USER MANUAL

DMC-52xx0

Manual Rev. 1.0c

EtherCAT®

Galil Motion Control, Inc.

270 Technology Way
Rocklin, California

916.626.0101
support@galil.com
www.galil.com

08/2016

Using This Manual

This user manual provides information for proper operation of the DMC-52xx0 controller. Two separate supplemental manuals, the DMC-52xx0 Command Reference and EtherCAT Setup Guide, contain a description of the commands available for use with this controller and information on how to set up Galil's supported EtherCAT drives. It is recommended that the user download the latest version of the Command Reference, EtherCAT Setup Guide, and User Manual from Galil's Website.

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

For drive specific setup information, view the Galil EtherCAT Setup Guide:

http://www.galil.com/download/manual/man_500x0_setup.pdf

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

Contents

Chapter 1 Overview

Introduction

The DMC-52xx0 EtherCAT master is Galil's first 32 axis motion controller. It's a pure EtherCAT controller with the ability to control up to 32 drives and 2 I/O modules. This space efficient package also provides uncommitted I/O for easy integration into any EtherCAT application.

The DMC-52xx0 is offered in 2, 4, 8, 16, and 32 axis formats. Coordinated moves can be done within banks of up to 8 axes allowing for minimal changes of Galil's programming language. The DMC-52xx0 operates in Cyclic Synchronous Position mode (CSP). In this mode, the servo control loop is closed on the EtherCAT drive while the Galil controller sends motion profile commands at a rate of 1 kHz. Galil supports a host of EtherCAT devices for virtually any application. See the list below for more details.

Designed to solve complex motion problems, the DMC-52xx0 can be used for applications involving jogging, point-to-point positioning, position tracking, contouring, linear and circular interpolation, electronic gearing, ECAM and PVT. The DMC-52xx0 makes configuration and programming easy with just a handful of EtherCAT configuration commands and Galil's intuitive, two-letter programming language.

The DMC-52xx0 features 8 uncommitted opto-isolated inputs and 8 uncommitted opto-isolated high power outputs. It also includes 8 uncommitted analog inputs and 8 uncommitted analog outputs (analog I/O is 12-bit standard, 16-bit option available).

The DMC-52xx0 comes with one Ethernet port for communication with a host PC and one EtherCAT port to communicate with EtherCAT drives. Multiple EtherCAT drives can be linked together in a daisy chain configuration and connected to the controller's EtherCAT port, simplifying wiring and decreasing setup time. One USB port is also provided for alternative communication with a host PC. The DMC-52xx0 is packaged in a compact metal enclosure measuring 9.75" x 5.00" x 1.60" and is powered by a single 90-250 VAC supply.

Part Numbers

The DMC-52xx0 has the part number format “DMC-52xx0(Y),” where x designates the number of axes and Y designates the configuration option. The DMC-52xx0 is available in 2, 4, 8, 16, and 32 axis variants. See Table 1.1 for configuration options.

Galil's on line part number generator is a easy to use tool for building your DMC-52xx0
http://www.galil.com/order/part-number-generator/dmc-52xx0

DMC-52xx0(Y)	
 *DMC-52xx0*	
DMC-52xx0 Option (Y)	
Part Number	**Description**
16bit	16-bit analog inputs

Table 1.1: DMC-52xx0 Functional Elements

Overview of Motor Types

The DMC-52xx0 controls EtherCAT Servo Drives in Cyclic Synchronous Position mode (CSP). In this mode, the servo control loop is closed on the EtherCAT drive while the Galil controller sends motion profile commands at a rate of 1 kHz. Via Distributed Clock, all drives on the EtherCAT Network share a common servo interrupt signal with the EtherCAT Master that is maintained via constant adjustment. This ensures tightly coordinated, synchronized motion between all drives, enabling real time control over large scale, distributed networks.

Overview of EtherCAT Amplifiers

EtherCAT Amplifiers

Standard support for an EtherCAT Drive with the DMC-52xx0 includes the following features:

- Cyclic Synchronous Position mode of operation utilizing Distributed Clock
- Remote forward and reverse limit switch inputs
- Remote home sensor
- Remote hardware latch/touchprobe and latch on index

Note: Galil has made every effort to match the supported drives' pinout as possible. Please see The EtherCAT Setup Guide for more details.

Functional Elements

The DMC-52xx0 circuitry can be divided into the following functional groups discussed below.

Microcomputer Section

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

Communication

The communication interface with the DMC-52xx0 consists of high speed USB and Ethernet. The Ethernet is 10/100Bt and the one USB channel, see Chapter 4 Communication, pg 22 for details.

The DMC-52xx0 communicates with EtherCAT servo drives over shielded CAT5 Ethernet Cable using the COE (CAN Over EtherCAT) protocol.

General I/O

The DMC-52xx0 provides interface circuitry for 8 bi-directional, optoisolated inputs, 8 high power sourcing optoisolated outputs, 8 analog inputs with 12-Bit ADC (16-Bit optional), and 8 analog outputs with 12-Bit ADC (16-Bit optional).

System Elements

As shown in Figure 1.1, the DMC-52xx0 is part of a motion control system which includes amplifiers, motors and encoders. These elements are described below.

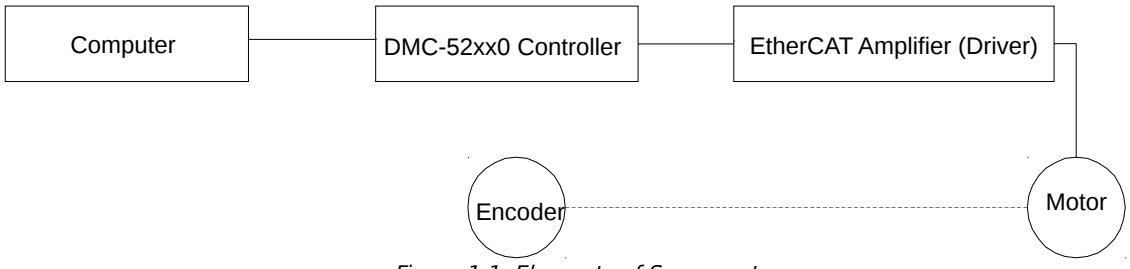

Figure 1.1: Elements of Servo systems

Motor

A motor converts current from the amplifier into torque which produces motion. Each axis of motion requires a motor sized properly to move the load at the required speed and acceleration.

EtherCAT Amplifier (Driver)

An EtherCAT amplifier uses the EtherCAT digital communication bus instead of an analog voltage command signal. This allows any EtherCAT master device to control the amplifier in different modes of operation. Each manufacturer's EtherCAT drive has different features and capabilities. Refer to the manufacturer's documentation to see which motors, position feedback options, and modes of operation are supported. The DMC-52xx0 supports the Cyclic Synchronous Position mode for EtherCAT amplifiers. When operating in this mode the DMC-52xx0 will function as the EtherCAT master and provide a commanded position value to the EtherCAT drive at a rate of 1 kHz.

Encoder

An encoder translates position into electrical pulses which are fed back into the EtherCAT drive. The drive uses this information to close the position control loop.

Chapter 2 Getting Started

Layout

DMC-52xx0

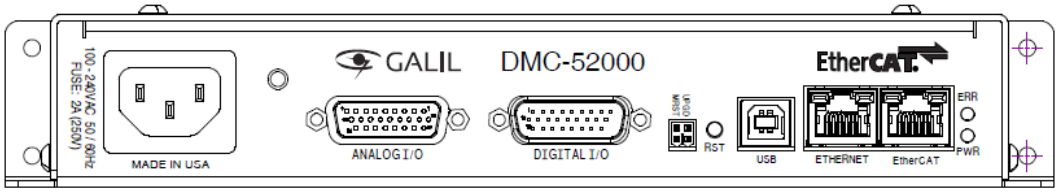

Figure 2.1: Outline of the DMC-52xx0

Power Connections

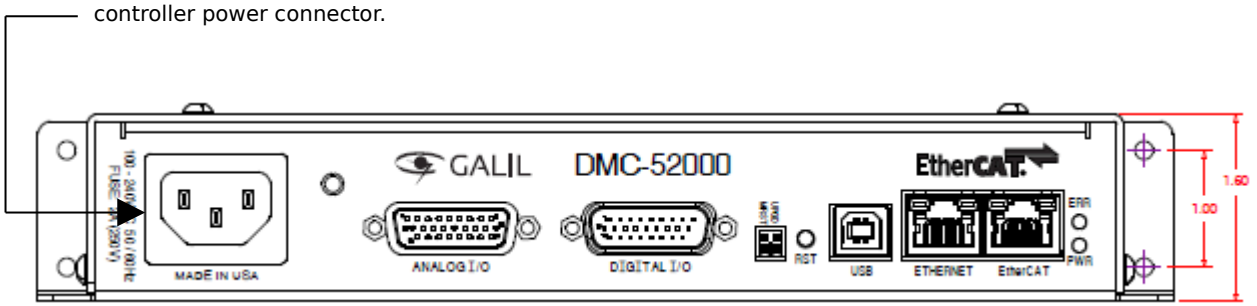

Figure 2.2: Power Connector location for the DMC-52xx0

For more information on powering your controller see Step 4. Power the Controller, pg 13. For more information regarding connector type and part numbers see Power Connector Part Numbers, pg 134. The power specifications for the controller are provided in Power Requirements, pg 132.

Dimensions

DMC-52xx0

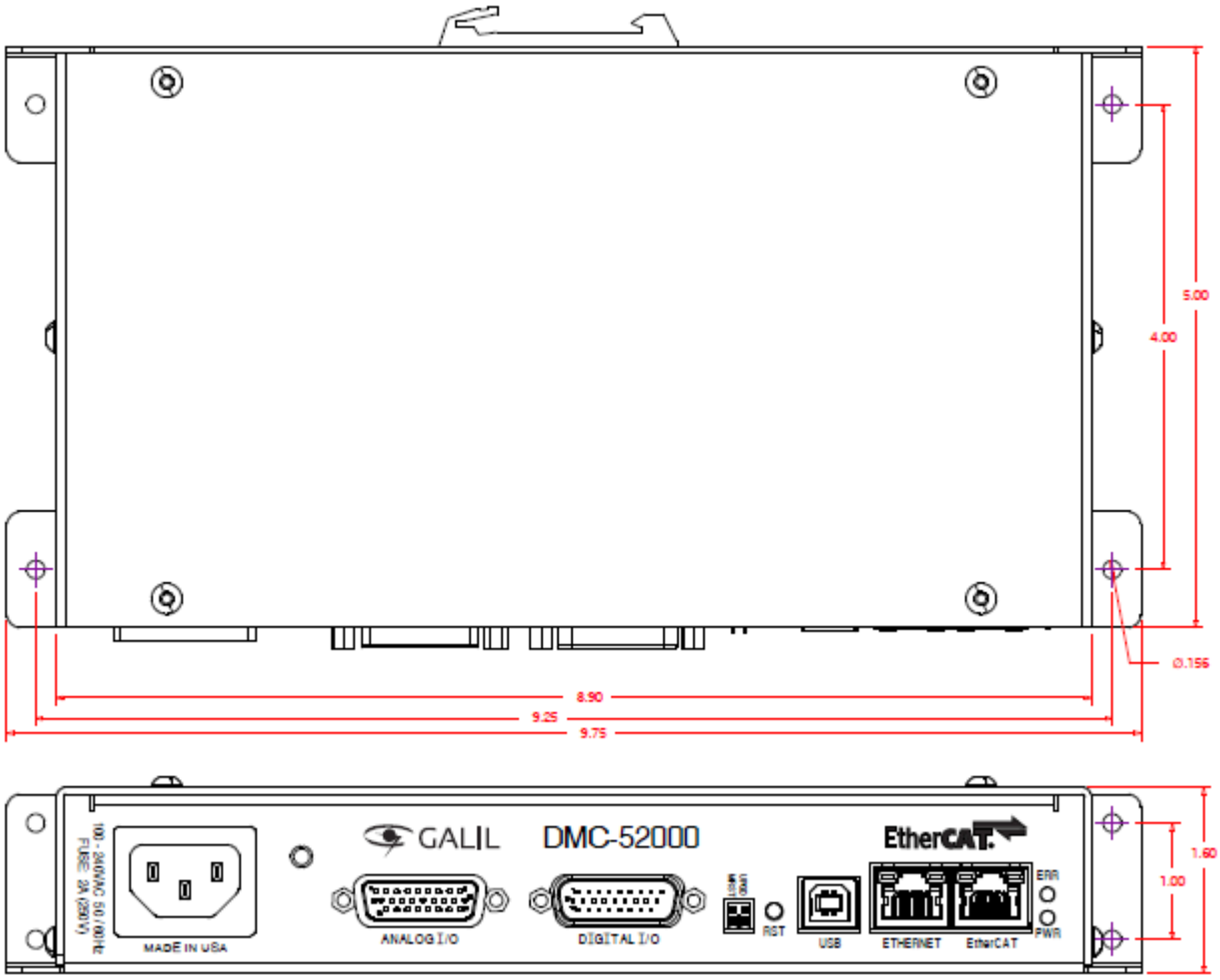

Figure 2.3: Dimensions (in inches) of DMC-52xx0

Elements You Need

For a complete system, Galil recommends the following elements:

1. DMC-52xx0, motion controller where the xx designates number of axes: 2, 4, 8, 16, or 32.
2. EtherCAT Drives and I/O
3. Power Supply for Amplifiers and controller
4. PC (Personal Computer – USB (serial) or Ethernet for DMC-52xx0)
5. Galil software package

Installing the DMC, Amplifiers, and Motors

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

WARNING	Electronics are dangerous! Only a certified electrical technician, electrical engineer, or electrical professional should wire the DMC product and related components. Galil shall not be liable or responsible for any incidental or consequential damages. 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 3rd party device.

Step 1. Determine Overall System Configuration

Before setting up the motion control system, the user must determine the desired motor configuration. DMC-52xx0 is configured to function with EtherCAT drives in Cyclic Synchronous Position mode (CSP). Galil has a verity of supported EtherCAT devices to accommodate any application. Please see the following link for a list of Galil's supported EtherCAT devices.

http://www.galil.com/ethercat

See Part Numbers, pg 5 for understanding your complete DMC unit and part number before continuing.

Step 2. Install Jumpers on the DMC-52xx0

The following jumpers are located in a rectangular cut-out near the digital I/O 26 pin HD D-Sub connector.

Master Reset and Upgrade Jumpers

Jumpers labeled MRST and UPGD are the Master Reset and Upgrade jumpers, respectively.

When the MRST pins are jumpered, the controller will perform a master reset upon a power cycle, the reset input pulled down, or a push-button reset. Whenever the controller has a master reset, all programs, arrays, variables, and motion control parameters stored in EEPROM will be erased and restored back to factory default settings.

The UPGD jumper enables the user to unconditionally update the controller's firmware. This jumper should not be used without first consulting Galil.

Step 3. 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 most recent copy of Galil's development software can be found here:

http://www.galil.com/downloads/software

Step 4. Power the Controller

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

The DMC-52xx0 accepts a single 90-250 V_{AC} power input. See Power Connections, pg 9 for the location of the power connections of the DMC-52xx0.

The DMC-52xx0 power should never be plugged in HOT. Always power down the power supply before installing or removing power connector(s) to or from controller.

The green power light indicator should go on when power is applied. The red error light should also go on but quickly turn off.

Step 5. Establish Communications with Galil Software

The DMC-52xx0 is connected to the EtherCAT drives by a CAT5 cable. The EtherCAT output port on the DMC-52xx0 EtherCAT Master is labeled as EtherCAT . The port labeled as Ethernet is used for communicating with the DMC-52xx0 EtherCAT Master over Ethernet from a PC or HMI.

See Ethernet Configuration, pg 25 for details on using Ethernet with the DMC-52xx0. To learn how to configure your network interface card to connect to a DMC controller, see this two-minute video:

http://www.galil.com/learn/online-videos/connecting-galil-ethernet-motion-controller

For connecting proper configuration of the DMC-52xx0 USB port using USB (serial), see USB Port, pg 23.

Step 6. Setting Safety Features before Wiring Motors

Step A. Set the Error Limit

When ER (error limit) and OE (off-on-error) are set, the controller will automatically shut down the motors when excess error (|TE| > ER) has occurred. This is an important safety feature during set up as wrong polarity can cause the motor to run away.

Step B. Other Safety Features

This section only provides a brief list of safety features that the DMC can provide. Other features include Automatic Subroutines to create an automated response to events such as limit switches toggling (#LIMSWI), command errors (#POSERR), EtherCAT related errors (EZ, EK, #ECATERR), and more. For a full list of features and how to configure them see Chapter 8 Hardware & Software Protection, pg 125. Also, consult the drives documentation for more information on drive related safety features.

Step 7. Connecting EtherCAT Amplifiers and Motors

Before use, each EtherCAT drive must be initialized and configured for operation with the DMC-52xx0 via the drive vendor's software. This step is required only once when first setting up the EtherCAT network. In most cases this consists of assigning digital inputs and disabling specific drive error handling routines, yielding control to the DMC-52xx0.

Please see Galil's EtherCAT Setup Guide for drive specific setup details:

http://www.galil.com/download/manual/man_500x0_setup.pdf

Once an EtherCAT Drive has been initialized and configured via software, connecting to the drive with the DMC-52xx0 is done by wiring the controller's EtherCAT port to the Drive's EtherCAT IN port via shielded CAT5 Ethernet Cable. Subsequent drives can be connected in a 'daisy chain' configuration where the OUT port of one drive is connected to the IN port of the next drive or IO module.

Step 8. Tune the Servo System

Proper servo control requires adjustment of the tuning parameters, commonly known as 'tuning' the servo system. The DMC-52xx0 operates in Cyclic Synchronous Position Mode, meaning servo control is handled on the EtherCAT Drive. In this case, all tuning of the motors is done on the drives using the vendor specific configuration software. Standard Galil PID parameters (KD, KP, KI, PL etc.) will have no effect and will return an error if issued from a host terminal or dmc code.

Consult the drive's documentation for more information on tuning.

Galil provides a library of tutorial videos on servo tuning here: http://www.galil.com/learn/online-videos

Chapter 3 Connecting Hardware

Overview

In addition to supporting up to two separate IO modules on an EtherCAT network, the DMC-52xx0 comes standard with a set of local digital and analog IO as well. Included are 8 optoisolated, uncommitted digital inputs as well as 8 high power sourcing optoisolated digital outputs. 8 programmable analog inputs and outputs are also provided. Additional digital IO includes a reset input, abort input, and error outputs.

Forward and reverse limit inputs, home sensor inputs and position latch/touchprobe inputs for each axis are wired into the drive's IO connector. See the EtherCAT Setup Guide for more information about Galil's supported EtherCAT drives.

This chapter describes the DMC-52xx0 local inputs, outputs, and their proper wiring.

Overview of Optoisolated Inputs

Abort Input

The function of the Abort Input is to immediately stop all motion and/or dmc code execution upon transition of the logic state. Depending on the value of the CN command, triggering the Abort input will also stop execution of any DMC code on the controller.

Note: When the Abort Input is activated, the controller stops generating motion commands immediately. The result is a rapid deceleration which may exceed the mechanical capabilities of the system. The Abort Input is meant to be used as a safety feature and should not be used in standard operation to halt motion.

Note: The effect of triggering the the Abort Input is dependent on the state of the Off-on-Error function (OE Command) for each axis. If the Off-on-Error function is enabled for any given axis, the motor for that axis will be turned off when the abort signal is generated. This could cause the motor to ‘coast’ to a stop since it is no longer under servo control. If the Off-on-Error function is disabled, the motor will decelerate to a stop as fast as mechanically possible and the motor will remain in a servo state.

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

The state of the Abort Input can be queried with the _AB operand.

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-52xx0 includes 8 optoisolated digital inputs. These inputs can be read individually using the @IN[x] operand where x specifies the input number (1 thru 8). These inputs are uncommitted and allow the user to create conditional statements related to events external to the controller. For example, the user may wish to have the A axis motor move 1000 counts in the positive direction when the logic state of Digital Input 1 goes high. Digital Inputs are queried using the TI command or the @IN[x] operand where x is a number 1 thru 8. Digital Inputs on the DMC-52xx0 are read on the same interrupt as EtherCAT Drive and IO module data to ensure deterministic operation. The TI command and @IN[x] operands will return the value read from hardware on the previous interrupt.

Optoisolated Input Electrical Information

Electrical Specifications

INCOM Max Voltage	24 V_{DC}
INCOM Min Voltage	0 V_{DC}
Minimum current to turn on Inputs	1.2 mA
Minimum current to turn off Inputs once activated (hysteresis)	0.5 mA
Maximum current per input[1]	11 mA
Internal resistance of inputs	2.2 kΩ

[1] See Input Current Limitations, pg 134 section for more details.

The DMC-52xx0's optoisolated inputs are rated to operate with a supply voltage of 5–24 VDC. The INCOM pin, located on the 26-pin male D-sub connector, provides power to DI[8:1] (Digital Inputs), the Abort Input (ABRT), and Reset Input (RST).

The full pinouts for each bank can be found in the Pinouts section of the Appendix, pg 132.

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.

Since these inputs are bidirectional they can be used as either active high or low. Connecting $+V_s$ to INCOM will configure the inputs for active low as current will flow through the diode when the inputs are pulled to the isolated ground. Connecting the isolated ground to INCOM will configure the inputs for active high as current will flow through the diode when the inputs are pulled up to $+V_s$.

The wiring diagram for Digital Inputs 1 thru 8 can be seen in Figure 3.1.

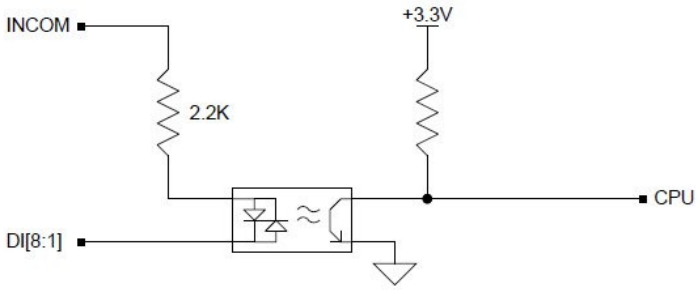

Figure 3.1: Digital Inputs 1-8 (DI[8:1])

Optoisolated Outputs

This section will describe the DMC-52xx0's 8 optically isolated 500mA sourcing outputs .

See the Appendix for your for pinouts: Pinouts or 132.

Description

The high power digital outputs are capable of sourcing up to 500mA per output and up to 3A total across all 8. The voltage range for the outputs is 12-24 VDC. These outputs include flyback diodes and are capable of driving inductive loads such as solenoids or relays. The outputs are configured for hi-side (sourcing) only. Digital Outputs on the DMC-52xx0 are written on the same interrupt as EtherCAT Drive and IO module data to ensure deterministic operation. The `SB,CB,OP,` and `OB` command values will be written to the digital output hardware on the next interrupt. Note that digital output values are not 'burnable'. A controller reset, power cycle, or master reset will return all Digital Outputs to an inactive state.

Electrical Specifications

Output PWR Max Voltage	24 VDC
Output PWR Min Voltage	12 VDC
Max Drive Current per Output	0.5 A (maximum 3A per Bank)

Wiring the Optoisolated Outputs

The output power supply should 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 Digital Outputs 1 thru 8 is shown in Figure 3.2.

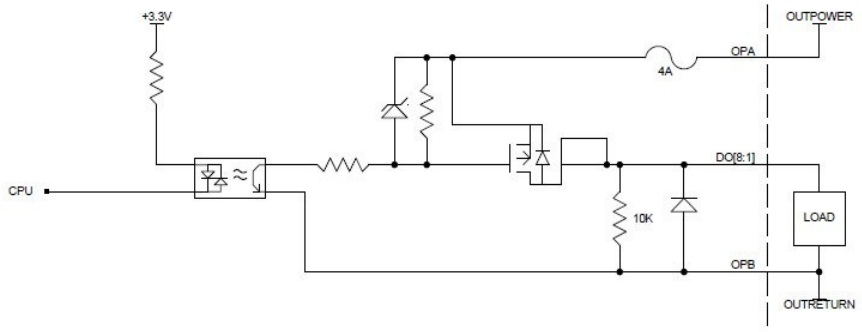

Figure 3.2: 500mA Sourcing wiring diagrams for DO[8:1]

Error Output

The Error Output consists of an optoisolator circuit that is tied to an NPN (sinking) photo transistor. When an error condition occurs, the controller CPU activates the photo transistor. The result is that the ERROR_C output will be brought low and the controller's red error LED will light.

An error occurs due to one of the following conditions:

1. At least one axis has a position error greater than the error limit (TE>ER). The position error is queried by using the TE command. 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.

For additional information on these conditions, seeChapter 9 Troubleshooting, pg 129.

Electrical Specifications

Output Voltage	0 – 30 VDC
Current Output	25 mA Sinking

Wiring the Error Outputs

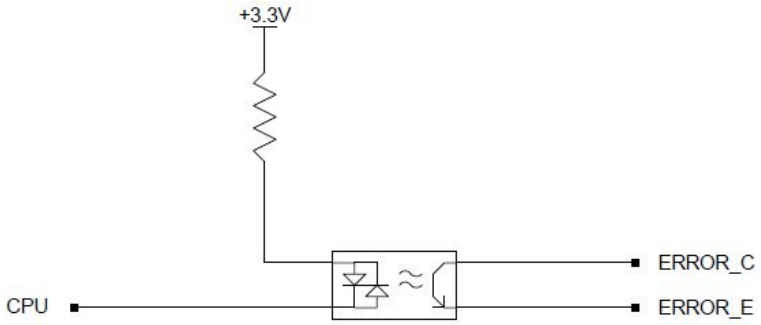

Figure 3.3: Error Output

Analog Inputs

The DMC-52xx0 has eight analog inputs user configurable for multiple ranges between -10V and +10V. The inputs are decoded by a 12-bit A/D conversion giving a voltage resolution of approximately .005V. A 16-bit A/D converter is available as an option (Ex. *DMC-52020(16bit)*). The analog inputs are queried using the @AN[x] operand where x is a number 1 thru 8. Analog Inputs on the DMC-52xx0 are read on the same interrupt as EtherCAT Drive and IO module data to ensure deterministic operation. The @AN[x] operand will return the value read from hardware on the most recent interrupt.

AQ settings

The analog inputs can be set to a range of ±10V, ±5V, 0-5V, or 0-10V. This allows for increased resolution when the full ±10V is not required. The inputs can also be configured for differential mode where analog inputs 2,4,6, or 8 can be set to the negative differential inputs for analog inputs 1,3,5, or 7 respectively. The AQ command is a configuration command and as such should be used only when configuring the system. Galil strongly recommends writing AQ settings to EEPROM via the BN Command or setting them on power up via the #AUTO routine. See the AQ command in the DMC-52xx0 Command Reference for more information.

Electrical Specifications

Input Impedance (12 and 16 bit) –

Unipolar (0-5V, 0-10V)	42kΩ
Bipolar (±5V, ±10V)	31kΩ

Analog Outputs

The DMC-52xx0 includes eight analog outputs configurable for multiple ranges between -10V and +10V. Standard resolution is 12-bit and a 16-bit DAC is available as an option (Ex. *DMC-52020(16bit)*). The analog outputs are set using the AO command. Analog outputs on the DMC-52xx0 are written on the same interrupt as EtherCAT Drive IO module data to ensure deterministic operation. The AO command values will be written to the analog output hardware on the next interrupt. Note that analog output values are not 'burnable'. A controller reset, power cycle, or master reset will return all analog outputs to 0V.

DQ settings

The analog outputs can be set to a range of ±10V, ±5V, 0-5V, or 0-10V, allowing for increased resolution when the full ±10V is not required. The DQ command is a configuration command and as such should be used only when configuring the system. Galil strongly recommends writing DQ settings to EEPROM via the BN Command or setting them on power up via the #AUTO routine. Note that issuing the DQ Command will set the output to 0 V. See the DQ command in the Command Reference for more information.

Electrical Specifications

Maximum Output Voltage	10V
Minimum Output Voltage	-10V

Resolution	12-bit default, 16-bit optional
Maximum Current Output	4mA (sink/source)

Chapter 4 Communication

Introduction

The DMC-52xx0 has one USB port, one EtherCAT port, and one Ethernet port. The USB port provides a serial connection to communicate with the controller. The Ethernet port provide a 10/100BASE-T connection that auto-negotiates the speed at half or full duplex. Ethernet port 0 is for TCP/IP communications with the controller, Ethernet port 1 is for communications with EtherCAT slave devices.

Galil current generation software is available for PC's to communicate with the DMC-52xx0 controller. In addition, a communication library 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 communication protocols used by the DMC-52xx0, and a brief introduction to the software tools and communication techniques used by Galil. At the application level, the current generation Galil software is what the majority of users will need in order to communicate with the controller, to perform basic setup, and to develop application code (.dmc programs). At the Galil API level, the current communication library is available for users who wish to develop their own custom application programs. These programs can utilize API function calls directly to our DLL's. See the following links to Galil's current generation software and communication library.

Current generation software: http://www.galil.com/downloads/software

Current generation communication library: http://www.galil.com/downloads/api

Controller Response to Commands

Most DMC-52xx0 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.

Instructions are sent in ASCII, and the DMC-52xx0 decodes each ASCII character (one byte) one at a time. It takes approximately 40 µsec for the controller to execute each command.

After the instruction is decoded, the DMC-52xx0 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. The controller will respond to commands which are sent via the USB port back through the USB 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-52xx0 will return the data followed by a carriage return, line feed and a colon (:) .

It is good practice to check for a colon after each command is sent. An echo function is provided to enable associating the DMC-520x0 response with the data sent. The echo is enabled by sending the command EO 1 to the controller. This capability is only available via USB.

Unsolicited Messages Generated by Controller

When the controller is executing a program, it may generate responses which will be sent via the USB port or Ethernet port. 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 which is the USB 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 current generation Galil 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.

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

Serial Communication Ports

USB Port

The USB port on the DMC-52xx0 is a USB to serial converter. It should be setup for 115.2kB, 8 Data bits, No Parity, 1 Stop Bit, and Flow Control set for Hardware. The USB port on the DMC-52xx0 is a Female Type B USB port. The standard cable when communicating to a PC will be a Male Type A to Male Type B USB cable.

When connected to a PC, the USB connection will be available as a new serial port connection (ex. with Galil Software “COM3 115200”).

115.2 kB
8 Data bits
No Parity
1 Stop Bit
CTS/RTS

Table 4.2: Serial Communication Settings

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-52xx0 supports two industry standard protocols, TCP/IP and UDP/IP. The controller will automatically respond in the format in which it is contacted.

TCP/IP is a "connection" protocol. The master, or client, connects to the slave, or 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.

In contrast, 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 highly recommended that the motion control network containing the controller and any other related devices be 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. The TCP handshaking is not required because handshaking is built into the Galil communication protocol through the use of colon or question mark responses to commands sent to the controller.

Packets must be limited to 512 data bytes (including UDP/TCP IP Header). Larger packets could cause the controller to lose communication.

Note: Prevent the lose in 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-52xx0 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-52xx0 unit, use the TH command. A sample is shown here with a unit that has a serial number of 3:

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

The second level of addressing is the IP address. This is a 32-bit (or 4 byte) number that usually looks like this: 192.168.15.1. The IP address is constrained by each local network and must be assigned locally. Assigning an IP address to the DMC-52xx0 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-52xx0 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-52xx0'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 controllers are on the Ethernet network – use the serial numbers to differentiate them.

CAUTION	Be sure that there is only one BOOT-P or DHCP server running. If your network has DHCP or BOOT-P running, it may automatically assign an IP address to the DMC-52xx0 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 USB port. (Note: The IA command is only valid if DH0 is set). The IP address may be entered as a 4 byte number delimited by commas (industry standard uses periods) or a signed 32 bit number (e.g. IA 124,51,29,31 or IA 2083724575). Type in BN to save the IP address to the DMC-52xx0 non-volatile memory.

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

The third level of Ethernet addressing is the UDP or TCP port number. The Galil board does not require a specific port number. The port number is established by the client or master each time it connects to the DMC-52xx0 board. Typical port numbers for applications are:

Port 23: Telnet
Port 502: Modbus

Communicating with Multiple Devices

The DMC-52xx0 is capable of supporting multiple masters and slaves. The masters may be multiple PC's that send commands to the controller. The slaves are typically peripheral I/O devices that receive commands from the controller.

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

An Ethernet handle is a communication resource within a device. The DMC-52xx0 can have a maximum of 8 Ethernet handles open at any time. When using TCP/IP, each master or slave uses an individual Ethernet handle. In UDP/IP, one handle may be used for all the masters, but each slave uses one. (Pings and ARPs do not occupy handles.) If all 8 handles are in use and a 9th master tries to connect, it will be sent a "reset packet" that generates the appropriate error in its windows application.

Note: There are a number of ways to soft reset the controller. Hardware reset (push reset button or power down controller) and software resets (through Ethernet or USB by entering RS).

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

IHB=151,25,255,9<179>2 This will open handle #2 and connect to the IP address 151.25.255.9, port 179, using TCP/IP)

Which devices receive what information from the controller depends on a number of things. If a device queries the controller, it will receive the response unless it explicitly tells the controller to send it to another device. If the command that generates a response is part of a downloaded program, the response will route to whichever port is specified as the default (unless explicitly told to go to another port with the CF command). To designate a specific destination for the information, add {Ex} to the end of the command. (Ex. MG{EC}"Hello" will send the message "Hello" to handle #3. TP,,?{EF} will send the C 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 that the user chooses. Ping is used to check the communication between the device at a specific IP address and the host computer.

The DMC-52xx0 can communicate with a host computer through any application that can send TCP/IP or UDP/IP packets. A good example of this is Telnet, a utility that comes with most Windows systems.

Modbus

An additional protocol layer is available for speaking to I/O devices. Modbus is an RS-485 protocol that packages information in binary packets that are sent as part of a TCP/IP packet. In this protocol, each slave has a 1 byte slave address. The DMC-52xx0 can use a specific slave address or default to the handle number. The port number for Modbus is 502. The DMC-52xx0 can only fill the role of a Modbus master.

The Modbus protocol has a set of commands called function codes. The DMC-52xx0 supports the 10 major function codes:

Function Code	Definition
01	Read Coil Status (Read Bits)
02	Read Input Status (Read Bits)
03	Read Holding Registers (Read Words)
04	Read Input Registers (Read Words)
05	Force Single Coil (Write One Bit)

06	Preset Single Register (Write One Word)
07	Read Exception Status (Read Error Code)
15	Force Multiple Coils (Write Multiple Bits)
16	Preset Multiple Registers (Write Words)
17	Report Slave ID

The DMC-52xx0 provides three levels of Modbus communication. The first level allows the user to create a raw packet and receive raw data. It uses the MBh command with a function code of –1. The format of the command is

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

array[] is the array with the data

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

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

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

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

Where HandleNum is the handle number from 1 (A) to 8 (H). 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-52040 connected as a Modbus master to a RIO-47120 via Modbus. The DMC-52040 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 by DMC-52040)
   ```

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[]
   ```

   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-52040 connected as a Modbus master to a 3rd party PLC. The DMC-52040 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-52040 connected as a Modbus master to a hydraulic pump. The DMC-52040 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

Data Record

The DMC-520x0 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 QR command will return 4 bytes of header information and specific blocks of information as specified by the command arguments:

```
QR ABCDEFGHST
```

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

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

General Controller Information and Status

ADDR	TYPE	ITEM	ADDR	TYPE	ITEM
00	UB	1st Byte of Header	30-31	SW	Reserved
01	UB	2nd Byte of Header	32-33	SW	Reserved
02	UB	3rd Byte of Header	34-35	SW	Reserved
03	UB	4th Byte of Header	36-37	SW	Reserved
04-05	UW	sample number	38-39	SW	Reserved
6	UB	general input block 0 (inputs 1-8)	40	UB	EtherCAT Bank
07	UB	general input block 1 (inputs 9-16)	41	UB	Reserved
08	UB	general input block 2 (inputs 17-24)	42	UB	Ethernet Handle A Status
09	UB	general input block 3 (inputs 25-32)	43	UB	Ethernet Handle B Status
10	UB	general input block 4 (inputs 33-40)	44	UB	Ethernet Handle C Status
11	UB	general input block 5 (inputs 41-48)	45	UB	Ethernet Handle D Status
12	UB	general input block 6 (inputs 49-56)	46	UB	Ethernet Handle E Status

13	UB	general input block 7 (inputs 57-64)	47	UB	Ethernet Handle F Status
14	UB	general input block 8 (inputs 65-72)	48	UB	Ethernet Handle G Status
15	UB	general input block 9 (inputs 73-80)	49	UB	Ethernet Handle H Status
16	UB	general output block 0 (outputs 1-8)	50	UB	error code
17	UB	general output block 1 (outputs 9-16)	51	UB	thread status – see bit field map below
18	UB	general output block 2 (outputs 17-24)	52-55	UL	Amplifier Status
19	UB	general output block 3 (outputs 25-32)	56-59	UL	Segment Count for Contour Mode
20	UB	general output block 4 (outputs 33-40)	60-61	UW	Buffer space remaining – Contour Mode
21	UB	general output block 5 (outputs 41-48)	62-63	UW	segment count of coordinated move for S plane
22	UB	general output block 6 (outputs 49-56)	64-65	UW	coordinated move status for S plane – see bit field map below
23	UB	general output block 7 (outputs 57-64)	66-69	SL	distance traveled in coordinated move for S plane
24	UB	general output block 8 (outputs 65-72)	70-71	UW	Buffer space remaining – S Plane
25	UB	general output block 9 (outputs 73-80)	72-73	UW	segment count of coordinated move for T plane
26-27	SW	Reserved	74-75	UW	Coordinated move status for T plane – see bit field map below
28-29	SW	Reserved	76-79	SL	distance traveled in coordinated move for T plane

Axis Information

ADDR	TYPE	ITEM	ADDR	TYPE	ITEM
80-81	UW	Buffer space remaining – T Plane	226-227	UW	E axis status – see bit field map below
82-83	UW	A axis status – see bit field map below	228	UB	E axis switches – see bit field map below
84	UB	A axis switches – see bit field map below	229	UB	E axis stop code
85	UB	A axis stop code	230-233	SL	E axis reference position
86-89	SL	A axis reference position	234-237	SL	E axis motor position
90-93	SL	A axis motor position	238-241	SL	E axis position error
94-97	SL	A axis position error	242-245	SL	E axis auxiliary position
98-101	SL	A axis auxiliary position	246-249	SL	E axis velocity
102-105	SL	A axis velocity	250-253	SL	E axis torque
106-109	SL	A axis torque	254-255	SW or UW [1]	E axis analog input
110-111	SW or UW [1]	A axis analog input	256	UB	E Hall Input Status
112	UB	A Hall Input Status	257	UB	Reserved
113	UB	Reserved	258-261	SL	E User defined variable (ZE)
114-117	SL	A User defined variable (ZA)	262-263	UW	F axis status – see bit field map below
118-119	UW	B axis status – see bit field map below	264	UB	F axis switches – see bit field map below
120	UB	B axis switches – see bit field map below	265	UB	F axis stop code
121	UB	B axis stop code	266-269	SL	F axis reference position
122-125	SL	B axis reference position	270-273	SL	F axis motor position
126-129	SL	B axis motor position	274-277	SL	F axis position error
130-133	SL	B axis position error	278-281	SL	F axis auxiliary position
134-137	SL	B axis auxiliary position	282-285	SL	F axis velocity
138-141	SL	B axis velocity	286-289	SL	F axis torque
142-145	SL	B axis torque	290-291	SW or UW [1]	F axis analog input
146-147	SW or UW [1]	B axis analog input	292	UB	F Hall Input Status
148	UB	B Hall Input Status	293	UB	Reserved
149	UB	Reserved	294-297	SL	F User defined variable (ZF)
150-153	SL	B User defined variable (ZB)	298-299	UW	G axis status – see bit field map below
154-155	UW	C axis status – see bit field map below	300	UB	G axis switches – see bit field map below
156	UB	C axis switches – see bit field map below	301	UB	G axis stop code
157	UB	C axis stop code	302-305	SL	G axis reference position
158-161	SL	C axis reference position	306-309	SL	G axis motor position
162-165	SL	C axis motor position	310-313	SL	G axis position error
166-169	SL	C axis position error	314-317	SL	G axis auxiliary position
170-173	SL	C axis auxiliary position	318-321	SL	G axis velocity
174-177	SL	C axis velocity	322-325	SL	G axis torque
178-181	SL	C axis torque	326-327	SW or UW [1]	G axis analog input
182-183	SW or UW [1]	C axis analog input	328	UB	G Hall Input Status
184	UB	C Hall Input Status	329	UB	Reserved
185	UB	Reserved	330-333	SL	G User defined variable (ZG)
186-189	SL	C User defined variable (ZC)	334-335	UW	H axis status – see bit field map below
190-191	UW	D axis status – see bit field map below	336	UB	H axis switches – see bit field map below
192	UB	D axis switches – see bit field map below	337	UB	H axis stop code
193	UB	D axis stop code	338-341	SL	H axis reference position
194-197	SL	D axis reference position	342-345	SL	H axis motor position
198-201	SL	D axis motor position	346-349	SL	H axis position error
202-205	SL	D axis position error	350-353	SL	H axis auxiliary position
206-209	SL	D axis auxiliary position	354-357	SL	H axis velocity
210-213	SL	D axis velocity	358-361	SL	H axis torque
214-217	SL	D axis torque	362-363	SW or UW [1]	H axis analog input
218-219	SW or UW [1]	D axis analog input	364	UB	H Hall Input Status
220	UB	D Hall Input Status	365	UB	Reserved
221	UB	Reserved	366-369	SL	H User defined variable (ZH)
222-225	SL	D User defined variable (ZD)			

[1] Will be either a Signed Word or Unsigned Word depending upon AQ setting. See AQ in the Command Reference for more information.

Data Record Bit Field Maps

Header Information - Byte 0, 1 of Header:

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

Bytes 2, 3 of Header:

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

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

Note: The header information of the data records is formatted in little endian (reversed network byte order).

Thread Status (1 Byte)

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

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

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

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

Axis Status (1 Word)

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

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Negative Direction Move	Mode of Motion Contour	Motion is slewing	Motion is stopping due to ST of Limit Switch	Motion is making final deceleration	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	N/A	N/A	N/A	N/A	N/A	ELO Active (Axis E-H)	ELO Active (Axis A-D)

BIT 23	BIT 22	BIT 21	BIT 20	BIT 19	BIT 18	BIT 17	BIT 16
Peak Current H-axis	Peak Current G-axis	Peak Current F-axis	Peak Current E-axis	Peak Current D-axis	Peak Current C-axis	Peak Current B-axis	Peak current A-axis

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
Hall Error H-axis	Hall Error G-axis	Hall Error F-axis	Hall Error E-axis	Hall Error D-axis	Hall Error C-axis	Hall Error B-axis	Hall Error A-axis

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Under Voltage Axis (E-H)	Over Temp. Axis (E-H)	Over Voltage Axis (E-H)	Over Current Axis (E-H)	Under Voltage Axis (A-D)	Over Temp. Axis (A-D)	Over Voltage Axis (A-D)	Over Current Axis (A-D)

EtherCAT Bank (1 Bytes)

BIT 7	BIT 6	BIT 5	BIT 5	BIT 3	BIT 2	BIT 1	BIT 0
N/A	N/A	N/A	N/A	Bank 3 Axes 25-32	Bank 2 Axes 17-24	Bank 1, Axes 9-16	Bank 0, Axes 1-8

Notes Regarding Velocity and Torque Information

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

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

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-52xx0 provides over 100 commands for specifying motion and machine parameters. These easy to use commands are sent in ASCII and 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 communication ports for immediate execution by the DMC-52xx0, or an entire group of commands can be downloaded into the DMC-52xx0 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-52xx0 instruction set and syntax. A summary of commands as well as a complete listing of all DMC-52xx0 instructions is included in the Command Reference.

Command Syntax - ASCII

DMC-52xx0 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 carriage return is used to terminate the instruction for processing by the DMC-52xx0 command interpreter.

Note: If you are using a Galil terminal program, commands will not be processed until a carriage 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 carriage return command.

Note	All DMC commands are two-letters 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 carriage return completes the instruction. The space between PR and 4000 is optional.

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

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

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

Explicit Notation

The DMC-52xx0 provides an alternative method for specifying data. Here, data is specified individually using a single axis specifier such as A, B, C, or D. An equals sign is used to assign data to that axis. For example:

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

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

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

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

`SG 0`	Select bank
`BG ABCDEFGH`	Begin all axes on selected bank
`BG D`	Begin D only

Coordinated Motion with more than 1 axis

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

`BG S`	Begin coordinated sequence for S coordinate system
`BG TD`	Begin coordinated sequence, T coordinate system and D axis on selected bank

Bank Switching and Global Command Arguments

The DMC-52xx0 has 2, 4, 8, 16, and 32 axis configurations. To facilitate control of all the axes, the axes are organizes into banks of 8. For controllers with more then 8 axes, the Select Group (SG) command is used to change between banks when querying axis data or commanding motion. All axis-specific command will only affect the axis on the currently selected bank. The SH, MO, ST, and BG commands can affect all axes on the controller at once by using "!", see command reference for details. For example:

```
SH!                 Servo all axes across all banks
SG 0                Select bank 0
PRA=2000            Relative move on Axis A, bank 0
PRB=4000            Relative move on Axis B, bank 0
SG 1                Select bank 9
PRA=2000            Relative move on Axis A, bank 1
PRD=4000            Relative move on Axis D, bank 1
BG!                 Begin all axes
```

Coordinated motion is only supported within the same bank and cannot span multiple banks. An example is the controller can command two independent circles to be profiled using axis C, D, E, and G on bank 0 while Linear interpolation is running with axes A, B, C, D, E, F, and G on bank 1. This type of motion is supported because the 2 circles are on bank 0 and the linear interpolation is all on bank 1, keeping the coordinated motion on the same bank. An example of unsupported motion is profiling a circle using axes H on bank 0 and A on bank 1. Use axes G and H on bank 0 or A and B on bank 1 instead.

The BG! command will begin independent motion on all axes. To begin coordinated motion, select the bank and issue BGS or BGT. Coordinate systems S, T, and virtical axes M and N are bank specific.

The SG command can be changed from either local DMC code or from one of the communication ports. It is the responsibility of the designer to manage how the SG command is used within the application. Please contact Galil Applications for best practices.

Controller Response to DATA

The DMC-52xx0 returns a : for valid commands and a ? for invalid commands.

For example, if the command BG is sent in lower case, the DMC-52xx0 will return a ?.

```
:bg                 invalid command, lower case
?                   DMC-52xx0 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 located in the TC command in the Command Reference.

Interrogating the Controller

Interrogation Commands

The DMC-52xx0 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) commands. See the Command Reference. The interrogation commands are relevant to the bank currently selected.

Summary of Interrogation Commands

RP	Report Command Position
RL*	Report Latch
^R^V	Firmware Revision Information
SC	Stop Code
TB	Tell Status
TC	Tell Error Code
TE	Tell Error
TI	Tell Input
TP	Tell Position
TR	Trace
TS	Tell Switches
TV	Tell Velocity

*On supported drives

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

```
SG 0                    Select bank 0
TP A                    Tell position A on bank 0
0                       Controller Response
TP AB                   Tell position A and B on bank 0
0,0                     Controller Response
SG 1                    Select bank 1
RP BC                   Tell reference position B and C on bank 1
2000,4000               Controller Response
```

Interrogating Current Commanded Values

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

```
SG 0                    Select bank 0
PR ?,?,?,?              Request A,B,C,D values
PR ,?                   Request B value only
```

The controller can also be interrogated with operands.

Operands

Most DMC-52xx0 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          Set variable to A axis encoder position
bank=_SG        Set variable to current bank value
```

The DMC-52xx0 Command Reference denotes all commands which have an equivalent operand as "Operand Usage". Also, see description of operands in Operand Summary - Independent Axis on pg. 43.

Command Summary

For a complete command summary, see Command Reference manual.

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

Chapter 6 Programming Motion

Overview

The DMC-52xx0 provides several modes of motion, including independent positioning and jogging, coordinated motion, electronic cam motion, and electronic gearing. Each one of these modes is discussed in the following sections.

The DMC-52xx0 as several different axis variants including 2, 4, 8, 16, and 32 axis where the xx designates the axis count. For controllers with more then 8 axes, see Bank Switching and Global Command Arguments, pg38.

The example applications described below will give instruction to the appropriate mode of motion.

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

EXAMPLE APPLICATION	MODE OF MOTION	COMMANDS
Turn on/off motors, start, and stop motion on all axes	Bank Switching and Global Command Arguments:	SH!, MO!, BG!, ST!
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.	Vector Mode: Linear and Circular Interpolation Motion:	VM, VP, CR, VS,VR, VA, VD, VE
Third axis must remain tangent to 2-D motion path.	Coordinated motion with Tangent Motion:	VM, VP, CR, VS,VA,VD, TN, VE

Electronic gearing where slave axes are scaled to master axis which can move in both directions.	Electronic Gearing:	GA, GD, _GP, GR, GM (if gantry)
Master/slave where slave axes must follow a master.	Electronic Gearing with 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
Following a trajectory based on a master encoder position	Electronic Cam:	EA, EM, EP, ET, EB, EG, EQ
Motion smoothing while operating in independent axis positioning	Independent Axis Positioning:	IT
Motion smoothing while operating in vector or linear interpolation positioning	Linear Interpolation Mode:	IT
Gantry - two axes or more axes mechanically coupled	Example - Gantry Mode:	GR, GM

Independent Axis Positioning

In this mode, motion between the specified axes is independent. 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-52xx0 profiler generates the corresponding trapezoidal or triangular velocity profile and position trajectory. The controller determines a new command position along the trajectory every sample until the specified profile is complete. Motion is complete when the last position command is sent by the DMC-52xx0 profiler.

Note: The physical 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 to control one or more axes in a bank, or all axes on the controller with BG!.

SG 0	Select bank 0
BG AC	Begin axes A and C on current bank
BG	Begin all axes on the current bank
BG!	Begin all axes on controller
ST ABCD	Stop motion on axes A, B, C, and D on selected bank
ST!	Stop all axes on controller

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. Note that, 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 Motion (BG). 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 A,B,C,D	Specifies relative distance
PA A,B,C,D	Specifies absolute position
SP A,B,C,D	Specifies slew speed
AC A,B,C,D	Specifies acceleration rate
DC A,B,C,D	Specifies deceleration rate
BG ABCD	Starts motion
ST ABCD	Stops motion before end of move
IP A,B,C,D	Changes position target
IT A,B,C,D	Time constant for independent motion smoothing
AM ABCD	Trippoint for profiler complete
MC ABCD	Trippoint for “in position”

The DMC-52xx0 also allows use of single axis specifiers such as PRA=2000.

Operand Summary - Independent Axis

OPERAND	DESCRIPTION
_ACx	Return acceleration rate for the axis specified by ‘x’ and selected bank
_DCx	Return deceleration rate for the axis specified by ‘x’ and selected bank
_SPx	Returns the speed for the axis specified by ‘x’ and selected bank
_PAx	Returns current destination if ‘x’ axis is moving, otherwise returns the current commanded position if in a move.
_PRx	Returns current incremental distance specified for the ‘x’ axis

Example - Absolute Position Movement

```
SG 0                    Select bank 0
PA 10000,20000          Specify absolute A,B position
AC 1000000,1000000      Acceleration for A,B
DC 1000000,1000000      Deceleration for A,B
SP 50000,30000          Speeds for A,B
BG A,B                  Begin motion on selected bank
```

Example - Multiple Move Sequence

Required Motion Profiles:

A-Axis	500 counts	Position
	20000 count/sec	Speed
	500000 counts/sec^2	Acceleration
B-Axis	1000 counts	Position
	10000 count/sec	Speed
	500000 counts/sec^2	Acceleration
C-Axis	100 counts	Position
	5000 counts/sec	Speed
	500000 counts/sec^2	Acceleration

This example will specify a relative position movement on A, B and C axes. The movement on each axis will be separated by 20 msec. Figure 6.1 shows the velocity profiles for the A, B and C axis.

```
#A                          Begin Program
SG 0                        Select bank 0
PR 2000,500,100             Specify relative position movement of 2000, 500 and 100 counts for A, B, and C axes.
SP 20000,10000,5000         Specify speed of 20000, 10000, and 5000 counts / sec
AC 500000,500000,500000     Specify acceleration of 500000 counts / sec2 for all axes
DC 500000,500000,500000     Specify deceleration of 500000 counts / sec2 for all axes
BG A                        Begin motion on the A axis
WT 20                       Wait 20 msec
BG B                        Begin motion on the B axis
WT 20                       Wait 20 msec
BG C                        Begin motion on C axis
EN                          End Program
```

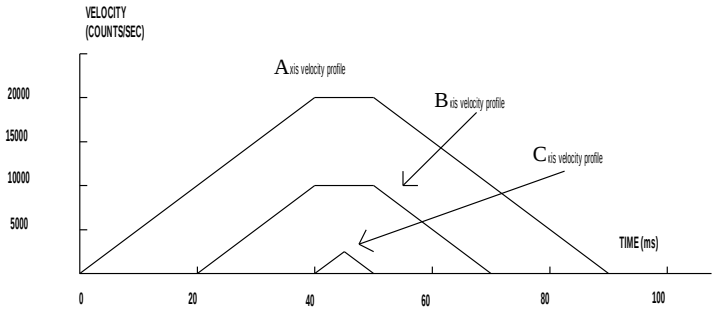

Figure 6.1: Velocity Profiles of ABC

Notes on Figure 6.1: The A and B axis have a 'trapezoidal' velocity profile, while the C axis has a 'triangular' velocity profile. The A and B axes accelerate to the specified speed, move at this constant speed, and then decelerate such that the final position agrees with the command position, PR. The C axis accelerates, but before the specified speed is achieved, must begin deceleration such that the axis will stop at the commanded position. All 3 axes have the same acceleration and deceleration rate, so the slope of the rising and falling edges of all 3 velocity profiles are the same.

Independent Jogging

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

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

Note that the controller operates as a closed-loop position controller while in the jog mode. The DMC-52xx0 converts the velocity profile into a position trajectory and a new position target is generated every sample period.

Command Summary - Jogging

COMMAND	DESCRIPTION
AC A,B,C,D	Specifies acceleration rate
BG ABCD	Begins motion
DC A,B,C,D	Specifies deceleration rate
IP A,B,C,D	Increments position instantly
IT A,B,C,D	Time constant for independent motion smoothing
JG ±A,B,C,D	Specifies jog speed and direction
ST ABCD	Stops motion

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

Operand Summary - Independent Axis

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

Example - Jog in X only

Jog A-axis motor at 50000 count/s. After A motor is at its jog speed, begin jogging C in reverse direction at 25000 count/s.

```
#a
SG 0                    Select bank 0
AC 20000,,20000         Specify A,C acceleration of 20000 counts / sec2
DC 20000,,20000         Specify A,C deceleration of 20000 counts / sec2
JG 50000,,-25000        Specify jog speed and direction for A and C axis
BG A                    Begin A motion
AS A                    Wait until A is at speed
BG C                    Begin C 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 V input the speed must be 50000 counts/sec.

```
#joy                    Label
SG 0                    Select bank 0
JG0                     Set in Jog Mode
BGA                     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 the controller updates the position information at the rate of 1msec. The controller generates a profiled point every other sample, and linearly interpolates one sample between each profiled point. Some examples of applications that may use this mode are satellite tracking, missile tracking, random pattern polishing of mirrors or lenses, or any application that requires the ability to change the endpoint without completing the previous move.

The PA command is typically used to command an axis or multiple axes to a specific absolute position. For some applications such as tracking an object, the controller must proceed towards a target and have the ability to change the target during the move. In a tracking application, this could occur at any time during the move or at regularly scheduled intervals. For example if a robot was designed to follow a moving object at a specified distance and the path of the object wasn't known the robot would be required to constantly monitor the motion of the object that it was following. To remain within a specified distance it would also need to constantly update the position target. Galil motion controllers support this type of motion with the position tracking mode. This mode allows scheduled or random updates to the current position target on the fly. Based on the new target the controller either continues in the direction it is heading, changes the direction it is moving, or decelerates 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. Also In contour 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 demonstrates the possible motions that may be commanded by the controller in position tracking mode. In this example, there is a host program that will generate the absolute position targets. The absolute target is determined based on the current information the host program has gathered on the object that it is tracking. The position tracking mode does allow for all of the axes on the controller to be in this mode, but for the sake of discussion, it is assumed that the robot is tracking only on the A-axis.

The controller must be placed in the position tracking mode to allow absolute position changes on the fly. This is performed with the PT command. To place the A-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 uses.

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 is set to 150,000 counts/sec^2 and the velocity is set to 50,000 counts/sec. The command sequence to perform this is listed below.

```
#ex1
SG0;'          Select bank 0
PT 1;'         Place the A axis in Position tracking mode
```

```
AC 150000;'    Set the A axis acceleration to 150000 counts/sec²
DC 150000;'    Set the A axis deceleration to 150000 counts/sec²
SP 50000;'     Set the A axis speed to 50000 counts/sec
PA 5000;'      Command the A axis to absolute position 5000 encoder counts
EN
```

The output from this code can be seen in Figure 6.2, a screen capture from the GalilTools scope.

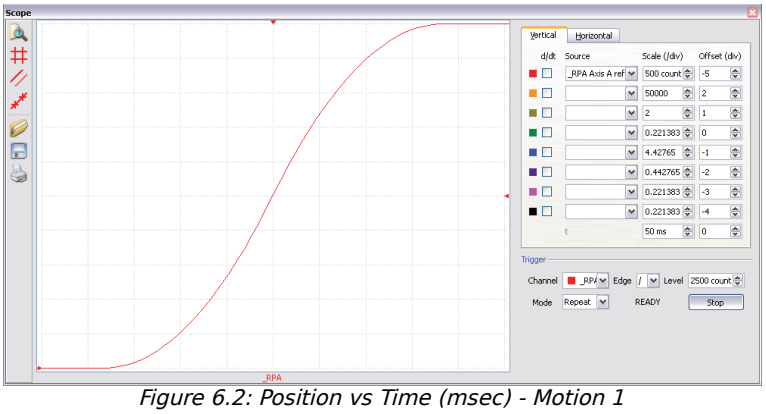

Figure 6.2: Position vs Time (msec) - Motion 1

Example - Motion 2:

The previous example showed the plot if the motion continued all the way to 5000. Partway through the motion, the object that was being tracked changed direction, so the host program determined that the actual target position should now be 2000 counts. The position target was modified when the robot was located at a position of 4200 counts (Figure 6.3). 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 up to the value set with SP in the opposite direction traveling to the new specified absolute position. In Figure 6.3 the velocity profile is triangular because the controller doesn't have sufficient time to reach the set speed of 50000 counts/sec before it is commanded to change direction.

The below code is used to simulate this scenario:

```
#ex2
SG 0;'         Select bank 0
PT 1;'         Place the A axis in Position tracking mode
AC 150000;'    Set the A axis acceleration to 150000 counts/sec²
DC 150000;'    Set the A axis deceleration to 150000 counts/sec²
SP 50000;'     Set the A axis speed to 50000 counts/sec
PA 5000;'      Command the A axis to abs position 5000 encoder counts
MF 4200
PA 2000;'      Change end point position to position 2000
EN
```

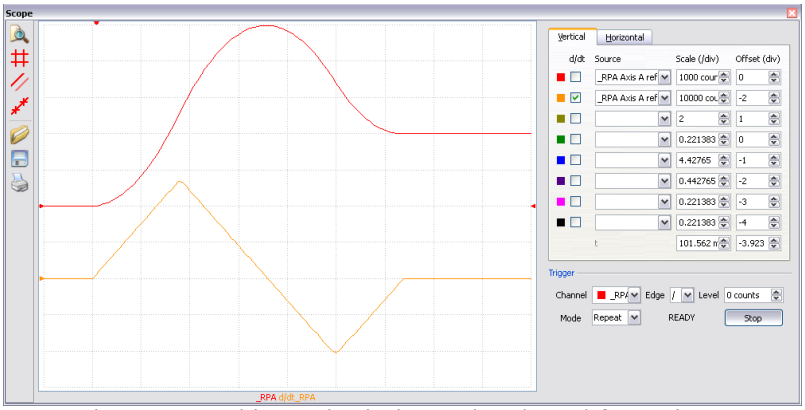

Figure 6.3: Position and Velocity vs Time (msec) for Motion 2

Example - Motion 3:

In this example, the host program commands the controller to begin motion towards position 5000, changes the target to -2000, and then changes it again to 8000. Below is the code that is used to simulate this scenario:

```
#ex3
SG 0;'      Select bank 0
PT 1;'      Place the A axis in Position tracking mode
AC 150000;' Set the A axis acceleration to 150000 counts/sec²
DC 150000;' Set the A axis deceleration to 150000 counts/sec²
SP 50000;'  Set the A axis speed to 50000 counts/sec
PA 5000;'   Command the A 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 6.5 demonstrates the use of motion smoothing (IT) on the velocity profile in this mode. The jerk in the system is also affected by the values set for AC and DC.

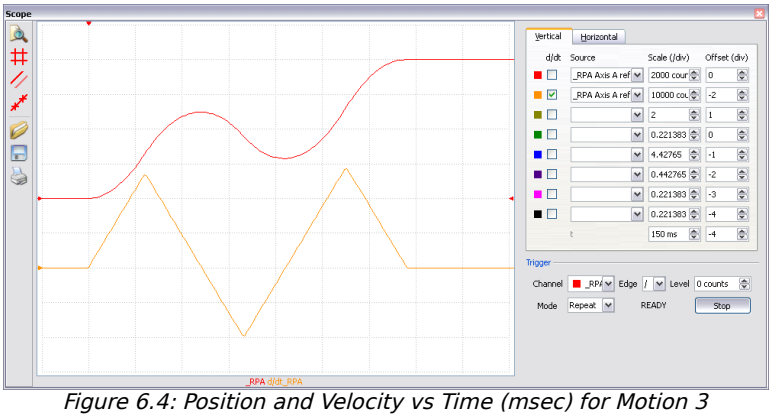

Figure 6.4: Position and Velocity vs Time (msec) for Motion 3

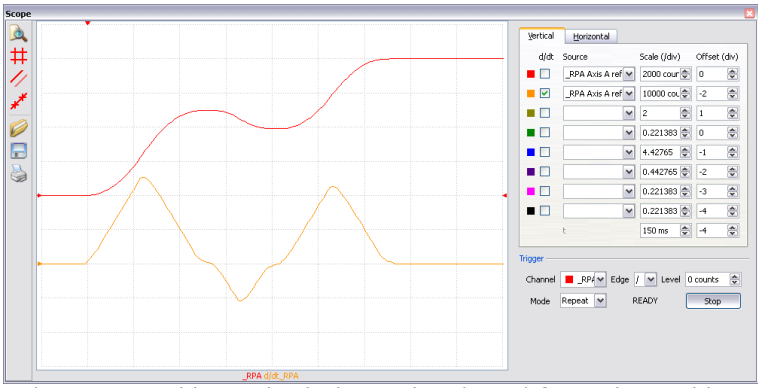

Figure 6.5: Position and Velocity vs Time (msec) for Motion 3 with IT 0.1

Note the controller treats the point where the velocity passes through zero as the end of one move, and the beginning of another move. Motion smoothing (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 MF, MR, or AP be used, as they involve motion in a specified direction, or the passing of a specific absolute position.

Command Summary – Position Tracking Mode

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

Linear Interpolation Mode

The DMC-52xx0 provides a linear interpolation mode for 2 or more axes on the same bank. In linear interpolation mode, motion between the axes is coordinated to maintain the prescribed vector speed, acceleration, and deceleration along the specified path. The motion path is described in terms of incremental distances for each axis. An unlimited number of incremental segments may be given in a continuous move sequence, making the linear interpolation mode ideal for following a piece-wise linear path. There is no limit to the total move length.

The LM command selects the Linear Interpolation mode and axes for interpolation. For example, LM BC selects only the B and C axes for linear interpolation for the currently selected bank.

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

Specifying Linear Segments

The command LI x,y,z,w or LI a,b,c,d,e,f,g,h specifies the incremental move distance for each axis. This means motion is prescribed with respect to the current axis position. Up to 511 incremental move segments may be given prior to the Begin Sequence (BGS) command for a single bank. There are separate LI buffers for each bank, if there are multiple banks. 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 ST 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-52xx0 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 end or last segment. There will be no controlled deceleration. LM? or _LM returns the available spaces for LI segments that can be sent to the buffer for the currently selected bank. 511 returned means the buffer is empty and 511 LI segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional LI segments can be sent.

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

Additional Commands

The commands VS n, VA n, and VD n are used to specify the vector speed, acceleration, and deceleration. The DMC-52xx0 computes the vector speed based on the axes specified in the LM mode. For example, LM ABC designates linear interpolation for the A, B, and C axes. The vector speed for this example would be computed using the equation:

VS2=AS2+BS2+CS2, where AS, BS and CS are the speed of the A, B, and C axes.

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

Motion smoothing (IT) is used to set the S-curve smoothing constant for axes in the coordinated moves. The command AV is the ‘After Vector’ trippoint, which pauses program execution until the vector distance of n has been reached.

An Example of Linear Interpolation Motion:

```
#lmove          Label
SG 0            Select bank 0
DP 0,0          Define position of A and B axes to be 0
LMAB            Define linear mode between A and B axes.
LI 5000,0       Specify first linear segment
LI 0,5000       Specify second linear segment
LE              End linear segments
VS 4000         Specify vector speed
BGS             Begin motion sequence
AV 4000         Set trippoint to wait until vector distance of 4000 is reached
VS 1000         Change vector speed
AV 5000         Set trippoint to wait until vector distance of 5000 is reached
VS 4000         Change vector speed
EN              Program end
```

In this example, the AB system is required to perform a 90□ turn. In order to slow the speed around the corner, the AV 4000 trippoint is used, which slows the speed to 1000 counts/s. Once the motors reach the corner, the speed is increased back to 4000 counts/s.

Specifying Vector Speed for Each Segment

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

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

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

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

Note, however, that the controller works with one '> m' command at a time. As a consequence, one '>m' 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 5000 at a different point.

Example:

```
#alt                            Label for alternative program
SG 0                             Select bank 0
DP 0,0                          Define Position of A and B axis to be 0
LMAB                            Define linear mode between A and B axes.
LI 4000,0 <4000 >1000           Specify first linear segment with a vector speed of 4000 and end
                                speed 1000
LI 1000,1000 < 4000 >1000       Specify second linear segment with a vector speed of 4000 and end
                                speed 1000
LI 0,5000 < 4000 >1000          Specify third linear segment with a vector speed of 4000 and end
                                speed 1000
LE                              End linear segments
BGS                             Begin motion sequence
EN                              Program end
```

Changing Feed Rate:

The command VR n allows the feed rate, VS, to be scaled between 0 and 10 with a resolution of .0001. This command takes effect immediately and causes VS to be scaled. VR also applies

when the vector speed is specified with the '<' operator. This is a useful feature for feed rate override. For example, VR .5 results in the specification VS 2000 to be divided in half.

Command Summary - Linear Interpolation

COMMAND	DESCRIPTION
LM ABCDEFGH	Specify axes for linear interpolation
LM?	Returns number of available spaces for linear segments in DMC-52xx0 sequence buffer. Zero means buffer full. 511 means buffer empty.
LI A,B,C,D,E,F,G,H < n	Specify incremental distances relative to current position, and assign vector speed n.
VS n	Specify vector speed
VA n	Specify vector acceleration
VD n	Specify vector deceleration
VR n	Specify the vector speed ratio
BGS	Begin Linear Sequence
CS	Clear sequence
LE	Linear End- Required at end of LI command sequence
LE?	Returns the length of the vector (resets after 2147483647)
AMS	Trippoint for After Sequence complete
AV n	Trippoint for After Relative Vector distance, n
IT	S curve smoothing constant for vector moves

Operand Summary - Linear Interpolation

OPERAND	DESCRIPTION
_AV	Return distance traveled
_CS	Segment counter - returns number of the segment in the sequence, starting at zero.
_LE	Returns length of vector (resets after 2147483647)
_LM	Returns number of available spaces for linear segments in DMC-52xx0 sequence buffer. Zero means buffer full. 511 means buffer empty.
_VPm	Return the absolute coordinate of the last data point along the trajectory. (m= A,B,C,D,E,F,G or H)

To illustrate the ability to interrogate the motion status, consider the first motion segment of the example where the A axis moves toward position A=5000. Suppose that when A=3000, the controller is interrogated using the command 'MG _AV'. The returned value will be 3000. The value of _CS, _VPA and _VPB will be zero.

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

Example - Linear Move

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

```
SG 0                    Select bank 0
LM CD                   Specify axes for linear interpolation
LI,,40000,30000         Specify CD distances
LE                      Specify end move
VS 100000               Specify vector speed
VA 1000000              Specify vector acceleration
VD 1000000              Specify vector deceleration
BGS                     Begin sequence
```

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

The result is shown in Figure 6.6: Linear Interpolation.

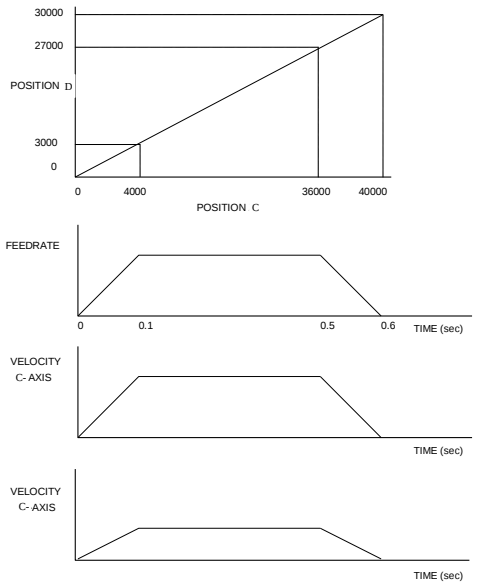

Figure 6.6: Linear Interpolation

Example - Multiple Moves

This example makes a coordinated linear move in the AB plane. The arrays VA[] and VB[] are used to store 750 incremental distances which are filled by the program #load.

```
#load                          Load Program
SG 0                           Select bank 0
DM VX [750],VY [750]           Define Array
count=0                        Initialize Counter
N=0                            Initialize position increment
#loop                          Loop label
VA [COUNT]=N                   Fill Array VA
VB [COUNT]=N                   Fill Array VB
N=N+10                         Increment position
count=count+1                  Increment counter
JP #loop,count<750             Loop if array not full
#a                             Label
LM AB                          Specify linear mode for AB
count=0                        Initialize array counter
#loop2;JP#loop2,_LM=0          If sequence buffer full, wait
JS#c,count=500                 Begin motion on 500th segment
LI VX[count],VY[count]         Specify linear segment
count=count+1                  Increment array counter
JP #loop2,count<750            Repeat until array done
LE                             End Linear Move
AMS                            After Move sequence done
MG "DONE"                      Send Message
EN                             End program
#c;BGS;EN                      Begin Motion Subroutine
```

Vector Mode: Linear and Circular Interpolation Motion

The DMC-52xx0 allows a long 2-D path consisting of linear and arc segments to be prescribed. Motion along the path is continuous at the prescribed vector speed even at transitions between linear and circular segments. The DMC-52xx0 performs all the complex computations of linear and circular interpolation, freeing the host PC from this time-intensive task.

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

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

Specifying the Coordinate Plane

The DMC-52xx0 allows for two separate sets of coordinate axes for linear interpolation mode or vector mode on each bank. These two sets are identified by the letters S and T.

To specify vector commands the coordinate plane must first be identified. This is done by issuing the command CAS to identify the S plane or CAT to identify the T plane. All vector commands will be applied to the active coordinate system until changed with the CA command. The SG command does not change the CA command but it will have an effect on the which axes are performing the motion. Example, if SG0 0; CA T is issued the next vector point will be assigned to the T plane on the bank 0. If, at this point SG 1 is issued, the next vector points will be assigned to the T plane on bank 1.

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 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 as the reference position for all movements in a sequence. Note: This 'local' definition of zero does not effect the absolute coordinate system or subsequent coordinated motion sequences.

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

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

The Clear Sequence (CS) command can be used to remove previous VP and CR commands which were stored in the buffer prior to the start of the motion. To stop the motion, use the instructions ST 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-52xx0 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 for the selected bank. 511 returned means the buffer is empty and 511 segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional segments can be sent at PC bus speeds.

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

Additional commands

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

The s-curve smoothing constant (IT) is used with coordinated motion.

Specifying Vector Speed for Each Segment:

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

VP x,y < n >m

CR r, ɵ,δ< n >m

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

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

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

Changing Feed Rate:

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

Compensating for Differences in Encoder Resolution:

By default, the DMC-52xx0 uses a scale factor of 1:1 for the encoder resolution when used in vector mode. If this is not the case, the command, ES can be used to scale the encoder counts. The ES command accepts two arguments which represent the number of counts for the two

encoders used for vector motion. The smaller ratio of the two numbers will be multiplied by the higher resolution encoder. For more information, see ES command in the Command Reference.

Many EtherCAT drives allow the encoder to be scaled. This scaling setting will change the position reported back from the drive and change actual motion accordingly. See EtherCAT drive manufacturer documentation for details.

Trippoints:

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

Tangent Motion:

Several applications, such as cutting, require a third axis (i.e. a knife blade), to remain tangent to the coordinated motion path. To handle these applications, the DMC-52xx0 allows one axis within the same bank to be specified as the tangent axis. The VM command provides parameter specifications for describing the coordinated axes and the tangent axis.

VM m,n,p	m, n specifies coordinated axes p specifies tangent axis such as A,B,C,D p=n turns off tangent axis

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

Example:

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

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

Command Summary - Coordinated Motion Sequence

COMMAND	DESCRIPTION

VM m,n	Specifies the axes for the planar motion where m and n represent the planar axes and p is the tangent axis.
VP m,n	Return coordinate of last point, where m=A, B, C, D, E, F, G, or H.
CR r,θ,δ<n>m	Specifies arc segment where r is the radius, θ is the starting angle and δ is the travel angle. Positive direction is CCW.
VS s,t	Specify vector speed or feed rate of sequence.
VA s,t	Specify vector acceleration along the sequence.
VD s,t	Specify vector deceleration along the sequence.
VR s,t	Specify vector speed ratio
BGST	Begin motion sequence, S or T
CSST	Clear sequence, S or T
AV s,t	Trippoint for After Relative Vector distance.
AMST	Holds execution of next command until Motion Sequence is complete.
TN m,n	Tangent scale and offset.
ES m,n	Ellipse scale factor.
IT s,t	S curve smoothing constant for coordinated moves
LM?	Return number of available spaces for linear and circular segments in DMC-52xx0 sequence buffer. Zero means buffer is full. 511 means buffer is empty.
CAS or CAT	Specifies which coordinate system is to be active (S or T)

Operand Summary - Coordinated Motion Sequence

OPERAND	DESCRIPTION
_VPM	The absolute coordinate of the axes at the last intersection along the sequence.
_AV	Distance traveled.
_LM	Number of available spaces for linear and circular segments in DMC-52xx0 sequence buffer. Zero means buffer is full. 511 means buffer is empty.
_CS	Segment counter - Number of the segment in the sequence, starting at zero.
_VE	Vector length of coordinated move sequence.

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

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

Example:

Traverse the path shown in Figure 6.7. Feed rate is 20000 counts/sec. Plane of motion is AB

```
SG 0                    Select bank 0
VM AB                   Specify motion plane
VS 20000                Specify vector speed
VA 1000000              Specify vector acceleration
VD 1000000              Specify vector deceleration
VP -4000,0              Segment AB
CR 1500,270,-180        Segment BC
VP 0,3000               Segment CD
CR 1500,90,-180         Segment DA
VE                      End of sequence
BG S                    Begin Sequence
```

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

The value of _AV is 2000

The value of _CS is 0

_VPX and _VPY contain the absolute coordinate of the point A

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

The value of _AV is 4000+1500$\frac{\pi}{2}$+2000=10,712

The value of _CS is 2

_VPX, _VPY contain the coordinates of the point C

Figure 6.7: The Required Path

Vector Mode - Mathematical Analysis

The terms of coordinated motion are best explained in terms of the vector motion. The vector velocity, Vs, which is also known as the feed rate, is the vector sum of the velocities along the A and B axes, Va and Vb.

$$Vs = \sqrt{Vx^2 + Vy^2}$$

The vector distance is the integral of Vs, or the total distance traveled along the path. To illustrate this further, suppose that a string was placed along the path in the A-B plane. The length of that string represents the distance traveled by the vector motion.

The vector velocity is specified independently of the path to allow continuous motion. The path is specified as a collection of segments. For the purpose of specifying the path, define a special A-B coordinate system whose origin is the starting point of the sequence. Each linear segment is specified by the A-B coordinate of the final point expressed in units of resolution, and each circular arc is defined by the arc radius, the starting angle, and the angular width of the arc. The zero angle corresponds to the positive direction of the A-axis and the CCW direction of rotation is positive. Angles are expressed in degrees, and the resolution is 1/256th of a degree. For example, the path shown in Figure 6.4 is specified by the instructions:

```
VP 0,10000
CR 10000, 180, -90
VP 20000, 20000
```

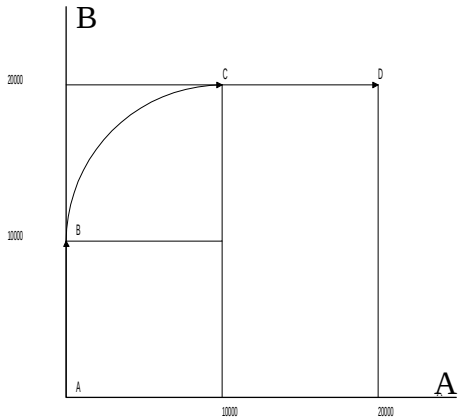

Figure A.8: A-B Motion Path

The first line describes the straight line vector segment between points A and B. The next segment is a circular arc, which starts at an angle of 180° and traverses -90°. Finally, the third line describes the linear segment between points C and D. Note that the total length of the motion consists of the segments:

A-B	Linear	10000 units
B-C	Circular	$\frac{R\lvert\Delta\theta\rvert 2\pi}{360}$ = 15708
C-D	Linear	10000
	Total	35708 counts

In general, the length of each linear segment is

$$L_k = \sqrt{Ak^2 + Bk^2}$$

Where Xk and Yk are the changes in A and B positions along the linear segment. The length of the circular arc is

$$L_k = R_k\lvert\Delta\Theta_k\rvert 2\pi/360$$

The total travel distance is given by

$$D = \sum_{k=1}^{n} L_k$$

The velocity profile may be specified independently in terms of the vector velocity and acceleration.

For example, the velocity profile corresponding to the path of Figure 6.4 may be specified in terms of the vector speed and acceleration.

```
VS 100000
VA 2000000
```

The resulting vector velocity is shown in Figure A.9.

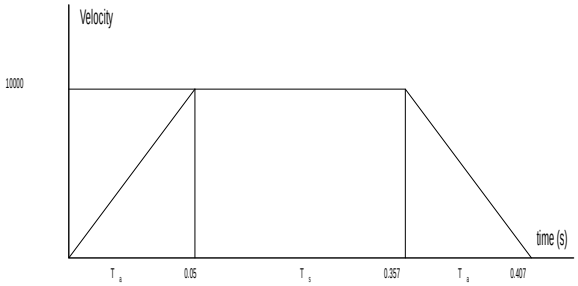

Figure A.9: Vector Velocity Profile

The acceleration time, Ta, is given by

$$T_a = \frac{VS}{VA} = \frac{100000}{2000000} = 0.05s$$

The slew time, Ts, is given by

$$T_s = \frac{D}{VS} - T_a = \frac{35708}{100000} - 0.05 = 0.307s$$

The total motion time, Tt, is given by:

$$T_t = \frac{D}{VS} + T_a = 0.407s$$

The velocities along the A and B axes are such that the direction of motion follows the specified path, yet the vector velocity fits the vector speed and acceleration requirements.

For example, the velocities along the A and B axes for the path shown in Figure 6.4 are given in Figure A.10.

Figure A.10 shows the vector velocity. It also indicates the position point along the path starting at A and ending at D. Between the points A and B, the motion is along the B-axis. Therefore,

Vb = Vs

and

Va = 0

Between the points B and C, the velocities vary gradually and finally, between the points C and D, the motion is in the A-axis direction.

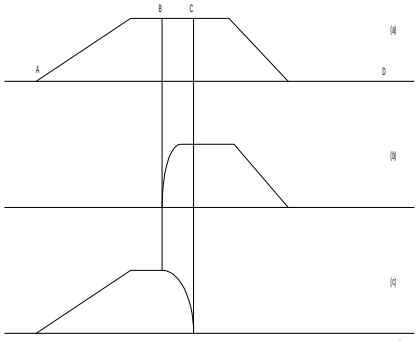

Figure A.10: Vector Axes Velocities

Electronic Gearing

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

The command `GA n,n,n,n,n,n,n,n` specifies the master axes where `n` is the master for that axis. `GR a,b,c,d,e,f,g,h` specifies the gear ratios for the slaves where the ratio may be a number between ±127.9999 with a fractional resolution of .0001. There are two modes: standard gearing and gantry mode. The gantry mode (enabled with the command `GM`) allows the gearing to stay enabled even if a limit is hit or an `ST` command is issued. `GR 0,0,0,0,0,0,0,0` turns off gearing in both modes.

The command `GM a,b,c,d,e,f,g,h` selects the axes to be controlled under the gantry mode. The parameter `1` enables gantry mode and `0` disables it.

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

When the master axis is driven by the controller in the jog mode or an independent motion mode, it is possible to define the master as the command position of that axis, rather than the actual position. The designation of the commanded position master is by the letter, C. For example, GA CY indicates that the gearing master for the A axis is the commanded position of B axis.

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

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

Ramped Gearing

In some applications, especially when the master is traveling at high speeds, it is desirable to have the gear ratio ramp gradually to minimize large changes in velocity on the slave axis when the gearing is engaged. For example if the master axis is already traveling at 500,000 counts/sec and the slave will be geared at a ratio of 1:1 when the gearing is engaged, the slave will instantly develop following error which will cause the controller to command maximum current to the motor. This can be a large shock to the system. For many applications it is acceptable to slowly ramp the engagement of gearing over a greater time frame. Galil allows the user to specify an interval of the master axis over which the gearing will be engaged. For example, the same master A-axis in this case travels at 500,000 counts/sec, and the gear ratio is 1:1, but the gearing is slowly engaged over 30,000 counts of the master axis, greatly diminishing the initial shock to the slave axis. Figure 6.11 below shows the velocity vs. time profile for instantaneous gearing. Figure 6.12 shows the velocity vs. time profile for the gradual gearing engagement.

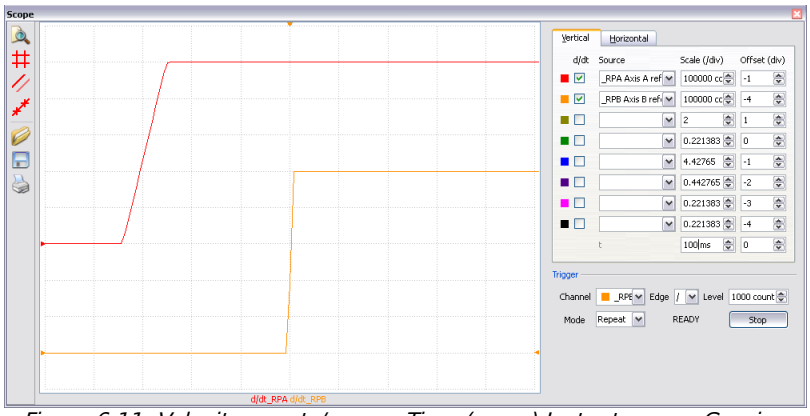

Figure 6.11: Velocity counts/sec vs. Time (msec) Instantaneous Gearing Engagement

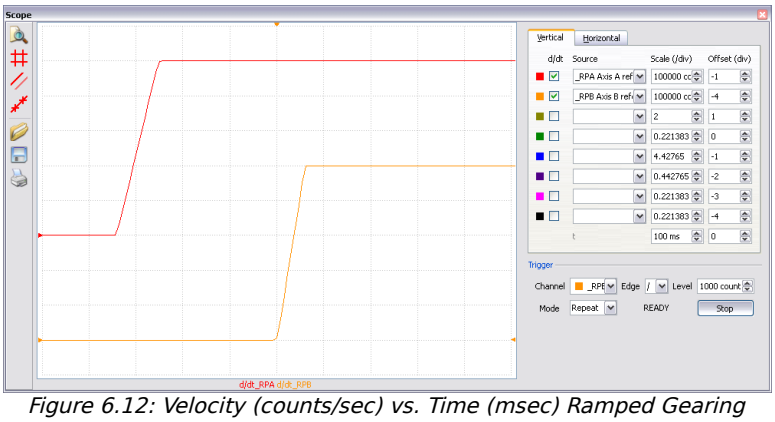

Figure 6.12: Velocity (counts/sec) vs. Time (msec) Ramped Gearing

The slave axis for each figure is shown on the bottom portion of the figure (red); the master axis is shown on the top portion (yellow). The shock to the slave axis will be significantly less in Figure 6.12 than in Figure 6.11. The ramped gearing does have one consequence: there isn't a true synchronization of the two axes until the gearing ramp is complete. The slave will lag behind the true ratio during the ramp period. If exact position synchronization is required from the point gearing is initiated, then the position must be commanded in addition to the gearing. The controller keeps track of this position phase lag with the `_GP` operand. The following example will demonstrate how the command is used.

Example – Electronic Gearing Over a Specified Interval

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

Solution: Use a DMC-52040 controller where the C-axis is the master and A and B are the geared axes. We will implement the gearing change over 6000 counts (3 revolutions) of the master axis.

```
SG0                 Select bank 0
MO C                Turn C off, for external master
GA C, C             Specify C as the master axis for both A and B.
GD 6000,6000        Specify ramped gearing over 6000 counts of the master axis.
GR 1.132,-.045      Specify gear ratios
```

Question: What is the effect of the ramped gearing?

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

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

Command Summary - Electronic Gearing

COMMAND	DESCRIPTION
`GA n`	Specifies master axes for gearing where: `n = X,Y,Z,W` or `A,B,C,D,E,F,G,H` for main encoder as master

	n = CX,CY,CZ,CW or CA,CB,CC,CD,CE,CF,CG,CH for commanded position.
	n = DX,DY,DZ,DW or DA,DB,DC,DD,DE,DF,DG,DH for auxiliary encoders
	n = S or T for gearing to coordinated motion.
GD a,b,c,d,e,f,g,h	Sets the distance the master will travel for the gearing change to take full effect.
_GPn	This operand keeps track of the difference between the theoretical distance traveled if gearing changes took effect immediately, and the distance traveled since gearing changes take effect over a specified interval.
GR a,b,c,d,e,f,g,h	Sets gear ratio for slave axes. 0 disables electronic gearing for specified axis.
GM a,b,c,d,e,f,g,h	X = 1 sets gantry mode, 0 disables gantry mode
MR n,n,n,n,n,n,n,n	Trippoint for reverse motion past specified value. Only one field may be used.
MF n,n,n,n,n,n,n,n	Trippoint for forward motion past specified value. Only one field may be used.

Example - Simple Master Slave

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

```
SG 0                Select bank 0
GA C,,C,C           Specify master axes as C
GR 5,,-.5,10        Set gear ratios
PR ,10000           Specify C position
SP ,100000          Specify C speed
BGC                 Begin motion
```

Example - Electronic Gearing

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

Solution: Use a DMC-52020 controller, where the C-axis is the master and A and B are the geared axes.

```
MO C                Turn C off, for external master
GA C, C             Specify C as the master axis for both A and B.
GR 1.132,-.045      Specify gear ratios
```

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

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

Example - Gantry Mode

In applications where both the master and the follower are controlled by the DMC-52xx0 controller, it may be desired to synchronize the follower with the commanded position of the master, rather than the actual position. This eliminates the coupling between the axes which may lead to oscillations.

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

```
SG 0              Select bank 0
GA, CA            Specify the commanded position of A as master for B.
GR,1              Set gear ratio for B as 1:1
GM,1              Set gantry mode
PR 3000           Command A motion
BG A              Start motion on A axis
```

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

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

Under these conditions, this IP command is equivalent to:

```
PR,10             Specify position relative movement of 10 on B axis
BGB               Begin motion on B axis
```

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

Example - Synchronize two conveyor belts with trapezoidal velocity correction

```
SG 0              Select bank 0
GA,A              Define A as the master axis for B.
GR,2              Set gear ratio 2:1 for B
PR,300            Specify correction distance
SP,5000           Specify correction speed
AC,100000         Specify correction acceleration
DC,100000         Specify correction deceleration
BGB               Start correction
```

Electronic Cam

The electronic cam is a motion control mode which enables the periodic synchronization of several axes of motion. Up to 7 axes on the same bank can be slaves to one master axis. There can only be one ECAM master on each bank.

The electronic cam is a more general type of electronic gearing which allows a table-based relationship between the axes. It allows synchronization between all the controller axes within each bank. For example, the DMC-52080 controllers may have one master and up to seven slaves on a single bank.

To illustrate the procedure of setting the cam mode, consider the cam relationship for the slave axis B, when the master is A. A graphical relationship is shown in Figure 6.13.

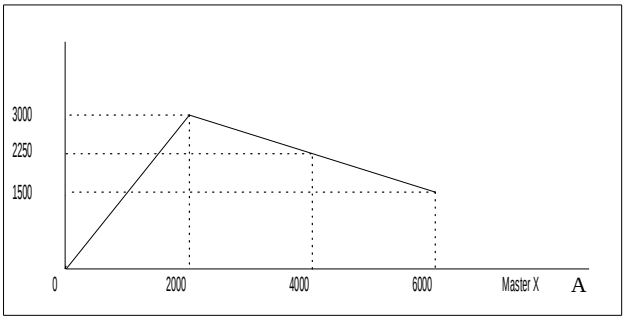

Figure 6.13: Electronic Cam Example

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 p where p = A,B,C,D,E,F,G,H

p is the selected master axis

For the given example, since the master is A, we specify EA A

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

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

```
EM n,n,n,n,n,n,n,n
```

where n specifies the cycle of the master and the total change of the slaves over one cycle.

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

```
EM 6000,1500
```

Step 3. Specify the master interval and starting point.

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

EP n_0,n_1

where n_0 is the interval width in counts, and n_1 is the starting point in counts.

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

```
EP 2000,0
```

Step 4. Specify the slave positions.

Next, we specify the slave positions with the instruction

ET[n_0]=n,n,n,n,n,n,n,n

where n_0 indicates the order of the point.

The value, n_0, starts at zero and may go up to 256. The parameters n indicate the corresponding slave position. For this example, the table may be specified by

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

This specifies the ECAM table.

Step 5. Enable the ECAM

To enable the ECAM mode, use the command

`EB n`

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

Step 6. Engage the slave motion

To engage the slave motion, use the instruction

`EG n,n,n,n,n,n,n,n`

where n is the master positions at which the corresponding slaves must be engaged.

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

Step 7. Disengage the slave motion

To disengage the cam, use the command

`EQ n,n,n,n,n,n,n,n`

where n is the master positions at which the corresponding slave axes are disengaged.

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

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

the equation:

$B = 0.5 * A + 100 \sin(0.18 * A)$

where A 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 setup.

The instruction `EA A` defines A as the master axis. The cycle of the master is 2000. Over that cycle, B varies by 1000. This leads to the instruction `EM 2000,1000`.

Suppose we want to define a table with 100 segments. This implies increments of 20 counts each. If the master points are to start at zero, the required instruction is `EP 20,0`.

The following routine computes the table points. As the phase equals 0.18A and A varies in increments of 20, the phase varies by increments of 3.6□. The program then computes the values of B according to the equation and assigns the values to the table with the instruction `ET[N] = ,B`.

INSTRUCTION	INTERPRETATION
`#setup`	Label
`SG 0`	Select bank 0
`EAA`	Select A as master
`EM 2000,1000`	Cam cycles
`EP 20,0`	Master position increments
`n = 0`	Index
`#loop`	Loop to construct table from equation
`p = n~3.6`	Note 3.6 = 0.18 * 20
`s = @SIN [p]*100`	Define sine position
`b = n*10+s`	Define slave position

```
ET [n] =, b          Define table
n = n+1
JP #loop, n<=100     Repeat the process
EN
```

Now suppose that the slave axis is engaged with a start signal, input 1, but that both the engagement and disengagement points must be done at the center of the cycle: A = 1000 and B = 500. This implies that B must be driven to that point to avoid a jump.

This is done with the program:

INSTRUCTION	INTERPRETATION
#run	Label
SG 0	Select bank 0
EB1	Enable cam
PA,500	starting position
SP,5000	B speed
BGB	Move B motor
AM	After B moved
AI1	Wait for start signal
EG,1000	Engage slave
AI - 1	Wait for stop signal
EQ,1000	Disengage slave
EN	End

Command Summary - Electronic CAM

Command	Description
EA m	Specifies master axes for electronic cam where: m = A,B,C,D,E,F,G,H for main encoder as master or M or N a for virtual axis master
EB n	Enables the ECAM
EC n	ECAM counter - sets the index into the ECAM table
EG n,n,n,n,n,n,n,n	Engages ECAM
EM n,n,n,n,n,n,n,n	Specifies the change in position for each axis of the CAM cycle
EP n_0,n_1	Defines CAM table entry size and offset
EQ n,n,n,n,n,n,n,n	Disengages ECAM at specified position
ET[n]	Defines the ECAM table entries
EW n_0=n_1,n_2=n_3	Widen Segment (see Application Note #2444)
EY n	Set ECAM cycle count

Operand Summary - Electronic CAM

Command	Description
_EB	Contains State of ECAM
_EC	Contains current ECAM index
_EGm	Contains ECAM status for each axis
_EMm	Contains size of cycle for each axis
_EP	Contains value of the ECAM table interval
_EQm	Contains ECAM status for each axis
_EY	Set ECAM cycle count

Example - Electronic CAM

The following example illustrates a cam program with a master axis C and two slaves, A and B.

INSTRUCTION	INTERPRETATION
#a;v1=0	Label; Initialize variable
SG 0	Select bank 0
PA 0,0;BGA;AMAB	Go to position 0,0 on A and B axes
SB 1	Select bank 0
EA C	C axis as the Master for ECAM
EM 0,0,4000	Change for C is 4000, zero for A, B
EP 400,0	ECAM interval is 400 counts with zero start
ET[0]=0,0	When master is at 0 position; 1st point.
ET[1]=40,20	2nd point in the ECAM table
ET[2]=120,60	3rd point in the ECAM table
ET[3]=240,120	4th point in the ECAM table
ET[4]=280,140	5th point in the ECAM table
ET[5]=280,140	6th point in the ECAM table
ET[6]=280,140	7th point in the ECAM table
ET[7]=240,120	8th point in the ECAM table
ET[8]=120,60	9th point in the ECAM table
ET[9]=40,20	10th point in the ECAM table
ET[10]=0,0	Starting point for next cycle
EB 1	Enable ECAM mode
JGC=4000	Set C to jog at 4000
EG 0,0	Engage both A and B when Master = 0
BG C	Begin jog on C axis
#loop;JP#loop,v1=0	Loop until the variable is set
EQ2000,2000	Disengage A and B when Master = 2000
MF,, 2000	Wait until the Master goes to 2000
ST C	Stop the C axis motion
EB 0	Exit the ECAM mode
EN	End of the program

The above example shows how the ECAM program is structured and how the commands can be given to the controller. Figure 6.12 shows the GalilTools scope capture of the ECAM profile. This shows how the motion will be seen during the ECAM cycles. The first trace is for the A axis, the second trace shows the cycle on the B axis and the third trace shows the cycle of the C axis.

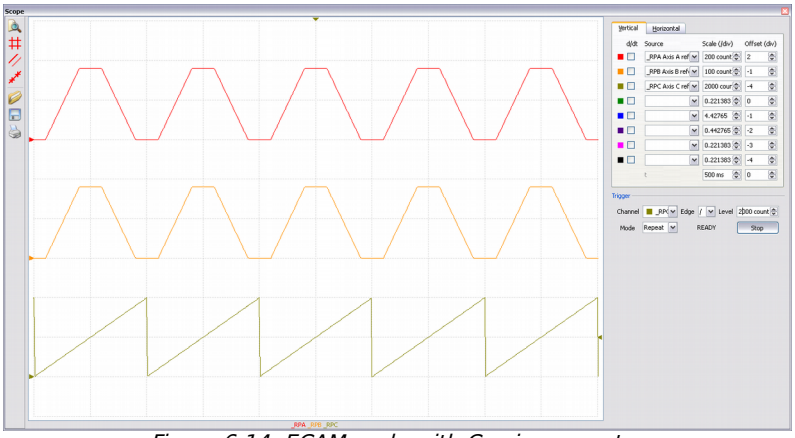

Figure 6.14: ECAM cycle with C axis as master

PVT Mode

The DMC-52xx0 controllers supports a mode of motion referred to as “PVT.” This mode allows arbitrary motion profiles to be defined by position, velocity, and time individually on all axes. 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 must be entered one axis at a time using the PVn command. The PV command includes the target distance to be moved and target velocity to be obtained over the specified time frame. 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 for each axis prior to motion beginning. The BT command will ensure that all axes begin motion simultaneously on the selected bank. It is not required for the time segment for each axis to be the same, however if they are then the axes will remain coordinated. Each axis has a 255 segment buffer. This buffer is a FIFO and the available space can be queried with the operand _PVn. As the buffer empties the user can add more PVT segments.

Exiting PVT Mode

To exit PVT mode the user must send the segment command PVn=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 PVn=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:PVT mode exited because buffer is empty`. During normal operation of PVT mode the stop code will be `30:Running in PVT mode`. If PVT mode is exited normally (PVn=0,0,0), then the stop code will be set to `31:PVT mode completed normally`.

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 and possibly beyond the capabilities of the system. This may result in excessive position error and damage to the system. 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.

Command Summary – PVT

COMMAND	DESCRIPTION
PVm = n_0, n_1, n_2	Specifies the segment of axis 'm' for a incremental PVT segment of 'n_0' counts, an end speed of 'n_1' counts/sec in a total time of 'n_2' samples.
_PVm	Contains the number of PV segments available in the PV buffer for a specified axes.
BTmm	Begin PVT mode
_BTm	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 6.15.

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

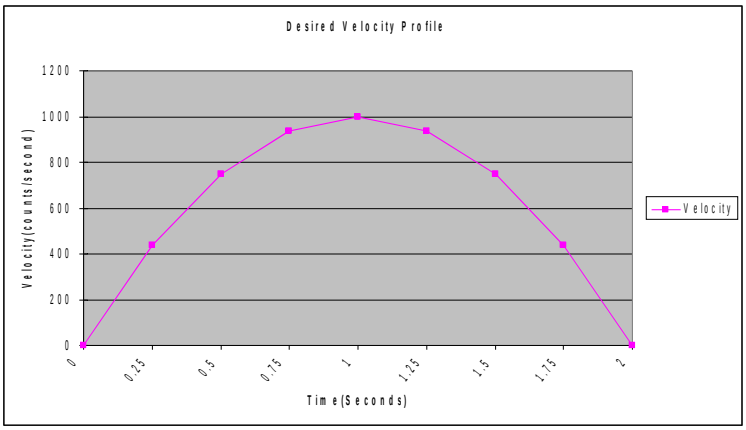

Figure 6.15: Parabolic Velocity Profile

To accomplish this we need to calculate the desired velocities and change in positions. In this example we will assume a delta time of ¼ of a second, which is 250 samples (1000 samples = 1 second).

Velocity(counts/second)	Position(counts)
$v(t) = -1000(t-1)^2 + 1000$	$p(t) = \int (-1000(t-1)^2 + 1000)dt$
$v(.25) = 437.5$	p(0 to .25) = 57
$v(.5) = 750$	p(.25 to .5) = 151

$v(.75) = _{937.5}$	p(.5 to .75) = 214
$v(1) = _{1000}$	p(.75 to 1) = 245
$v(1.25) = _{937.5}$	p(1 to 1.25) = 245
$v(1.5) = _{750}$	p(1.25 to 1.5) = 214
$v(1.75) = _{437.5}$	p(1.5 to 1.75) = 151
$v(2) = _{0}$	p(1.75 to 2) =57

The DMC program is shown below and the results can be seen in Figure 6.14.

INSTRUCTION	INTERPRETATION
`#pvt`	Label
`SG 0`	Select bank 0
`PVA = 57,437,256`	Incremental move of 57 counts in 250 samples with a final velocity of 437 counts/sec
`PVA = 151,750,256`	Incremental move of 151 counts in 250 samples with a final velocity of 750 counts/sec
`PVA = 214,937,256`	Incremental move of 214 counts in 250 samples with a final velocity of 937 counts/sec
`PVA = 245,1000,256`	Incremental move of 245 counts in 250 samples with a final velocity of 1000 counts/sec
`PVA = 245,937,256`	Incremental move of 245 counts in 250 samples with a final velocity of 937 counts/sec
`PVA = 214,750,256`	Incremental move of 214 counts in 250 samples with a final velocity of 750 counts/sec
`PVA = 151,437,256`	Incremental move of 151 counts in 250 samples with a final velocity of 437 counts/sec
`PVA = 57,0,256`	Incremental move of 57 counts in 250 samples with a final velocity of 0 counts/sec
`PVA = 0,0,0`	Termination of PVT buffer
`BTA`	Begin PVT
`EN`	

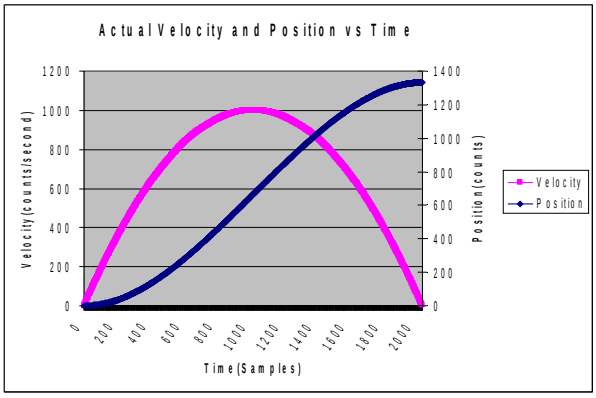

Figure 6.16: Actual Velocity and Position vs Time of Parabolic Velocity Profile

Multi-Axis Coordinated Move

Figure 6.17: Required AB Points

A-Axis	B-Axis
500	500
1500	5000
2500	4000
3300	4200
7300	3300

Many applications require moving two or more axes in a coordinated move yet still require smooth motion. These applications are ideal candidates for PVT mode. In this example, there is a two dimensional stage that needs to follow a specific profile. The application requires that certain points be met however the path between points is not important. Smooth motion between points is critical.

The resulting DMC program is shown below. The position points are dictated by the application requirements and the velocities and times were chosen to create smooth, yet quick, motion. For example, in the second segment the B axis is slowed to 0 at the end of the move in anticipation of reversing direction during the next segment.

INSTRUCTION	INTERPRETATION
`#pvt`	Label
`SG 0`	Select bank 0
`PVA = 500,2000,500`	1st point in Figure 6.16 - A axis
`PVB = 500,5000,500`	1st point in Figure 6.16 - B axis
`PVA = 1000,4000,1200`	2nd point in Figure 6.16 - A axis
`PVB = 4500,0,1200`	2nd point in Figure 6.16 - B axis
`PVA = 1000,4000,750`	3rd point in Figure 6.16 - A axis
`PVB = -1000,1000,750`	3rd point in Figure 6.16 - B axis
`BTAB`	Begin PVT mode for A and B axes
`PVA = 800,10000,250`	4th point in Figure 6.16 - A axis
`PVB = 200,1000,250`	4th point in Figure 6.16 - B axis
`PVA = 4000,0,1000`	5th point in Figure 6.16 - A axis
`PVB = -900,0,1000`	5th point in Figure 6.16 - B axis
`PVA = 0,0,0`	Termination of PVT buffer for A axis
`PVB = 0,0,0`	Termination of PVT buffer for B axis
`EN`	

Note: The BT command is issued prior to filling the PVT buffers and additional PV commands are added during motion for demonstration purposes only. The BT command could have been issued at the end of all the PVT points in this example.

The resultant A vs. B position graph is shown in Figure 6.18, with the specified PVT points enlarged.

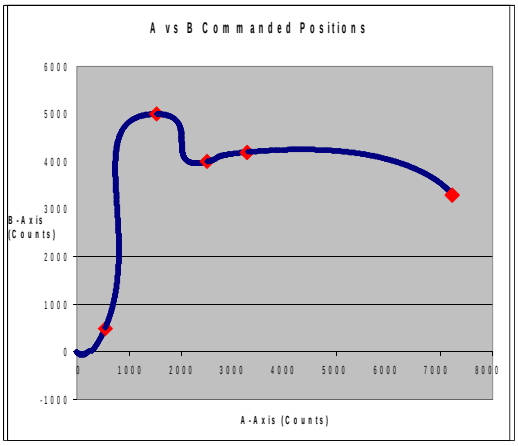

Figure 6.18: A vs B Commanded Positions for Multi-Axis Coordinated Move

Contour Mode

The DMC-52xx0 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

Contour Mode is specified with the command CM. For example, CMAB specifies contouring on the A and B axes. Any axes that are not being used in the contouring mode may be operated in other modes.

A contour consists of position increments which are described with the command CD n,n,n,n,n,n,n,n over a time interval DT n, which is the same for all banks. The parameter 'n' specifies the time interval. The time interval is defined as 2^n sample period, 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 n,n,n,n,n,n,n,n=t where 't' is the new DT value.

Consider, for example, the trajectory shown in Figure 6.19. The position A may be described by the points:

Point 1	A=0 at T=0ms
Point 2	A=48 at T=4ms
Point 3	A=288 at T=12ms
Point 4	A=336 at T=28ms

The same trajectory may be represented by the increments

Increment 1	DA=48	Time=4	DT=2

Increment 2	DA=240	Time=8	DT=3
Increment 3	DA=48	Time=16	DT=4

When the controller receives the command to generate a trajectory along these points, it linearly interpolates 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
SG 0                        Select bank 0
CMA                         Specifies A 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
```

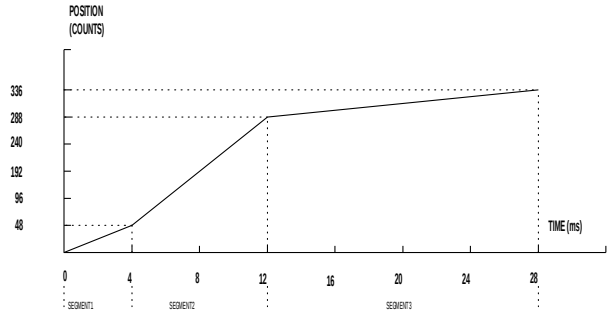

Figure 6.19: The Required Trajectory

Additional Information

_CM gives the amount of space available in the contour buffer (511 maximum) for the selected bank. Zero parameters for DT followed by zero parameters for CD will exit the contour mode.

If no new data is found and the controller is still in the contour mode, and will 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 mm	Contour axes for DMC-52xx0 with 8 axis or more
CD n,n,n,n,n,n,n,n=t	Specifies position increment over time interval. Range is ±32,000. CD 0,0,0.. .=0 ends the contour buffer. This is much like the LE or VE commands.
DT n	Specifies time interval 2^n sample periods (1 ms) 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

Contour Mode is ideal for generating any arbitrary velocity profiles. Velocity profiles 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.

Example: Generating an Array

Consider the velocity and position profiles shown in Figure 6.20. 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: ω is the angular velocity; X is the position; and T is the variable, time, in milliseconds.

In the given example, A=6000 and B=120, the position and velocity profiles are:

$$X = 50T - (6000/2\pi) \sin (2\pi T/120)$$

Note that the velocity, ω, in count/ms, is

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

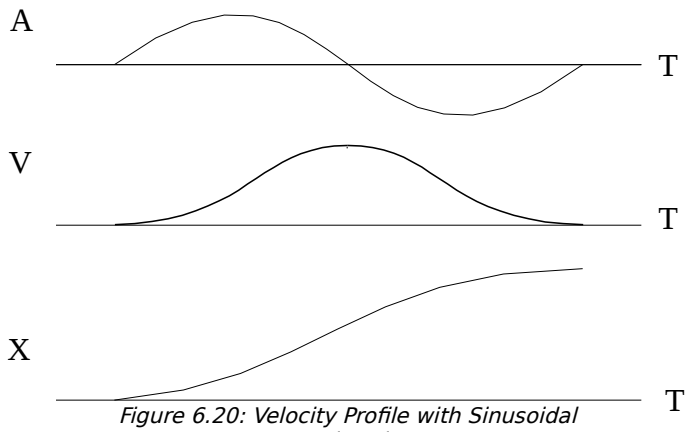

Figure 6.20: Velocity Profile with Sinusoidal Acceleration

The DMC-52xx0 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[]`.

Contour Mode Example

INSTRUCTION	INTERPRETATION
`#points`	Program defines A points
`SG 0`	Select bank 0
`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
CMA                         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 by using the DMC-52xx0 automatic array capture feature to capture position data. The captured data may then be played back in contour mode. The following array commands are used:

`DM c[n]`	Dimension array
`RA c[]`	Specify array for automatic record (up to 8)
`RD _TPA`	Specify data for capturing (such as _TPA or _TPB)
`RC` n_0, n_1	Specify capture time interval where n_0 is 2^n sample periods (1 ms), n_1 is number of records to be captured
`RC? or _RC`	Returns a 1 if recording

Record and Playback Example:

```
#record                 Begin Program
SG 0                    Select bank 0
DM apos[501]            Dimension array with 501 elements
RA apos[]               Specify automatic record
RD _TPA                 Specify A position to be captured
MOA                     Turn A motor off
RC2                     Begin recording 4 msec interval
#a;JP#a,_RC=1           Continue until done recording
#compute                Compute da[]
DM da[500]              Dimension array for da[]
c=0                     Initialize counter
#l                      Label
d=c+1
delta=apos[d]-apos[c]   Compute the difference
da[c]=delta             Store difference in array
c=c+1                   Increment index
JP #l,c<500             Repeat until done
#playbck                Begin Playback
CMA                     Specify contour mode
DT2                     Specify time increment
i=0                     Initialize array counter
#b                      Loop counter
CD da[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 Arrays section, pg109.

Virtual Axis

The DMC-52xx0 controller has two additional virtual axes per bank designated as the M and N axes. These axes have no encoder and no DAC. However, they can be commanded by the commands:

AC, DC, JG, SP, PR, PA, BG, IT, GA, VM, VP, CR, ST, DP, RP

The main use of the virtual axes is to serve as a virtual master in ECAM modes, and to perform an unnecessary part of a vector mode. These applications are illustrated by the following examples.

ECAM Master Example

Suppose that the motion of the AB 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 AB axes to move to the corresponding points on the motion cycle.

Sinusoidal Motion Example

The A-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 A 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
SG 0	Select bank 0
VMAN	Select Axes
VA 68000000	Maximum Acceleration
VD 68000000	Maximum Deceleration
VS 125664	VS for 20 Hz
CR 1000, -90, 3600	Ten Cycles
VE	
BGS	

Motion Smoothing

The DMC-52xx0 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:

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 a reduction in mechanical vibrations.

The smoothing function is specified by the following commands:

IT n,n,n,n,n,n,n,n	Independent time constant

The command, IT, is used for smoothing independent moves of the type JG, PR, PA. It can also be used to smooth vector moves such as VM and LM **for the selected bank**.

The smoothing parameter 'n' is a number between 0.004 and 1 and determines 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 6.21 shows the trapezoidal velocity profile and the modified acceleration and velocity.

Note: 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 A	Begin

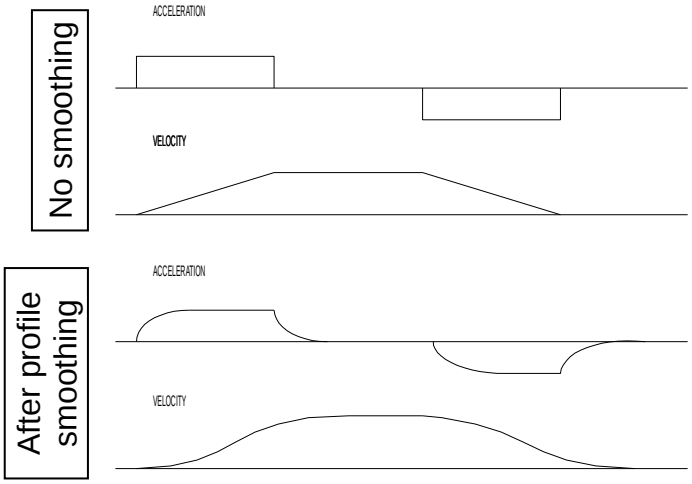

Figure 6.21: Trapezoidal velocity and smooth velocity profiles

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 on the EtherCAT slave. 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 on the EtherCAT slave. When the Find Edge command and Begin Motion (BG) 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 enables finer positioning on initialization. The HM command and BG command causes the following sequence of events to occur.

Stage 1:

After issuing the BG command, 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. The CN command is 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 a change in state of the Home switch, 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 the speed specified by the HV command in the opposite direction of Stage 1 until the Home switch toggles again.

Note: 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 the speed specified by the HV command until the encoder index pulse is detected. The motor then decelerates to a stop and goes back to the index. The index may not be supported on some drives. See the EtherCAT drives' manufacturer for more details.

The DMC-52xx0 defines the home position as the position at which the index was detected and sets the encoder reading at this point to zero.

The four 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 6.22 shows the velocity profile from the homing sequence of the example program above. For this profile, the switch is normally closed and CN,-1.

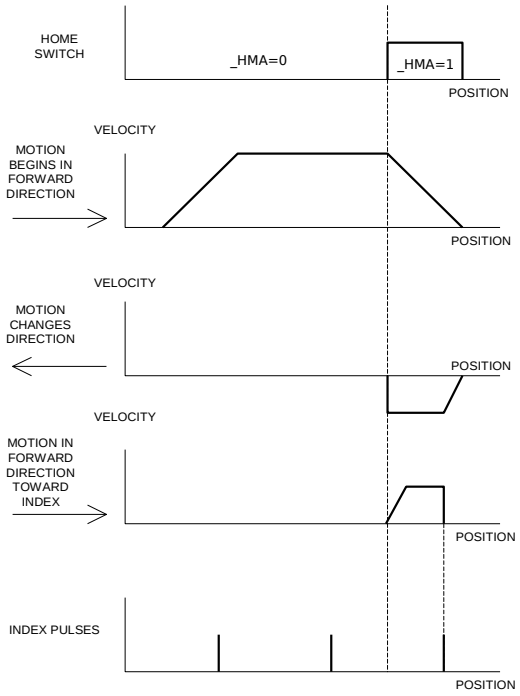

Figure 6.22: Homing Sequence for Normally Closed Switch and CN,-1

Example: Find Edge

```
#edge                Label
SG 0                 Select bank 0
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 mm	Find Edge Routine. This routine monitors the Home Input
FI mm	Find Index Routine - This routine monitors the Index Input
HM mm	Home Routine - This routine combines FE and FI as described above
SC mm	Stop Code for selected bank
TS mm	Tell Status of Switches and Inputs for selected bank

Operand Summary - Homing Operation

Operand	Description
_HMm	Contains the value of the state of the Home Input
_SCm	Contains stop code
_TSm	Contains status of switches and inputs

Chapter 7 Application Programming

Overview

The DMC-52xx0 provides a powerful programming language that allows users to customize the controller for their particular application. Programs can be downloaded into the DMC-52xx0 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.

Each set of 8 axes on the DMC-52xx0 is referred to as a bank. For example the DMC-52160 is a 16 axis controller, it would have 2 banks of 8 axes. The user can specify which bank of axes they would like to work with using the **SG** command. Where SG 0 would set the current bank of axes as the 1st bank, SG 1 would set the current bank of axes to the 2nd bank, and so on. Each bank can have a number of axes that can range from 1-8 (A-H). If a bank of axes is not specified the controller will default to the bank 0, which is comprised of the first 8 axes. Coordinated motion can be achieved within the 8 axes of any given bank.

In addition to standard motion commands, the DMC-52xx0 provides commands that allow the DMC-52xx0 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-52xx0 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 4000 lines.

Program Format

A DMC-52xx0 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-52xx0 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-52xx0 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 label.

The maximum number of labels which may be defined is 510.

Valid labels

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

Invalid labels

```
#1Square
#123
```

A Simple Example Program:

`#START`	Beginning of the Program
`SG 0;'`	Set bank 0, axes 1-8
`PR 10000,20000;'`	Specify relative distances on X and Y axes
`BG XY;'`	Begin Motion on the X and Y axes on the first bank.
`AM XY;'`	Wait for motion to complete on axes X and Y on the first bank.
`WT 2000;'`	Wait 2 sec
`JP #START;'`	Jump to label START
`EN;'`	End of Program

The above program moves X and Y axes on the first bank 10000 and 20000 units. After the motion is complete, the motors rest for 2 seconds. The cycle repeats indefinitely until the stop command is issued.

Special Labels

The DMC-52xx0 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 ___

Label Name	Description
`#AUTO`	Label that will automatically run upon the controller exiting a reset (power-on)
`#AUTOERR`	Label that will automatically run if there is an EEPROM error out of reset
`#CMDERR`	Label for incorrect command subroutine
`#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 operation NO or Apostrophe (‘)

The DMC-52xx0 provides a command, NO, for commenting programs or single apostrophe. This command allows the user to include up to 78 characters on a single line after the NO command and can be used to include comments from the programmer as in the following example:

```
#PATH
‘ 2-D CIRCULAR PATH
VMXY
‘  VECTOR MOTION ON X AND Y
VS 10000
‘  VECTOR SPEED IS 10000
VP -4000,0
‘  BOTTOM LINE
CR 1500,270,-180
‘  HALF CIRCLE MOTION
VP 0,3000
‘  TOP LINE
CR 1500,90,-180
‘  HALF CIRCLE MOTION
VE
‘  END VECTOR SEQUENCE
BGS
‘  BEGIN SEQUENCE MOTION
EN
‘  END OF PROGRAM
```

Note: The NO command is an actual controller command. Therefore, inclusion of the NO commands will require process time by the controller.

Difference between NO and ' using the GalilTools software

The GalilTools software will treat an apostrophe (') commend different from an NO when the compression algorithm is activated upon a program download (line > 80 characters or program memory > 4000 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-52xx0 can run up to 8 independent programs simultaneously. These programs are called threads and are numbered 0 through 7, where 0 is the main thread. Multitasking is useful for executing independent operations such as PLC functions that occur independently of motion.

The main thread differs from the others in the following ways:

1. 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.

SG 0;'	Set bank 0, axes 1-8 on the controller
#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 move for axis X on the first bank.
BGX;'	Begin motion for the X axis on the first bank.
AMX;'	After motion is completed on the X axis on the first bank.
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-52xx0 provides commands and operands which are useful in debugging application programs. These commands include interrogation commands to monitor program execution, determine the state of the controller and the contents of the controllers program, array, and variable space. Operands also contain important status information which can help to debug a program.

Trace Commands

The trace command causes the controller to send each line in a program to the host computer immediately prior to execution. Tracing is enabled with the command, TR1. TR0 turns the trace function off. Note: When the trace function is enabled, the line numbers as well as the command line will be displayed as each command line is executed.

Note: When the trace function is enabled, the line numbers as well as the command line will be displayed as each command line is executed.

Data which is output from the controller is stored in the output UART. The UART buffer can store up to 512 characters of information. In normal operation, the controller places output into the FIFO buffer. When the trace mode is enabled, the controller will send information to the UART buffer at a very high rate. In general, the UART will become full because the hardware handshake line will halt serial data until the correct data is read. When the UART becomes full, program execution will be delayed until it is cleared. If the user wants to avoid this delay, the command CW,1 can be given. This command causes the controller to throw away the data which can not be placed into the FIFO. In this case, the controller does not delay program execution.

Error Code Command

When there is a program error, the DMC-52xx0 halts the program execution at the point where the error occurs. To display the last line number of program execution, issue the command, MG _ED.

The user can obtain information about the type of error condition that occurred by using the command, TC1. This command reports back a number and a text message which describes the error condition. The command, TC0 or TC, will return the error code without the text message. For more information about the command, TC, see the Command Reference.

Stop Code Command

The status of motion for each axis can be determined by using the stop code command, SC. This can be useful when motion on an axis has stopped unexpectedly. The command SC will return a number representing the motion status. See the command reference for further information.

RAM Memory Interrogation Commands

For debugging the status of the program memory, array memory, or variable memory, the DMC-52xx0 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-52xx0 will have a maximum of 24000 array elements in up to 30 arrays. If an array of 100 elements is defined, the command DM ? will return the value 23900 and the command DA ? will return 29.

To list the contents of the variable space, use the interrogation command LV (List Variables). To list the contents of array space, use the interrogation command, LA (List Arrays). To list the

contents of the Program space, use the interrogation command, LS (List). To list the application program labels only, use the interrogation command, LL (List Labels).

Operands

In general, all operands provide information which may be useful in debugging an application program. Below is a list of operands which are particularly valuable for program debugging. To display the value of an operand, the message command may be used. For example, since the operand, _ED contains the last line of program execution, the command MG _ED will display this line number.

_ED contains the last line of program execution. Useful to determine where program stopped.
_DL contains the number of available labels.
_UL contains the number of available variables.
_DA contains the number of available arrays.
_DM contains the number of available array elements.
_AB contains the state of the Abort Input
_LFx contains the state of the forward limit switch for the 'x' axis
_LRx contains the state of the reverse limit switch for the 'x' axis

Debugging Example:

The following program has an error. It attempts to specify a relative movement while the X-axis is already in motion. When the program is executed, the controller stops at line 003. The user can then query the controller using the command, TC1. The controller responds with the corresponding explanation:

```
Download Code
#A;'                    Program Label
SG 1;'                  Select bank 2 , axes 9-16.
PR1000;'                Position Relative of 1000 counts for the X axis on the 2nd bank.
BGX;'                   Begin motion on the X axis on the 2nd bank.
PR5000;'                Position Relative 5000 for the X axis on the 2nd bank.
EN;'                    End
From Terminal
:XQ #A                  Execute #A
?004 PR5000             Error on Line 4
:TC1                    Tell Error Code
?7 Command not valid    Command not valid while running
while running.
                        Change the BGX line to BGX;AMX and re-download the program.
:XQ #A                  Execute #A
```

Program Flow Commands

The DMC-52xx0 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-52xx0 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-52xx0 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-52xx0 Event Triggers

Command	Function
AM X Y Z W or S (A B C D E F G H)	Halts program execution until motion is complete on the specified axes or motion sequence(s). AM with no parameter tests for motion complete on all axes in a bank. This command is useful for separating motion sequences in a program.
AD X or Y or Z or W (A or B or C or D or E or F or G or H)	Halts program execution until position command has reached the specified relative distance from the start of the move. Only one axis may be specified at a time.
AR X or Y or Z or W (A or B or C or D or E or F or G or H)	Halts program execution until after specified distance from the last AR or AD command has elapsed. Only one axis may be specified at a time.
AP X or Y or Z or W (A or B or C or D or E or F or G or H)	Halts program execution until after absolute position occurs. Only one axis may be specified at a time.
MF X or Y or Z or W (A or B or C or D or E or F or G or H)	Halt program execution until after forward motion reached absolute position. Only one axis may be specified. If position is already past the point, then MF will trip immediately. Will function on geared axis or aux. inputs.
MR X or Y or Z or W (A or B or C or D or E or F or G or H)	Halt program execution until after reverse motion reached absolute position. Only one axis may be specified. If position is already past the point, then MR will trip immediately. Will function on geared axis or aux. inputs.
MC X or Y or Z or W (A or B or C or D or E or F or G or H)	Halt program execution until after the motion profile has been completed and the encoder has entered or passed the specified position. TW x,y,z,w sets timeout to declare an error if not in position. If timeout occurs, then the trippoint will clear and the stop code will be set to 99. An application program will jump to label #MCTIME.
AI ± n	Halts program execution until after specified input is at specified logic level. n specifies input line. Positive is high logic level, negative is low level. n=1 through 8 for the DMC-52xx0.

AS X Y Z W S (A B C D E F G H)	Halts program execution until specified 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
SG 1;'              Select bank 1, Axes 9-16
PR 2000;'           Position Command
BGX;'               Begin Motion for the X axis on the 2nd bank.
AMX;'               Wait for Motion Complete for the X axis on the 2nd bank.
PR 4000;'           Next Position Move for the X axis on the 2nd bank.
BGX;'               Begin 2nd move for the X axis on the 2nd bank.
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
SG 0;'              Select bank 0, Axes 1-8
SP 10000;'          Speed is 10000 move for the X axis on the 1st bank.
PA 20000;'          Specify Absolute position for the X axis on the 1st bank.
BGX;'               Begin motion for the X axis on the 1st bank.
AD 1000;'           Wait until 1000 counts have been traveled for the X axis on the 1st bank.
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.

```
SG 1;'              Select bank 1, Axes 9-16
#TRIP;'             Label
JG 50000;'          Specify Jog Speed for X axis on the 2nd bank.
BGX;n=0;'           Begin Motion for X axis on the 2nd bank.
#REPEAT;'           # Repeat Loop
AR 10000;'          Wait 10000 counts for X axis on the 2nd bank.
TPX;'               Tell Position of the  X axis on the 2nd bank.
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 motion for the  X axis on the 2nd bank.
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.

SG 1;'	Select bank 1, axes 9-16.
#INPUT;'	Program Label
AI-1;'	Wait for input 1 low
PR 10000;'	Position Relative move for X axis on the 2nd bank.
BGX;'	Begin motion for X axis on the 2nd bank.
EN;'	End program

Event Trigger - Set output when At speed

SG 0;'	Select bank 0, axes 1-8.
#ATSPEED;'	Program Label
JG 50000;'	Specify jog speed for X axis on the 1st bank.
AC 10000;'	Acceleration rate for X axis on the 1st bank.
BGX;'	Begin motion for X axis on the 1st bank.
ASX;'	Wait for at slew speed 50000 for X axis on the 1st bank.
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.

SG 1;'	Select bank 1, axes 9-16.
#VECTOR;'	Label
VMXY;VS 5000;'	Enable Vector Mode for axes X and Y on the 2nd bank, with a vector speed of 5000.
VP 10000,20000;'	Vector position for axes X and Y on the 2nd bank.
VP 20000,30000;'	Vector position for axes X and Y on the 2nd bank.
VE;'	End vector sequence for axes X and Y on the 2nd bank.
BGS;'	Begin sequence on the S motion plane for axes X and Y on the 2nd bank.
AV 5000;'	After vector distance
VS 1000;'	Reduce vector 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.

SG 1;'	Select bank 1, axes 9-16.
#MOVES;'	Label
PR 12000;'	Position Relative move for X axis on the 2nd bank.
SP 20000;'	Speed for X axis on the 2nd bank.
AC 100000;'	Acceleration for X axis on the 2nd bank.
BGX;'	Start Motion for X axis on the 2nd bank.
AD 10000;'	Wait a distance of 10,000 counts for X axis on the 2nd bank.
SP 5000;'	New Speed for X axis on the 2nd bank.
AMX;'	Wait until motion is completed for X axis on the 2nd bank.
WT 200;'	Wait 200 ms
PR -10000;'	New Position Relative move for X axis on the 2nd bank.
SP 30000;'	New Speed for X axis on the 2nd bank.
AC 150000;'	New Acceleration for X axis on the 2nd bank.
BGX;'	Start Motion for X axis on the 2nd bank.
EN;'	End

Define Output Waveform Using AT

The following program causes Output 1 to be high for 10 msec and low for 40 msec. The cycle repeats every 50 msec.

```
#OUTPUT;'          Program label
AT0;'              Initialize time reference
SB1;'              Set Output 1
#LOOP;'            Loop
AT 10;'            After 10 msec from reference,
CB1;'              Clear Output 1
AT -40;'           Wait 40 msec from reference and reset reference
SB1;'              Set Output 1
JP #LOOP;'         Loop
EN;'               End Program
```

Conditional Jumps

The DMC-52xx0 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-52xx0 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-52xx0 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-52xx0 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-52xx0 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.

```
SG 1;'              Select bank 1, axes 9-16.
#BEGIN;'            Begin Program
Count=10;'          Initialize loop counter
#LOOP;'             Begin loop
PA 1000;'           Position Absolute move of 1000 for X axis on the 2nd bank.
BGX;'               Begin move for X axis on the 2nd bank.
AMX;'               Wait for motion complete for X axis on the 2nd bank.
WT 100;'            Wait 100 msec
PA 0;'              Position Absolute move of 0 for X axis on the 2nd bank.
BGX;'               Begin move for X axis on the 2nd bank.
AMX;'               Wait for motion complete for X axis on the 2nd bank.
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-52xx0 provides a structured approach to conditional statements using IF, ELSE and ENDIF commands.

Using the IF and ENDIF Commands

An IF conditional statement is formed by the combination of an IF and ENDIF command. The IF command has as it's arguments one or more conditional statements. If the conditional statement(s) evaluates true, the command interpreter will continue executing commands which follow the IF command. If the conditional statement evaluates false, the controller will ignore commands until the associated ENDIF command is executed OR an ELSE command occurs in the program (see discussion of ELSE command below).

Note: An ENDIF command must always be executed for every IF command that has been executed. It is recommended that the user not include jump commands inside IF conditional statements since this causes re-direction of command execution. In this case, the command interpreter may not execute an ENDIF command.

Using the ELSE Command

The ELSE command is an optional part of an IF conditional statement and allows for the execution of command only when the argument of the IF command evaluates False. The ELSE command must occur after an IF command and has no arguments. If the argument of the IF command evaluates false, the controller will skip commands until the ELSE command. If the argument for the IF command evaluates true, the controller will execute the commands between the IF and ELSE command.

Nesting IF Conditional Statements

The DMC-52xx0 allows for IF conditional statements to be included within other IF conditional statements. This technique is known as 'nesting' and the DMC-52xx0 allows up to 255 IF conditional statements to be nested. This is a very powerful technique allowing the user to specify a variety of different cases for branching.

Command Format - IF, ELSE and ENDIF

Format:	Description
IF conditional statement(s)	Execute commands proceeding IF command (up to ELSE command) if conditional statement(s) is true, otherwise continue executing at ENDIF command or optional ELSE command.
ELSE	Optional command. Allows for commands to be executed when argument of IF command evaluates not true. Can only be used with IF command.
ENDIF	Command to end IF conditional statement. Program must have an ENDIF command for every IF command.

Example using IF, ELSE and ENDIF:

`#TEST;'`	Begin Main Program "TEST"
`II,,3;'`	Enable input interrupts on input 1 and input 2
`MG "WAITING FOR INPUT 1, INPUT 2";'`	Output message
`#LOOP;'`	Label to be used for endless loop
`JP #LOOP;'`	Endless loop
`EN;'`	End of main program
`#ININT;'`	Input Interrupt Subroutine
`IF (@IN[1]=0);'`	IF conditional statement based on input 1
`IF (@IN[2]=0);'`	2nd IF conditional statement executed if 1st IF conditional true
`MG "INPUT 1 AND INPUT 2 ARE ACTIVE";'`	Message to be executed if 2nd IF conditional is true
`ELSE;'`	ELSE command for 2nd IF conditional statement
`MG "ONLY INPUT 1 IS ACTIVE";'`	Message to be executed if 2nd IF conditional is false
`ENDIF;'`	End of 2nd conditional statement
`ELSE;'`	ELSE command for 1st IF conditional statement
`MG"ONLY INPUT 2 IS ACTIVE";'`	Message to be executed if 1st IF conditional statement is false
`ENDIF;'`	End of 1st conditional statement
`#WAIT;'`	Label to be used for a loop
`JP#WAIT,(@IN[1]=0) | (@IN[2]=0);'`	Loop until both input 1 and input 2 are not active
`RI0;'`	End Input Interrupt Routine without restoring trippoints

Subroutines

A subroutine is a group of instructions beginning with a label and ending with an end command (EN). Subroutines are called from the main program with the jump subroutine instruction JS, followed by a label or line number, and conditional statement. Up to 8 subroutines can be nested. After the subroutine is executed, the program sequencer returns to the program location where the subroutine was called unless the subroutine stack is manipulated as described in the following section.

Example:

An example of a subroutine to draw a square 500 counts per side is given below. The square is drawn at vector position 1000,1000.

```
SG 1                    Select bank 1, Axes 9-16.
#M                      Begin Main Program
CB1                     Clear Output Bit 1 (pick up pen)
VP 1000,1000;VE;BGS     Vector Position of 1000  for axes X and Y on the 2nd bank
AMS                     Wait for motion to complete on axes X and Y on the 2nd bank
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 motion for the X axis on the 2nd bank.
AMX;BGY;AMY             After motion is completed for the X and Y axis on the 2nd bank
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-52xx0 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-52xx0 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)
#ECATERR	Executes whenever there is an EtherCAT network error.

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-52xx0 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.

```
SG 0                          Select bank 0, axes 1-8.
#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 X axis on the 1st bank.
:BGX                          Begin Motion for the X axis on the 1st bank.
```

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
SG 0;'                        Select bank 0, Axes 1-8.
v1=_TEX;'                     Read Position Error for the X axis on the 1st bank.
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 of 100000 for the X axis on the 1st
                              bank.
:BGX                          Begin Motion for the X axis on the 1st bank.
```

Example - Input Interrupt

```
#A;'                          Label
```

```
SG 0;'                          Select bank 0, axes 1-8.
II1;'                           Input Interrupt on 1
JG 30000,,,60000                Jog for Axes X and W on the 1st bank.
BGXW                            Begin Motion for Axes X and W on the 1st bank.
#LOOP;JP#LOOP;EN                Loop
#ININT                          Input Interrupt
STXW;AM                         Stop Motion for Axes X and W on the 1st bank Wait for motion to
                                complete
#TEST;JP #TEST, @IN[1]=0;'      Test for Input 1 still low
JG 30000,,,6000;'               Restore Velocities for Axes X and W on the 1st bank.
BGXW;'                          Begin Motion for Axes X and W on the 1st bank.
RI0;'                           Return from interrupt routine to Main Program and do not re-
                                enable trippoints
```

Example - Motion Complete Timeout

```
#BEGIN                  Begin main program
SG 0;'                  Select bank 0, axes 1-8.
TW 1000;'               Set the time out to 1000 ms
PA 10000;'              Position Absolute command of 10000 for the X axis on the 1st bank.
BGX;'                   Begin motion for the X axis on the 1st bank.
MCX;'                   Motion Complete trippoint for the X axis on the 1st bank.
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
SG 0;'                      Select bank 0, axes 1-8.
Speed = 2000;'              Set variable for speed
JG speed;BGX;'              Begin motion on the X axis of the 1st bank.
#LOOP;'                     Start of #LOOP routine.
JG speed;WT100;'            Update speed for the X axis on the 1st bank  based upon speed
                            variable
JP #LOOP;'                  Jump to the label #LOOP
EN;'                        End main program
#CMDERR;'                   Command error automatic routine
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                       End If Statement
EN                          End of command error routine.
```

Example – Ethernet Communication Error

This simple program executes in the DMC-52xx0 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 – EtherCAT Error

```
ST!;'                                                    Stop Motion on all axes on all banks.
SG 0;'                                                   Select bank 0, axes 1-8.
 AMX;'                                                   After motion is complete on A axis of the 1st bank.
 MOX;'                                                   Turn off Axis A of the 1st bank.
 EU0;'                                                   Bring down the EtherCAT network
 MT10,10;'                                               Set the motor type for A and B axes of the 1st bank.
 EX-1,-2;'                                               Set Axis A as  AMC drive and Set Axis B  as Yaskawa on 1st bank.
 EU1;'                                                   Bing up the EtherCAT network
 SHAB;'                                                  Turn on the A and B axes on the 1st bank.
 JG 1000,2000;'                                          Jog A and B axes on the 1st bank.
 BGAB;'                                                  Begin motion for A and B axes on the 1st bank.
 EN;'                                                    End Program

 #ECATERR;'                                              EtherCAT Error Routine Label
 EZ0;'                                                   Supress EtherCAT Errors
 JS#amcerr,((_EU1 & $01) <>0);'                          Jump to #amcerr routine if error is present
 JS#yaserr,((_EU1 & $02) <>0);'                          Jump to #yaserr routine if error is present
 RE;'                                                    Return from Error Routine

 #amcerr;'                                               #amcerr routine label
 IF(_EZA2 =4);'                                          If the 2nd status word returns a Hall error
  MG "Hall Error on A Axis AMC Drive";'                  Message out Hall error
  MG "Check Encoder Cable";'                             Message out to check encoder cable
 ENDIF;'                                                 End If statement
EN;'                                                     End Routine.

 #yaserr;'
 IF(_EZB =$0C90);'                                       If the drive reports an encoder error
   MG "Encoder Error on B Axis Yaskawa Drive";'          Message out Encoder Error is present
   MG "Check Encoder and power cycle drive to clear error";'  Message out to check encoder cable
 ENDIF;'                                                 End If statement
EN;'                                                     End Routine.
```

JS Subroutine Stack Variables (^a, ^b, ^c, ^d, ^e, ^f, ^g, ^h)

There are 8 variables that may be passed on the subroutine stack when using the JS command. Passing values on the stack is advanced DMC programming, and is recommended for experienced DMC programmers familiar with the concept of passing arguments by value and by reference.

Notes:

1. Passing parameters has no type checking, so it is important to exercise good programming style when passing parameters. See examples below for recommended syntax.
2. Do not use spaces in expressions containing ^.
3. Global variables MUST be assigned prior to any use in subroutines where variables are passed by reference.
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                                 ;NO(^a,^b,^c,^d,^e,^f,^g,^h) syntax note for use
EN,,(^a+^b+^c+^d+^e+^f+^g+^h)        ;' return sum

:Executed program from program1.dmc
36.0000
```

Example: Variable, and an Important Note about Creating Global Variables

```
#Var
value=5                              ;'a value to be passed by reference
global=8                             ;'a global variable
JS#SUM(&value,1,2,3,4,5,6,7)         ;'note first arg passed by reference
MG value                             ;'message out value after subroutine.
MG _JS                               ;'message out returned value
EN
'
#SUM                                 ;NO(* ^a,^b,^c,^d,^e,^f,^g)
^a=^b+^c+^d+^e+^f+^g+^h+global
EN,,^a
'notes:
'do not use spaces when working with ^
'If using global variables, they MUST be created before the subroutine is run


Executed program from program2.dmc
36.0000
36.0000
```

Example: Working with Arrays

```
#Array
DM speeds[8]
DM other[256]
JS#zeroAry("speeds",0)              ;'zero out all buckets in speeds[]
JS#zeroAry("other",0)               ;'zero out all buckers in other[]
EN
'
#zeroAry                            ;NO(array ^a, ^b) zeros array starting at index ^b
^a[^b]=0
^b=^b+1
JP#zeroAry,(^b<^a[-1])              ;'[-1] returns the length of an array
EN
```

Example: Abstracting Axes

```
#Axes
JS#runMove(0,10000,1000,100000,100000)
MG "Position:",_JS
EN
'
#runMove              ;NO(axis ^a, PR ^b, SP ^c, AC ^d, DC ^e) Profile movement for axis
~a=^a                 ;'~a is global, so use carefully in subroutines
                       'try one variable axis a-h for a bank for each thread A-H
PR~a=^b
SP~a=^c
AC~a=^d
DC~a=^e
BG~a
MC~a
EN,,_TP~a
```

Example: Local Scope

```
#Local
JS#POWER(2,2)
MG_JS
JS#POWER(2,16)
MG_JS
JS#POWER(2,-8)
MG_JS
'
#POWER          ;NO(base ^a,exponent^b) Returns base^exponent power. ± integer only
^c=1            ;'unpassed variable space (^c-^h) can be used as local scope variables
IF ^b=0         ;'special case, exponent = 0
 EN,,1
ENDIF
IF ^b<0         ;'special case, exponent < 0, invert result
 ^d=1
 ^b=@ABS[^b]
ELSE
 ^d=0
ENDIF
#PWRHLPR
^c=^c*^a
^b=^b-1
JP#PWRHLPR,^b>0
IF ^d=1         ;'if inversion required
 ^c=(1/^c)
ENDIF
EN,,^c

Executed program from program1.dmc
4.0000
65536.0000
0.0039
```

Example: Recursion

```
'although the stack depth is only 16, Galil DMC code does support recursion
'this example shows axis data for Axes A-H on the 1st bank.
JS#AxsInfo(0)
MG{Z2.0}"Recursed through ",_JS," stacks"
EN
'
#AxsInfo                            ;NO(axis ^a) List info for axes
~h=^a
^b=(^a+$41)*$1000000                ;'convert to Galil String
MG^b{S1}, " Axis: "{N}
MG{F8.0}"Position: ",_TP~h," Error:",_TE~h," Torque:",_TT~h{F1.4}
IF ^a=7                             ;'recursion exit condition
EN,,1
ENDIF
JS#AxsInfo(^a + 1)                  ;'stack up recursion
EN,,_JS+1                           ;' as recursion closes, add up stack depths

Executed program from program1.dmc
A Axis: Position: 00029319 Error: 00001312 Torque: 9.9982
B Axis: Position: -00001612 Error: 00000936 Torque: 1.7253
C Axis: Position: 00001696 Error:-00001076 Torque:-1.9834
D Axis: Position: -00002020 Error: 00001156 Torque: 2.1309
E Axis: Position: 00000700 Error:-00001300 Torque:-2.3963
```

```
F Axis: Position: 00000156 Error:-00000792 Torque:-1.4599
G Axis: Position: -00002212 Error: 00001732 Torque: 3.1926
H Axis: Position: 00002665 Error:-00001721 Torque:-3.1723
Recursed through 8 stacks
```

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.

The GalilTools software will treat an apostrophe (') comment different from an NO when the compression algorithm is activated upon a program download (line > 80 characters or program memory > 4000 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.

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-52xx0, the output from the above program returned a 116, which indicates that it took 116 samples to process the commands from 't=TIME' to 'MG TIME-t'. This is about 114ms ±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-52xx0, the output from the program shown below returned a 62, which indicates that it took 62 samples to process the commands from 't=TIME' to 'MG TIME-t'. This is about 60ms ±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=_TI0&$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
AT-500
JP#scan
```

Mathematical and Functional Expressions

Mathematical Operators

For manipulation of data, the DMC-52xx0 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 ± 2,147,483,647.9999. The smallest number representable (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-52xx0 numeric operand, including variables, array elements, numeric values, functions, keywords, and arithmetic expressions. The bit-wise operators may also be used with strings. This is useful for separating characters from an input string. When using the input command for string input, the input variable will hold up to 6 characters. These characters are combined into a single value which is represented as 32 bits of integer and 16 bits of fraction. Each ASCII character is represented as one byte (8 bits), therefore the input variable can hold up to six characters. The first character of the string will be placed in the top byte of the variable and the last character will be placed in the lowest significant byte of the fraction. The characters can be individually separated by using bit-wise operations as illustrated in the following example:

```
#TEST                          Begin main program
IN “ENTER”,len{S6}             Input character string of up to 6 characters into variable ‘len’
Flen=@FRAC[len]                Define variable ‘Flen’ as fractional part of variable ‘len’
Flen=$10000*Flen               Shift Flen by 32 bits (IE - convert fraction, Flen, to integer)
len1=(Flen&$00FF)              Mask top byte of Flen and set this value to variable ‘len1’
len2=(Flen&$FF00)/$100         Let variable, ‘len2’ = top byte of Flen
len3=len&$000000FF             Let variable, ‘len3’ = bottom byte of len
len4=(len&$0000FF00)/$100      Let variable, ‘len4’ = second byte of len
len5=(len&$00FF0000)/$10000    Let variable, ‘len5’ = third byte of len
len6=(len&$FF000000)/$1000000  Let variable, ‘len6’ = fourth byte of len
MG len6 {S4}                   Display ‘len6’ as string message of up to 4 chars
MG len5 {S4}                   Display ‘len5’ as string message of up to 4 chars
MG len4 {S4}                   Display ‘len4’ as string message of up to 4 chars
MG len3 {S4}                   Display ‘len3’ as string message of up to 4 chars
MG len2 {S4}                   Display ‘len2’ as string message of up to 4 chars
MG len1 {S4}                   Display ‘len1’ as string message of up to 4 chars
EN
```

This program will accept a string input of up to 6 characters, parse each character, and then display each character. Notice also that the values used for masking are represented in hexadecimal (as denoted by the preceding ‘$’). For more information, see section Sending Messages.

To illustrate further, if the user types in the string “TESTME” at the input prompt, the controller will respond with the following:

```
T                              Response from command MG len6 {S4}
E                              Response from command MG len5 {S4}
S                              Response from command MG len4 {S4}
T                              Response from command MG len3 {S4}
M                              Response from command MG len2 {S4}
E                              Response from command MG len1 {S4}
```

Functions

FUNCTION	DESCRIPTION
@SIN[n]	Sine of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@COS[n]	Cosine of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)

@TAN[n]	Tangent of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@ASIN*[n]	Arc Sine of n, between -90□ and +90□. Angle resolution in 1/64000 degrees.
@ACOS*[n]	Arc Cosine of n, between 0 and 180□. Angle resolution in 1/64000 degrees.
@ATAN*[n]	Arc Tangent of n, between -90□ and +90□. Angle resolution in 1/64000 degrees
@COM[n]	1's Complement of n
@ABS[n]	Absolute value of n
@FRAC[n]	Fraction portion of n
@INT[n]	Integer portion of n
@RND[n]	Round of n (Rounds up if the fractional part of n is .5 or greater)
@SQR[n]	Square root of n (Accuracy is ±.004)
@IN[n]	Return digital input at general input n (where n starts at 1)
@OUT[n]	Return digital output at general output n (where n starts at 1)
@AN[n]	Return analog input at general analog in n (where n starts at 1)

*Note that these functions are multi-valued. An application program may be used to find the correct band.

Functions may be combined with mathematical expressions. The order of execution of mathematical expressions is from left to right and can be over-ridden by using parentheses.

Examples:

```
v1=@ABS[V7]       The variable, v1, is equal to the absolute value of variable v7.
v2=5*@SIN[pos]    The variable, v2, is equal to five times the sine of the variable, pos.
v3=@IN[1]         The variable, v3, is equal to the digital value of input 1.
v4=2*(5+@AN[5])   The variable, v4, is equal to the value of analog input 5 plus 5, then multiplied by 2.
```

Variables

For applications that require a parameter that is variable, the DMC-52xx0 provides 510 variables. These variables can be numbers or strings. A program can be written in which certain parameters, such as position or speed, are defined as variables. The variables can later be assigned by the operator or determined by program calculations. For example, a cut-to-length application may require that a cut length be variable.

Example:

```
posx=5000          Assigns the value of 5000 to the variable posx
PR posx            Assigns variable posx to PR command
JG rpmY*70         Assigns variable rpmY multiplied by 70 to JG command.
```

Programmable Variables

The DMC-52xx0 allows the user to create up to 510 variables. Each variable is defined by a name which can be up to eight characters. The name must start with an alphabetic character; however, numbers are permitted in the rest of the name. Spaces are not permitted. Variable names should not be the same as DMC-52xx0 instructions. For example, PR is not a good choice for a variable name. Also variables are global and available on all banks.

Note: Although upper case variable names are allowed, Galil strongly recommeds using lower-case variable names so there is no confusion between Galil commands and variable names.

Examples of valid and invalid variable names are:

Valid Variable Names

```
posx
pos1
speedZ
```

Invalid Variable Names

```
RealLongName   ; 'Cannot have more than 8 characters
123            ; 'Cannot begin variable name with a number
speed Z        ; 'Cannot have spaces in the name
```

Assigning Values to Variables:

Assigned values can be numbers, internal variables and keywords, functions, controller parameters and strings. The range for numeric variable values is 4 bytes of integer (231) followed by two bytes of fraction (±2,147,483,647.9999).

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

Any valid DMC-52xx0 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)
`bank=_SG`	Set variable to current bank value

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, where:

10 Volts = 3000 rpm = 200000 c/sec

Speed/Analog input = 200000/10 = 20000

`#JOYSTIK`	Label
`SG 1;'`	Select bank 1, axes 9-16.
`JG 0,0;'`	Set in Jog mode for the X and Y axes on the 2nd bank.
`BGXY;'`	Begin Motion for the X and Y axes on the 2nd bank.
`AT0;'`	Set AT time reference
`#LOOP;'`	Loop
`vX=@AN[1]*20000;'`	Read joystick X
`vY=@AN[2]*20000;'`	Read joystick Y
`JG vX,vY;'`	Jog at variable vX,vY for the X and Y axes on the 2nd bank.
`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-52xx0 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-52xx0 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-52xx0 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=_KDZ*2       Assigns value from KDZ 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-52xx0 provides a few additional operands which give access to internal variables that are not accessible by standard DMC-52xx0 commands.

Keyword	Function
_BGn	*Returns a 1 if motion on axis ‘n’ is complete, otherwise returns 0.
_BN	*Returns serial # of the board.
_DA	*Returns the number of arrays available
_DL	*Returns the number of available labels for programming
_DM	*Returns the available array memory
_HMn	*Returns status of Home Switch (equals 0 or 1)
_LFn	Returns status of Forward Limit switch input of axis ‘n’ (equals 0 or 1)
_LRX	Returns status of Reverse Limit switch input of axis ‘n’ (equals 0 or 1)
_UL	*Returns the number of available variables
TIME	Free-Running Real Time Clock (off by 2.4% - Resets with power-on). Note: TIME does not use an underscore character (_) as other keywords.

* - These keywords have corresponding commands while the keywords _LF, _LR, and TIME do not have any associated commands. All keywords are listed in the Command Reference.

Examples of Keywords:

```
v1=_LFX      Assign V1 the logical state of the Forward Limit Switch on the X-axis
v3=TIME      Assign V3 the current value of the time clock
v4=_HMW      Assign V4 the logical state of the Home input on the W-axis
```

Arrays

For storing and collecting numerical data, the DMC-52xx0 provides array space for 24000 elements. The arrays are one dimensional and up to 30 different arrays may be defined. Each array element has a numeric range of 4 bytes of integer (2^{31}) followed by two bytes of fraction (+/-2,147,483,647.9999).

Arrays can be used to capture real-time data, such as position, torque and analog input values. In the contouring mode, arrays are convenient for holding the points of a position trajectory in a record and playback application. Also, array data is global and is available on any bank.

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[10]=_TPX`	Assigns the 10th 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
`SG 0;'`	Select Bank 0, Axes 1-8.
`count=0;DM pos[10];'`	Initialize counter and define array.
`#LOOP;'`	Begin loop.
`WT 10;'`	Wait 10 msec.
`pos[count]=_TPX;'`	Record position of X axis on the 1st bank 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 GalilTools software is recommended for downloading and uploading array data from the controller. The GalilTools Communication library 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-52xx0 provides a special feature for automatic capture of data such as position, position error, inputs or torque. This is useful for teaching motion trajectories or observing system performance. Up to eight types of data can be captured and stored in eight arrays. The capture rate or time interval may be specified. Recording can done as a one time event or as a circular continuous recording.

Note: If axis specific data is being recorded, do not switch banks during recording.

Command Summary - Automatic Data Capture

Command	Description
`RA n[ ],m[ ],o[ ],p[ ]`	Selects up to eight arrays for data capture. The arrays must be defined with the DM command.
`RD type1,type2,type3,type4`	Selects the type of data to be recorded, where type1, type2, type3, and type 4 represent the various types of data (see table below). The order of data type is important and corresponds with the order of n,m,o,p arrays in the RA command.
`RC n,m`	The RC command begins data collection. Sets data capture time interval where n is an integer between 1 and 8 and designates 2^n msec between data. m is optional and specifies the number of elements to be captured. If m is not defined, the number of elements defaults to the smallest array defined by DM. When m is a negative number, the recording is done continuously in a circular manner. _RD is the recording pointer and indicates the address of the next array element. n=0 stops recording.
`RC?`	Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress

Data Types for Recording:

Data type	Description
`TIME`	Controller time as reported by the TIME command
`_AFn`	Analog input (n=X,Y,Z,W,E,F,G,H, for AN inputs 1-8)
`_NOX`	Status bits
`_OP`	Output
`_RLX`	Latched position
`_RPX`	Commanded position
`_SCX`	Stop code
`_TEX`	Position error
`_TI`	Inputs
`_TPX`	Encoder position

_TSX	Switches (only bit 0-4 valid)

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.

```
SG 1                                  Select bank 1, axes 9 – 16.
#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 for the 2nd bank.
PR 10000,20000                        Specify move distance
RC1                                   Start recording now, at rate of 2 msec
BG XY                                 Begin motion for the X and Y axes on the 2nd bank.
#A;JP #A,_RC=1                        Loop until done recording.
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)

Sending Data from a Host

The DMC unit 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 controller are stored with 4 bytes of integer and 2 bytes of fractional data.

Inputting String Variables

String variables with up to six characters may be input using the specifier, {Sn} where n represents the number of string characters to be input. If n is not specified, six characters will be accepted.

The DMC-52xx0, stores all variables as 6 bytes of information. When a variable is specified as a number, the value of the variable is represented as 4 bytes of integer and 2 bytes of fraction. When a variable is specified as a string, the variable can hold up to 6 characters (each ASCII character is 1 byte). When using the IN command for string input, the first input character will be placed in the top byte of the variable and the last character will be placed in the lowest significant byte of the fraction. The characters can be individually separated by using bit-wise operations, see section Bit-wise Operators.

Output of Data (Numeric and String)

Numerical and string data can be output from the controller using several methods. The message command, MG, can output string and numerical data. Also, the controller can be commanded to return the values of variables and arrays, as well as other information using the interrogation commands (the interrogation commands are described in chapter 5).

Sending Messages

Messages may be sent to the bus using the message command, MG. This command sends specified text and numerical or string data from variables or arrays to the screen.

Text strings are specified in quotes and variable or array data is designated by the name of the variable or array. For example:

```
MG "The Final Value is", result
```

In addition to variables, functions and commands, responses can be used in the message command. For example:

```
MG "Analog input is", @AN[1]
MG "The Position of A is", _TPA
```

Specifying the Port for Messages:

The port can be specified with the specifier, {P1} for the main serial port or {En} for the Ethernet port.

```
MG {P1} "Hello World"          Sends message to Main Serial 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 Main Serial Port or Ethernet Port
{$n.m}	Formats numeric values in hexadecimal
{^n}	Sends ASCII character specified by integer n
{N}	Suppresses carriage return/line feed
{Sn}	Sends the first n characters of a string variable, where n is 1 thru 6.

Displaying Variables and Arrays

Variables and arrays may be sent to the screen using the format, variable= or array[x]=. For example, v1= returns the value of v1.

Example - Printing a Variable and an Array element

Instruction	Interpretation
#DISPLAY	Label
SG 0	Select bank 0, axes 1-8.
DM posA[7]	Define Array posA with 7 entries
PR 1000	Position Relative move for X axis on the 1st bank.
BGX	Begin motion for X axis on the 1st bank.
AMX	After Motion for X axis on the 1st bank.
v1=_TPA	Assign Variable v1 to the position for X axis on the 1st bank.
posA[1]=_TPA	Assign the first entry of array to the position for X axis on the 1st bank.
v1=	Print v1

Interrogation Commands

The DMC-52xx0 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
FFFB.00,$0005.00,$0000.00,$0007.00	Response from Interrogation Command

Formatting Variables and Array Elements

The Variable Format (VF) command is used to format variables and array elements. The VF command is specified by:

VF m.n

where m is the number of digits to the left of the decimal point (0 thru 10) and n is the number of digits to the right of the decimal point (0 thru 4).

A negative sign for m specifies hexadecimal format. The default format for VF is VF 10.4

Hex values are returned preceded by a $ and in 2's complement.

Instruction	Interpretation
v1=10	Assign v1
v1=	Return v1

:0000000010.0000	Response - Default format
VF2.2	Change format
v1=	Return v1
:10.00	Response - New format
VF-2.2	Specify hex format
v1=	Return v1
$0A.00	Response - Hex value
VF1	Change format
v1=	Return v1
:9	Response - Overflow

Local Formatting of Variables

PF and VF commands are global format commands that affect the format of all relevant returned values and variables. Variables may also be formatted locally. To format locally, use the command, {Fn.m} or {$n.m} following the variable name and the '=' symbol. F specifies decimal and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal.

Instruction	Interpretation
v1=10	Assign v1
v1=	Return v1
:0000000010.0000	Default Format
v1={F4.2}	Specify local format
:0010.00	New format
v1={$4.2}	Specify hex format
:$000A.00	Hex value
v1="ALPHA"	Assign string "ALPHA" to v1
v1={S4}	Specify string format first 4 characters
:ALPH	

The local format is also used with the MG command.

Converting to User Units

Variables and arithmetic operations make it easy to input data in desired user units such as inches or RPM.

The DMC-52xx0 position parameters such as PR, PA and VP have units of quadrature counts. Speed parameters such as SP, JG and VS have units of counts/sec. Acceleration parameters such as AC, DC, VA and VD have units of counts/sec2. The controller interprets time in milliseconds.

All input parameters must be converted into these units. For example, an operator can be prompted to input a number in revolutions. A program could be used such that the input number is converted into counts by multiplying it by the number of counts/revolution.

Instruction	Interpretation
#RUN	Label
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-52xx0 has an 8-bit uncommitted output port. Each bit on the output port may be set and cleared with the software instructions SB (Set Bit) and CB (Clear Bit), or OB (define output bit). Outputs can be set from any bank.

Example- Set Bit and Clear Bit

Instruction	Interpretation
SB6	Sets bit 6 of output port
CB4	Clears bit 4 of output port

Example- Output Bit

The Output Bit (OB) instruction is useful for setting or clearing outputs depending on the value of a variable, array, input or expression. Any non-zero value results in a set bit.

Instruction	Interpretation
OB1, POS	Set Output 1 if the variable POS is non-zero. Clear Output 1 if POS equals 0.
OB 2, @IN [1]	Set Output 2 if Input 1 is high. If Input 1 is low, clear Output 2.
OB 3, @IN [1]&@IN [2]	Set Output 3 only if Input 1 and Input 2 are high.
OB 4, COUNT [1]	Set Output 4 if element 1 in the array COUNT is non-zero.

The output port can be set by specifying an 16-bit word using the instruction OP (Output Port). This instruction allows a single command to define the state of the entire 16-bit output port, where bit 0 is output 1, bit1 is output2 and so on. A 1 designates that the output is on.

Example Output Port

Instruction	Interpretation
OP6	Sets outputs 2 and 3 of output port to high. All other bits are 0. ($2^1 + 2^2 = 6$)
OP0	Clears all bits of output port to zero
OP 255	Sets all bits of output port to one. ($2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7$)

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

Example - Turn on output after move

Instruction	Interpretation
#OUTPUT;'	Label
SG 0;'	Select bank 0, axes 1-8.
PR 2000;'	Position Command for the A axis on the 1st bank.
BG;'	Begin movie for the A axis on the 1st bank.
AM;'	After move is complete on the A axis on the 1st bank.
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

Instruction	Interpretation
`JP #A,@IN[1]=0`	Jump to A if input 1 is low
`JP #B,@IN[2]=1`	Jump to B if input 2 is high
`AI 7`	Wait until input 7 is high
`AI -6`	Wait until input 6 is low

Example - Start Motion on Switch

Motor A must turn at 4000 counts/sec when the user flips a panel switch to on. When panel switch is turned to off position, motor A must stop turning.

Solution: Connect panel switch to input 1 of DMC-52xx0. High on input 1 means switch is in on position.

Instruction	Interpretation
`SG 1`	Select bank 1, axes 9-16.
`#S;JG 4000`	Set speed for X axis on the 1st bank.
`AI 1;BGA`	Begin motion for the X axis on the 1st bank after input 1 goes high
`AI -1;STA`	Stop motion for X axis on the 1st bank after input 1 goes low
`AMA;JP #S`	After motion is complete for the X axis on the 1st bank, repeat
`EN`	End

Input Interrupt Function

The DMC-52xx0 provides an input interrupt function which causes the program to automatically execute the instructions following the #ININT label. This function is enabled using the II m,n,o command. The m specifies the beginning input and n specifies the final input in the range. The parameter o is an interrupt mask. If m and n are unused, o contains a number with the mask. For example, II,,5 enables inputs 1 and 3.

A low input on any of the specified inputs will cause automatic execution of the #ININT subroutine. The Return from Interrupt (RI) command is used to return from this subroutine to the place in the program where the interrupt had occurred. If it is desired to return to somewhere else in the program after the execution of the #ININT subroutine, the Zero Stack (ZS) command is used followed by unconditional jump statements.

Note: The input interrupt function can only run on local I/O.

IMPORTANT	Use the RI command (not EN) to return from the #ININT subroutine.

Example - Input Interrupt

Instruction	Interpretation
SG 1	Select bank 1, axes 9-16.
#A	Label #A
II 1	Enable input 1 for interrupt function
JG 30000,-20000	Set speeds for A and B axes on the 2nd bank.
BG AB	Begin motion for A and B axes on the 2nd bank.
#B	Label #B
TP AB	Report axes positions for A and B axes on the 2nd bank.
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 for A and B axes on the 2nd bank.
#LOOP;JP #LOOP,@IN[1]=0	Loop until Interrupt cleared
JG 15000,10000	Specify new speeds for A and B axes on the 2nd bank.
WT 300	Wait 300 milliseconds
BG AB	Begin motion for A and B axes on the 2nd bank.
RI	Return from Interrupt subroutine

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.

http://www.galil.com/support/appnotes/optima/note2418.pdf

Analog Inputs

The DMC-52xx0 provides eight analog inputs. The value of these inputs in volts may be read using the @AN[n] function where n is the analog input 1 through 8. The resolution of the Analog-to-Digital conversion is 12 bits (16-bit ADC is available as an option). Analog inputs are useful for reading special sensors such as temperature, tension or pressure.

The following examples show programs which cause the motor to follow an analog signal. The first example is a point-to-point move. The second example shows a continuous move.

Analog Outputs

Example - Position Follower (Point-to-Point)

Objective - The motor must follow an analog signal. When the analog signal varies by 10V, motor must move 10000 counts.

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

Instruction	Interpretation
SG 0	Select bank 0, axes 1-8.
#POINTS	Label
SP 7000	Speed for A axis on the 1st bank.
AC 80000;DC 80000	Acceleration and Deceleration for A axis on the 1st bank.
#LOOP	Label
VP=@AN[1]*1000	Read and analog input, compute position
PA VP	Absolute Position move for the A axis on the 1st bank.
BGA	Start motion for the A axis on the 1st bank.
AMA	After move is completed for the A axis on the 1st bank.
JP #LOOP	Repeat
EN	End

Example - Position Follower (Continuous Move)

Method: Read the analog input, compute the commanded position and the position error. Command the motor to run at a speed in proportions to the position error.

Instruction	Interpretation
SG 0	Select bank 0, axes 1-8.
#CONT	Label
AC 80000;DC 80000	Acceleration and Deceleration for the X axis on the 1st bank.
JG 0	Start jog mode for the X axis on the 1st bank.
BGX	Start motion for the X axis on the 1st bank.
#LOOP	Label
vp=@AN[1]*1000	Compute desired position
ve=vp-_TPA	Find position error for the X axis on the 1st bank.
vel=ve*20	Compute velocity
JG vel	Change velocity for the X axis on the 1st bank.
JP #LOOP	Jump back to repeat the process.
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 therefor 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

Speed Control by Joystick

The speed of a motor is controlled by a joystick. The joystick produces a signal in the range between -10V and +10V. The objective is to drive the motor at a speed proportional to the input voltage.

Assume that a full voltage of 10 Volts must produce a motor speed of 3000 rpm with an encoder resolution of 1000 lines or 4000 count/rev. This speed equals:

3000 rpm = 50 rev/sec = 200000 count/sec

The program reads the input voltage periodically and assigns its value to the variable VIN. To get a speed of 200,000 ct/sec for 10 volts, we select the speed as:

Speed = 20000 x VIN

The corresponding velocity for the motor is assigned to the VEL variable.

Instruction	Function
SG 0	Select bank 0, axes 1-8.
#A	Label
JG0	Jog the X axis on the 1st bank to zero.
BGX	Begin motion for the X axis on the 1st bank.
#B	Label
VIN=@AN[1]	Assign the variable VIN to the value of Analog Input 1.
VEL=VIN*20000	Calculate the velocity variable.
JG VEL	Jog the X axis on the 1st bank at a speed set by the variable VEL.
JP #B	Jump back to #B to repeat the process.
EN	End

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
SG 1	Select bank 1, axes 9-16.
#A	Label
V3=5	Initial position ratio
DP0	Define the starting position of 0 for the X axis on the 1st bank.
JG0	Set motor in jog mode as zero for the X axis on the 1st bank.
BGX	Start motion for the X axis on the 1st bank.
#B	
VIN=@AN[1]	Read analog input 1 and assign the value to the variable VIN
V2=V1*V3	Compute the desired position
V4=V2-_TPX-_TEX	Find the following error for the X axis on the 1st bank.
V5=V4*20	Compute a proportional speed
JG V5	Change the speed for the X axis on the 1st bank to the variable V5.
JP #B	Jump back to #B to repeat the process
EN	End

Using the DMC Editor to Enter Programs

The Galil software package provides an editor and utilities that allow the upload and download of DMC programs to the motion controller.

Application programs for the DMC-52xx0 may also be created and edited locally using the DMC-52xx0.

The DMC-52xx0 provides a line Editor for entering and modifying programs. The Edit mode is entered with the ED instruction. (Note: The ED command can only be given when the controller is in the non-edit mode, which is signified by a colon prompt).

In the Edit Mode, each program line is automatically numbered sequentially starting with 000. If no parameter follows the ED command, the editor prompter will default to the last line of the last program in memory. If desired, the user can edit a specific line number or label by specifying a line number or label following ED.

```
ED                  Puts Editor at end of last program
:ED 5               Puts Editor at line 5
:ED #BEGIN          Puts Editor at label #BEGIN
```

Line numbers appear as 000,001,002 and so on. Program commands are entered following the line numbers. Multiple commands may be given on a single line as long as the total number of characters doesn't exceed 80 characters per line.

While in the Edit Mode, the programmer has access to special instructions for saving, inserting and deleting program lines. These special instructions are listed below:

Edit Mode Commands

<RETURN>

Typing the return key causes the current line of entered instructions to be saved. The editor will automatically advance to the next line. Thus, hitting a series of <RETURN> will cause the editor to advance a series of lines. Note, changes on a program line will not be saved unless a <return> is given.

<cntrl>P

The <cntrl>P command moves the editor to the previous line.

<cntrl>I

The <cntrl>I command inserts a line above the current line. For example, if the editor is at line number 2 and <cntrl>I is applied, a new line will be inserted between lines 1 and 2. This new line will be labeled line 2. The old line number 2 is renumbered as line 3.

<cntrl>D

The <cntrl>D command deletes the line currently being edited. For example, if the editor is at line number 2 and <cntrl>D is applied, line 2 will be deleted. The previous line number 3 is now renumbered as line number 2.

<cntrl>Q

The <cntrl>Q quits the editor mode. In response, the DMC-52xx0 will return a colon.

After the Edit session is over, the user may list the entered program using the LS command. If no operand follows the LS command, the entire program will be listed. The user can start listing at a specific line or label using the operand n. A command and new line number or label following the start listing operand specifies the location at which listing is to stop.

Example:

Instruction	Interpretation
:LS	List entire program
:LS 5	Begin listing at line 5
:LS 5,9	List lines 5 thru 9
:LS #A,9	List line label #A thru line 9
:LS #A, #A +5	List line label #A and additional 5 lines

Chapter 8 Hardware & Software Protection

Introduction

The DMC-52xx0 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-52xx0 is an integral part of the machine, the engineer should design his overall system with protection against a possible component failure on the DMC-52xx0. Galil shall not be liable or responsible for any incidental or consequential damages.

Hardware Protection

The DMC-52xx0 includes hardware input and output protection lines for various error and mechanical limit conditions. These include:

Output Protection Lines

Error Output

The error output is a TTL open-collector 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 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: in Chapter 9 Troubleshooting.

Input Protection Lines

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 function is further discussed in this chapter.

The Abort input by default will also halt program execution; this can be changed by changing the 5th field of the CN command. See the CN command in the command reference for more information.

Software Protection

The DMC-52xx0 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-52xx0. Along with position error and encoder failure detection, then DMC-52xx0 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 200,300,400,500`	Set A-axis error limit for 200, B-axis error limit to 300, C-axis error limit to 400 counts, D-axis error limit to 500 counts
`ER,1,,10`	Set B-axis error limit to 1 count, set D-axis error limit to 10 counts.

The units of the error limit are quadrature counts. The error is the difference between the command position and actual encoder position. If the absolute value of the error exceeds the value specified by ER, the controller will generate several signals to warn the host system of the error condition. These signals include:

Signal or Function	State if Error Occurs
`# POSERR`	Jumps to automatic excess position error subroutine
`Error Light`	Turns on
`OE Function`	Shuts motor off if OE1 or OE3
`AEN Output`	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 A, B, C and D can be monitored during execution using the TE command.

Programmable Position Limits

The DMC-52xx0 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-52xx0 will not accept position commands beyond the limit. Motion beyond the limit is also prevented.

Example:

`DP0,0,0`	Define Position
`BL -2000,-4000,-8000`	Set Reverse position limit
`FL 2000,4000,8000`	Set Forward position limit
`JG 2000,2000,2000`	Jog
`BG ABC`	Begin

(motion stops at forward limits)

Off-On-Error

The DMC-52xx0 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 3 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 1,1,1,1`	Enable off-on-error for A,B,C and D
`OE 0,1,0,1`	Enable off-on-error for B and D axes and disable off-on-error for A and C axes
`OE 2,3`	Enable off-on-error for limit switch for the A axis, and position error (or abort input) and limit switch for the B axis

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. This function will operate across all banks.

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
`STA`	Stop motor
`AMA`	After motor stops
`SHA`	Servo motor here to clear error
`RE`	Return to main program

Limit Switch Routine

The DMC-52xx0 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. This function will operate across all banks.

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. A,B,C, or D 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=_LFA`	Check if forward limit
`V2=_LRA`	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
STA;AMA                 Stop motion
PR-1000;BGA;AMA         Move in reverse
JP#END                  End
#LR                     #LR
MG “REVERSE LIMIT”      Send message
STA;AMA                 Stop motion
PR1000;BGA;AMA          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 are as follows:

1. Error Light (Red LED)

The various symptoms along with the cause and the remedy are described in the following tables.

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 GalilTools 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 MRST 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.

Appendices

Electrical Specifications

Note	Electrical specifications are only valid once controller is out of reset.

Input / Output

Opto-isolated Inputs: DI[8:1], 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 digital inputs use one common voltage (INCOM) which can accept up to 24 volts. Voltages above 24 volts require an additional resistor. ≥ 1 mA = ON; ≤ 0.5 mA = OFF
Analog Inputs: AI[8:1]	±10 volts 12-Bit Analog-to-Digital converter 16-bit optional
Analog Outputs:AO[8:1]	+/-10 volts 12-Bit Digital-to-Analog converter 16-bit optional
Optoisolated Digital Outputs: DO[8:1]	500mA Sourcing per output. 3A total source current.

Power Requirements

90-250 V_{AC} (50-60 Hz)	5W at 25° C

Pinouts

Digital I/O 26 pin HD D-Sub Connector (Male)

Pin	Label	Description	Pin	Label	Description
1	OPA	Digital Output Power	14	DI1	Digital Input 1
2	DO3	Digital Output 3	15	DI4	Digital Input 4
3	DO6	Digital Output 6	16	DI7	Digital Input 7
4	OPB	Digital Output Return	17	RST	Reset Input
5	DI2	Digital Input 2	18	NC	No Connect
6	DI5	Digital Input 5	19	DO1	Digital Output 1
7	DI8	Digital Input 8	20	DO4	Digital Output 4
8	ERROR_C	Error Output Collector	21	DO7	Digital Output 7
9	NC	No Connect	22	INCOM	Digital Input Common
10	OPA	Digital Output Power	23	DI3	Digital Input 3
11	DO2	Digital Output 2	24	DI6	Digital Input 6
12	DO5	Digital Output 4	25	ABRT	Abort Input
13	DO8	Digital Output 8	26	ERROR_E	Error Output Emitter

Analog I/O 26 pin HD D-Sub Connector (Female)

Pin	Label	Description	Pin	Label	Description
1	NC	No Connect	14	AI3	Analog Input 3
2	+12V	Controller +12V	15	GND	Analog Ground
3	AI8	Analog Input 8	16	AO7	Analog Output 7
4	AI5	Analog Input 5	17	AO4	Analog Output 4
5	AI2	Analog Input 2	18	AO1	Analog Output 1
6	GND	Analog Ground	19	NC	No Connect
7	AO6	Analog Output 6	20	-12V	Controller - 12V
8	AO3	Analog Output 3	21	AI7	Analog Input 7
9	GND	Analog Ground	22	AI4	analog Input 4
10	NC	No Connect	23	AI1	Analog Input 1
11	+5V	Controller +5V	24	AO8	Analog Output 8
12	GND	Analog Ground	25	AO5	Analog Output 5
13	AI6	Analog Input 6	26	AO2	Analog Output 2

Performance Specifications

EtherCAT Cycle Time/Memory:

	Normal
EtherCAT Cycle Time	1000µsec
Position Accuracy	±1 quadrature count
Velocity Accuracy	
Long Term	Phase-locked, better than 0.005%
Short Term	System dependent
Position Range	±2147483647 counts per move
Velocity Range	Up to 1,073,741,824 counts/sec for EtherCAT drives
Velocity Resolution	2 counts/sec
Variable Range	±2 billion
Variable Resolution	1×10^{-4}
Number of Variables	510
Array Size	24000 elements, 30 arrays
Program Size	4000 lines x 80 characters
Command Processing	~40 µsec per command

Environmental

Operating Temperature	0-70 deg C
Humidity	20-90% RH, non-condensing

Ordering Options

Overview

The DMC-52xx0 has a single option for upgrading the local analog I/O from 12 bit to 16bit. For information on pricing and ordering a controller with these options, see our DMC-52xx0 part number generator on our website.

http://www.galil.com/order/part-number-generator/dmc-52xx0

-16 bit – 16 bit Analog I/O

The -16 bit option provides 16 bit analog inputs and outputs on the DMC-52xx0 motion controller. The standard resolution of the analog inputs and outputs are 12 bits.

Part number ordering example: DMC-52020(-16bit)

Power Connector Part Numbers

Overview

The DMC-52xx0 uses a standard 3-pin AC power cable and can be plugged into 100-240VAC – 50/60Hz

Input Current Limitations

The current for an optoisolated input shall not exceed 11mA. Some applications may require the use of an external resistor (R) to limit the amount of current for an input. These external resistors can be placed in series between the inputs and their power supply (Vs). To determine if an additional resistor (R) is required, follow Equation A.1: Current limitation requirements for each input below for guidance.

$$1\,mA < \frac{Vs}{R + 2200\,\Omega} < 11\,mA$$

Equation A.1: Current limitation requirements for each input

Serial Cable Connections

XX – USB

The USB port on the DMC-52xx0 is a Female Type B USB port. The standard cable when communicating to a PC will be a Male Type A –> Male Type B USB cable.

Signal Descriptions

Outputs

Error	The signal goes low when the position error on any axis exceeds the value specified by the error limit command, ER.
Output 1-Output 8	The high power optically isolated outputs are uncommitted and may be designated by the user to toggle relays and trigger external events. The output lines are toggled by Set Bit, SB, and Clear Bit, CB, instructions. The OP instruction is used to define the state of all the bits of the Output port.

Inputs

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.
Abort	A low input stops commanded motion instantly without a controlled deceleration. Also aborts motion program
Input 1 - Input 8 (opto-isolated)	Uncommitted inputs. May be defined by the user to trigger events. Inputs are checked with the Conditional Jump instruction and After Input instruction or Input Interrupt.

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 750,000 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 30+ years, Galil has actively shared their market insights in a no-nonsense way for a world of engineers on the move. In fact, over 15,000 engineers have attended Galil seminars. The tradition continues with three different seminars, each designed for your particular skill set-from beginner to the most advanced.

MOTION CONTROL MADE EASY

WHO SHOULD ATTEND

Those who need a basic introduction or refresher on how to successfully implement servo motion control systems.

TIME: 4 hours (8:30 am-12:30 pm)

ADVANCED MOTION CONTROL

WHO SHOULD ATTEND

Those who consider themselves a "servo specialist" and require an in-depth knowledge of motion control systems to ensure outstanding controller performance. Also, prior completion of “Motion Control Made Easy" or equivalent is required. Analysis and design tools as well as several design examples will be provided.

TIME: 8 hours (8:00 am-5:00 pm)

PRODUCT WORKSHOP

WHO SHOULD ATTEND

Current users of Galil motion controllers. Conducted at Galil’s headquarters in Rocklin, CA, students will gain detailed understanding about connecting systems elements, system tuning and motion programming. This is a “hands-on” seminar and students can test their application on actual hardware and review it with Galil specialists.

Attendees must have a current application and recently purchased a Galil controller to attend this course.

TIME: Two days (8:30-4:30pm)

Contacting Us

Galil Motion Control

270 Technology Way

Rocklin, CA 95765

Phone: 916-626-0101

Fax: 916-626-0102

E-Mail Address: support@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.