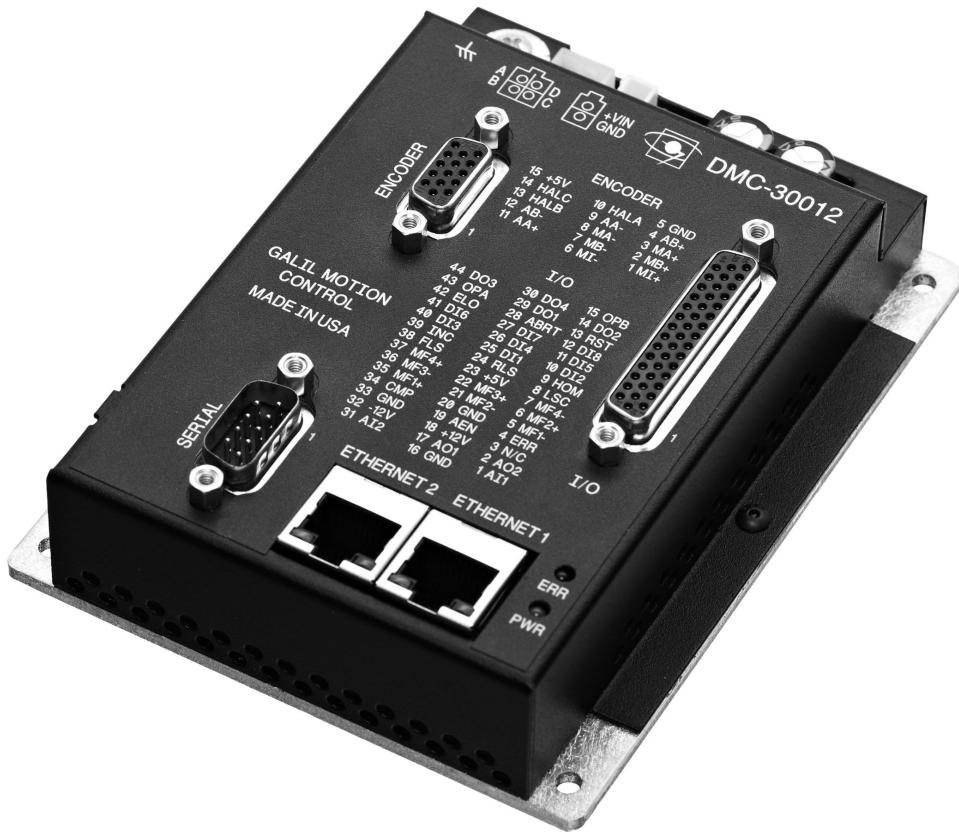


# DMC-30000

Manual Rev. 1.4e



By Galil Motion Control, Inc.

270 Technology Way  
Rocklin, California 95765  
916.626.0101

[support@galil.com](mailto:support@galil.com)  
[galil.com](http://galil.com)

10/2024

---

## Using This Manual

This user manual provides information for proper operation of the DMC-30000 controller. A separate supplemental manual, the Command Reference, contains a description of the commands available for use with this controller. It is recommended that the user download the latest version of the Command Reference and User Manual from the Galil Website.

<http://www.galil.com/downloads/manuals-and-data-sheets>

The DMC-30000 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.

<b>WARNING</b>	<p>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.</p>
----------------	--

# Contents

<a href="#">Using This Manual.....</a>	.ii
<b>Contents</b>	<b>iii</b>
<b>Chapter 1 Overview</b>	<b>1</b>
<a href="#">Introduction.....</a>	1
<a href="#">Part Numbers.....</a>	2
<a href="#">Overview of Motor Types.....</a>	4
<a href="#">Galil Internal Amplifiers and Drivers.....</a>	5
<a href="#">Overview of External Amplifiers.....</a>	6
<a href="#">Functional Elements.....</a>	7
<b>Chapter 2 Getting Started</b>	<b>10</b>
<a href="#">Dimensions.....</a>	10
<a href="#">Elements You Need.....</a>	14
<a href="#">Installing the DMC, Amplifiers, and Motors.....</a>	15
<b>Chapter 3 Connecting Hardware</b>	<b>23</b>
<a href="#">Overview.....</a>	23
<a href="#">Overview of Optoisolated Inputs.....</a>	23
<a href="#">Optoisolated Input Electrical Information.....</a>	26
<a href="#">Optoisolated Outputs.....</a>	28
<a href="#">Feedback Inputs and Multi-Function (MF) Pins.....</a>	31
<a href="#">TTL Outputs.....</a>	34
<a href="#">Analog Inputs.....</a>	35
<a href="#">Analog Outputs.....</a>	36
<a href="#">External Amplifier Interface.....</a>	36
<b>Chapter 4 Software Tools and Communication</b>	<b>38</b>
<a href="#">Introduction.....</a>	38
<a href="#">Controller Response to Commands.....</a>	38
<a href="#">Unsolicited Messages Generated by Controller.....</a>	39
<a href="#">Serial Communication Port.....</a>	39
<a href="#">Ethernet Configuration.....</a>	40
<a href="#">Modbus.....</a>	43
<a href="#">Data Record.....</a>	46
<a href="#">Galil Software.....</a>	49
<a href="#">Creating Custom Software Interfaces.....</a>	49
<b>Chapter 5 Command Basics</b>	<b>50</b>
<a href="#">Introduction.....</a>	50
<a href="#">Command Syntax - ASCII.....</a>	50
<a href="#">Controller Response to DATA.....</a>	51
<a href="#">Interrogating the Controller.....</a>	51
<b>Chapter 6 Programming Motion</b>	<b>53</b>
<a href="#">Overview.....</a>	53
<a href="#">Independent Axis Positioning.....</a>	54
<a href="#">Independent Jogging.....</a>	55

<u>Position Tracking</u>	56
<u>Linear Interpolation Mode</u>	60
<u>Vector Mode: Linear and Circular Interpolation Motion</u>	62
<u>Electronic Gearing</u>	65
<u>Electronic Cam</u>	67
<u>PVT Mode</u>	69
<u>Contour Mode</u>	72
<u>Virtual Axis</u>	75
<u>Stepper Motor Operation</u>	76
<u>Stepper Position Maintenance Mode (SPM)</u>	78
<u>Dual Loop (Auxiliary Encoder)</u>	82
<u>Motion Smoothing</u>	83
<u>Homing</u>	85
<u>High Speed Position Capture (The Latch Function)</u>	87
<u>Real Time Clock</u>	88

## Chapter 7 Application Programming 89

<u>Overview</u>	89
<u>Program Format</u>	89
<u>Executing Programs - Multitasking</u>	91
<u>Debugging Programs</u>	91
<u>Program Flow Commands</u>	93
<u>Mathematical and Functional Expressions</u>	108
<u>Variables</u>	110
<u>Operands</u>	112
<u>Arrays</u>	112
<u>Input of Data (Numeric and String)</u>	115
<u>Output of Data (Numeric and String)</u>	117
<u>Hardware I/O</u>	122
<u>Example Applications</u>	126

## Chapter 8 Hardware & Software Protection 129

<u>Introduction</u>	129
<u>Hardware Protection</u>	129
<u>Software Protection</u>	130

## Chapter 9 Troubleshooting 133

<u>Overview</u>	133
-----------------	-----

## Chapter 10 Theory of Operation 136

<u>Overview</u>	136
<u>Operation of Closed-Loop Systems</u>	138
<u>System Modeling</u>	139
<u>System Analysis</u>	143
<u>System Design and Compensation</u>	144

## Appendices 148

<u>Electrical Specifications</u>	148
<u>Certifications</u>	149
<u>Performance Specifications</u>	150
<u>Ordering Options</u>	151
<u>Power Connector Part Numbers</u>	156
<u>Power Wiring Diagrams</u>	157

<u>Cable Shielding.....</u>	163
<u>Serial Cable Connections.....</u>	164
<u>Input Current Limitations.....</u>	166
<u>Pin-outs.....</u>	168
<u>Signal Descriptions.....</u>	170
<u>List of Other Publications.....</u>	171
<u>Training Seminars.....</u>	171
<u>Contacting Us.....</u>	172
<u>WARRANTY .....</u>	172
<b>A1 – DMC-30012</b>	<b>173</b>
<u>Description.....</u>	173
<u>Electrical Specifications.....</u>	174
<u>Servo Motor Operation.....</u>	175
<u>Error Monitoring and Protection.....</u>	178
<b>A2 – DMC-30014</b>	<b>180</b>
<u>Description.....</u>	180
<u>Electrical Specifications.....</u>	181
<u>Operation.....</u>	183
<u>Error Monitoring and Protection.....</u>	185
<b>A3 – DMC-30016</b>	<b>186</b>
<u>Description.....</u>	186
<u>Electrical Specifications.....</u>	187
<u>Stepper Motor Operation.....</u>	188
<u>Error Monitoring and Protection.....</u>	189
<b>A4 – DMC-30017</b>	<b>190</b>
<u>Description.....</u>	190
<u>Electrical Specifications.....</u>	191
<u>Servo Motor Operation.....</u>	192
<u>Stepper Motor Operation.....</u>	195
<u>Error Monitoring and Protection.....</u>	196
<b>A5 – DMC-31000</b>	<b>198</b>
<u>Description.....</u>	198
<u>Theory of Operation.....</u>	199

# Chapter 1 Overview

---

## Introduction

The DMC-30000 Series is Galil's latest generation single-axis motion controller. It uses a 32-bit RISC processor to provide higher speed than older models. The DMC-30000 is available as a compact card-level or box-level unit and connects to a stepper or servo motor amplifier of any power range. The DMC-300xx can be purchased with internal drives which can minimize space, cost and wiring. The motion controller can operate stand-alone or can be networked to a PC via Ethernet.

Features include PID compensation with both velocity and acceleration feed-forward, program memory with multitasking for concurrent execution of multiple programs, and uncommitted optically isolated inputs and outputs for synchronizing motion with external events. Modes of motion include point-to-point positioning, jogging, contouring, PVT, electronic gearing and electronic cam. Like all Galil motion controllers, these controllers use a simple, English-like command language which makes them very easy to program. Galil Design Kit (GDK) software further simplifies system set-up with "one button" servo tuning and real-time display of position and velocity information.

Designed to solve complex motion problems, the DMC-30000 can be used for applications involving jogging, point-to-point positioning, vector positioning, electronic gearing, multiple move sequences, contouring and PVT Mode. The controller eliminates jerk by programmable acceleration and deceleration with profile smoothing. For the smooth following of complex contours, the DMC-30000 can receive a continuous stream of any 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-30000 provides uncommitted I/O, including 8 optoisolated digital inputs, 4 optically isolated outputs, 2 analog inputs for interface to joysticks, sensors, and pressure transducers and 1 uncommitted analog output. An additional 2 digital inputs (per axis) are available if the auxiliary encoder inputs are not in use. The DMC-30000 also has digital inputs for home switches, abort signals, and forward and reverse limits (1 each per axis).

Commands are sent in ASCII. Additional software is available for automatic tuning, trajectory viewing on a PC screen, and program development using many environments such as VB.NET, C/C++, Python and others. Drivers for Windows 10 (32 & 64 bit) and Linux are available.

## Part Numbers

The DMC-30000 is internally made of up three different boards: the I/O board which also acts as a break-out board, the DMC controller board, and the optional internal amplifier (AMP/SDM). Figure 1.1 below shows how the internal DMC-30000 is laid out.

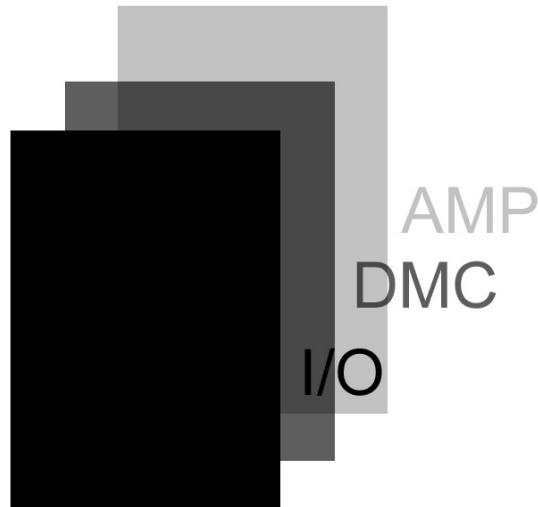


Figure 1.1: Abstract internal layout of the DMC-30000 PCB Boards

The DMC-30000 part numbers modify each of these boards. As shown in Figure 1.2 the first "X" (DMC-3X000) modifies which I/O board is used, while the last "X", (DMC-3000X) modifies either the optional amplifier board and/or how the unit is powered. In addition, the user also can specify whether the DMC-30000 comes in a card or box form factor by specifying either "-CARD" or "BOX" in the "-XXXX" place holder. Lastly, the DMC-30000 has several additional configuration options that can be added to the "Y" place holder. Note, several Y-options can be added by placing a comma between options.

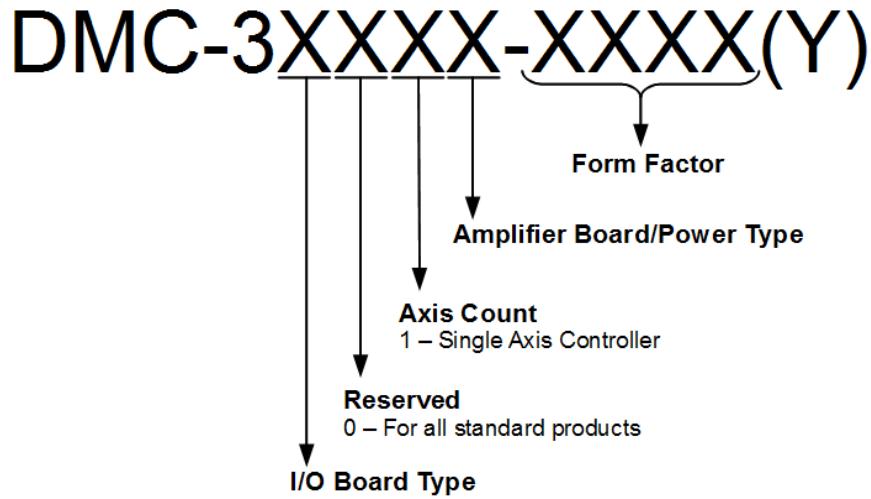


Figure 1.2: DMC-30000 Part Number Layout

The following tables provide a summary of the DMC-30000 options and where find the appropriate documentation for each.

## I/O Board Options

DMC-3X010, where X is	Description	Documentation
0	12-bit, 0-5V analog inputs, Default Option	
1	Adds 16-bit, $\pm 10V$ configurable analog inputs and Sin/Cos feedback	A5 – DMC-31000, pg 199

Table 1.1: I/O Board Options

## Amplifier Board/Power Type Options

DMC-3001X, where X is	Description	Documentation
0	Amplifier: None Power: 5, $\pm 12$ VDC	Input Power Requirements, pg 150
1	Amplifier: None Power: 9-48 VDC Default	Power Connector Part Numbers, pg 157 Power Wiring Diagrams, pg 158
2	Amplifier: 800W sine drive Power: 20-80 VDC	A1 – DMC-30012, pg 174
4	Amplifier: 20W Linear sine drive Power: 15-40 VDC	A2 – DMC-30014, pg 181
6	Amplifier: 1.A stepper drive Power: 12-30 VDC	A3 – DMC-30016, pg 187
7	Amplifier: Servo and stepper drive Power: 20-80 VDC	A4 – DMC-30017, pg 191

Table 1.2: Amplifier Board and Power Options

## Form Factor Options

Where -XXXX is	Description	Documentation
-CARD		DMC-30010-CARD, pg 10 DMC-30011-CARD, pg 11
-BOX		DMC-30010-BOX and DMC-30011-BOX, pg 12 DMC-30012-BOX, DMC-30014-BOX, DMC-30016-BOX, and DMC-30017-BOX, pg 13 The -BOX option is required for DMC-30000 models that use an internal amplifier.

Table 1.3: Form Factor Options

## Y Additional Options

Where (Y) is	Description	Documentation
LSNK	25mA Sinking outputs	25mA Sinking Optoisolated Outputs (LSNK), pg 30
LSRC	25mA sourcing outputs	No longer available, contact Galil
HSRC	500mA sourcing outputs	500mA Sourcing Optoisolated Outputs (HSRC), pg 31
HSNK	500mA sinking outputs	No longer available, contact Galil
4-20mA	4-20mA analog inputs	
TRES	Termination Resistors Differential encoder required	
SR90	90V Shunt Regulator	
SR190	190V Shunt Regulator Only available with 160V option	
SER	SSI and BiSS Feedback	
HALLF	Filtered Hall Sensor inputs	
ISCNTL	Isolate controller and AMP power	
INVELO	Inverted ELO Logic	
160V	160V DC Max Amplifier Input Voltage Valid with the DMC-30012. ISCNTL, SR190, TRES required	
422	RS-422 on Serial Port	
DIN	DIN Rail Mounting Clips -box required	
MO	Motor-off jumper installed	

Table 1.4: Form Factor Options

<b>WARNING</b>	If no option is specified, the default optoisolated outputs for the DMC-30000 are 4mA sinking, see Standard 4mA Sinking Optoisolated Outputs, pg 30 for further details.
----------------	--

---

## Overview of Motor Types

The DMC-30000 can provide the following types of motor control:

1. Standard servo motors with  $\pm 10$  volt command signals
2. Step motors with step and direction signals
3. Other actuators such as hydraulics and ceramic motors - For more information, contact Galil.

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

### Standard Servo Motor with $\pm 10$ Volt Command Signal

The DMC-30000 achieves superior precision through the 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 ( $\pm 10$  volts) to connect to a servo amplifier. This connection is described in Chapter 2.

## Stepper Motor with Step and Direction Signals



The DMC-30000 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 the stepper motor has encoder feedback, users can run the stepper motor in a closed loop fashion with a 2PB supported amplifier. See the A4 – DMC-30017 for more information.

Alternatively, if the stepper motor has encoder feedback, but the user prefers to run the motor in an open-loop configuration, 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.

---

## Galil Internal Amplifiers and Drivers

Galil offers a variety of internal servo amplifiers and stepper drivers that are integrated directly into the enclosure of the DMC-30000. By doing so, Galil can provide a simple, straightforward motion control solution in a single box. This saves space and prevents the hassle of configuring a separate device.

### **DMC-30012 (DMC-30000 with 800W Sinusoidal Amplifier)**

The DMC-30012 (A1 – DMC-30012) provides an amplifier that drives motors operating at 20–80 VDC, up to 10 Amps continuous, 15 Amps peak. The gain settings of the amplifier are user-programmable at 0.4 Amp/Volt, 0.8 Amp/Volt and 1.6 Amp/Volt. The switching frequency is 33 kHz. The amplifier offers protection for over-voltage, under-voltage, over-current, and short-circuit. The SR90 – 90V Shunt Regulator Option is also available for the DMC-30012.

### **DMC-30014 (DMC-30000 with 20W Sinusoidal Linear Amplifier)**

The DMC-30014 (A2 – DMC-30014) provides an linear amplifier that drives motors operating at 15–40 VDC, up to 1 Amps continuous, 2 Amps peak. The gain settings of the amplifier are set at 0.2 A/V. The amplifier offers protection for over-temp.

### **DMC-30016 (DMC-30000 with 1.4 Amp stepper driver)**

The DMC-30016 (A3 – DMC-30016) includes a microstepping drive for operating two-phase bipolar stepper motors.

The DMC-30016 drive operates a two-phase bipolar stepper motor in full-step, half-step, 1/4 step or 1/16 step. It is user configurable from 0.5A to 1.4A per phase in ~7 mA increments at 12–30VDC. The dimensions of the DMC-30016 controller and drive package are 3.9" x 5.0" x 1.5", and no external heatsink is required.

### **DMC-30017 (DMC-30000 with 6 Amp stepper driver or 800W Sinusoidal Amplifier)**

The DMC-30017 (A4 – DMC-30017) includes a microstepping drive for operating two-phase bipolar stepper motors, the drive can also be configured for a sinusoidally commutated, PWM amplifier for driving brushed or brushless servo motors.

**Microstepping Drive:** The microstepping drive produces 256 microsteps per full step or 1024 steps per full cycle which results in 51,200 steps/rev for a standard 200-step motor. The maximum step rate generated by the controller is 3,000,000 microsteps/second. The DMC-30017 can drive stepper motors at up to 6 Amps at 20-80VDC. There are four selectable current gains: 0.75 A, 1.5 A, 3 A and 6A. A selectable low current mode reduces the current by 75% when the motor is not in motion.

**Sinusoidally Commutated Amplifier:** When set to servo mode, the DMC-30017 has the same specs as the DMC-30012.

---

## 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  $\pm 10$  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  $\pm 10$  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



In this mode, the controller provides two signals to the driver of the stepper motor: Step and Direction.

# Functional Elements

The DMC-30000 circuitry can be divided into the following functional groups as shown in Figure 1.3 and discussed below.

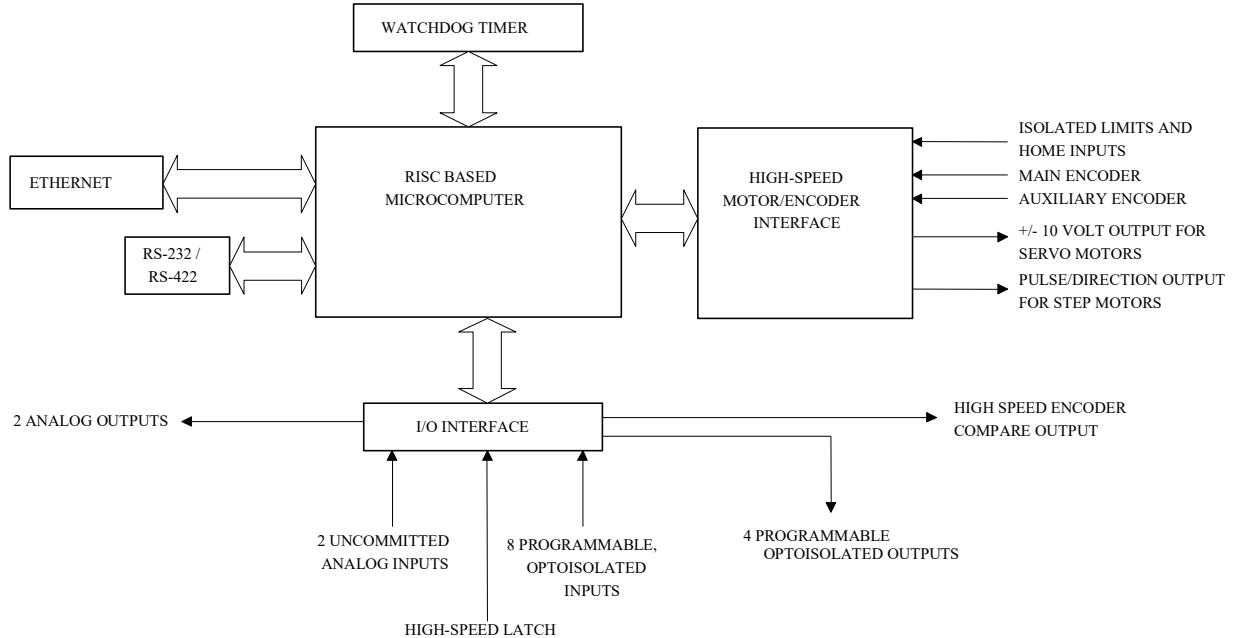


Figure 1.3: DMC-30000 Functional Elements

## Microcomputer Section

The main processing unit of the controller is a specialized Microcomputer with RAM and Flash FLASH. The RAM provides memory for variables, array elements, and application programs. The flash FLASH 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 15 MHz. For standard servo operation, the controller generates a  $\pm 10$  volt analog signal (16-bit DAC). For stepper motor operation, the controller generates a step and direction signal.

## Communication

The communication interface with the DMC-30000 consists of a daisy-chainable Ethernet 100 Base-T port and a 115kbaud RS-232 programming port.

## General I/O

The DMC-30000 provides interface circuitry for 8 bi-directional, optoisolated inputs, 4 optoisolated outputs and 2 analog inputs with 12-bit ADC (16-bit optional). Unused auxiliary encoder inputs may also be used as additional inputs (2 inputs). The general inputs as well as the index pulse can also be used as high speed latches for each axis. A high speed encoder compare output is also provided.

## System Elements

As shown in Figure 1.4, the DMC-30000 is part of a motion control system which includes amplifiers, motors and encoders. These elements are described below.

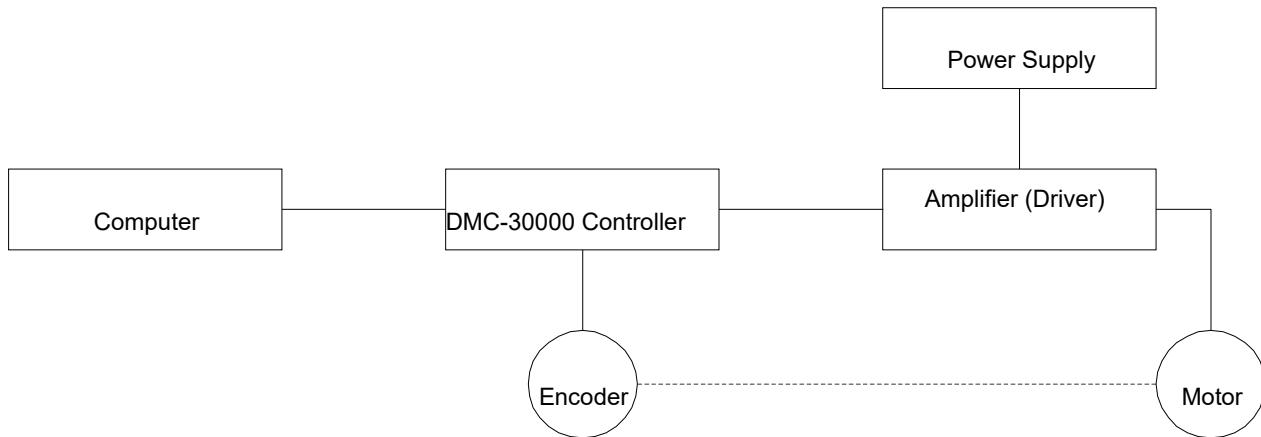


Figure 1.4: Elements of Servo Systems

## 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.

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-30000.

## Amplifier (Driver)

The power amplifier converts a  $\pm 10$  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-30000. See the A1 – DMC-30012, A3 – DMC-30016 and the A4 – DMC-30017 sections in the Appendices or <http://www.galil.com/motion-controllers/single-axis/dmc-3x01x> for more information.

## Encoder

An encoder translates motion into electrical pulses which are fed back into the controller. The DMC-30000 accepts feedback from either a rotary or linear encoder. Typical encoders provide two channels in quadrature, known as MA and MB. This type of encoder is known as a quadrature encoder. Quadrature encoders may be either single-

ended (MA and MB) or differential (MA+, MA- and MB+, MB-). The DMC-30000 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-30000 can be ordered with  $120\ \Omega$  termination resistors installed on the encoder inputs. See the Ordering Options in the Appendix for more information.

The DMC-30000 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 3,750,000 full encoder cycles/second (15,000,000 quadrature counts/sec). For example, if the encoder line density is 10,000 cycles per inch, the maximum speed is 200 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-30000. Single-ended 12 Volt signals require a bias voltage input to the complementary inputs).

The DMC-30000 can accept analog feedback instead of an encoder. For electrical specifications see [Analog Inputs](#), pg 36 and the AF in the Command Reference for configuration.

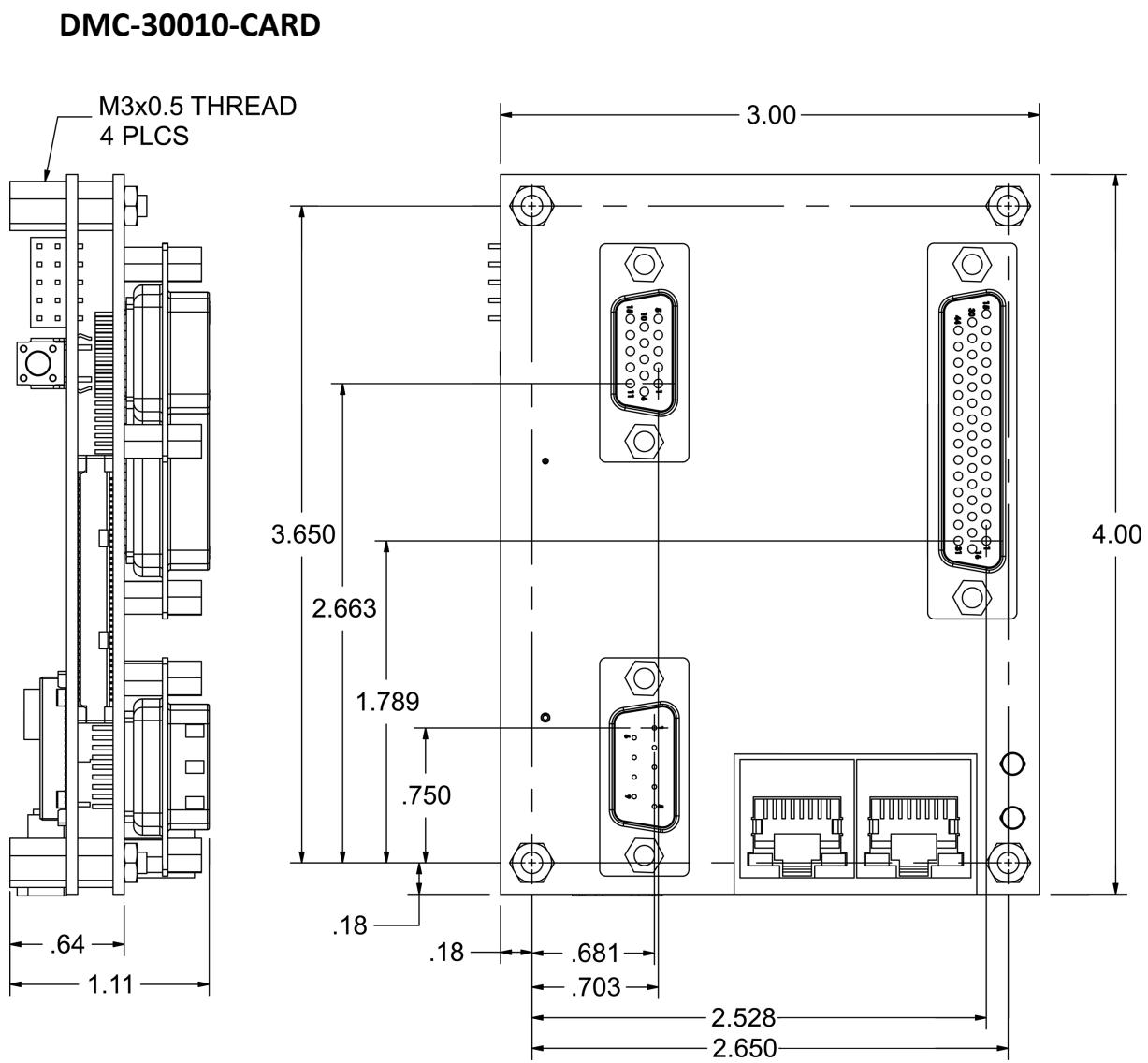
To interface with other types of position sensors such as absolute encoders, Galil can customize the controller and command set. Please contact Galil to discuss particular system and application requirements with an Applications Engineer.

## Watch Dog Timer

The DMC-30000 provides an internal watch dog timer which checks for proper microprocessor operation. The timer toggles the Amplifier Enable Output (AEN) which can be used to switch the amplifiers off in the event of a serious DMC-30000 failure. The AEN output is normally high. During power-up and if the microprocessor ceases to function properly, the AEN output will go low. The error light will also turn on at this stage. A reset is required to restore the DMC-30000 to normal operation. Consult the factory for a Return Materials Authorization (RMA) Number if the DMC-30000 is damaged.

# Chapter 2 Getting Started

## Dimensions



## DMC-30011-CARD

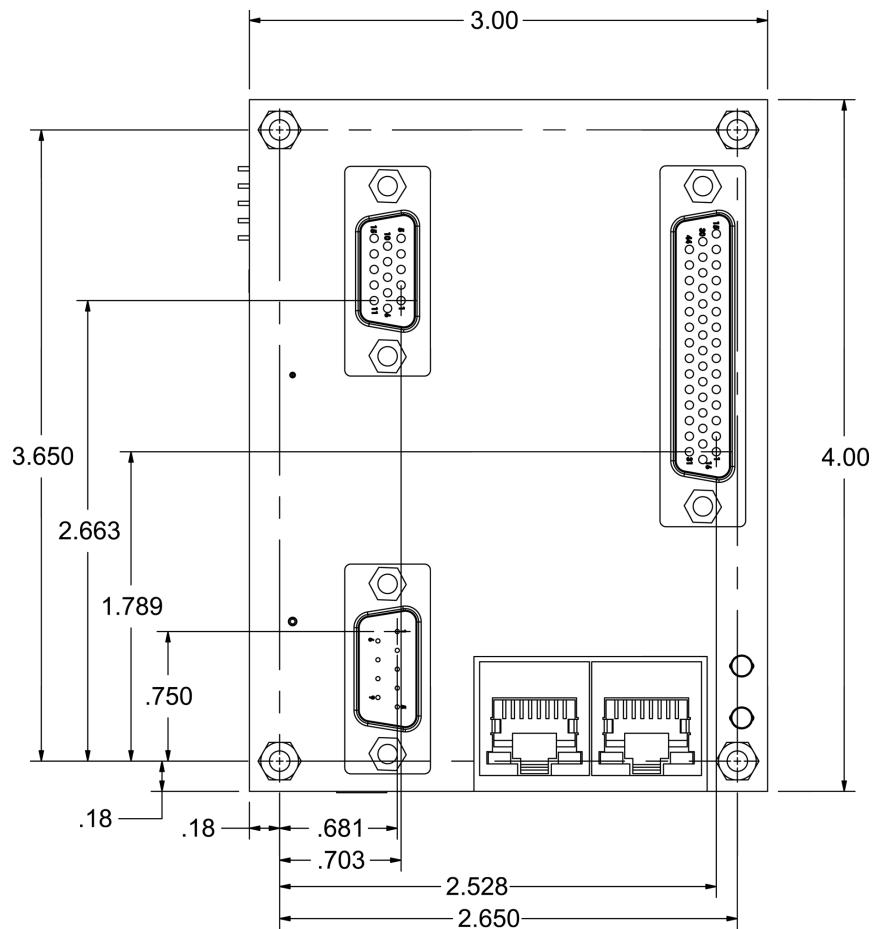
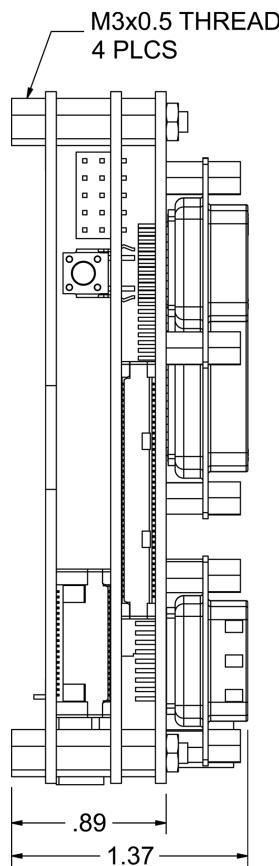


Figure 2.2: DMC-30011-CARD Dimensions (in inches)

## DMC-30010-BOX and DMC-30011-BOX

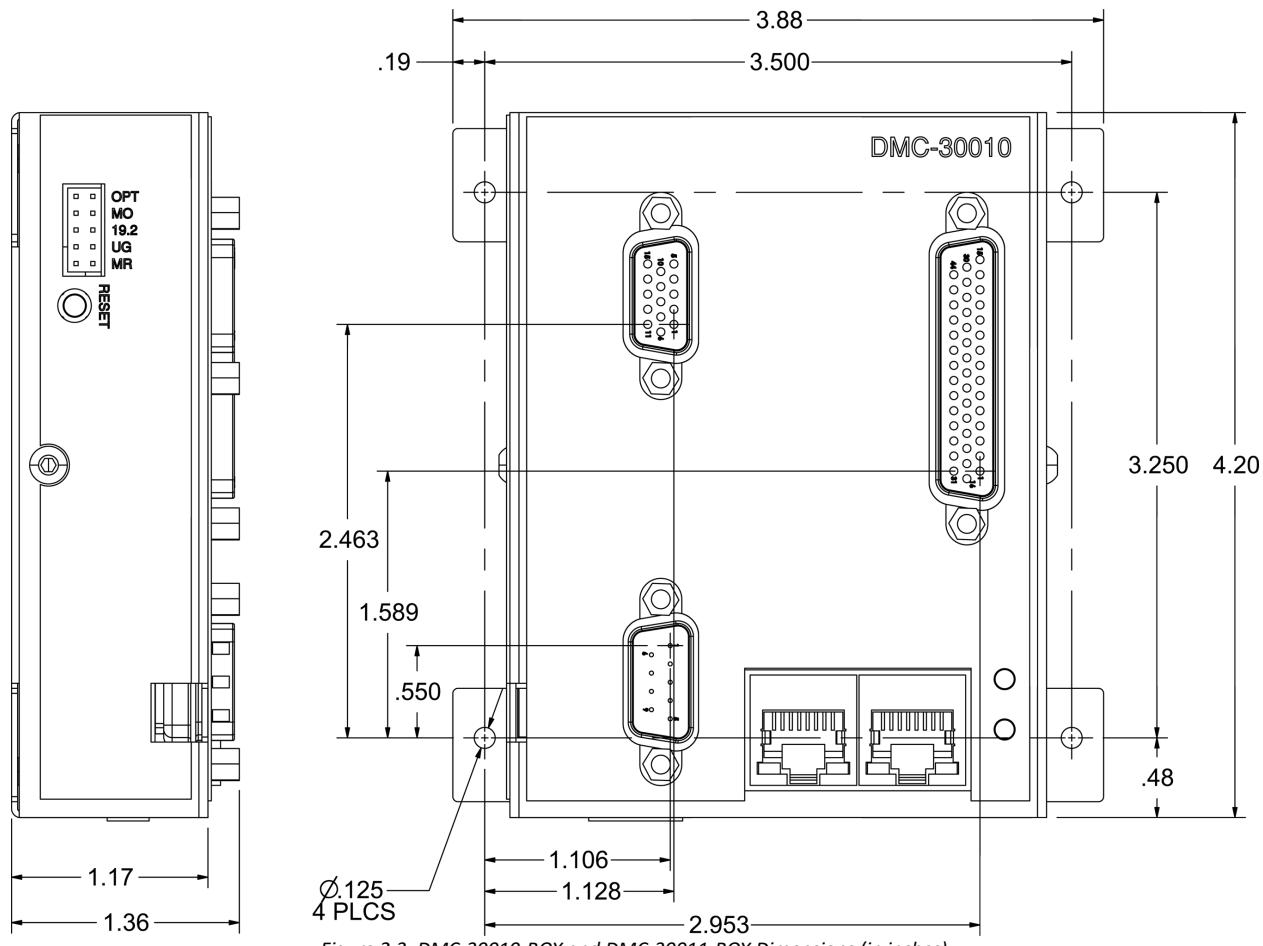


Figure 2.3: DMC-30010-BOX and DMC-30011-BOX Dimensions (in inches)

## DMC-30012-BOX, DMC-30014-BOX, DMC-30016-BOX, and DMC-30017-BOX

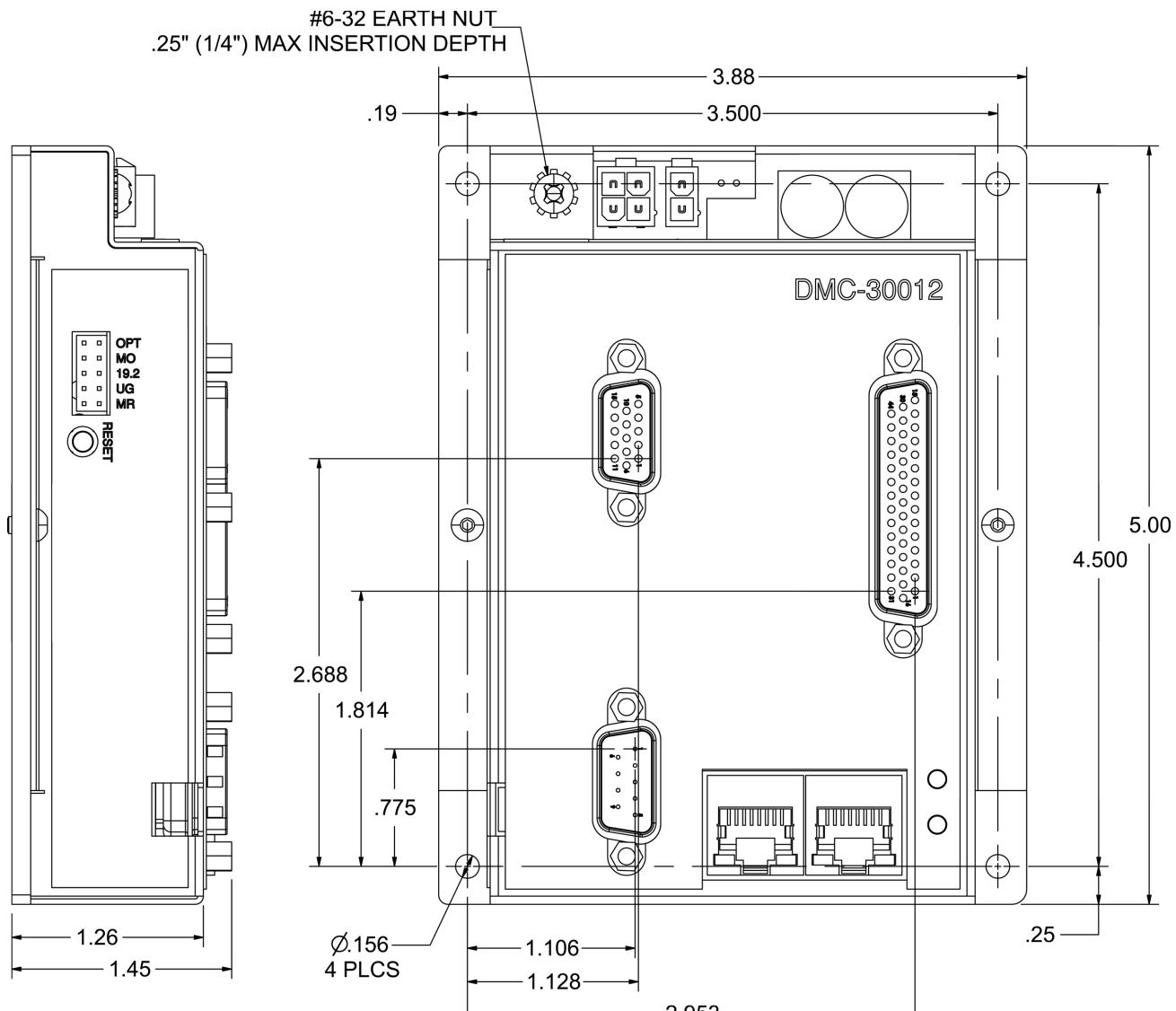


Figure 2.4: DMC-30012-BOX, DMC-30014-BOX, DMC-30016-BOX, and DMC-30017-BOX Dimensions (in inches)

---

## Elements You Need

For a complete system, Galil recommends the following elements:

1. DMC-30000 series motion controller
2. Motor Amplifier (integrated when using Galil amplifier)
3. Power Supply for controller and amplifier
4. Brushed or brushless servo motor with encoder or stepper motor
5. Cables for connecting motor and encoder to DMC-30000
6. PC running Windows 7 or newer (64-bit) or Linux (64-bit)
7. Ethernet cable or straight-through serial cable
8. GDK software

GDK is recommended for first time users of the DMC-30000. It provides interactive instructions for system connection, tuning, and analysis.

# Installing the DMC, Amplifiers, and Motors

Installation of a complete, operational motion control system consists of the following steps:

- |  |                                 |
|--|---------------------------------|
| Step 1. Connecting Encoder Feedback, pg 16                   | <i>Optional for steppers</i>    |
| Step 2a. Wiring Motors to Galil's Internal Amplifiers, pg 17 | <i>Internal Amplifiers only</i> |
| Step 2b. Connecting External Amplifiers and Motors, pg 18    | <i>External Amplifiers only</i> |
| Step 3. Power the Controller, pg 19                          |                                 |
| Step 4. Install the Communications Software, pg 20           |                                 |
| Step 5. Establish Communications with Galil Software, pg 20  |                                 |
| Step 6. Motor Setup with GDK Step-By-Step, pg 20             | <i>Internal Amplifiers only</i> |
| Step 7. Setting Safety Features, pg 21                       |                                 |
| Step 8. Tune the Servo System, pg 22                         | <i>Servo motors only</i>        |

**WARNING**

All wiring procedures and suggestions mentioned in the following sections should be done with the controller in a powered-off state. Failing to do so can cause harm to the user or to the controller.

**NOTE**

The following instructions are given for Galil products only. If wiring an non-Galil device, follow the instructions provided with that product. Galil shall not be liable or responsible for any incidental or consequential damages that occur to a 3<sup>rd</sup> party device.

## Step 1. Connecting Encoder Feedback

The type of feedback the controller is capable of interfacing with depends on the additional options ordered for the controller. Table 2.1 shows the different encoder feedback types available for the DMC-30000 including which options are required. Note that each feedback type has a different configuration command. See the Command Reference for full details on how to properly configure the axis.

<b>NOTE</b>	<a href="#">GDK's Step-By-Step tool</a> provides an interactive walk-through for configuring motor encoders.
-------------	--

Feedback Type	Configuration Command	Part Number Required	Connection Location	Firmware Required
Standard Quadrature	CE	Standard on all units	Encoder	Standard
Pulse and Direction	CE	Standard on all units	Encoder	Standard
Analog <sup>1</sup>	AF	Standard on all units (12-bit Standard, 16-bit optional)	Analog	Standard
SSI	SI	-SER option	I/O	-SER
BiSS	SS	-SER option	I/O	-SER
Sin/Cos, 1 V <sub>pk-pk</sub>	AF	DMC-31000	Encoder	dmc-31000
None <sup>2</sup>	—	—	—	—
Other		Contact Galil		

Table 2.1: Configuration commands, ICM/Part numbers required for a given feedback type

<sup>1</sup> All wiring/electrical information regarding using analog inputs can be found in Analog Inputs, pg 36.

<sup>2</sup> Although stepper systems do not require feedback, Galil supports a feedback sensor on each stepper axis. Servo motors require a position sensor.

Different feedback types can be used on the controller. By default, the axis is configured for standard quadrature.

When a stepper motor is used, the auxiliary encoder for the axis is unavailable for an external connection. If an encoder is used for position feedback with a stepper motor, connect the encoder to the main encoder input corresponding to that axis. The commanded position of the stepper can be interrogated with TD and the encoder position can be interrogated with TP.

See Multi-Function Pins (MF), pg 34 for pin-outs and electrical information for SSI and BiSS options.

Encoder pin-outs can be found here:

Encoder 15 pin HD D-Sub Connector (Female), pg 170

## Step 2a. Wiring Motors to Galil's Internal Amplifiers

If an external amplifier is being used, proceed to Step 2b. Connecting External Amplifiers and Motors. Table 2.2 below provides a general overview of the connections required for connecting different types of motors to Galil internal amplifiers.

Motor Type	Required Connections
Brushless servo motor	<ul style="list-style-type: none"><li>• Power to controller and internal amplifier</li><li>• Motor power leads to internal amplifier</li><li>• Encoder feedback</li><li>• Hall sensors (Optional for sinusoidal amplifier)</li></ul>
Brushed servo motor	<ul style="list-style-type: none"><li>• Power to controller and internal amplifier</li><li>• Motor power leads to internal amplifier</li><li>• Encoder feedback</li></ul>
Stepper motor	<ul style="list-style-type: none"><li>• Power to controller and internal drive</li><li>• Motor power leads to internal drive</li><li>• Encoder feedback (optional)</li></ul>

Table 2.2: Synopsis of connections required to connect a motor to Galil's internal amplifiers

Table 2.3 lists each of Galil's internal amplifiers and where to find pin-outs of the amplifier connections and electrical specifications. The commutation method and whether hall sensors are required are also listed.

Amplifier	Commutation	Hall Sensors Required
A1 – DMC-30012, pg 174	Sinusoidal or Brushed	Optional for brushless motors
A2 – DMC-30014, pg 181	Sinusoidal	Optional for brushless motors
A3 – DMC-30016, pg 187	N/A, Stepper	No
A4 – DMC-30017, pg 191	Sinusoidal, Stepper, or Brushed	Optional for brushless motors

Table 2.3: Amplifier documentation location, commutation method(s), and hall sensor requirements for each internal amplifier.

Pin-outs for the hall sensor inputs can be found here:

Encoder 15 pin HD D-Sub Connector (Female), pg 170

## Step 2b. Connecting External Amplifiers and Motors

System connection procedures will depend on system components and motor type.

Table 2.4 below shows a brief synopsis of the connections required, the full walkthrough guide is provided below.

Motor Type	Connection Requirements
Servo motors (Brushed and Brushless)	<ul style="list-style-type: none"><li>• Power to controller and amplifier</li><li>• Amplifier Enable</li><li>• Motor Command</li><li>• Encoder Feedback</li><li>• See amplifier documentation for motor connections</li></ul>
Stepper motor	<ul style="list-style-type: none"><li>• Power to controller and amplifier</li><li>• Amplifier enable</li><li>• Step and Direction lines</li><li>• Encoder feedback (optional)</li><li>• See amplifier documentation for motor connections</li></ul>

*Table 2.4: Synopsis of connections required to connect an external amplifier*

### Step A. Connect the motor to the amplifier

Initially do so 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 amplifier is operating properly, the motor should stand still even when the amplifier is powered up.

### Step B. Connect the amplifier enable signal

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

<b>WARNING</b>	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 defaulted to 5V, high amp enable (the amplifier enable signal will be high when the controller expects the amplifier to be enabled).

Pin-outs for the amplifier enable signal can be found here:

I/O 44 pin HD D-Sub Connector (Female), pg 169

For full electrical specifications and wiring diagrams refer to:

External Amplifier Interface, pg 37

Once the amplifier enable signal is correctly wired, issuing an MO will disable the amplifier and an SH will enable it.

### **Step C.** Connect the command signals

The DMC-30000 has two ways of controlling amplifiers:

1. Using a motor command line ( $\pm 10V$  analog output). The motor and the amplifier may be configured in torque or velocity mode. In the torque mode, the amplifier gain should be such that a  $10V$  signal generates the maximum required current. In the velocity mode, a command signal of  $10V$  should run the motor at the maximum required speed.
2. Using step (0-5V) and direction (0-5V toggling line), this is referred to as step/dir for short.

The pin-outs for the command signals can be found here:

I/O 44 pin HD D-Sub Connector (Female), pg 169

See External Amplifier Interface, pg 37 for full electrical specifications.

### **Step D.** Issue the appropriate configuration commands

The Motor Type (MT) command is used to configure the axis for a stepper or servo motor. The Configure Encoder (CE) command is used to configure the encoder for the axis.

## **Step 3. Power the Controller**

### **WARNING**

Exercise caution with 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.

Different options may effect which connections and what bus voltages are appropriate. If using an internal amplifier, the ISCNTL – Isolate Controller Power, pg 155 option will require multiple connections, one to power the controller board and another to power the amplifier.

<sup>1</sup> If the ISCNTL option is ordered with an amplifier, the supplies that power the controller and amplifier need to share the same ground.

### **WARNING**

Controller or amplifier power should never be plugged in HOT. Always power down the power supply before installing or removing power connector(s) to/from controller or amplifier.

For more information regarding connector type and part numbers see Power Connector Part Numbers, pg 157. The power specifications for the controller are provided in Input Power Requirements, pg 150 and the power specifications for each amplifier are found under their specific section in the appendix, see Table 2.3.

Any emergency stop or disconnect switches should be installed on the AC input to the DC power supply. Relays and/or other switches should not be installed on the DC line between the Galil and the Power supply. An example system is shown in Figure 2.5 with a DMC-30012-BOX:

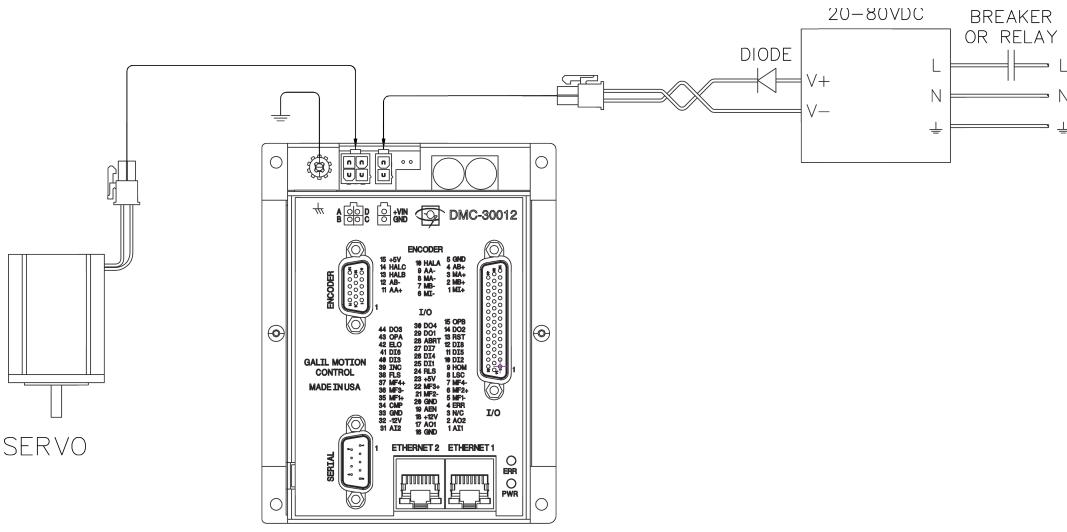


Figure 2.5: Wiring for DMC-30012 with Amplifiers

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

## Step 4. Install the Communications Software

After applying power to the controller, a PC is used for programming. Galil's development software enables communication between the controller and the host device. The current version of Galil's development software can be found here: <http://galil.com/downloads/software/gdk>

## Step 5. Establish Communications with Galil Software

**NOTE**

[GDK](#)'s Step-By-Step tool provides an interactive and easy to follow guide to setting up a Windows PC for communication with Galil controllers.

GDK will automatically setup the computer for serial communication. For Ethernet communication, see this video for how to configure your NIC card using Windows to connect to a DMC controller:

<http://galil.com/learn/online-videos/basics-galil-design-kit-manager-terminal-and-editor>

See the GDK manual for using the software to communicate: <http://galil.com/sw/pub/all/doc/gdk/man/>

## Step 6. Motor Setup with GDK Step-By-Step

**NOTE**

[GDK](#)'s Step-By-Step tool provides an interactive guide for the automatic set up of brushed, brushless, and stepper motors when using internal amplifiers.

For manual setup instructions, refer to the specific amplifier appendix section which were listed in Table 2.3.

## Step 7. Setting Safety Features

### Step A. Setting Amplifier Gains

A transconductance (current) amplifier takes a  $\pm 10V$  command signal and produces a current in the motor proportional to that signal. This ratio is the amplifier gain (amps/volts) and can be set via the Amplifier Gain ( $AG$ ) command when using Galil's internal amplifiers. See the  $AG$  command in the Command Reference for proper current gain settings for Galil's internal amplifiers.

The Current Loop Gain ( $AU$ ) command sets the current loop gain for the amplifier and is set based on the bus voltage powering the amplifier as well as the inductance of the motor. Consult the amplifier appendix section or Command Reference for more details.

### Step B. Setting Torque Limits

The Torque Limit ( $TL$ ) command will limit the output voltage of  $\pm 10V$  motor command line. This command should be used to avoid excessive torque and speed when initially setting up a servo system. The Peak Torque Limit ( $TK$ ) command sets the peak torque limit of the motor output allowing the command line to momentarily exceed  $TL$ . See the  $TL$  and  $TK$  settings in the Command Reference for more information. Amplifier gain needs to be taken into account when setting both  $TL$  and  $TK$ . For example, if a particular motor has a continuous current rating of 2.0 A and a peak current rating of 5.0 A and the amplifier gain is 0.8A/V,  $TL$  and  $TK$  can be calculated as follows:

$$TL \text{ setting} = (2.0\text{A})/(0.8\text{A/V}) = 2.5\text{V} \quad (TL \text{ n}=2.5)$$

$$TK \text{ setting} = (5.0\text{A})/(0.8\text{A/V}) = 6.25\text{V} \quad (TK \text{ n}=6.25)$$

The user is responsible for determining the relationship between the motor command line and the amplifier torque/velocity using the documentation of the motor and/or amplifier.

### Step C. Setting Error Limits

When an Error Limit ( $ER$ ) and Off-on-Error ( $OE$ ) is set, the controller will automatically shut down the motors when excess error ( $|TE| > ER$ ) has occurred.

$OE$  requires the amplifier enable signal to be connected from the controller to the amplifier. When using Galil's internal amplifiers, the amplifier enable is already configured. See Step 2b. Connecting External Amplifiers and Motors, pg 18 when using external amplifiers.

### Step D. Other Safety Features

This section only provides a brief list of safety features that the DMC can provide. Other features include Encoder Failure Detection (OA, OT, OV), Automatic Subroutines to create an automated response to events such as limit switches toggling (#LIMSWI), large following error (#POSERR), amplifier errors (TA, #AMPERR), and more. For a full list of features and how to program each see Chapter 8 Hardware & Software Protection, pg 130.

## **Step 8. Tune the Servo System**

Adjusting the tuning parameters is required when using a servo motor. The controller's default set of PID's are not optimized and should not be used in practice.

For the theory of operation and a full explanation of all the PID and other filter parameters, see Chapter 10 Theory of Operation, pg 137.

For additional tuning resources and guides, see the following Application Notes:

Manual Tuning Methods: <http://www.galil.com/download/application-note/note3413.pdf>

Manual Tuning using the Velocity Zone method: <http://www.galil.com/download/application-note/note5491.pdf>

For information about the Autotuning Tool in GDK: <http://galil.com/sw/pub/all/doc/gdk/man/tuner.html>

# Chapter 3 Connecting Hardware

---

## Overview

The DMC-30000 provides optoisolated digital inputs for forward limit, reverse limit, home, and abort signals. The controller also has 8 optoisolated uncommitted inputs, 4 optoisolated outputs, 2 analog inputs (0-5V, 12-bit ADC) and 1 uncommitted analog output ( $\pm 10V$ , 16-bit DAC).

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

Pin-out information can be found in the Pin-outs section in the Appendices.

---

## Overview of Optoisolated Inputs

### Limit Switch Input

The forward limit switch (FLS) inhibits motion in the forward direction immediately upon activation of the switch. The reverse limit switch (RLS) 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. The controller can be configured to disable the axis upon the activation of a limit switch, see the OE command in the Command Reference for further detail.

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. Any attempt at further motion before the logic state has been reset will result in the following error: "22 - Begin not possible due to limit switch" error.

The operands, \_LFA and \_LRA, contain the state of the forward and reverse limit switch respectively. 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\_LFA or MG\_LRA. 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-30000: Find Edge (FE), Find Index (FI), and Standard Home (HM).

The Find Edge routine is initiated by the command sequence: FEX, BGX. 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, BGX. 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, BGX. 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 (OE Command). If the Off-On-Error function is enabled the motor 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 individual sections in the Appendices.

## **Reset Input/Reset Button**

When the Reset line is triggered the controller will be reset. The reset line and reset button will not Master Reset the controller unless the MRST jumper is installed during a controller reset.

## **Uncommitted Digital Inputs**

The DMC-30000 has 8 optoisolated 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 motor move 1000 counts in the positive direction when the logic state of DI1 goes high.

Digital Input 1 can be used has a high speed position latch, see High Speed Position Capture (The Latch Function) for more information.

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

# Optoisolated Input Electrical Information

## Electrical Specifications

INC/LSC Max Voltage	24 V <sub>DC</sub>
INC/LSC Min Voltage	0 V <sub>DC</sub>
Internal resistance of inputs	2.2 kΩ

The current limiting resistors chosen for the inputs allow for the flexibility to have 5-24V optoisolated inputs. Because of this added range, applications powering the inputs with 18.5 V<sub>DC</sub> or greater with normally closed switches may need an additional resistor to limit the current to the inputs, see the Input Current Limitations, pg 167 for more details or simply place a 1.2kΩ resistor in series with INCOM/LSCOM as shown in Figure A.5 in the 167.

The optoisolated inputs are powered in banks. For example, INC, located on the 44-pin I/O D-sub connector, provides power to DI[8:1] (digital inputs), the abort input (ABRT), reset (RST), and electric lock-out (ELO). Table 3.1 shows all the input banks power commons and their corresponding inputs.

Common Signal	Common Signal Location	Powers Inputs Labeled
INC	I/O D-Sub Connector	DI[8:1], ABRT, RST, ELO
LSC	I/O D-Sub Connector	FLS, RLS, HOM

Table 3.1: 1-4 axis controller INCOM and LSCOM banks and corresponding inputs powered

The full pin-outs for each bank can be found in the I/O 44 pin HD D-Sub Connector (Female), pg 169.

## Wiring the Optoisolated Digital Inputs

To take full advantage of optoisolation, an isolated power supply should be used to provide the voltage at the input common connection. Connecting the ground of the isolated power to the ground of the controller will bypass optoisolation and is not recommended if true optoisolation is desired.

If there is not an isolated supply available, the 5 V<sub>DC</sub>, 12 V<sub>DC</sub>, and GND controller references may be used to power INC/LSC. The current supplied by the controller references are limited, see +5, ±12V Power Output Specifications, pg 150 in the Appendices for electrical specifications. Using the controller reference power completely bypasses optoisolation and is not recommended for most applications.

The optoisolators are bidirectional, meaning current can flow through them in either direction. Connecting +Vs to INCOM/LSCOM will cause current to flow out of the limit switch pin when it is pulled to the isolated ground. Connecting the isolated ground to INCOM/LSCOM will cause current to flow into the limit switch pin when it is pulled up to +Vs. Regardless of the direction of current flow, the input is active low – the CPU will read a logical 0 when current is flowing and a logical 1 when no current is flowing.

Figure 3.1 and Figure 3.2 below illustrate the two possible wiring configurations for the limit switches.

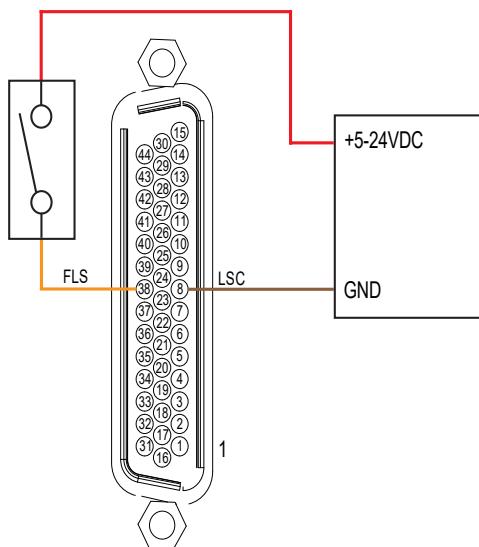


Figure 3.1: Wiring limit switch for sinking current

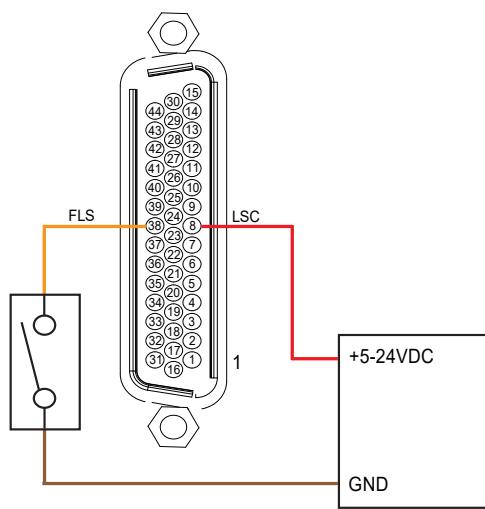


Figure 3.2: Wiring limit switch for sourcing current

Table 3.2 below shows the limit and home switch connections

44-pin Female D-sub		
Pin	Label	Description
8	LSC	Limit switch common
9	HOM	Home switch input
24	RLS	Reverse limit switch input
38	FLS	Forward limit switch input

Table 3.2: Limit and home switch inputs

Figure 3.3 - Figure 3.5 are the optoisolated wiring diagrams for powering INC/LSC.

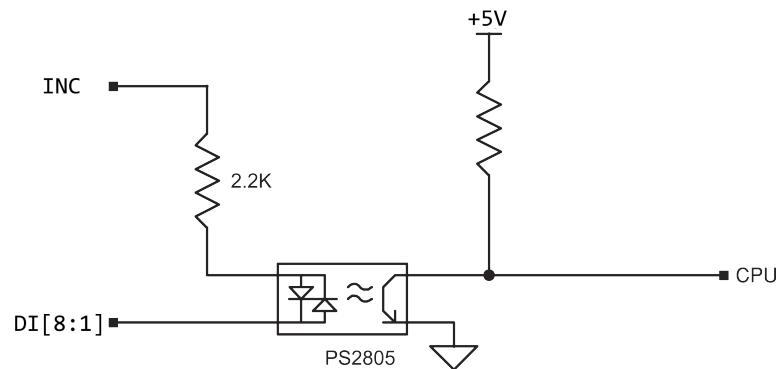


Figure 3.3: Uncommitted digital inputs on the DMC-30000

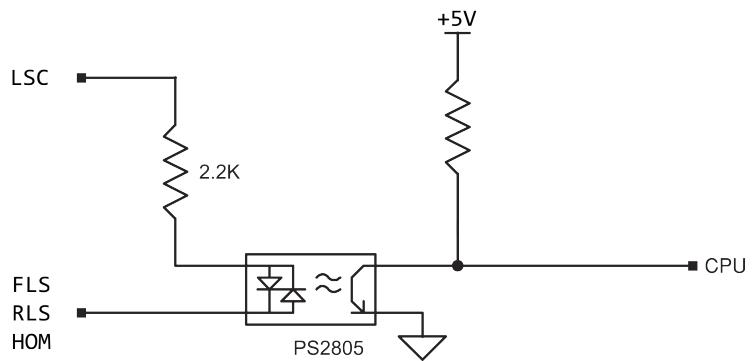


Figure 3.4: Limit Switch Inputs on the DMC-30000

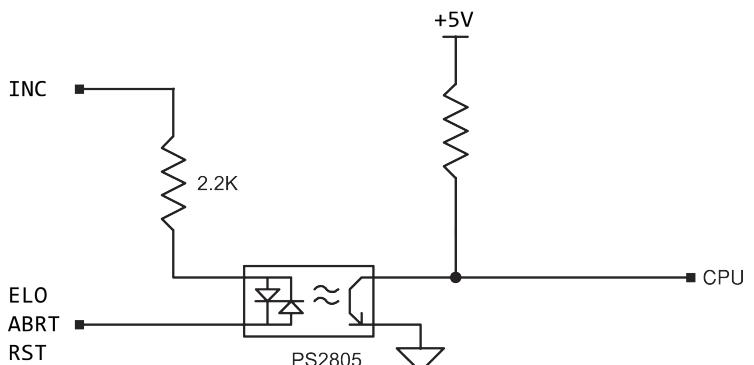


Figure 3.5: ELO, Abort and Reset Inputs on the DMC-30000

---

## Optoisolated Outputs

The DMC-30000 has several different options for the uncommitted digital outputs (labeled as DO). The default outputs are 4mA sinking which are ideal for interfacing to TTL level devices. Additional options include 25mA sinking (lower power sinking, LSNK) and 500mA sourcing (high power sourcing, HSRC). Please refer to your part number to determine which option you have.

The DMC-30000 has only has a single bank (Bank 0) of 4 optoisolated outputs, powered through the Output PWR and Output GND pins located on I/O 44 pin HD D-Sub Connector (Female). Please see the Pin-outs in the Appendix for pin-outs.

Wiring diagrams, electrical specifications, and details for each output type are provided below.

### Brake Output

When using the brake outputs, it is recommended to order the controller with 500mA sourcing output option (HSRC).

Outputs 1 is the brake output.

The BW command sets the delay between when the brake is turned on and when the amp is turned off. When the controller goes into a motor-off (MO) state, this is the time (in samples) between when the brake digital output changes state and when the amp enable digital output changes state. The brake is actuated immediately upon MO and the delay is to account for the time it takes for the brake to engage mechanically once it is energized electrically. The brake is released immediately upon SH.

See the BW command in the DMC-30000 Command Reference for more information.

## Standard 4mA Sinking Optoisolated Outputs

### Description

The default outputs of the DMC-30000 are capable of 4mA and are configured as sinking outputs. The voltage range for the outputs is 5-24 VDC. These outputs should not be used to drive inductive loads directly.

### Electrical Specifications

Output PWR Max Voltage	24 VDC
Output PWR Min Voltage	5 VDC
ON Voltage (No Load, Output PWR= 5VDC)	0.1 VDC
Max Drive Current per Output	4mA – Sinking

### Wiring the Standard 4mA outputs

With this configuration, the output power supply will be connected to Output PWR (labeled OPB) and the power supply return will be connected to Output GND (labeled OPA). Note that the load is wired between Output PWR and DO. The wiring diagram for Bank 0 is shown in Figure 3.6. Refer to Pin-outs in the Appendix for pin-out information.

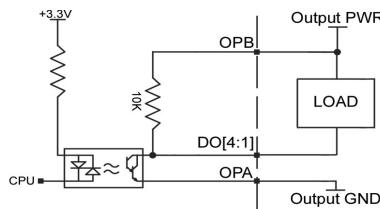


Figure 3.6: 4mA sinking wiring diagram for Bank 0, DO [4:1]

## 25mA Sinking Optoisolated Outputs (LSNK)

### Description

The 25mA sinking option, referred to as lower power sinking (LSNK), are capable of sinking up to 25mA per output. The voltage range for the outputs is 5-24 VDC. These outputs should not be used to drive inductive loads directly.

### Electrical Specifications

Output PWR Max Voltage	24 VDC
Output PWR Min Voltage	5 VDC
ON Voltage (No Load, Output PWR= 5 VDC)	1.2 VDC
Max Drive Current per Output	25mA, Sinking

### Wiring the 25mA Sinking Outputs

The 25mA sinking outputs the load is wired in the same fashion as the 4mA sinking outputs: The output power supply will be connected to Output PWR (labeled OPB) and the power supply return will be connected to Output

GND (labeled OPA). Note that the load is wired between Output PWR and DO. The wiring diagram for Bank 0 is shown in Figure 3.7. Refer to Pin-outs in the Appendix for pin-out information.

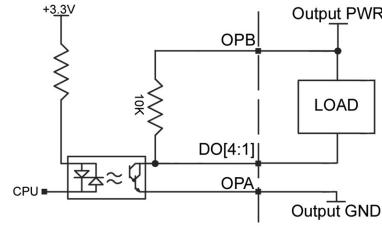


Figure 3.7: 25mA sinking wiring diagram for Bank 0, DO[4:1]

## 500mA Sourcing Optoisolated Outputs (HSRC)

### Description

The 500mA sourcing option, referred to as high power sourcing (HSRC), is capable of sourcing up to 500mA per output and up to 1.5 A per bank. The voltage range for the outputs is 12-24 VDC. These outputs are capable of driving inductive loads such as solenoids or relays. The outputs are configured for hi-side (sourcing).

### Electrical Specifications

Output PWR Max Voltage	24 VDC
Output PWR Min Voltage	12 VDC
Max Drive Current per Output	0.5 A (not to exceed 1.5 A for all 4 outputs)

### Wiring the 500mA Sourcing Optoisolated Outputs

With this configuration, the output power supply will be connected to Output PWR (labeled OPA) and the power supply return will be connected to Output GND (labeled OPB). Note that the load is wired between DO and Output GND. The wiring diagram for Bank 0 is shown in Figure 3.8. Refer to Pin-outs in the Appendix for pin-out information.

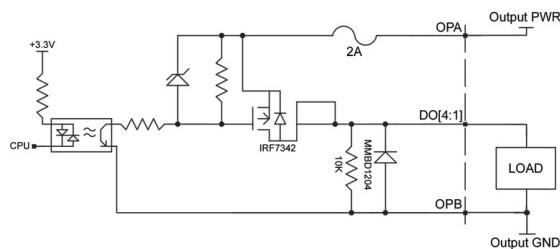


Figure 3.8: 500mA sourcing wiring diagrams for Bank 0, DO[4:1]

# Feedback Inputs and Multi-Function (MF) Pins

## Feedback Options

There are many different options for feedback with the DMC-30000 series controllers. Table 3.3 indicates which feedback options are available with each configuration, and the inputs for those feedback options.

DMC-30000 Feedback Options	DMC-3001x	DMC-3001x-SER	DMC-3101x	DMC-3101x-SER
Main Digital Encoder	MA/MB	MA/MB	MA/MB	MA/MB
Aux Digital Encoder	AA/AB	AA/AB	AA/AB	AA/AB
Analog Feedback (0-5V)	AI1	AI1	AI1 (AQ)	AI1 (AQ)
Analog Feedback (16 bit configurable +/-10V)	-	-	AI1 (AQ)	AI1 (AQ)
SSI/BiSS Channel 0	-	MF 1,2 – Main	-	MF 1,2 – Main
SSI/BiSS Channel 1	-	MF 3,4 – Aux	-	MF 3,4 – Aux

Table 3.3: Feedback options for DMC-30000 series controllers

- MA/MB are the Main Encoder inputs
- AA/AB are the Aux Encoder Inputs
- AI 1 is Analog Input 1
- MF 1,2 are Multi-function Inputs 1 and 2
- MF 3,4 are Multi-function Inputs 3 and 4

## Main Encoder Inputs

The main encoder inputs can be configured for quadrature (default) or pulse and direction inputs. This configuration is set through the CE command. The encoder connections are found on the 15 pin HD D-sub Encoder connectors and are labeled MA+, MA-, MB+, MB-. The '-' (negative) inputs are the differential inputs to the encoder inputs; if the encoder is a single ended 5V encoder, then the negative input should be left floating (except for the DMC-31000; see A5 – DMC-31000 for details). If the encoder is a single ended and outputs a 0-12V signal then the negative input should be tied to the 5V line on the DMC-30000.

When the encoders are setup as step and direction inputs the MA channel will be the step or pulse input, and the MB channel will be the direction input.

The encoder inputs can be ordered with  $120\ \Omega$  termination resistors installed. See TRES – Encoder Termination Resistors in the Appendix for more information.

## Electrical Specifications

Maximum Voltage	12 VDC
Minimum Voltage	-12 VDC
Maximum Frequency (Quadrature)	15 MHz
'+' inputs are internally pulled-up to 5V through a $4.7\ k\Omega$ resistor	
'-' inputs are internally biased to $\sim 1.3V$	
pulled up to 5V through a $7.1\ k\Omega$ resistor	
pulled down to GND through a $2.5\ k\Omega$ resistor	

## The Auxiliary Encoder Inputs

The auxiliary encoder inputs can be used for general use. The controller has one auxiliary encoder which consists of two inputs, channel A and channel B. The auxiliary encoder inputs are mapped to the inputs 81 and 82. The Aux encoder inputs are not available when the controller is configured for step and direction outputs (stepper).

Each input from the auxiliary encoder is a differential line receiver and can accept voltage levels between  $\pm 12$  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  $\sim \frac{1}{2}$  of the full voltage range (for example, connect the '-' input to the 5 volts on the Galil if the signal is 0 - 12V logic).

### Electrical Specifications

Maximum Voltage	12 VDC
Minimum Voltage	-12 VDC
'+' inputs are internally pulled-up to 5V through a $4.7\text{k}\Omega$ resistor	
'-' inputs are internally biased to $\sim 1.3\text{V}$	
	pulled up to 5V through a $7.1\text{k}\Omega$ resistor
	pulled down to GND through a $2.5\text{k}\Omega$ resistor

## Multi-Function Pins (MF)

### Multi-Functional Pins (MF ±n)

The Multi-Functional Pins on the DMC-30000 have different functionalities dependent upon how the controller was ordered and how the controller is setup by the user. If the controller is ordered with -SER (serial encoder interface), then the MF pins can be used to interface to a serial encoder (SSI and BiSS). MF1 and MF2 are only used for the Main serial encoder input, MF3 and MF4 are used for the Aux serial encoder input. See the SI and SS commands in the Command Reference for more detail.

When the controller is setup for stepper motor operation, the MF 2 and MF4 pins are used for step and direction respectively.

Label	Pin #	MT ±2 or ±2.5	-SER option with SSI or BiSS Enabled
MF1 +	35	No Connect	Main Encoder Data +
MF1 -	5	No Connect	Main Encoder Data -
MF2 +	6	STEP +	Main Encoder Clock +
MF2 -	21	STEP -	Main Encoder Clock -
MF3 +	22	No Connect	Aux Encoder Data +
MF3 -	36	No Connect	Aux Encoder Data -
MF4 +	37	DIR +	Aux Encoder Clock +
MF4 -	7	DIR -	Aux Encoder Clock -

**NOTE:** The -SER option cannot be used when controlling an external drive with Step/Dir. For more information, contact Support at Galil.

Table 3.4: Signal Description for Multi-Functional Pins

### Electrical Specifications (MF2, MF4)

Output Voltage	0 – 5 VDC <sup>1</sup>
Current Output	20 mA Sink/Source

<sup>1</sup> DMC-31000 units shipped before February 2013 and DMC-30000 units shipped before April 2013 have a different specification as shown below:

Output Voltage	0 – 3.3 VDC
Current Output	4 mA Sink/Source

### Electrical Specifications (MF1, MF3)

Maximum Input Voltage	5 VDC
Minimum Input Voltage	0 VDC

---

## TTL Outputs

### Output Compare

The output compare signal is a TTL output signal and is available on the I/O D-Sub connector labeled as CMP. Output compare is controlled by the position of the main encoder input on the controller.

Output compare is controlled by the position of any of the main encoder inputs on the controller. The output can be programmed to produce either a brief, active low pulse (510 nsec) based on an incremental encoder value or to activate once ("one shot") when an axis position has been passed. When setup for a one shot, the output will stay low until the OC command is called again. For further information, see the command OC in the Command Reference.

<b>NOTE</b>	Output compare is not valid with sampled feedback types such as: SSI, BiSS, Sin/Cos, and Analog
-------------	---

### Electrical Specifications

Output Voltage	0 – 5 VDC
Current Output	20 mA Sink/Source

### 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.

For additional information see Error Light (Red LED) in Chapter 9 Troubleshooting.

### Electrical Specifications

Output Voltage	0 – 5 VDC
Current Output	20 mA Sink/Source

# Analog Inputs

## DMC-30000

The DMC-30000 has two 0-5V analog inputs. The inputs are decoded by a 12-bit A/D decoder giving a voltage resolution of approximately 1.22mV. The analog inputs are specified as AN[x] where x is a number 1 or 2.

The analog inputs can be set to a differential mode where analog input 2 is the differential input to analog input 1.

### Electrical Specifications

Maximum Voltage	5V
Minimum Voltage	0V
Resolution	12-bit
Input Impedance	100 kΩ

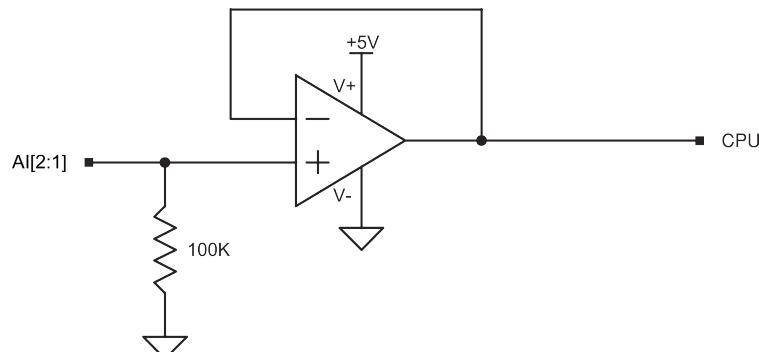


Figure 3.9: Analog Inputs for the DMC-30000

## DMC-31000

The DMC-31000 has two analog inputs configured for the range between -10V and 10V. The inputs are decoded by a 16-bit A/D decoder giving a voltage resolution of approximately .0003V. The analog inputs are specified as AN[x] where x is a number 1 thru 2.

### AQ settings

With the DMC-31000, the analog inputs can be set to a range of  $\pm 10V$ ,  $\pm 5V$ , 0-5V or 0-10V, this allows for increased resolution when the full  $\pm 10V$  is not required. The inputs can also be set into a differential mode where analog input 2 can be set to the negative differential input for analog input 1. See the AQ command in the Command Reference for more information.

### Electrical Specifications

Resolution	16-bit
Input Impedance –	
Unipolar (0-5V, 0-10V)	42kΩ
Bipolar ( $\pm 5V$ , $\pm 10V$ )	31kΩ

---

## Analog Outputs

Analog output 1 (AO1) has different functions depending on the option ordered. AO1 functions as the motor command line for DMC-30010 and DMC-30011 models and cannot be used as a general analog output. For DMC-30000 models ordered with internal amplifiers, AO1 can be configured to work as either a motor command line for an external amplifier or as a general analog output, see [Configuring AO1 with Internal Amplifiers](#) below for more details.

Analog output 2 (AO2) is always available as a general use analog output.

### Electrical Specifications

Output Voltage Range	$\pm 10V$
Resolution	16-bit
Maximum Current Output	4mA (sink/source)
Output Impedance	500 $\Omega$

### Configuring AO1 with Internal Amplifiers

#### Using AO1 as A Motor Command Line

With the controller set to servo mode (MT 1 or -1) to drive an external servo amplifier, the BR command must be set to a -1. This setting will disable the requirement to have the BA, BM and BX or BZ commands executed prior to being able to issue the SH command for that axis.

#### Using AO1 as a General Purpose Output

With internal amplifiers, AO1 can be used as a general purpose output. The analog output can be set with the AO command once MT is set for a servo (or 2PB motor) and the BA command is set for the A axis (BA A). Do not change the MT setting once the analog output is set using the AO command. For additional information, refer to the AO and MT commands in the [DMC-30000 Command Reference](#).

---

## External Amplifier Interface

### Overview

The DMC-30000 provides both amplifier enable and control signals to external amplifiers. For servo motors, it provides a  $\pm 10V$  analog output called the motor command line. For stepper motors, various pulse and direction signal types are provided depending on the MT setting—these are called the Step/Direction Lines. The amplifier enable signal is a simple digital signal that toggles the amplifier “on” or “off.” A summary of these features on the DMC-30000 is provided below.

### Motor Command Line

AO1 can either act as the motor command line or as a general output depending on the option ordered, see [Configuring AO1 with Internal Amplifiers](#) under the Analog Outputs section, pg 37 for more details.

The DMC-30000 motor command voltage ranges between  $\pm 10V$  and is output on the motor command line (AO1). 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.

### **Electrical Specifications**

See Analog Outputs, Electrical Specifications, pg 37.

### **Step/Direction Lines**

With the controller set to stepper mode (MT -2, 2, 2.5 or -2.5), the step and direction outputs are found on the I/O connector list as MF2 and MF4.

### **Electrical Specifications**

Details and pin-out information can be found in the Multi-Function Pins (MF) section of Chapter 3.

### **Amplifier Enable**

The DMC-30000 provides an amplifier enable signal, labeled AEN. This signal toggles under the following conditions: The MO “motor off” command is given, the watchdog timer activates, or a particular set of conditions, controlled by the OE “off on error” command occurs.

The AEN enable signal is 5V active high amp enable (HAEN). In other words, the AEN signal will be high when the controller expects the amplifier to be enabled.

If higher voltages or optoisolation is required, the sixth argument of the “CN” command will configure the unit to use output 4 as the amplifier enable signal in lieu of AEN. The electrical specifications and wiring of output 4 is dependent on the outputs ordered.

**NOTE:** Many amplifiers designate the enable input as ‘inhibit’.

### **Electrical Specifications**

#### **AEN - Default**

Max Amplifier Enable Voltage	5V
Max Amplifier Enable Current	sink/source      20 mA

#### **Output 4 Specifications**

Optoisolated Outputs, pg 29 and CN in the Command Reference for details.

# Chapter 4 Software Tools and Communication

---

## Introduction

The default configuration DMC-30000 has one RS-232 port and two Ethernet ports. The RS-232 port baud rate defaults to 115200 bps and can be configured for 19200 bps via jumpers on the side of the controller. The Ethernet ports are 100BASE-T connections that auto-negotiate half or full duplex.

The Galil Design Kit (GDK) software package is available for PC computers running Microsoft Windows or Linux to communicate with the DMC-30000 controller. This software package has been developed to operate under Windows and Linux, and include all the necessary drivers to communicate to the controller. In addition, GDK 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, GDK is the basic program 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 gclib 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.

---

## Controller Response to Commands

Most DMC-30000 instructions are represented by two characters followed by the appropriate parameters. Each instruction must be terminated by a carriage return. Multiple commands may be concatenated by inserting a semicolon between each command.

After the instruction is decoded, the DMC-30000 returns a response to the port from which the command was generated. If the instruction was valid, the controller returns a colon (:) or the controller will respond with a question mark (?) if the instruction was not valid. For example, the controller will respond to commands which are sent via the 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-30000 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-30000 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 RS-232 port or Ethernet handles. 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 serial port. When communicating via an Ethernet connection, the unsolicited messages must be sent through a handle that is not the main communication handle from the host. The Galil Design Kit (GDK) software automatically establishes this second communication handle.

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.

---

## Serial Communication Port

### RS-232 Configuration

The pin-outs and cabling advice for the RS-232 ports can be found in the Serial Cable Specifications, pg 165.

### Configuration

The Galil Design Kit (GDK) software will automatically configure your PC for 8-bit data, one start-bit, one stop-bit, full duplex and no parity. The baud rate for the RS-232 communication can be selected by setting the proper switch configuration on the front panel according to the table below.

#### Baud Rate Selection

JP1 19.2 JUMPER	BAUD RATE
ON	19200
OFF (recommended)	115200

### Handshaking

The RS-232 main port is set for hardware handshaking. Hardware Handshaking uses the RTS and CTS lines. The CTS line will go high whenever the DMC-30000 is not ready to receive additional characters. The RTS line will inhibit the DMC-30000 from sending additional characters. Note, the RTS line goes high for inhibit.

## RS-422 Configuration

The DMC-30000 can be ordered with the serial 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.

See DMC-30000 RS-422 Serial Port Specifications, pg 166 for pin-outs and details of the RS-422 options.

---

## 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-30000 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 Client connects to the Server through a series of packet handshakes in order to begin communicating. Each packet sent is acknowledged when received. If no acknowledgment is received, the information is assumed lost and is resent.

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

It is recommended that the motion control network containing the controller and any other related devices is placed on a "closed" network. If this recommendation is followed, UDP/IP communication to the controller may be utilized instead of a TCP connection. With UDP there is less overhead, resulting in higher throughput. Also, there is no need to reconnect to the controller with a UDP connection. Because handshaking is built into the Galil communication protocol through the use of colon or question mark responses to commands sent to the controller, the TCP handshaking is not required.

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, the user must wait for the controller's response 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-30000 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-30000 unit, use the TH command. A sample is shown here with a unit that has a serial number of 11:

Sample MAC Ethernet Address: 00-50-4C-40-00-0B

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-30000 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-30000 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-30000'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.

<b>CAUTION</b>	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 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 DHO 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-30000 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 each time it connects to the DMC-30000 board. Typical port numbers for applications are:

Port 23: Telnet

Port 502: Modbus

## Communicating with Multiple Devices

The DMC-30000 is capable of supporting multiple Clients and Servers. The Clients may be multiple PC's that send commands to the controller. The Servers are typically peripheral I/O devices that receive commands from the controller.

An Ethernet handle is a communication resource within a device. The DMC-30000 can have a maximum of 6 Ethernet handles open at any time. When using TCP/IP, each Client or Server uses an individual Ethernet handle. In UDP/IP, one handle may be used for all the Clients, but each Server uses one. (Pings and ARPs do not occupy handles.) If all 6 handles are in use and a 7<sup>th</sup> Client 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 RS-232 by entering RS).

When the Galil controller acts as the Client, the IH command is used to assign handles and connect to its Servers. 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 Server. (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-30000 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 Server has a 1 byte Server address. The DMC-30000 can use a specific Server address or default to the handle number. The port number for Modbus is 502. The DMC-30000 can fill the role of either a Modbus Client or Modbus Server.

The Modbus protocol has a set of commands called function codes. The DMC-30000 supports the 10 major function codes as a Modbus Client:

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 Server ID

The DMC-30000 supports 2 major function codes as a Modbus Server:

Function Code	Definition
03	Read Holding Registers (Read Words)
16	Preset Multiple Registers (Write Words)

A Modbus Client has the ability to read and write array data on the DMC-30000<sup>1</sup> acting as a Server (up to 1000 elements are available). Each element is accessible as a 16-bit unsigned integer (Modbus register 1xxx) or as a 32-bit floating point number (Modbus registers 2xxx). The ability to write to the Servers array table is enabled by setting the ME command (ME is not needed to read array data). See the DMC-30000 Command Reference for further details.

<sup>1</sup> Only DMC-30000 firmware revisions Rev 1.2a and later support Modbus Server capability via the ME command.

The DMC-30000 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 Server 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 Server address is not necessary, the I/O number to be used can be calculated with the following:

$$\text{I/O Number} = (\text{HandleNum} * 1000) + ((\text{Module}-1)*4) + (\text{BitNum}-1)$$

Where HandleNum is the handle number from 1 (A) to 8 (8). 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.

## Modbus Examples

### Example #1

DMC-30000 connected as a Modbus Client to a RIO-47120 via Modbus. The DMC-30000 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>2 (Issued to DMC-30000)

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 01010101 in 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-30000 connected as a Modbus Client to a 3rd party PLC. The DMC-30000 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

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-30000 connected as a Modbus Client to a hydraulic pump. The DMC-30000 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
```

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 List of Other Publications in the Appendices.

---

## Data Record

The DMC-30000 can provide a binary block of status information with the use of the QR and DR commands. These commands along with the QZ command can be very useful for accessing complete controller status. The following is the byte map for the binary data. See the QR, QZ and DR command for specific command usage information.

Data Record Map Key	
Acronym	Meaning
UB	Unsigned byte
UW	Unsigned word
SW	Signed word
SL	Signed long
UL	Unsigned long

ADDR	TYPE	ITEM
00	UB	1 <sup>st</sup> Byte of Header
01	UB	2 <sup>nd</sup> Byte of Header
02	UB	3 <sup>rd</sup> Byte of Header
03	UB	4 <sup>th</sup> Byte of Header
04-05	UW	sample number
06-07	UW	general input block 0 (inputs 1-16) <sup>1</sup>
08-09	UW	general output block 0 (outputs 1-16) <sup>1</sup>
10	UB	error code
11	UB	thread status – see bit field map below
12-13	UW	analog input 2
14-15	UW	analog output 1
16-17	UW	analog output 2
18-21	UL	amplifier status
22-25	UL	Segment Count for Contour Mode
26-27	UW	Buffer space remaining – Contour Mode
28-29	UW	segment count of coordinated move for S plane
30-31	UW	coordinated move status for S plane – see bit field map below
32-35	SL	distance traveled in coordinated move for S plane
36-37	UW	Buffer space remaining – S Plane
38-39	UW	A axis status – see bit field map below
40	UB	A axis switches – see bit field map below
41	UB	A axis stop code
42-45	SL	A axis reference position
46-49	SL	A axis motor position
50-53	SL	A axis position error
54-57	SL	A axis auxiliary position
58-61	SL	A axis velocity
62-65	SL	A axis torque
66-67	UW	analog input 1
68	UB	A Hall Input Status
69	UB	Reserved
70-73	SL	A User defined variable (ZA)

<sup>1</sup> Not all I/O shown in the data record are available on the standard DMC-30000 controller.

## 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	N/A	S Block Present in Data Record
BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
N/A	N/A	N/A	N/A	N/A	N/A	N/A	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 (reversed network byte order).

### Thread Status (1 Byte)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
N/A	N/A	Thread 5 Running	Thread 4 Running	Thread 3 Running	Thread 2 Running	Thread 1 Running	Thread 0 Running

### Coordinated Motion Status (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	1 <sup>st</sup> Phase of HM complete	2 <sup>nd</sup> 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 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

### Amplifier Status (4 Bytes)

BIT 31	BIT 30	BIT 29	BIT 28	BIT 27	BIT 26	BIT 25	BIT 24
N/A	ELO Active						

BIT 23	BIT 22	BIT 21	BIT 20	BIT 19	BIT 18	BIT 17	BIT 16
N/A	Peak current						

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
N/A	N/A	N/A	N/A	N/A	N/A	N/A	Hall Error

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
N/A	N/A	N/A	N/A	Under Voltage	Over Temp.	Over Voltage	Over Current

### 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  $\pm 32767$ . 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' for the DMC-30000)
1	number of bytes in general block of data record (18 for the DMC-30000)
2	number of bytes in coordinate plane block of data record (16 for the DMC-30000)
3	number of bytes the axis block of data record (36 for the DMC-30000)

---

## Galil Software

Galil provides a variety of software tools available to make communication and configuration easier for the user. Galil's latest generation software is available on the Galil website at:

<http://galil.com/downloads/software>

---

## Creating Custom Software Interfaces

Galil provides programming tools so that users can develop their own custom software interfaces to a Galil controller. For new applications, Galil recommends the current generation communication libraries located on the Galil Website:

<http://galil.com/downloads/api>

Please visit the API examples page under the Learn section for details on getting started developing custom software interfaces for Galil controllers:

<http://galil.com/learn/api-examples>

# Chapter 5 Command Basics

---

## Introduction

The DMC-30000 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 are sent in ASCII.

The DMC-30000 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.

Commands can be sent "live" over the communications port for immediate execution by the DMC-30000, or an entire group of commands can be downloaded into the DMC-30000 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.

This section describes the DMC-30000 instruction set and syntax. A summary of commands as well as a complete listing of all DMC-30000 instructions is included in the *Command Reference*.

[www.galil.com/downloads/manuals-and-data-sheets](http://www.galil.com/downloads/manuals-and-data-sheets)

---

## Command Syntax - ASCII

DMC-30000 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-30000 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.

<b>IMPORTANT</b>	All DMC-30000 commands are sent in upper case.
------------------	--

For example, the command

PR 4000 <return> Position relative

## Implicit Notation

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.

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

PR 1000	Specify a relative move of 1000
PR ?	Request relative move value

## Explicit Notation

The DMC-30000 provides an alternative method for specifying data. Here data is specified individually using the single axis specifier A. 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
ACA= 200000	Specify acceleration as 200000

---

## Controller Response to DATA

The DMC-30000 returns a : for valid commands and a ? for invalid commands.

For example, if the command BG is sent in lower case, the DMC-30000 will return a ?.

:bg	invalid command, lower case
?	DMC-30000 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	Tell Code command
1 Unrecognized command	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-30000 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 Application Programming and the Command Reference.

## Summary of Interrogation Commands

RP	Report Command Position
RL	Report Latch
^R^V	Firmware Revision Information
SC	Stop Code
TA	Tell Amplifier Error
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 A axis:

TP A                              Tell position A  
0                                  Controllers Response

## Interrogating Current Commanded Values.

Most commands can be interrogated by using a question mark (?) as the axis specifier.

PR ?                              Request PR setting

The controller can also be interrogated with operands.

## Operands

Most DMC-30000 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 "Operand Usage". Also, see description of operands in Chapter 7 Application Programming.

# Chapter 6 Programming Motion

## Overview

The DMC-30000 provides several modes of motion, each of which are discussed in the following sections.

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, SP, AC, DC
Velocity control where no final endpoint is prescribed. Motion stops on Stop command.	Independent Jogging	JG, AC, DC, ST
Absolute positioning mode where absolute position targets may be sent to the controller while the axis is in motion.	Position Tracking	PA, AC, DC, SP, PT
Motion Path described as incremental position points versus time.	Contour Mode	CM, CD, DT
Motion Path described as incremental position, velocity and delta time	PVT Mode	PV, BT
2 to 8 axis coordinated motion where path is described by linear segments.	Linear Interpolation Mode	LM, LI, LE, VS, VR, VA, VD
2-D motion path consisting of arc segments and linear segments, such as engraving or quilting.	Vector Mode: Linear and Circular Interpolation Motion	VM, VP, CR, VS, VR, VA, VD, VE
Electronic gearing where Follower axes are scaled to Master axis which can move in both directions.	Electronic Gearing	GA, GD, _GP, GR, GM (if gantry)
Master/Follower where Follower axes must follow a Master such as conveyer speed.	Electronic Gearing and Ramped Gearing	GA, GD, _GP, GR
Moving along arbitrary profiles or mathematically prescribed profiles such as sine or cosine trajectories.	Contour Mode	CM, CD, DT
Teaching or Record and Play Back	Contour Mode with Teach (Record and Play-Back)	CM, CD, DT, RA, RD, RC
Backlash Correction	Dual Loop (Auxiliary Encoder)	DV
Following a trajectory based on a Master encoder position	Electronic Cam	EA, EM, EP, ET, EB, EG, EQ
Smooth motion while operating in independent axis positioning	Motion Smoothing	IT
Smooth motion while operating in vector or linear interpolation positioning	Motion Smoothing	IT
Smooth motion while operating with stepper motors	Stepper Motion Smoothing	KS

---

## 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-30000 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-30000 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	Specifies relative distance
PA x	Specifies absolute position
SP x	Specifies slew speed
AC x	Specifies acceleration rate
DC x	Specifies deceleration rate
BG A	Starts motion
ST X	Stops motion before end of move
IP x	Changes position target
IT x	Time constant for independent motion smoothing
AM X	Trippoint for profiler complete
MC A	Trippoint for "in position"

The lower case specifiers (x) represent position values for each axis.

The DMC-30000 also allows use of explicit notation such as PRX=2000

## Operand Summary - Independent Axis

OPERAND	DESCRIPTION
_ACA	Return acceleration rate
_DCA	Return deceleration rate
_SPA	Returns the speed
_PAA	Returns current destination if the axis is moving, otherwise returns the current commanded position if in a move.
_PRA	Returns current incremental distance

### Example - Absolute Position Movement

PA 10000	Specify absolute position
AC 1000000	Acceleration
DC 1000000	Deceleration
SP 50000	Speed
BG X	Begin motion

## 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. 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-30000 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	Specifies acceleration rate
BG X	Begins motion
DC x	Specifies deceleration rate
IP x	Increments position instantly
IT x	Time constant for independent motion smoothing
JG ±x	Specifies jog speed and direction
ST A	Stops motion

Parameters can be set with explicit notation such as JGA=2000.

## Operand Summary - Independent Axis

OPERAND	DESCRIPTION
_ACA	Return acceleration rate
_DCA	Return deceleration rate
_SPA	Returns the jog speed
_TVA	Returns the actual velocity (averaged over 256 samples)

### Example - Jog in X only

Jog motor at 50000 count/s.

```
#A  
AC 20000      Specify acceleration of 20000 counts / sec^2  
DC 20000      Specify deceleration of 20000 counts / sec^2  
JG 50000      Specify jog speed  
BG X          Begin 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 the axis 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 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. 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 countts/sec<sup>2</sup> and the velocity is set to 50,000 counts/sec. The command sequence to perform this is listed below.

```
#EX1
PT 1; ' Place the X axis in Position tracking mode
AC 150000; ' Set the X axis acceleration to 150000 counts/sec^2
DC 150000; ' Set the X axis deceleration to 150000 counts/sec^2
SP 50000; ' Set the X axis speed to 50000 counts/sec
PA 5000; ' Command the X axis to absolute position 5000 encoder counts
EN
```

The output from this code can be seen in Figure 6.1, a screen capture from the GDK scope.

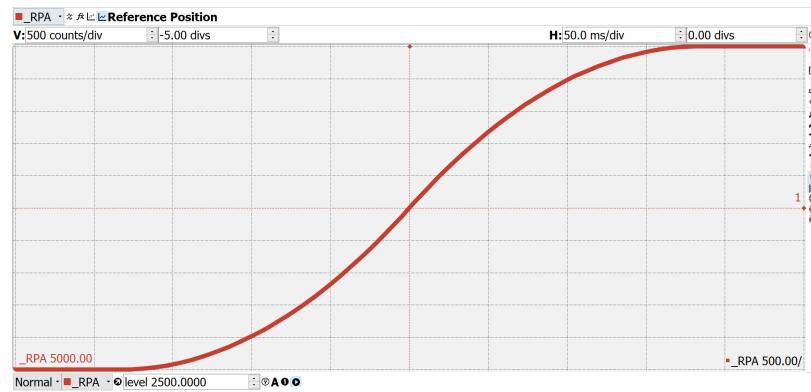


Figure 6.1: Position vs Time (msec) - Motion 1

### 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 counts at that time. Figure 6.1 shows what the position profile would look like if the move was allowed to complete to 5000 counts. The position was modified when the robot was at a position of 4200 counts (Figure 6.2). Note that the robot actually travels to a distance of almost 5000 counts 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 6.2 the velocity profile is triangular because the controller doesn't have sufficient time to reach the set speed of 50000 counts/sec before it is commanded to change direction.

The below code is used to simulate this scenario:

```

#EX2
PT 1;          Place the X axis in Position tracking mode
AC 150000;    Set the X axis acceleration to 150000 counts/sec2
DC 150000;    Set the X axis deceleration to 150000 counts/sec2
SP 50000;     Set the X axis speed to 50000 counts/sec
PA 5000;      Command the X axis to abs position 5000 encoder counts
MF 4200
PA 2000;      Change end point position to position 2000
EN

```

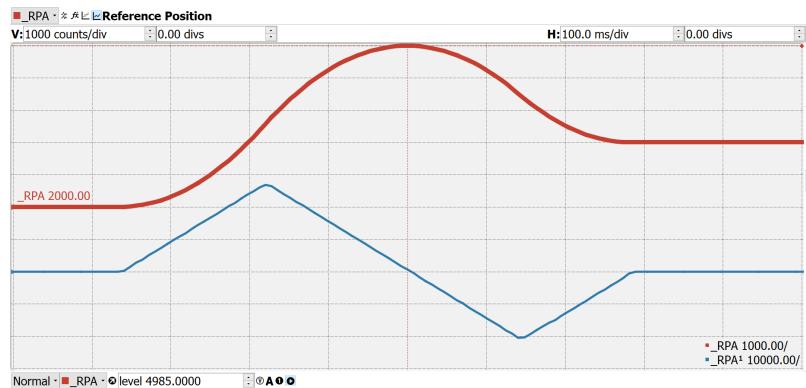


Figure 6.2: Position and Velocity vs Time(msec) for Motion 2

### Example - Motion 3:

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 6.3 shows the plot of position vs. time and velocity vs. time. Below is the code that is used to simulate this scenario:

```

#EX3
PT 1;          Place the X axis in Position tracking mode
AC 150000;    Set the X axis acceleration to 150000 counts/sec2
DC 150000;    Set the X axis deceleration to 150000 counts/sec2
SP 50000;     Set the X axis speed to 50000 counts/sec
PA 5000;      Command the X axis to abs position 5000 encoder counts
WT 300
PA -2000;    Change end point position to -2000
WT 200
PA 8000;      Change end point position to 8000
EN

```

Figure 6.4 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.

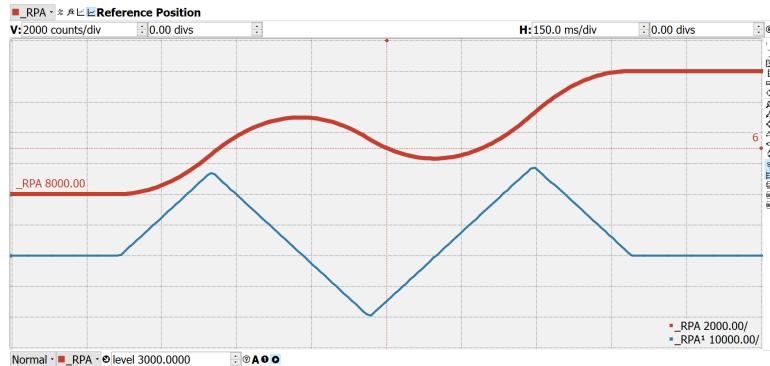


Figure 6.3: Position and Velocity vs Time (msec) for Motion 3

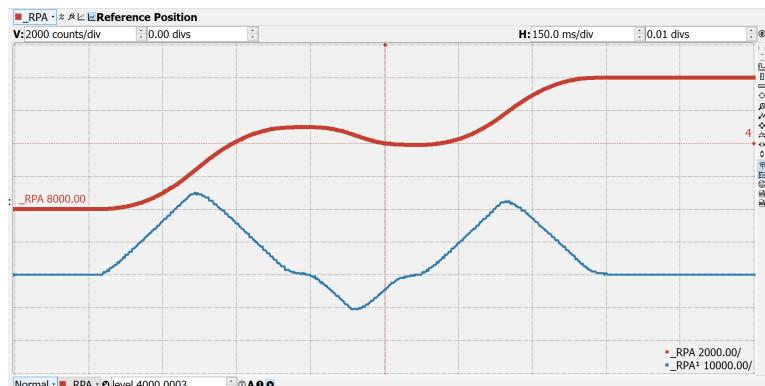


Figure 6.4: Position and Velocity vs Time (msec) for Motion 3 with IT 0.1

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.

## Trippoints

Most trippoints 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 AR, 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	Acceleration settings
AP n	trippoint that holds up program execution until an absolute position has been reached
DC n	Deceleration settings
MF n	trippoint to hold up program execution until n number of counts have passed in the forward direction.
MR n	trippoint to hold up program execution until n number of counts have passed in the reverse direction.
PT n	Command used to enter and exit the Trajectory Modification Mode
PA n	Command Used to specify the absolute position target
SP n	Speed settings

---

## Linear Interpolation Mode

The DMC-30000 provides a linear interpolation mode that allows the buffering of relative moves for a single axis. In linear interpolation mode 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 ("LM A") command selects the Linear Interpolation mode.

### Specifying Linear Segments

The command LI x specifies the incremental move distance. This means motion is prescribed with respect to the current axis position. Up to 31 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-30000 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. 31 returned means the buffer is empty and 31 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.

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.

#### An Example of Linear Interpolation Motion:

#LMOVE	label
DP 0	Define position of 0
LM X	Enable LM mode
LI 5000	Specify first linear segment
LI -10000	Specify second linear segment
LE	End linear segments
VS 4000	Specify vector speed
BG S	Begin motion sequence
EN	Program end

### 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, 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 < n >m

The first command, < n, is equivalent to commanding VS n 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	Define Position of 0
LM XY	Enable LM mode
LI 4000<4000>1000	Specify first linear segment with a vector speed of 4000 and end speed 1000
LI 1000<4000>1000	Specify second linear segment with a vector speed of 4000 and end speed 1000
LI 5000<4000>1000	Specify third linear segment with a vector speed of 4000 and end speed 1000
LE	End linear segments
BG S	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 A	Enable linear interpolation
LM ? or _LMS	Returns number of available spaces for linear segments in DMC-30000 sequence buffer. Zero means buffer full. 31 means buffer empty.
LI x<n>m	Specify incremental distances relative to current position, and assign vector speed n and m.
VS n	Specify vector speed
VA n	Specify vector acceleration
VD n	Specify vector deceleration
VR n	Specify the vector speed ratio
BG S	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)
AM S	Trippoint for After Sequence complete
AV n	Trippoint for After Relative Vector distance, n

## 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-30000 sequence buffer. Zero means buffer full. 31 means buffer empty.
_VPA	Return the absolute coordinate of the last data point along the trajectory.

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 and \_VPA will be zero.

---

## Vector Mode: Linear and Circular Interpolation Motion

The DMC-30000 provides a vector mode that allows the buffering of absolute moves (from the starting position) for a single axis.

The coordinated motion mode is similar to the linear interpolation mode, but the linear segments are specified as absolute positions from the starting position of the A axis.

The command VM AN where A is the A axis, and N is the virtually axis.

### 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. 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. The CR command is useful for producing a sine wave as the move output as a single axis of a circle is a sinusoidal profile.

Up to 31 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-30000 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. 31 returned means the buffer is empty and 31

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.

### 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,e, $\delta$ <n>m

The first command, <n>, is equivalent to commanding VS n 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.

### 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.

## Command Summary - Coordinated Motion Sequence

COMMAND	DESCRIPTION
VM AN	Enable Vector Mode
VP m, n	Specify the Vector segment
CR r, $\theta$ , $\delta\theta$	Specifies arc segment where r is the radius, $\Theta$ is the starting angle and $\Delta\Theta$ is the travel angle. Positive direction is CCW.
VS s	Specify vector speed or feed rate of sequence.
VA s	Specify vector acceleration along the sequence.
VD s	Specify vector deceleration along the sequence.
VR s	Specify vector speed ratio
BG S	Begin motion sequence
CS S	Clear sequence
AV s	Trippoint for After Relative Vector distance.
AMST	Holds execution of next command until Motion Sequence is complete.
LM?	Return number of available spaces for linear and circular segments in DMC-30000 sequence buffer. Zero means buffer is full. 31 means buffer is empty.

## Operand Summary - Coordinated Motion Sequence

OPERAND	DESCRIPTION
_VPA	The absolute coordinate of the axis at the last intersection along the sequence.
_AV	Distance traveled.
_LM	Number of available spaces for linear and circular segments in DMC-30000 sequence buffer. Zero means buffer is full. 31 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 operand \_VPX can be used to return the coordinates of the last point specified along the path.

### Example (Sine Wave Output):

The CR command can be used to command sinusoidal motion to the axis. The below code and scope output shown in Figure 6.5 show an example of how this can be achieved. The frequency and amplitude of the output can be modified by changing the radius in the CR command and by changing the vector speed.

```

REM frequency output (Hz) = (VS*(pi/2)) / (r*10)
REM ex VS 12000 and r=1590
REM frequency(Hz)=(12000*1.57) / (1590*10)=1.18Hz
#SinWv
'vector speed
VS12000
'1/2 amplitude of sine wave
r=1590
VM AN
CR r,-90,90
CR r,0,720
CR r,0,720
CR r,0,720
VE
BGS
'Continue to create sine wave
#LOOP
CR r,0,720;CR r,0,720
#wt;JP#wt,_LM<30
JP#LOOP

```

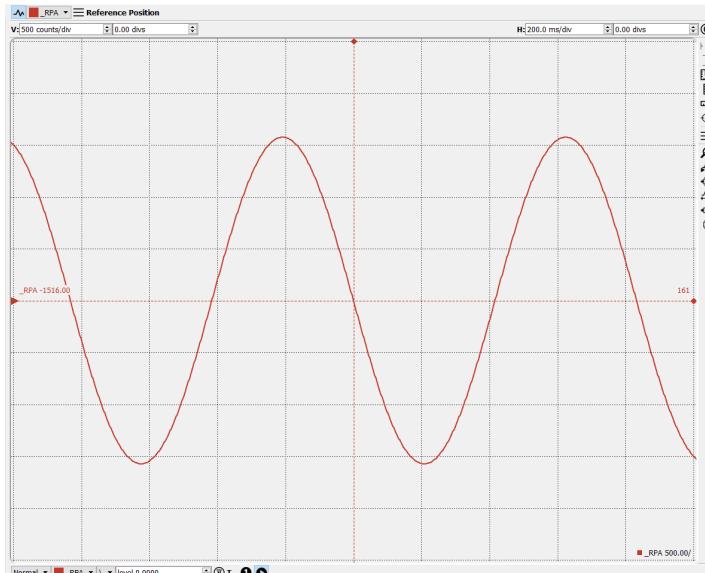


Figure 6.5: Sine Output with Vector Mode

## Electronic Gearing

This mode allows the A axis to be electronically geared to the Auxiliary encoder or the virtual axis. The Master may rotate in both directions and the axis will follow at the specified gear ratio. The gear ratio may be changed during motion.

The GA command specifies the Master axes and the GR command specifies the gear ratio for the Follower where the ratio may be a number between  $\pm 127.9999$  with a fractional resolution of .0001. There are two modes: electronic gearing (the default) 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 turns off gearing in both modes.

Electronic gearing allows the geared motor to perform a second independent 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 Follower when the gearing is engaged. For example, if the Master is already traveling at 500,000 counts/sec and the Follower is to be geared at a ratio of 1:1 when the gearing is engaged, the Follower will instantly develop following error and the controller will command maximum allowable torque to the axis. This can result in excessive shock to the system mechanics. For many applications it is acceptable to ramp the engagement of gearing over a specified distance. Galil allows the user to specify an interval of the Master axis encoder counts over which the gearing will be engaged. For the same Master X axis slewing at 500,000 counts/sec with the gear ratio set to 1:1, with the gearing ramped over 30,000 counts of the Master axis the result will be reduced shock to the Follower axis. Figure 6.6 below shows the velocity vs. time profile for instantaneous gearing. Figure 6.7 shows the velocity vs. time profile for the gradual gearing engagement.



Figure 6.6: Velocity counts/sec vs. Time (msec) Instantaneous Gearing Engagement

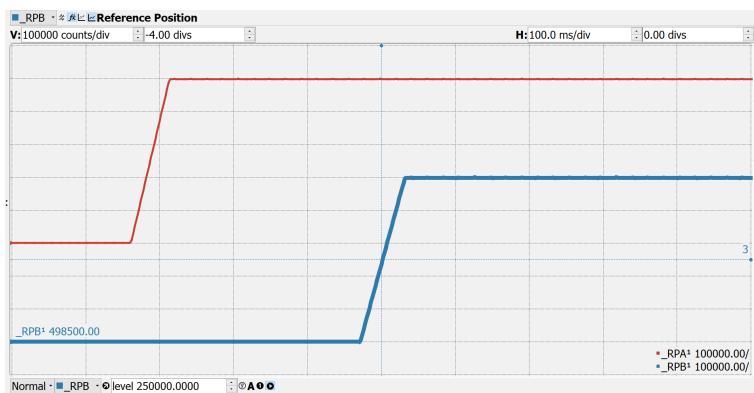


Figure 6.7: Velocity (counts/sec) vs. Time (msec) Ramped Gearing

The Follower 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 Follower axis will be significantly less in Figure 6.7 than in Figure 6.6. The ramped gearing does have one consequence. There isn't a true synchronization of the two axes, until the gearing ramp is complete. The Follower 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.

## Command Summary - Electronic Gearing

COMMAND	DESCRIPTION
GA n	Specifies Master axes for gearing where: n = DA, S or N
GD a	Sets the distance the Master will travel for the gearing change to take full effect.
_GPA	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	Sets gear ratio. 0 disables electronic gearing for specified axis.
GM a	a = 1 sets gantry mode, 0 disables gantry mode
MR x	Trippoint for reverse motion past specified value.
MF x	Trippoint for forward motion past specified value.

## Example - Simple Master Follower

Master axis is the virtual axis and moves 10000 counts, A axis will move 50000 counts.

GA N	Specify Master axes as the N axis
GR 5	Set gear ratio of 5x
PRN=10000	Specify N position
BGN	Begin motion

---

## Electronic Cam

The electronic cam is a motion control mode which enables the periodic synchronization the motor. The Master axis encoder can be the auxiliary encoder input or the virtual axis.

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.

To illustrate the procedure of setting the cam mode, consider the cam relationship shown in Figure 6.8.

### 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

EA<sub>p</sub> where p = DA or N

p is the selected Master axis

For the given example, since the Master is the aux encoder input, we specify EA DA

### Step 2. Specify the Master cycle and the change in the Follower 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 the Master axis is always expressed in the range between 0 and 6000. Similarly, the Follower 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 Follower is 1500, the positions of both x and y are redefined as zero. The MM command specifies the Master modulus, and the EM command specifies the Follower modulus.

The cycle of the Master is limited to 8,388,607 whereas the Follower 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 Follower is 1500. Therefore, we use the instructions:

MM 6000

EM 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 n0,n1

where n0 is the interval width in counts, and n1 is the phase shift.

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 Follower positions.**

Next, we specify the Follower positions with the instruction

`ET[n]=x`

where n indicates the order of the point.

The value, n, starts at zero and may go up to 256. The parameter x will indicate the corresponding Follower 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 Follower motion**

To engage the Follower motion, use the instruction

`EG x`

where x is the Master positions at which the corresponding Follower must be engaged.

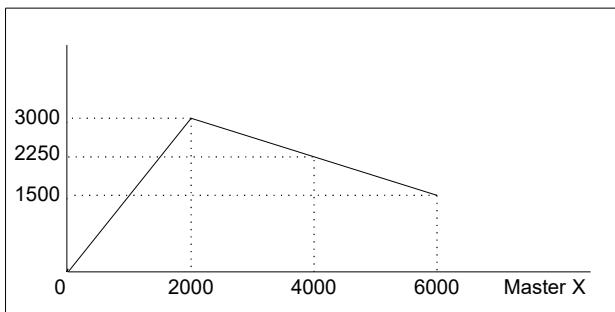
If the value of any parameter is outside the range of one cycle, the cam engages immediately. When the cam is engaged, the Follower position is redefined, modulo one cycle.

#### **Step 7. Disengage the Follower motion**

To disengage the cam, use the command

`EQ x`

where x is the Master positions at which the corresponding Follower axis is disengaged.



*Figure 6.8: Electronic Cam Example*

This disengages the Follower axis at a specified Master position. If the parameter is outside the Master cycle, the stopping is instantaneous.

## ECAM - Example

To illustrate the complete process, consider the cam relationship described by the equation:

$$X = 0.5 * N + 100 \sin (0.18*N)$$

where N (virtual axis) 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 EA N defines virtual axis as the Master axis. The cycle of the Master is 2000. Over that cycle, the Follower varies by 1000. This leads to the instructions MMN= 2000 and EMA= 1000.

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  $3.6^\circ$ . The program then computes the values of Y according to the equation and assigns the values to the table with the instruction ET[i] = x.

INSTRUCTION	INTERPRETATION
#SETUP	Label
EAN	Select X as Master
EMA= 1000	Follower Modulus
MMN=2000	
EP 20,0	Master position increments
i = 0	Index
#LOOP	Loop to construct table from equation
p = i * 3.6	Note $3.6 = 0.18 * 20$
s = @SIN[p]*100	Define sine position
x = i*10+s	Define Follower position
ET [i] = x	Define table
i = i+1	
JP #LOOP, i<=100	Repeat the process
EN	

---

## PVT Mode

The DMC-30000 controllers now supports a mode of motion referred to as "PVT." This mode allows arbitrary motion profiles to be defined by position, velocity and time. This motion is designed for systems where the load must traverse a series of coordinates with no discontinuities in velocity. By specifying the target position, velocity and time to achieve those parameters the user has control over the velocity profile. Taking advantage of the built in buffering the user can create virtually any profile including those with infinite path lengths.

### Specifying PVT Segments

PVT segments are commanded using the PV command. The PV command includes the target distance to be moved and target velocity to be obtained over the specified timeframe. Positions are entered as relative moves, similar to the standard PR command, in units of encoder counts and velocity is entered in counts/second. The controller will interpolate the motion profile between subsequent PV commands using a 3rd order polynomial equation. During a PV segment, jerk is held constant, and accelerations, velocities, and positions will be calculated every other sample.

Motion will not begin until a BT command is issued, much like the standard BG command. This means that the user can fill the PVT buffer prior to motion beginning. PVT mode has a 127 segment buffer. This buffer is a FIFO and the available space can be queried with the operand \_PVA. As the buffer empties the user can add more PVT segments by issuing new PV commands.

## Exiting PVT Mode

To exit PVT mode the user must send the segment command PVA=0,0,0. This will exit the mode once the segment is reached in the buffer. To avoid an abrupt stop the user should slow the motion to a zero velocity prior to executing this command. The controller will instantly command a zero velocity once a PVA=0,0,0 is executed. In addition, a ST command will also exit PVT mode. Motion will come to a controlled stop using the DC value for deceleration. The same controlled stop will occur if a limit switch is activated in the direction of motion. As a result, the controller will be switched to a jog mode of motion.

## Error Conditions and Stop Codes

If the buffer is allowed to empty while in PVT mode then the profiling will be aborted and the motor will come to a controlled stop on that axis with a deceleration specified by the DC command. Also, PVT mode will be exited and the stop code will be set to 32. During normal operation of PVT mode the stop code will be 30. If PVT mode is exited normally (PVA=0,0,0), then the stop code will be set to 31.

## Additional PVT Information

It is the users' responsibility to enter PVT data that the system's mechanics and power system can respond to in a reasonable manner. Because this mode of motion is not constrained by the AC, DC or SP values, if a large velocity or position is entered with a short period to achieve it, the acceleration can be very high, beyond the capabilities of the system, resulting in excessive position error. The position and velocity at the end of the segment are guaranteed to be accurate but it is important to remember that the required path to obtain the position and velocity in the specified time may be different based on the PVT values. Mismatched values for PVT can result in different interpolated profiles than expected but the final velocity and position will be accurate.

The "t" value is entered in samples, which will depend on the TM setting. With the default TM of 1000, one sample is 976us. This means that a "t" value of 1024 will yield one second of motion. The velocity value, "v" will always be in units of counts per second, regardless of the TM setting.

## Command Summary – PVT

COMMAND	DESCRIPTION
PVA = p,v,t	Specifies the segment for an incremental PVT segment of 'p' counts, an end speed of 'v' counts/sec in a total time of 't' samples.
_PVA	Contains the number of PV segments available in the PV buffer.
BT	Begin PVT mode
_BTA	Contains the number PV segments that have executed

## PVT Examples

### Parabolic Velocity Profile

In this example we will assume that the user wants to start from zero velocity, accelerate to a maximum velocity of 1000 counts/second in 1 second and then back down to 0 counts/second within an additional second. The velocity profile would be described by the following equation and shown in Figure 6.9.

$$v(t) = -1000(t-1)^2 + 1000$$

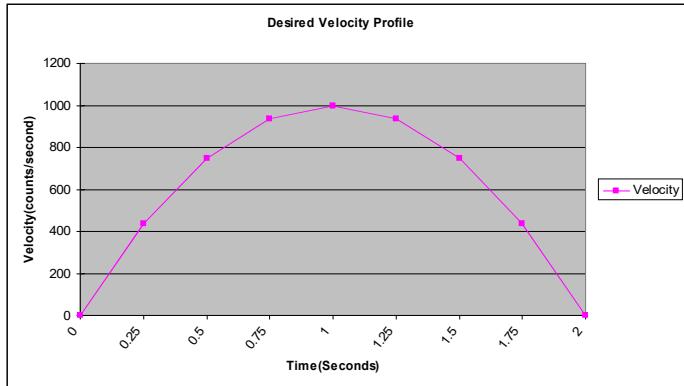


Figure 6.9: Parabolic Velocity Profile

To accomplish this we need to calculate the desired velocities and change in positions. In this example we will assume a delta time of  $\frac{1}{4}$  of a second, which is 256 samples (1024 samples = 1 second with the default TM of 1000).

Velocity(counts/second)	Position(counts)
437.5	$p(0 \text{ to } .25) = 57$
750	$p(.25 \text{ to } .5) = 151$
937.5	$p(.5 \text{ to } .75) = 214$
1000	$p(.75 \text{ to } 1) = 245$
937.5	$p(1 \text{ to } 1.25) = 245$
750	$p(1.25 \text{ to } 1.5) = 214$
437.5	$p(1.5 \text{ to } 1.75) = 151$
0	$p(1.75 \text{ to } 2) = 57$

The DMC program is shown below and the results can be seen in Figure 6.10.

INSTRUCTION	INTERPRETATION
#PVT	Label
PVX = 57,437,256	Incremental move of 57 counts in 256 samples with a final velocity of 437 counts/sec
PVX = 151,750,256	Incremental move of 151 counts in 256 samples with a final velocity of 750 counts/sec
PVX = 214,937,256	Incremental move of 214 counts in 256 samples with a final velocity of 937 counts/sec
PVX = 245,1000,256	Incremental move of 245 counts in 256 samples with a final velocity of 1000 counts/sec
PVX = 245,937,256	Incremental move of 245 counts in 256 samples with a final velocity of 937 counts/sec
PVX = 214,750,256	Incremental move of 214 counts in 256 samples with a final velocity of 750 counts/sec
PVX = 151,437,256	Incremental move of 151 counts in 256 samples with a final velocity of 437 counts/sec
PVX = 57,0,256	Incremental move of 57 counts in 256 samples with a final velocity of 0 counts/sec
PVX = 0,0,0	Termination of PVT buffer
BTX	Begin PVT
EN	

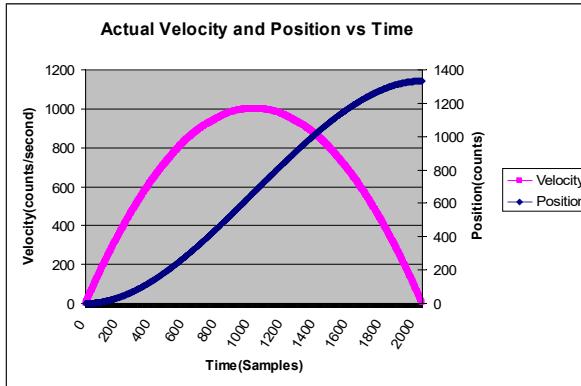


Figure 6.10: Actual Velocity and Position vs Time of Parabolic Velocity Profile

## Contour Mode

The DMC-30000 also provides a contouring mode. This mode allows any arbitrary position curve to be prescribed. 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.

A contour is described by position increments which are described with the command, CD x over a time interval, DT n. The parameter, n, specifies the time interval. The time interval is defined as  $2^n$  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=n

where n is the new DT value.

Consider, for example, the trajectory shown in Figure 6.11. 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
```

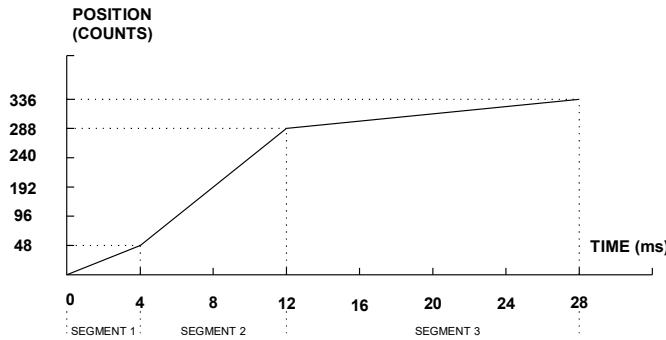


Figure 6.11: The Required Trajectory

## 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 will 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 ?.

Specifying a -1 for the DT or as the time interval in the CD command will pause the contour buffer.

Issuing the CM command will clear the contour buffer.

## Command Summary - Contour Mode

COMMAND	DESCRIPTION
CM A	Specifies contour mode.
CD x	Specifies position increment over time interval. Range is $\pm 32,000$ . CD 0=0 ends the contour buffer. This is much like the LE or VE commands.
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 Figure 6.12. 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:

**NOTE:**  $\omega$  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,  $\omega$ , in count/ms, is

$$\omega = 50 [1 - \cos 2\pi T/120]$$

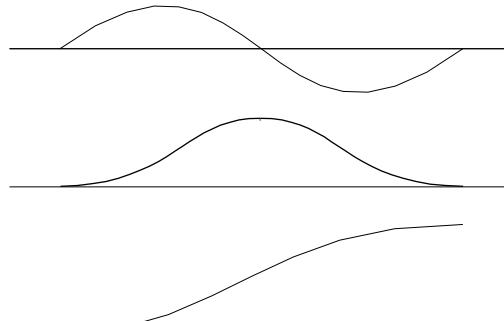


Figure 6.12: Velocity Profile with Sinusoidal Acceleration

The DMC-30000 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-30000 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-30000)
RD _TPX	Specify data for capturing (such as _TPX or _TPZ)
RC n,m	Specify capture time interval where n is 2 <sup>n</sup> 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 contour buffer
#Wait;JP#Wait,_CM>>511	Wait until path is done
EN	End program

For additional information about automatic array capture, see [Chapter 7 Application Programming](#).

---

## Virtual Axis

The DMC-30000 controller has an additional virtual axis designated as the N axis. This axis has no encoder and no DAC. However, it 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 axis is to serve as a virtual Master in ECAM mode, 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
VM XN	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

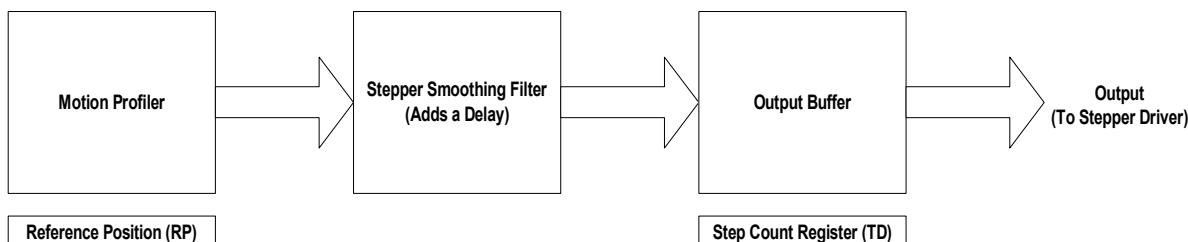
For proper controller operation, it is necessary to make sure that the controller has completed generating all step pulses before making additional moves. This is most particularly important if you are moving back and forth. For example, when operating with servo motors, the trippoint AM (After Motion) 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.

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 possible with a 2PB supported amplifier. See the A4 – DMC-30017 for more information.

## 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
_DEA	Contains the value of the step count register
_DPA	Contains the value of the main encoder
_ITA	Contains the value of the Independent Time constant
_KSA	Contains the value of the Stepper Motor Smoothing constant
_MTA	Contains the motor type value
_RPA	Contains the commanded position generated by the profiler
_TDA	Contains the value of the step count register
_TPA	Contains the value of the main encoder

---

## 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 OEM=1 for an axis, it will be stopped. Under SPM mode, the motor will not be disabled by position error.
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
```

**1/64<sup>th</sup> 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 in order to detect the 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
OE 1;          Set the profiler to stop axis upon error
KS 16;         Set step smoothing
MT -2,-2,-2,-2;  Motor type set to stepper
YA 2;          Step resolution of the drive
YB 200;        Motor resolution (full steps per revolution)
YC 4000;       Encoder resolution (counts per revolution)
SH A;          Enable axis
WT 100;        Allow slight settle time

#motion;
SP 512;        Perform motion
PR 1000;       Set the speed
BG A;          Prepare mode of motion
EN;            Begin motion
EN;            End of program subroutine
```

REM When error occurs, the axis will stop due to OE1. In  
REM #POSEERR, query the status YS and the error QS, correct,  
REM and return to the main code.

```

#POSERR; '          Automatic subroutine is called when _YS=2
WT 100; '          Wait helps user see the correction
spsave=_SPA; '      Save current speed setting
JP #return,_YSA<>2; '  Return to thread zero if invalid error
SP64; '             Set slow speed setting for correction
MG "ERROR= ",_QSA
YRA=_QSA; '
MC A; '             Wait for motion to complete
MG "CORRECTED, ERROR NOW= ",_QSX
WT 100; '           Wait helps user see the correction

#return
SPA=spsave; '       Return the speed to previous setting
RE 0; '            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 1/256th 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; '            Motor type set to stepper
YA256; '           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;
SP2048; '           Save speed value
WT100; '            Set a new slow correction speed
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-30000 provides an interface for a second encoder except when the controller is configured for stepper motor operation or 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 where the parameter x is 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, can be used to define the position of the auxiliary encoders. For example,

DE 500

sets the value to 500. The positions of the auxiliary encoders may be interrogated with the command, DE? or the operand \_DEA.

The command TD X returns the current position of the auxiliary encoder.

The command DV 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

activates dual loop and

DV 0

disables dual loop.

**NOTE:** 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 KPA=0, KIA=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-30000 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 command:

IT x                          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 parameter, x, is a number 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. Figure 6.13 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

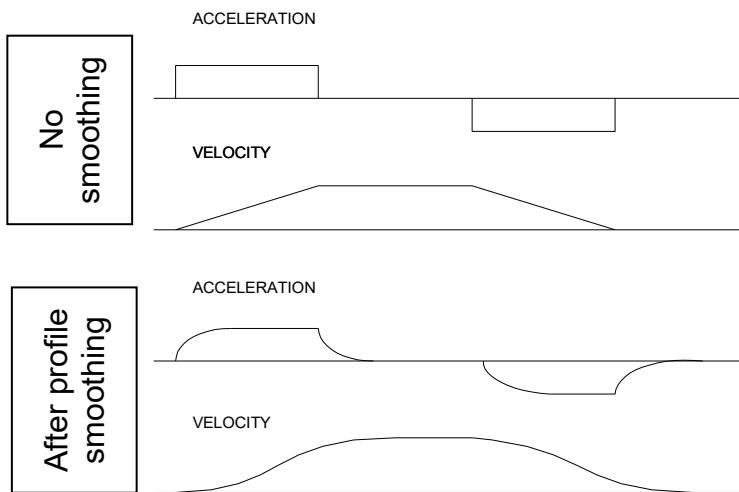


Figure 6.13: Trapezoidal velocity and smooth velocity profiles

## 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                          where x is an integer from 0.5 to 128 and represents the amount of smoothing

The smoothing parameters, x,y,z,w and n are numbers between 0.5 and 128 and determine the degree of filtering. The minimum value of 0.5 implies the least 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-30000 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 6.14 shows the velocity profile from the homing sequence of the example program above. For this profile, the switch is normally closed and CN,-1.

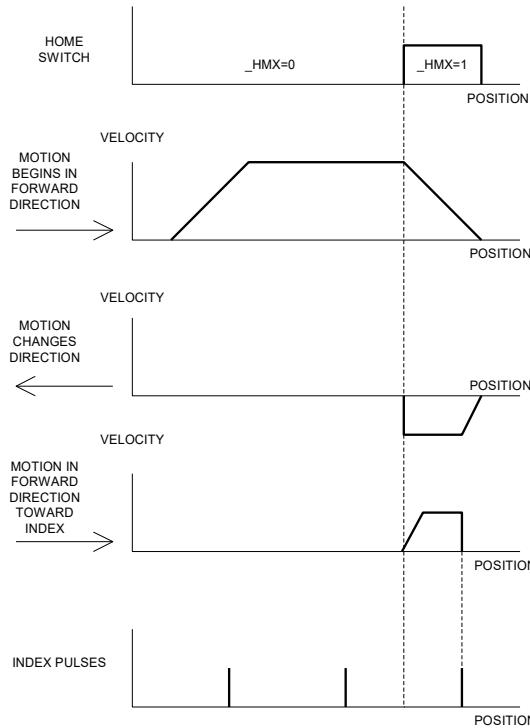


Figure 6.14: Homing Sequence for Normally Closed Switch and CN,-1

### 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 A	Find Edge Routine. This routine monitors the Home Input
FI A	Find Index Routine - This routine monitors the Index Input
HM A	Home Routine - This routine combines FE and FI as Described Above
SC A	Stop Code
TS A	Tell Status of Switches and Inputs

### Operand Summary - Homing Operation

Operand	Description
_HMA	Contains the value of the state of the Home Input
_SCA	Contains stop code
_TSA	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. Position capture can be programmed to latch on either a corresponding input (see Table 6.1) or on the index pulse for that axis. The position can be captured for either the main or auxiliary encoder within 25 microseconds of an high-to-low transition.

Input 1	A-axis latch
---------	--------------

Table 6.1: Inputs and corresponding axis latch

**NOTE**

Latching is not valid with sampled feedback types such as: SSI, BiSS, Sin/Cos, and Analog

To insure a position capture within 25 microseconds, the input signal must be a transition from high to low. Low to high transitions may have greater delay.

The software commands, AL and RL, are used to arm the latch and report the latched position respectively. The latch must be re-armed after each latching event. See the Command Reference for more details on these commands.

---

## Real Time Clock

The DMC-30000 is equipped with a real time clock feature. The real time clock provides true time in seconds, minutes and hours. The RT command provides a method to set the time and operands to return the current time. The default real time clock does not persist through a power cycle and must be set whenever power is restored.

The DMC-30000 can be ordered with a clock upgrade (RTC) including a higher precision clock than the default, and a battery backup for the time hardware. All hardware is within the standard sheet metal footprint. The RTC clock will continue to run when power is removed from the controller. The RTC option also provides a calendar function including year, month of year, day of month, and day of week. This feature can be set and queried through the RY command.

Both versions of the real time clock can be set to a TIME protocol (RFC 868) server. Using IH, the DMC-30000 can connect to a TIME server over TCP on port 37 and receive the 32-bit response. The firmware will then set the time and calendar (if applicable) to the TIME server value. The command RO is used to set the GMT time zone offset for localization of the current time. The TIME protocol synchronization is designed to connect to a server on the local network. Contact Galil if a local server is not available (e.g. an Internet Gateway is required to contact NIST).

See the RTC – Real Time Clock, pg 156 section in the Appendix for further details and specifications for the RTC option.

# Chapter 7 Application Programming

---

## Overview

The DMC-30000 provides a powerful programming language that allows users to customize the controller for their particular application. Programs can be downloaded into the DMC-30000 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-30000 provides commands that allow the DMC-30000 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-30000 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 1000 lines.

<b>WARNING</b>	During a program download the servo will disable for a brief period, roughly 300ms. This can be prevented by first issuing an MO. Motors with a brake (BW) must be in motor off (MO) before program download to engage the brake. This is not applicable to stepper motors.
----------------	---

---

## Program Format

A DMC-30000 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-30000 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-30000 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 in a label.

The maximum number of labels which may be defined is 126.

#### Valid labels

```
#BEGIN  
#SQUARE  
#X1  
#BEGIN1
```

#### Invalid labels

```
#1Square  
#123
```

A Simple Example Program:

#START	Beginning of the Program
PR 10000	Specify relative distance
BG A	Begin Motion
AM A	Wait for motion complete
WT 2000	Wait 2 sec
JP #START	Jump to label START
EN	End of Program

The above program moves 10000 counts. 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-30000 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 \_\_\_\_\_

#AMPERR	Label for Amplifier error routine
#AUTO	Label that will automatically run upon the controller exiting a reset (power-on)
#AUTOERR	Label that will automatically run if there is an FLASH error out of reset
#CMDERR	Label for incorrect command subroutine
#COMINT	Label for Communications Interrupt (See CC Command)
#ININT	Label for Input Interrupt subroutine (See II Command)
#LIMSWI	Label for Limit Switch subroutine
#MCTIME	Label for timeout on Motion Complete trippoint
#POSERR	Label for excess Position Error subroutine
#TCPERR	Label for errors over a TCP connection (error code 123)

## Commenting Programs

### Using the command, NO or Apostrophe ('')

The DMC-30000 provides a command, NO, for commenting programs or single apostrophe. This command allows the user to include up to 38 characters on a single line after the NO command and can be used to include comments from the programmer.

**NOTE:** The NO and ('') commands are actual controller commands. Therefore, inclusion of the NO or ('') commands will require process time by the controller, see General Program Flow and Timing information for more details.

### Difference between NO and ' using the Galil Design Kit (GDK) software

The Galil Design Kit (GDK) software will treat an apostrophe ('') command different from an NO when the compression algorithm is activated upon a program download (line > 80 characters or program memory > 1000 lines). In this case the software will remove all ('') comments as part of the compression and it will download all NO comments to the controller.

---

## Executing Programs - Multitasking

The DMC-30000 can run up to 6 independent programs simultaneously. These programs are called threads and are numbered 0 through 5, 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. 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-30000 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.

### Error Code Command

When there is a program error, the DMC-30000 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, TCO 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.

### Flash Memory Interrogation Commands

For debugging the status of the program memory, array memory, or variable memory, the DMC-30000 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-30000 will have a maximum of 3000 array elements in up to 6 arrays. If an array of 100 elements is defined, the command DM ? will return the value 2900 and the command DA ? will return 5.

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:

```

Download code
#A
PR1000
BGX
PR5000
EN
From Terminal
:XQ #A
?003 PR5000
:TC1
?7 Command not valid while running.
:XQ #A

Program Label
Position Relative 1000
Begin
Position Relative 5000
End
Execute #A
Error on Line 3
Tell Error Code
Command not valid while running.
Change the BGX line to BGX;AMX and re-download the program.
Execute #A

```

---

## Program Flow Commands

The DMC-30000 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-30000 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-30000 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-30000 Event Triggers

Command	Function
AM A or S	Halts program execution until motion is complete on the specified axes or motion sequence(s). This command is useful for separating motion sequences in a program.
AD A	Halts program execution until position command has reached the specified relative distance from the start of the move.
AR A	Halts program execution until after specified distance from the last AR or AD command has elapsed.
AP A	Halts program execution until after absolute position occurs.
MF A	Halt program execution until after forward motion reached absolute position. If position is already past the point, then MF will trip immediately. Will function on geared axis or aux. inputs.
MR A	Halt program execution until after reverse motion reached absolute position. If position is already past the point, then MR will trip immediately. Will function on geared axis or aux. inputs.
MC A	Halt program execution until after the motion profile has been completed and the encoder has entered or passed the specified position. TW x 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
AS A	Halts program execution until the axis has reached its slew speed.
AT ±n,m	For m=omitted or 0, 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. For m=1. Same functionality except that n is number of samples rather than msec
AV n	Halts program execution until specified distance along a coordinated path has occurred.
WT n,m	For m=omitted or 0, halts program execution until specified time in msec has elapsed. For m=1. Same functionality except that n is number of samples rather than msec.

## 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
VM XN;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
ATO; '              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; '               End Program
```

## Using AT/WT with non-default TM rates

By default both WT and AT are defined to hold up program execution for 'n' number of milliseconds (WT n or AT n). The second field of both AT and WT can be used to have the program execution be held-up for 'n' number of samples rather than milliseconds. For example WT 400 or WT 400,0 will hold up program execution for 400 msec regardless of what is set for TM. By contrast WT 400,1 will hold up program execution for 400 samples. For the default TM of 1000 the servo update rate is 976us per sample, so the difference between WT n,0 and WT n,1 is minimal. The difference comes when the servo update rate is changed. With a low servo update rate, it is often useful to be able to time loops based upon samples rather than msec, and this is where the "unscaled" WT and AT are useful. For example:

```
#MAIN; '           Label
TM 250; '           250us update rate
#MOVE; '             Label
PRX=1000; '          Position Relative Move
BGX; '              Begin Motion
MCX; '              Wait for motion to complete
WT 2,1; '            Wait 2 samples (500us)
SB1; '              Set bit 1
EN; '               End Program
```

In the above example, without using an unscaled WT, the output would either need to be set directly after the motion was complete, or 2 ms after the motion was complete. By using WT n,1 and a lower TM, greater delay resolution was achieved.

## Conditional Jumps

The DMC-30000 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-30000 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-30000 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-30000 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-30000 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-30000 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 its 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-30000 allows for IF conditional statements to be included within other IF conditional statements. This technique is known as ‘nesting’ and the DMC-30000 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
II,,3
MG "WAITING FOR INPUT 1, INPUT 2"
#LOOP
JP #LOOP
EN
#ININT
IF (@IN[1]=0)
IF (@IN[2]=0)
MG "INPUT 1 AND INPUT 2 ARE ACTIVE"
ELSE
MG "ONLY INPUT 1 IS ACTIVE"
ENDIF
ELSE
MG "ONLY INPUT 2 IS ACTIVE"
ENDIF
#WAIT
JP#WAIT, (@IN[1]=0) | (@IN[2]=0)
RIO
```

Begin Main Program “TEST”  
Enable input interrupts on input 1 and input 2  
Output message  
Label to be used for endless loop  
Endless loop  
End of main program  
Input Interrupt Subroutine  
IF conditional statement based on input 1  
2<sup>nd</sup> IF conditional statement executed if 1<sup>st</sup> IF conditional true  
Message to be executed if 2<sup>nd</sup> IF conditional is true  
ELSE command for 2<sup>nd</sup> IF conditional statement  
Message to be executed if 2<sup>nd</sup> IF conditional is false  
End of 2<sup>nd</sup> conditional statement  
ELSE command for 1<sup>st</sup> IF conditional statement  
Message to be executed if 1<sup>st</sup> IF conditional statement is false  
End of 1<sup>st</sup> conditional statement  
Label to be used for a loop  
Loop until both input 1 and input 2 are not active  
End Input Interrupt Routine without restoring tripoints

## 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 16 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-30000 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-30000 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. bit 0 for variable checksum error bit 1 for parameter checksum error bit 2 for program checksum error bit 3 for Master Reset error (there should be no program )
#AMPERR	Error from internal Galil amplifier

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-30000 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.

#LOOP	Dummy Program
JP #LOOP;EN	Jump to Loop
#LIMSWI	Limit Switch Label
MG "LIMIT OCCURRED"	Print Message
RE	Return to main program
	Download Program
: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

#LOOP	Dummy Program
JP #LOOP;EN	Loop
#POSERR	Position Error Routine
V1= _TEX	Read Position Error
MG "EXCESS POSITION ERROR"	Print Message
MG "ERROR=", V1=	Print Error
RE	Return from Error
	Download program
: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	Jog
BGX	Begin Motion
#LOOP;JP#LOOP;EN	Loop
#ININT	Input Interrupt
STX;AM	Stop Motion
#TEST;JP #TEST, @IN[1]=0	Test for Input 1 still low
JG 30000	Restore Jog
BGX	Begin motion
RIO	Return from interrupt routine to Main Program and do not re-enable tripoints

## 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 trippoint
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
speed = 2000	Set variable for speed
JG speed;BGX;	Begin motion
#LOOP	
JG speed;WT100	Update Jog speed based upon speed variable
JP #LOOP	
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-30000 is used to move the axis back and forth from 0 to 10000. This motion can be paused, resumed and stopped via input from an RS-232 device.

```
#BEGIN                             Label for beginning of program
CI 2                               Setup communication interrupt for serial port
MG {P1}"Type 0 to stop motion"    Message out of serial port
MG {P1}"Type 1 to pause motion"   Message out of serial port
MG {P1}"Type 2 to resume motion"  Message out of serial 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,P1CH="0"                  Check for S (stop motion)
JP #PAUSE,P1CH="1"                 Check for P (pause motion)
JP #RESUME,P1CH="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 tripoint and communication interrupt
#RESUME                           Routine for resuming motion
SPA=rate                           Set speed on A axis to original speed
EN1,1                              Re-enable tripoint and communication interrupt
```

For additional information, see section on Using Communication Interrupt.

## Example – Ethernet Communication Error

This simple program executes in the DMC-30000 and indicates (via the serial port) when a communication handle fails. By monitoring the serial port, the user can re-establish communication if needed.

```
#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
```

## Example – Amplifier Error

The program below will execute upon the detection of an error from an internal Galil Amplifier. The bits in TA1 will be set for all axes that have an invalid hall state even if BR1 is set for those axes, this is handled with the mask variable shown in the code below.

```
#AMPERR
REM mask out if in brushed mode for _TA1
mask=@COM[_BRA]
mask=(_TA1&mask)&$0000FFFF
REM amplifier error status
MG"A-ER TA0",_TA0
MG"A-ER TA1",mask
MG"A-ER TA2",_TA2
MG"A-ER TA3",_TA3
WT5000
REM the sum of the amperr bits should be 0 with no amplifier error
er=_TA0+mask+_TA2+_TA3
JP#AMPERR,er0
REM Notify user amperr has cleared
MG"AMPERR RESOLVED"
WT3000
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.

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.
4. Please refer to the JS command in the controller's Command Reference for further important information.

### 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
EN,, (^a+^b+^c+^d+^e+^f+^g+^h) ;NO(^a,^b,^c,^d,^e,^f,^g,^h) syntax note for use
;                                ;' 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 buckets 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. finteger 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

```

## General Program Flow and Timing information

This section will discuss general programming flow and timing information for Galil programming.

### REM vs. NO or ' comments

There are 2 ways to add comments to a .dmc program. REM statements or NO/ ' comments. The main difference between the 2 is that REM statements are stripped from the program upon download to the controller and NO or ' comments are left in the program. In most instances the reason for using REM statements instead of NO or ' is to save program memory. The other benefit to using REM commands comes when command execution of a loop, thread or any section of code is critical. Although they do not take much time, NO and ' comments still take time to process. So when command execution time is critical, REM statements should be used. The 2 examples below demonstrate the difference in command execution of a loop containing comments.

**NOTE:** Actual processing time will vary depending upon number of axes, communication activity, number of threads currently executing etc.

```

#a
i=0;'initialize a counter
t= TIME;' set an initial time reference
#loop
NO this comment takes time to process
'this comment takes time to process
i=i+1;'this comment takes time `
to process
JP#loop,i<1000
MG TIME-t;'display number of samples `
from initial time reference
EN

```

When executed on a DMC-30012, the output from the above program returned a 158, which indicates that it took 158 samples (TM 1000) to process the commands from 't=TIME' to 'MG TIME-t'. This is about 154ms ±2ms.

Now when the comments inside of the #loop routine are changed into REM statements (a REM statement must always start on a new line), the processing is greatly reduced.

When executed on the same DMC-30012, the output from the program shown below returned a 84, which indicates that it took 84 samples to process the commands from 't=TIME' to 'MG TIME-t'. This is about 82ms ±2ms, and about 50% faster than when the comments where downloaded to the controller.

```

#a
i=0;'initialize a counter
t= TIME;' set an initial time reference
#loop
REM this comment is removed upon download and takes no time to process
REM this comment is removed upon download and takes no time to process
i=i+1
REM this comment is removed upon download and takes no time to process
JP#loop,i<1000
MG TIME-t;'display number of samples`  

    from initial time reference
EN

```

### WT vs AT and coding deterministic loops

The main difference between WT and AT is that WT will hold up execution of the next command for the specified time from the execution of the WT command, AT will hold up execution of the next command for the specified time from the last time reference set with the AT command.

```

#A
AT0;'set initial AT time reference
WT 1000,1;'wait 1000 samples
t1=TIME
AT 4000,1;'wait 4000 samples from last time reference
t2=TIME-t1
REM in the above scenario, t2 will be ~3000 because AT 4000,1 will have
REM paused program execution from the time reference of AT0
REM since the WT 1000,1 took 1000 samples, there was only 3000 samples left
REM of the "4000" samples for AT 4000,1
MG t,t2;'this should output 1000,3000
EN;'End program

```

Where the functionality of the operation of the AT command is very useful is when it is required to have a deterministic loop operating on the controller. These instances range from writing PLC-type scan threads to writing custom control algorithms. The key to having a deterministic loop time is to have a trippoint that will wait a specified time independent of the time it took to execute the loop code. In this definition, the AT command is a perfect fit. The below code is an example of a PLC-type scan thread that runs at a 500ms loop rate. A typical implementation would be to run this code in a separate thread (ex XQ#plcscan,2).

```

REM this code will set output 3 high if
REM inputs 1 and 2 are high, and input 3 is low
REM else output 3 will be low
REM if input 4 is low, output 1 will be high
REM and ouput 3 will be low regardless of the
REM states of inputs 1,2 or 3
#plcscan
AT0;'set initial time reference
#scan
REM mask inputs 1-4
ti=_TIO&$F
REM variables for bit 1 and bit 3
b1=0;b3=0
REM if input 4 is high set bit 1 and clear bit 3
REM ti&8 - gets 4th bit, if 4th bit is high result = 8
IF ti&8=8;b1=1;ELSE
REM ti&7 get lower 3 bits, if 011 then result = 3
IF ti&7=3;b3=1;ENDIF;ENDIF
REM set output bits 1 and 3 accordingly
REM set outputs at the end for a PLC scan
OB1,b1;OB3,b3
REM wait 500ms (for 500 samples use AT-500,1)
REM the '-' will reset the time reference

```

## Mathematical and Functional Expressions

### Mathematical Operators

For manipulation of data, the DMC-30000 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

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
```

### Mathematical Operation Precision and Range

The controller stores non-integers in a fixed point representation (not floating point). Numbers are stored as 4 bytes of integer and 2 bytes of fraction within the range of  $\pm 2,147,483,647.9999$ . The smallest number represented (and thus the precision) is 1/65536 or approximately 0.000015.

#### Example:

Using basic mathematics it is known that  $1.4 * (80,000) = 112,000$ . However, using a basic terminal, a DMC controller would calculate the following:

```
:var= 1.4*80000;          Storing the result of 1.4*80000 in var
:MG var;                  Prints variable "var" to screen
111999.5117
:
```

The reason for this error relies in the precision of the controller. 1.4 must be stored to the nearest multiple of 1/65536, which is  $91750/65536 = 1.3999$ . Thus,  $(91750/65536)*80000 = 111999.5117$  and reveals the source of the error.

By ignoring decimals and multiplying by integers first (since they carry no error), and then adding the decimal back in by dividing by a factor of 10 will allow the user to avoid any errors caused by the limitations of precision of the controller. Continuing from the example above:

```
:var= 14*80000;          Ignore decimals
:MG var;                  Print result
1120000.0000
:var= var/10;            Divide by 10 to add in decimal
:MG var;                  Print correct result
112000.0000
:
```

## 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-30000 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
len="123456"
Flen=@FRAC[len]
Flen=$10000*Flen
len1=(Flen&$00FF)
len2=(Flen&$FF00) /$100
len3=len&$000000FF
len4=(len&$0000FF00) /$100
len5=(len&$00FF0000) /$10000
len6=(len&$FF000000) /$1000000
MG len6 {S4}
MG len5 {S4}
MG len4 {S4}
MG len3 {S4}
MG len2 {S4}
MG len1 {S4}
EN
```

Begin main program  
Set len to a string value  
Define variable 'Flen' as fractional part of variable 'len'  
Shift Flen by 32 bits (IE - convert fraction, Flen, to integer)  
Mask top byte of Flen and set this value to variable 'len1'  
Let variable, 'len2' = top byte of Flen  
Let variable, 'len3' = bottom byte of len  
Let variable, 'len4' = second byte of len  
Let variable, 'len5' = third byte of len  
Let variable, 'len6' = fourth byte of len  
Display 'len6' as string message of up to 4 chars  
Display 'len5' as string message of up to 4 chars  
Display 'len4' as string message of up to 4 chars  
Display 'len3' as string message of up to 4 chars  
Display 'len2' as string message of up to 4 chars  
Display 'len1' as string message of up to 4 chars

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-30000 provides 254 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-30000 allows the user to create up to 254 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-30000 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 ( $\pm 2,147,483,647.9999$ ).

Numeric values can be assigned to programmable variables using the equal sign.

Any valid DMC-30000 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 SP 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.

#JOYSTIK	Label
JG 0	Set in Jog mode
BGX	Begin Motion
ATO	Set AT time reference
#LOOP	Loop
vX=@AN[1]*20000	Read joystick X
JG vX	Jog at variable vX
AT-4	Wait 4ms from last time reference, creates a deterministic loop time
JP#LOOP	Repeat
EN	End

---

## Operands

Operands allow motion or status parameters of the DMC-30000 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-30000 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-30000 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.
deriv=_KDX*2	Assigns value from KDX multiplied by two to variable, deriv.
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: \_KDX=2 is invalid.

## Special Operands (Keywords)

The DMC-30000 provides a few additional operands which give access to internal variables that are not accessible by standard DMC-30000 commands.

Keyword	Function
_BGA	*Returns a 1 if motion on the axis, 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
_HMA	*Returns status of Home Switch (equals 0 or 1)
_LFA	Returns status of Forward Limit switch input (equals 0 or 1)
_LRA	Returns status of Reverse Limit switch input (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). 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=_LFA	Assign V1 the logical state of the Forward Limit Switch
v3=TIME	Assign V3 the current value of the time clock
v4=_HMA	Assign V4 the logical state of the Home input

---

## Arrays

For storing and collecting numerical data, the DMC-30000 provides array space for 3000 elements. The arrays are one dimensional and up to 6 different arrays may be defined. Each array element has a numeric range of 4 bytes of integer ( $2^{31}$ ) followed by two bytes of fraction ( $\pm 2,147,483,647.9999$ ).

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 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
DA posx[]	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[0]=7650.2	Assigns the first element of the array, 'speed' the value 7650.2
speed[0]=	Returns array element value
posx[9]=_TPX	Assigns the 10 <sup>th</sup> element of the array 'posx' the returned value from the tell position command.
con[1]=@COS[pos]*2	Assigns the second element of the array 'con' the cosine of the variable POS multiplied by 2.
timer[0]=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

The Galil Design Kit (GDK) software is recommended for downloading and uploading array data from the controller. The gclib API also provides function calls for downloading and uploading array data from the controller to/from a buffer or a file.

Arrays may also 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-30000 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 six types of data can be captured and stored in six 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 0 and 254 and designates 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
_AFA	Analog input
_DEA	2 <sup>nd</sup> encoder position (dual encoder)
_NO	Status bits
_OP	Output
_RLA	Latched position
_RPA	Commanded position
_SCA	Stop code
_TEA	Position error
_TI	Inputs
_TPA	Encoder position
_TSA	Switches (only bit 0-4 valid)
_TTA	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)

**NOTE:** The IN command has been removed from the DMC-30000 firmware. Variables should be entered by sending data directly from the host application.

### Sending Data from a Host

The DMC-30000 can accept ASCII strings from a host. This is the most common way to send data to the controller such as setting variables to numbers or strings. Any variable can be stored in a string format up to 6 characters by simply specifying defining that variable to the string value with quotes, for example:

varS = "STRING"

Will assign the variable 'varS' to a string value of "STRING".

To assign a variable a numerical value, the direct number is used, for example:

varN = 123456

Will assign the variable 'varN' to a number of 123,456.

All variables on the DMC-30000 controller are stored with 6 bytes of integer and 4 bytes of fractional data.

## Operator Data Entry Mode

The Operator Data Entry Mode provides for un-buffered data entry through the main RS-232 port. In this mode, the DMC-30000 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.

To capture and decode characters in the Operator Data Mode, the DMC-30000 provides special the following keywords:

Keyword	Function
P1CH	Contains the last character received
P1ST	Contains the received string
P1NM	Contains the received number
P1CD	Contains the status code: -1 mode disabled 0 nothing received 1 received character, but not <enter> 2 received string, not a number 3 received number

**NOTE:** The value of P1CD 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, P1CD< >3	Checks to see if status code is 3 (number received)
JP #P, P1CH="V"	Checks if last character received was a V
PR P1NM	Assigns received number to position
JS #XAXIS, P1ST="X"	Checks to see if received string is X

## Using Communication Interrupt

The DMC-30000 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 1
n = 1	Interrupt on <enter> Port 1
n = 2	Interrupt on any character Port 1
n = -1	Clear any characters in buffer

The #COMINT label is used for the communication interrupt. For example, the DMC-30000 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-30000 is used to jog the axis. 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.

<b>Instruction</b>	<b>Interpretation</b>
#AUTO	Label for Auto Execute
speedA=10000	Initial A speed
speedB=10000	Initial B speed
CI 2	Set Port 1 for Character Interrupt
JG speedA	Specify jog mode speed
BGX	Begin motion
#PRINT	Routine to print message to terminal
MG{P1} "TO CHANGE SPEEDS"	Print message
MG{P1} "TYPE A"	
MG{P1} "TYPE S TO STOP"	
#JOGLOOP	Loop to change Jog speeds
JG speedA	Set new jog speed
JP #JOGLOOP	
EN	End of main program
#COMINT	Interrupt routine
JP #A, P2CH="A"	Check for A
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
#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", P1CH{S}, "AXIS	Prompt for value
SPEED" {N}	
#NUMLOOP; CI-1	Check for enter
#NMLP	Routine to check input from terminal
JP #NMLP, P1CD<2	Jump to error if string
JP #ERROR, P1CD=2	Read value
val=P1NM	
EN	End subroutine
#ERROR; CI-1	Error Routine
MG "INVALID-TRY AGAIN"	Error message
JP #NMLP	
EN	End

## 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 RS-232 port, or {En} for the Ethernet port.

```
MG {P1} "Hello World"           Sends message to RS-232 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 {F5.0} {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} or {En}	Send message to RS-232 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-30000 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

#### Adding Leading Zeros from Response to Interrogation Commands

The leading zeros on data returned as a response to interrogation commands can be added by the use of the command, LZ. The LZ command is set to a default of 1.

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
FFFFB.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.

<b>Instruction</b>	<b>Interpretation</b>
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.

<b>Instruction</b>	<b>Interpretation</b>
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-30000 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/sec<sup>2</sup>. 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
MG "ENTER # OF REVOLUTIONS";n1=-1	Prompt for revs
#rev;JP#rev,n1=-1	Wait until user enters new value for n1
PR n1*2000	Convert to counts
MG "ENTER SPEED IN RPM";s1=-1	Prompt for RPMs
#spd;JP#spd,s1=-1	Wait for user to enter new value for s1
SP s1*2000/60	Convert to counts/sec
MG "ENTER ACCEL IN RAD/SEC2";a1=-1	Prompt for ACCEL
#acc;JP#acc,a1=-1	Wait for user to enter new value for a1
AC a1*2000/(2*3.14)	Convert to counts/sec2
BG	Begin motion
EN	End program

## Hardware I/O

### Digital Outputs

The DMC-30000 has 4-bit uncommitted digital outputs output port. Each bit 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
SB3	Sets bit 3 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, bit 1 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 15	Sets all bits of output port to one. $(2^0 + 2^1 + 2^2 + 2^3)$

The output port is useful for setting relays or controlling external switches and events during a motion sequence.

### **Example - Turn on output after move**

<b>Instruction</b>	<b>Interpretation</b>
#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 8.

### **Example - Using Inputs to control program flow**

<b>Instruction</b>	<b>Interpretation</b>
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-30000. High on input 1 means switch is in on position.

<b>Instruction</b>	<b>Interpretation</b>
#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. The controller has one auxiliary encoder which consists of two inputs, channel A and channel B. The auxiliary encoder input is mapped to the inputs 81 and 82.

The auxiliary encoder is a differential line receiver and can accept voltage levels between  $\pm 12$  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  $\frac{1}{2}$  of the full voltage range (for example, connect the - input to 5 volts if the signal is a 0 - 12 volt logic).

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

## **Input Interrupt Function**

The DMC-30000 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,p command. The m argument specifies the beginning input and n argument specifies the final input in the range. The o argument is an interrupt bit mask. The p argument is used to specify the required input state for interrupt trigger. 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.

<b>IMPORTANT</b>	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	Loop until Interrupt cleared
JP #LOOP,@IN[1]=0	Specify new speeds
JG 15000,10000	Wait 300 milliseconds
WT 300	Begin motion on A and B axes
BG AB	Return from Interrupt subroutine
RI	

#### Jumping back to main program with #ININT

To jump back to the main program using the JP command, the RI command must be issued in a subroutine and then the ZS command must be issued prior to the JP command. See Application Note # 2418 for more information.

[www.galil.com/learn/application-notes](http://www.galil.com/learn/application-notes)

## Analog Inputs

The DMC-30000 provides two 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 2. The resolution of the Analog-to-Digital conversion is 12-bit. 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 5V, motor must move 5000 counts.

Method: Read the analog input and command A to move to that point.

<b>Instruction</b>	<b>Interpretation</b>
#POINTS	Label
SP 7000	Speed
AC 80000;DC 80000	Acceleration
#LOOP	
VP=@AN[1]*1000	Read 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 proportion to the position error.

<b>Instruction</b>	<b>Interpretation</b>
#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

### Example – Low Pass Digital Filter for the Analog inputs

Because the analog inputs on the Galil controller can be used to close a position loop, they have a very high bandwidth and will therefore read noise that comes in on the analog input. Often when an analog input is used in a motion control system, but not for closed loop control, the higher bandwidth is not required. In this case a simple digital filter may be applied to the analog input, and the output of the filter can be used for in the motion control application. This example shows how to apply a simple single pole low-pass digital filter to an analog input. This code is commonly run in a separate thread (XQ#filt,1 – example of executing in thread 1).

```

#filt
REM an1 = filtered output. Use this instead of @AN[1]
an1=@AN[1];'set initial value
REM k1+k2=1 this condition must be met
REM use division of m/2^n for elimination of round off
REM increase k1 = less filtering
REM increase k2 = more filtering
k1=32/64;k2=32/64
AT0;'set initial time reference
#loop
REM calculate filtered output and then way 2 samples from last
REM time reference (last AT-2,1 or AT0)
an1=(k1*@AN[1])+(k2*an1);AT-2,1
JP#loop

```

# 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:

$$4000/2\pi = 637 \text{ count/inch}$$

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 Figure 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

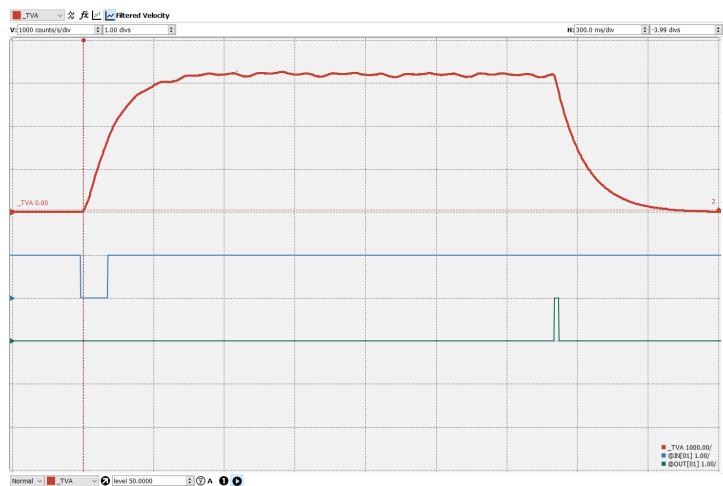


Figure 7.1: Motor Velocity and the Associated Input/Output signals

## 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:

$$3000 \text{ rpm} = 50 \text{ rev/sec} = 200000 \text{ count/sec}$$

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:

$$\text{Speed} = 20000 \times \text{VIN}$$

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
DPO	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  $\pm 2$  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  $\pm 2$  counts. Often, this is performed in one correction cycle.

**Example:**

INSTRUCTION	FUNCTION
#A	Label
DPO	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	

# Chapter 8 Hardware & Software Protection

---

## Introduction

The DMC-30000 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-30000 is an integral part of the machine, the engineer should design his overall system with protection against a possible component failure on the DMC-30000. Galil shall not be liable or responsible for any incidental or consequential damages.

---

## Hardware Protection

The DMC-30000 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. This signal also goes low when the watch-dog timer is activated, or upon reset.

#### 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 an error condition and the Error Light on the controller will be illuminated. For details on the reasons why the error output would be active see Error Light (Red LED) in Chapter 9.

### 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 5<sup>th</sup> field of the CN command. See the CN command in the Command Reference for more information.

## **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 DMC-300xx models behave when the ELO is triggered, see the appendix section corresponding to your particular model.

## **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. The OE command can also be configured so that the axis will be disabled upon the activation of a limit switch.

## **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. The OE command can also be configured so that the axis will be disabled upon the activation of a limit switch.

## Software Protection

The DMC-30000 provides a programmable error limit as well as encoder failure detection. It is recommended that both the position error and encoder failure detection be used when running servo motors with the DMC-30000. Along with position error and encoder failure detection, then DMC-30000 has the ability to have programmable software limit.

## Position Error

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 Set X-axis error limit for 200

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:

<b>Signal or Function</b>	<b>State if Error Occurs</b>
# POSERR	Jumps to automatic excess position error subroutine
Error Light	Turns on
OE Function	Shuts motor off if OE1 or OE3
AEN Output Line	Switches to Motor Off state

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.

## Encoder Failure detection

The encoder failure detection on the controller operates based upon two factors that are user settable, a threshold of motor command output (OV), a time above that threshold (OT) in which there is no more than 4 counts of

change on the encoder input for that axis. The encoder failure detection is activated with the OA command. When an encoder failure is detected and OA is set to 1 for that axis, the same conditions will occur as a position error.

### Conditions for proper operation of Encoder Failure detection

- The axis must have a non-zero KI setting order to detect an encoder failure when the axis is not profiling.
- The IL command must be set to a value greater than the OV setting
- The TL command must be set to a value greater than the OV setting

#### Example:

The A axis is setup with the following settings for encoder failure detection:

```
OA 1  
OT 500  
OV 3  
OE 1  
ER 1000
```

The A axis is commanded to move 300 counts, but the B channel on the encoder has failed and no longer operates. Because the ER setting is greater than the commanded move, the error will not be detected by using the OE and ER commands, but this condition will be detected as an encoder failure. When the axis is commanded to move a 300 counts, the position error will cause the motor command voltage to be increased to a value that will be greater than the OV value, 3 volts in this case. Once the motor command output is greater than the OV threshold for more than the 500ms defined by the OT command AND there has been less than 4 counts of change on the encoder, then the controller will turn off that axis due to an encoder failure. The motor will have moved some distance during this operation, but it will be shut down before a full runaway condition occurs.

### Using Encoder Failure to detect a hard stop or stalled motor

The encoder failure detection can also be used to detect when an axis is up against a hard stop. In this scenario the motor command will be commanded above the OV threshold, but because the motor is not moving the controller will detect this scenario as an encoder failure.

## Programmable Position Limits

The DMC-30000 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-30000 will not accept position commands beyond the limit. Motion beyond the limit is also prevented.

#### Example:

DP 0	Define Position
BL -2000	Set Reverse position limit
FL 2000	Set Forward position limit
JG 2000	Jog
BG X	Begin

*(motion stops at forward limits)*

## Off-On-Error

The DMC-30000 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, 2 or 3 for that axis. To disable this function, specify 0 for the axes. When this function is enabled, the specified motor will be disabled under the following 4 conditions:

1. The position error for the specified axis exceeds the limit set with the command, ER
2. A hardware limit is reached

3. The abort command is given
4. 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      Enable off-on-error

## 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, a encoder failure is detected, or the abort input is triggered. 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 cleared.

**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-30000 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
4. Error Light (Red LED)

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.  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).  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.  2. Encoder is damaged  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.  Replace encoder  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.  2. Encoder configuration incorrect.  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.  Check CE command  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 Avoid placing power cables near encoder cables Avoid Ground Loops Use differential encoders 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

## Error Light (Red LED)

The red error LED has multiple meanings for Galil controllers. Here is a list of reasons the error light will come on and possible solutions:

### Under Voltage

If the controller is not receiving enough voltage to power up.

### Under Current

If the power supply does not have enough current, the red LED will cycle on and off along with the green power LED.

### Position Error

If any axis that is set up as a servo (MT command) has a position error value (TE) that exceeds the error limit (ER) - the error light will come on to signify there is an axis that has exceeded the position error limit. Use a DP\*=0 to set all encoder positions to zero or a SH (Servo Here) command to eliminate position error.

### Invalid Firmware

If the controller is interrupted during a firmware update or an incorrect version of firmware is installed - the error light will come on. The prompt will show up as a greater than sign ">" instead of the standard colon ":" prompt. Use Galil Design Kit (GDK) software to install the correct version of firmware to fix this problem.

## **Self Test**

During the first few seconds of power up, it is normal for the red LED to turn on while it is performing a self test. If the self test detects a problem such as corrupted memory or damaged hardware - the error light will stay on to signal a problem with the board. To fix this problem, a Master Reset may be required. The Master Reset will set the controller back to factory default conditions so it is recommended that all motor and I/O cables be removed for safety while performing the Master Reset. Cables can be plugged back in after the correct settings have been loaded back to the controller (when necessary). To perform a Master Reset - find the jumper location labeled MR or MR on the controller and put a jumper across the two pins. Power up with the jumper installed. The Self-Test will take slightly longer - up to 5seconds. After the error light shuts off, it is safe to power down and remove the Master Reset jumper. If performing a Master Reset does not get rid of the error light, the controller may need to be sent back to the factory to be repaired. Contact Galil for more information.

# 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 Figure 10.1.

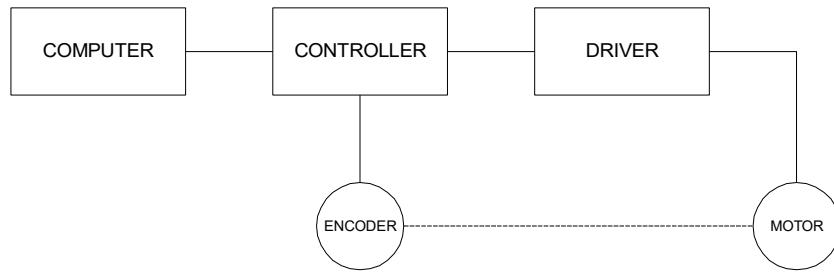


Figure 10.1: Elements of Servo Systems

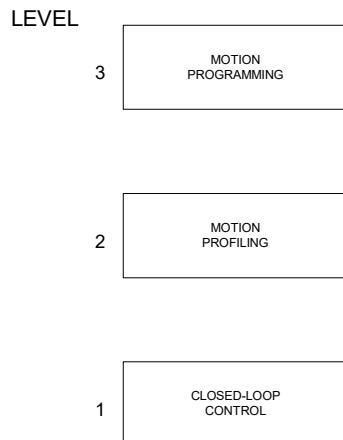
The operation of such a system can be divided into three levels, as illustrated in Figure 10.2. The levels are:

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.

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.

```

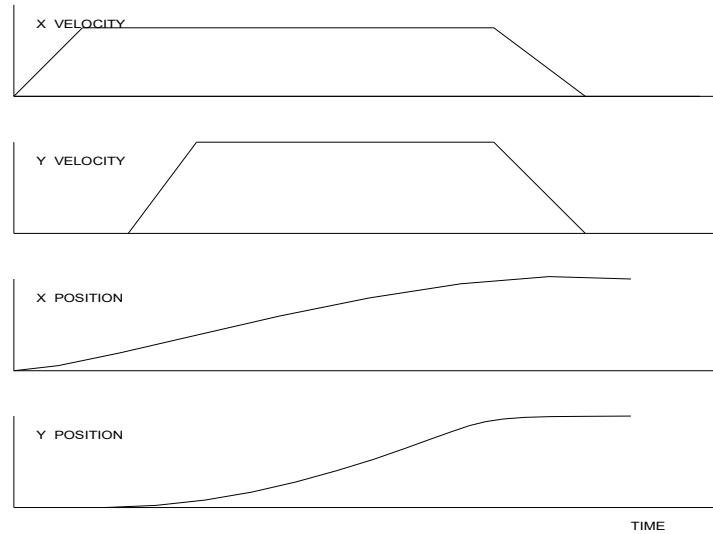
PR 6000
SP 20000
AC 200000
BG X
EN

```

This program corresponds to the velocity profiles shown in Figure 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 Figure 10.4. The mathematical model of the various components is given below.

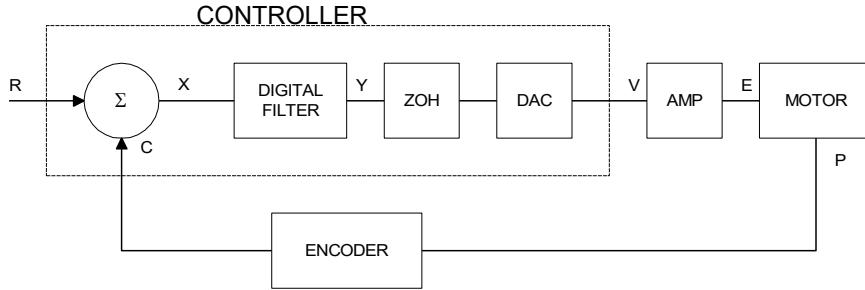


Figure 10.4: Functional Elements of a Motion Control System

## 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  $K_v$  [V/V]. The transfer function relating the input voltage,  $V$ , to the motor position,  $P$ , is

$$P/V = K_v / [K_t S(ST_m + 1)(ST_e + 1)]$$

where

$$T_m = RJ/K_t^2 \quad [\text{s}]$$

and

$$T_e = L/R \quad [\text{s}]$$

and the motor parameters and units are

$K_t$	Torque constant [Nm/A]
$R$	Armature Resistance $\Omega$
$J$	Combined inertia of motor and load [ $\text{kg}\cdot\text{m}^2$ ]
$L$	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:

$$K_t = 14.16 \text{ oz-in/A} = 0.1 \text{ Nm/A}$$

$$R = 2 \Omega$$

$$J = 0.0283 \text{ oz-in-s}^2 = 2 * 10^{-4} \text{ kg . m}^2$$

$$L = 0.004H$$

Then the corresponding time constants are

$$T_m = 0.04 \text{ sec}$$

and

$$T_e = 0.002 \text{ sec}$$

Assuming that the amplifier gain is  $K_v = 4$ , 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  $K_a$ . The resulting transfer function in this case is

$$P/V = K_a K_t / Js^2$$

where  $K_t$  and  $J$  are as defined previously. For example, a current amplifier with  $K_a = 2 \text{ A/V}$  with the motor described by the previous example will have the transfer function:

$$P/V = 1000/s^2 [\text{rad/V}]$$

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 Figure 10.5. Note that the transfer function between the input voltage  $V$  and the velocity  $\omega$  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,  $T_1$ , equals

$$T_1 = J/K_a K_t K_g$$

This leads to the transfer function

$$P/V = 1/[K_g s(T_1+1)]$$

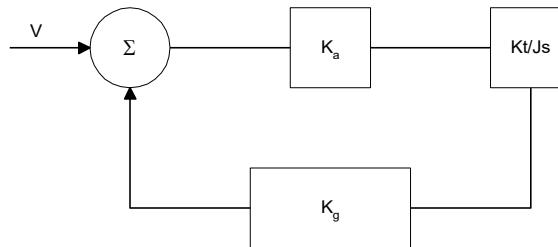


Figure 10.5: Elements of velocity loops

The resulting functions derived above are illustrated by the block diagram of Figure 10.6.

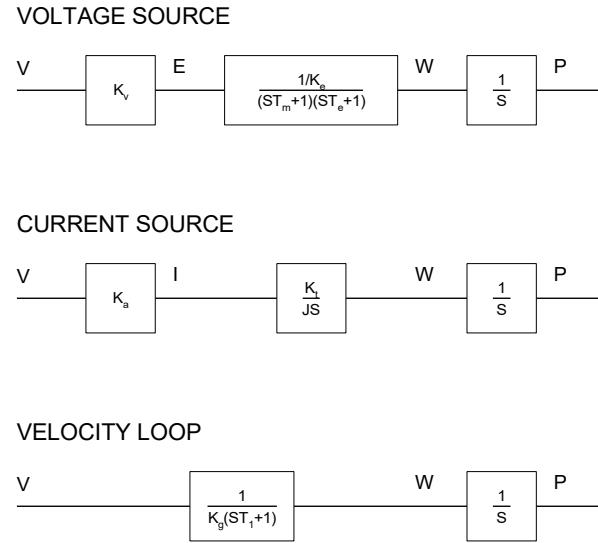


Figure 10.6: Mathematical model of the motor and amplifier in three operational modes

## 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

$$K_f = 4N/2\pi \quad [\text{count}/\text{rad}]$$

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  $\pm 10V$  or  $20V$ . Therefore, the effective gain of the DAC is

$$K = 20/65536 = 0.0003 \quad [\text{V}/\text{count}]$$

## 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:

$$\begin{aligned} \text{PID} \quad D(z) &= \frac{K(z - A)}{Z} + \frac{Cz}{Z - 1} \\ \text{Low-pass} \quad L(z) &= \frac{1 - B}{Z - B} \end{aligned}$$

$$\text{Notch } N(z) = \frac{(Z - z)(Z - \bar{z})}{(Z - p)(Z - \bar{p})}$$

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:

$$\begin{aligned} K &= (KP + KD) \\ A &= KD/(KP + KD) \\ C &= KI \\ B &= PL \end{aligned}$$

The PID and low-pass elements are equivalent to the.

continuous transfer function G(s).

$$G(s) = (P + sD + I/s) \cdot a / (s + a)$$

where,

$$\begin{aligned} P &= KP \\ D &= T \cdot KD \\ I &= KI/T \\ a &= \frac{1}{T} \ln\left(\frac{1}{B}\right) \end{aligned}$$

where T is the sampling period, and B is the pole setting

For example, if the filter parameters of the DMC-30000 are

$$\begin{aligned} KP &= 16 \\ KD &= 144 \\ KI &= 2 \\ PL &= 0.75 \\ T &= 0.001 \text{ s} \end{aligned}$$

the digital filter coefficients are

$$\begin{aligned} K &= 160 \\ A &= 0.9 \\ C &= 2 \\ a &= 250 \text{ rad/s} \end{aligned}$$

and the equivalent continuous filter, G(s), is

$$G(s) = [16 + 0.144s + 2000/s] \cdot 250 / (s+250)$$

The notch filter has two complex zeros, z and , and two complex poles, p and .

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-30000 controller and the following parameters:

$K_t = 0.1$	Nm/A	Torque constant
$J = 2 * 10^{-4}$	kg.m <sup>2</sup>	System moment of inertia
$R = 2$	$\Omega$	Motor resistance
$K_a = 4$	Amp/Volt	Current amplifier gain
$K_P = 12.5$		Digital filter gain
$K_D = 245$		Digital filter zero
$K_I = 0$		No integrator
$N = 500$	Counts/rev	Encoder line density
$T = 1$	ms	Sample period

The transfer function of the system elements are:

### Motor

$$M(s) = P/I = K_t/Js^2 = 500/s^2 \text{ [rad/A]}$$

### Amp

$$K_a = 4 \text{ [Amp/V]}$$

### DAC

$$K_d = 0.0003 \text{ [V/count]}$$

### Encoder

$$K_f = 4N/2\pi = 318 \text{ [count/rad]}$$

### ZOH

$$2000/(s+2000)$$

### Digital Filter

$$K_P = 12.5, K_D = 245, T = 0.001$$

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 Figure 10.7.

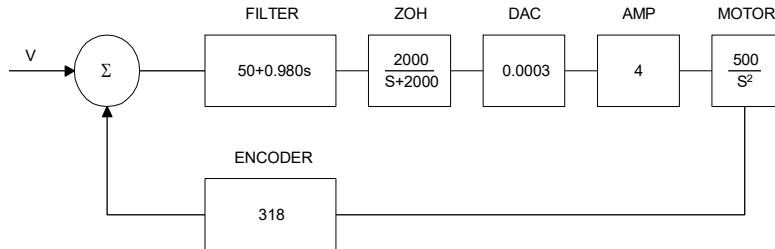


Figure 10.7: Mathematical model of the control system

The open loop transfer function,  $A(s)$ , is the product of all the elements in the loop.

$$A(s) = 390,000 (s+51)/[s^2(s+2000)]$$

To analyze the system stability, determine the crossover frequency,  $\omega_c$  at which  $A(j\omega_c)$  equals one. This can be done by the Bode plot of  $A(j\omega_c)$ , as shown in Figure 10.8.

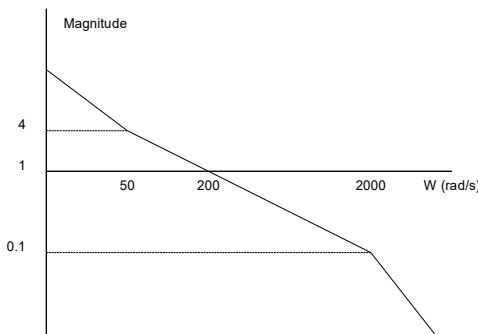


Figure 10.8: Bode plot of the open loop transfer function

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.

$$A(j200) = 390,000 (j200+51)/[(j200)^2 \cdot (j200 + 2000)]$$

$$\alpha = \text{Arg}[A(j200)] = \tan^{-1}(200/51) - 180^\circ - \tan^{-1}(200/2000)$$

$$\alpha = 76^\circ - 180^\circ - 6^\circ = -110^\circ$$

Finally, the phase margin, PM, equals

$$PM = 180^\circ + \alpha = 70^\circ$$

As long as PM is positive, the system is stable. However, for a well damped system, PM should be between 30° and 45°. The phase margin of 70° given above indicated over-damped 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-30000 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,  $\omega_c$ , 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:

$K_t = 0.2$	Nm/A	Torque constant
$J = 2 * 10^{-4}$	kg.m <sup>2</sup>	System moment of inertia
$R = 2$	$\Omega$	Motor resistance
$K_a = 2$	Amp/Volt	Current amplifier gain
$N = 1000$	Counts/rev	Encoder line density

The DAC of the DMC-30000 outputs  $\pm 10V$  for a 16-bit command of  $\pm 32768$  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

$$M(s) = P/I = K_t/Js^2 = 1000/s^2$$

#### Amp

$$K_a = 2 \quad [\text{Amp/V}]$$

#### DAC

$$K_d = 10/32768 = .0003$$

#### Encoder

$$K_f = 4N/2\pi = 636$$

#### 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,  $L(s)$ .

$$L(s) = M(s) K_a K_d K_f H(s) = 3.17 \cdot 10^6 / [s^2(s+2000)]$$

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  $\omega_c = 500$ .

$$L(j500) = 3.17 \cdot 10^6 / [(j500)^2 (j500+2000)]$$

This function has a magnitude of

$$|L(j500)| = 0.00625$$

and a phase

$$\text{Arg}[L(j500)] = -180^\circ - \tan^{-1}[500/2000] = -194^\circ$$

$G(s)$  is selected so that  $A(s)$  has a crossover frequency of 500 rad/s and a phase margin of 45 degrees. This requires that

$$|A(j500)| = 1$$

$$\text{Arg}[A(j500)] = -135^\circ$$

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

$$\arg [G(j500)] = \arg [A(j500)] - \arg [L(j500)] = -135^\circ + 194^\circ = 59^\circ$$

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

$$\arg [G(j500)] = \tan^{-1}[500D/P] = 59^\circ$$

The solution of these equations leads to:

$$P = 160\cos 59^\circ = 82.4$$

$$500D = 160\sin 59^\circ = 137$$

Therefore,

$$D = 0.274$$

and

$$G = 82.4 + 0.274s$$

The function  $G$  is equivalent to a digital filter of the form:

$$D(z) = KP + KD(1-z^{-1})$$

where

$$P = KP$$

$$D = KD \cdot T$$

and

$$KD = D/T$$

Assuming a sampling period of  $T=1\text{ms}$ , the parameters of the digital filter are:

$$KP = 82.4$$

$$KD = 274$$

The DMC-30000 can be programmed with the instruction:

$$KP\ 82.4$$

$$KD\ 274$$

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-30000**

Digital  $D(z) = [K(z-A/z) + Cz/(z-1)] \cdot (1-B)/(z-B)$

$$KP, KD, KI, PLK = (KP + KD)$$

$$A = KD/(KP+KD)$$

$$C = KI$$

$$B = PL$$

Digital

$$D(z) = [KP + KD(1-z^{-1}) + KI/2(1-z^{-1})] \cdot (1-PL)/(Z-PL)$$

Continuous

$$G(s) = (P + Ds + I/s) \cdot a/(s+a)$$

PID, T

$$P = KP$$

$$D = T * KD$$

$$I = KI / T$$

$$a = 1/T \ln(1/PL)$$

# Appendices

## Electrical Specifications

<b>NOTE</b>	Electrical specifications are only valid once controller is out of reset.
-------------	---

### Servo Control

Motor command line	±10 V analog signal Resolution: 16-bit DAC or 0.0003 volts 3 mA maximum. Output impedance – 500 Ω
Main and auxiliary encoder inputs	TTL compatible, but can accept up to ±12 volts Quadrature phase on CHA, CHB Single-ended or differential Maximum A, B edge rate: 15 MHz Minimum IDX pulse width: 30 nsec

### Stepper Control

Step (MF2+ MF2-)	Differential (0-5 V <sup>1</sup> ) level at 50% duty cycle 3,000,000 pulses/sec maximum frequency
Direction (MF4+ MF4- )	Differential (0-5 V <sup>1</sup> )

<sup>1</sup> DMC-31000 units shipped before February 2013 and DMC-30000 units shipped before April 2013 have a different specification, see Multi-Function Pins (MF), pg 34.

### Input / Output

Opto-isolated Inputs: DI[8:1], Limit switches, home, abort, reset	2.2 kΩ 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.
Analog Inputs: AI[2:1]	0-5V Standard, 12-bit Analog-to-Digital converter ±10 volts and 16-bit with DMC-31xxx option

Analog outputs, AO[2:1]	AO1 can act as motor command line or as a general use analog output AO[2:1] have the same specifications as the motor command line
Opto-isolated Digital Outputs: DO[4:1]	4mA sinking Options: 25 mA sinking, 500mA sourcing;
Auxiliary Inputs as Uncommitted Inputs: DI[82:81]	The auxiliary pins can be used as uncommitted inputs and are assigned to the following bits: DI81, DI82 These inputs have the same specifications as listed above for encoder inputs.

## Input Power Requirements

Controller Model	Input Voltage Requirement	Current/Power Requirement <sup>1</sup>
DMC-30010	+5 VDC ( $\pm 5\%$ ) +12VDC ( $\pm 10\%$ ) -12VDC ( $\pm 10\%$ )	0.5 Amps 0.05 Amps 0.05 Amps
DMC-30011	9-48 VDC	3 Watts
DMC-30012	20-80 VDC	5 Watts <sup>2</sup>
DMC-30014	15-40 VDC	3 Watts <sup>2</sup>
DMC-30016	12-30 VDC	3 Watts <sup>2</sup>
DMC-30017	20-80 VDC	5 Watts <sup>2</sup>

<sup>1</sup> Power Requirements the required power with no external I/O connections

<sup>2</sup> Does not include power for the motor. The power supply should be sized based upon load and motor specifications.

## +5, $\pm 12$ V Power Output Specifications

Output Voltage	Tolerance	Max Current Output
+5V	$\pm 5\%$	0.5A
+12V	$\pm 10\%$	10mA
-12V	$\pm 10\%$	10mA

## Certifications

The DMC-30000 is certified for the following when the product or package is marked.

### UL

<https://www.galil.com/about/certifications>

---

## Performance Specifications

Minimum servo loop update rate	125 $\mu$ sec
Position Accuracy	$\pm 1$ quadrature count
Velocity Accuracy	
Long Term	Phase-locked, better than 0.005%
Short Term	System dependent
Position Range	$\pm 2147483647$ counts per move
Maximum Velocity	
Servo motor	15,000,000 counts/sec
Stepper	3,000,000 pulses/sec
Velocity Resolution	2 counts/sec
Motor Command Resolution	16-bit or 0.0003 V
Variable Range	$\pm 2$ billion
Variable Resolution	$1 \times 10^{-4}$
Number of Variables	254
Array Size	3000 elements, 6 arrays
Program Size <sup>2</sup>	1000 lines x 80 characters
Number of threads <sup>2</sup>	6
Maximum Burn Cycles	
Rev A & B	10,000
Rev C <sup>1</sup>	100,000

<sup>1</sup> Rev C boards are those with serial numbers BZ-579 and greater. In addition, Rev C boards are required to have firmware revision 1.1b or greater in order to have this extended burn cycle capability.

<sup>2</sup> DMC-30000 models with firmware revisions 1.1c and earlier have a greater limitation on programming memory and multi-threading. Please refer to the DMC-30000 firmware release notes for details:

[www.galil.com/fw/pub/30000.dmc/doc/rn.html](http://www.galil.com/fw/pub/30000.dmc/doc/rn.html)

---

# Ordering Options

## Overview

The DMC-30000 can be ordered in many different configurations and with different options. This section provides information regarding the different "Y" options (See Chapter 1 Overview, Part Numbers, on page 2) available on the DMC-30000 motion controller, interconnect modules and internal amplifiers. For information on pricing and how to order your controller with these options, see our DMC-30000 part number generator on our website.

[www.galil.com/order/part-number-generator/dmc-3x01x](http://www.galil.com/order/part-number-generator/dmc-3x01x)

## I/O Options

### 4-20mA – 4-20mA Analog Inputs

The 4-20mA option converts the analog inputs into 4-20mA analog inputs. This is accomplished by installing 237  $\Omega$  precision resistors between the analog inputs and ground. The equation for calculating the current is:

$$I_{ma} = 2.11 \text{ V}$$

Where  $I_{ma}$  = current in mA

V = Voltage reading from DMC-30000

Part number ordering example: DMC-30010-CARD(4-20mA)

### LSNK – 25mA Sinking Outputs

The LSNK option modifies the digital outputs on the DMC-30000 to be capable of sinking up to 25mA per output. For detailed information see the 25mA Sinking Optoisolated Outputs (LSNK) section in Chapter 3 Connecting Hardware.

Part number ordering example: DMC-30010-CARD-(LSNK)

### HSRC – 500mA Sourcing Outputs

The HSRC option modifies the digital outputs on the DMC-30000 to be capable of sourcing up to 500mA per output. For detailed information see the 500mA Sourcing Optoisolated Outputs (HSRC) section in Chapter 3 Connecting Hardware.

Part number ordering example: DMC-30010-CARD(HSRC)

## Feedback Options

### TRES – Encoder Termination Resistors

The TRES option provides termination resistors on all of the main and auxiliary encoder inputs on the DMC-30000 motion controller. The termination resistors are  $120\ \Omega$ , and are placed between the positive and negative differential inputs on the Main A, B, Index channels as well as the Auxiliary A and B channels as in Figure A.1.

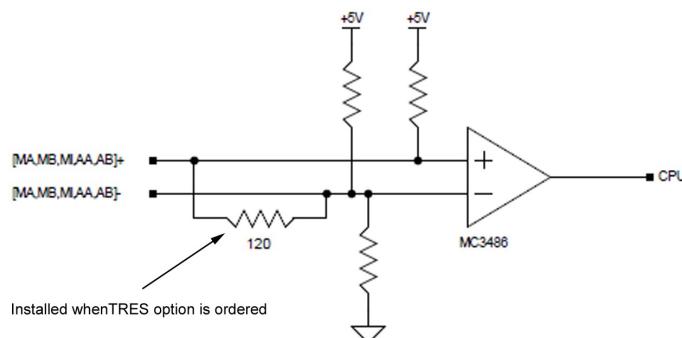


Figure A.1: Encoder Inputs with -TRES option

**Single-Ended Encoders:** Single-ended encoders will not operate correctly with the termination resistors installed. If a combination of differential encoder inputs with termination resistors and single ended encoders is required on the same controller, contact Galil directly.

**DMC-31xxx:** When ordered with the DMC-31xxx (Sin/Cos Encoder option), termination resistors will be placed on the Aux Encoder inputs. By default the DMC-31xxx already has termination resistors on the Main Encoder Inputs.

Part number ordering example: DMC-30010-CARD(TRES)

### SER – BiSS and SSI Absolute Encoder Option

The DMC-3x01x controller can be configured to support BiSS and SSI encoders.

For the BiSS protocol, this option configures the DMC-3x01x with hardware and firmware that supports acquiring position data from BiSS-B and BiSS-C absolute encoders. Galil supports point-to-point configuration and one Server device per axis of the DMC-3x01x. Galil does not support bus configuration. For more information, please contact Support at Galil.

For the SSI protocol, this option configures the DMC-3x01x with hardware and firmware that supports acquiring position data from SSI absolute encoders. Galil supports one Server device per axis of the DMC-3x01x. For more information, please contact Support at Galil.

For more information, see the [SS](#) command for BiSS and the [SI](#) command for SSI in the DMC-3x01x Command Reference. For pin-outs, refer to the [Multi-Functional Pins \(MFn±\)](#) section.

Part number ordering example: DMC-30010-CARD(SER)

### HALLF - Filtered Hall Sensor Inputs

The HALLF option will place a capacitor between the hall input and digital GND to filter unwanted noise. This results in cleaner, more reliable hall sensor reads. The HALLF option is only available for Galil's internal PWM amplifiers.

Part number ordering example: DMC-30012-BOX(HALLF)

## Communication Options

### RS-422 – Serial Port Serial Communication

The default serial configuration for the DMC-30000 is to have RS-232 communication on the serial port. The controller can be ordered to have RS-422. RS-422 communication is a differentially driven serial communication protocol that should be used when long distance serial communication is required in an application.

See DMC-30000 RS-422 Serial Port Specifications, pg 166 for pin-outs and details of the RS-422 options.

### JP2 – RS-422 Termination Jumpers

Label	Function (If jumpered)
RXD	RS-422 Option Only: Connects a 120Ohm Termination resistor between the differential "Receive" inputs on the Serial port. Pins 2 and 7 on RS-422 Serial Port.
CTS	RS-422 Option Only: Connects a 120Ohm Termination resistor between the differential "Clear To Send" inputs on the Serial port. Pins 1 and 6 on RS-422 Serial Port.

Part number ordering example: DMC-30010-CARD(422)

## Mounting Options

### DIN – DIN Rail Mounting

The DIN option on the DMC-30000 motion controller provides DIN rail mounts on the base of the controller. This will allow the controller to be mounted to any standard DIN rail. Requires -BOX option.

Part number ordering example: DMC-30010-BOX-DIN

## Internal Amplifier Options

### ISCNTL – Isolate Controller Power

The ISCNTL option isolates the power input for the controller from the power input of the amplifiers. When using this option, it is important that users connect the returns of the power supplies together. With this option, the power is brought in through the 2 pin Molex connector on the side of the controller as shown in the Power Wiring Diagrams section of the Appendix. This option is not valid when Galil amplifiers are not ordered with the DMC-30000.

Part number ordering example: DMC-30012-BOX(ISCNTL)

### SR90 – 90V Shunt Regulator Option

The SR90 is a shunt regulator for the DMC-30000 controller and internal amplifiers. The purpose of the shunt regulator is to protect the DMC-30000 from an over-voltage condition. A blocking diode should be used to protect the power supply. Contact Galil for power supply protection recommendations. To calculate if a given system requires a Shunt Regulator, see Application Note #5448: "Shunt Regulator Operation" linked below:

[www.galil.com/learn/application-notes](http://www.galil.com/learn/application-notes)

The SR90 is installed inside the box of the DMC-30000 controller, so it does not effect the form of the unit.

With the -SR90 option, the shunt regulator activates when the voltage supplied to the amplifier rises above 90V. When activated, the power from the power supply is dissipated through a 5Ω 20W power resistor.

Part number ordering example: DMC-30012-BOX-SR90

<b>WARNING</b>	<p>If the ISCNTL option is ordered with the SR90 option, never apply power to the amplifier if the controller is not powered.</p> <p>When powering the unit ON, apply power to the controller first, then the amplifier. When powering the unit OFF, power down the amplifier first, then the controller.</p>
----------------	---

### SR190 – 190V Shunt Regulator Option

The SR190 option is required upon order of the 160V option. For all other configurations, use SR90 if a shunt regulator is required.

The SR190 is a shunt regulator for the DMC-30000 controller and internal amplifiers. The purpose of the shunt regulator is to protect the DMC-30000 from an over-voltage condition. A blocking diode should be used to protect the power supply. Contact Galil for power supply protection recommendations.

The SR190 is installed inside the box of the DMC-30000 controller, so it does not effect the form of the unit.

With the -SR190 option, the shunt regulator activates when the voltage supplied to the amplifier rises above 190V. When activated, the power from the power supply is dissipated through a 10Ω 20W power resistor.

Part number ordering example: DMC-30012-BOX(160V,TRES,ISCNTL)-SR190

<b>WARNING</b>	<p>If the 160V option is ordered, never apply power to the amplifier if the controller is not powered.</p> <p>When powering the unit ON, apply power to the controller first, then the amplifier. When powering the unit OFF, power down the amplifier first, then the controller.</p>
----------------	--

## **INVELO – Inverted ELO (Electronic Lock-Out) Logic**

This option will invert the logic for the ELO input. With this option, current flowing through the ELO circuit (normally closed) is normal operation and will allow Galil internal amplifiers to be powered. An open circuit or no current flowing will activate the ELO and cut power to the amplifier.

Part number ordering example: DMC-30012-BOX(INVELO)

## **160V – 160V DC Max Amplifier Input Voltage**

The 160V option is available on the DMC-3x012 controller and internal amplifier configurations. This option allows the amplifier to support a 160VDC power bus. The 160V option *must* be ordered with the IS\_CNTL (Isolate Controller Power), SR190 (190V Shunt Regulator), and TRES (Termination Resistor) options. Contact Galil if the 160V option is required with DMC-3x017.

The following are the modified specifications for the DMC-3x012 with the 160V option:

Amplifier Supply Voltage	20 VDC – 160 VDC
Controller Supply Voltage	20 VDC – 80 VDC
Maximum Continuous Current	5 Amps
Maximum Peak Current	8 Amps
Amplifier Gains	0.2 A/V, 0.4 A/V, 0.8 A/V
PWM Frequency	20 kHz

Table A.1: DMC-3x012 160V option Amplifier Electrical Specifications

Part number ordering example: DMC-30012-BOX(160V,TRES,IS\_CNTL)-SR190

<b>WARNING</b>	<p>If the 160V option is ordered, never apply power to the amplifier if the controller is not powered.</p> <p>When powering the unit ON, apply power to the controller first, then the amplifier.</p> <p>When powering the unit OFF, power down the amplifier first, then the controller.</p>
----------------	---

## **Miscellaneous Options**

### **RTC – Real Time Clock**

The DMC-30000 provides a real time clock feature. The RTC option provides an extended feature set. For details see the Real Time Clock section in Chapter 6.

Real time clock	DMC-30000	DMC-30000(RTC)
RT providing Hours, Minutes, Seconds	Yes	Yes
RY providing Year, Month of year, Day of month, Day of week	No	Yes
Settable via TIME protocol server (IH and RO commands)	Yes	Yes
Clock persists through DMC power loss	No	Yes
C No-power clock battery life	N/A	1 week

Part number ordering example: DMC-30010-BOX(RTC)

### **MO – Motor Off Jumpers Installed**

When a jumper is installed on the “MO” pins, the controller will be powered up in the “motor off” state. This option will cause jumper to be installed at the factory.

Part number ordering example: DMC-30012-BOX(MO)

# Power Connector Part Numbers

## Overview

The DMC-30000 uses different connectors depending upon the type of controller used. The following section details the part numbers used on the controller for the different ordering options. Table A.2: Connector Part Numbers details the connector part numbers used on the DMC-30000 series controllers. Table A.3: Connectors listed by DMC-30000 part number lists the on-board connectors for the different DMC-30000 controller options.

On Board Connector	Common Mating Connectors	Crimp Part Number	Type
Molex# 39-31-0040	Molex# 39-01-2045	Molex# 44476-3112	4 Position
Molex# 39-31-0020	Molex# 39-01-2025	Molex# 44476-3112	2 Position
TE Connectivity# 5-104362-1	Molex# 50-57-9402 Adam-Tech: CDH-02 Oupiin:4077-02HB (1k Min)	Molex# 16-02-0103 Adam-Tech: CDH-C-B (Bulk) Adam-Tech: CDH-C-R (Reel) Oupiin: 4077-PIN-T-T15K (15k Reel)	2 Position
TE Connectivity# 5-104362-3	Molex# 50-57-9404 Adam-Tech: CDH-04 Oupiin: 4077-04HB (1k Min)	Molex# 16-02-0103 Adam-Tech: CDH-C-B (Bulk) Adam-Tech: CDH-C-R (Reel) Oupiin: 4077-PIN-T-T15K (15k Reel)	4 Position

Table A.2: Connector Part Numbers

DMC-30000 Part Number	On Board Connector	Type
DMC-30010	Power	TE Connectivity# 5-104362-3
DMC-30011	Power	TE Connectivity# 5-104362-1
DMC-30012	Power	Molex# 39-31-0020
	Motor	Molex# 39-31-0040
DMC-30014	Power	Molex# 39-31-0020
	Motor	Molex# 39-31-0040
DMC-30016	Power	Molex# 39-31-0020
	Motor	Molex# 39-31-0040
DMC-30017	Power	Molex# 39-31-0020
	Motor	Molex# 39-31-0040
DMC-30012(ISCNTL)	Power (Controller)	TE Connectivity# 5-104362-1
	Power (Amplifier)	Molex# 39-31-0020
	Motor	Molex# 39-31-0040
DMC-30014(ISCNTL)	Power (Controller)	TE Connectivity# 5-104362-1
	Power (Amplifier)	Molex# 39-31-0020
	Motor	Molex# 39-31-0040
DMC-30016(ISCNTL)	Power (Controller)	TE Connectivity# 5-104362-1
	Power (Amplifier)	Molex# 39-31-0020
	Motor	Molex# 39-31-0040
DMC-30017(ISCNTL)	Power (Controller)	TE Connectivity# 5-104362-1
	Power (Amplifier)	Molex# 39-31-0020
	Motor	Molex# 39-31-0040

Table A.3: Connectors listed by DMC-30000 part number

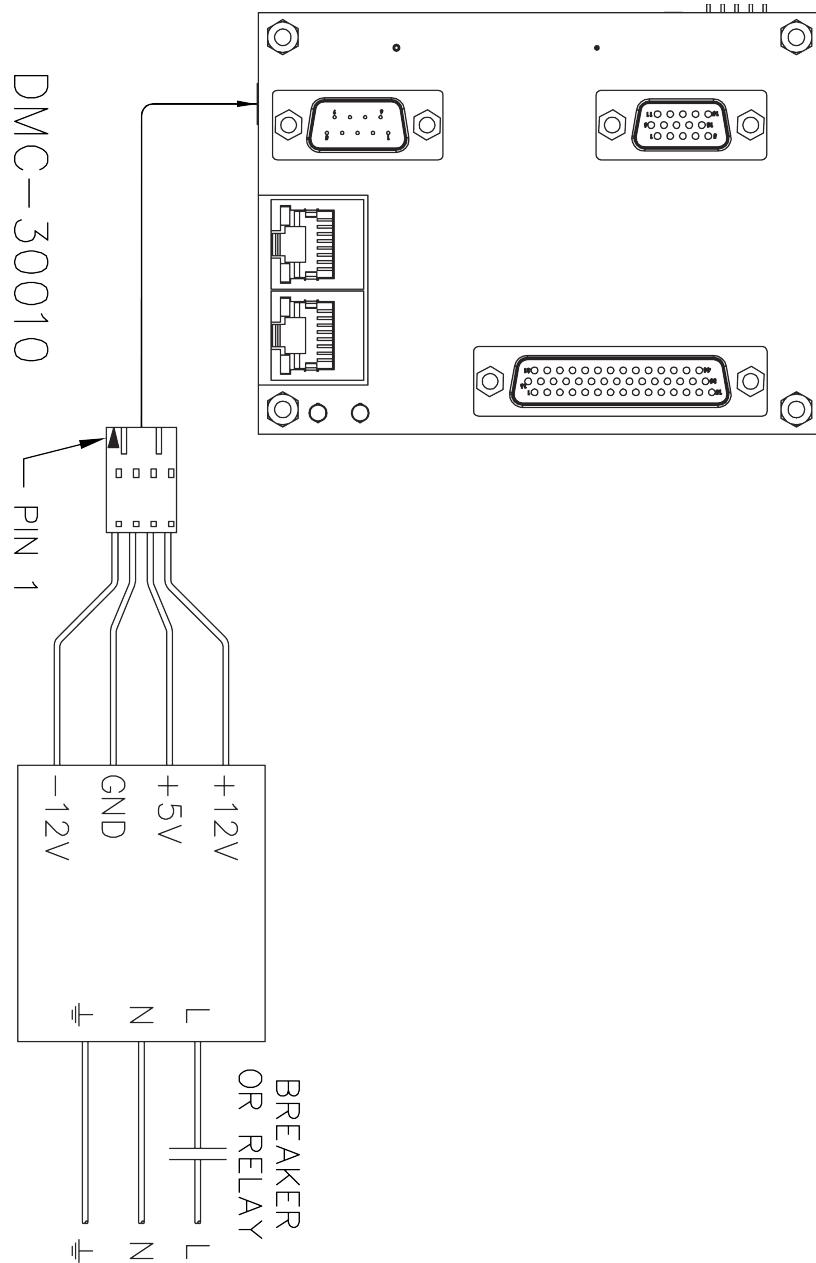
## Power Wiring Diagrams

The following diagrams shows how to power the different models of the DMC-30000 family. The connectors are keyed and indicate the correct orientation and pin numbers for the power input.

See Input Power Requirements for detailed Electrical Specifications.

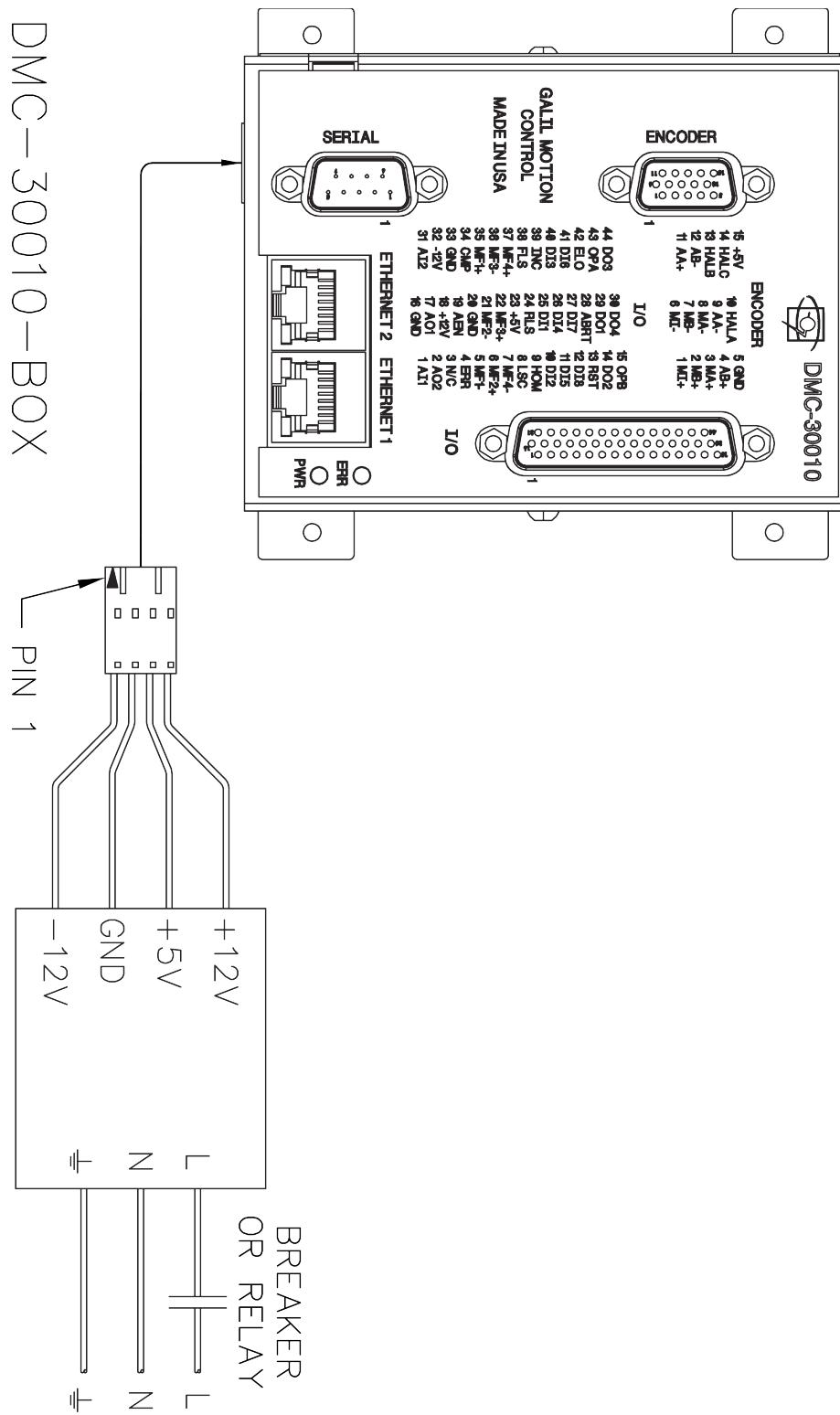
### DMC-30010-CARD

Requires a +5VDC and ±12VDC triple power supply.



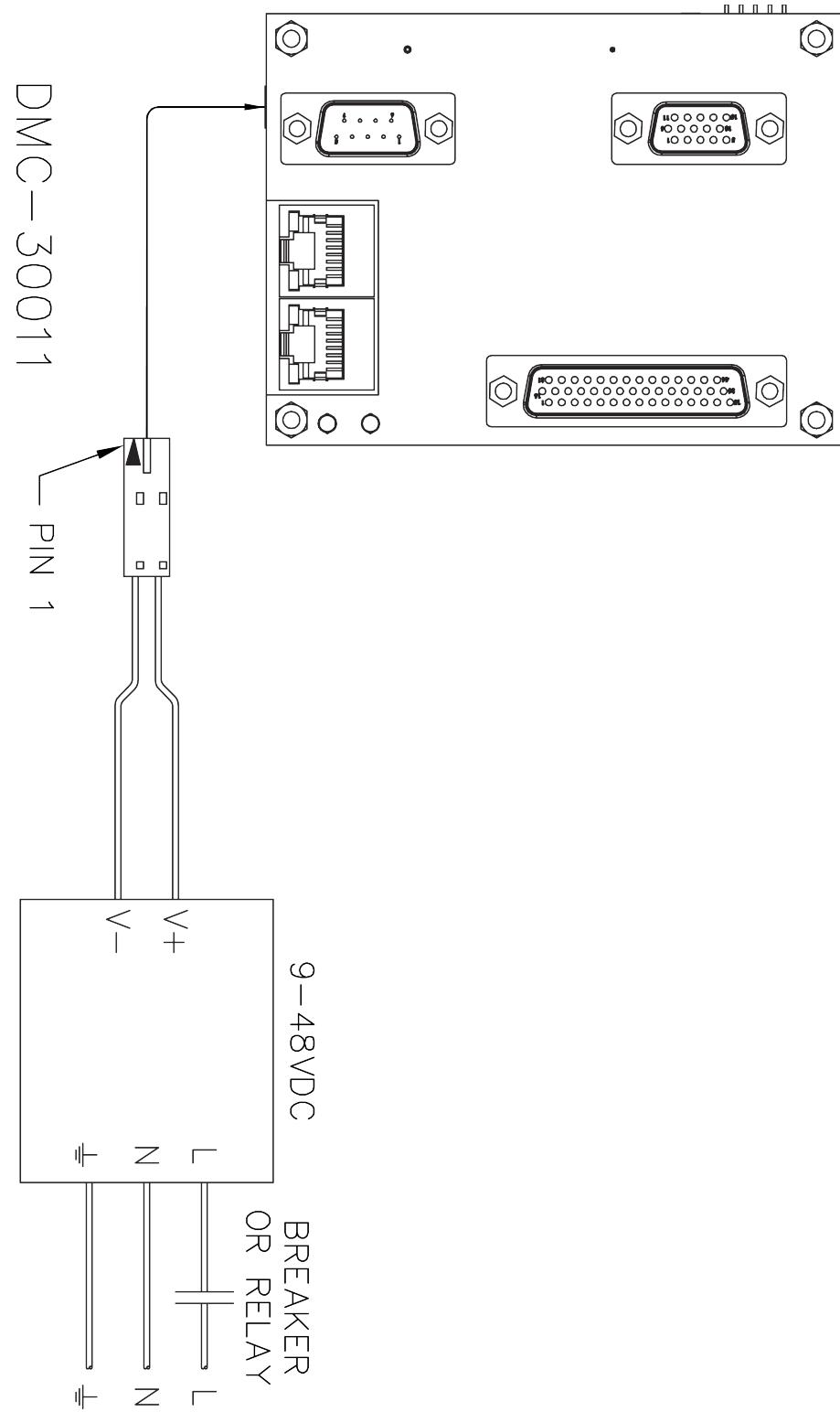
## DMC-30010-BOX

Requires a +5VDC and ±12VDC triple power supply.



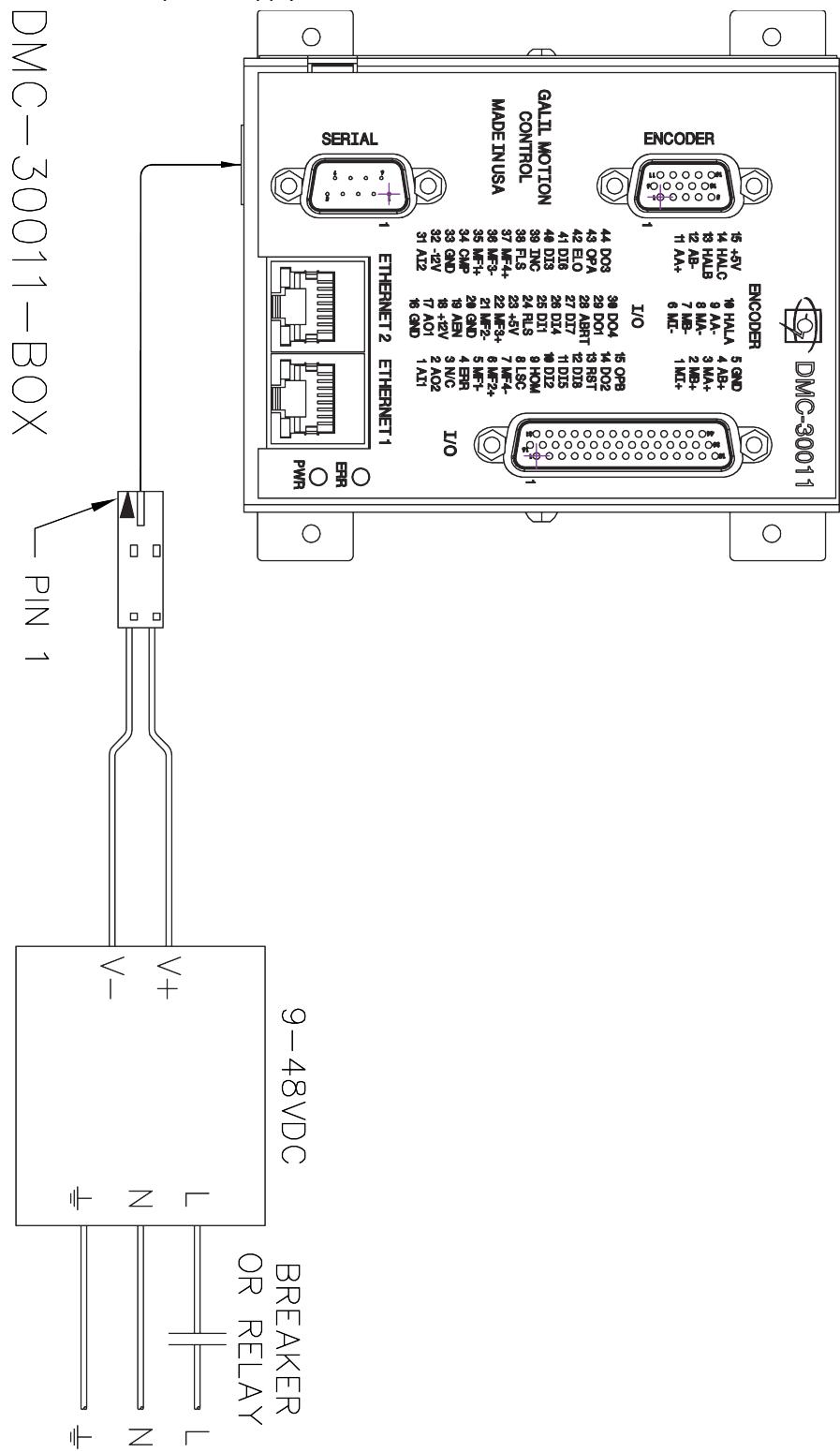
## DMC-30011-CARD

Requires a +9VDC to +48VDC power supply.



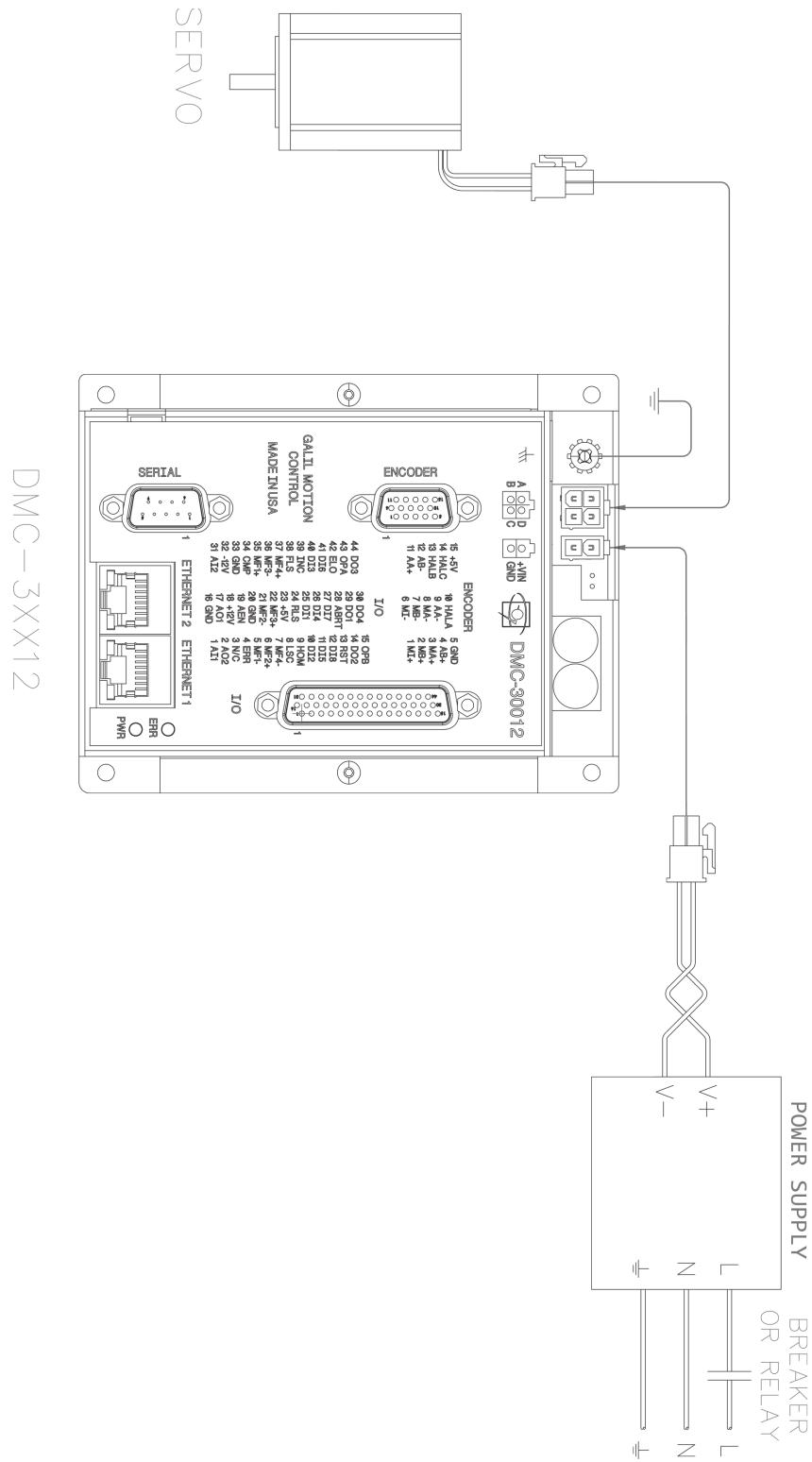
## DMC-30011-BOX

Requires a +9VDC to +48VDC power supply.



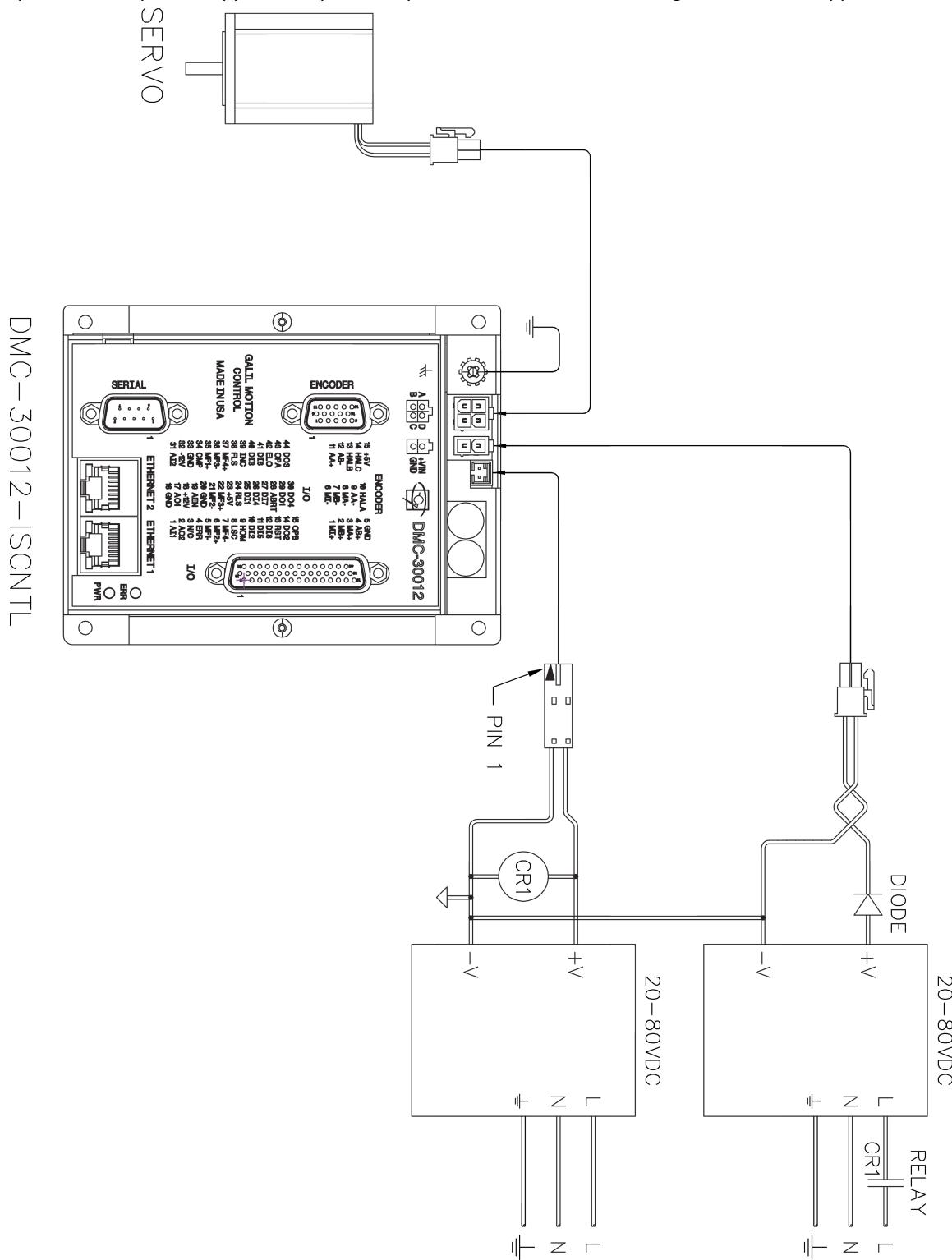
## DMC-30012-BOX, DMC-30014-BOX, DMC-30016-BOX and DMC-30017-BOX

See power requirements for individual configurations in the Appendices.



## DMC-30012-BOX(ISCNTL), DMC-30014-BOX(ISCNTL), DMC-30016-BOX(ISCNTL) and DMC-30017-BOX(ISCNTL)

Requires two DC power supplies, see power requirements for individual configurations in the Appendices.



## Cable Shielding

Electrical noise can negatively impact system performance by affecting different machine components. For example, it can cause the incorrect reporting of digital input states or position data. One simple and effective way to mitigate electrical noise is to add a shield around the encoder and motor phase cables. For proper shielding, the shield's drain wire should be terminated on only one side of the cable. The following diagrams illustrate the best means of achieving this with the DMC-30000, with the expectation that the controller's baseplate is mounted to something with a secure connection to Earth Ground.

Figure A.2 shows how to connect the encoder cable's shield drain wire. This side of the cable should then go to the controller's 15 pin encoder connector, making a connection to Earth Ground.

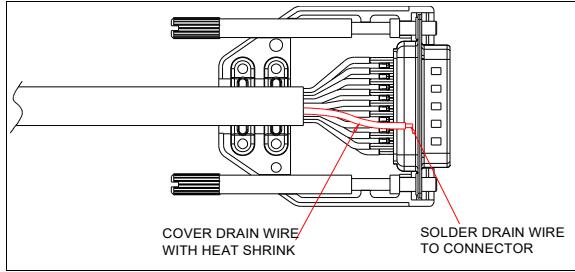


Figure A.2: Drain wire of the encoder cable shield

Figure A.3 shows how to connect the drain wire to the motor phase shield.

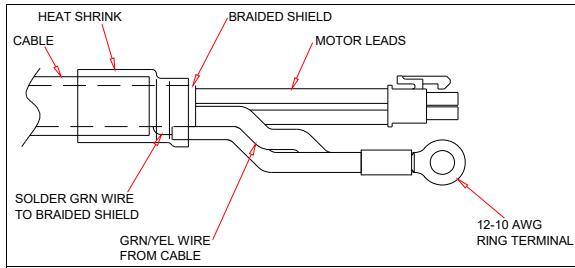


Figure A.3: Drain wire of the motor phase shield

The motor phase drain wire is fastened to Earth ground on the controller box, shown in Figure A.4.

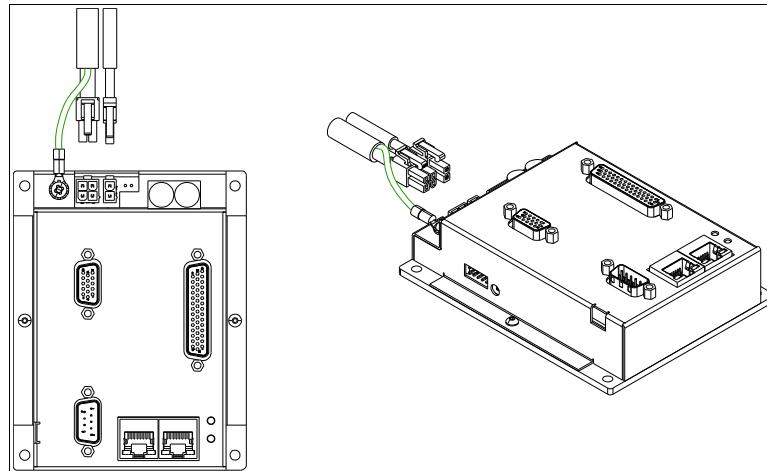


Figure A.4: Ring terminal of motor phase drain wire connected to box of DMC-30012

For more information on the theory behind electrical noise and its different sources, see Application Note #5438, <https://www.galil.com/download/application-note/note5438.pdf>

---

## Serial Cable Connections

The DMC-30000 is capable of Serial communication through the Serial port. The Serial port uses a DB-9 Male connector and can be used to interpret DMC commands. For more information on communicating using the Serial ports, refer to the Serial Communication Port section. For signal descriptions and cabling advice, please see below.

### Standard RS-232 Specifications

#### DB-9 Male Serial Connector (on Host PC)

Pin #	Signal Direction	Signal Name	Signal Function
1	Output	CD	Carrier Detected
2	Input	RxD	Receive Data
3	Output	TxD	Transmit Data
4	Output	DTR	Data Terminal Ready
5	-	GND	Ground
6	Input	DSR	Data Set Ready
7	Input	RTS	Request To Send
8	Output	CTS	Clear To Send
9	Output	RI	Ring Indicator

### DMC-30000 RS-232 Serial Port Specifications

#### DB-9 Male Serial Connector (Serial Port)

Pin #	Signal Direction	Signal Name	Signal Function
1	-	-	-
2	Output	TxD	Transmit Data
3	Input	RxD	Receive Data
4	-	-	-
5	-	GND	Ground
6	-	-	-
7	Output	CTS	Clear To Send
8	Input	RTS	Request To Send
9	-	-	-

### Serial Cable Specifications

#### USB-to-RS232 9-Pin Female Adapter (Serial Port)

To communicate with a Host PC's USB port using the Main Serial port of the DMC-30000, users must use a USB-to-RS232 adapter with a straight-through configuration. Typically, a USB-to-RS232 Female adapter will be in a null-modem configuration. To convert a null-modem configuration to a straight-through configuration the user must use an additional null-modem adapter with the USB-to-RS232 adapter. This will route the signals like that of the straight-through configuration. For more information, please contact Applications Support at Galil.

#### 9-Pin Female to 9-Pin Female Cable (Serial Port)

To communicate with the Main Serial port of the DMC-30000 using a 9-Pin Female to 9-Pin Female cable, users must have a cable with a straight-through configuration.

## DMC-30000 RS-422 Serial Port Specifications

### DB-9 Male Serial Connector (Main Serial Port)

Pin #	Signal Direction	Signal Name	Signal Function
1	Input	RTS-	Request To Send
2	Output	TXD-	Transmit Data
3	Input	RXD-	Receive Data
4	Output	CTS-	Clear To Send
5	-	GND	Ground
6	Input	RTS+	Request To Send
7	Output	TXD+	Transmit Data
8	Input	RXD+	Receive Data
9	Output	CTS+	Clear To Send

## Input Current Limitations

Each bank of inputs uses a current limiting resistor pack. These resistor packs have a maximum power rating that should never be exceeding during use. Over time, damage can occur to the resistor pack resulting in non-functional inputs. For most applications this is not an issue, but applications using greater than 18.5 V<sub>DC</sub> with normally closed switches should consider placing an additional current limiting resistor. Contact Galil if there is any uncertainty if your system requires the use of an additional resistor.

If a resistor is required, a resistor of 1.2kΩ should be added in series with each input to limit the amount of current. An example is shown with INCOM (Bank 0) to power digital input 1, in Figure A.5 below where R is 1.2 kΩ. This covers the “worst case scenario” where INCOM is powered with the maximum 24 V<sub>DC</sub> and all 8 of its inputs are normally closed. To calculate the exact resistance required the table below includes the maximum power rating per resistor pack and the maximum amount of inputs-pins used per pack.

	INCOM	LSCOM
Max. number of pins used on single resistor pack	8-inputs	6-inputs
Max. power rating per resistor pack	1.25 W	1.00 W

An example in the Input Current Limitation Calculations section below provides how to use this table to calculate the resistance required.

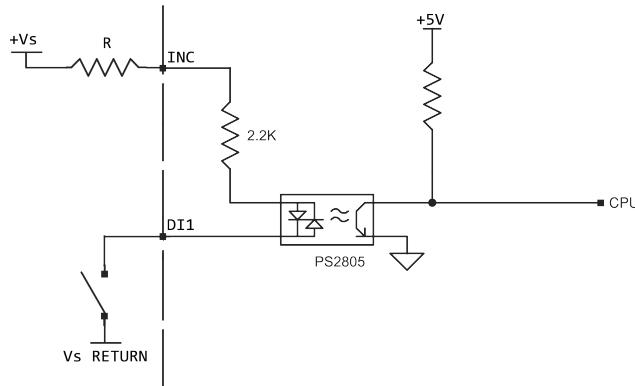


Figure A.5: Digital input wiring for +Vs to INCOM0 with additional current limiting resistor, R

## Input Current Limitation Calculations

The follow calculations are based upon the most extreme condition where all 8-inputs of INCOM are active at all times.

Assumptions:

- The resistor pack has a 1.25 W total limitation
- INCOM uses 8 resistors of a single resistor pack
- All inputs are continuously on at the same time
- The maximum voltage of 24 V<sub>DC</sub> is used

$$\frac{1.25 \text{ W}}{8 \text{ inputs}} = .156 \frac{\text{W}}{\text{input}}$$

Using  $P=IV$  and assuming that 1V is lost to the diode,

$$.156 \frac{W}{\text{input}} = I \times (24 V_{DC} - 1 V_{DC})$$

$$I = 6.8 \frac{mA}{\text{input}}$$

Using  $V=IR$  and assuming that 1V is lost to the diode,

$$(24 V_{DC} - 1 V_{DC}) = 6.8 \frac{mA}{\text{input}} \times R_{tot}$$

$$R_{tot} = 3382 \Omega$$

Since the inputs already have  $2200\Omega$  current limiting resistor internally,

$$R = 3382 \Omega - 2200 \Omega = 1182 \Omega$$

Which is most close to the standard resistor, **1.2kΩ**.

## Pin-outs

### I/O 44 pin HD D-Sub Connector (Female)

Pin #	Label	Description	Pin #	Label	Description	Pin #	Label	Description
1	AI1	Analog Input 1	16	AGND	Analog Ground	31	AI2	Analog Input 2
2	AO2	Analog Output 2	17	AO1	MCM/Analog Output 1 <sup>1</sup>	32	-12V	-12V
3	N/C	No Connect	18	+12V	+12V	33	GND	Ground
4	ERR	Error Output	19	AEN	Amplifier Enable	34	CMP	Output Compare
5	MF1 -	Multi-Function 1 -	20	GND	Ground	35	MF1 +	Multi-Function 1 +
6	MF2 +	Multi-Function 2 +	21	MF2 -	Multi-Function 2 -	36	MF3 -	Multi-Function 3 -
7	MF4 -	Multi-Function 4 -	22	MF3 +	Multi-Function 3 +	37	MF4 +	Multi-Function 4 +
8	LSC	Limit Switch Common	23	+5V	+5V	38	FLS	Forward Limit Switch
9	HOM	Home Switch Input	24	RLS	Reverse Limit	39	INC	Input Common
10	DI2	Digital Input 2	25	DI1	Digital Input 1	40	DI3	Digital Input 3
11	DI5	Digital Input 5	26	DI4	Digital Input 4	41	DI6	Digital Input 6
12	DI8	Digital Input 8	27	DI7	Digital Input 7	42	ELO	Electronic Lockout
13	RST	Reset Input	28	ABRT	Abort Input	43	OPA	Output GND/PWR (Bank 0)
14	DO2	Digital Output 2	29	DO1	Digital Output 1	44	DO3	Digital Output 3
15	OPB	Output PWR/GND (Bank 0)	30	DO4	Digital Output 4 <sup>2</sup>			

<sup>1</sup>AO1, Analog Output 1 is used as the motor command output for the DMC-3xx10 and DMC-3xx11.

<sup>2</sup> Can be used as an alternative amplifier enable line, see Amplifier Enable, pg 38

### Multi-Functional Pins (MF<sub>n</sub>±)

The Multi-Functional Pins on the DMC-30000 have different functionality depending upon how the controller was ordered and how the controller is setup by the user. If the controller is ordered with -SER (serial encoder interface), then the MF pins can be used to interface to a serial encoder. MF1 and MF2 are only used for the Main serial encoder input, MF3 and MF4 are used for the Aux serial encoder input. See the SI and SS commands in the Command Reference for more detail.

When the controller is setup for stepper motor operation, the MF 2 and MF4 pins are used for step and direction respectively.

Single Description for Multi-Functional Pins			
Label	Pin #	MT ±2 or ±2.5	-SER option with SSI or BiSS Enabled
MF1 +	35	No Connect	Main Encoder Data +
MF1 -	5	No Connect	Main Encoder Data -
MF2 +	6	STEP +	Main Encoder Clock +
MF2 -	21	STEP -	Main Encoder Clock -
MF3 +	22	No Connect	Aux Encoder Data +
MF3 -	36	No Connect	Aux Encoder Data -
MF4 +	37	DIR +	Aux Encoder Clock +
MF4 -	7	DIR -	Aux Encoder Clock -

**NOTE:** The -SER option cannot be used when controlling an external drive with Step/Dir. For more information, contact Support at Galil.

### RJ45 Connector - Ethernet

The Ethernet connection is Auto MDIX, 100bT/10bT.

Pin #	Signal
1	TXP
2	TXN
3	RXP
4	NC
5	NC
6	RXN
7	NC
8	NC

## Encoder 15 pin HD D-Sub Connector (Female)

Pin #	Label	Description
1	MI+	I+ Index Pulse Input
2	MB+	B+ Main Encoder Input
3	MA+	A+ Main Encoder Input
4	AB+	B+ Aux Encoder Input
5	GND	Digital Ground
6	MI-	I- Index Pulse Input
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

## DB-9 Male Serial Connector (Serial Port)

Pin #	Signal Direction	Signal Name	Signal Function
1	-	-	-
2	Output	TxD	Transmit Data
3	Input	RxD	Receive Data
4	-	-	-
5	-	GND	Ground
6	-	-	-
7	Output	CTS	Clear To Send
8	Input	RTS	Request To Send
9	-	-	-

## Jumper Description for DMC-30000

Label	Function (If jumpered)
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.
19.2	Baud Rate setting – see table below
UG	Used to upgrade controller firmware when resident firmware is corrupt.
MR	Master Reset enable. Returns controller to factory default settings and erases FLASH. Requires power-on or RESET to be activated.

### Baud Rate Jumper Settings

19.2	BAUD RATE
ON	19200
OFF	115200 (Recommended)

# Signal Descriptions

## Outputs

Analog Outputs 1,2 / Motor Command	± 10 Volt range signal for driving amplifier or for a general purpose analog output. 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.
Step Output, MF2	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%. For sign/magnitude mode, please contact a Galil Applications Engineer.
Direction, MF4	Provides the direction signal for step motors. For sign/magnitude mode, please contact a Galil Applications Engineer.
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 4	The optically isolated outputs are uncommitted and may be designated by the user to 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 15,000,000 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	Controllers with Internal Amplifiers Only. 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	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 A 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 A.

---

## 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)

[www.galil.com/learn/classes](http://www.galil.com/learn/classes)

---

## Contacting Us

**Galil Motion Control**

270 Technology Way

Rocklin, CA 95765

Phone: 916-626-0101

Fax: 916-626-0102

E-Mail Address: support@galil.com

Web: <http://www.galil.com/>

---

## 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.

**COPYRIGHT (3-97)**

The software code contained in this Galil product is protected by copyright and must not be reproduced or disassembled in any form without prior written consent of Galil Motion Control, Inc.

# A1 – DMC-30012

---

## Description

The DMC-30012 includes a sinusoidally commutated, PWM amplifier for driving brushless servo motors or a brushed motor. Each amplifier drives motors operating at up to 10 Amps continuous, 15 Amps peak, 20–80 VDC. The gain settings of the amplifier are user-programmable at 0.4 Amp/Volt, 0.8 Amp/Volt and 1.6 Amp/Volt. The switching frequency is 33 kHz. The amplifier offers protection for over-voltage, under-voltage, over-current, short-circuit and over-temperature. A shunt regulator option is available. If higher voltages are required, please contact Galil.

If the application has a potential for regenerative energy it is recommended to order the controller with the ISCNTL – Isolate Controller Power option and the SR90 – 90V Shunt Regulator Option.

[GDK](#)'s Step-By-Step tool provides an interactive guide for configuring brushless and brushed motors.

**WARNING**

Do not “hot swap” the motor power or supply voltage power input connections. If the amplifier is enabled when the motor connector is connected or disconnected, damage to the amplifier can occur. Galil recommends powering the controller and amplifier down before changing the connector, and breaking the AC side of the power supply connection in order to power down the amplifier.

# Electrical Specifications

	Servo (Brushless/Brush)
<b>Supply Voltage</b>	20-80 VDC
<b>Continuous Current</b>	10 Amps
<b>Peak Current</b>	15 Amps
<b>Switching Frequency</b>	33 kHz
<b>Amplifier Gains</b>	0.4, 0.8, 1.6 Amps/Volt
<b>Minimum Load Inductance</b>	$L(mH) = \frac{Vs(V)}{264 * I_{Ripple}(A)}$

$V_s$  = Supply Voltage,  $I_{\text{ripple}} = 10\%$  of the maximum current at chosen gain

Table A1.1: Amplifier Electrical Specifications

## Mating Connectors

	Cable Connectors	Terminal Pins	On board Connector
<b>POWER</b>	2-pin Mini-Fit, Jr. <sup>TM</sup> (Male) Molex #0039012025	Molex #0444761112	2-pin Mini-Fit, Jr. <sup>TM</sup> (Female) Molex #0039310020
<b>A,B,C,D: 4-pin Motor Power Connectors</b>	4-pin Mini-Fit, Jr. <sup>TM</sup> (Male) Molex #0039012045	Molex #0444761112	4-pin Mini-Fit, Jr. <sup>TM</sup> (Female) Molex #0039310040

Table A1.2: Molex part numbers for connectors and terminal pin

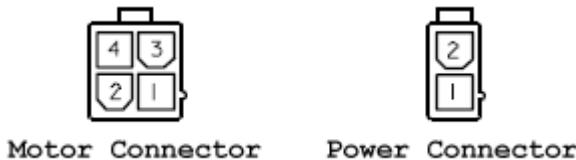


Figure A1.1: On Board Power and Motor Connector Diagram

Power Connector		
Pin Number	Connection	
1	DC Power Supply Ground	
2	+VS (DC Power)	
Motor Connector		
Pin Number	Brushless	Brushed
1	Phase C	Phase A-
2	Phase B	No Connect
3	No Connect	No Connect
4	Phase A	Phase A+

Table A1.3: Power and Motor Connector Pin-outs

---

# Servo Motor Operation

## Brushless Motor Operation

For Brushless motor operation, **MT** must be set to **1** for the axis. The 6 commands used for setup are the **BA**, **BM**, **BX**, **BZ**, **BC** and **BI**. Please refer to the Command Reference for details. Below is the setup procedure for configuring an axis to operate a Brushless motor.

1. Issue the **BA** command to specify on which axis to configure sinusoidal commutation.
2. The brushless modulus is the number of encoder counts per magnetic cycle and is set with the **BM** command. For example, in a rotary motor that has 2 pole pairs and 10000 counts per revolution, the number of encoder counts per magnetic cycle would be 10000/2. **BM** should be assigned to a value of **5000**.
3. Issue either the **BI** and **BC**, **BZ**, or **BX** command to initialize the motor for sinusoidal commutation. The command must be executed on every reset or power-up of the controller.

### Setup Commands

1. **BI/BC** Commands – If the motor includes hall sensors, Galil recommends using the **BI** and **BC** commands. The **BI** command uses the hall sensors to set the approximate location of the motor in the magnetic cycle. The **BC** command will precisely set the commutation angle upon detecting a hall state transition.
2. **BZ** Command – If the motor does not include hall sensors, Galil recommends using the **BZ** command. The **BZ** command forces the motor to a known location in the magnetic cycle. Note that this can cause up to  $\frac{1}{2}$  a magnetic cycle of motion.
3. **BX** Command – If the motor does not have hall sensors and the **BZ** command generates too much motion for the application, Galil recommends using the **BX** command. This command utilizes a minimal movement algorithm in order to determine the correct commutation of the motor.

## Brushed Motor Operation

For brushed motor operation, set **MT** and **BR** to **1** for the axis.

## Using External Amplifiers

Use D-Sub connectors on the top face of the controller to access necessary signals to run external amplifiers. For more information on connecting external amplifiers, see Step 2b. Connecting External Amplifiers and Motors in Chapter 2.

## Setting Amplifier Gain and Current Loop Gain

The [AG](#) command will set the amplifier gain (Amps/Volt). The [AU](#) command will set the current loop gain.

### AG Command:

The DMC-30012 has 3 amplifier gain settings. The gain is set with the [AG](#) command. When configuring the amplifier gain, the axis must be in a motor off ([MO](#)) state prior to execution of the [AG](#) command. For more information on the [AG](#) command, refer to the command reference for the DMC-3x01x.

AG setting	Gain Value
n = 0	0.4 A/V
n = 1	0.8 A/V
n = 2	1.6 A/V

*Table A1.4: Amplifier Gain Settings*

### AU Command:

Proper configuration of the [AU](#) command is essential to optimize the operation of the DMC-30012. This command sets the gain for the current loop on the amplifier. Table A1.5 indicates the recommended [AU](#) settings for 24 and 48 VDC power supplies based on motor inductance. When configuring the current loop gain, the axis must be in a motor off ([MO](#)) state prior to execution of the [AU](#) command. For more information on the [AU](#) command, refer to the command reference for the DMC-3x01x.

Vsupply VDC	Inductance L (mH)	n
24	L < 1	9
24	1 ≤ L < 2.3	10
24	2.3 ≤ L < 4.2	11
24	4.2 ≤ L	12
48	L < 2.4	9
48	2.4 ≤ L < 4.2	10
48	4.2 ≤ L < 7	11
48	7 ≤ L	12

*Table A1.5: Amplifier Current Loop Gain Settings*

## Setting Peak and Continuous Torque Limits

### TK and TL Commands:

The peak and continuous torque limits can be set through the **TK** and **TL** commands respectively. The controller will command the torque signal (**TT**) up to the value specified by **TK** for a maximum of one second, as shown in Figure A1.2. It will then limit the torque to the value specified by **TL**. The more time the controller commands the torque below **TL**, the longer it will command it to **TK** when necessary. The DMC code example below exhibits the behavior of **TT** being limited by **TL** and **TK**, as shown in Table A1.2.

```
#tk;          'begin program
MO;          'disable all axes
AGA=2;        'set max amplifier gain of 1.6 A/V
TLA=5;        'set continuous torque limit of 5V
TKA=9.9982;   'set max peak torque limit to 9.9982V
SHA;          'enable A axis
OFA=9.9982;   'command torque signal to 9.9982V
EN;          'end program
```

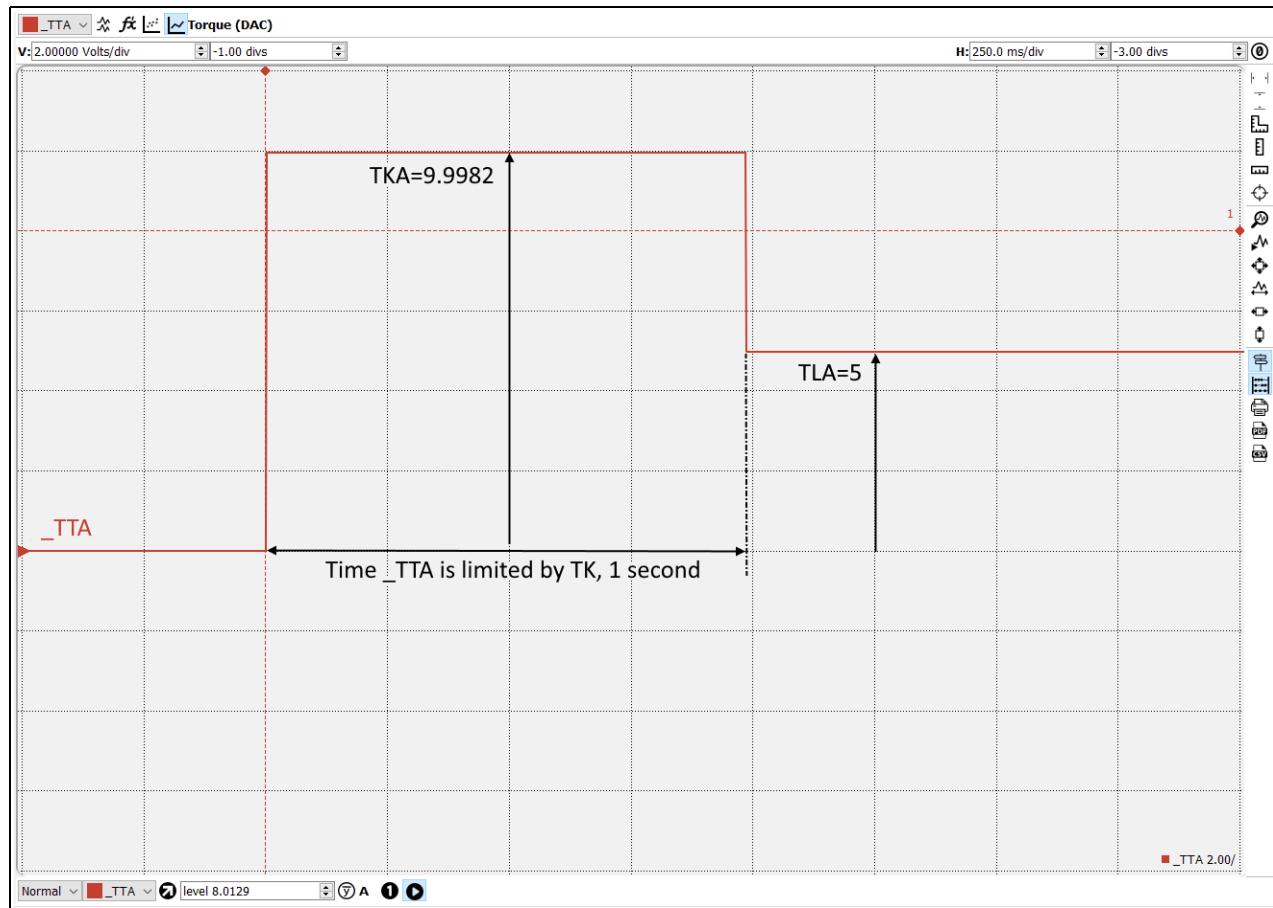


Figure A1.2: Peak Current Operation

# Error Monitoring and Protection

This amplifier is protected against Over-Current, Over-Voltage, Over-Temperature, and Under-Voltage conditions. The amplifier errors can be monitored using the [TA](#) command. This command will return a bit mask number representing specific error conditions. It will also report whether or not an amplifier error is latched. For more information, refer to the [TA](#) command in the Command Reference.

To handle amplifier error conditions, users can include the [#AMPERR](#) automatic subroutine in their DMC program. This subroutine is user defined. Two conditions must be met for the [#AMPERR](#) subroutine to run: the [#AMPERR](#) subroutine must be defined in the DMC program; and an amplifier error must be detected. When these conditions are met, the program will automatically interrupt thread 0 and jump to the [#AMPERR](#) subroutine.

## Over-Current Protection

If there is a short circuit in the amplifier or motor, an Over-Current error will be reported and 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. The amplifier can be re-enabled with the [SH](#) command when there is no longer an over-current condition.

<b>WARNING</b>	If this error occurs, it is indicative of a problem at the system level. An Over-Current error is usually due to a short across the motor leads or a short from a motor lead to ground.
----------------	---

## Over-Voltage Protection

If the voltage supplied to the amplifier rises above its rated supply voltage, an Over-Voltage error will be reported. While the amplifier is experiencing this condition, current is no longer commanded and the motor phases are shorted together. This results in a drag on the motors causing them to coast to a stop. The amplifier will resume commanding current when the voltage drops below the maximum rated supply voltage.

The over voltage condition will not permanently shut down the amplifier and will not trigger the [#AMPERR](#) routine.

<b>WARNING</b>	If this error occurs, it is indicative of a problem at the system level. An Over-Voltage error is usually due to regeneration from the motor and load. Consider using a shunt regulator if this occurs during normal operation.
----------------	---

## Over-Temperature Protection

If the average amplifier temperature rises above 80°C, an Over-Temperature error will be reported and the amplifier will be disabled. 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).

<b>WARNING</b>	An Over-Temperature error is usually due to insufficient heat dissipation from the amplifier. Heat must be drawn away from the base of the controller with forced air flow or a heat sink.
----------------	--

## **Under-Voltage Protection**

If the voltage supplied to the amplifier drops below its rated supply voltage, an Under-Voltage error will be reported and the amplifier will be disabled.

If there is an **#AMPERR** routine and the controller is powered before the amplifier, then the **#AMPERR** routine will automatically be triggered. The amplifier will return to normal operation once the supply is raised above the rated supply voltage assuming it did not cause the position error to exceed the error limit.

## **Peak Current**

If the commanded torque value (**TT**) on any axis equals the peak torque limit set by **TK**, a Peak Current condition will be reported for that axis, and the axis bit in the **TA2** bit mask will be set. Peak Current does not cause **#AMPERR** to run. The Peak Current condition reports live and is not latching.

## **ELO Input**

If the ELO input on the controller is asserted, the power stage of the amplifier will be disabled. When the ELO input is triggered, **TA3** will change state and the **#AMPERR** routine will run on Thread 0. To recover from an ELO, the user must issue the **MO** command followed by the **SH** command. To recover from an ELO in a DMC program, use **MO;WT2;SH**.

It is recommended that **OE1** be used for all axes when the ELO is used in an application.

See the ELO (Electronic Lock-Out) Input section in Chapter 3 Connecting Hardware for information on connecting the ELO input.

# A2 – DMC-30014

---

## Description

The DMC-30014 contains a trans-conductance linear drive for sinusoidally commutating brushless motors. The DMC-30014 requires a single 15–40 VDC input, and outputs a typical power of 20 W. The amplifier gain is locked to 0.2 A/V. Typically a 24VDC supply will deliver 1A continuous and 2A peak. The current loop bandwidth is approximately 4 kHz.

[GDK](#)'s Step-By-Step tool provides an interactive guide for configuring brushless motors.

<b>WARNING</b>	Do not “hot swap” the motor power or supply voltage power input connections. If the amplifier is enabled when the motor connector is connected or disconnected, damage to the amplifier can occur. Galil recommends powering the controller and amplifier down before changing the connector, and breaking the AC side of the power supply connection in order to power down the amplifier.
----------------	---

## Electrical Specifications

	Servo (Brushless)
Supply Voltage	15-40 VDC
Continuous Current	1.0 Amps
Peak Current	2.0 Amps
Total Max. Power Output	20 W

Table A2.1: Amplifier Electrical Specifications

## Mating Connectors

	Cable Connectors	Terminal Pins	On board Connector
POWER	2-pin Mini-Fit, Jr. <sup>TM</sup> (Male) Molex #0039012025	Molex #0444761112	2-pin Mini-Fit, Jr. <sup>TM</sup> (Female) Molex #0039310020
A,B,C,D: 4-pin Motor Power Connectors	4-pin Mini-Fit, Jr. <sup>TM</sup> (Male) Molex #0039012045	Molex #0444761112	4-pin Mini-Fit, Jr. <sup>TM</sup> (Female) Molex #0039310040

Table A2.2: Molex part numbers for connectors and terminal pin

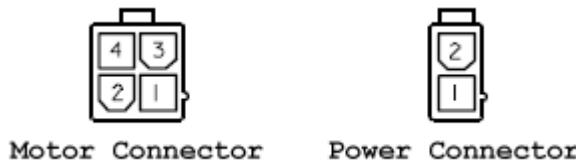


Figure A2.1: On Board Power and Motor Connector Diagram

Power Connector	
Pin Number	Connection
1	DC Power Supply Ground
2	+VS (DC Power)
Motor Connector	
1	Phase C
2	Phase B(N/C for Brushed Motors)
3	No Connect
4	Phase A

Table A2.3: Power and Motor Connector Pin-outs

## Power

Unlike a switching amplifier a linear amplifier does not have a straightforward relationship between the power delivered to the motor and the power lost in the amplifier. Therefore, determining the available power to the motor is dependent on the supply voltage, the characteristics of the load motor, and the required velocity and current.

All of the power delivered by the power supply is either used in the motor or lost in the amplifier.

$$\text{Power of Power Supply } P_{ps} = P_m + P_A$$

The power to the motor is both the power used to provide motion and the power lost to heat.

$$\text{Power of the motor } P_m = \text{Work} + \text{Power Lost in Motor} \quad P_m = K_e * \text{Velocity} * i + i^2 R_m$$

$$\text{Power of amplifier } P_A = (V_s - i * R_m - K_e * \text{Velocity}) * i$$

In addition there is a minimum power dissipated by the amplifier when powered regardless of load. The minimum power that the amplifier will consume is roughly

$$P_{A,\min} \approx \text{drop across op amp power stages} + \text{drop across sense resistor} + \text{op amp supply}$$

$$P_{A,\min} \approx 4 * i + i^2 * .5 + N$$

Where N = 1.5W for 24V and N = 3W for 48V

For example: assume a 24VDC supply and a motor with  $R_m = 4\text{ohms}$  and  $K_e = 5V / RPM$  and desired output currents of 1 and .5 amps.

First calculate the minimum power used in the amplifier.

$$P_{A,\min}(1amp) \approx 4 * i + i^2 * .5 + 1.5 = 6W$$

$$P_{A,\min}(.5amp) \approx 4 * .5 + .5^2 * .5 + 3 = 5.125W$$

The power used by the motor will vary by its velocity even though the power lost in the motor is a constant for each value of current. The more power sent to the motor, the less power will be dissipated by the amplifier as heat.

---

# Operation

## Brushless Motor Operation

For Brushless motor operation, **MT** must be set to **1** for the axis. The 6 commands used for setup are the **BA**, **BM**, **BX**, **BZ**, **BC** and **BI**. Please refer to the Command Reference for details. Below is the setup procedure for configuring an axis to operate a Brushless motor.

1. Issue the **BA** command to specify on which axis to configure sinusoidal commutation.
2. The brushless modulus is the number of encoder counts per magnetic cycle and is set with the **BM** command. For example, in a rotary motor that has 2 pole pairs and 10000 counts per revolution, the number of encoder counts per magnetic cycle would be 10000/2. **BM** should be assigned to a value of **5000**.
3. Issue either the **BI** and **BC**, **BZ**, or **BX** command to initialize the motor for sinusoidal commutation. The command must be executed on every reset or power-up of the controller.

### Setup Commands

1. **BI/BC** Commands – If the motor includes hall sensors, Galil recommends using the **BI** and **BC** commands. The **BI** command uses the hall sensors to set the approximate location of the motor in the magnetic cycle. The **BC** command will precisely set the commutation angle upon detecting a hall state transition.
2. **BZ** Command – If the motor does not include hall sensors, Galil recommends using the **BZ** command. The **BZ** command forces the motor to a known location in the magnetic cycle. Note that this can cause up to  $\frac{1}{2}$  a magnetic cycle of motion.
3. **BX** Command – If the motor does not have hall sensors and the **BZ** command generates too much motion for the application, Galil recommends using the **BX** command. This command utilizes a minimal movement algorithm in order to determine the correct commutation of the motor.

## Brushed Motor Operation

The DMC-30014 must be configured for brushed motor operation at the factory. Contact Galil prior to placing the order. Once the amplifier is configured for a brushed motor, the controller needs to be set for brushed mode by setting the **BR** command to a value of **1**. The A and C motor phases are used for connecting to the brushed motor (B phase is a no connect).

## Using External Amplifiers

Use D-Sub connectors on the top face of the controller to access necessary signals to run external amplifiers. For more information on connecting external amplifiers, see Step 2b. Connecting External Amplifiers and Motors in Chapter 2.

## Setting Peak and Continuous Torque Limits

### TK and TL Commands:

The peak and continuous torque limits can be set through the **TK** and **TL** commands respectively. The controller will command the torque signal (**TT**) up to the value specified by **TK** for a maximum of one second, as shown in [Figure A2.2](#). It will then limit the torque to the value specified by **TL**. The more time the controller commands the torque below **TL**, the longer it will command it to **TK** when necessary. The DMC code example below exhibits the behavior of **TT** being limited by **TL** and **TK**, as shown in [Figure A2.2](#).

```
#tk;          'begin program
MO;          'disable all axes
TLA=5;        'set continuous torque limit of 5V
TKA=9.9982;   'set max peak torque limit to 9.9982V
SHA;          'enable A axis
OFA=9.9982;   'command torque signal to 9.9982V
EN;          'end program
```

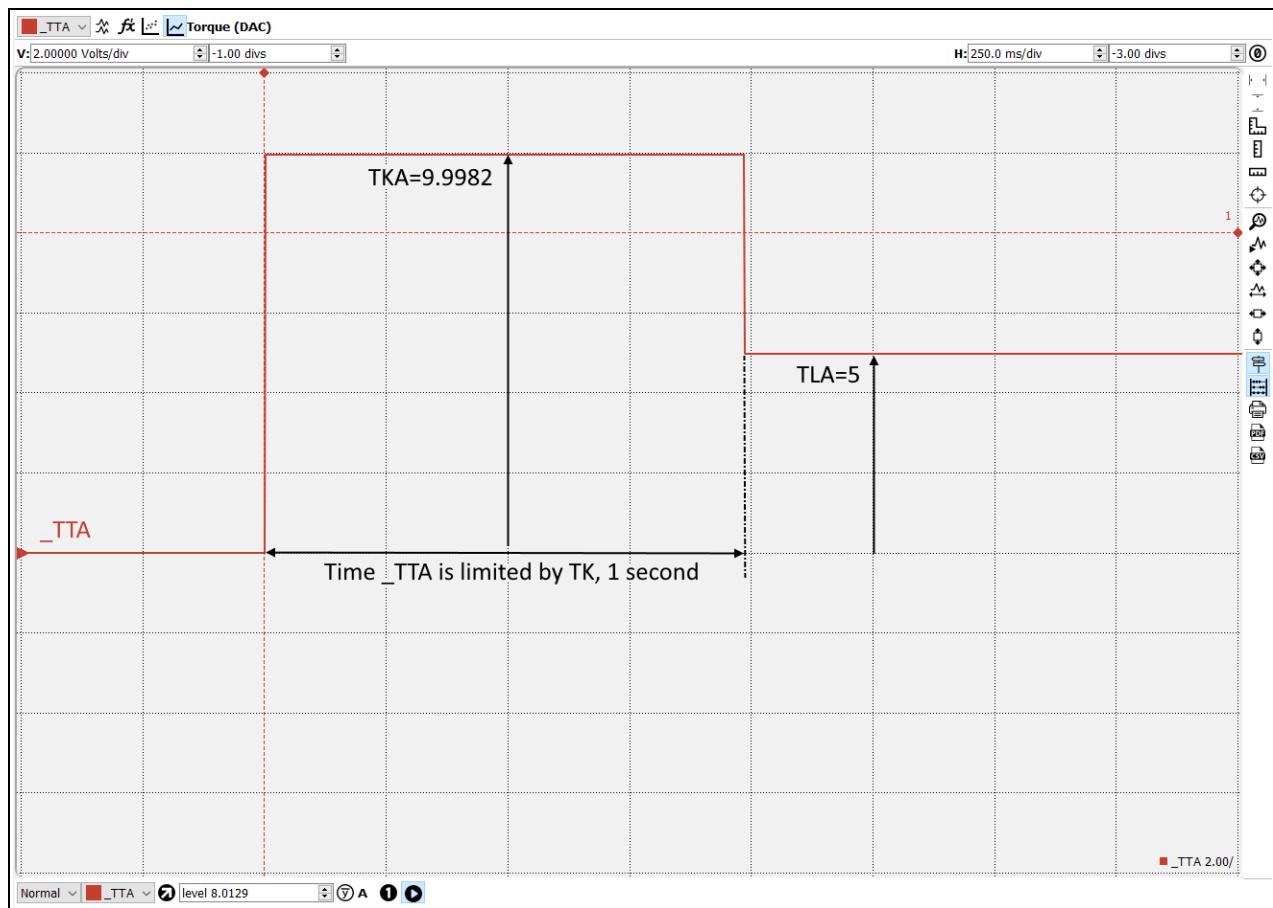


Figure A2.2: Peak Current Operation

# Error Monitoring and Protection

This amplifier is protected against Over-Temperature conditions. The amplifier errors can be monitored using the [TA](#) command. This command will return a bit mask number representing specific error conditions. For more information, refer to the [TA](#) command in the Command Reference.

To handle amplifier error conditions, users can include the [#AMPERR](#) automatic subroutine in their DMC program. This subroutine is user defined. Two conditions must be met for the [#AMPERR](#) subroutine to run: the [#AMPERR](#) subroutine must be defined in the DMC program; and an amplifier error must be detected. When these conditions are met, the program will automatically interrupt thread 0 and jump to the [#AMPERR](#) subroutine.

## Over-Temperature Protection

The amplifier is also equipped with over-temperature protection.

If the average heat sink temperature rises above 80°C, then the amplifier will be disabled. The over-temperature condition will trigger the [#AMPERR](#) routine if included in the program on the controller.

The amplifier can be re-enabled with the [SH](#) command once the temperature drops below 80°C.

**WARNING**

An Over-Temperature error is usually due to insufficient heat dissipation from the amplifier. Heat must be drawn away from the base of the controller with forced air flow or a heat sink.

## Peak Current

If the commanded torque value ([TT](#)) on any axis equals the peak torque limit set by [TK](#), a Peak Current condition will be reported for that axis, and the axis bit in the [TA2](#) bit mask will be set. Peak Current does not cause [#AMPERR](#) to run. The Peak Current condition reports live and is not latching.

## ELO Input

If the ELO input on the controller is asserted, the power stage of the amplifier will be disabled. When the ELO input is triggered, [TA3](#) will change state and the [#AMPERR](#) routine will run on Thread 0. To recover from an ELO, the user must issue the [MO](#) command followed by the [SH](#) command. To recover from an ELO in a DMC program, use [MO;WT2;SH](#).

It is recommended that [OE1](#) be used for all axes when the ELO is used in an application.

See the ELO (Electronic Lock-Out) Input section in Chapter 3 Connecting Hardware for information on connecting the ELO input.

# A3 – DMC-30016

---

## Description

The DMC-30016 contains a drive for operating two-phase bipolar step motors. The DMC-30016 requires a single 12-30 VDC input. The unit is user-configurable for 0.5 to 1.4 Amps per phase and for full-step, half-step, 1/4 step or 1/16 step.

[GDK](#)'s Step-By-Step tool provides an interactive guide for configuring stepper motors.

**WARNING**

Do not “hot swap” the motor power or supply voltage power input connections. If the amplifier is enabled when the motor connector is connected or disconnected, damage to the amplifier can occur. Galil recommends powering the controller and amplifier down before changing the connector, and breaking the AC side of the power supply connection in order to power down the amplifier.

## Electrical Specifications

	Stepper
Supply Voltage	12-30 VDC
Max Current (per axis)	1.4 Amps/Phase (User-configurable)
Max Step Frequency	3 MHz
Motor Type	Bipolar 2-Phase

Table A3.1: Amplifier Electrical Specifications

## Mating Connectors

	Cable Connectors	Terminal Pins	On board Connector
POWER	2-pin Mini-Fit, Jr. <sup>TM</sup> (Male) Molex #0039012025	Molex #0444761112	2-pin Mini-Fit, Jr. <sup>TM</sup> (Female) Molex #0039310020
A,B,C,D: 4-pin Motor Power Connectors	4-pin Mini-Fit, Jr. <sup>TM</sup> (Male) Molex #0039012045	Molex #0444761112	4-pin Mini-Fit, Jr. <sup>TM</sup> (Female) Molex #0039310040

Table A3.2: Molex part numbers for connectors and terminal pin

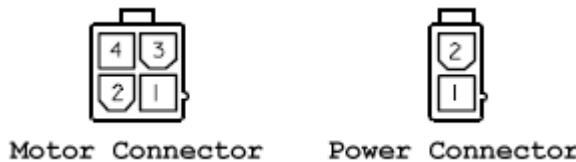


Figure A3.1: On Board Power and Motor Connector Diagram

Power Connector	
Pin Number	Connection
1	DC Power Supply Ground
2	+VS (DC Power)
Motor Connector	
1	B-
2	B+
3	A-
4	A+

Table A3.3: Power and Motor Connector Pin-outs

---

# Stepper Motor Operation

## Stepper Mode

The DMC-30016 should be configured for active high step pulses via the [MT](#) command. For more information on the [MT](#) command, check the command reference of the DMC-30016.

## Current Level Setup (AG Command)

The DMC-30016 has a User-configurable current gain setting, set by the [AG](#) command. When configuring the amplifier gain, the axis must be in a motor off ([MO](#)) state prior to execution of the [AG](#) command. For more information on the [AG](#) command, refer to the command reference for the DMC-30016.

## Low Current Setting (LC Command)

By default, 100% of the current set by the [AG](#) command will be delivered to the motor while the axis is holding position. An axis can be configured for Low Current mode so 25% of the current will be sent to the motor while holding position. Low Current mode can also be configured to wait a specified number of samples after a move is completed before lowering the current to 0%. For more information, refer to the [LC](#) command in the command reference for the DMC-30016.

LC setting	Effect on Stepper Phase Current
$n > 1$	$n$ samples after move finishes before going to 0% holding current
$n = 1$	25% holding current immediately after move finishes
$n = 0$	100% holding current
$n < 0$	$n$ samples after move finishes before going to 25% holding current

*Table A8.4: Low Current Mode configurations*

## Step Drive Resolution Setting (YA command)

When using the DMC-30016, the step drive resolution can be set with the [YA](#) command. Note, when running in full step mode – the current to the motor is 70% of maximum. All microstep settings are able to deliver full current.

AG setting	Gain Value
$n = 1$	Full
$n = 2$	Half
$n = 4$	1/4
$n = 16$	1/16

*Table A8.5: Amplifier Gain Settings*

## Using External Amplifiers

Use D-Sub connectors on the top face of the controller to access necessary signals to run external amplifiers. For more information on connecting external amplifiers, see Step 2b. Connecting External Amplifiers and Motors in Chapter 2.

# Error Monitoring and Protection

This amplifier is protected against Over-Current conditions. This amplifier error can be monitored using the [TA](#) command. This command will return a bit mask number representing specific error conditions. For more information, refer to the [TA](#) command in the Command Reference.

To handle amplifier error conditions, users can include the [#AMPERR](#) automatic subroutine in their DMC program. This subroutine is user defined. Two conditions must be met for the [#AMPERR](#) subroutine to run: the [#AMPERR](#) subroutine must be defined in the DMC program; and an amplifier error must be detected. When these conditions are met, the program will automatically interrupt thread 0 and jump to the [#AMPERR](#) subroutine.

## Over-Current Protection

If there is a short circuit in the amplifier or motor, an Over-Current error will be reported and the amplifier will be disabled. The amplifier can be re-enabled with the [SH](#) command when there is no longer an Over-Current condition.

**WARNING**

If this error occurs, it is indicative of a problem at the system level. An Over-Current error is usually due to a short across the motor leads or a short from a motor lead to ground.

## ELO Input

If the ELO input on the controller is asserted, the power stage of the amplifier will be disabled. When the ELO input is triggered, [TA3](#) will change state and the [#AMPERR](#) routine will run on Thread 0. To recover from an ELO, the user must issue the [MO](#) command followed by the [SH](#) command. To recover from an ELO in a DMC program, use [MO;WT2;SH](#).

It is recommended that [OE1](#) be used for all axes when the ELO is used in an application.

See the ELO (Electronic Lock-Out) Input section in Chapter 3 Connecting Hardware for information on connecting the ELO input.

# A4 – DMC-30017

## Description

The DMC-30017 includes a sinusoidally commutated, 16-bit PWM amplifier that can also be configured as a microstepping drive. This amplifier is unique in that it can run either servo or stepper motors. The DMC-30017 can be configured for all of the motor below with just a few commands.

- Brushless Motors
- Brushed Motors
- Stepper Motors
- 2PB Motors

[GDK](#)'s Step-By-Step tool provides an interactive guide for configuring brushless, brushed, and stepper motors.

**WARNING**

Do not “hot swap” the motor power or supply voltage power input connections. If the amplifier is enabled when the motor connector is connected or disconnected, damage to the amplifier can occur. Galil recommends powering the controller and amplifier down before changing the connector, and breaking the AC side of the power supply connection in order to power down the amplifier.

## Electrical Specifications

	Brushless/Brushed/2PB	Stepper
<b>Supply Voltage</b>	20-80 VDC	
<b>Continuous Current</b>	10 Amps	-
<b>Peak Current</b>	15 Amps	-
<b>Maximum Current per Phase</b>	-	6 Amps
<b>Step Resolution</b>	-	256 microsteps/full step
<b>Switching Frequency</b>	33 kHz	
<b>Amplifier Gains</b>	0.4, 0.8, 1.6 Amps/Volt	0.75, 1.5, 3, 6 Amps/Phase
<b>Minimum Load Inductance</b>	$L(mH) = \frac{Vs(V)}{264 * I_{Ripple}(A)}$	

$V_s$  = Supply Voltage,  $I_{\text{ripple}} = 10\%$  of the maximum current at chosen gain

Table A4.1: Amplifier Electrical Specifications

## Mating Connectors

	Cable Connectors	Terminal Pins	On board Connector
<b>POWER</b>	2-pin Mini-Fit, Jr. <sup>TM</sup> (Male) Molex #0039012025	Molex #0444761112	2-pin Mini-Fit, Jr. <sup>TM</sup> (Female) Molex #0039310020
<b>A,B,C,D: 4-pin Motor Power Connectors</b>	4-pin Mini-Fit, Jr. <sup>TM</sup> (Male) Molex #0039012045	Molex #0444761112	4-pin Mini-Fit, Jr. <sup>TM</sup> (Female) Molex #0039310040

Table A4.2: Molex part numbers for connectors and terminal pin

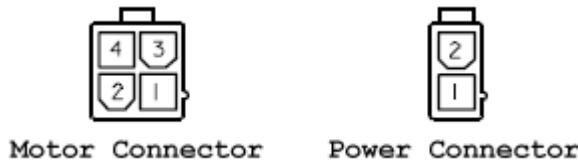


Figure A4.1: On Board Power and Motor Connector Diagram

Power Connector			
Pin Number	Connection		
1	DC Power Supply Ground		
2	+VS (DC Power)		
Motor Connector			
Pin Number	Brushless	Brushed	Stepper/2PB
1	Phase C	Phase A-	Phase B-
2	Phase B	No Connect	Phase B+
3	No Connect	No Connect	Phase A-
4	Phase A	Phase A+	Phase A+

Table A4.3: Power and Motor Connector Pin-outs

---

# Servo Motor Operation

## Brushless Motor Operation

For Brushless motor operation, **MT** must be set to **1** for the axis. The 6 commands used for setup are the **BA**, **BM**, **BX**, **BZ**, **BC** and **BI**. Please refer to the Command Reference for details. Below is the setup procedure for configuring an axis to operate a Brushless motor.

1. Issue the **BA** command to specify on which axis to configure sinusoidal commutation.
2. The brushless modulus is the number of encoder counts per magnetic cycle and is set with the **BM** command. For example, in a rotary motor that has 2 pole pairs and 10000 counts per revolution, the number of encoder counts per magnetic cycle would be 10000/2. **BM** should be assigned to a value of **5000**.
3. Issue either the **BI** and **BC**, **BZ**, or **BX** command to initialize the motor for sinusoidal commutation. The command must be executed on every reset or power-up of the controller.

### Setup Commands

1. **BI/BC** Commands – If the motor includes hall sensors, Galil recommends using the **BI** and **BC** commands. The **BI** command uses the hall sensors to set the approximate location of the motor in the magnetic cycle. The **BC** command will precisely set the commutation angle upon detecting a hall state transition.
2. **BZ** Command – If the motor does not include hall sensors, Galil recommends using the **BZ** command. The **BZ** command forces the motor to a known location in the magnetic cycle. Note that this can cause up to  $\frac{1}{2}$  a magnetic cycle of motion.
3. **BX** Command – If the motor does not have hall sensors and the **BZ** command generates too much motion for the application, Galil recommends using the **BX** command. This command utilizes a minimal movement algorithm in order to determine the correct commutation of the motor.

## 2PB Motor Operation

For 2PB motor operation, **MT** must be set to **4** for the axis. Initialization and setup of a 2PB motor is similar to the Brushless setup described above. The **BZ** command must be used to initialize commutation.

### Minimum Encoder Resolution

Due to the density of the magnetic cycle in 2PB motors, operation requires the use of an encoder with a high enough resolution resulting in a minimum **BM** value of **80** encoder counts per magnetic cycle. Contact a Galil Applications Engineer to review minimum encoder resolution requirements.

## Brushed Motor Operation

For brushed motor operation, set **MT** and **BR** to **1** for the axis.

## Setting Amplifier Gain and Current Loop Gain

The [AG](#) command will set the amplifier gain (Amps/Volt). The [AU](#) command will set the current loop gain.

### AG Command:

The DMC-30017 has 3 amplifier gain settings when configured for servo motors. The gain is set with the [AG](#) command. When configuring the amplifier gain, the axis must be in a motor off ([MO](#)) state prior to execution of the [AG](#) command. For more information on the [AG](#) command, refer to the command reference for the DMC-30017.

AG setting	Gain Value
n = 0	0.4 A/V
n = 1	0.8 A/V
n = 2	1.6 A/V

*Table A4.4: Amplifier Gain Settings*

### AU Command:

Proper configuration of the [AU](#) command is essential to optimize the operation of the DMC-30017. This command sets the gain for the current loop on the amplifier. Table A4.5 indicates the recommended [AU](#) settings for 24 and 48 VDC power supplies based on motor inductance. When configuring the current loop gain, the axis must be in a motor off ([MO](#)) state prior to execution of the [AU](#) command. For more information on the [AU](#) command, refer to the command reference for the DMC-30017.

Vsupply VDC	Inductance L (mH)	n
24	L < 1	9
24	1 ≤ L < 2.3	10
24	2.3 ≤ L < 4.2	11
24	4.2 ≤ L	12
48	L < 2.4	9
48	2.4 ≤ L < 4.2	10
48	4.2 ≤ L < 7	11
48	7 ≤ L	12

*Table A4.5: Amplifier Current Loop Gain Settings*

## Setting Peak and Continuous Torque Limits

### TK and TL Commands:

The peak and continuous torque limits can be set through the **TK** and **TL** commands respectively. The controller will command the torque signal (**TT**) up to the value specified by **TK** for a maximum of one second, as shown in [Figure A4.2](#). It will then limit the torque to the value specified by **TL**. The more time the controller commands the torque below **TL**, the longer it will command it to **TK** when necessary. The DMC code example below exhibits the behavior of **TT** being limited by **TL** and **TK**, as shown in [Figure A4.2](#).

```
#tk;          'begin program
MO;          'disable all axes
AGA=2;       'set max amplifier gain of 1.6 A/V
TLA=5;       'set continuous torque limit of 5V
TKA=9.9982;  'set max peak torque limit to 9.9982V
SHA;         'enable A axis
OFA=9.9982;  'command torque signal to 9.9982V
EN;          'end program
```

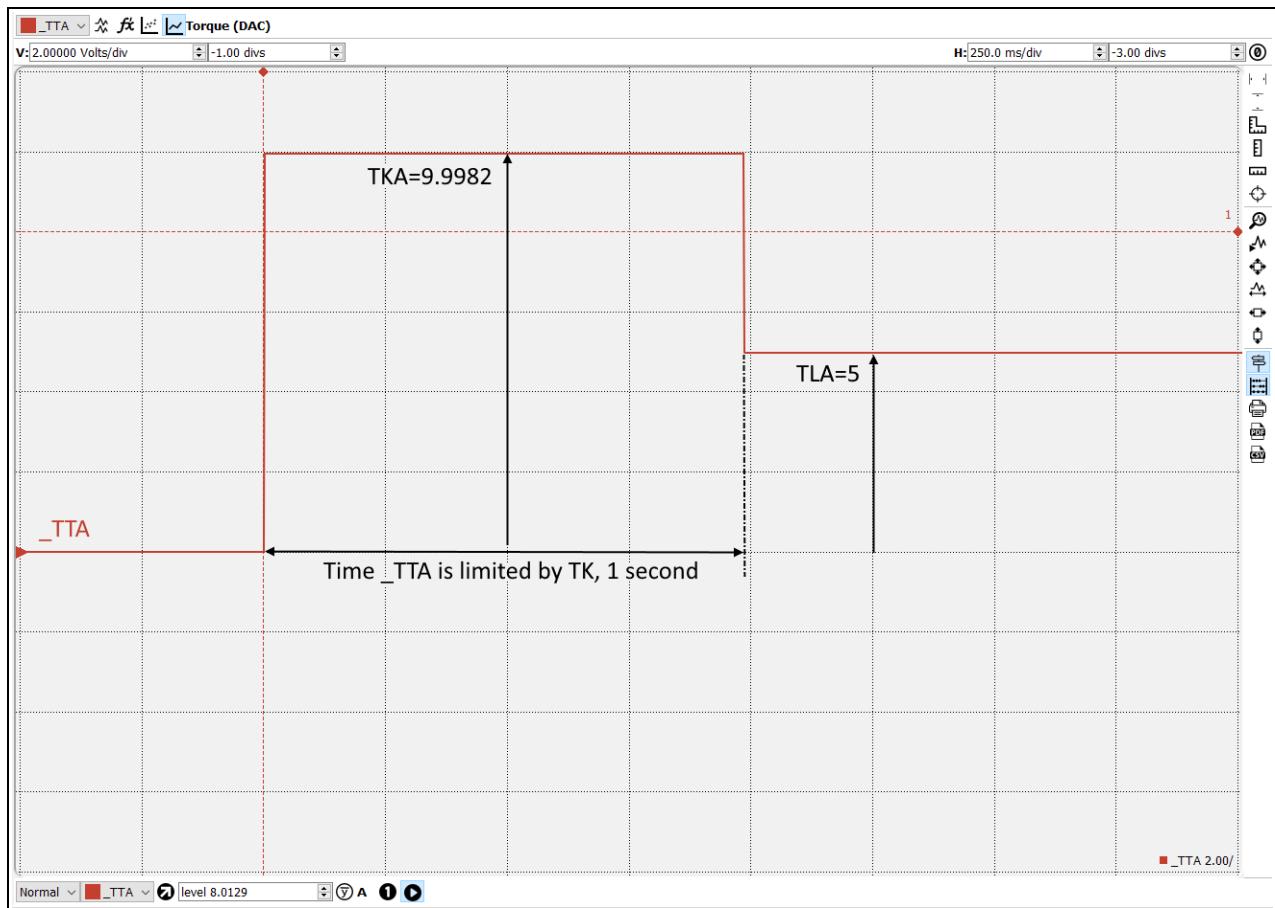


Figure A4.2: Peak Current Operation

---

# Stepper Motor Operation

To configure an axis for a bipolar stepper motor, set **MT** to **-2** for that axis.

## Setting Amplifier Gain and Current Loop Gain

The **AG** command will set the amplifier gain (Amps per phase). The **AU** command will set the current loop gain. Refer to Table A4.5 for more information.

### AG Command:

The DMC-30017 has 4 amplifier gain settings when configured for stepper motors. The gain is set with the **AG** command. When configuring the amplifier gain, the axis must be in a motor off (**MO**) state prior to execution of the **AG** command. For more information on the **AG** command, refer to the command reference for the DMC-30017.

AG setting	Gain Value
n = 0	0.75 A/Phase
n = 1	1.5 A/Phase
n = 2	3 A/Phase
n = 3	6 A/Phase

Table A4.6: Amplifier Gain Settings

### AU Command:

Proper configuration of the **AU** command is essential to optimize the operation of the DMC-30017. This command sets the gain for the current loop on the amplifier. Table A4.5 indicates the recommended **AU** settings for 24 and 48 VDC power supplies based on motor inductance. When configuring the current loop gain, the axis must be in a motor off (**MO**) state prior to execution of the **AU** command. For more information on the **AU** command, refer to the command reference for the DMC-30017.

## Low Current Mode

By default, 100% of the current set by the **AG** command will be delivered to the motor while the axis is holding position. An axis can be configured for Low Current mode so 25% of the current will be sent to the motor while holding position. Low Current mode can also be configured to wait a specified number of samples after a move is completed before lowering the current to 0%. For more information, refer to the **LC** command in the command reference for the DMC-30017.

LC Setting	Description
n = 0	100% holding current
n > 0	25% holding current immediately after move finishes current
n < 0	100% current n samples after move finishes before going to 25% holding current

Table A4.7: Low Current Mode Configurations

# Error Monitoring and Protection

This amplifier is protected against Over-Current, Over-Voltage, Over-Temperature, and Under-Voltage conditions. The amplifier errors can be monitored using the [TA](#) command. This command will return a bit mask number representing specific error conditions. It will also report whether or not an amplifier error is latched. For more information, refer to the [TA](#) command in the Command Reference.

To handle amplifier error conditions, users can include the [#AMPERR](#) automatic subroutine in their DMC program. This subroutine is user defined. Two conditions must be met for the [#AMPERR](#) subroutine to run: the [#AMPERR](#) subroutine must be defined in the DMC program; and an amplifier error must be detected. When these conditions are met, the program will automatically interrupt thread 0 and jump to the [#AMPERR](#) subroutine.

## Over-Current Protection

If there is a short circuit in the amplifier or motor, an Over-Current error will be reported and 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. The amplifier can be re-enabled with the [SH](#) command when there is no longer an over-current condition.

<b>WARNING</b>	If this error occurs, it is indicative of a problem at the system level. An Over-Current error is usually due to a short across the motor leads or a short from a motor lead to ground.
----------------	---

## Over-Voltage Protection

If the voltage supplied to the amplifier rises above its rated supply voltage, an Over-Voltage error will be reported. While the amplifier is experiencing this condition, current is no longer commanded and the motor phases are shorted together. This results in a drag on the motors causing them to coast to a stop. The amplifier will resume commanding current when the voltage drops below the maximum rated supply voltage.

The over voltage condition will not permanently shut down the amplifier and will not trigger the [#AMPERR](#) routine.

<b>WARNING</b>	If this error occurs, it is indicative of a problem at the system level. An Over-Voltage error is usually due to regeneration from the motor and load. Consider using a shunt regulator if this occurs during normal operation.
----------------	---

## Over-Temperature Protection

If the average amplifier temperature rises above 80°C, an Over-Temperature error will be reported and the amplifier will be disabled. 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).

<b>WARNING</b>	An Over-Temperature error is usually due to insufficient heat dissipation from the amplifier. Heat must be drawn away from the base of the controller with forced air flow or a heat sink.
----------------	--

## **Under-Voltage Protection**

If the voltage supplied to the amplifier drops below its rated supply voltage, an Under-Voltage error will be reported and the amplifier will be disabled.

If there is an **#AMPERR** routine and the controller is powered before the amplifier, then the **#AMPERR** routine will automatically be triggered. The amplifier will return to normal operation once the supply is raised above the rated supply voltage assuming it did not cause the position error to exceed the error limit.

## **Peak Current**

If the commanded torque value (**TT**) on any axis equals the peak torque limit set by **TK**, a Peak Current condition will be reported for that axis, and the axis bit in the **TA2** bit mask will be set. Peak Current does not cause **#AMPERR** to run. The Peak Current condition reports live and is not latching.

## **ELO Input**

If the ELO input on the controller is asserted, the power stage of the amplifier will be disabled. When the ELO input is triggered, **TA3** will change state and the **#AMPERR** routine will run on Thread 0. To recover from an ELO, the user must issue the **MO** command followed by the **SH** command. To recover from an ELO in a DMC program, use **MO;WT2;SH**.

It is recommended that **OE1** be used for all axes when the ELO is used in an application.

See the ELO (Electronic Lock-Out) Input section in Chapter 3 Connecting Hardware for information on connecting the ELO input.

# A5 – DMC-31000

## Description

The DMC-31000 is an option that allows for the controller to accept sinusoidal encoder signals in addition to standard, differential quadrature encoder signals<sup>1</sup>. The DMC-31000 option can provide interpolation for a single 1 V<sub>pk-pk</sub> differential sinusoidal encoder resulting in a higher position resolution. The AF command is used to select the degree of interpolation, see AF in the Command Reference for more details.

**<sup>1</sup>NOTE**

The DMC-31000 options *requires* differential signals if using a quadrature encoder.  
*See the ID command in the Command Reference for board identification.*

When wiring either Sin/Cos or standard differential encoders on an axis will use the same pins, see DMC-31000 Encoder 15 pin HD D-Sub Connector (Female), pg 200 for pin-outs.

The DMC-31000 requires specific firmware for the implementation of Sin/Cos encoders. Any unit ordered with this option will automatically be loaded with this firmware at the factory. With this firmware, the maximum speed settings will be increased from 22,000,000 [cts/s] to 50,000,000 [cts/s].

See Theory of Operation, pg 200 and Calculating Equivalent Counts, pg 201 for learning how the DMC-31000 interpolates Sin/Cos signals.

## Analog Inputs

With the DMC-31000 the analog inputs are upgraded to 16-bit, ±10V configurable, see the Analog Inputs section in Chapter 3 Connecting Hardware for more information.

## DMC-31000 Encoder 15 pin HD D-Sub Connector (Female)

Pin #	Label	Sin/Cos Feedback	Standard Quadrature
1	MI+	$V_0+$ Index Pulse Input	I+ Index Pulse Input
2	MB+	$V_2+$ Main Encoder Input	B+ Main Encoder Input
3	MA+	$V_1+$ Main Encoder Input	A+ Main Encoder Input
4	AB+		B+ Aux Encoder Input
5	GND	Digital Ground	
6	MI-	$V_0-$ Index Pulse Input	Index Pulse Input
7	MB-	$V_2-$ Main Encoder Input	B- Main Encoder Input
8	MA-	$V_1-$ Main Encoder Input	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

---

## Theory of Operation

Traditional quadrature rotary encoders work by having two sets of lines inscribed radially around the circumference of an optical disk. A light is passed through each of these two sets of lines. On the other side of the gratings, photo sensors detect the presence (or absence) of these lines. These two sets of lines are offset from each other such that one leads the other by one quarter of a complete cycle as shown in Figure A5.1 below. These signals are commonly referred to as the Channels A and B. The direction of rotation of the encoder can be inferred by which of the A and B signals leads the other. Each rising or falling edge indicates one quadrature count. Thus for a complete cycle of the square wave there are a total of four encoder counts.

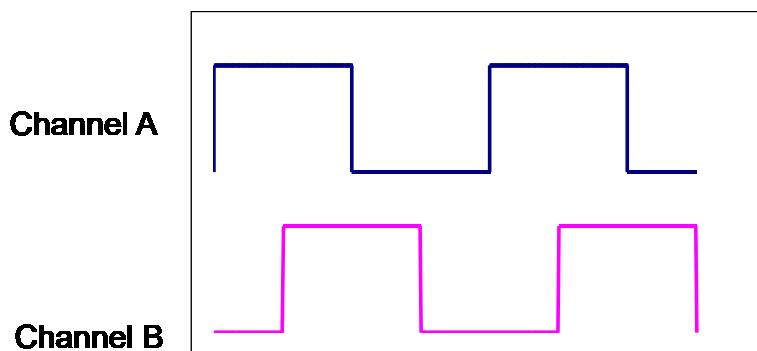


Figure A5.1: Quadrature Encoder Signals

A sinusoidal encoder is similar to a quadrature encoder in that it produces two signals that are read from two sets of lines inscribed on an optical disk. The difference is that the two signals are output as analog sinusoidal waves as shown in Figure A5.2.

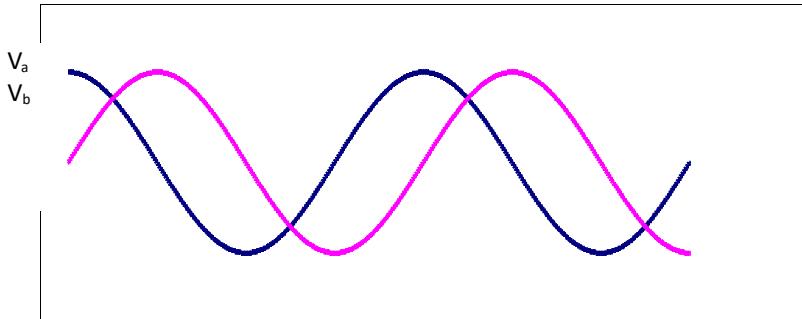


Figure A5.2: Sinusoidal Encoder Signals

When the DMC is ordered with the DMC-31000 option, the position is tracked on two levels. First, the number of coarse cycles is counted much like is done with a quadrature encoder. On the fine level the precise position inside the cycle is determined from the two sinusoidal signals using bit-wise interpolation. This interpolation can be set by the user in the range of  $2^5$  through  $2^{12}$  points per sinusoidal cycle via AF command. See the AF command in the Command Reference for more information.

$$\text{Fine} = \frac{2^n}{360} \tan^{-1}\left(\frac{V_b}{V_a}\right)$$

The overall position can be determined using:

$$\text{Position} = \text{Coarse\_cycles} \cdot 2^n + \text{Fine}$$

Where:

$n$  is the number of bits of resolution that were used in the conversion.

`Coarse_cycles` is the whole number of cycles counted.

`Fine` is the interpolated position within one cycle.

$V_b$  and  $V_a$  are the two signals as indicated in Figure A5.2.

For example, if the encoder cycle is 40 microns, AF10 results in  $2^{10}=1024$  counts per cycle, or a resolution of 39 nanometers per count.

## Calculating Equivalent Counts

The units of distance is counts in DMC code and all feedback types, including sin/cos feedback, is translated into equivalent counts. Below is a brief example of how a user would calculate sin/cos periods into counts which is helpful in determining the resolution of the system.

Example:

Assume that a motor has 1000 sin/cos periods per revolution. With *no* interpolation the controller will interpret a single sin/cos period as 4 equivalent counts. Thus the total counts per revolution would be as follows:

$$1000 \left[ \frac{\text{sin/cos period}}{\text{rev}} \right] \times 4 \left[ \frac{\text{counts}}{\text{sin/cos period}} \right] = 4000 \text{ [counts/rev]}$$

Using “AF 5” the user has selected to interpolate the sin/cos signal to  $2^5$  counts per sin/cos period. The following counts/rev would be calculated as follows:

$$1000 \left[ \frac{\text{sin/cos period}}{\text{rev}} \right] \times 2^5 \left[ \frac{\text{counts}}{\text{sin/cos period}} \right] = 32,000 \text{ [counts/rev]}$$

### Latching with Sinusoidal Encoders:

The function of the hardware latch feature for the DMC-31000 is illustrated with an example below. Figure A5.3 shows a sinusoidal encoder signal and its equivalent quadrature representation. Position latching is reported using the `RL` command and latching can be triggered based on the index or digital input via the `AI` command. The value that `RL` reports is the position of the sinusoidal encoder at the last quadrature transition rounded towards 0.

As an example, the latch takes place between points “B” and “C” and position 0 is to the right of both “B” and “C”. The position that `RL` will report will be the sinusoidal position at point “C”. Alternatively, if the latch takes place between points “B” and “C” and position 0 is to the left of “B” and “C”, the position that `RL` will report will be the sinusoidal position at point “B”.

**NOTE:** The latching for sinusoidal encoders is valid on DMC-310xx firmware versions 1.2i and later.

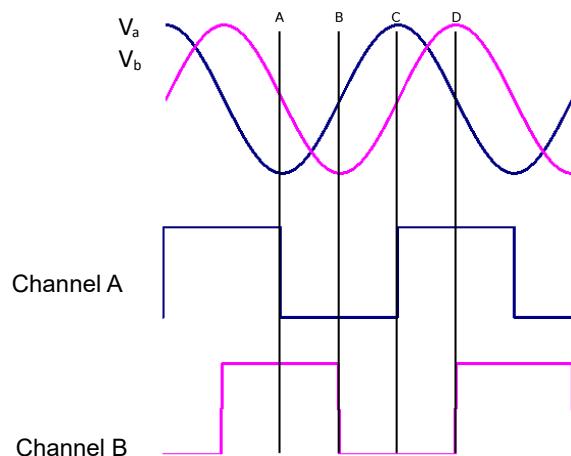


Figure A5.3