fractional part of variable 'len'
Flen=$10000*Flen               Shift Flen by 32 bits (IE - convert fraction, Flen, to integer)
len1=(Flen&$00FF)              Mask top byte of Flen and set this value to variable 'len1'
len2=(Flen&$FF00)/$100         Let variable, 'len2' = top byte of Flen
len3=len&$000000FF             Let variable, 'len3' = bottom byte of len
len4=(len&$0000FF00)/$100      Let variable, 'len4' = second byte of len
len5=(len&$00FF0000)/$10000    Let variable, 'len5' = third byte of len
len6=(len&$FF000000)/$1000000  Let variable, 'len6' = fourth byte of len
MG len6 {S4}                   Display 'len6' as string message of up to 4 chars
```

```
MG len5 {S4}                    Display 'len5' as string message of up to 4 chars
MG len4 {S4}                    Display 'len4' as string message of up to 4 chars
MG len3 {S4}                    Display 'len3' as string message of up to 4 chars
MG len2 {S4}                    Display 'len2' as string message of up to 4 chars
MG len1 {S4}                    Display 'len1' as string message of up to 4 chars
EN
```

This program will accept a string input of up to 6 characters, parse each character, and then display each character. Notice also that the values used for masking are represented in hexadecimal (as denoted by the preceding '$'). For more information, see section Sending Messages.

To illustrate further, if the user types in the string "TESTME" at the input prompt, the controller will respond with the following:

```
T                               Response from command MG len6 {S4}
E                               Response from command MG len5 {S4}
S                               Response from command MG len4 {S4}
T                               Response from command MG len3 {S4}
M                               Response from command MG len2 {S4}
E                               Response from command MG len1 {S4}
```

## Functions

| FUNCTION | DESCRIPTION |
|---|---|
| @SIN[n] | Sine of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution) |
| @COS[n] | Cosine of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution) |
| @TAN[n] | Tangent of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution) |
| @ASIN*[n] | Arc Sine of n, between -90° and +90°. Angle resolution in 1/64000 degrees. |
| @ACOS*[n] | Arc Cosine of n, between 0 and 180°. Angle resolution in 1/64000 degrees. |
| @ATAN*[n] | Arc Tangent of n, between -90° and +90°. Angle resolution in 1/64000 degrees |
| @COM[n] | 1's Complement of n |
| @ABS[n] | Absolute value of n |
| @FRAC[n] | Fraction portion of n |
| @INT[n] | Integer portion of n |
| @RND[n] | Round of n (Rounds up if the fractional part of n is .5 or greater) |
| @SQR[n] | Square root of n (Accuracy is +/-.004) |
| @IN[n] | Return digital input at general input n (where n starts at 1) |
| @OUT[n] | Return digital output at general output n (where n starts at 1) |
| @AN[n] | Return analog input at general analog in n (where n starts at 1) |

*Note that these functions are multi-valued. An application program may be used to find the correct band.

Functions may be combined with mathematical expressions. The order of execution of mathematical expressions is from left to right and can be over-ridden by using parentheses.

**Examples:**

```
v1=@ABS[V7]       The variable, v1, is equal to the absolute value of variable v7.
v2=5*@SIN[pos]    The variable, v2, is equal to five times the sine of the variable, pos.
v3=@IN[1]         The variable, v3, is equal to the digital value of input 1.
v4=2*(5+@AN[5])   The variable, v4, is equal to the value of analog input 5 plus 5, then
                  multiplied by 2.
```

# Variables

For applications that require a parameter that is variable, the DMC-40x0 provides 510 variables. These variables can be numbers or strings. A program can be written in which certain parameters, such as position or speed, are defined as variables. The variables can later be assigned by the operator or determined by program calculations. For example, a cut-to-length application may require that a cut length be variable.

**Example:**

```
posx=5000            Assigns the value of 5000 to the variable posx
PR posx              Assigns variable posx to PR command
JG rpmY*70           Assigns variable rpmY multiplied by 70 to JG command.
```

## Programmable Variables

The DMC-40x0 allows the user to create up to 510 variables. Each variable is defined by a name which can be up to eight characters. The name must start with an alphabetic character; however, numbers are permitted in the rest of the name. Spaces are not permitted. Variable names should not be the same as DMC-40x0 instructions. For example, PR is not a good choice for a variable name.

**Note: It is generally a good idea to use lower-case variable names so there is no confusion between Galil commands and variable names.**

Examples of valid and invalid variable names are:

### Valid Variable Names

```
posx
pos1
speedZ
```

### Invalid Variable Names

```
RealLongName   ; 'Cannot have more than 8 characters
123            ; 'Cannot begin variable name with a number
speed Z        ; 'Cannot have spaces in the name
```

### Assigning Values to Variables:

Assigned values can be numbers, internal variables and keywords, functions, controller parameters and strings. The range for numeric variable values is 4 bytes of integer (231) followed by two bytes of fraction (+/- 2,147,483,647.9999).

Numeric values can be assigned to programmable variables using the equal sign.

Any valid DMC-40x0 function can be used to assign a value to a variable. For example, v1=@ABS[v2] or v2=@IN[1]. Arithmetic operations are also permitted.

To assign a string value, the string must be in quotations. String variables can contain up to six characters which must be in quotation.

**Examples:**

```
posX=_TPX            Assigns returned value from TPX command to variable posx.
speed=5.75           Assigns value 5.75 to variable speed
input=@IN[2]         Assigns logical value of input 2 to variable input
v2=v1+v3*v4          Assigns the value of v1 plus v3 times v4 to the variable v2.
var="CAT"            Assign the string, CAT, to var
MG var{S3}           Displays the variable var – (CAT)
```

### Assigning Variable Values to Controller Parameters

Variable values may be assigned to controller parameters such as GN or PR.

```
PR v1                Assign v1 to PR command
SP vS*2000           Assign vS*2000 to SP command
```

### Displaying the value of variables at the terminal

Variables may be sent to the screen using the format, variable=. For example, v1= , returns the value of the variable v1.

### Example - Using Variables for Joystick

The example below reads the voltage of an X-Y joystick and assigns it to variables vX and vY to drive the motors at proportional velocities, where:

10 Volts = 3000 rpm = 200000 c/sec

Speed/Analog input = 200000/10 = 20000

```
#JOYSTIK             Label
JG 0,0               Set in Jog mode
BGXY                 Begin Motion
#LOOP                Loop
vX=@AN[1]*20000      Read joystick X
vY=@AN[2]*20000      Read joystick Y
JG vX,vY             Jog at variable vX,vY
JP#LOOP              Repeat
EN                   End
```

# Operands

Operands allow motion or status parameters of the DMC-40x0 to be incorporated into programmable variables and expressions. Most DMC commands have an equivalent operand - which are designated by adding an underscore (_) prior to the DMC-40x0 command. The command reference indicates which commands have an associated operand.

Status commands such as Tell Position return actual values, whereas action commands such as KP or SP return the values in the DMC-40x0 registers. The axis designation is required following the command.

### Examples of Internal Variables:

```
POSX=_TPX            Assigns value from Tell Position X to the variable POSX.
GAIN=_GNZ*2          Assigns value from GNZ multiplied by two to variable, GAIN.
JP #LOOP,_TEX>5      Jump to #LOOP if the position error of X is greater than 5
JP #ERROR,_TC=1      Jump to #ERROR if the error code equals 1.
```

Operands can be used in an expression and assigned to a programmable variable, but they cannot be assigned a value. For example: _GNX=2 is invalid.

## Special Operands (Keywords)

The DMC-40x0 provides a few additional operands which give access to internal variables that are not accessible by standard DMC-40x0 commands.

| Keyword | Function |
|---|---|

| | |
|---|---|
| _BGn | *Returns a 1 if motion on axis 'n' is complete, otherwise returns 0. |
| _BN | *Returns serial # of the board. |
| _DA | *Returns the number of arrays available |
| _DL | *Returns the number of available labels for programming |
| _DM | *Returns the available array memory |
| _HMn | *Returns status of Home Switch (equals 0 or 1) |
| _LFn | Returns status of Forward Limit switch input of axis 'n' (equals 0 or 1) |
| _LRX | Returns status of Reverse Limit switch input of axis 'n' (equals 0 or 1) |
| _UL | *Returns the number of available variables |
| TIME | Free-Running Real Time Clock (off by 2.4% - Resets with power-on).<br>Note: TIME does not use an underscore character (_) as other keywords. |

* - These keywords have corresponding commands while the keywords _LF, _LR, and TIME do not have any associated commands. All keywords are listed in the Command Reference.

### Examples of Keywords:

```
V1=_LFX      Assign V1 the logical state of the Forward Limit Switch on the X-axis
V3=TIME      Assign V3 the current value of the time clock
 V4=_HMW     Assign V4 the logical state of the Home input on the W-axis
```

# Arrays

For storing and collecting numerical data, the DMC-40x0 provides array space for 16000 elements. The arrays are one dimensional and up to 30 different arrays may be defined. Each array element has a numeric range of 4 bytes of integer (231)followed by two bytes of fraction (+/-2,147,483,647.9999).

Arrays can be used to capture real-time data, such as position, torque and analog input values. In the contouring mode, arrays are convenient for holding the points of a position trajectory in a record and playback application.

## Defining Arrays

An array is defined with the command DM. The user must specify a name and the number of entries to be held in the array. An array name can contain up to eight characters, starting with an uppercase alphabetic character. The number of entries in the defined array is enclosed in [ ].

**Example:**

```
DM POSX[7]           Defines an array names POSX with seven entries
DM SPEED[100]        Defines an array named speed with 100 entries
DM POSX[0]           Frees array space
```

## Assignment of Array Entries

Like variables, each array element can be assigned a value. Assigned values can be numbers or returned values from instructions, functions and keywords.

Array elements are addressed starting at count 0. For example the first element in the POSX array (defined with the DM command, DM POSX[7]) would be specified as POSX[0].

Values are assigned to array entries using the equal sign. Assignments are made one element at a time by specifying the element number with the associated array name.

**NOTE:** Arrays must be defined using the command, DM, before assigning entry values.

**Examples:**

```
DM SPEED[10]          Dimension Speed Array
SPEED[1]=7650.2       Assigns the first element of the array, SPEED the value 7650.2
SPEED[1]=             Returns array element value
POSX[10]=_TPX         Assigns the 10th element of the array POSX the returned value
                      from the tell position command.
CON[2]=@COS[POS]*2    Assigns the second element of the array CON the cosine of the
                      variable POS multiplied by 2.
TIMER[1]=TIME         Assigns the first element of the array timer the returned
                      value of the TIME keyword.
```

### Using a Variable to Address Array Elements

An array element number can also be a variable. This allows array entries to be assigned sequentially using a counter.

**Example:**

```
#A                    Begin Program
COUNT=0;DM POS[10]    Initialize counter and define array
#LOOP                 Begin loop
WT 10                 Wait 10 msec
POS[COUNT]=_TPX       Record position into array element
POS[COUNT]=           Report position
COUNT=COUNT+1         Increment counter
JP #LOOP,COUNT<10     Loop until 10 elements have been stored
EN                    End Program
```

The above example records 10 position values at a rate of one value per 10 msec. The values are stored in an array named POS. The variable, COUNT, is used to increment the array element counter. The above example can also be executed with the automatic data capture feature described below.

### Uploading and Downloading Arrays to On Board Memory

Arrays may be uploaded and downloaded using the QU and QD commands.

QU array[],start,end,delim

QD array[],start,end

where array is an array name such as A[].

start is the first element of array (default=0)

end is the last element of array (default=last element)

delim specifies whether the array data is separated by a comma (delim=1) or a carriage return (delim=0).

The file is terminated using <control>Z, <control>Q, <control>D or \.

## Automatic Data Capture into Arrays

The DMC-40x0 provides a special feature for automatic capture of data such as position, position error, inputs or torque. This is useful for teaching motion trajectories or observing system performance. Up to eight types of data can be captured and stored in eight arrays. The capture rate or time interval may be specified. Recording can done as a one time event or as a circular continuous recording.

### Command Summary - Automatic Data Capture

| Command | Description |
|---|---|

| | |
|---|---|
| RA n[ ],m[ ],o[ ],p[ ] | Selects up to eight arrays for data capture. The arrays must be defined with the DM command. |
| RD type1,type2,type3,type4 | Selects the type of data to be recorded, where type1, type2, type3, and type 4 represent the various types of data (see table below). The order of data type is important and corresponds with the order of n,m,o,p arrays in the RA command. |
| RC n,m | The RC command begins data collection. Sets data capture time interval where n is an integer between 1 and 8 and designates $2^n$ msec between data. m is optional and specifies the number of elements to be captured. If m is not defined, the number of elements defaults to the smallest array defined by DM. When m is a negative number, the recording is done continuously in a circular manner. _RD is the recording pointer and indicates the address of the next array element. n=0 stops recording. |
| RC? | Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress |

## Data Types for Recording:

| Data type | Description |
|---|---|
| TIME | Controller time as reported by the TIME command |
| _AFn | Analog input (n=X,Y,Z,W,E,F,G,H, for AN inputs 1-8) |
| _DEX | $2^{nd}$ encoder position (dual encoder) |
| _NOX | Status bits |
| _OP | Output |
| _RLX | Latched position |
| _RPX | Commanded position |
| _SCX | Stop code |
| _TEX | Position error |
| _TI | Inputs |
| _TPX | Encoder position |
| _TSX | Switches (only bit 0-4 valid) |
| _TTX | Torque (reports digital value +/-32544) |

**Note**: X may be replaced by Y,Z or W for capturing data on other axes.

## Operand Summary - Automatic Data Capture

| | |
|---|---|
| _RC | Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress |
| _RD | Returns address of next array element. |

## Example - Recording into An Array

During a position move, store the X and Y positions and position error every 2 msec.

```
#RECORD                          Begin program
DM XPOS[300],YPOS[300]           Define X,Y position arrays
DM XERR[300],YERR[300]           Define X,Y error arrays
RA XPOS[],XERR[],YPOS[],YERR[]   Select arrays for capture
RD _TPX,_TEX,_TPY,_TEY           Select data types
PR 10000,20000                   Specify move distance
RC1                              Start recording now, at rate of 2 msec
BG XY                            Begin motion
#A;JP #A,_RC=1                   Loop until done
MG "DONE"                        Print message
```

```
EN                              End program
#PLAY                           Play back
N=0                             Initial Counter
JP# DONE,N>300                  Exit if done
N=                              Print Counter
X POS[N]=                       Print X position
Y POS[N]=                       Print Y position
XERR[N]=                        Print X error
YERR[N]=                        Print Y error
N=N+1                           Increment Counter
#DONE                           Done
EN                              End Program
```

### De-allocating Array Space

Array space may be de-allocated using the DA command followed by the array name. DA*[0] deallocates all the arrays.

# Input of Data (Numeric and String)

## Input of Data

The command, IN, is used to prompt the user to input numeric or string data. Using the IN command, the user may specify a message prompt by placing a message in quotations. When the controller executes an IN command, the controller will wait for the input of data. The input data is assigned to the specified variable or array element.

**An Example for Inputting Numeric Data**

```
#A
IN “Enter Length”, lenX
EN
```

In this example, the message “Enter Length” is displayed on the computer screen. The controller waits for the operator to enter a value. The operator enters the numeric value which is assigned to the variable, lenX.

### Cut-to-Length Example

In this example, a length of material is to be advanced a specified distance. When the motion is complete, a cutting head is activated to cut the material. The length is variable, and the operator is prompted to input it in inches. Motion starts with a start button which is connected to input 1.

The load is coupled with a 2 pitch lead screw. A 2000 count/rev encoder is on the motor, resulting in a resolution of 4000 counts/inch. The program below uses the variable len, to length. The IN command is used to prompt the operator to enter the length, and the entered value is assigned to the variable len.

```
#BEGIN                      LABEL
AC 800000                   Acceleration
DC 800000                   Deceleration
SP 5000                     Speed
len=3.4                     Initial length in inches
#CUT                        Cut routine
AI1                         Wait for start signal
IN “enter Length(IN)”, len  Prompt operator for length in inches
PR len *4000                Specify position in counts
```

```
BGX                     Begin motion to move material
AMX                     Wait for motion done
SB1                     Set output to cut
WT100;CB1               Wait 100 msec, then turn off cutter
JP #CUT                 Repeat process
EN                      End program
```

## Operator Data Entry Mode

The Operator Data Entry Mode provides for un-buffered data entry through the auxiliary RS-232 port. In this mode, the DMC-40x0 provides a buffer for receiving characters. This mode may only be used when executing an applications program.

The Operator Data Entry Mode may be specified for Port 2 only. This mode may be exited with the \ or <escape> key.

**NOTE**: Operator Data Entry Mode cannot be used for high rate data transfer.

Set the third field of the CC command to one to set the Operator Data Entry Mode.

To capture and decode characters in the Operator Data Mode, the DMC-40x0 provides special the following keywords:

| Keyword | Function |
|---|---|
| P2CH | Contains the last character received |
| P2ST | Contains the received string |
| P2NM | Contains the received number |
| P2CD | Contains the status code:<br>-1 mode disabled<br>0 nothing received<br>1 received character, but not <enter><br>2 received string, not a number<br>3 received number |

**NOTE**: The value of P2CD returns to zero after the corresponding string or number is read.

These keywords may be used in an applications program to decode data and they may also be used in conditional statements with logical operators.

**Example**

| Instruction | Interpretation |
|---|---|
| `JP #LOOP,P2CD< >3` | Checks to see if status code is 3 (number received) |
| `JP #P,P1CH="V"` | Checks if last character received was a V |
| `PR P2NM` | Assigns received number to position |
| `JS #XAXIS,P1ST="X"` | Checks to see if received string is X |

## Using Communication Interrupt

The DMC-40x0 provides a special interrupt for communication allowing the application program to be interrupted by input from the user. The interrupt is enabled using the CI command. The syntax for the command is CI n:

| | |
|---|---|
| n = 0 | Don't interrupt Port 2 |
| n = 1 | Interrupt on <enter> Port 2 |
| n = 2 | Interrupt on any character Port 2 |

n = -1 Clear any characters in buffer

The #COMINT label is used for the communication interrupt. For example, the DMC-40x0 can be configured to interrupt on any character received on Port 2. The #COMINT subroutine is entered when a character is received and the subroutine can decode the characters. At the end of the routine the EN command is used. EN,1 will re-enable the interrupt and return to the line of the program where the interrupt was called, EN will just return to the line of the program where it was called without re-enabling the interrupt. As with any automatic subroutine, a program must be running in thread 0 at all times for it to be enabled.

**Example**

A DMC-40x0 is used to jog the A and B axis. This program automatically begins upon power-up and allows the user to input values from the main serial port terminal. The speed of either axis may be changed during motion by specifying the axis letter followed by the new speed value. An S stops motion on both axes.

| Instruction | Interpretation |
|---|---|
| #AUTO | Label for Auto Execute |
| speedA=10000 | Initial A speed |
| speedB=10000 | Initial B speed |
| CI 2 | Set Port 2 for Character Interrupt |
| JG speedA, speedB | Specify jog mode speed for A and B axis |
| BGXY | Begin motion |
| #PRINT | Routine to print message to terminal |
| MG{P2}"TO CHANGE SPEEDS" | Print message |
| MG{P2}"TYPE A OR B" | |
| MG{P2}"TYPE S TO STOP" | |
| #JOGLOOP | Loop to change Jog speeds |
| JG speedA, speedB | Set new jog speed |
| JP #JOGLOOP | |
| EN | End of main program |
| #COMINT | Interrupt routine |
| JP #A,P2CH="A" | Check for A |
| JP #B,P2CH="B" | Check for B |
| JP #C,P2CH="S" | Check for S |
| ZS1;CI2;JP#JOGLOOP | Jump if not X,Y,S |
| #A;JS#NUM | |
| speedX=val | New X speed |
| ZS1;CI2;JP#PRINT | Jump to Print |
| #B;JS#NUM | |
| speedY=val | New Y speed |
| ZS1;CI2;JP#PRINT | Jump to Print |
| #C;ST;AMX;CI-1 | Stop motion on S |
| MG{^8}, "THE END" | |
| ZS;EN,1 | End-Re-enable interrupt |
| #NUM | Routine for entering new jog speed |
| MG "ENTER",P2CH{S},"AXIS SPEED" {N} | Prompt for value |
| #NUMLOOP; CI-1 | Check for enter |
| #NMLP | Routine to check input from terminal |
| JP #NMLP,P2CD<2 | Jump to error if string |
| JP #ERROR,P2CD=2 | Read value |
| val=P2NM | |
| EN | End subroutine |

```
#ERROR;CI-1                    Error Routine
MG "INVALID-TRY AGAIN"         Error message
JP #NMLP
EN                             End
```

### Inputting String Variables

String variables with up to six characters may be input using the specifier, {Sn} where n represents the number of string characters to be input. If n is not specified, six characters will be accepted. For example, IN "Enter A,B or C", V{S} specifies a string variable to be input.

The DMC-40x0, stores all variables as 6 bytes of information. When a variable is specified as a number, the value of the variable is represented as 4 bytes of integer and 2 bytes of fraction. When a variable is specified as a string, the variable can hold up to 6 characters (each ASCII character is 1 byte). When using the IN command for string input, the first input character will be placed in the top byte of the variable and the last character will be placed in the lowest significant byte of the fraction. The characters can be individually separated by using bit-wise operations, see section Bit-wise Operators.

# Output of Data (Numeric and String)

Numerical and string data can be output from the controller using several methods. The message command, MG, can output string and numerical data. Also, the controller can be commanded to return the values of variables and arrays, as well as other information using the interrogation commands (the interrogation commands are described in chapter 5).

## Sending Messages

Messages may be sent to the bus using the message command, MG. This command sends specified text and numerical or string data from variables or arrays to the screen.

Text strings are specified in quotes and variable or array data is designated by the name of the variable or array. For example:

```
MG "The Final Value is", result
```

In addition to variables, functions and commands, responses can be used in the message command. For example:

```
MG "Analog input is", @AN[1]
MG "The Position of A is", _TPA
```

### Specifying the Port for Messages:

The port can be specified with the specifier, {P1} for the main serial port {P2} for auxiliary serial port, or {En} for the Ethernet port.

```
MG {P2} "Hello World"          Sends message to Auxiliary Port
```

### Formatting Messages

String variables can be formatted using the specifier, {Sn} where n is the number of characters, 1 thru 6. For example:

```
MG STR {S3}
```

This statement returns 3 characters of the string variable named STR.

Numeric data may be formatted using the {Fn.m} expression following the completed MG statement. {$n.m} formats data in HEX instead of decimal. The actual numerical value will be formatted with n characters to the left of the decimal and m characters to the right of the decimal. Leading zeros will be used to display specified format.

For example:

```
MG "The Final Value is", result {F5.2}
```

If the value of the variable result is equal to 4.1, this statement returns the following:

The Final Value is 00004.10

If the value of the variable result is equal to 999999.999, the above message statement returns the following:

The Final Value is 99999.99

The message command normally sends a carriage return and line feed following the statement. The carriage return and the line feed may be suppressed by sending {N} at the end of the statement. This is useful when a text string needs to surround a numeric value.

Example:

```
 #A
JG 50000;BGA;ASA
MG "The Speed is", _TVA {F5.1} {N}
MG "counts/sec"
EN
```

When #A is executed, the above example will appear on the screen as:

The speed is 50000 counts/sec

### Using the MG Command to Configure Terminals

The MG command can be used to configure a terminal. Any ASCII character can be sent by using the format {^n} where n is any integer between 1 and 255.

Example:

```
MG {^07} {^255}
```

sends the ASCII characters represented by 7 and 255 to the bus.

### Summary of Message Functions

| function | description |
|---|---|
| " " | Surrounds text string |
| {Fn.m} | Formats numeric values in decimal n digits to the left of the decimal point and m digits to the right |
| {P1}, {P2} or {E} | Send message to Main Serial Port, Auxiliary Serial Port or Ethernet Port |
| {$n.m} | Formats numeric values in hexadecimal |
| {^n} | Sends ASCII character specified by integer n |
| {N} | Suppresses carriage return/line feed |
| {Sn} | Sends the first n characters of a string variable, where n is 1 thru 6. |

## Displaying Variables and Arrays

Variables and arrays may be sent to the screen using the format, variable= or array[x]=. For example, v1= returns the value of v1.

### Example - Printing a Variable and an Array element

| Instruction | Interpretation |
|---|---|
| #DISPLAY | Label |
| DM posA[7] | Define Array POSA with 7 entries |
| PR 1000 | Position Command |

```
BGX                         Begin
AMX                         After Motion
v1=_TPA                     Assign Variable v1
posA[1]=_TPA                Assign the first entry
v1=                         Print v1
```

## Interrogation Commands

The DMC-40x0 has a set of commands that directly interrogate the controller. When these command are entered, the requested data is returned in decimal format on the next line followed by a carriage return and line feed. The format of the returned data can be changed using the Position Format (PF), and Leading Zeros (LZ) command. For a complete description of interrogation commands, see Chapter 5.

### Using the PF Command to Format Response from Interrogation Commands

The command, PF, can change format of the values returned by theses interrogation commands:

| | |
|---|---|
| BL ? | LE ? |
| DE ? | PA ? |
| DP ? | PR ? |
| EM ? | TN ? |
| FL ? | VE ? |
| IP ? | TE |
| TP | |

The numeric values may be formatted in decimal or hexadecimal with a specified number of digits to the right and left of the decimal point using the PF command.

Position Format is specified by:

PF m.n

where m is the number of digits to the left of the decimal point (0 thru 10) and n is the number of digits to the right of the decimal point (0 thru 4) A negative sign for m specifies hexadecimal format.

Hex values are returned preceded by a $ and in 2's complement. Hex values should be input as signed 2's complement, where negative numbers have a negative sign. The default format is PF 10.0.

If the number of decimal places specified by PF is less than the actual value, a nine appears in all the decimal places.

**Example**

| Instruction | Interpretation |
|---|---|
| :DP21 | Define position |
| :TPA | Tell position |
| 0000000021 | Default format |
| :PF4 | Change format to 4 places |
| :TPA | Tell position |
| 0021 | New format |
| :PF-4 | Change to hexadecimal format |
| :TPA | Tell Position |
| $0015 | Hexadecimal value |
| :PF2 | Format 2 places |
| :TPA | Tell Position |
| 99 | Returns 99 if position greater than 99 |

### Removing Leading Zeros from Response to Interrogation Commands

The leading zeros on data returned as a response to interrogation commands can be removed by the use of the command, LZ.

```
LZ0                                 Disables the LZ function
TP                                  Tell Position Interrogation Command
-0000000009, 0000000005             Response (With Leading Zeros)
LZ1                                 Enables the LZ function
TP                                  Tell Position Interrogation Command
-9, 5                               Response (Without Leading Zeros)
```

### Local Formatting of Response of Interrogation Commands

The response of interrogation commands may be formatted locally. To format locally, use the command, {Fn.m} or {$n.m} on the same line as the interrogation command. The symbol F specifies that the response should be returned in decimal format and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal.

```
TP {F2.2}                           Tell Position in decimal format 2.2
-05.00, 05.00, 00.00, 07.00         Response from Interrogation Command
TP {$4.2}                           Tell Position in hexadecimal format 4.2
FFFB.00,$0005.00,$0000.00,$0007.00  Response from Interrogation Command
```

## Formatting Variables and Array Elements

The Variable Format (VF) command is used to format variables and array elements. The VF command is specified by:

```
VF m.n
```

where m is the number of digits to the left of the decimal point (0 thru 10) and n is the number of digits to the right of the decimal point (0 thru 4).

A negative sign for m specifies hexadecimal format. The default format for VF is VF 10.4

Hex values are returned preceded by a $ and in 2's complement.

| Instruction | Interpretation |
|---|---|
| v1=10 | Assign v1 |
| v1= | Return v1 |
| :0000000010.0000 | Response - Default format |
| VF2.2 | Change format |
| v1= | Return v1 |
| :10.00 | Response - New format |
| VF-2.2 | Specify hex format |
| v1= | Return v1 |
| $0A.00 | Response - Hex value |
| VF1 | Change format |
| v1= | Return v1 |
| :9 | Response - Overflow |

### Local Formatting of Variables

PF and VF commands are global format commands that affect the format of all relevant returned values and variables. Variables may also be formatted locally. To format locally, use the command, {Fn.m} or {$n.m} following the variable name and the '=' symbol. F specifies decimal and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal.

| Instruction | Interpretation |
|---|---|
| v1=10 | Assign v1 |
| v1= | Return v1 |
| :0000000010.0000 | Default Format |
| v1={F4.2} | Specify local format |
| :0010.00 | New format |
| v1={$4.2} | Specify hex format |
| :$000A.00 | Hex value |
| v1="ALPHA" | Assign string "ALPHA" to v1 |
| v1={S4} | Specify string format first 4 characters |
| :ALPH | |

The local format is also used with the MG command.

### Converting to User Units

Variables and arithmetic operations make it easy to input data in desired user units such as inches or RPM.

The DMC-40x0 position parameters such as PR, PA and VP have units of quadrature counts. Speed parameters such as SP, JG and VS have units of counts/sec. Acceleration parameters such as AC, DC, VA and VD have units of counts/sec2. The controller interprets time in milliseconds.

All input parameters must be converted into these units. For example, an operator can be prompted to input a number in revolutions. A program could be used such that the input number is converted into counts by multiplying it by the number of counts/revolution.

| Instruction | Interpretation |
|---|---|
| #RUN | Label |
| IN "ENTER # OF REVOLUTIONS",n1 | Prompt for revs |
| PR n1*2000 | Convert to counts |
| IN "ENTER SPEED IN RPM",s1 | Prompt for RPMs |
| SP s1*2000/60 | Convert to counts/sec |
| IN "ENTER ACCEL IN RAD/SEC2",a1 | Prompt for ACCEL |
| AC a1*2000/(2*3.14) | Convert to counts/sec2 |
| BG | Begin motion |
| EN | End program |

# Hardware I/O

## Digital Outputs

The DMC-40x0 has an 8-bit uncommitted output port and an additional 32 I/O which may be configured as inputs or outputs with the CO command for controlling external events. The DMC-4050 through DMC-4080 has an additional 8 outputs. Each bit on the output port may be set and cleared with the software instructions SB (Set Bit) and CB (Clear Bit), or OB (define output bit).

### Example- Set Bit and Clear Bit

| Instruction | Interpretation |
|---|---|
| SB6 | Sets bit 6 of output port |
| CB4 | Clears bit 4 of output port |

### Example- Output Bit

The Output Bit (OB) instruction is useful for setting or clearing outputs depending on the value of a variable, array, input or expression. Any non-zero value results in a set bit.

| Instruction | Interpretation |
|---|---|
| OB1, POS | Set Output 1 if the variable POS is non-zero. Clear Output 1 if POS equals 0. |
| OB 2, @IN [1] | Set Output 2 if Input 1 is high. If Input 1 is low, clear Output 2. |
| OB 3, @IN [1]&@IN [2] | Set Output 3 only if Input 1 and Input 2 are high. |
| OB 4, COUNT [1] | Set Output 4 if element 1 in the array COUNT is non-zero. |

The output port can be set by specifying an 16-bit word using the instruction OP (Output Port). This instruction allows a single command to define the state of the entire 16-bit output port, where bit 0 is output 1, bit1 is output2 and so on. A 1 designates that the output is on.

### Example- Output Port

| Instruction | Interpretation |
|---|---|
| OP6 | Sets outputs 2 and 3 of output port to high. All other bits are 0. ($2^1 + 2^2 = 6$) |
| OP0 | Clears all bits of output port to zero |
| OP 255 | Sets all bits of output port to one. ($2^2 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7$) |

The output port is useful for setting relays or controlling external switches and events during a motion sequence.

### Example - Turn on output after move

| Instruction | Interpretation |
|---|---|
| #OUTPUT | Label |
| PR 2000 | Position Command |
| BG | Begin |
| AM | After move |
| SB1 | Set Output 1 |
| WT 1000 | Wait 1000 msec |
| CB1 | Clear Output 1 |
| EN | End |

## Digital Inputs

The general digital inputs for are accessed by using the @IN[n] function or the TI command. The @IN[n] function returns the logic level of the specified input, n, where n is a number 1 through 48.

### Example - Using Inputs to control program flow

| Instruction | Interpretation |
|---|---|
| JP #A,@IN[1]=0 | Jump to A if input 1 is low |
| JP #B,@IN[2]=1 | Jump to B if input 2 is high |
| AI 7 | Wait until input 7 is high |
| AI -6 | Wait until input 6 is low |

### Example - Start Motion on Switch

Motor A must turn at 4000 counts/sec when the user flips a panel switch to on. When panel switch is turned to off position, motor A must stop turning.

Solution: Connect panel switch to input 1 of DMC-40x0. High on input 1 means switch is in on position.

| Instruction | Interpretation |
|---|---|
| #S;JG 4000 | Set speed |
| AI 1;BGA | Begin after input 1 goes high |
| AI -1;STA | Stop after input 1 goes low |
| AMA;JP #S | After motion, repeat |
| EN | |

## The Auxiliary Encoder Inputs

The auxiliary encoder inputs can be used for general use. For each axis, the controller has one auxiliary encoder and each auxiliary encoder consists of two inputs, channel A and channel B. The auxiliary encoder inputs are mapped to the inputs 81-96.

Each input from the auxiliary encoder is a differential line receiver and can accept voltage levels between +/- 12 volts. The inputs have been configured to accept TTL level signals. To connect TTL signals, simply connect the signal to the + input and leave the - input disconnected. For other signal levels, the - input should be connected to a voltage that is ½ of the full voltage range (for example, connect the - input to 6 volts if the signal is a 0 - 12 volt logic).

Example:

A DMC-4010 has one auxiliary encoder. This encoder has two inputs (channel A and channel B). Channel A input is mapped to input 81 and Channel B input is mapped to input 82. To use this input for 2 TTL signals, the first signal will be connected to AA+ and the second to AB+. AA- and AB- will be left unconnected. To access this input, use the function @IN[81] and @IN[82].

**NOTE**: The auxiliary encoder inputs are not available for any axis that is configured for stepper motor.

## Input Interrupt Function

The DMC-40x0 provides an input interrupt function which causes the program to automatically execute the instructions following the #ININT label. This function is enabled using the II m,n,o command. The m specifies the beginning input and n specifies the final input in the range. The parameter o is an interrupt mask. If m and n are unused, o contains a number with the mask. For example, II,,5 enables inputs 1 and 3.

A low input on any of the specified inputs will cause automatic execution of the #ININT subroutine. The Return from Interrupt (RI) command is used to return from this subroutine to the place in the program where the interrupt had occurred. If it is desired to return to somewhere else in the program after the execution of the #ININT subroutine, the Zero Stack (ZS) command is used followed by unconditional jump statements.

**Important: Use the RI command (not EN) to return from the #ININT subroutine.**

### Example - Input Interrupt

| Instruction | Interpretation |
|---|---|
| #A | Label #A |
| II 1 | Enable input 1 for interrupt function |
| JG 30000,-20000 | Set speeds on A and B axes |
| BG AB | Begin motion on A and B axes |

```
#B                          Label #B
TP AB                       Report A and B axes positions
WT 1000                     Wait 1000 milliseconds
JP #B                       Jump to #B
EN                          End of program
#ININT                      Interrupt subroutine
MG "Interrupt has occurred" Displays the message
ST AB                       Stops motion on A and B axes
#LOOP;JP #LOOP,@IN[1]=0     Loop until Interrupt cleared
JG 15000,10000              Specify new speeds
WT 300                      Wait 300 milliseconds
BG AB                       Begin motion on A and B axes
RI                          Return from Interrupt subroutine
```

## Analog Inputs

The DMC-40x0 provides eight analog inputs. The value of these inputs in volts may be read using the @AN[n] function where n is the analog input 1 through 8. The resolution of the Analog-to-Digital conversion is 12 bits (16-bit ADC is available as an option). Analog inputs are useful for reading special sensors such as temperature, tension or pressure.

The following examples show programs which cause the motor to follow an analog signal. The first example is a point-to-point move. The second example shows a continuous move.

### Example - Position Follower (Point-to-Point)

Objective - The motor must follow an analog signal. When the analog signal varies by 10V, motor must move 10000 counts.

Method: Read the analog input and command A to move to that point.

```
Instruction          Interpretation
#POINTS              Label
SP 7000              Speed
AC 80000;DC 80000    Acceleration
#LOOP
VP=@AN[1]*1000       Read and analog input, compute position
PA VP                Command position
BGA                  Start motion
AMA                  After completion
JP #LOOP             Repeat
EN                   End
```

### Example - Position Follower (Continuous Move)

Method: Read the analog input, compute the commanded position and the position error. Command the motor to run at a speed in proportions to the position error.

```
Instruction          Interpretation
#CONT                Label
AC 80000;DC 80000    Acceleration rate
JG 0                 Start job mode
BGX                  Start motion
#LOOP
```

```
vp=@AN[1]*1000      Compute desired position
ve=vp-_TPA          Find position error
vel=ve*20           Compute velocity
JG vel              Change velocity
JP #LOOP            Change velocity
EN                  End
```

# Extended I/O of the DMC-40x0 Controller

The DMC-40x0 controller offers 32 extended I/O points which can be configured as inputs or outputs in 8 bit increments through software. The I/O points are accessed through 1 44 pin high density connector.

## Configuring the I/O of the DMC-40x0

The 32 extended I/O points of the DMC-40x0 series controller can be configured in blocks of 8. The extended I/O is denoted as blocks 2-5 or bits 17-48.

The command, CO, is used to configure the extended I/O as inputs or outputs. The CO command has one field:

CO n

where n is a decimal value which represents a binary number. Each bit of the binary number represents one block of extended I/O. When set to 1, the corresponding block is configured as an output.

The least significant bit represents block 2 and the most significant bit represents block 5. The decimal value can be calculated by the following formula. n = n2 + 2*n3 + 4*n4 + 5*n5 where nx represents the block. If the nx value is a one, then the block of 8 I/O points is to be configured as an output. If the nx value is a zero, then the block of 8 I/O points will be configured as an input. For example, if block 4 and 5 is to be configured as an output, CO 12 is issued.

| 8-Bit I/O Block | Block | Binary Representation | Decimal Value for Block |
|---|---|---|---|
| 17-24 | 2 | $2^0$ | 1 |
| 25-32 | 3 | $2^1$ | 2 |
| 33-40 | 4 | $2^2$ | 4 |
| 41-48 | 5 | $2^3$ | 8 |

The simplest method for determining n:

**Step 1.** Determine which 8-bit I/O blocks to be configured as outputs.

**Step 2.** From the table, determine the decimal value for each I/O block to be set as an output.

**Step 3.** Add up all of the values determined in step 2. This is the value to be used for n.

For example, if blocks 2 and 3 are to be outputs, then n is 3 and the command, CO3, should be issued.

**NOTE**: This calculation is identical to the formula: n = n2 + 2*n3 + 4*n4 + 5*n5 where nx represents the block.

## Saving the State of the Outputs in Non-Volatile Memory

The configuration of the extended I/O and the state of the outputs can be stored in the non-volatile flash memory with the BN command. If no value has been set, the default of CO 0 is used (all blocks are inputs).

## Accessing Extended I/O

When configured as an output, each I/O point may be defined with the SBn and CBn commands (where n=1 through 8 and 17 through 48). Outputs may also be defined with the conditional command, OBn (where n=1 through 8 and 17 through 48).

**4080** For 5-8 axis controllers, each I/O point may be defined with the SBn and CBn commands (where n=1 through 48).

The command, OP, may also be used to set output bits, specified as blocks of data. The OP command accepts 3 parameters. The first parameter sets the values of the main output port of the controller (Outputs 1-8, block 0). The additional parameters set the value of the extended I/O as outlined:

OP m,a,b

where m is the decimal representation of the bits 1-8 (values from 0 to 255) and a,b,c,d represent the extended I/O in consecutive groups of 16 bits (values from 0 to 65535). Arguments which are given for I/O points which are configured as inputs will be ignored. The following table describes the arguments used to set the state of outputs.

| Argument | Blocks | Bits | Description |
|---|---|---|---|
| m | 0 | 1-8 | General Outputs |
| a | 2,3 | 17-32 | Extended I/O |
| b | 4,5 | 33-48 | Extended I/O |

For example, if block 8 is configured as an output, the following command may be issued:

OP 7,,7

This command will set bits 1,2,3 (block 0) and bits 33,34,35 (block 4) to 1. Bits 4 through 8 and bits 36 through 48 will be set to 0. All other bits are unaffected.

When accessing I/O blocks configured as inputs, use the TIn command. The argument 'n' refers to the block to be read (n=0,2,3 or 4). The value returned will be a decimal representation of the corresponding bits.

Individual bits can be queried using the @IN[n] function (where n=1 through 8 or 17 through 48). If the following command is issued;

**4080** Individual bits can be queried using the @IN[n] function (where n=1 through 48).

MG @IN[17]

the controller will return the state of the least significant bit of block 2 (assuming block 2 is configured as an input).

# Example Applications

## Wire Cutter

An operator activates a start switch. This causes a motor to advance the wire a distance of 10". When the motion stops, the controller generates an output signal which activates the cutter. Allowing 100 ms for the cutting completes the cycle.

Suppose that the motor drives the wire by a roller with a 2" diameter. Also assume that the encoder resolution is 1000 lines per revolution. Since the circumference of the roller equals $2\pi$ inches, and it corresponds to 4000 quadrature, one inch of travel equals:

**$4000/2\pi = 637$ count/inch**

This implies that a distance of 10 inches equals 6370 counts, and a slew speed of 5 inches per second, for example, equals 3185 count/sec.

The input signal may be applied to I1, for example, and the output signal is chosen as output 1. The motor velocity profile and the related input and output signals are shown in Fig. 7.1.

The program starts at a state that we define as #A. Here the controller waits for the input pulse on I1. As soon as the pulse is given, the controller starts the forward motion.

Upon completion of the forward move, the controller outputs a pulse for 20 ms and then waits an additional 80 ms before returning to #A for a new cycle.

| INSTRUCTION | FUNCTION |
|---|---|
| #A | Label |
| AI1 | Wait for input 1 |
| PR 6370 | Distance |
| SP 3185 | Speed |
| BGX | Start Motion |
| AMX | After motion is complete |
| SB1 | Set output bit 1 |
| WT 20 | Wait 20 ms |
| CB1 | Clear output bit 1 |
| WT 80 | Wait 80 ms |
| JP #A | Repeat the process |

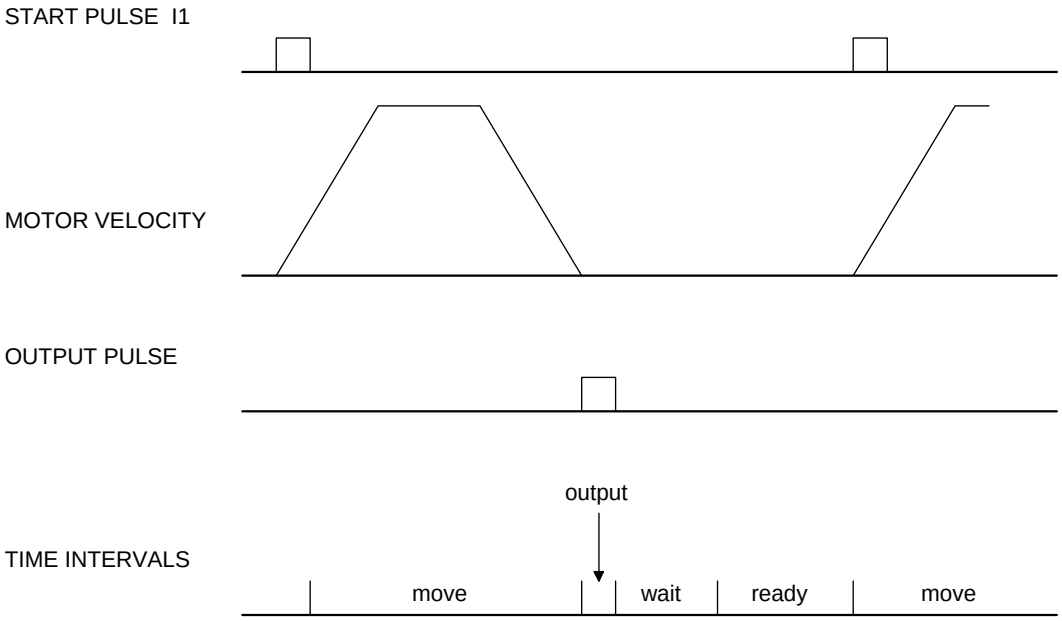



*Figure 7.1 - Motor Velocity and the Associated Input/Output signals*

## X-Y Table Controller

An X-Y-Z system must cut the pattern shown in Fig. 7.2. The X-Y table moves the plate while the Z-axis raises and lowers the cutting tool.

The solid curves in Fig. 7.2 indicate sections where cutting takes place. Those must be performed at a feed rate of 1 inch per second. The dashed line corresponds to non-cutting moves and should be performed at 5 inch per second. The acceleration rate is 0.1 g.

The motion starts at point A, with the Z-axis raised. An X-Y motion to point B is followed by lowering the Z-axis and performing a cut along the circle. Once the circular motion is completed, the Z-axis is raised and the motion continues to point C, etc.

Assume that all of the 3 axes are driven by lead screws with 10 turns-per-inch pitch. Also assume encoder resolution of 1000 lines per revolution. This results in the relationship:

1 inch = 40,000 counts

and the speeds of

1 in/sec = 40,000 count/sec
5 in/sec = 200,000 count/sec

an acceleration rate of 0.1g equals

$0.1g = 38.6 \text{ in/s2} = 1{,}544{,}000 \text{ count/s}^2$

Note that the circular path has a radius of 2" or 80000 counts, and the motion starts at the angle of 270° and traverses 360° in the CW (negative direction). Such a path is specified with the instruction

```
CR 80000,270,-360
```

Further assume that the Z must move 2” at a linear speed of 2” per second. The required motion is performed by the following instructions:

| INSTRUCTION | FUNCTION |
|---|---|
| #A | Label |
| VM XY | Circular interpolation for XY |
| VP 160000,160000 | Positions |
| VE | End Vector Motion |
| VS 200000 | Vector Speed |
| VA 1544000 | Vector Acceleration |
| BGS | Start Motion |
| AMS | When motion is complete |
| PR,,-80000 | Move Z down |
| SP,,80000 | Z speed |
| BGZ | Start Z motion |
| AMZ | Wait for completion of Z motion |
| CR 80000,270,-360 | Circle |
| VE | |
| VS 40000 | Feed rate |
| BGS | Start circular move |
| AMS | Wait for completion |
| PR,,80000 | Move Z up |
| BGZ | Start Z move |
| AMZ | Wait for Z completion |
| PR -21600 | Move X |
| SP 20000 | Speed X |
| BGX | Start X |
| AMX | Wait for X completion |
| PR,,-80000 | Lower Z |
| BGZ | |
| AMZ | |
| CR 80000,270,-360 | Z second circle move |
| VE | |
| VS 40000 | |
| BGS | |
| AMS | |
| PR,,80000 | Raise Z |
| BGZ | |
| AMZ | |
| VP -37600,-16000 | Return XY to start |
| VE | |
| VS 200000 | |
| BGS | |
| AMS | |
| EN | |

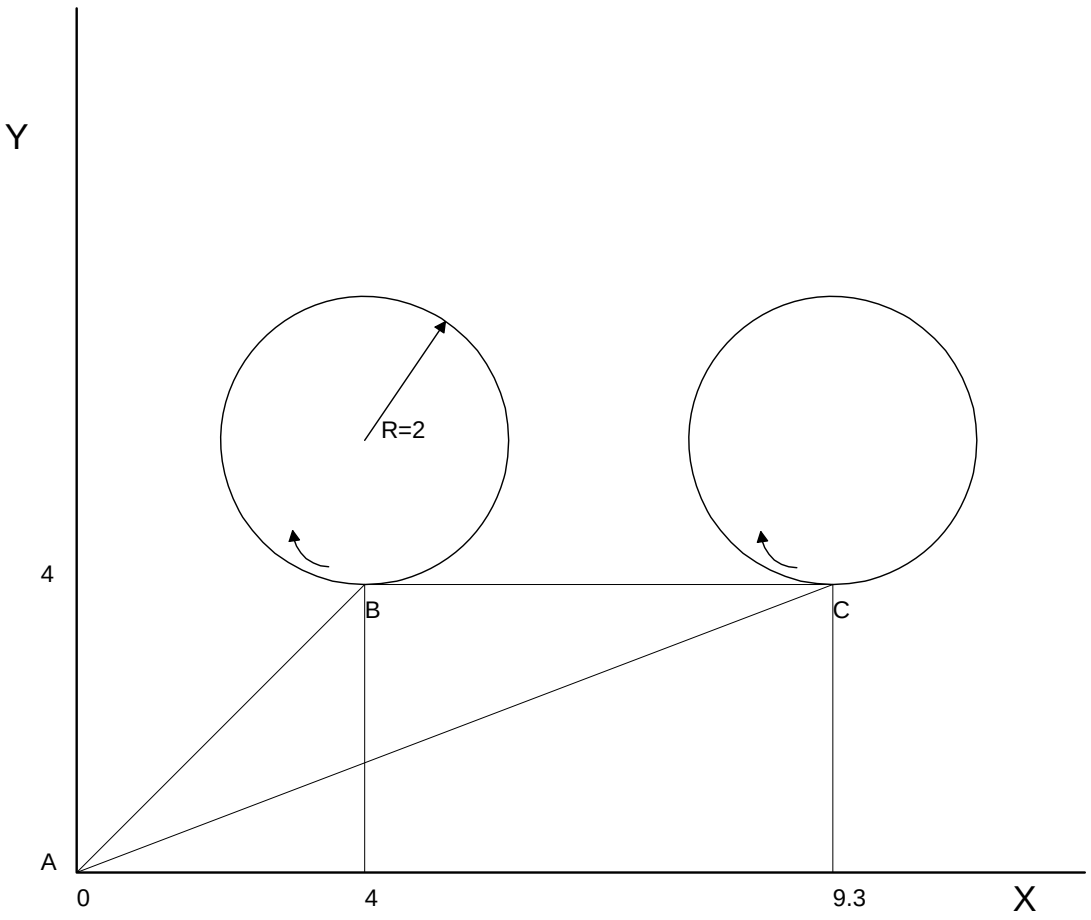



*Figure 7.2 - Motor Velocity and the Associated Input/Output signals*

## Speed Control by Joystick

The speed of a motor is controlled by a joystick. The joystick produces a signal in the range between -10V and +10V. The objective is to drive the motor at a speed proportional to the input voltage.

Assume that a full voltage of 10 Volts must produce a motor speed of 3000 rpm with an encoder resolution of 1000 lines or 4000 count/rev. This speed equals:

3000 rpm = 50 rev/sec = 200000 count/sec

The program reads the input voltage periodically and assigns its value to the variable VIN. To get a speed of 200,000 ct/sec for 10 volts, we select the speed as:

Speed = 20000 x VIN

The corresponding velocity for the motor is assigned to the VEL variable.

**Instruction**

```
#A
JG0
BGX
#B
VIN=@AN[1]
VEL=VIN*20000
JG VEL
JP #B
EN
```

## Position Control by Joystick

This system requires the position of the motor to be proportional to the joystick angle. Furthermore, the ratio between the two positions must be programmable. For example, if the control ratio is 5:1, it implies that when the joystick voltage is 5 Volts, corresponding to 1028 counts, the required motor position must be 5120 counts. The variable V3 changes the position ratio.

| INSTRUCTION | FUNCTION |
|---|---|
| #A | Label |
| V3=5 | Initial position ratio |
| DP0 | Define the starting position |
| JG0 | Set motor in jog mode as zero |
| BGX | Start |
| #B | |
| VIN=@AN[1] | Read analog input |
| V2=V1*V3 | Compute the desired position |
| V4=V2-_TPX-_TEX | Find the following error |
| V5=V4*20 | Compute a proportional speed |
| JG V5 | Change the speed |
| JP #B | Repeat the process |
| EN | End |

## Backlash Compensation by Sampled Dual-Loop

The continuous dual loop, enabled by the DV1 function is an effective way to compensate for backlash. In some cases, however, when the backlash magnitude is large, it may be difficult to stabilize the system. In those cases, it may be easier to use the sampled dual loop method described below.

This design example addresses the basic problems of backlash in motion control systems. The objective is to control the position of a linear slide precisely. The slide is to be controlled by a rotary motor, which is coupled to the slide by a lead screw. Such a lead screw has a backlash of 4 micron, and the required position accuracy is for 0.5 micron.

The basic dilemma is where to mount the sensor. If you use a rotary sensor, you get a 4 micron backlash error. On the other hand, if you use a linear encoder, the backlash in the feedback loop will cause oscillations due to instability.

An alternative approach is the dual-loop, where we use two sensors, rotary and linear. The rotary sensor assures stability (because the position loop is closed before the backlash) whereas the linear sensor provides accurate load position information. The operation principle is to drive the motor to a given rotary position near the final point. Once there, the load position is read to find the position error and the controller commands the motor to move to a new rotary position which eliminates the position error.

Since the required accuracy is 0.5 micron, the resolution of the linear sensor should preferably be twice finer. A linear sensor with a resolution of 0.25 micron allows a position error of +/-2 counts.

The dual-loop approach requires the resolution of the rotary sensor to be equal or better than that of the linear system. Assuming that the pitch of the lead screw is 2.5mm (approximately 10 turns per inch), a rotary encoder of 2500 lines per turn or 10,000 count per revolution results in a rotary resolution of 0.25 micron. This results in equal resolution on both linear and rotary sensors.

To illustrate the control method, assume that the rotary encoder is used as a feedback for the X-axis, and that the linear sensor is read and stored in the variable LINPOS. Further assume that at the start, both the position of X and the value of LINPOS are equal to zero. Now assume that the objective is to move the linear load to the position of 1000.

The first step is to command the X motor to move to the rotary position of 1000. Once it arrives we check the position of the load. If, for example, the load position is 980 counts, it implies that a correction of 20 counts must be made. However, when the X-axis is commanded to be at the position of 1000, suppose that the actual position is only 995, implying that X has a position error of 5 counts, which will be eliminated once the motor settles. This implies that the correction needs to be only 15 counts, since 5 counts out of the 20 would be corrected by the X-axis. Accordingly, the motion correction should be:

Correction = Load Position Error - Rotary Position Error

The correction can be performed a few times until the error drops below +/-2 counts. Often, this is performed in one correction cycle.

**Example:**

| INSTRUCTION | FUNCTION |
|---|---|
| #A | Label |
| DP0 | Define starting positions as zero |
| LINPOS=0 | |
| PR 1000 | Required distance |
| BGX | Start motion |
| #B | |
| AMX | Wait for completion |
| WT 50 | Wait 50 msec |
| LINPOS = _DEX | Read linear position |
| ERR=1000-LINPOS-_TEX | Find the correction |
| JP #C,@ABS[ERR]<2 | Exit if error is small |
| PR ERR | Command correction |
| BGX | |
| JP #B | Repeat the process |
| #C | |
| EN | |

THIS PAGE LEFT BLANK INTENTIONALLY

# Chapter 8 Hardware & Software Protection

## Introduction

The DMC-40x0 provides several hardware and software features to check for error conditions and to inhibit the motor on error. These features help protect the various system components from damage.

**WARNING**: Machinery in motion can be dangerous! It is the responsibility of the user to design effective error handling and safety protection as part of the machine. Since the DMC-40x0 is an integral part of the machine, the engineer should design his overall system with protection against a possible component failure on the DMC-40x0. Galil shall not be liable or responsible for any incidental or consequential damages.

## Hardware Protection

The DMC-40x0 includes hardware input and output protection lines for various error and mechanical limit conditions. These include:

### Output Protection Lines

#### Amp Enable

This signal goes low when the motor off command is given, when the position error exceeds the value specified by the Error Limit (ER) command, or when off-on-error condition is enabled (OE1) and the abort command is given. Each axis amplifier has separate amplifier enable lines. This signal also goes low when the watch-dog timer is activated, or upon reset.

**Note:** The standard configuration of the AEN signal is TTL active low. Both the polarity and the amplitude can be changed. To make these changes, see section entitled Amplifier Circuit in Chapter 3.

#### Error Output

The error output is a TTL signal which indicates an error condition in the controller. This signal is available on the interconnect module as ERR. When the error signal is low, this indicates one of the following error conditions:

1. At least one axis has a position error greater than the error limit. The error limit is set by using the command ER.
2. The reset line on the controller is held low or is being affected by noise.
3. There is a failure on the controller and the processor is resetting itself.
4. There is a failure with the output IC which drives the error signal.

## Input Protection Lines

### General Abort

A low input stops commanded motion instantly without a controlled deceleration. For any axis in which the Off-On-Error function is enabled, the amplifiers will be disabled. This could cause the motor to 'coast' to a stop. If the Off-On-Error function is not enabled, the motor will instantaneously stop and servo at the current position. The Off-On-Error function is further discussed in this chapter.

The Abort input by default will also halt program execution; this can be changed by changing the 5th field of the CN command. See the CN command in the command reference for more information.

### Selective Abort

The controller can be configured to provide an individual abort for each axis. Activation of the selective abort signal will act the same as the Abort Input but only on the specific axis. To configure the controller for selective abort, issue the command CN,,,1. This configures the inputs 5,6,7,8,13,14,15,16 to act as selective aborts for axes A,B,C,D,E,F,G,H respectively.

### ELO (Electronic Lock Out)

Used in conjunction with Galil amplifiers, this input allows the user the shutdown the amplifier at a hardware level. For more detailed information on how specific Galil amplifiers behave when the ELO is triggered, see Integrated Amplifiers and Drivers in the Appendices.

### Forward Limit Switch

Low input inhibits motion in forward direction. If the motor is moving in the forward direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the forward direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user). The CN command can be used to change the polarity of the limit switches.

### Reverse Limit Switch

Low input inhibits motion in reverse direction. If the motor is moving in the reverse direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the reverse direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user). The CN command can be used to change the polarity of the limit switches.

# Software Protection

The DMC-40x0 provides a programmable error limit. The error limit can be set for any number between 0 and 2147483647 using the ER n command. The default value for ER is 16384.

**Example:**

```
ER 200,300,400,500      Set X-axis error limit for 200, Y-axis error limit to 300, Z-
                        axis error limit to 400 counts, W-axis error limit to 500
                        counts
ER,1,,10                Set Y-axis error limit to 1 count, set W-axis error limit to
                        10 counts.
```

The units of the error limit are quadrature counts. The error is the difference between the command position and actual encoder position. If the absolute value of the error exceeds the value specified by ER, the controller will generate several signals to warn the host system of the error condition. These signals include:

| Signal or Function | State if Error Occurs |
|---|---|
| # POSERR | Jumps to automatic excess position error subroutine |
| Error Light | Turns on |
| OE Function | Shuts motor off if OE1 |
| AEN Output Line | Goes low |

The Jump on Condition statement is useful for branching on a given error within a program. The position error of X,Y,Z and W can be monitored during execution using the TE command.

## Programmable Position Limits

The DMC-40x0 provides programmable forward and reverse position limits. These are set by the BL and FL software commands. Once a position limit is specified, the DMC-40x0 will not accept position commands beyond the limit. Motion beyond the limit is also prevented.

**Example:**

```
DP0,0,0               Define Position
BL -2000,-4000,-8000  Set Reverse position limit
FL 2000,4000,8000     Set Forward position limit
JG 2000,2000,2000     Jog
BG XYZ                Begin
```

*(motion stops at forward limits)*

## Off-On-Error

The DMC-40x0 controller has a built in function which can turn off the motors under certain error conditions. This function is known as 'Off-On-Error". To activate the OE function for each axis, specify 1 for X,Y,Z and W axis. To disable this function, specify 0 for the axes. When this function is enabled, the specified motor will be disabled under the following 3 conditions:

1. The position error for the specified axis exceeds the limit set with the command, ER
2. The abort command is given
3. The abort input is activated with a low signal.

**Note:** If the motors are disabled while they are moving, they may 'coast' to a stop because they are no longer under servo control.

To re-enable the system, use the Reset (RS) or Servo Here (SH) command.

**Examples:**

```
OE 1,1,1,1     Enable off-on-error for X,Y,Z and W
OE 0,1,0,1     Enable off-on-error for Y and W axes and disable off-on-error for W and Z axes
```

## Automatic Error Routine

The #POSERR label causes the statements following to be automatically executed if error on any axis exceeds the error limit specified by ER. The error routine must be closed with the RE command. The RE command returns from the error subroutine to the main program.

**NOTE**: The Error Subroutine will be entered again unless the error condition is gone.

**Example:**

```
#A;JP #A;EN          "Dummy" program
#POSERR              Start error routine on error
MG"error"            Send message
SB 1                 Fire relay
STX                  Stop motor
AMX                  After motor stops
SHX                  Servo motor here to clear error
RE                   Return to main program
```

## Limit Switch Routine

The DMC-40x0 provides forward and reverse limit switches which inhibit motion in the respective direction. There is also a special label for automatic execution of a limit switch subroutine. The #LIMSWI label specifies the start of the limit switch subroutine. This label causes the statements following to be automatically executed if any limit switch is activated and that axis motor is moving in that direction. The RE command ends the subroutine.

The state of the forward and reverse limit switches may also be tested during the jump-on-condition statement. The _LR condition specifies the reverse limit and _LF specifies the forward limit. X,Y,Z, or W following LR or LF specifies the axis. The CN command can be used to configure the polarity of the limit switches.

**Limit Switch Example:**

```
#A;JP #A;EN           Dummy Program
#LIMSWI               Limit Switch Utility
V1=_LFX               Check if forward limit
V2=_LRX               Check if reverse limit
JP#LF,V1=0            Jump to #LF if forward
JP#LR,V2=0            Jump to #LR if reverse
JP#END                Jump to end
#LF                   #LF
MG "FORWARD LIMIT"    Send message
STX;AMX               Stop motion
PR-1000;BGX;AMX       Move in reverse
JP#END                End
#LR                   #LR
MG "REVERSE LIMIT"    Send message
STX;AMX               Stop motion
PR1000;BGX;AMX        Move forward
#END                  End
RE                    Return to main program
```

# Chapter 9 Troubleshooting

## Overview

The following discussion may help you get your system to work.

Potential problems have been divided into groups as follows:

1. Installation
2. Stability and Compensation
3. Operation

The various symptoms along with the cause and the remedy are described in the following tables.

## Installation

| SYMPTOM | DIAGNOSIS | CAUSE | REMEDY |
|---|---|---|---|
| Motor runs away with no connections from controller to amplifier input. | Adjusting offset causes the motor to change speed. | 1. Amplifier has an internal offset.<br><br>2. Damaged amplifier. | Adjust amplifier offset. Amplifier offset may also be compensated by use of the offset configuration on the controller (see the OF command).<br><br>Replace amplifier. |
| Motor is enabled even when MO command is given | The SH command disables the motor | 1. The amplifier requires the a different Amplifier Enable setting on the Interconnect Module | Refer to Chapter 3 or contact Galil. |
| Unable to read main or auxiliary encoder input. | The encoder does not work when swapped with another encoder input. | 1. Wrong encoder connections.<br><br>2. Encoder is damaged<br><br>3. Encoder configuration incorrect. | Check encoder wiring. For single ended encoders (CHA and CHB only) do not make any connections to the CHA- and CHB- inputs.<br>Replace encoder<br><br>Check CE command |

| | | | |
|---|---|---|---|
| Unable to read main or auxiliary encoder input. | The encoder works correctly when swapped with another encoder input. | 1. Wrong encoder connections.<br><br>2. Encoder configuration incorrect.<br><br>3. Encoder input or controller is damaged | Check encoder wiring. For single ended encoders (MA+ and MB+ only) do not make any connections to the MA- and MB- inputs.<br>Check CE command<br><br>Contact Galil |
| Encoder Position Drifts | Swapping cables fixes the problem | 1. Poor Connections / intermittent cable | Review all connections and connector contacts. |
| Encoder Position Drifts | Significant noise can be seen on MA+ and / or MB+ encoder signals | 1. Noise | Shield encoder cables<br>Avoid placing power cables near encoder cables<br>Avoid Ground Loops<br>Use differential encoders<br>Use +/-12V encoders |

# Stability

| SYMPTOM | DIAGNOSIS | CAUSE | REMEDY |
|---|---|---|---|
| Servo motor runs away when the loop is closed. | Reversed Motor Type corrects situation (MT -1) | 1. Wrong feedback polarity. | Reverse Motor or Encoder Wiring (remember to set Motor Type back to default value: MT 1) |
| Motor oscillates. | | 2. Too high gain or too little damping. | Decrease KI and KP. Increase KD. |

# Operation

| SYMPTOM | DIAGNOSIS | CAUSE | REMEDY |
|---|---|---|---|
| Controller rejects commands. | Response of controller from TC1 diagnoses error. | 1. Anything | Correct problem reported by TC1 |
| Motor Doesn't Move | Response of controller from TC1 diagnoses error. | 2. Anything | Correct problem reported by SC |

# Chapter 10 Theory of Operation

## Overview

The following discussion covers the operation of motion control systems. A typical motion control system consists of the elements shown in Fig 10.1.

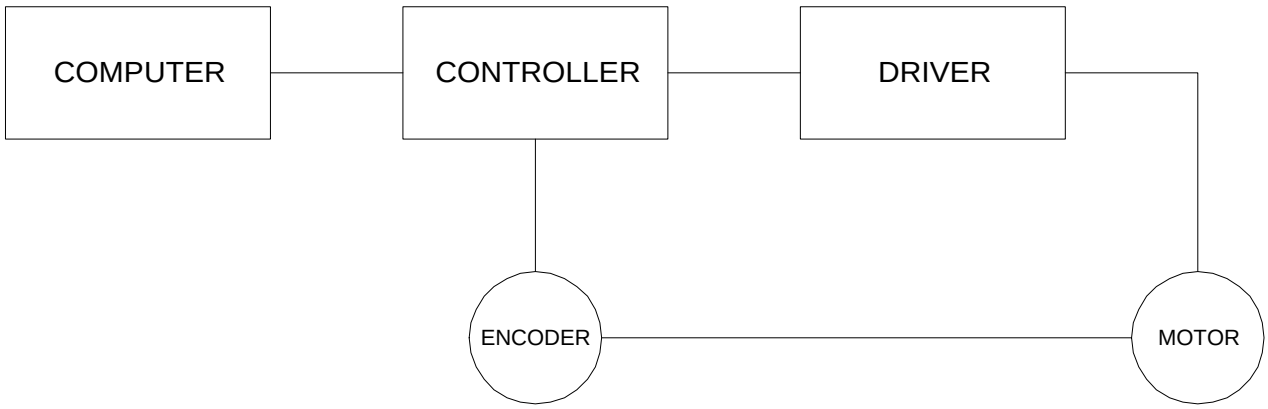



*Figure 10.1 - Elements of Servo Systems*

The operation of such a system can be divided into three levels, as illustrated in Fig. 10.2. The levels are:

1. Closing the Loop
2. Motion Profiling
3. Motion Programming

The first level, the closing of the loop, assures that the motor follows the commanded position. This is done by closing the position loop using a sensor. The operation at the basic level of closing the loop involves the subjects of modeling, analysis, and design. These subjects will be covered in the following discussions.

The motion profiling is the generation of the desired position function. This function, R(t), describes where the motor should be at every sampling period. Note that the profiling and the closing of the loop are independent functions. The profiling function determines where the motor should be and the closing of the loop forces the motor to follow the commanded position

The highest level of control is the motion program. This can be stored in the host computer or in the controller. This program describes the tasks in terms of the motors that need to be controlled, the distances and the speed.

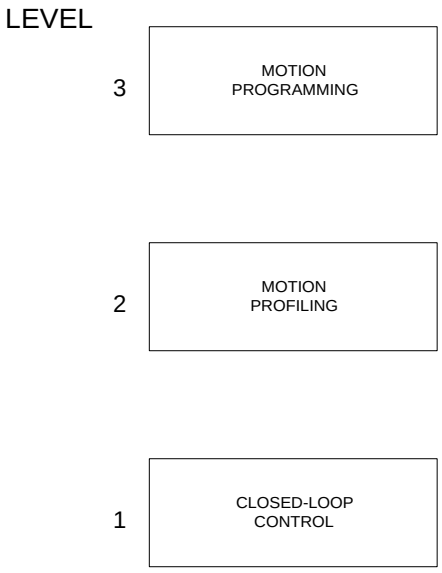



*Figure 10.2 - Levels of Control Functions*

The three levels of control may be viewed as different levels of management. The top manager, the motion program, may specify the following instruction, for example.

```
PR 6000,4000
SP 20000,20000
AC 200000,00000
BG X
AD 2000
BG Y
EN
```

This program corresponds to the velocity profiles shown in Fig. 10.3. Note that the profiled positions show where the motors must be at any instant of time.

Finally, it remains up to the servo system to verify that the motor follows the profiled position by closing the servo loop.

The following section explains the operation of the servo system. First, it is explained qualitatively, and then the explanation is repeated using analytical tools for those who are more theoretically inclined.

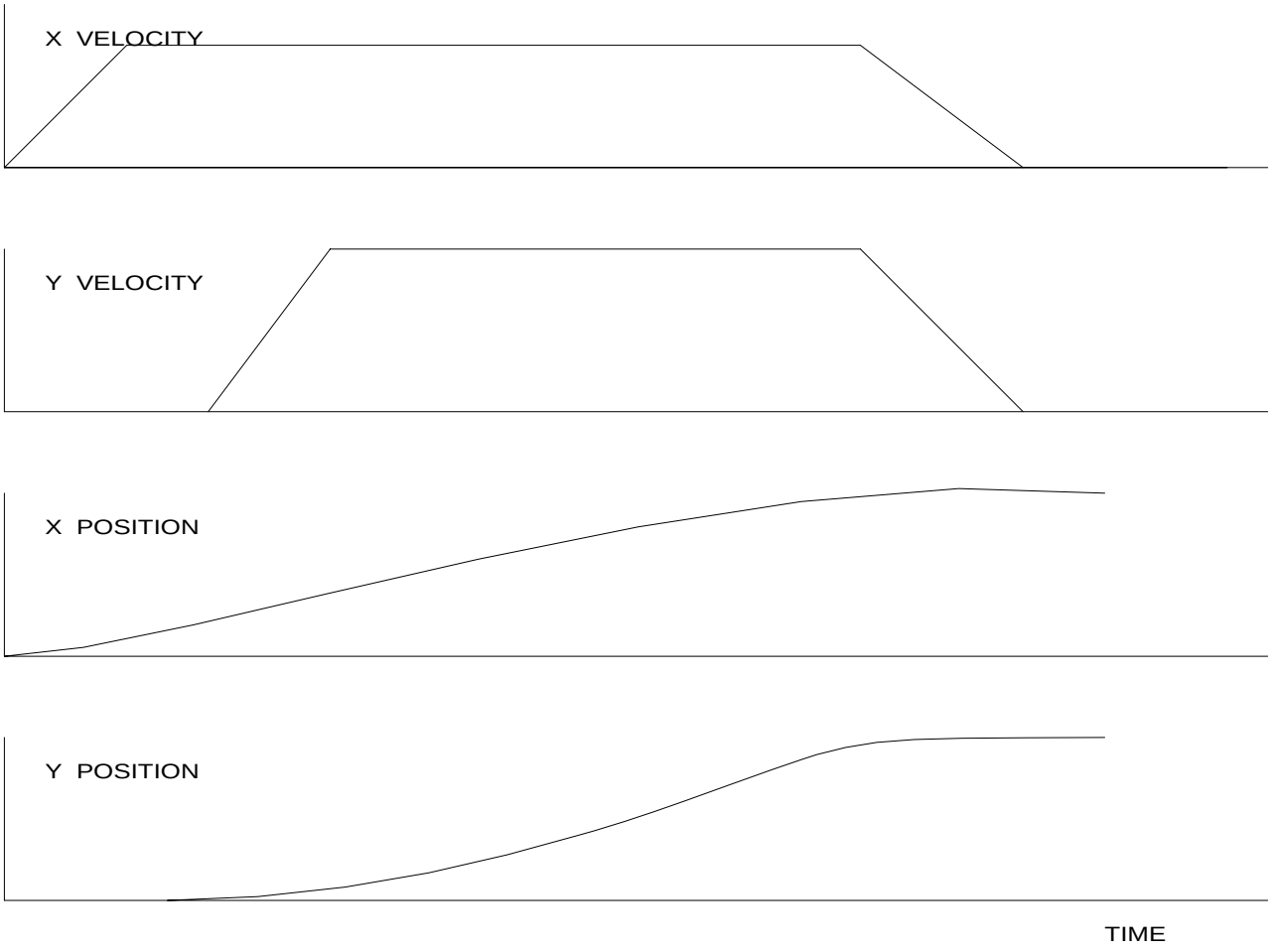



*Figure 10.3 - Velocity and Position Profiles*

# Operation of Closed-Loop Systems

To understand the operation of a servo system, we may compare it to a familiar closed-loop operation, adjusting the water temperature in the shower. One control objective is to keep the temperature at a comfortable level, say 90 degrees F. To achieve that, our skin serves as a temperature sensor and reports to the brain (controller). The brain compares the actual temperature, which is called the feedback signal, with the desired level of 90 degrees F. The difference between the two levels is called the error signal. If the feedback temperature is too low, the error is positive, and it triggers an action which raises the water temperature until the temperature error is reduced sufficiently.

The closing of the servo loop is very similar. Suppose that we want the motor position to be at 90 degrees. The motor position is measured by a position sensor, often an encoder, and the position feedback is sent to the controller. Like the brain, the controller determines the position error, which is the difference between the commanded position of 90 degrees and the position feedback. The controller then outputs a signal that is proportional to the position error. This signal produces a proportional current in the motor, which causes a motion until the error is reduced. Once the error becomes small, the resulting current will be too small to overcome the friction, causing the motor to stop.

The analogy between adjusting the water temperature and closing the position loop carries further. We have all learned the hard way, that the hot water faucet should be turned at the “right” rate. If you turn it too slowly, the temperature response will be slow, causing discomfort. Such a slow reaction is called over-damped response.

The results may be worse if we turn the faucet too fast. The overreaction results in temperature oscillations. When the response of the system oscillates, we say that the system is unstable. Clearly, unstable responses are bad when we want a constant level.

What causes the oscillations? The basic cause for the instability is a combination of delayed reaction and high gain. In the case of the temperature control, the delay is due to the water flowing in the pipes. When the human reaction is too strong, the response becomes unstable.

Servo systems also become unstable if their gain is too high. The delay in servo systems is between the application of the current and its effect on the position. Note that the current must be applied long enough to cause a significant effect on the velocity, and the velocity change must last long enough to cause a position change. This delay, when coupled with high gain, causes instability.

This motion controller includes a special filter which is designed to help the stability and accuracy. Typically, such a filter produces, in addition to the proportional gain, damping and integrator. The combination of the three functions is referred to as a PID filter.

The filter parameters are represented by the three constants KP, KI and KD, which correspond to the proportional, integral and derivative term respectively.

The damping element of the filter acts as a predictor, thereby reducing the delay associated with the motor response.

The integrator function, represented by the parameter KI, improves the system accuracy. With the KI parameter, the motor does not stop until it reaches the desired position exactly, regardless of the level of friction or opposing torque.

The integrator also reduces the system stability. Therefore, it can be used only when the loop is stable and has a high gain.

The output of the filter is applied to a digital-to-analog converter (DAC). The resulting output signal in the range between +10 and -10 Volts is then applied to the amplifier and the motor.

The motor position, whether rotary or linear is measured by a sensor. The resulting signal, called position feedback, is returned to the controller for closing the loop.

The following section describes the operation in a detailed mathematical form, including modeling, analysis and design.

# System Modeling

The elements of a servo system include the motor, driver, encoder and the controller. These elements are shown in Fig. 10.4. The mathematical model of the various components is given below.

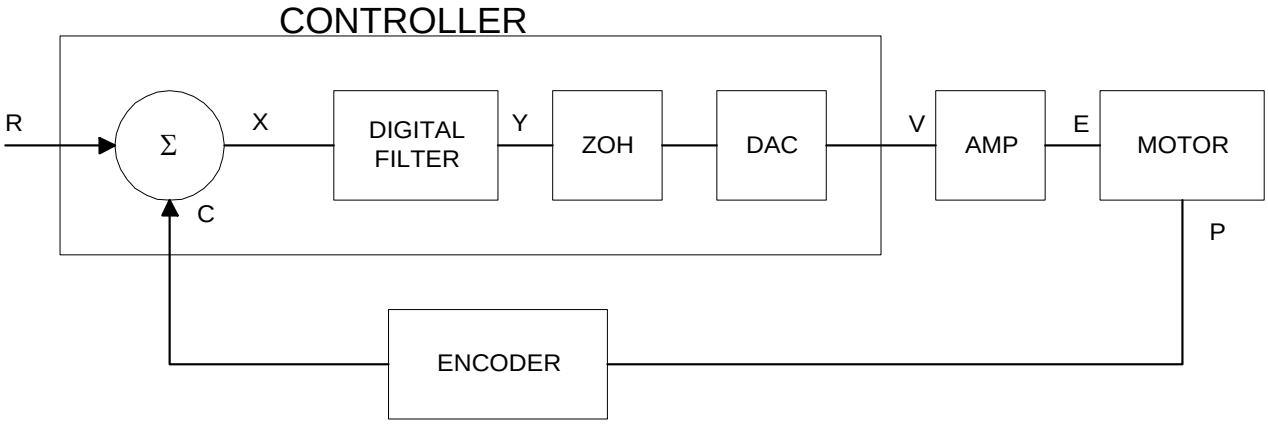



*Figure 10.4 - Functional Elements of a Motion Control System*

## Motor-Amplifier

The motor amplifier may be configured in three modes:

1. Voltage Drive
2. Current Drive
3. Velocity Loop

The operation and modeling in the three modes is as follows:

### Voltage Drive

The amplifier is a voltage source with a gain of Kv [V/V]. The transfer function relating the input voltage, V, to the motor position, P, is

$$P/V = K_V / \left[ K_t S (ST_m + 1)(ST_e + 1) \right]$$

where

$$T_m = RJ / K_t^2 \quad [s]$$

and

$$T_e = L/R \qquad [s]$$

and the motor parameters and units are

| | |
|---|---|
| Kt | Torque constant [Nm/A] |
| R | Armature Resistance □ |
| J | Combined inertia of motor and load [kg.m2] |
| L | Armature Inductance [H] |

When the motor parameters are given in English units, it is necessary to convert the quantities to MKS units. For example, consider a motor with the parameters:

Kt = 14.16 oz - in/A = 0.1 Nm/A

R = 2 Ω

J = 0.0283 oz-in-s2 = 2.10-4 kg . m2

L = 0.004H

Then the corresponding time constants are

Tm = 0.04 sec

and

Te = 0.002 sec

Assuming that the amplifier gain is Kv = 4, the resulting transfer function is

P/V = 40/[s(0.04s+1)(0.002s+1)]

### Current Drive

The current drive generates a current I, which is proportional to the input voltage, V, with a gain of Ka. The resulting transfer function in this case is

$P/V = Ka\ Kt / Js^2$

where Kt and J are as defined previously. For example, a current amplifier with Ka = 2 A/V with the motor described by the previous example will have the transfer function:

$P/V = 1000/s^2$ [rad/V]

If the motor is a DC brushless motor, it is driven by an amplifier that performs the commutation. The combined transfer function of motor amplifier combination is the same as that of a similar brush motor, as described by the previous equations.

## Velocity Loop

The motor driver system may include a velocity loop where the motor velocity is sensed by a tachometer and is fed back to the amplifier. Such a system is illustrated in Fig. 10.5. Note that the transfer function between the input voltage V and the velocity ω is:

$\omega /V = [K_a K_t/Js]/[1+K_a K_t K_g/Js] = 1/[K_g(sT_1+1)]$

where the velocity time constant, T1, equals

$T1 = J/K_a K_t K_g$

This leads to the transfer function

$P/V = 1/[K_g s(sT1+1)]$

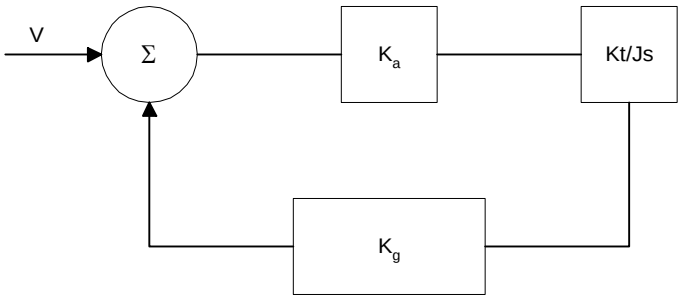



*Figure 10.5 - Elements of velocity loops*

The resulting functions derived above are illustrated by the block diagram of Fig. 10.6.

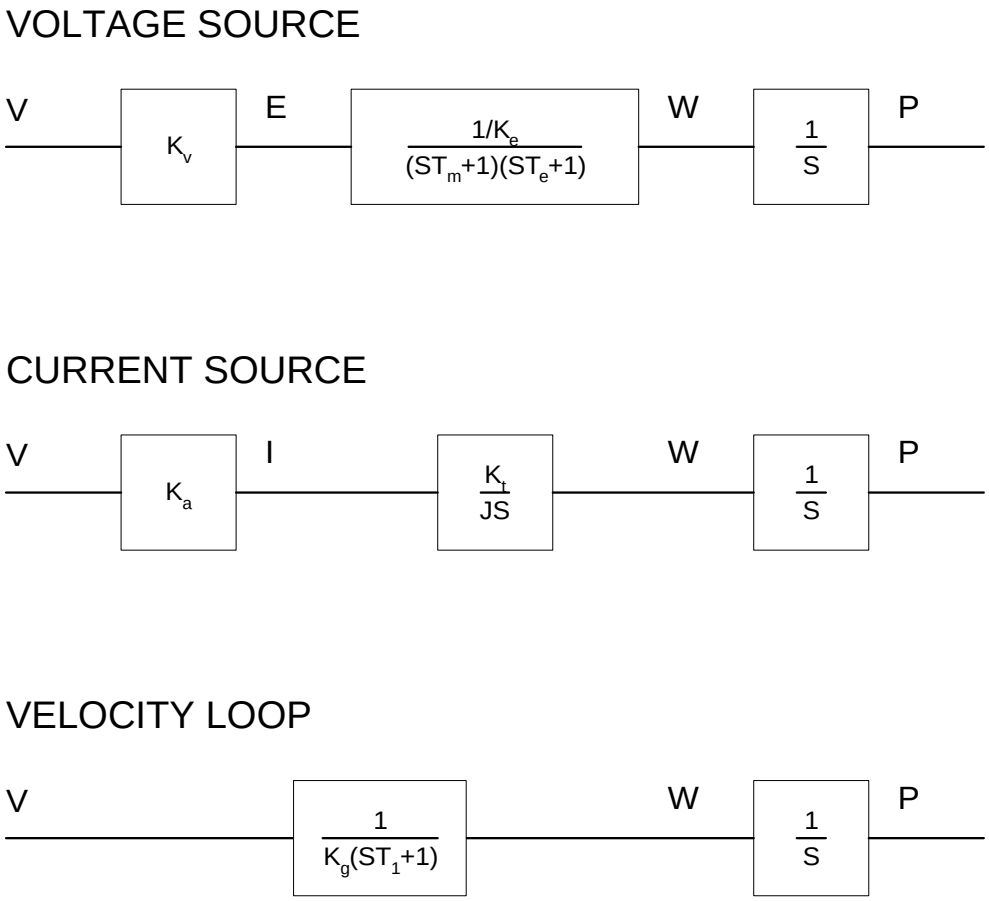



*Figure 10.6 - Mathematical model of the motor and amplifier in three operational modes*

## Encoder

The encoder generates N pulses per revolution. It outputs two signals, Channel A and B, which are in quadrature. Due to the quadrature relationship between the encoder channels, the position resolution is increased to 4N quadrature counts/rev.

The model of the encoder can be represented by a gain of

$K_f = 4N/2\pi$ [count/rad]

For example, a 1000 lines/rev encoder is modeled as

$K_f = 638$

## DAC

The DAC or D-to-A converter converts a 16-bit number to an analog voltage. The input range of the numbers is 65536 and the output voltage range is +/-10V or 20V. Therefore, the effective gain of the DAC is

K= 20/65536 = 0.0003 [V/count]

## Digital Filter

The digital filter has three element in series: PID, low-pass and a notch filter. The transfer function of the filter. The transfer function of the filter elements are:

PID $D(z) = \frac{K(Z-A)}{Z} + \frac{CZ}{Z-1}$

Low-pass $L(z) = \frac{1-B}{Z-B}$

Notch $N(z) = \frac{(Z-z)(Z-\bar{z})}{(Z-p)(Z-\bar{p})}$

The filter parameters, K, A, C and B are selected by the instructions KP, KD, KI and PL, respectively. The relationship between the filter coefficients and the instructions are:

K = (KP + KD)
A = KD/(KP + KD)
C = KI/2
B = PL

The PID and low-pass elements are equivalent to the continuous transfer function G(s).

G(s) = (P + sD + I/s) ∗ a / (s + a)

where,

P = KP
D = T • KD
I = KI/2T

$a = \frac{1}{T} ln\left(\frac{1}{B}\right)$

where T is the sampling period, and B is the pole setting

For example, if the filter parameters of the DMC-40x0 are

KP = 16
KD = 144
KI = 2
PL = 0.75
T = 0.001 s

the digital filter coefficients are

K = 160
A = 0.9
C = 1
a = 250 rad/s

and the equivalent continuous filter, G(s), is

G(s) = [16 + 0.144s + 1000/s} ∗ 250/ (s+250)

The notch filter has two complex zeros, Z and z, and two complex poles, P and p.

The effect of the notch filter is to cancel the resonance affect by placing the complex zeros on top of the resonance poles. The notch poles, P and p, are programmable and are selected to have sufficient damping. It is best to select

the notch parameters by the frequency terms. The poles and zeros have a frequency in Hz, selected by the command NF. The real part of the poles is set by NB and the real part of the zeros is set by NZ.

The most simple procedure for setting the notch filter, identify the resonance frequency and set NF to the same value. Set NB to about one half of NF and set NZ to a low value between zero and 5.

### ZOH

The ZOH, or zero-order-hold, represents the effect of the sampling process, where the motor command is updated once per sampling period. The effect of the ZOH can be modeled by the transfer function

$H(s) = 1/(1+sT/2)$

If the sampling period is T = 0.001, for example, H(s) becomes:

$H(s) = 2000/(s+2000)$

However, in most applications, H(s) may be approximated as one.

This completes the modeling of the system elements. Next, we discuss the system analysis.

## System Analysis

To analyze the system, we start with a block diagram model of the system elements. The analysis procedure is illustrated in terms of the following example.

Consider a position control system with the DMC-40x0 controller and the following parameters:

| | | |
|---|---|---|
| Kt = 0.1 | Nm/A | Torque constant |
| J = 2.10-4 | kg.m2 | System moment of inertia |
| R = 2 | Ω | Motor resistance |
| Ka = 4 | Amp/Volt | Current amplifier gain |
| KP = 12.5 | | Digital filter gain |
| KD = 245 | | Digital filter zero |
| KI = 0 | | No integrator |
| N = 500 | Counts/rev | Encoder line density |
| T = 1 | ms | Sample period |

The transfer function of the system elements are:

**Motor**

$M(s) = P/I = Kt/Js2 = 500/s^2$ [rad/A]

**Amp**

$K_a = 4$ [Amp/V]

**DAC**

$K_d = 0.0003$ [V/count]

**Encoder**

$K_f = 4N/2\pi = 318$ [count/rad]

**ZOH**

$2000/(s+2000)$

**Digital Filter**

KP = 12.5, KD = 245, T = 0.001

Therefore,

$D(z) = 1030 (z-0.95)/Z$

Accordingly, the coefficients of the continuous filter are:

P = 50

D = 0.98

The filter equation may be written in the continuous equivalent form:

$G(s) = 50 + 0.98s = .098\ (s+51)$

The system elements are shown in Fig. 10.7.

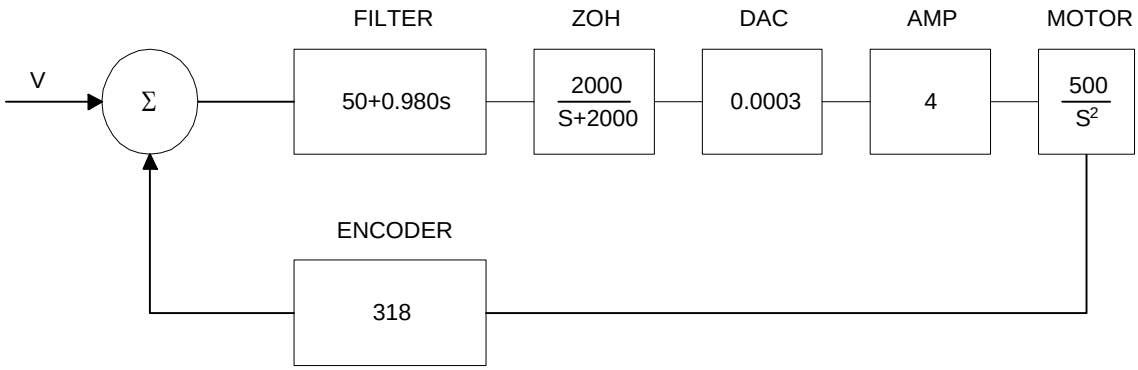



*Figure 10.7 - Mathematical model of the control system*

The open loop transfer function, A(s), is the product of all the elements in the loop.

$A = 390{,}000\ (s+51)/[s^2(s+2000)]$

To analyze the system stability, determine the crossover frequency, ωc at which A(j ωc) equals one. This can be done by the Bode plot of A(j ωc), as shown in Fig. 10.8.

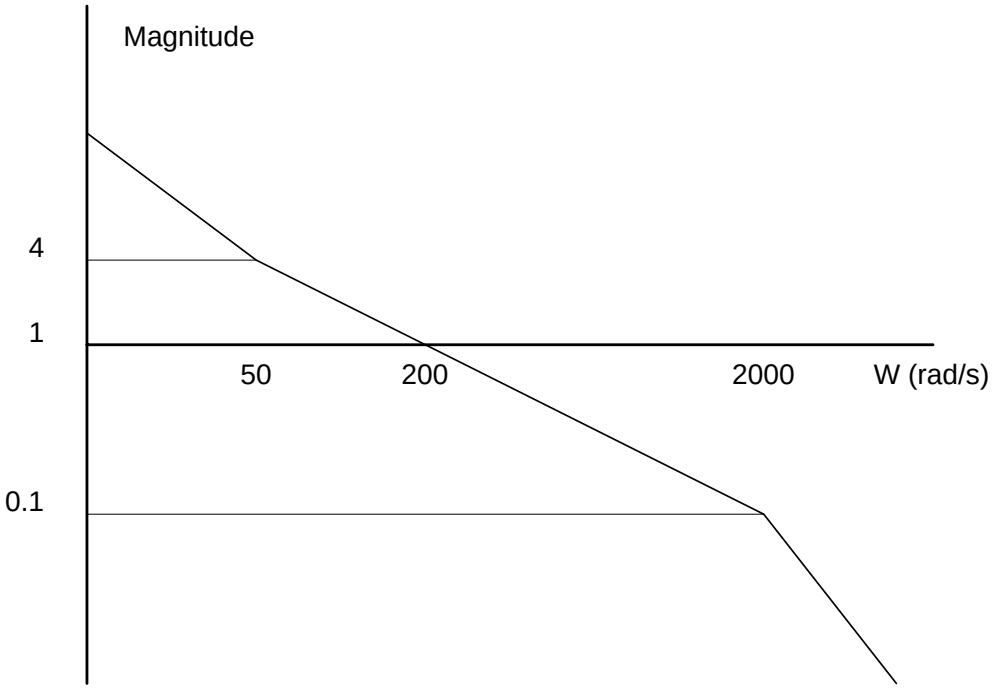



*Figure 10.8 - Bode plot of the open loop transfer function*

For the given example, the crossover frequency was computed numerically resulting in 200 rad/s.

Next, we determine the phase of A(s) at the crossover frequency.

$$A(j200) = 390{,}000\ (j200+51)/[(j200)^2\ .\ (j200 + 2000)]$$

$$\alpha = \text{Arg}[A(j200)] = \tan^{-1}(200/51) - 180° - \tan^{-1}(200/2000)$$

$$\alpha = 76° - 180° - 6° = -110°$$

Finally, the phase margin, PM, equals

$$PM = 180° + \alpha = 70°$$

As long as PM is positive, the system is stable. However, for a well damped system, PM should be between 30 degrees and 45 degrees. The phase margin of 70 degrees given above indicated overdamped response.

Next, we discuss the design of control systems.

# System Design and Compensation

The closed-loop control system can be stabilized by a digital filter, which is preprogrammed in the DMC-40x0 controller. The filter parameters can be selected by the user for the best compensation. The following discussion presents an analytical design method.

## The Analytical Method

The analytical design method is aimed at closing the loop at a crossover frequency, □c, with a phase margin PM. The system parameters are assumed known. The design procedure is best illustrated by a design example.

Consider a system with the following parameters:

| | | |
|---|---|---|
| Kt | Nm/A | Torque constant |
| J = 2.10-4 | kg.m2 | System moment of inertia |
| R = 2 | Ω | Motor resistance |
| Ka = 2 | Amp/Volt | Current amplifier gain |
| N = 1000 | Counts/rev | Encoder line density |

The DAC of theDMC-40x0 outputs +/-10V for a 16-bit command of +/-32768 counts.

The design objective is to select the filter parameters in order to close a position loop with a crossover frequency of $\omega_c$ = 500 rad/s and a phase margin of 45 degrees.

The first step is to develop a mathematical model of the system, as discussed in the previous system.

**Motor**

M(s) = P/I = Kt/Js2 = 1000/s2

**Amp**

Ka = 2 [Amp/V]

**DAC**

Kd = 10/32768 = .0003

**Encoder**

Kf = 4N/2□ = 636

**ZOH**

H(s) = 2000/(s+2000)

**Compensation Filter**

$G(s) = P + sD$

The next step is to combine all the system elements, with the exception of G(s), into one function, L(s).

$$L(s) = M(s)\, K_a\, K_d\, K_f\, H(s) = 3.17 * 10^6/[s^2(s+2000)]$$

Then the open loop transfer function, A(s), is

$$A(s) = L(s)\, G(s)$$

Now, determine the magnitude and phase of L(s) at the frequency $\omega_c = 500$.

$$L(j500) = 3.17 * 10^6/[(j500)^2\ (j500+2000)]$$

This function has a magnitude of

$$|L(j500)| = 0.00625$$

and a phase

$$\text{Arg}[L(j500)] = -180° - \tan^{-1}(500/2000) = -194°$$

G(s) is selected so that A(s) has a crossover frequency of 500 rad/s and a phase margin of 45 degrees. This requires that

$$|A(j500)| = 1$$
$$\text{Arg}\,[A(j500)] = -135°$$

However, since

$$A(s) = L(s)\, G(s)$$

then it follows that G(s) must have magnitude of

$$|G(j500)| = |A(j500)/L(j500)| = 160$$

and a phase

$$\arg\,[G(j500)] = \arg\,[A(j500)] - \arg\,[L(j500)] = -135° + 194° = 59°$$

In other words, we need to select a filter function G(s) of the form

**G(s) = P + sD**

so that at the frequency $\omega_c = 500$, the function would have a magnitude of 160 and a phase lead of 59 degrees.

These requirements may be expressed as:

$$|G(j500)| = |P + (j500D)| = 160$$

and

$$\arg\,[G(j500)] = \tan^{-1}[500D/P] = 59°$$

The solution of these equations leads to:

$$P = 160\cos 59° = 82.4$$
$$500D = 160\sin 59° = 137$$

Therefore,

$$D = 0.274$$

and

$$G = 82.4 + 0.2744s$$

The function G is equivalent to a digital filter of the form:

$$D(z) = KP + KD(1-z^{-1})$$

where

$P = KP$

$D = KD * T$

and

$KD = D/T$

Assuming a sampling period of T=1ms, the parameters of the digital filter are:

$KP = 82.4$

$KD = 247.4$

The DMC-40x0 can be programmed with the instruction:

KP 82.4

KD 68.6

In a similar manner, other filters can be programmed. The procedure is simplified by the following table, which summarizes the relationship between the various filters.

## Equivalent Filter Form - DMC-40x0

| | |
|---|---|
| Digital | $D(z) = [K(z-A/z) + Cz/(z-1)] * (1-B)/(Z-B)$ |
| Digital | $D(z) = [KP + KD(1-z^{-1}) + KI/2(1-z^{-1})] * (1-B)/(Z-B)$ |
| KP, KD, KI, PL | $K = (KP + KD)$ |
| | $A = KD/(KP+KD)$ |
| | $C = KI/2$ |
| | $B = PL$ |
| Continuous | $G(s) = (P + Ds + I/s) * a/s+a$ |
| PID, T | $P = KP$ |
| | $D = T * KD$ |
| | $I = KI / 2T$ |
| | $a = 1/T \ln(1/PL)$ |

**THIS PAGE LEFT BLANK INTENTIONALLY**

# Appendices

## Electrical Specifications

### Servo Control

| | |
|---|---|
| MCMn Amplifier Command: | +/-10 volt analog signal. Resolution 16-bit DAC or 0.0003 volts. 3 mA maximum.<br>Output impedance – 500Ω |
| MA+,MA-,MB+,MB-,MI+,MI- Encoder and Auxiliary | TTL compatible, but can accept up to +/-12 volts. Quadrature phase on CHA, CHB. Can accept single-ended (A+,B+ only) or differential (A+,A-,B+,B-). Maximum A, B edge rate: 12 MHz. Minimum IDX pulse width: 80 nsec. |

### Stepper Control

| | |
|---|---|
| STPn (Step) | TTL (0-5 volts) level at 50% duty cycle. 3,000,000 pulses/sec maximum frequency |
| DIRn (Direction) | TTL (0-5 volts) |

### Input / Output

| | |
|---|---|
| Limit Switch Inputs, Home Inputs.<br>DI1 thru DI8 Uncommitted Inputs and Abort Input<br>DI9 thru DI16 Uncommitted Inputs (DMC-4050 through DMC-4080 only) | 2.2K ohm in series with opto-isolator. Active high or low requires at least 1mA to activate. Once activated, the input requires the current to go below 0.5ma. All Limit Switch and Home inputs use one common voltage (LSCOM) which can accept up to 24 volts. Voltages above 24 volts require an additional resistor.<br>$\geq$ 1 mA = ON; $\leq$ 0.5 mA = OFF |
| AI1 thru AI8 Analog Inputs: | Standard configuration is +/-10 volts. 12-Bit Analog-to-Digital converter. 16-bit optional. |
| DO1 thru DO8 Outputs: | High power Opto-Isolated – 500mA souring |
| DO9 thru DO16 Outputs:<br>(DMC-4050 through DMC-4080 only) | High power Opto-Isolated – 500mA sourcing |

| | |
|---|---|
| IO17 thru IO48 | Extended configurable I/O<br>Standard 3.3V logic with 5V option |
| DI81, DI82 | Auxiliary Encoder Inputs for A (X) axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 volts. |
| DI83, DI84<br>(DMC-4020 through DMC-4080 only) | Auxiliary Encoder Inputs for B (Y) axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 volts. |
| DI85, DI86<br>(DMC-4030 through DMC-4080 only) | Auxiliary Encoder Inputs for C (Z) axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 volts. |
| DI87, DI88<br>(DMC-4040 through DMC-4080 only) | Auxiliary Encoder Inputs for D (W) axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 volts. |
| DI89, DI90<br>(DMC-4050 through DMC-4080 only) | Auxiliary Encoder Inputs for E axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 volts. |
| DI91, DI92<br>(DMC-4060 through DMC-4080 only) | Auxiliary Encoder Inputs for F axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 volts. |
| DI93, DI94<br>(DMC-4070 through DMC-4080 only) | Auxiliary Encoder Inputs for G axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 volts. |
| DI95, DI96<br>(DMC-4080 only) | Auxiliary Encoder Inputs for H axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 volts. |

## Power Requirements

| | |
|---|---|
| 20-80 VDC | 12-16W at 25C |

## Max Power Output

| | |
|---|---|
| +5V | 1.1A |
| +12V | 40mA |
| -12V | 40mA |

# Performance Specifications

## Minimum Servo Loop Update Time:

| | Normal | Fast Firmware |
|---|---|---|
| Minimum Servo Loop Update Time: | | |
| DMC-4010 | 62.5 μsec | 31.25 μsec |
| DMC-4020 | 62.5 μsec | 31.25 μsec |
| DMC-4030 | 125 μsec | 62.5 μsec |
| DMC-4040 | 125 μsec | 62.5 μsec |
| DMC-4050 | 156.25 μsec | 93.75 μsec |
| DMC-4060 | 156.25 μsec | 93.75 μsec |
| DMC-4070 | 187.5 μsec | 125 μsec |
| DMC-4080 | 187.5 μsec | 125 μsec |
| Position Accuracy: | +/-1 quadrature count | |
| Velocity Accuracy: | | |
| Long Term | Phase-locked, better than 0.005% | |
| Short Term | System dependent | |
| Position Range: | +/-2147483647 counts per move | |
| Velocity Range: | Up to 22,000,000 counts/sec servo; 6,000,000 pulses/sec-stepper | |
| Velocity Resolution: | 2 counts/sec | |
| Motor Command Resolution: | 16 bit or 0.0003 V | |
| Variable Range: | +/-2 billion | |
| Variable Resolution: | 1 · 10-4 | |
| Array Size: | 16000 elements, 30 arrays | |
| Program Size: | 2000 lines x 80 characters | |

# Fast Update Rate Mode

The DMC-40x0 can operate with much faster servo update rates than the default of every millisecond. This mode is known as 'fast mode' and allows the controller to operate with the following update rates:

| | |
|---|---|
| DMC-4010 | 31.25 μsec |
| DMC-4020 | 31.25 μsec |
| DMC-4030 | 62.5 μsec |
| DMC-4040 | 62.5 μsec |
| DMC-4050 | 93.75 μsec |
| DMC-4060 | 93.75 μsec |
| DMC-4070 | 125 μsec |
| DMC-4080 | 125 μsec |

In order to run the DMC-40x0 motion controller in fast mode, the fast firmware must be uploaded. This can be done through the GalilTools communication software. The fast firmware is included with the original DMC-40x0 utilities.

In order to set the desired update rates, use the command TM.

When the controller is operating with the fast firmware, the following functions are **disabled**:

Gearing mode

Ecam mode

Pole (PL)

Analog Feedback (AF)

Stepper Motor Operation (MT 2,-2,2.5,-2.5)

Trippoints in thread 2-8

Tell Velocity Interrogation Command (TV)

Aux Encoders (TD)

Dual Velocity (DV)

Peak Torque Limit (TK)

Notch Filter (NB, NF, NZ )

# Power Connectors for the DMC-40x0

## Overview

The DMC-40x0 uses Molex Pitch Mini-Fit, Jr.™ Receptacle Housing connectors for connecting DC Power to the Amplifiers, Controller, and Motors. This section gives the specifications of these connectors. For information specific to your Galil amplifier or driver, refer to the specific amplifier/driver in the Integrated Amplifiers and Drivers section.

## Molex Part Numbers Used

There are 3 different Molex connectors used with the DMC-40x0. The type of connectors on any given controller will be determined be the Amplifiers/Drivers that were ordered. Below are tables indicating the type of Molex Connectors used and the specific part numbers used on each Amplifier or Driver. For more information on the connectors, go to http://www.molex.com/.

**Note**: These part numbers list the connectors that are found on the controller. For more information see the Molex website.

| Molex Part Number | Crimp Part Number | Type |
|---|---|---|
| 39-31-0060 | 44476-3112 | 6 Position |
| 39-31-0040 | 44476-3112 | 4 Position |
| 39-31-0020 | 44476-3112 | 2 Position |

| Galil Amplifier / Driver | | Molex Part Number | Type |
|---|---|---|---|
| None | | 39-31-0020 | 2 Position |
| AMP-43040 | Power | 39-31-0060 | 6 Position |
| | Motor | 39-31-0040 | 4 Position |
| AMP-43140 | Power | 39-31-0040 | 4 Position |
| | Motor | 39-31-0020 | 2 Position |
| SDM-44040 | Power | 39-31-0060 | 6 Position |
| | Motor | 39-31-0040 | 4 Position |
| SMD-44140 | Power | 39-31-0060 | 6 Position |
| | Motor | 39-31-0040 | 4 Position |

# Connectors for ICM-42000 Interconnect Board

## ICM-42000 I/O (A-D) 44 pin HD D-Sub Connector (Female)

| Pin# | Label | Description | Pin# | Label | Description | Pin# | Label | Description |
|---|---|---|---|---|---|---|---|---|
| 1 | ERR | Error Output | 16 | RST | Reset Input | 31 | GND | Digital Ground |
| 2 | DI1 | Digital Input 1/ A latch | 17 | INCOM | Input Common | 32 | DI2 | Digital Input 2 / B latch |
| 3 | DI4 | Digital Input 4 / D latch | 18 | DI3 | Digital Input 3 / C latch | 33 | DI5 | Digital Input 5 |
| 4 | DI7 | Digital Input 7 | 19 | DI6 | Digital Input 6 | 34 | DI8 | Digital Input 8 |
| 5 | ELO | Electronic Lock Out | 20 | ABRT | Abort Input | 35 | GND | Digital Ground |
| 6 | LSCOM | Limit Switch Common | 21 | N/C | No Connect | 36 | FLSA | Forward Limit Switch A |
| 7 | HOMA | Home Switch A | 22 | RLSA | Reverse Limit Switch A | 37 | FLSB | Forward Limit Switch B |
| 8 | HOMB | Home Switch B | 23 | RLSB | Reverse Limit Switch B | 38 | FLSC | Forward Limit Switch C |
| 9 | HOMC | Home Switch C | 24 | RLSC | Reverse Limit Switch C | 39 | FLSD | Forward Limit Switch D |
| 10 | HOMD | Home Switch D | 25 | RLSD | Reverse Limit Switch D | 40 | GND | Digital Ground |
| 11 | OPWR | Output Power | 26 | N/C | No Connect | 41 | DO1 | Digital Output 1 |
| 12 | DO3 | Digital Output 3 | 27 | DO2 | Digital Output 2 | 42 | DO4 | Digital Output 4 |
| 13 | DO6 | Digital Output 6 | 28 | DO5 | Digital Output 5 | 43 | DO7 | Digital Output 7 |
| 14 | ORET | Output Return | 29 | DO8 | Digital Output 8 | 44 | CMP | Output Compare |
| 15 | +5V | +5V from Controller | 30 | +5V | +5V | | | |

## ICM-42000 DMC-40x0 I/O (E-H) 44 pin HD D-Sub Connector (Female)

**4080** For DMC-4050 thru DMC-4080 controllers only.

| Pin# | Label | Description | Pin# | Label | Description | Pin# | Label | Description |
|---|---|---|---|---|---|---|---|---|
| 1 | ERR | Error Output | 16 | RST | Reset Input | 31 | GND | Digital Ground |
| 2 | DI9 | Digital Input 9 / E latch | 17 | INCOM | Input Common | 32 | DI10 | Digital Input 10 / F latch |
| 3 | DI12 | Digital Input 12/H latch | 18 | DI11 | Digital Input 11 / G latch | 33 | DI13 | Digital Input 13 |
| 4 | DI15 | Digital Input 15 | 19 | DI14 | Digital Input 14 | 34 | DI16 | Digital Input 16 |
| 5 | ELO | Electronic Lock Out | 20 | ABRT | Abort Input | 35 | GND | Digital Ground |
| 6 | LSCOM | Limit Switch Common | 21 | N/C | No Connect | 36 | FLSE | Forward Limit Switch E |
| 7 | HOME | Home Switch E | 22 | RLSE | Reverse Limit Switch E | 37 | FLSF | Forward Limit Switch F |
| 8 | HOMF | Home Switch F | 23 | RLSF | Reverse Limit Switch F | 38 | FLSG | Forward Limit Switch G |
| 9 | HOMG | Home Switch G | 24 | RLSG | Reverse Limit Switch G | 39 | FLSH | Forward Limit Switch H |
| 10 | HOMH | Home Switch H | 25 | RLSH | Reverse Limit Switch H | 40 | GND | Digital Ground |
| 11 | OPWR | Output Power | 26 | N/C | No Connect | 41 | DO9 | Digital Output 9 |
| 12 | DO11 | Digital Output 11 | 27 | DO10 | Digital Output 10 | 42 | DO12 | Digital Output 12 |
| 13 | DO14 | Digital Output 14 | 28 | DO13 | Digital Output 13 | 43 | DO15 | Digital Output 15 |
| 14 | ORET | Output Return | 29 | DO16 | Digital Output 16 | 44 | CMP | Output Compare |
| 15 | +5V | +5V from Controller | 30 | +5V | +5V | | | |

# ICM-42000 External Driver (A-D) 44 pin HD D-Sub Connector (Male)

| Pin# | Label | Description | Pin# | Label | Description | Pin# | Label | Description |
|---|---|---|---|---|---|---|---|---|
| 1 | RES | Reserved | 16 | STPA | PWM / Step A | 31 | STPB | PWM / Step B |
| 2 | STPC | PWM / Step C | 17 | RES | Reserved | 32 | RES | Reserved |
| 3 | RES | Reserved | 18 | STPD | PWM / Step D | 33 | GND | Digital Ground |
| 4 | RES | Reserved | 19 | DIRA | Sign / Direction A | 34 | DIRB | Sign / Direction B |
| 5 | DIRC | Sign / Direction C | 20 | RES | Reserved | 35 | RES | Reserved |
| 6 | RES | Reserved | 21 | DIRD | Sign / Direction D | 36 | GND | Digital Ground |
| 7 | AENA | Amplifier Enable A | 22 | AEC1 | Amp Enable Common 1 | 37 | AENB | Amplifier Enable B |
| 8 | AEND | Amplifier Enable D | 23 | AENC | Amplifier Enable C | 38 | AEC2 | Amp Enable Common 2 |
| 9 | N/C | No Connect | 24 | N/C | No Connect | 39 | GND | Digital Ground |
| 10 | -12V | -12V from Controller | 25 | +12V | +12V from Controller | 40 | MCMA | Motor Command A |
| 11 | MCMB | Motor Command B | 26 | RES | Reserved / MCMDA_N [1] | 41 | RES | Reserved / MCMDB_N [1] |
| 12 | RES | Reserved / MCMDC_N [1] | 27 | MCMC | Motor Command C | 42 | MCMD | Motor Command D |
| 13 | N/C | No Connect | 28 | RES | Reserved / MCMDD_N [1] | 43 | GND | Digital Ground |
| 14 | N/C | No Connect | 29 | N/C | No Connect | 44 | N/C | No Connect |
| 15 | +5V | +5V from Controller | 30 | N/C | No Connect | | | |

# ICM-42000 External Driver (E-H) 44 pin HD D-Sub Connector (Male)

**4080** For DMC-4050 thru DMC-4080 controllers only.

| Pin# | Label | Description | Pin# | Label | Description | Pin# | Label | Description |
|---|---|---|---|---|---|---|---|---|
| 1 | RES | Reserved | 16 | STPE | PWM / Step E | 31 | STPF | PWM / Step F |
| 2 | STPG | PWM / Step G | 17 | RES | Reserved | 32 | RES | Reserved |
| 3 | RES | Reserved | 18 | STPH | PWM / Step H | 33 | GND | Digital Ground |
| 4 | RES | Reserved | 19 | DIRE | Sign / Direction E | 34 | DIRF | Sign / Direction F |
| 5 | DIRF | Sign / Direction F | 20 | RES | Reserved | 35 | RES | Reserved |
| 6 | RES | Reserved | 21 | DIRH | Sign / Direction H | 36 | GND | Digital Ground |
| 7 | AENE | Amplifier Enable E | 22 | AEC1 | Amp Enable Common 1 | 37 | AENF | Amplifier Enable F |
| 8 | AENH | Amplifier Enable H | 23 | AENG | Amplifier Enable G | 38 | AEC2 | Amp Enable Common 2 |
| 9 | N/C | No Connect | 24 | N/C | No Connect | 39 | GND | Digital Ground |
| 10 | -12V | -12V from Controller | 25 | +12V | +12V from Controller | 40 | MCME | Motor Command E |
| 11 | MCMF | Motor Command F | 26 | RES | Reserved / MCMDE_N [1] | 41 | RES | Reserved / MCMDF_N [1] |
| 12 | RES | Reserved / MCMDG_N [1] | 27 | MCMG | Motor Command G | 42 | MCMH | Motor Command H |
| 13 | N/C | No Connect | 28 | RES | Reserved / MCMDH_N [1] | 43 | GND | Digital Ground |
| 14 | N/C | No Connect | 29 | N/C | No Connect | 44 | N/C | No Connect |
| 15 | +5V | +5V from Controller | 30 | N/C | No Connect | | | |

**Notes:**

1 Negative differential motor command outputs when (DIFF) option is ordered on ICM. Ex *DMC-4040-C012-I000(DIFF)*. These pins may be used for other functions when (DIFF) option is not ordered.

## ICM-42000 Encoder 15 pin HD D-Sub Connector (Female)

| Pin # | Label | Description |
|---|---|---|
| 1 | MI+ | Index Pulse |
| 2 | MB+ | B+ Main Encoder Input |
| 3 | MA+ | A+ Main Encoder Input |
| 4 | AB+ | B+ Aux Encoder Input |
| 5 | GND | Digital Ground |
| 6 | MI- | /Index Pulse |
| 7 | MB- | B- Main Encoder Input |
| 8 | MA- | A- Main Encoder Input |
| 9 | AA- | A- Aux Encoder Input |
| 10 | HALA | A Channel Hall Sensor |
| 11 | AA+ | A+ Aux Encoder Input |
| 12 | AB- | B- Aux Encoder Input |
| 13 | HALB | B Channel Hall Sensor |
| 14 | HALC | C Channel Hall Sensor |
| 15 | +5V | +5V from Controller |

## ICM-42000 Analog 15 pin D-sub Connector (Male)

| Pin # | Label | Description |
|---|---|---|
| 1 | AGND | Analog Ground |
| 2 | AI1 | Analog Input 1 |
| 3 | AI3 | Analog Input 3 |
| 4 | AI5 | Analog Input 5 |
| 5 | AI7 | Analog Input 7 |
| 6 | AGND | Analog Ground |
| 7 | -12V | -12V from Controller |
| 8 | +5V | +5V from Controller |
| 9 | AGND | Analog Ground |
| 10 | AI2 | Analog Input 2 |
| 11 | AI4 | Analog Input 4 |
| 12 | AI6 | Analog Input 6 |
| 13 | AI8 | Analog Input 8 |
| 14 | N/C | No Connect |
| 15 | +12V | +12V from Controller |

# Connectors for ICM-42200 Interconnect Board

## ICM-42200 I/O (A-D) 44 pin HD D-Sub Connector (Female)

| Pin# | Label | Description | Pin# | Label | Description | Pin# | Label | Description |
|---|---|---|---|---|---|---|---|---|
| 1 | ERR | Error Output | 16 | RST | Reset Input | 31 | GND | Digital Ground |
| 2 | DI1 | Digital Input 1/ A latch | 17 | INCOM | Input Common | 32 | DI2 | Digital Input 2 / B latch |
| 3 | DI4 | Digital Input 4 / D latch | 18 | DI3 | Digital Input 3 / C latch | 33 | DI5 | Digital Input 5 |
| 4 | DI7 | Digital Input 7 | 19 | DI6 | Digital Input 6 | 34 | DI8 | Digital Input 8 |
| 5 | ELO | Electronic Lock Out | 20 | ABRT | Abort Input | 35 | GND | Digital Ground |
| 6 | LSCOM | Limit Switch Common | 21 | N/C | No Connect | 36 | FLSA | Forward Limit Switch A |
| 7 | HOMA | Home Switch A | 22 | RLSA | Reverse Limit Switch A | 37 | FLSB | Forward Limit Switch B |
| 8 | HOMB | Home Switch B | 23 | RLSB | Reverse Limit Switch B | 38 | FLSC | Forward Limit Switch C |
| 9 | HOMC | Home Switch C | 24 | RLSC | Reverse Limit Switch C | 39 | FLSD | Forward Limit Switch D |
| 10 | HOMD | Home Switch D | 25 | RLSD | Reverse Limit Switch D | 40 | GND | Digital Ground |
| 11 | OPWR | Output Power | 26 | N/C | No Connect | 41 | DO1 | Digital Output 1 |
| 12 | DO3 | Digital Output 3 | 27 | DO2 | Digital Output 2 | 42 | DO4 | Digital Output 4 |
| 13 | DO6 | Digital Output 6 | 28 | DO5 | Digital Output 5 | 43 | DO7 | Digital Output 7 |
| 14 | ORET | Output Return | 29 | DO8 | Digital Output 8 | 44 | CMP | Output Compare |
| 15 | +5V | +5V | 30 | +5V | +5V | | | |

## ICM-42200 DMC-40x0 I/O (E-H) 44 pin HD D-Sub Connector (Female)

**4080** For DMC-4050 thru DMC-4080 controllers only.

| Pin# | Label | Description | Pin# | Label | Description | Pin# | Label | Description |
|---|---|---|---|---|---|---|---|---|
| 1 | ERR | Error Output | 16 | RST | Reset Input | 31 | GND | Digital Ground |
| 2 | DI9 | Digital Input 9 / E latch | 17 | INCOM | Input Common | 32 | DI10 | Digital Input 10 / F latch |
| 3 | DI12 | Digital Input 12/H latch | 18 | DI11 | Digital Input 11 / G latch | 33 | DI13 | Digital Input 13 |
| 4 | DI15 | Digital Input 15 | 19 | DI14 | Digital Input 14 | 34 | DI16 | Digital Input 16 |
| 5 | ELO | Electronic Lock Out | 20 | ABRT | Abort Input | 35 | GND | Digital Ground |
| 6 | LSCOM | Limit Switch Common | 21 | N/C | No Connect | 36 | FLSE | Forward Limit Switch E |
| 7 | HOME | Home Switch E | 22 | RLSE | Reverse Limit Switch E | 37 | FLSF | Forward Limit Switch F |
| 8 | HOMF | Home Switch F | 23 | RLSF | Reverse Limit Switch F | 38 | FLSG | Forward Limit Switch G |
| 9 | HOMG | Home Switch G | 24 | RLSG | Reverse Limit Switch G | 39 | FLSH | Forward Limit Switch H |
| 10 | HOMH | Home Switch H | 25 | RLSH | Reverse Limit Switch H | 40 | GND | Digital Ground |
| 11 | OPWR | Output Power | 26 | N/C | No Connect | 41 | DO9 | Digital Output 9 |
| 12 | DO11 | Digital Output 11 | 27 | DO10 | Digital Output 10 | 42 | DO12 | Digital Output 12 |
| 13 | DO14 | Digital Output 14 | 28 | DO13 | Digital Output 13 | 43 | DO15 | Digital Output 15 |
| 14 | ORET | Output Return | 29 | DO16 | Digital Output 16 | 44 | CMP | Output Compare |
| 15 | +5V | +5V | 30 | +5V | +5V | | | |

## ICM-42200 Encoder 26 pin HD D-Sub Connector (Female)

| Pin # | Label | Description | Pin # | Label | Description |
|---|---|---|---|---|---|
| 1 | RES | Reserved [1] | 14 | FLS | Forward Limit Switch Input |
| 2 | AEN | Amplifier Enable | 15 | AB+ | B+ Aux Encoder Input |
| 3 | DIR | Direction | 16 | MI- | /Index Pulse Input |
| 4 | HOM | Home | 17 | MB+ | B+ Main Encoder Input |
| 5 | LSCOM | Limit Switch Common | 18 | GND | Digital Ground |
| 6 | AA- | A- Aux Encoder Input | 19 | MCMD | Motor Command |
| 7 | MI+ | Index Pulse Input | 20 | ENBL+ | Amp Enable Power |
| 8 | MA- | A- Main Encoder Input | 21 | RES | Reserved [3] |
| 9 | +5V | +5V From Controller | 22 | RLS | Reverse Limit Switch Input |
| 10 | GND | Digital Ground | 23 | AB- | B- Aux Encoder Input |
| 11 | ENBL- | Amp Enable Return | 24 | AA+ | A+ Aux Encoder Input |
| 12 | RES | Reserved [2] | 25 | MB- | B- Main Encoder Input |
| 13 | STP | PWM/Step | 26 | MA+ | A+ Main Encoder Input |

## ICM-42200 Analog 15 pin D-sub Connector (Male)

| Pin # | Label | Description |
|---|---|---|
| 1 | AGND | Analog Ground |
| 2 | AI1 | Analog Input 1 |
| 3 | AI3 | Analog Input 3 |
| 4 | AI5 | Analog Input 5 |
| 5 | AI7 | Analog Input 7 |
| 6 | AGND | Analog Ground |
| 7 | -12V | -12V from Controller |
| 8 | +5V | +5V from Controller |
| 9 | AGND | Analog Ground |
| 10 | AI2 | Analog Input 2 |
| 11 | AI4 | Analog Input 4 |
| 12 | AI6 | Analog Input 6 |
| 13 | AI8 | Analog Input 8 |
| 14 | N/C | No Connect |
| 15 | +12V | +12V from Controller |

**Notes:**

1 Negative differential motor command output when (DIFF) option is ordered on ICM. Ex *DMC-4040-C012-I200(DIFF)*.
Hall Input 2 when ordered with internal amplifier AMP-43040. Ex *DMC-4040-C012-I200-D3040*
Connected to GND in standard configuration.

2 Hall Input 1 when ordered with internal amplifier AMP-43040. Ex *DMC-4040-C012-I200-D3040*
Connected to GND in standard configuration.

3 Hall Input 0 when ordered with internal amplifier AMP-43040. Ex *DMC-4040-C012-I200-D3040*
Connected to GND in standard configuration.

# Connectors for CMB-41012 Interconnect Board

## CMB-41012 Extended I/O 44 pin HD D-Sub Connector (Male)

| Pin# | Label | Description | Pin# | Label | Description | Pin# | Label | Description |
|---|---|---|---|---|---|---|---|---|
| 1 | IO18 | Configurable I/O bit 18 | 16 | IO17 | Configurable I/O bit 17 | 31 | IO19 | Configurable I/O bit 19 |
| 2 | IO21 | Configurable I/O bit 21 | 17 | IO20 | Configurable I/O bit 20 | 32 | IO22 | Configurable I/O bit 22 |
| 3 | IO24 | Configurable I/O bit 24 | 18 | IO23 | Configurable I/O bit 23 | 33 | GND | Digital Ground |
| 4 | IO26 | Configurable I/O bit 26 | 19 | IO25 | Configurable I/O bit 25 | 34 | IO27 | Configurable I/O bit 27 |
| 5 | IO29 | Configurable I/O bit 29 | 20 | IO28 | Configurable I/O bit 28 | 35 | IO30 | Configurable I/O bit 30 |
| 6 | IO32 | Configurable I/O bit 32 | 21 | IO31 | Configurable I/O bit 31 | 36 | GND | Digital Ground |
| 7 | IO33 | Configurable I/O bit 33 | 22 | N/C | No Connect | 37 | IO34 | Configurable I/O bit 34 |
| 8 | IO36 | Configurable I/O bit 36 | 23 | IO35 | Configurable I/O bit 35 | 38 | N/C | No Connect |
| 9 | IO38 | Configurable I/O bit 38 | 24 | IO37 | Configurable I/O bit 37 | 39 | GND | Digital Ground |
| 10 | N/C | No Connect | 25 | N/C | No Connect | 40 | IO39 | Configurable I/O bit 39 |
| 11 | IO41 | Configurable I/O bit 41 | 26 | IO40 | Configurable I/O bit 40 | 41 | IO42 | Configurable I/O bit 42 |
| 12 | IO44 | Configurable I/O bit 44 | 27 | IO43 | Configurable I/O bit 43 | 42 | IO45 | Configurable I/O bit 45 |
| 13 | IO47 | Configurable I/O bit 47 | 28 | IO46 | Configurable I/O bit 46 | 43 | GND | Digital Ground |
| 14 | N/C | No Connect | 29 | IO48 | Configurable I/O bit 48 | 44 | N/C | No Connect |
| 15 | RES | Reserved [1] | 30 | +3.3V | +3.3V [2] | | | |

## RS-232-Main Port (Male)

Standard connector and cable, 9Pin

| Pin | Signal |
|---|---|
| 1 | NC |
| 2 | TXD |
| 3 | RXD |
| 4 | NC |
| 5 | GND |
| 6 | NC |
| 7 | CTS |
| 8 | RTS |
| 9 | NC |

**Notes:**

1 5V when (5V) option is ordered on CMB. Ex *DMC-4040-C012(5V)-I200*

2 Reserved when (5V) option is ordered on CMB

### Baud Rate Jumper Settings

| 19.2 | 38.4 | BAUD RATE |
|---|---|---|
| ON | ON | 9600 |
| ON | OFF | 19200 |
| OFF | ON | 38400 |
| OFF | OFF | 115200 |

## RS-232-Auxiliary Port (Female)

Standard connector and cable, 9Pin

| Pin # | Signal |
|---|---|
| 1 | NC |
| 2 | TXD |
| 3 | RXD |
| 4 | NC |
| 5 | GND |
| 6 | NC |
| 7 | RTS |
| 8 | CTS |
| 9 | NC (5V with APWR Jumper) |

## RS-422-Main Port (Non-Standard Option)

Standard connector and cable when DMC-40x0 is ordered with RS-422 Option.

| Pin # | Signal |
|---|---|
| 1 | RTS- |
| 2 | TXD- |
| 3 | RXD- |
| 4 | CTS- |
| 5 | GND |
| 6 | RTS+ |
| 7 | TXD+ |
| 8 | RXD+ |
| 9 | CTS+ |

## RS-422-Auxiliary Port (Non-Standard Option)

Standard connector and cable when DMC-40x0 is ordered with RS-422 Option.

| Pin # | Signal |
|---|---|
| 1 | CTS- |
| 2 | RXD- |
| 3 | TXD- |
| 4 | RTS- |
| 5 | GND |
| 6 | CTS+ |
| 7 | RXD+ |
| 8 | TXD+ |
| 9 | RTS+ |

## Ethernet

100 BASE-T/10 BASE-T - Kycon GS-NS-88-3.5

| Pin # | Signal |
|---|---|
| 1 | TXP |
| 2 | TXN |
| 3 | RXP |
| 4 | NC |
| 5 | NC |
| 6 | RXN |
| 7 | NC |
| 8 | NC |

10 BASE-2- AMP 227161-7

10 BASE-F- HP HFBR-1414 (TX, Transmitter)

HP HFBR-2416 (RX, Receiver)

# Jumper Description for ICM-42000 and CMB-41012

| Jumper | Label | Function (If jumpered) |
|---|---|---|
| Communications | OPT | Reserved |
| | MO | When controller is powered on or reset, Amplifier Enable lines will be in a Motor Off state. A SH will be required to re-enable the motors. |
| | 38.4K | Baud Rate setting – see table above |
| | 19.2K | Baud Rate setting – see table above |
| | UPGD | Used to upgrade controller firmware when resident firmware is corrupt. |
| | MRST | Master Reset enable. Returns controller to factory default settings and erases EEPROM. Requires power-on or RESET to be activated. |
| Amplifier Enable | GND | Connect AECOM1 or AECOM2 to Digital Ground |
| | +5V | Connect AECOM1 or AECOM2 to Controller +5V |
| | +12V | Connect AECOM1 or AECOM2 to Controller +12V |
| | AEC1 | Connect AECOM1 to AEC1 pin on External Driver D-Sub |
| | AEC2 | Connect AECOM2 to AEC2 pin on External Driver D-Sub |

# Cable Connections for DMC-40x0

The DMC-40x0 requires the transmit, receive, and ground for slow communication rates. (i.e. 9600 baud) For faster rates the handshake lines are required. The connection tables below contain the handshake lines.

## Standard RS-232 Specifications

### 25 pin Serial Connector (Male, D-type)

This table describes the pinout for standard serial ports found on most computers.

| Pin # | Function |
|---|---|
| 1 | NC |
| 2 | TXD |
| 3 | RXD |
| 4 | RTS |
| 5 | CTS |
| 6 | DSR |
| 7 | GND |
| 8 | DCD |
| 9 | NC |
| 10 | NC |
| 11 | NC |
| 12 | NC |
| 13 | NC |
| 14 | NC |
| 15 | NC |
| 16 | NC |
| 17 | NC |
| 18 | NC |
| 19 | NC |
| 20 | DTR |
| 21 | NC |
| 22 | RI |
| 23 | NC |
| 24 | NC |
| 25 | NC |

### 9 Pin Serial Connector (Male, D-type)

Standard serial port connections found on most computers.

| Pin # | Function |
|---|---|
| 1 | DCD |
| 2 | RXD |
| 3 | TXD |
| 4 | RTS |
| 5 | GND |
| 6 | DSR |
| 7 | RTS |
| 8 | CTS |
| 9 | RI |

## DMC-40x0 Serial Cable Specifications

### Cable to Connect Computer 25 pin to Main Serial Port

| 25 Pin (Male - computer) | 9 Pin (female - controller) |
|---|---|
| 5 CTS | 8 RTS |
| 3 RXD | 2 TXD |
| 2 TXD | 3 RXD |
| 4 RTS | 7 CTS |
| 7 GND | 5 GND |

### Cable to Connect Computer 9 pin to Main Serial Port Cable (9 pin)

| 9 Pin (FEMALE - Computer) | 9 Pin (FEMALE - Controller) |
|---|---|
| 2 RXD | 2 TXD |
| 3 TXD | 3 RXD |
| 5 GND | 5 GND |
| 7 RTS | 7 CTS |
| 8 CTS | 8 RTS |

### Cable to Connect Computer 25 pin to Auxiliary Serial Port Cable (9 pin)

| 25 Pin (Male - terminal) | 9 Pin (male - controller) |
|---|---|
| 4 RTS | 8 CTS |
| 2 TXD | 2 RXD |
| 3 RXD | 3 TXD |
| 5 CTS | 7 RTS |
| 7 GND | 5 GND |
| Computer +5V | 9 (Jumper APWR if required) |

### Cable to Connect Computer 9 pin to Auxiliary Serial Port Cable (9 pin)

| 9 Pin (FEMALE - terminal) | 9 Pin (MALE - Controller) |
|---|---|
| 4 RTS | 8 CTS |
| 3 TXD | 2 RXD |
| 2 RXD | 3 TXD |
| 1 CTS | 7 RTS |
| 5 GND | 5 GND |
| Controller +5V | 9 (Jumper APWR if required) |

# Pin-Out Description for DMC-40x0

### Outputs

| | |
|---|---|
| Motor Command | +/- 10 Volt range signal for driving amplifier. In servo mode, motor command output is updated at the controller sample rate. In the motor off mode, this output is held at the OF command level. |
| Amplifier Enable | Signal to disable and enable an amplifier. Amp Enable goes low on Abort and OE1. |
| PWM / Step | PWM/STEP OUT is used for directly driving power bridges for DC servo motors or for driving step motor amplifiers. For servo motors: If you are using a conventional amplifier that accepts a +/-10 Volt analog signal, this pin is not used and should be left open. The PWM output is available in two formats: Inverter and Sign Magnitude. In the Inverter mode, the PWM (64kHz) signal is .2% duty cycle for full negative voltage, 50% for 0 Voltage and 99.8% for full positive voltage (64kHz Switching Frequency). In the Sign Magnitude Mode (MT1.5), the PWM (128 kHz) signal is 0% for 0 Voltage, 99.6% for full voltage and the sign of the Motor Command is available at the sign output (128kHz Switching Frequency). |
| PWM / Step | For stepper motors: The STEP OUT pin produces a series of pulses for input to a step motor driver. The pulses may either be low or high. The pulse width is 50%. |
| Sign / Direction | Used with PWM signal to give the sign of the motor command for servo amplifiers or direction for step motors. |
| Error | The signal goes low when the position error on any axis exceeds the value specified by the error limit command, ER. |
| Output 1-Output 8<br>Output 9-Output 16<br>(DMC-4050 thru 4080) | The high power optically isolated outputs are uncommitted and may be designated by the user to toggle relays and trigger external events. The output lines are toggled by Set Bit, SB, and Clear Bit, CB, instructions. The OP instruction is used to define the state of all the bits of the Output port. |

## Inputs

| | |
|---|---|
| Encoder, MA+, MB+ | Position feedback from incremental encoder with two channels in quadrature, CHA and CHB. The encoder may be analog or TTL. Any resolution encoder may be used as long as the maximum frequency does not exceed 22,000,000 quadrature states/sec. The controller performs quadrature decoding of the encoder signals resulting in a resolution of quadrature counts (4 x encoder cycles). Note: Encoders that produce outputs in the format of pulses and direction may also be used by inputting the pulses into CHA and direction into Channel B and using the CE command to configure this mode. |
| Encoder Index, MI+ | Once-Per-Revolution encoder pulse. Used in Homing sequence or Find Index command to define home on an encoder index. |
| Encoder, MA-, MB-, MI- | Differential inputs from encoder. May be input along with CHA, CHB for noise immunity of encoder signals. The CHA- and CHB- inputs are optional. |
| Auxiliary Encoder, AA+, AB+, Aux A-, Aux B- | Inputs for additional encoder. Used when an encoder on both the motor and the load is required. Not available on axes configured for step motors. |
| Abort | A low input stops commanded motion instantly without a controlled deceleration. Also aborts motion program. |
| Reset | A low input resets the state of the processor to its power-on condition. The previously saved state of the controller, along with parameter values, and saved sequences are restored. |
| Electronic Lock Out | Input that when triggered will shut down the amplifiers at a hardware level. Useful for safety applications where amplifiers must be shut down at a hardware level. |
| Forward Limit Switch | When active, inhibits motion in forward direction. Also causes execution of limit switch subroutine, #LIMSWI. The polarity of the limit switch may be set with the CN command. |
| Reverse Limit Switch | When active, inhibits motion in reverse direction. Also causes execution of limit switch subroutine, #LIMSWI. The polarity of the limit switch may be set with the CN command. |
| Home Switch | Input for Homing (HM) and Find Edge (FE) instructions. Upon BG following HM or FE, the motor accelerates to slew speed. A transition on this input will cause the motor to decelerate to a stop. The polarity of the Home Switch may be set with the CN command. |
| Input 1 - Input 8 isolated<br>Input 9 - Input 16 isolated | Uncommitted inputs. May be defined by the user to trigger events. Inputs are checked with the Conditional Jump instruction and After Input instruction or Input Interrupt. Input 1 is latch X, Input 2 is latch Y, Input 3 is latch Z and Input 4 is latch W if the high speed position latch function is enabled. |
| Latch | High speed position latch to capture axis position on occurrence of latch signal. AL command arms latch. Input 1 is latch X, Input 2 is latch Y, Input 3 is latch Z and Input 4 is latch W. Input 9 is latch E, input 10 is latch F, input 11 is latch G, input 12 is latch H. |

# Configuring the Amplifier Enable Circuit

## ICM-42000 and ICM-42100

The following section details the steps needed to change the amplifier enable configuration for the DMC-40x0 controller with an ICM-42000 or ICM-42100. For detailed instruction on changing the amplifier enable configuration on a DMC-40x0 with an ICM-42200 see the section in Chapter 3 labeled ICM-42200 Amplifier Enable Configuration. For electrical details about the amplifier enable circuit, see the ICM-42000 and ICM-42100 Amplifier Enable Circuit section in Chapter 3.

For DMC-4080 refer to DMC-4080 (Steps 1 and 2) section below.

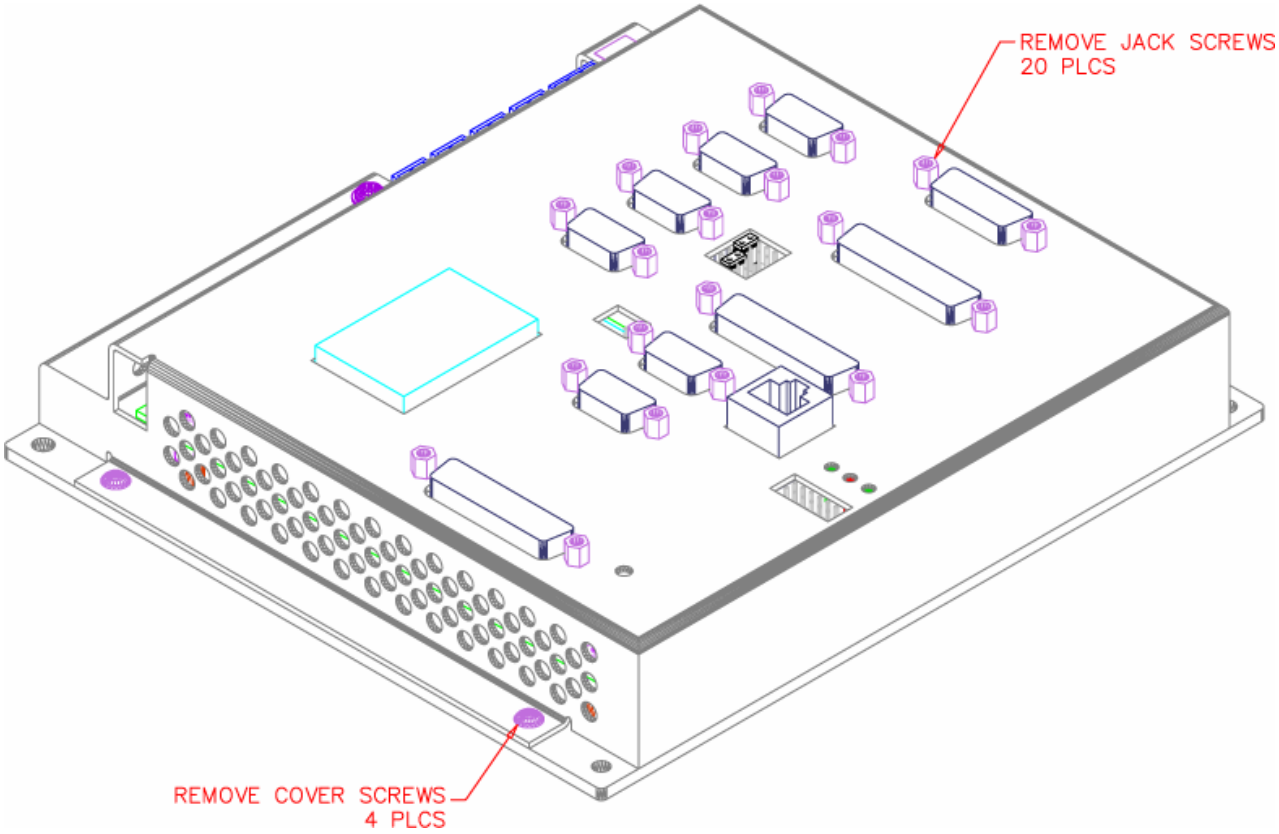
**NOTE:** From the default configuration, the configuration for +12V High Amp Enable Sinking Configuration does not require the remove of the metal cover. This can be achieved by simply changing the jumpers.

## DMC-4040 (Steps 1 and 2)

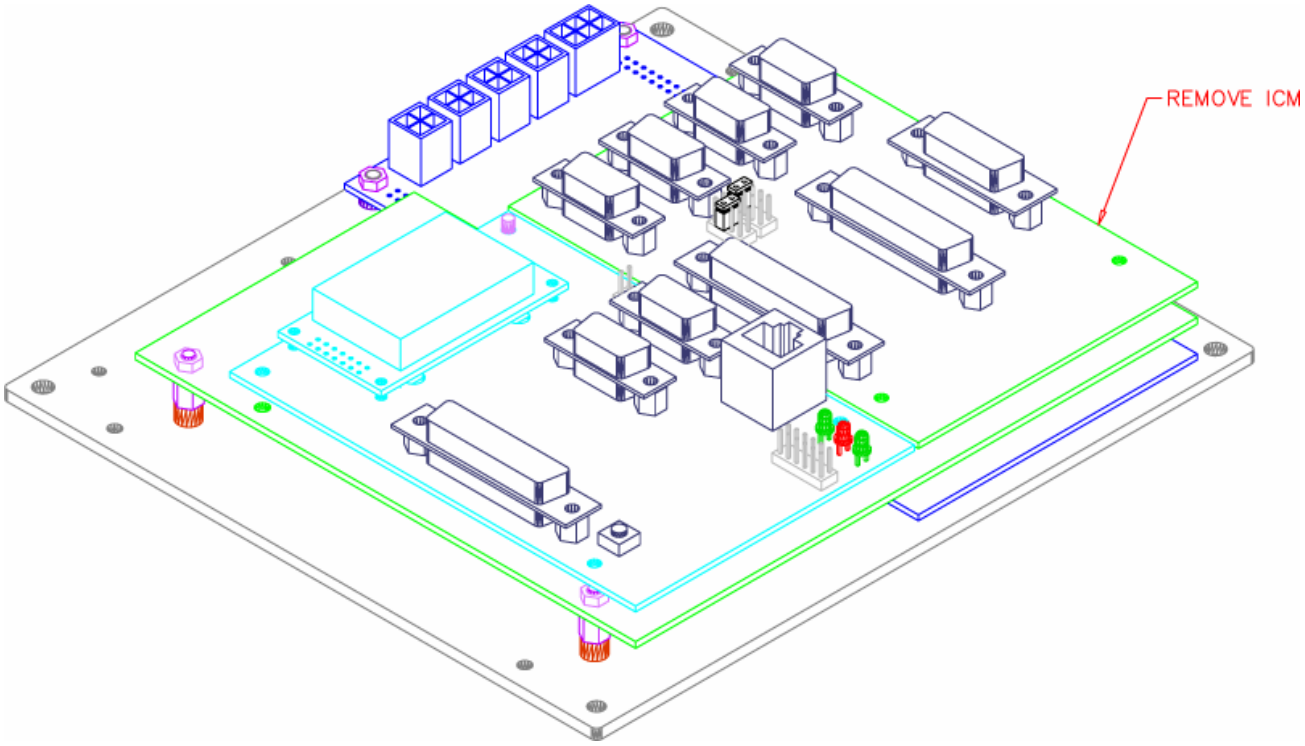
### Step 1: Remove Cover

Notes:

1. Cover Removal:
   - A. Remove Jack Screws (20 Places)
   - B. Remove #6-32x3/16" Button Head Cover Screws (4 Places)
2. Lift Cover Straight Up and Away from Unit.



### Step 2: Remove ICM



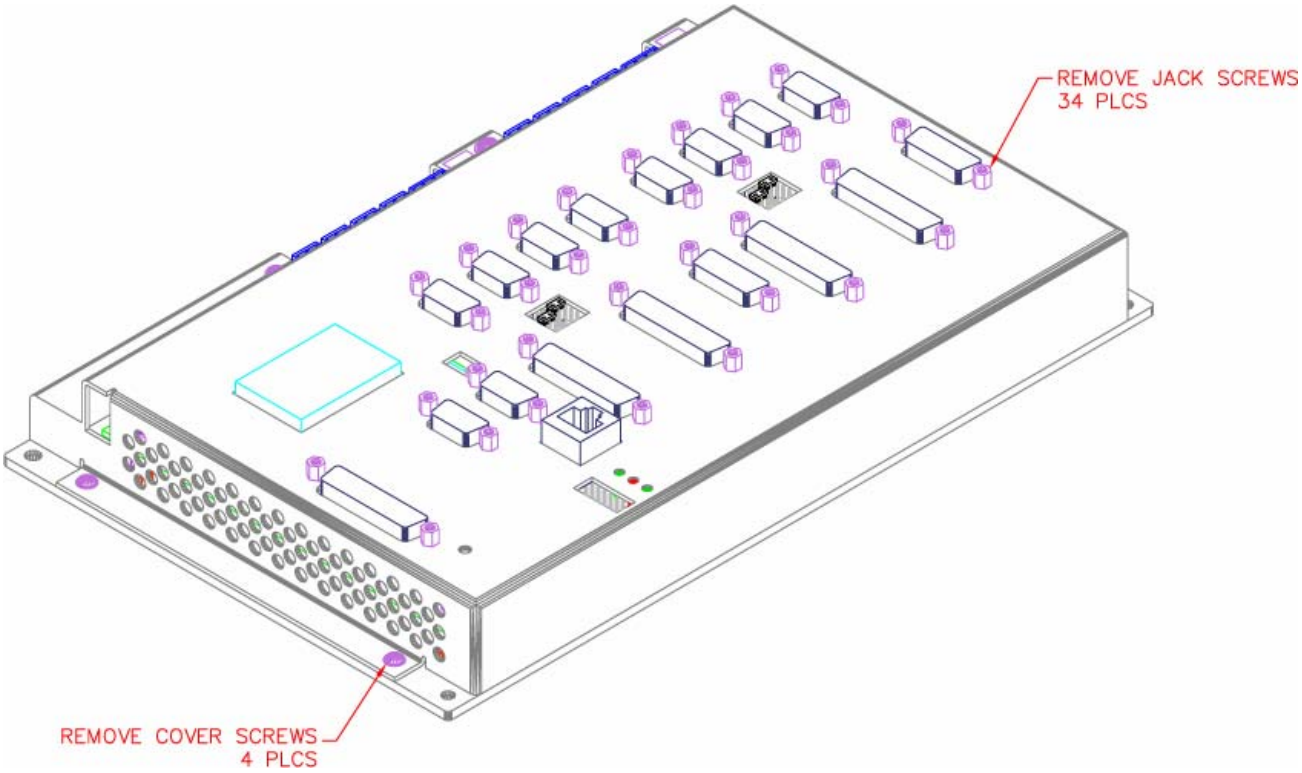

For DMC-4040 – Proceed to Step 3: Configure Circuit

# DMC-4080 (Steps 1 and 2)

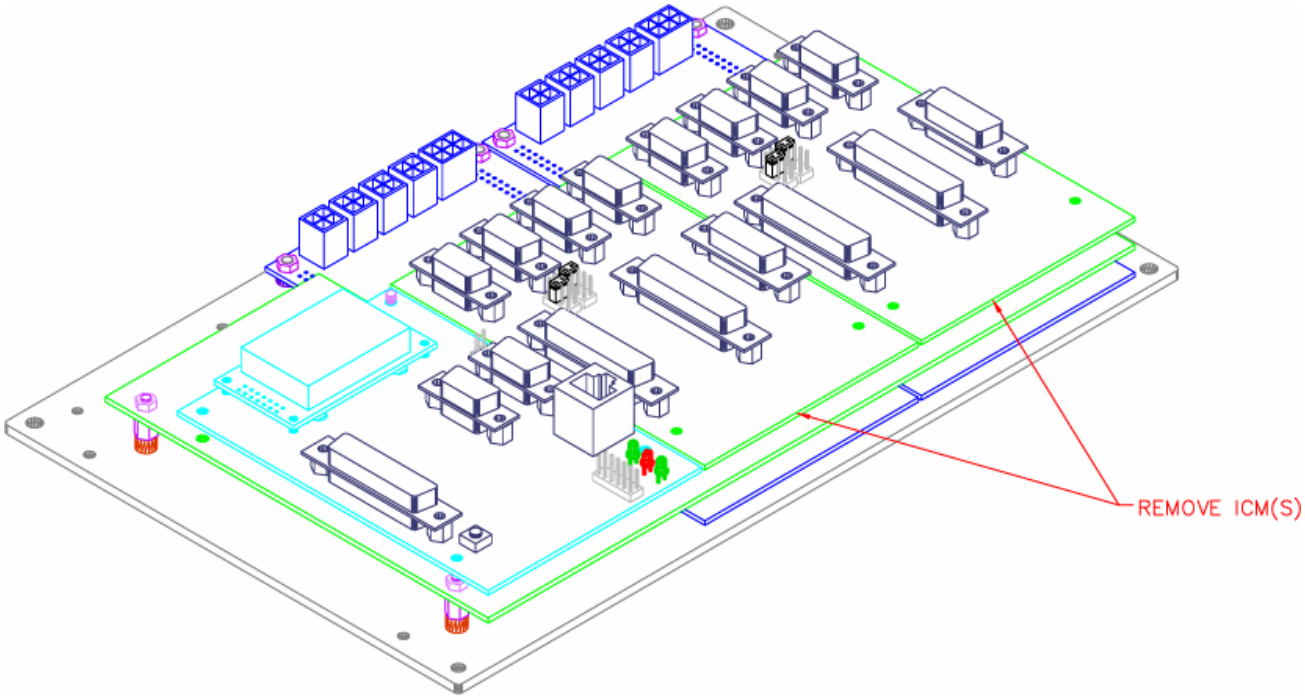
## Step 1: Remove Cover

Notes:

1. Cover Removal:
   A. Remove Jack Screws (34 Places)
   B. Remove #6-32x3/16" Button Head Cover Screws (4 Places)
2. Lift Cover Straight Up and Away from Unit.



### Step 2: Remove ICM(s)

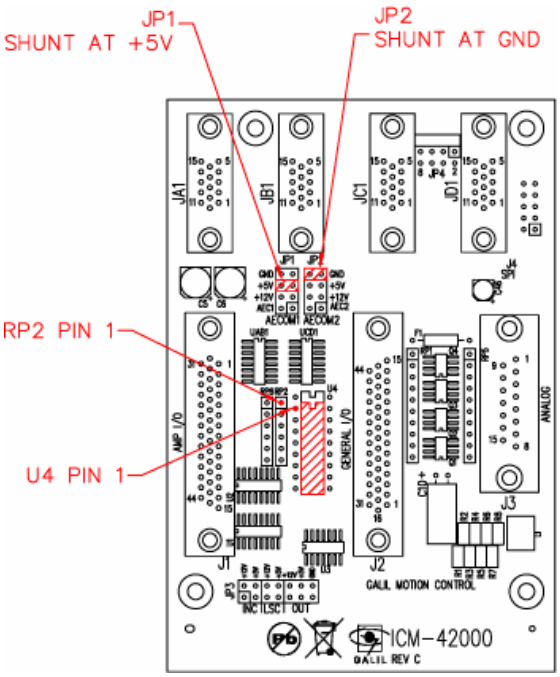


## DMC-4040 and DMC-4080 (Step 3)

### Step 3: Configure Circuit

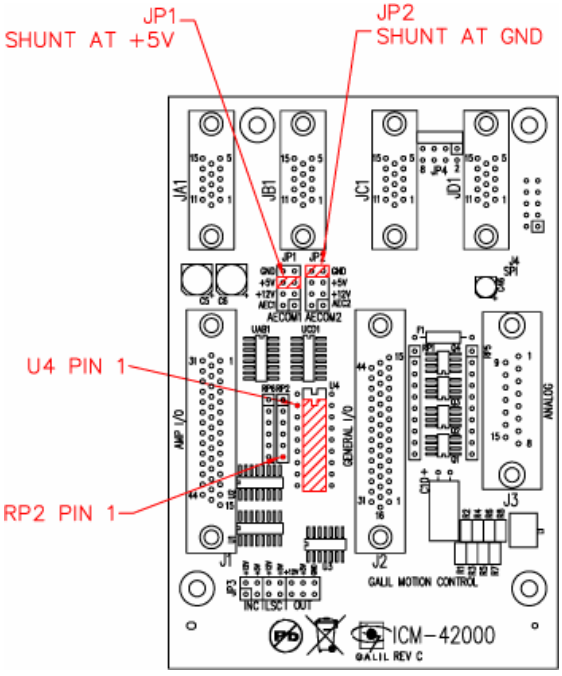
Reference the instructions below for the desired configuration, and then proceed to Step 4.

## +5V High Amp Enable Sinking Configuration (Default)




Default Configuration Shipped with controller when no specific setup is ordered.

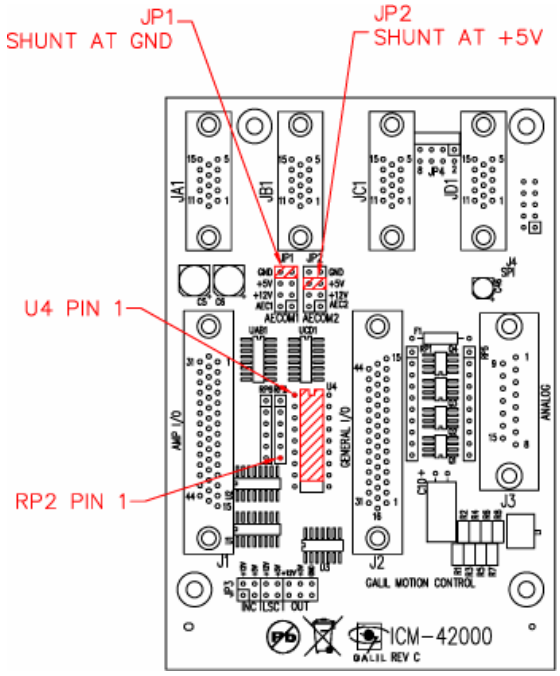
## +5V Low Amp Enable Sinking Configuration




From Default Configuration:

1. Reverse RP2

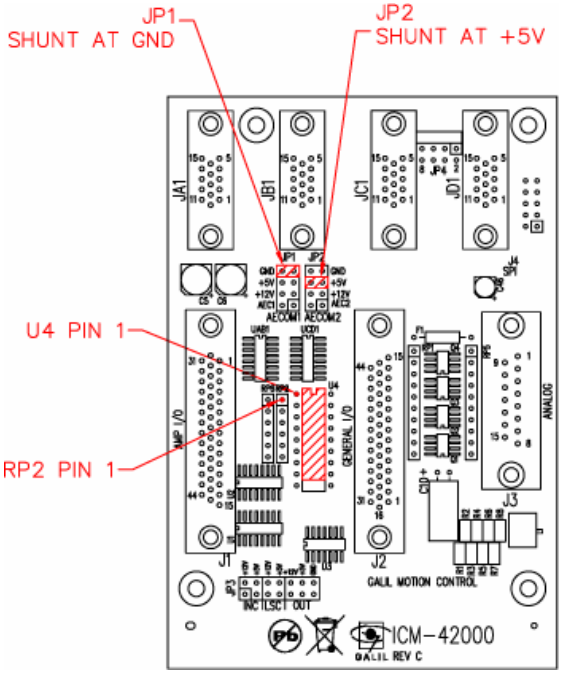
## +5V High Amp Enable Sourcing Configuration




From Default Configuration:

1. Move U4 up one pin location on socket
2. Reverse RP2
3. Change JP1 to GND
4. Change JP2 to +5V

## +5V Low Amp Enable Sourcing Configuration

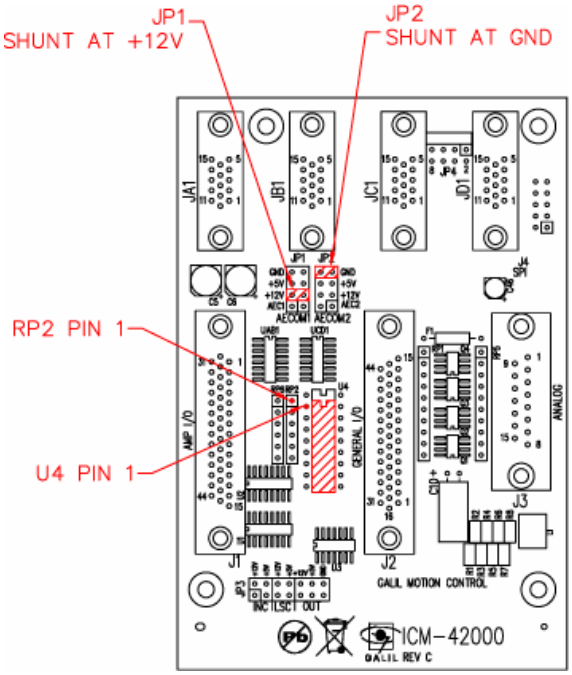



From Default Configuration:

1. Move U4 up one pin location on socket
2. Change JP1 to GND
3. Change JP2 to +5V

## +12V High Amp Enable Sinking Configuration

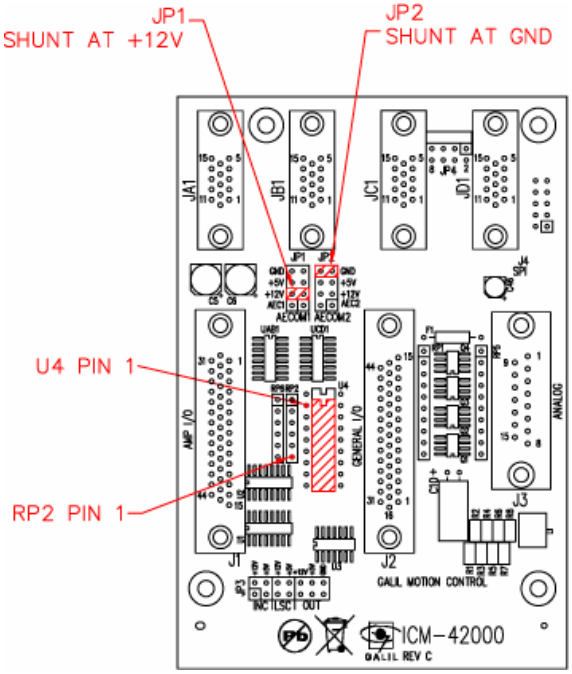
(Does not require the removal of Metal)




From Default Configuration:

1. Change JP1 to +12V

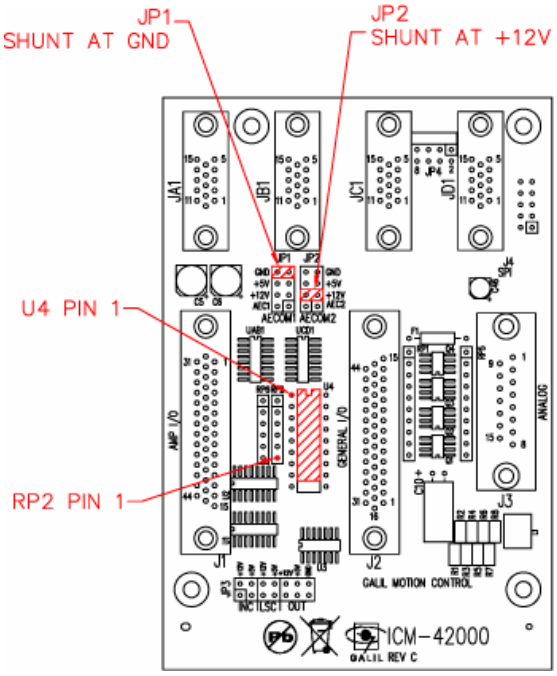
## +12V Low Amp Enable Sinking Configuration




From Default Configuration:

1. Reverse RP2
2. Change JP1 to +12V

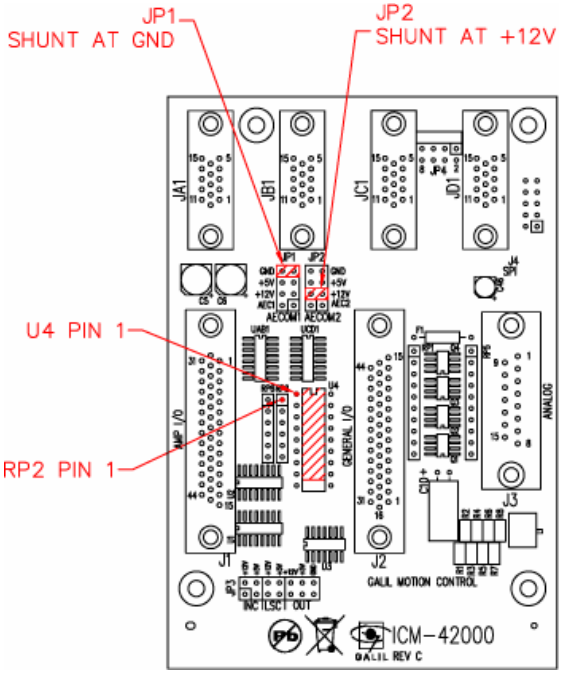
## +12V High Amp Enable Sourcing Configuration




From Default Configuration:

1. Move U4 up one pin location on socket
2. Reverse RP2
3. Change JP1 to GND
4. Change JP2 to +12V

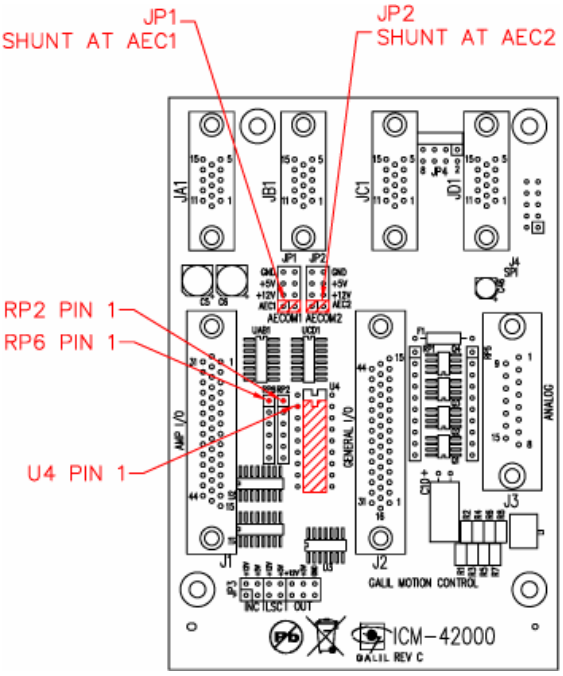
## +12V Low Amp Enable Sourcing Configuration




From Default Configuration:

1. Move U4 up one pin location on socket
2. Change JP1 to GND
3. Change JP2 to +12V

## Isolated Power High Amp Enable Sinking Configuration

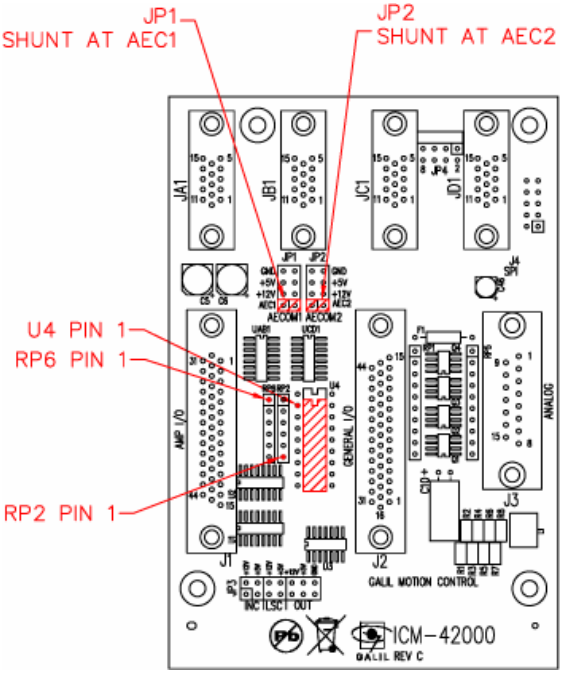



```
AEC1 = V+
AEC2 = V-
For +5V to +12V, RP6 = 820 Ohms
For +13V to +24V, RP6 = 4.7K Ohms
```

From Default Configuration:

1. Change JP1 to AEC1
2. Change JP2 to AEC2
3. If AEC1 is +13V to +24V, Replace RP6 with 4.7K Resistor Pack

## Isolated Power Low Amp Enable Sinking Configuration

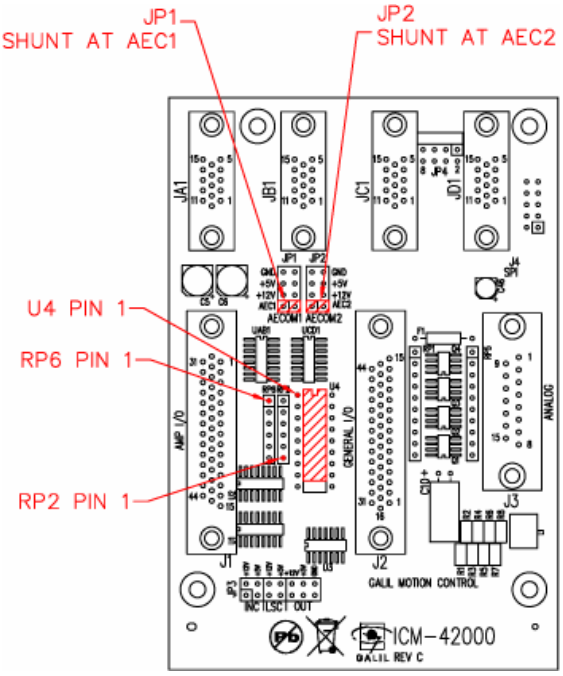



```
AEC1 = V+
AEC2 = V-
For +5V to +12V, RP6 = 820 Ohms
For +13V to +24V, RP6 = 4.7K Ohms
```

From Default Configuration:

1. Reverse RP2
2. Change JP1 to AEC1
3. Change JP2 to AEC2
4. If AEC1 is +13V to +24V, Replace RP6 with 4.7K Resistor Pack

## Isolated Power High Amp Enable Sourcing Configuration

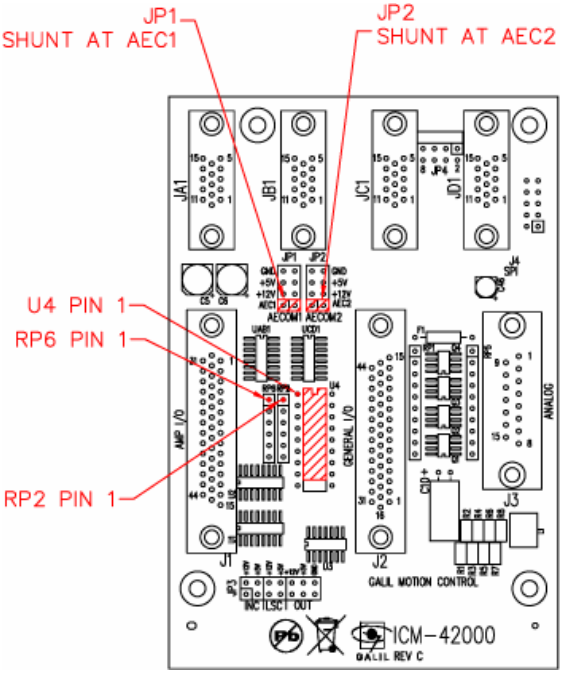



```
AEC1 = V-
AEC2 = V+
For +5V to +12V, RP6 = 820 Ohms
For +13V to +24V, RP6 = 4.7K Ohms
```

From Default Configuration:

1. Move U4 up one pin location on socket
2. Reverse RP2
3. Change JP1 to AEC1
4. Change JP2 to AEC2
5. If AEC2 is +13V to +24V, Replace RP6 with 4.7K Resistor Pack

## Isolated Power Low Amp Enable Sourcing Configuration




```
AEC1 = V-
AEC2 = V+
For +5V to +12V, RP6 = 820 Ohms
For +13V to +24V, RP6 = 4.7K Ohms
```

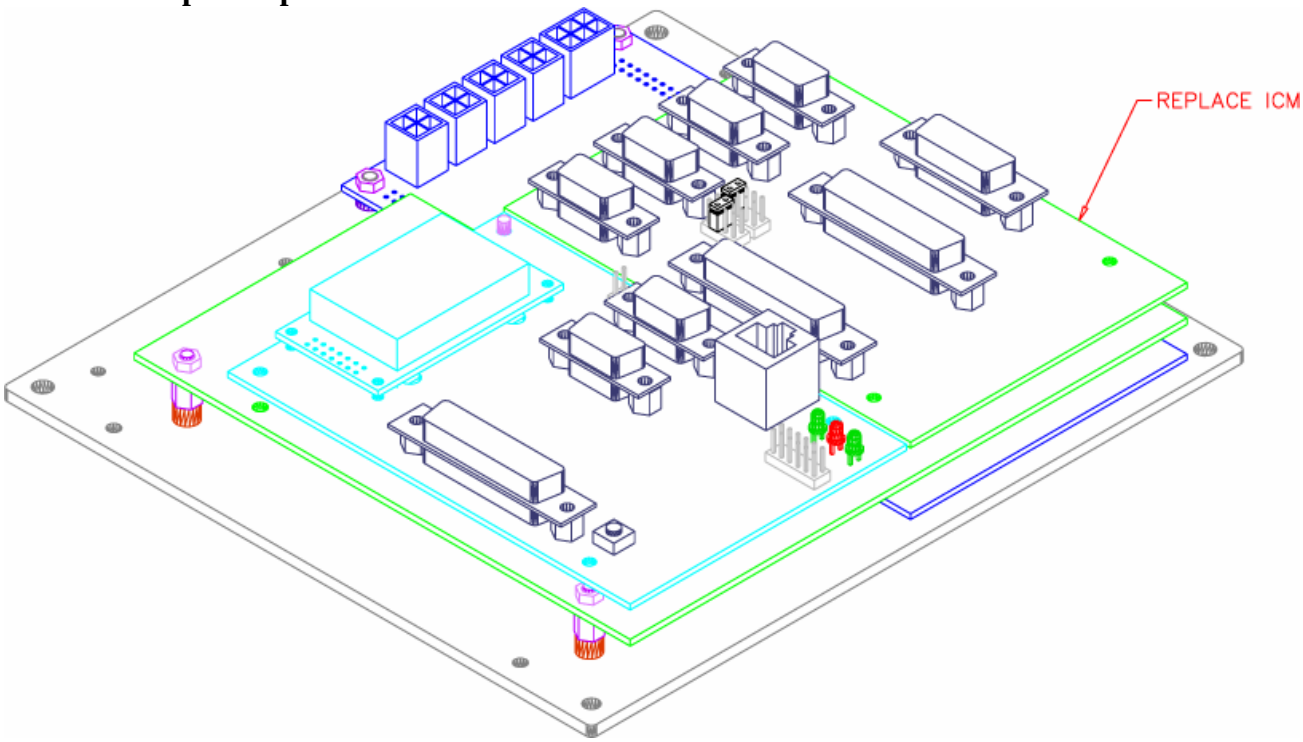
From Default Configuration:

1. Move U4 up one pin location on socket
2. Change JP1 to AEC1
3. Change JP2 to AEC2
4. If AEC2 is +13V to +24V, Replace RP6 with 4.7K Resistor Pack

For Steps 4 and 5 with a DMC-4080 refer to DMC-4080 (Steps 4 and 5) section below.

## DMC-4040 (Steps 4 and 5)

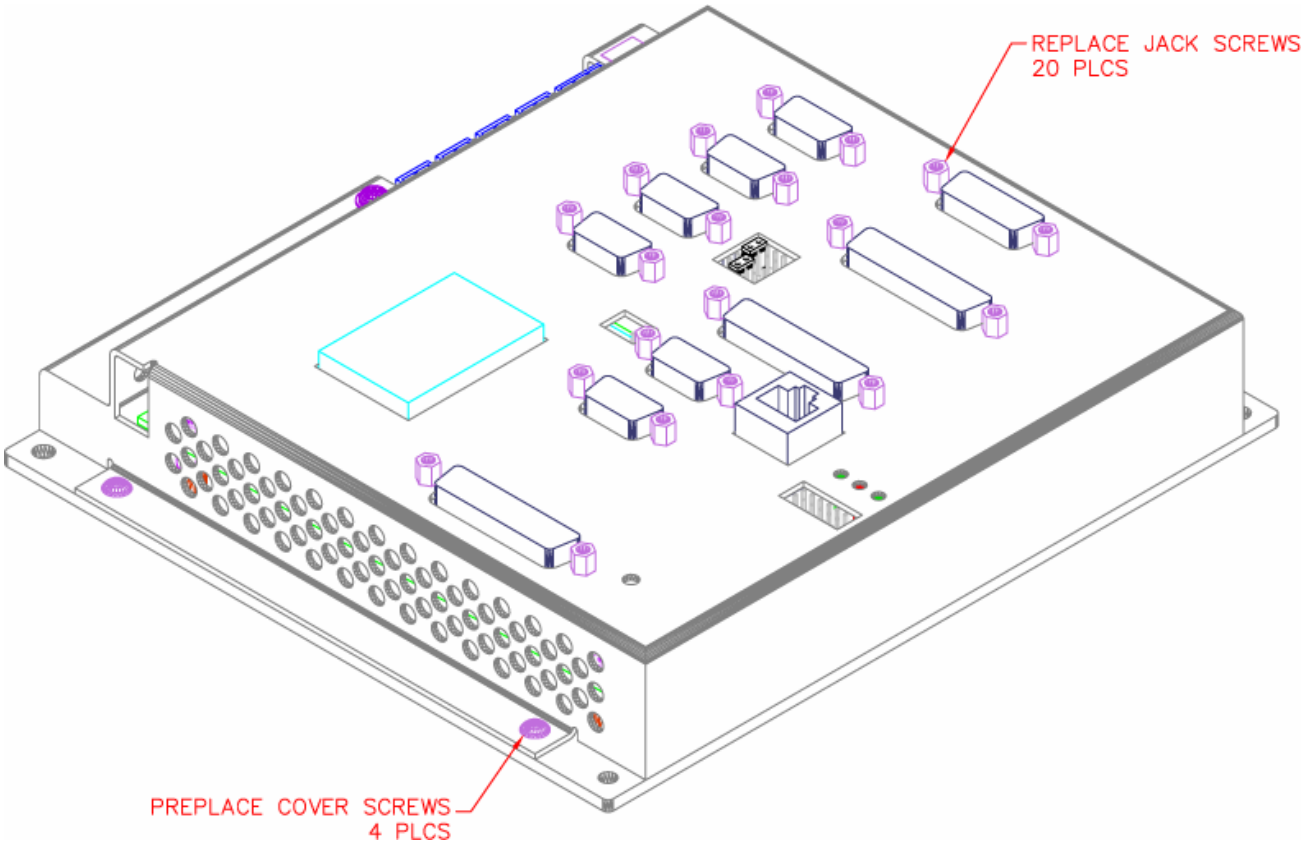
### Step 4: Replace ICM



## Step 5: Replace Cover

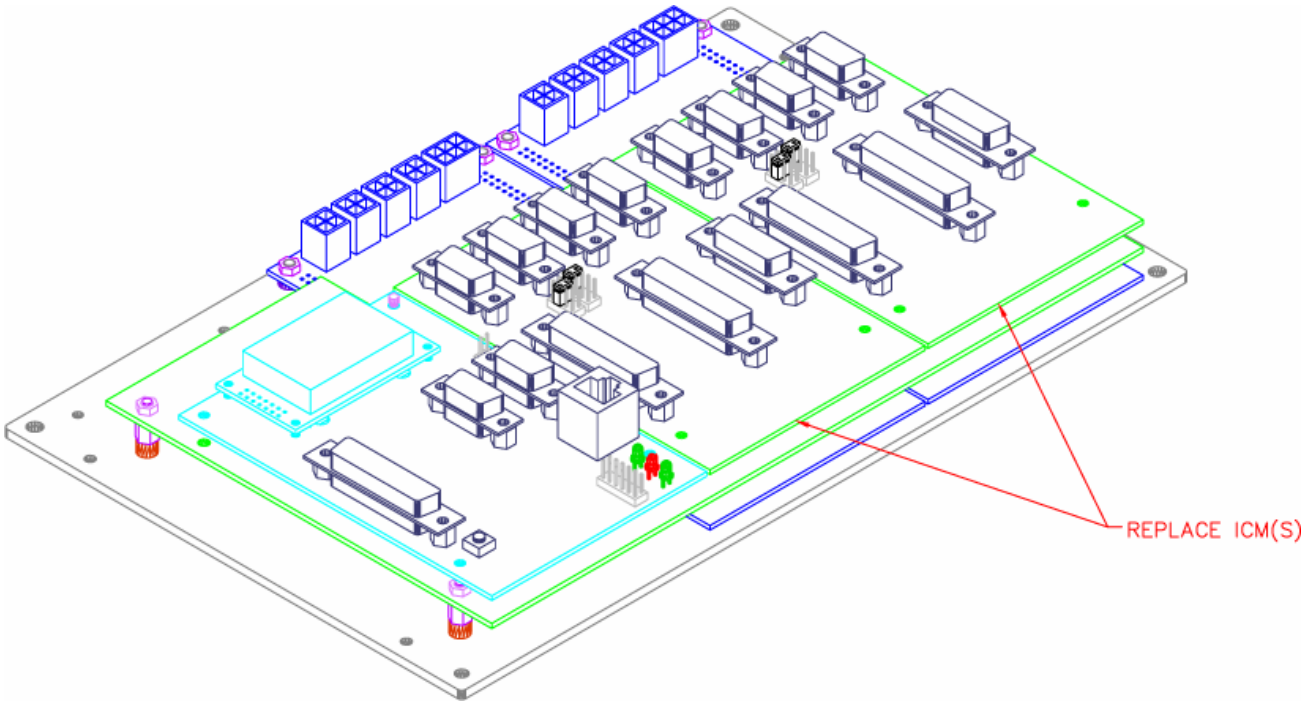
Notes:

1. Cover Installation:
   - A. Install Jack Screws (20 Places)
   - B. Install #6-32x3/16” Button Head Cover Screws(4 Places)



## DMC-4080 (Steps 4 and 5)

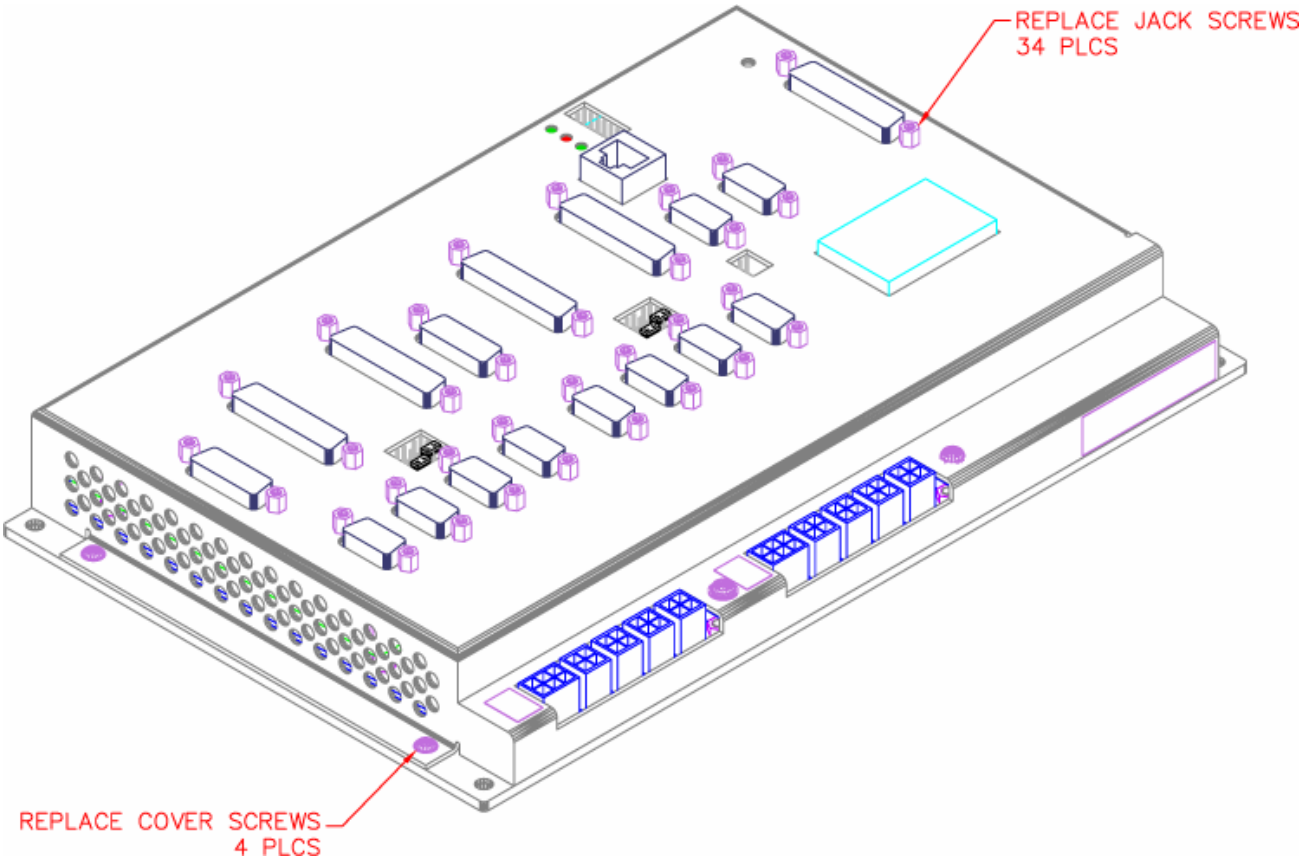
### Step 4: Replace ICM(s)



### Step 5: Replace Cover

Notes:

1. Cover Installation:
   - A. Install Jack Screws (34 Places)
   - B. Install #6-32x3/16” Button Head Cover Screws(4 Places)



# Coordinated Motion - Mathematical Analysis

The terms of coordinated motion are best explained in terms of the vector motion. The vector velocity, Vs, which is also known as the feed rate, is the vector sum of the velocities along the X and Y axes, Vx and Vy.

$$Vs = \sqrt{Vx^2 + Vy^2}$$

The vector distance is the integral of Vs, or the total distance traveled along the path. To illustrate this further, suppose that a string was placed along the path in the X-Y plane. The length of that string represents the distance traveled by the vector motion.

The vector velocity is specified independently of the path to allow continuous motion. The path is specified as a collection of segments. For the purpose of specifying the path, define a special X-Y coordinate system whose origin is the starting point of the sequence. Each linear segment is specified by the X-Y coordinate of the final point expressed in units of resolution, and each circular arc is defined by the arc radius, the starting angle, and the angular width of the arc. The zero angle corresponds to the positive direction of the X-axis and the CCW direction of rotation is positive. Angles are expressed in degrees, and the resolution is 1/256th of a degree. For example, the path shown in Fig. A.1 is specified by the instructions:

```
VP                  0,10000
CR                  10000, 180, -90
VP                  20000, 20000
```

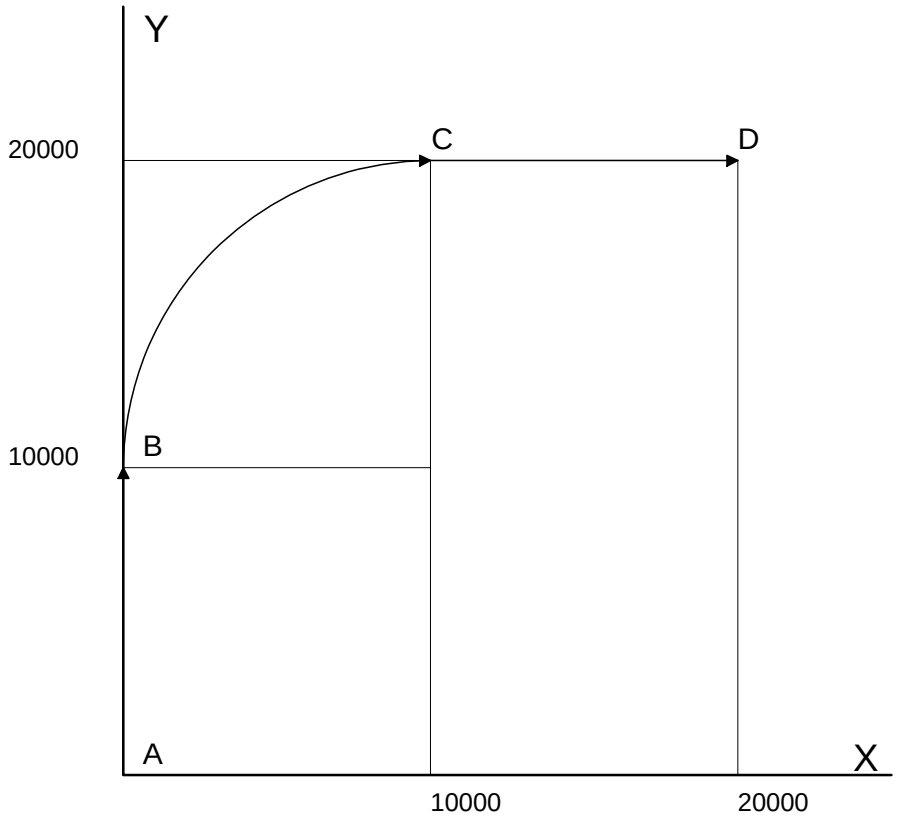



*Figure A..1 - X-Y Motion Path*

The first line describes the straight line vector segment between points A and B. The next segment is a circular arc, which starts at an angle of 180° and traverses -90°. Finally, the third line describes the linear segment between points C and D. Note that the total length of the motion consists of the segments:

| | | |
|---|---|---|
| A-B | Linear | 10000 units |
| B-C | Circular | $\frac{R\lvert\Delta\theta\rvert 2\pi}{360} = 15708$ |
| C-D | Linear | 10000 |
| | Total | 35708 counts |

In general, the length of each linear segment is

$$L_k = \sqrt{Xk^2 + Yk^2}$$

Where Xk and Yk are the changes in X and Y positions along the linear segment. The length of the circular arc is

$$L_k = R_k\lvert\Delta\Theta_k\rvert 2\pi/360$$

The total travel distance is given by

$$D = \sum_{k=1}^{n} L_k$$

The velocity profile may be specified independently in terms of the vector velocity and acceleration.

For example, the velocity profile corresponding to the path of Fig. A.2 may be specified in terms of the vector speed and acceleration.

```
VS    100000
VA    2000000
```

The resulting vector velocity is shown in Fig. A.2.

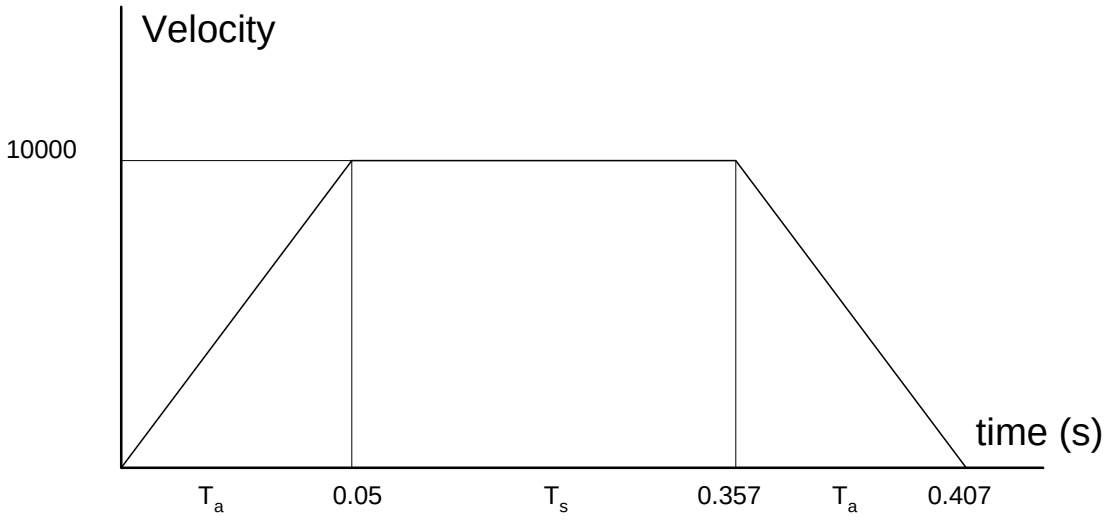



*Figure A..2 - Vector Velocity Profile*

The acceleration time, Ta, is given by

$$T_a = \frac{VS}{VA} = \frac{100000}{2000000} = 0.05s$$

The slew time, Ts, is given by

$$T_s = \frac{D}{VS} - T_a = \frac{35708}{100000} - 0.05 = 0.307s$$

The total motion time, Tt, is given by:

$$T_t = \frac{D}{VS} + T_a = 0.407s$$

The velocities along the X and Y axes are such that the direction of motion follows the specified path, yet the vector velocity fits the vector speed and acceleration requirements.

For example, the velocities along the X and Y axes for the path shown in Fig. A.1 are given in Fig. A.3.

Fig. A.3a shows the vector velocity. It also indicates the position point along the path starting at A and ending at D. Between the points A and B, the motion is along the Y axis. Therefore,

```
Vy = Vs
```

and

```
Vx = 0
```

Between the points B and C, the velocities vary gradually and finally, between the points C and D, the motion is in the X direction.

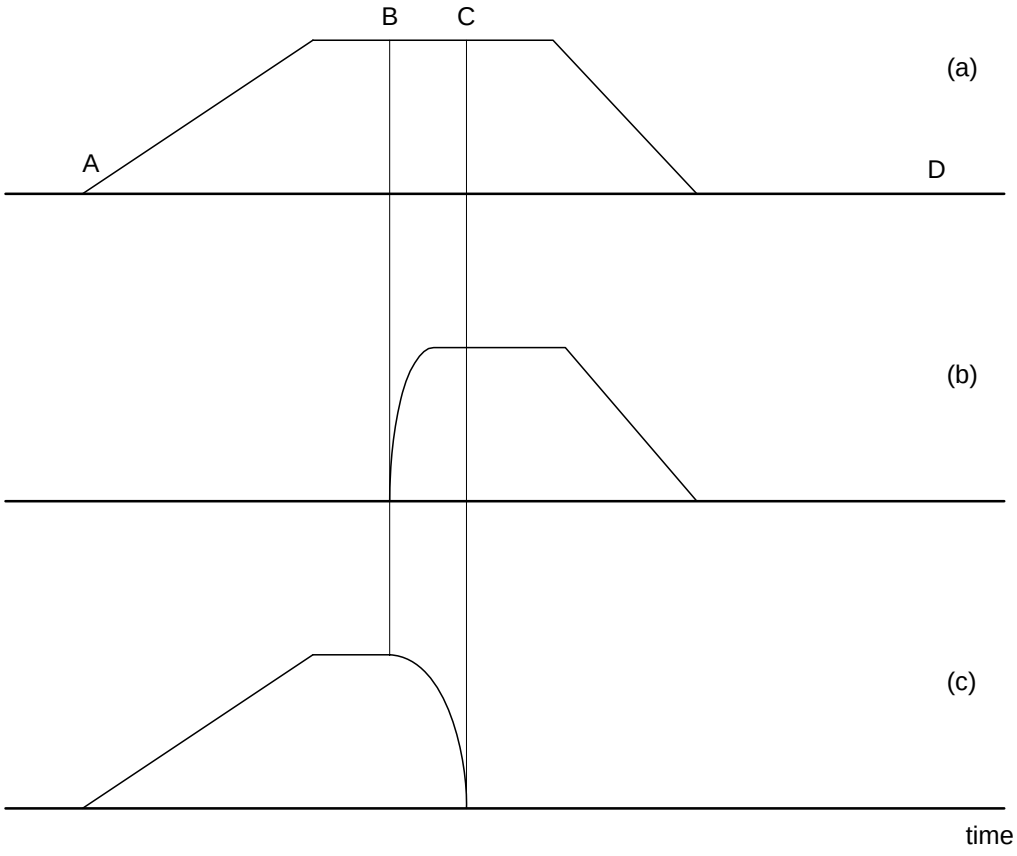



*Figure A.3 - Vector and Axes Velocities*

# Example- Communicating with OPTO-22 SNAP-B3000-ENET

Controller is connected to OPTO-22 via handle F. The OPTO-22's IP address is 131.29.50.30. The Rack has the following configuration:

| | |
|---|---|
| Digital Inputs | Module 1 |
| Digital Outputs | Module 2 |
| Analog Outputs (+/-10V) | Module 3 |
| Analog Inputs (+/-10V) | Module 4 |

| Instruction | Interpretation |
|---|---|
| #CONFIG | Label |
| IHF=131,29,50,30<502>2 | Establish connection |
| WT10 | Wait 10 milliseconds |
| JP #CFGERR,_IHF2=0 | Jump to subroutine |
| JS #CFGDOUT | Configure digital outputs |
| JS #CFGAOUT | Configure analog outputs |
| JS #CFGAIN | Configure analog inputs |
| MBF = 6,6,1025,1 | Save configuration to OPTO-22 |
| EN | End |
| | |
| #CFGDOUT | Label |
| MODULE=2 | Set variable |
| CFGVALUE=$180 | Set variable |
| NUMOFIO=4 | Set variable |
| JP #CFGJOIN | Jump to subroutine |
| | |
| #CFGAOUT | Label |
| MODULE=3 | Set variable |
| CFGVALUE=$A7 | Set variable |
| NUMOFIO=2 | Set variable |
| JP #CFGJOIN | Jump to subroutine |
| | |
| #CFGAIN | Label |
| MODULE=5 | Set variable |
| CFGVALUE=12 | Set variable |
| NUMOFIO=2 | Set variable |
| JP#CFGJOIN | Jump to subroutine |
| | |
| #CFGJOIN | Label |
| DM A[8] | Dimension array |
| I=0 | Set variable |
| #CFGLOOP | Loop subroutine |
| A[I]=0 | Set array element |
| I=I+1 | Increment |
| A[I]=CFGVALUE | Set array element |
| I=I+1 | Increment |

```
JP #CFGLOOP,I<(2*NUMOFIO)     Conditional statement
MBF=6,16,632+(MODULE*8),NUM   Configure I/O using Modbus function code 16 where
OFIO*2,A[]                    the starting register is 632+(MODULE*8), number
                              of registers is NUMOFIO*2 and A[] contains the
                              data.
EN                            end

#CFERR                        Label
MG"UNABLE TO ESTABLISH        Message
CONNECTION"
EN                            End
```

Using the equation:

I/O number = (Handlenum*1000) + ((Module-1)*4) + (Bitnum-1)

`MG @IN[6001]` display level of input at handle 6, module 1, bit 2

`SB 6006` set bit of output at handle 6, module 2, bit 3

or to one

`OB 6006,1`

`AO 608,3.6` set analog output at handle 6, module 53, bit 1 to 3.6 volts

`MG @AN[6017]` display voltage value of analog input at handle6, module 5, bit 2

# DMC-40x0/DMC-2200 Comparison

| BENEFIT | DMC-40x0 | DMC-2200 |
|---|---|---|
| Higher servo bandwidth | Up to 32kHz update rate | Up to 8kHz update rate |
| Faster Processing power | ~10X faster than DMC-20x0 | |
| Increased Program Storage | 2000lines x 80 Characters | 1000 lines x 80 characters |
| Increased Array Storage | 16000 array elements in 30 arrays | 8000 array elements in 30 arrays |
| Increased Variable Storage | 510 Variables | 254 labels |
| Faster servo operation – good for very high resolution sensors | 22 MHz encoder speed for servos | 12 MHz |
| Faster stepper operation | 6 MHz stepper rate | 3 MHz |
| Improved EMI | Connections broken out through D-Sub connectors and high power amps run away from control lines. | 100-pin high density connector and cable. |
| Contour Buffer | 511 elements | 1 element |
| New commands/features | ^L^K, PW, %, OV, OT, OA, ALTX, TR1,1, HV, LD, M axis, _EY, ZA, OE2, TM scaling, _NO, LC1000, MT1.5, #POSERR, #LIMSWI, #MCTIME, and #ININT run without a thread | |
| Removed Commands | VT, WC | |

# List of Other Publications

"Step by Step Design of Motion Control Systems"

by Dr. Jacob Tal

"Motion Control Applications"

by Dr. Jacob Tal

"Motion Control by Microprocessors"

by Dr. Jacob Tal

# Training Seminars

Galil, a leader in motion control with over 500,000 controllers working worldwide, has a proud reputation for anticipating and setting the trends in motion control. Galil understands your need to keep abreast with these trends in order to remain resourceful and competitive. Through a series of seminars and workshops held over the past 20 years, Galil has actively shared their market insights in a no-nonsense way for a world of engineers on the move. In fact, over 10,000 engineers have attended Galil seminars. The tradition continues with three different seminars, each designed for your particular skill set-from beginner to the most advanced.

**MOTION CONTROL MADE EASY**

WHO SHOULD ATTEND

Those who need a basic introduction or refresher on how to successfully implement servo motion control systems.

TIME: 4 hours (8:30 am-12:30 pm)

**ADVANCED MOTION CONTROL**

WHO SHOULD ATTEND

Those who consider themselves a "servo specialist" and require an in-depth knowledge of motion control systems to ensure outstanding controller performance. Also, prior completion of “Motion Control Made Easy" or equivalent is required. Analysis and design tools as well as several design examples will be provided.

TIME: 8 hours (8:00 am-5:00 pm)

**PRODUCT WORKSHOP**

WHO SHOULD ATTEND

Current users of Galil motion controllers. Conducted at Galil’s headquarters in Rocklin, CA, students will gain detailed understanding about connecting systems elements, system tuning and motion programming. This is a “hands-on” seminar and students can test their application on actual hardware and review it with Galil specialists.

Attendees must have a current application and recently purchased a Galil controller to attend this course.

TIME: Two days (8:30-4:30pm)

# Contacting Us

**Galil Motion Control**

270 Technology Way

Rocklin, CA 95765

Phone:  916-626-0101

Fax:  916-626-0102

E-Mail Address: support@galilmc.com

URL: http://galilmc.com/

FTP: http://galilmc.com/ftp/

# WARRANTY

All controllers manufactured by Galil Motion Control are warranted against defects in materials and workmanship for a period of 18 months after shipment. Motors, and Power supplies are warranted for 1 year. Extended warranties are available.

In the event of any defects in materials or workmanship, Galil Motion Control will, at its sole option, repair or replace the defective product covered by this warranty without charge. To obtain warranty service, the defective product must be returned within 30 days of the expiration of the applicable warranty period to Galil Motion Control, properly packaged and with transportation and insurance prepaid. We will reship at our expense only to destinations in the United States and for products within warranty.

Call Galil to receive a Return Materials Authorization (RMA) number prior to returning product to Galil.

Any defect in materials or workmanship determined by Galil Motion Control to be attributable to customer alteration, modification, negligence or misuse is not covered by this warranty.

EXCEPT AS SET FORTH ABOVE, GALIL MOTION CONTROL WILL MAKE NO WARRANTIES EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO SUCH PRODUCTS, AND SHALL NOT BE LIABLE OR RESPONSIBLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES.

# Integrated Amplifiers and Drivers

## Overview

### A1 – AMP-43040 (-D3040)

The AMP-43040 (four-axis) and AMP-43020 (two-axis) are multi-axis brush/brushless amplifiers that are capable of handling 500 watts of continuous power per axis. The AMP-43040/43020 Brushless drive modules are connected to a DMC-40x0. The standard amplifier accepts DC supply voltages from 18-80 VDC.

### A2 – AMP-43140 (-D3140)

The AMP-43140 contains four linear drives for operating small brush-type servo motors. The AMP-43140 requires a ± 12–30 DC Volt input. Output power is 20 W per amplifier or 60 W total. The gain of each transconductance linear amplifier is 0.1 A/V at 1 A maximum current. The typical current loop bandwidth is 4 kHz.

### A3 – SDM-44040 (-D4040)

The SDM-44040 is a stepper driver module capable of driving up to four bipolar two-phase stepper motors. The current is selectable with options of 0.5, 0.75, 1.0, and 1.4 Amps/Phase. The step resolution is selectable with options of full, half, 1/4 and 1/16.

### A4 – SDM-44140 (-D4140)

The SDM-44140 microstepper module drives four bipolar two-phase stepper motors with 1/64 microstep resolution (the SDM-44140 drives two). The current is selectable with options of 0.5, 1.0, 2.0, & 3.0 Amps per axis.

# A1 – AMP-43040

## Introduction

The AMP-43040 (four-axis) and AMP-43020 (two-axis) are multi-axis brush/brushless amplifiers that are capable of handling 500 watts of continuous power per axis. The AMP-43040/43020 Brushless drive modules are connected to a DMC-40x0. The standard amplifier accepts DC supply voltages from 18-80 VDC. If higher voltages are required, please contact Galil.


*Figure A1-1 – DMC-4040-C012-I000-D3040 (DMC-4040 with AMP-43040)*

# Electrical Specifications

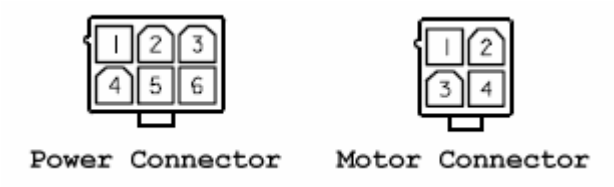
The amplifier is a brush/brushless trans-conductance PWM amplifier. The amplifier operates in torque mode, and will output a motor current proportional to the command signal input.

| | |
|---|---|
| Supply Voltage: | 18-80 VDC |
| Continuous Current: | 7 Amps |
| Peak Current | 10 Amps |
| Nominal Amplifier Gain | 0.7 Amps/Volt |
| Switching Frequency | 60 kHz (up to 140 kHz available-contact Galil) |
| Minimum Load Inductance: | 0.5 mH (Inverter mode) 0.2 mH (Chopper Mode) |
| Brushless Motor Commutation angle | 120° (60° option available) |

## Mating Connectors

| | On Board Connector | Terminal Pins |
|---|---|---|
| **POWER** | 6-pin MATE-N-LOK<br>MOLEX# 39-31-0060 | MOLEX#44476-3112 |
| **A,B,C,D: 4-pin Motor Power Connectors** | 4-pin MATE-N-LOK<br>MOLEX# 39-31-0040 | MOLEX#44476-3112 |

For mating connectors see http://www.molex.com/




| **Power Connector** | |
|---|---|
| Pin Number | Connection |
| 1,2,3 | DC Power Supply Ground |
| 4,5,6 | +VS (DC Power) |
| **Motor Connector** | |
| 1 | Phase C (N/C for Bushed Motors) |
| 2 | Phase B |
| 3 | No Connect |
| 4 | Phase A |

# Operation

## Brushless Motor Setup

**Note**: If you purchased a Galil motor with the amplifier, it is ready for use. No additional setup is necessary.

To begin the setup of the brushless motor and amplifier, it is first necessary to have communications with the motion controller. Refer to the user manual supplied with your controller for questions regarding controller communications. It is also necessary to have the motor hardware connected and the amplifier powered to begin the setup phase. After the encoders and motor leads are connected, the controller and amplifier need to be configured correctly in software. Take all appropriate safety precautions. For example, set a small error limit (ER*=1000), a low torque limit (TL*=3), and set off on Error to 1 for all axes (OE*=1). Review the command reference and controller user manual for further details.

There are 3 settings for the amplifier gain: 0.4 A/V, 0.7 A/V, and 1.0 A/V corresponding to AG (amplifier gain) 0, 1 and 2. If the gain is set to 0.7 A/V, a torque limit of 3 (TLn=3) will allow the amplifier to output no more than 2.1 amps of current on the specified axis. The controller has been programmed to test whether the Hall commutation order is correct. To test the commutation for the X axis, issue the BS command (BSX= n,m). The controller will attempt to move the motor through one revolution. If the motor is unable to move, the controller will return "unknown Hall transition", check wiring, and execute BS again'. It may be necessary to issue more voltage to create motion. The default for the BS command is BSn=0.25,1000 which will send 0.25 volts to the amplifier for 1 second. BSX=0.5,300 will issue 0.5 volts from the controller for 300 milliseconds. If the controller is able to move the motor and the Hall transitions are not correct, the controller will alert the operator and recommend which motor phases to change. For example, the controller might return "Wire A to Terminal B, Wire B to Terminal A." If the controller finds that the commutation order is correct, but the motor would run away due to positive feedback, the controller will prompt the user to "Wire Phase B to C and C to B. Exchange Hall Sensors A and B…". After making any necessary changes to the motor phase wiring, confirm correct operation by reissuing the BS command. Once the axis is wired correctly, the controller is ready to perform closed-loop motion.

## Brushless Amplifier Software Setup

Select the amplifier gain that is appropriate for the motor. The amplifier gain command (AG) can be set to 0, 1, or 2 corresponding to 0.4, 0.7, and 1.0 A/V. In addition to the gain, peak and continuous torque limits can be set through TK and TL respectively. The TK and TL values are entered in volts on an axis by axis basis. The peak limit will set the maximum voltage that will be output from the controller to the amplifier. The continuous current will set what the maximum average current is over a one second interval. The following figure captured with WSDK is indicative of the operation of the continuous and peak operation. In this figure, the continuous limit was configured for 2 volts, and the peak limit was configured for 10 volts.

## Chopper Mode

The AMP-43040 can be put into what is called a "Chopper" mode. The chopper mode is in contrast to the normal inverter mode in which the amplifier sends PWM power to the motor of +/-VS. In chopper mode, the amplifier sends a 0 to +VS PWM to the motor when moving in the forward direction, and a 0 to –VS PWM to the motor when moving in the negative direction.

This mode is useful when using low inductance motors because it reduces the losses due to switching voltages across the motor windings. It is recommended to use chopper mode when using motors with 200-500μH inductance.

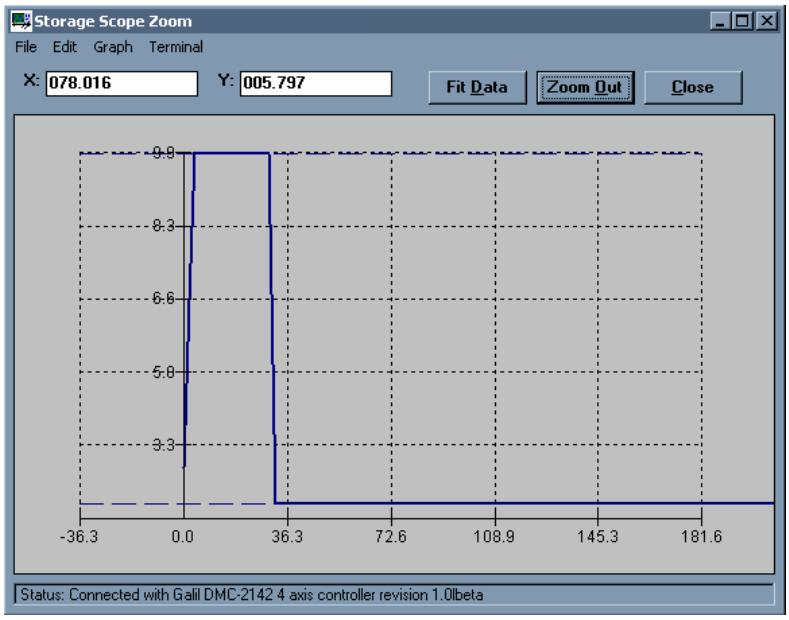



*Figure A1-2 Peak Current Operation*

With the AMP-43040 and 43020, the user is also given the ability to choose between normal and high current bandwidth (AU). In addition, the user can calculate what the bandwidth of the current loop is for their specific combination (AW). To select normal current loop gain for the X axis and high current loop gain for the Y axis, issue AU 0,1. The command AW is used to calculate the bandwidth of the amplifier using the basic amplifier parameters. To calculate the bandwidth for the X axis, issue AWX=v,l,n where v represents the DC voltage input to the card, l represents the inductance of the motor in millihenries, and n represents 0 or 1 for the AU setting.

**Note**: For most applications, unless the motor has more than 5 mH of inductance with a 24V supply, or 10 mH of inductance with a 48 volts supply, the normal current loop bandwidth option should be chosen. AW will return the current loop bandwidth in Hertz.

## Brush Amplifier Operation

The AMP-43040 and AMP-43020 also allow for brush operation. To configure an axis for brush-type operation, connect the 2 motor leads to Phase A and Phase B connections for the axis. Connect the encoders, homes, and limits as required. Set the controller into brush-axis operation by issuing BR n,n,n,n. By setting n=1, the controller will operate in brushed mode on that axis. For example, BR0,1,0,0 sets the Y-axis as brush-type, all others as brushless. If an axis is set to brush-type, the amplifier has no need for the Hall inputs. These inputs can subsequently be used as general-use inputs, queried with the QH command. The gain settings for the amplifier are identical for the brush and brushless operation. The gain settings can be set to 0.4, 0.7, or 1.0 A/V, represented by gain values of 0, 1, and 2 (e.g.. AG 0,0,2,1). The current loop gain AU can also be set to either 0 for normal, or 1 for high current loop gain.

## Using External Amplifiers

Use connectors on top of controller to access necessary signals to run external amplifiers. In order to use the full torque limit, make sure the AG setting for the axes using external amplifiers are set to 0 or 1. For more information on connecting external amplifiers, see Connecting to External Amplifiers in Chapter 2.

## Error Monitoring and Protection

The amplifier is protected against over-voltage, under-voltage, over-temperature, and over-current for brush and brushless operation. The controller will also monitor for illegal Hall states (000 or 111 with 120° phasing). The controller will monitor the error conditions and respond as programmed in the application. The errors are monitored

via the TA command. TA n may be used to monitor the errors with n = 0, 1, 2, or 3. The command will return an eight bit number representing specific conditions. TA0 will return errors with regard to under voltage, over voltage, over current, and over temperature. TA1 will return hall errors on the appropriate axes, TA2 will monitor if the amplifier current exceeds the continuous setting, and TA3 will return if the ELO input has been triggered.

The user also has the option to include the special label #AMPERR in their program to handle soft or hard errors. As long as a program is executing in thread zero, and the #AMPERR label is included, when an error is detected, the program will jump to the label and execute the user defined routine. Note that the TA command is a monitoring function only, and does not generate an error condition. The over voltage condition will not permanently shut down the amplifier or trigger the #AMPERR routine. The amplifier will be momentarily disabled; when the condition goes away, the amplifier will continue normal operation assuming it did not cause the position error to exceed the error limit.

## Hall Error Protection

During normal operation, the controller should not have any Hall errors. Hall errors can be caused by a faulty Hall-effect sensor or a noisy environment. If at any time the Halls are in an invalid state, the appropriate bit of TA1 will be set. The state of the Hall inputs can also be monitored through the QH command. Hall errors will cause the amplifier to be disabled if OE 1 is set, and will cause the controller to enter the #AMPERR subroutine if it is included in a running program.

## Under-Voltage Protection

If the supply to the amplifier drops below 12 VDC, the amplifier will be disabled. The amplifier will return to normal operation once the supply is raised above the 12V threshold; bit 3 of the error status (TA0) will tell the user whether the supply is in the acceptable range.

**Note**: If there is an #AMPERR routine and the controller is powered before the amplifier, then the #AMPERR routine will automatically be triggered.

## Over-Voltage Protection

If the voltage supply to the amplifier rises above 92 VDC, then the amplifier will automatically disable. The amplifier will re-enable when the supply drops below 90 V. This error is monitored with bit 1 of the TA0 command.

## Over-Current Protection

The amplifier also has circuitry to protect against over-current. If the total current from a set of 2 axes (ie A and B or C and D) exceeds 20 A, the amplifier will be disabled. The amplifier will not be re-enabled until there is no longer an over-current draw and then either SH command has been sent or the controller is reset. Since the AMP-43040 is a trans-conductance amplifier, the amplifier will never go into this mode during normal operation. The amplifier will be shut down regardless of the setting of OE, or the presence of the #AMPERR routine. Bit 0 of TA0 will be set.

**Note**: If this fault occurs, it is indicative of a problem at the system level. An over-current fault is usually due to a short across the motor leads or a short from a motor lead to ground.

## Over-Temperature Protection

The controller is also equipped with over-temperature protection.

**Rev A and Rev B amplifiers:**

If the average heat sink temperature rises above 100°C, then the amplifier will be disabled. Bit 2 of TA0 will be set when the over-temperature occurs on the A-D axis amplifier, and Bit 6 of TA0 will be set when the over-temperature occurs on the E-H axis amplifier. The over-temperature condition will trigger the #AMPERR routine if included in the program on the controller.

The amplifier will re-enable when the temperature drops below 100 °C.

**Rev C and newer amplifiers:**

If the average heat sink temperature rises above 80°C, then the amplifier will be disabled. Bit 2 of TA0 will be set when the over-temperature occurs on the A-D axis amplifier, and Bit 6 of TA0 will be set when the over-temperature occurs on the E-H axis amplifier. The over-temperature condition will trigger the #AMPERR routine if included in the program on the controller.

The amplifier will not be re-enabled until the temperature drops below 80°C and then either an SH command is sent to the controller, or the controller is reset (RS command or power cycle).

Rev C Amplifiers began shipping in December 2008.

## ELO Input

If the ELO input on the controller is triggered, then the amplifier will be shut down at a hardware level, the motors will be essentially in a Motor Off (MO) state. TA3 will return a 3 and the #AMPERR routine will run when the ELO input is triggered. To recover from an ELO, an MO then SH must be issued, or the controller must be reset.

It is recommended that OE1 be used for all axes when the ELO is used in an application.

# A2 – AMP-43140

## Introduction

The AMP-43140 contains four linear drives for operating small brush-type servo motors. The AMP-43140 requires a ± 12–30 DC Volt input. Output power is 20 W per amplifier or 60 W total. The gain of each transconductance linear amplifier is 0.1 A/V at 1 A maximum current. The typical current loop bandwidth is 4 kHz.

The AMP-43140 can be ordered to have a 100mA maximum current output where the gain of the amplifier is 10mA/V. Order as (-D3140-100mA).

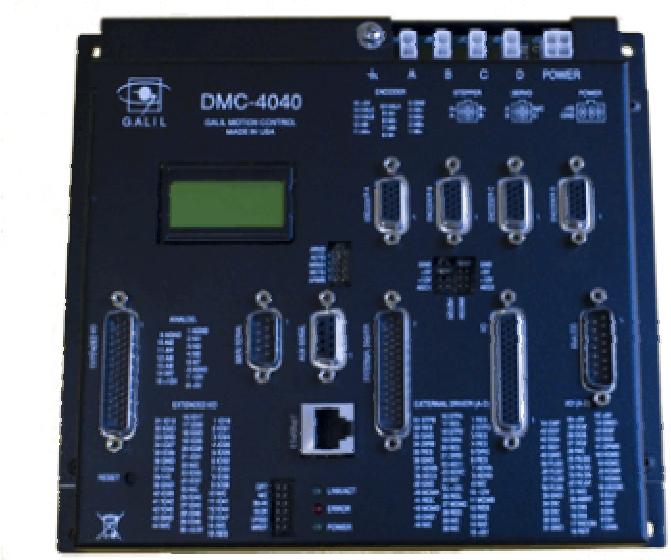



*Figure A2-1 – DMC-4040-C012-I000-D3140 (DMC-4040 with AMP-43140)*

# Electrical Specifications

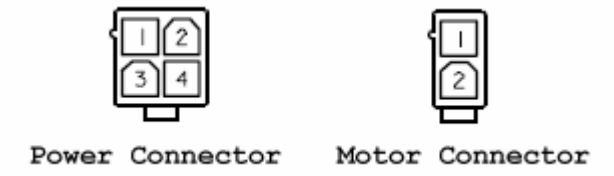
The amplifier is a brush type trans-conductance linear amplifier. The amplifier operates in torque mode, and will output a motor current proportional to the command signal input.

| | |
|---|---|
| DC Supply Voltage: | +/-12-30 VDC (bipolar) |
| Max Current (per axis) | 1.0 Amps (100mA option) |
| Amplifier gain: | 0.1 A/V (10mA/V option) |
| Power output (per channel): | 20 W |
| Total max. power output: | 60 W |

## Mating Connectors

| | On Board Connector | Terminal Pins |
|---|---|---|
| **POWER** | 4-pin MATE-N-LOK<br>MOLEX# 39-01-2045 | MOLEX#44476-3112 |
| **A,B,C,D: 4-pin Motor Power Connectors** | 2-pin MATE-N-LOK<br>MOLEX# 39-01-2025 | MOLEX#44476-3112 |

For mating connectors see http://www.molex.com/




| Power Connector | |
|---|---|
| Pin Number | Connection |
| 1,2 | Power Supply Ground |
| 3 | -VS (-DC Power) |
| 4 | +VS (DC Power) |
| **Motor Connector** | |
| 1 | Motor Lead A- |
| 2 | Motor Lead A+ |

# Operation

## Using External Amplifiers

Use connectors on top of controller to access necessary signals to run external amplifiers. For more information on connecting external amplifiers, see Connecting to External Amplifiers in Chapter 2.

## ELO Input

If the ELO input on the controller is triggered, then the amplifier will be shut down at a hardware level, the motors will be essentially in a Motor Off (MO) state. TA3 will return a 3 and the #AMPERR routine will run when the ELO input is triggered. To recover from an ELO, an MO then SH must be issued, or the controller must be reset.

It is recommended that OE1 be used for all axes when the ELO is used in an application.

# A3 – SDM-44040

## Introduction

The SDM-44040 is a stepper driver module capable of driving up to four bipolar two-phase stepper motors. The current is selectable with options of 0.5, 0.75, 1.0, and 1.4 Amps/Phase. The step resolution is selectable with options of full, half, 1/4 and 1/16.


*Figure A3-1 – DMC-4040-C012-I000-D4040 (DMC-4040 with SDM-44040)*

# Electrical Specifications

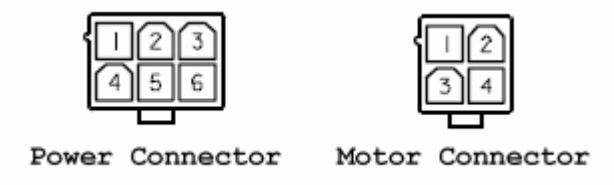
The amplifier is a brush type trans-conductance linear amplifier. The amplifier operates in torque mode, and will output a motor current proportional to the command signal input.

| | |
|---|---|
| DC Supply Voltage: | 12-30 VDC |
| Max Current (per axis) | 1.4 Amps/Phase Amps (Selectable with AG command) |
| Maximum Step Frequency: | 6 MHz |
| Motor Type: | Bipolar 2 Phase |

## Mating Connectors

| | On Board Connector | Terminal Pins |
|---|---|---|
| **POWER** | 6-pin MATE-N-LOK<br>MOLEX# 39-31-0060 | MOLEX#44476-3112 |
| **A,B,C,D: 4-pin Motor Power Connectors** | 4-pin MATE-N-LOK<br>MOLEX# 39-31-0040 | MOLEX#44476-3112 |

For mating connectors see http://www.molex.com/




| **Power Connector** | |
|---|---|
| Pin Number | Connection |
| 1,2,3 | DC Power Supply Ground |
| 4,5,6 | +VS (DC Power) |
| **Motor Connector** | |
| 1 | B- |
| 2 | A- |
| 3 | B+ |
| 4 | A+ |

# Operation

The AG command sets the current on each axis, the LC command configures each axis's behavior when holding position and the YA command sets the step driver resolution. These commands are detailed below, see also the command reference for more information:

## Current Level Setup (AG Command)

AG configures how much current the SDM-206x0 delivers to each motor. Four options are available: 0.5A, 0.75A, 1.0A, and 1.4 Amps

Drive Current Selection per Axis: AG n,n,n,n,n,n,n,n

n = 0 0.5 A

n = 1 0.75 A (default)

n = 2 1.0 A

n = 3 1.4 A

## Low Current Setting (LC Command)

LC configures each motor's behavior when holding position (when RP is constant) and multiple configurations:

LC command set to 0 "Full Current Mode" - causes motor to use 100% of peak current (AG) while at a "resting" state (profiler is not commanding motion). This is the default setting.

LC command set to 1 "Low Current Mode" - causes motor to use 25% of peak current while at a "resting" state. This is the recommended configuration to minimize heat generation and power consumption.

LC command set to an integer between 2 and 32767 specifying the number of samples to wait between the end of the move and when the amp enable line toggles

Percentage of full (AG) current used while holding position with LC n,n,n,n,n,n,n,n

| | |
|---|---|
| **n = 0** | 100% |
| **n = 1** | 25% |

The LC command must be entered after the motor type has been selected for stepper motor operation (i.e. MT-2,-2,-2,-2). LC is axis-specific, thus LC1 will cause only the X-axis to operate in "Low Current" mode.

## Step Drive Resolution Setting (YA command)

When using the SDM-44040, the step drive resolution can be set with the YA command

Step Drive Resolution per Axis: YA n,n,n,n,n,n,n,n

n = 1 Full

n = 2 Half

n = 4 1/4

n = 16 1/16

## ELO Input

If the ELO input on the controller is triggered, then the amplifier will be shut down at a hardware level, the motors will be essentially in a Motor Off (MO) state. TA3 will return a 3 and the #AMPERR routine will run when the ELO input is triggered. To recover from an ELO, an MO then SH must be issued, or the controller must be reset.

It is recommended that OE1 be used for all axes when the ELO is used in an application.

# A4 – SDM-44140

## Introduction

The SDM-44140 microstepper module drives four bipolar two-phase stepper motors with 1/64 microstep resolution. The current is selectable with options of 0.5, 1.0, 2.0, & 3.0 Amps per axis.


*Figure A1-1 – DMC-4040-C012-I000-D4140 (DMC-4040 with SDM-44140)*

# Electrical Specifications

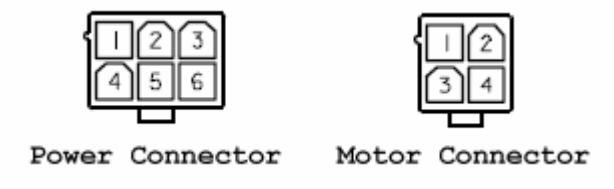
The amplifier is a brush type trans-conductance linear amplifier. The amplifier operates in torque mode, and will output a motor current proportional to the command signal input.

| | |
|---|---|
| DC Supply Voltage: | 12-60 VDC |
| Max Current (per axis) | 3.0 Amps (Selectable with AG command) |
| Max Step Frequency: | 6 MHz |
| Motor Type: | Bipolar 2 Phase |
| Switching Frequency: | 60 kHz |
| Minimum Load Inductance: | 0.5 mH |

## Mating Connectors

| | On Board Connector | Terminal Pins |
|---|---|---|
| **POWER** | 6-pin MATE-N-LOK<br>MOLEX# 39-01-2065 | MOLEX#44476-3112 |
| **A,B,C,D: 4-pin Motor Power Connectors** | 4-pin MATE-N-LOK<br>MOLEX# 39-01-2045 | MOLEX#44476-3112 |

For mating connectors see http://www.molex.com/




| **Power Connector** | |
|---|---|
| Pin Number | Connection |
| 1,2,3 | DC Power Supply Ground |
| 4,5,6 | +VS (DC Power) |
| **Motor Connector** | |
| 1 | B- |
| 2 | A- |
| 3 | B+ |
| 4 | A+ |

# Operation

The AG command sets the current on each axis and the LC command configures each axis's behavior when holding position. These commands are detailed below:

## Current Level Setup (AG Command)

AG configures how much current the SDM-44140 delivers to each motor. Four options are available: 0.5A, 1.0A, 2.0A, and 3.0Amps (**Note**: when using the 3.0A setting, mounting the unit to a metal or heat dissipating surface is recommended).

Drive Current Selection per Axis: AG n,n,n,n,n,n,n,n

n = 0 0.5 A

n = 1 1 A (default)

n = 2 2 A

n = 3 3.0 A

## Low Current Setting (LC Command)

LC configures each motor's behavior when holding position (when RP is constant) and multiple configurations:

LC command set to 0 "Full Current Mode" - causes motor to use 100% of peak current (AG) while at a "resting" state (profiler is not commanding motion). This is the default setting.

LC command set to 1 "Low Current Mode" - causes motor to use 25% of peak current while at a "resting" state. This is the recommended configuration to minimize heat generation and power consumption.

LC command set to an integer between 2 and 32767 specifying the number of samples to wait between the end of the move and when the amp enable line toggles

Percentage of full (AG) current used while holding position with LC n,n,n,n,n,n,n,n

| | |
|---|---|
| **n = 0** | 100% |
| **n = 1** | 25% |

The LC command must be entered after the motor type has been selected for stepper motor operation (i.e. MT-2,-2,-2,-2). LC is axis-specific, thus LC1 will cause only the X-axis to operate in "Low Current" mode.

## ELO Input

If the ELO input on the controller is triggered, then the amplifier will be shut down at a hardware level, the motors will be essentially in a Motor Off (MO) state. TA3 will return a 3 and the #AMPERR routine will run when the ELO input is triggered. To recover from an ELO, an MO then SH must be issued, or the controller must be reset.

It is recommended that OE1 be used for all axes when the ELO is used in an application.

# Index