USER MANUAL

DMC-1600

Manual Rev. 1.0h

By Galil Motion Control, Inc.

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

Rev 8/2011

Using This Manual

Your DMC-1600 motion controller has been designed to work with both servo and stepper type motors. Installation and system setup will vary depending upon whether the controller will be used with stepper motors or servo motors. To make finding the appropriate instructions faster and easier, icons will be next to any information that applies exclusively to one type of system. Otherwise, assume that the instructions apply to all types of systems. The icon legend is shown below.

Attention: Pertains to servo motor use.

Attention: Pertains to stepper motor use.

Please note that many examples are written for the DMC-1640 four-axes controller or the DMC-1680 eight axes controller. Users of the DMC-1630 3-axis controller, DMC-1620 2-axes controller or DMC-1610 1-axis controller should note that the DMC-1630 uses the axes denoted as XYZ, the DMC-1620 uses the axes denoted as XY, and the DMC-1610 uses the X-axis only.

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. Galil shall not be liable or responsible for any incidental or consequential damages.

Contents

Chapter 1 Overview

Introduction

The DMC-1600 series motion control cards install directly into a compact PCI bus. This controller series offers many enhanced features including high speed communications, non-volatile program memory, fast encoder speeds, and improved cabling for EMI reduction.

The DMC-1600 provides two communication channels: a high speed FIFO for sending and receiving commands and a secondary channel which gives high speed access to status and parameters. The DMC-1600 allows for high speed servo control up to 12 million encoder counts/sec and step motor control up to 3 million steps per second. Sample rates as low as 62.5μsec per axis are available.

A 2 Meg Flash EEPROM provides non-volatile memory for storing application programs, parameters, arrays and firmware. New firmware revisions are easily upgraded in the field without removing the controller from the system.

The DMC-1600 can be used with step motors, servo motors, and hydraulics, on any combination of axes. Each axis is configurable by the user for optimum flexibility.

The DMC-1600 achieves superior precision through use of a 16-bit motor command output DAC and a sophisticated PID filter that features velocity and acceleration feedforward an extra pole filter, and integration limits. Designed to solve complex motion problems, the DMC-1600 can be used for applications involving jogging, point-to-point positioning, vector positioning, electronic gearing, multiple move sequences, and contouring. The controller eliminates jerk by programmable acceleration and deceleration with profile smoothing. For smooth following of complex contours, the DMC-1600 provides continuous vector feed of an infinite number of linear and arc segments. The DMC-1600 electronic gearing mode features operation for multiple masters axes as well as gantry.

For synchronization with outside events, the DMC-1600 provides uncommitted I/O, including 8 general use digital inputs, 72 general use digital outputs, and 8 analog inputs for interface to joysticks, sensors, and pressure transducers. Dedicated optoisolated inputs are provided for forward and reverse limits, abort, home, and definable input interrupts. The DMC-1600 is a plug and play device making it easy to set-up. Commands can be sent in either Binary or ASCII. Additional software is available to autotune, view trajectories on a PC screen, translate CAD.DXF files into motion, and create powerful, application-specific operator interfaces with Visual Basic. Drivers for Dos, Windows 3.1, 95, 98, 2000, ME, XP and NT are available.

Overview of Motor Types

The DMC-1600 can provide the following types of motor control:

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

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

Standard Servo Motor with +/- 10 Volt Command Signal

The DMC-1600 achieves superior precision through use of a 16-bit motor command output DAC and a sophisticated PID filter that features velocity and acceleration feedforward, an extra pole filter, and integration limits.

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

Brushless Servo Motor with Sinusoidal Commutation

The DMC-1600 can provide sinusoidal commutation for brushless motors (BLM). In this configuration, the controller generates two sinusoidal signals for connection with amplifiers specifically designed for this purpose.

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

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

To simplify the wiring, the controller provides a one-time, automatic set-up procedure. The parameters determined by this procedure can then be saved in non-volatile memory to be used whenever the system is powered on.

The DMC-1600 can control BLMs equipped with or without Hall sensors. If hall sensors are available, once the controller has been setup, the controller will automatically estimates the commutation phase upon reset. This allows the motor to function immediately upon power up. The Hall effect sensors also provide a method for setting the precise commutation phase. Chapter 2 describes the proper connection and procedure for using sinusoidal commutation of brushless motors.

* 6 Milliseconds per magnetic cycle assumes a servo update of 1 msec (default rate).

Stepper Motor with Step and Direction Signals

The DMC-1600 can control stepper motors. In this mode, the controller provides two signals to connect to the stepper motor: Step and Direction. For stepper motor operation, the controller does not require an encoder and operates the stepper motor in an open loop fashion. Chapter 2 describes the proper connection and procedure for using stepper motors.

DMC-1600 Functional Elements

The DMC-1600 circuitry can be divided into the following functional groups as shown in Figure 1.1 and discussed below.

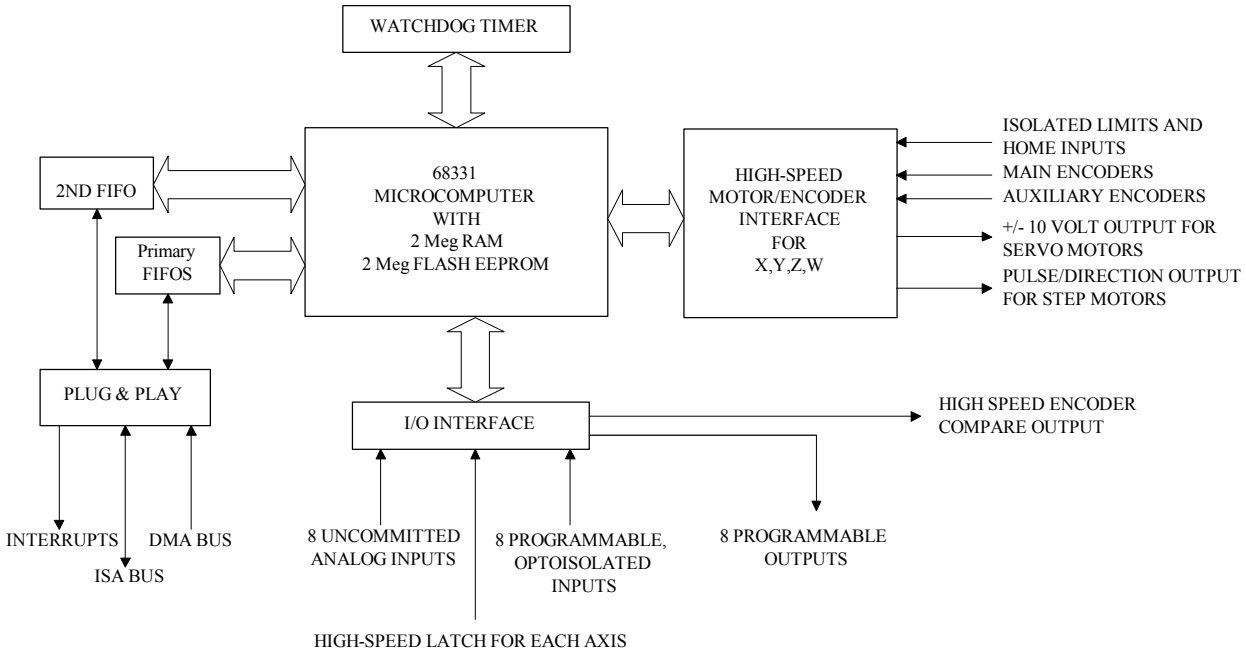


Figure 1.1 - DMC-1600 Functional Elements

Microcomputer Section

The main processing unit of the DMC-1600 is a specialized 32-bit Motorola 68331 Series Microcomputer with 256K RAM and 256K 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. It also contains the DMC-1600 firmware.

Motor Interface

Galil's GL-1800 custom, sub-micron gate array performs quadrature decoding of each encoder at up to 12 MHz, generates a +/-10 Volt analog signal (16 Bit D-to-A) for input to a servo amplifier, and generates step and direction signal for step motor drivers.

Communication

The communication interface with the host PC contains a primary and secondary communication channel. The primary channel uses a bi-directional FIFO (AM470) and includes PC interrupt

handling circuitry. The secondary channel can be enabled where data is placed into the DMC-1600 FIFO buffer.

General I/O

The DMC-1600 provides interface circuitry for 8 bidirectional, optoisolated inputs, 8 TTL outputs and 8 analog inputs with 12-Bit ADC (16-bit optional). The general inputs can also be used as high speed latches for each axes. A high speed encoder compare output is also provided.

System Elements

As shown in Fig. 1.2, the DMC-1600 is part of a motion control system which includes amplifiers, motors and encoders. These elements are described below.

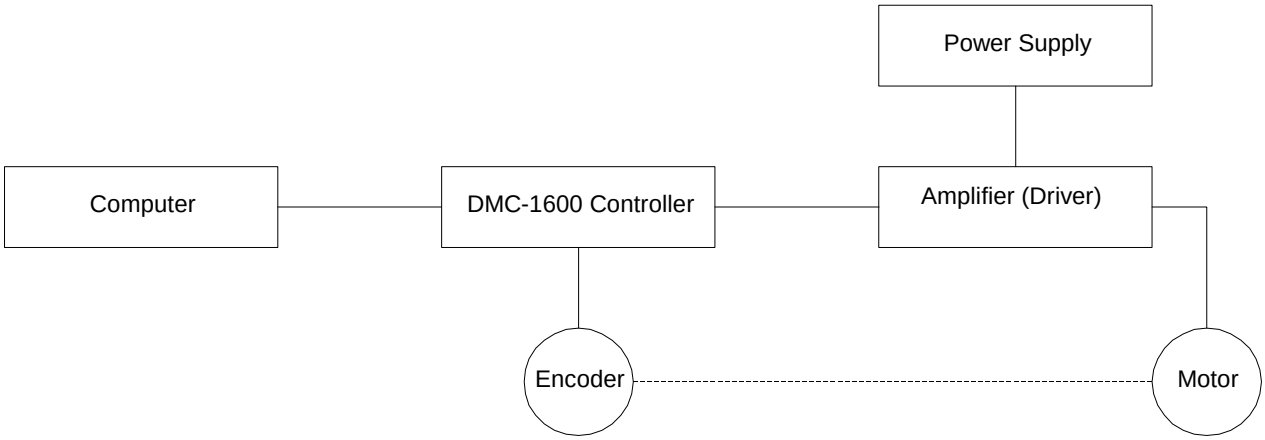


Figure 1.2 - Elements of Servo systems

Motor

A motor converts current into torque which produces motion. Each axis of motion requires a motor sized properly to move the load at the required speed and acceleration. (Galil's "Motion Component Selector" software can help you with motor sizing). Contact Galil at 800-377-6329 if you would like this product.

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

Amplifier (Driver)

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

Encoder

An encoder translates motion into electrical pulses which are fed back into the controller. The DMC-1600 accepts feedback from either a rotary or linear encoder. Typical encoders provide two channels in quadrature, known as CHA and CHB. This type of encoder is known as a quadrature encoder. Quadrature encoders may be either single-ended (CHA and CHB) or differential (CHA,CHA-,CHB,CHB-). The DMC-1600 decodes either type into quadrature states or four times the number of cycles. Encoders may also have a third channel (or index) for synchronization.

For stepper motors, the DMC-1600 can also interface to encoders with pulse and direction signals.

There is no limit on encoder line density, however, the input frequency to the controller must not exceed 3,000,000 full encoder cycles/second (12,000,000 quadrature counts/sec). For example, if the encoder line density is 10000 cycles per inch, the maximum speed is 300 inches/second. If higher encoder frequency is required, please consult the factory.

The standard voltage level is TTL (zero to five volts), however, voltage levels up to 12 Volts are acceptable. (If using differential signals, 12 Volts can be input directly to the DMC-1600. Single-ended 12 Volt signals require a bias voltage input to the complementary inputs).

The DMC-1600 can accept analog feedback instead of an encoder for any axis. For more information see description of analog feedback in Chapter 2 under section entitled "Test the encoder operation".

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

Watch Dog Timer

The DMC-1600 provides an internal watch dog timer which checks for proper microprocessor operation. The timer toggles the Amplifier Enable Output (AEN) which can be used to switch the amplifiers off in the event of a serious DMC-1600 failure. The AEN output is normally high. During power-up and if the microprocessor ceases to function properly, the AEN output will go low. The error light for each axis will also turn on at this stage. A reset is required to restore the DMC-1600 to normal operation. Consult the factory for a Return Materials Authorization (RMA) Number if your DMC-1600 is damaged.

THIS PAGE LEFT BLANK INTENTIONALLY

Chapter 2 Getting Started

The DMC-1600 Motion Controller

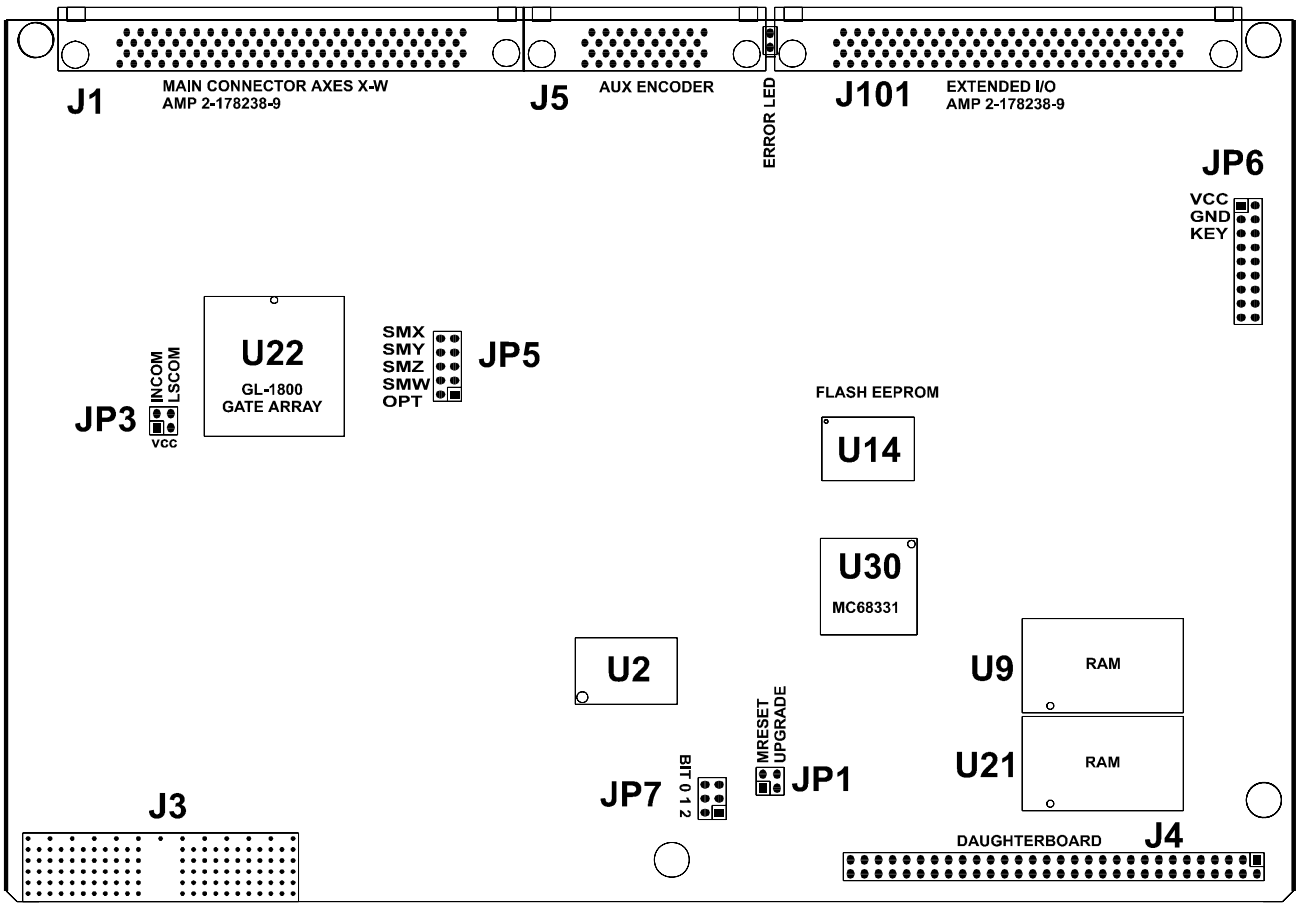


Figure 2-1 - Outline of the DMC-1600

U14	Flash EEPROM	JP1	Master Reset & UPGRD jumpers
U9/U21	RAM	JP3	INCOM & LSCOM jumpers. Used for bypassing opto-isolation for the limit, home, and abort switches and the digital inputs IN1 - IN8. See section "Bypassing Opto-Isolation", Chap3.
U30	Motorola 68331 microprocessor	JP5	Jumpers used for configuring stepper motor operation on axes X-W
U22	GL-1800 custom gate array	JP6	?
J1	100-pin high density connector (Axes X-W) (Part number Amp #2-178238-9)	JP7	?
J5	36-pin high density connector for the auxiliary encoder signals.		
J101	100-pin high density connector for extended I/O (Part number Amp #2-178238-9)		

Elements You Need

Before you start, you must get all the necessary system elements. These include:

1. DMC-1610, 1620, 1630, or DMC-1640 Motion Controller, (1) 100-pin cable, and (1) ICM-1900 interconnect module.
2. Servo motors with Optical Encoder (one per axis) or step motors.
3. Power Amplifiers.
4. Power Supply for Amplifiers.
5. PC (Personal Computer - ISA bus).
6. Utilities Disk for DMC-1600 which contains Plug and Play drivers, and communication utilities.
7. WSDK-16 or WSDK-32 is optional but recommend for first time users.

The motors may be servo (brush type or brushless) or steppers. The amplifiers should be suitable for the motor and may be linear or pulse-width-modulated. An amplifier may have current feedback, voltage feedback or velocity feedback.

For servo motors in current mode, the amplifiers should accept an analog signal in the +/-10 Volt range as a command. The amplifier gain should be set so that a +10V command will generate the maximum required current. For example, if the motor peak current is 10A, the amplifier gain should be 1 A/V. For velocity mode amplifiers, a command signal of 10 Volts should run the motor at the maximum required speed. Set the velocity gain so that an input signal of 10V, runs the motor at the maximum required speed.

For step motors, the amplifiers should accept step and direction signals. For start-up of a step motor system refer to "Connecting Step Motors" on page 20.

The WSDK software is highly recommended for first time users of the DMC-1600. It provides step-by-step instructions for system connection, tuning and analysis.

Installing the DMC-1600

Installation of a complete, operational DMC-1600 system consists of 9 steps.

Step 1. Determine overall motor configuration.

Step 2. Install Jumpers on the DMC-1600.

Step 3. Install the communications software.

Step 4. Install the DMC-1600 in the PC.

Step 5. Establish communications with the Galil Communication Software.

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

Step 7. Make connections to amplifier and encoder.

Step 8a. Connect standard servo motors.

Step 8b. Connect sinusoidal commutation motors

Step 8c. Connect step motors.

Step 9. Tune the servo system

Step 1. Determine Overall Motor Configuration

Before setting up the motion control system, the user must determine the desired motor configuration. The DMC-1600 can control any combination of standard servo motors, sinusoidally commutated brushless motors, and stepper motors. Other types of actuators, such as hydraulics can also be controlled, please consult Galil.

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

Standard Servo Motor Operation:

The DMC-1600 has been setup by the factory for standard servo motor operation providing an analog command signal of +/- 10V. No hardware or software configuration is required for standard servo motor operation.

Sinusoidal Commutation:

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

Each sinusoidally commutated motor requires two DAC's. In standard servo operation, the DMC-1600 has one DAC per axis. In order to have the additional DAC for sinusoidal commutation, the controller must be designated as having one additional axis for each sinusoidal commutation axis. For example, to control two standard servo axes and one axis of sinusoidal commutation, the controller will require a total of four DAC's and the controller must be a DMC-1640.

Sinusoidal commutation is configured with the command, BA. For example, BAX sets the X axis to be sinusoidally commutated. The second DAC for the sinusoidal signal will be the highest available DAC on the controller. For example: Using a DMC-1640, the command BAX will configure the X axis to be the main sinusoidal signal and the 'W' axis to be the second sinusoidal signal.

The BA command also reconfigures the controller to indicate that the controller has one less axis of 'standard' control for each axis of sinusoidal commutation. For example, if the command BAX is given to a DMC-1640 controller, the controller will be re-configured to a DMC-1630 controller. By definition, a DMC-1630 controls 3 axes: X,Y and Z. The 'W' axis is no longer available since the output DAC is being used for sinusoidal commutation.

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

Stepper Motor Operation:

To configure the DMC-1600 for stepper motor operation, the controller requires a jumper for each stepper motor and the command, MT, must be given. The installation of the stepper motor jumper is discussed in the following section entitled "Installing Jumpers on the DMC-1600". Further instructions for stepper motor connections are discussed in Step 8c.

Step 2. Install Jumpers on the DMC-1600

Master Reset and Upgrade Jumpers

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

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

Opto-Isolation Jumpers

The inputs and limit switches are optoisolated. If you are not using an isolated power supply, the internal +5V supply from the PC may be used to power the optoisolators. This is done by installing the jumpers on JP3. LSCOM to VCC will power the limit switch optoisolators, and INCOM to VCC will power the general input optoisolators.

Note: Using the PC's internal +5V supply to power the optoisolators will effectively bypass the optoisolation and fail to provide any isolation for the inputs and limit switches.

Stepper Motor Jumpers

For each axis that will be used for stepper motor operation, the corresponding stepper mode (SM) jumper must be connected. The stepper motor jumpers, labeled JP5 for axes X through W are located directly beside the GL-1800 IC's on the main board (see the diagram for the DMC-1600). The individual jumpers are labeled SMX, SMY, SMZ and SMW for axes 1 through 4.

(Optional) Motor Off Jumper

The state of the motor upon power up may be selected with the placement of a hardware jumper on the controller. With a jumper installed at the OPT location (next to the stepper motor jumpers), the controller will be powered up in the 'motor off' state. The SH command will then need to be issued in order for the motor(s) to be enabled. With no jumper installed, the controller will immediately enable the motor(s) upon power up.

Step 3. Install the Communications Software

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

Using Dos:

Using the Galil Software CD-ROM, go to the directory DMCDOS. Type "INSTALL" at the DOS prompt and follow the directions.

Using Windows SE, 98, NT, ME, XP or 2000 (32 bit versions):

The Galil Software CD-ROM will automatically begin the installation procedure when the CD-ROM is installed. After installing the Galil CD-ROM software on your computer, you can easily install other software components as desired. To install the basic communications software, run the Galil Software CD-ROM and choose "DMCWIN32 – Windows Utilities and Programming Libraries, WIN95, 98, NT". This will install the Galil Terminal that can be used for communication.

Step 4. Install the DMC-1600 in the PC

The DMC-1600 is installed directly into the Compact PCI bus. The procedure is outlined below.

1. Make sure the host computer is in the power-off condition.
2. Remove unit cover.
3. Remove the metal plate covering the expansion bus slot where the DMC-1600 will be inserted.
4. Insert DMC-1600 card in the expansion bus and secure with screw.
5. Attach 100-pin cable to your controller card. If you are using a Galil ICM-1900 or AMP-19X0, this cable connects into the J2 connection on the interconnect module. If you are not using a Galil interconnect module, you will need to appropriately terminate the cable to your system components, see the appendix for cable pin outs. The auxiliary encoder connections are accessed through the 36-pin connector, J5.

Note: The part number for the 100-pin connector is #2-178238-9 from AMP.

6. Turn power on to computer.
7. The operating system should recognize the DMC-1600 as a new device.
8. The Plug and Play feature will automatically configure the controller for your computers available resources. The installation will also automatically add this information to the Galil Registry.

Step 5. Establish Communication using Galil Software

DMC-1600 DOS Users

All that is required for installation of the DMC-1600 in DOS is the DMCTERM software. Within DMCTERM, the DMC-1600 should be selected from the card options, and the DOS drivers will automatically register the card with a correct address and IRQ.

After providing the setup information, the terminal should indicate, “Attempting to connect to controller”, followed by the colon “:” being sent. This indicates a successful connection. Note: The BIOS for your PC when using DOS should be set for Non-Plug and Play OS for successful communication.

Using Windows98SE, ME, NT, 2000, XP:

In order for the windows software to communicate with a Galil controller, the controller must be registered in the Galil Registry, listing the controller information and address. Under Windows NT/2000, the DMC-1600 controller is registered by initializing the Galil Driver. To do this, install the DTERM32 software program (DMCWIN32.exe) or the Galil Servo Design Kit Software (WSDK32.exe). Begin the software and add the DMC-1600 controller to the registry. The registry is available from the pull down menu, Registry, from DTERM32 and from the pull down menu, File, from the WSDK software. After registering the controller, re-boot the system. Under the Options Menu, select, Start Device Driver. Then select the terminal to begin communicating directly with the controller.

Dterm32 will find the DMC-1600 controller and automatically insert the controller information into the registry. The registry entry also displays timeout and delay information. These are advanced parameters that should only be modified by advanced users (see software documentation for more information).

Communication is established by selecting the Terminal Menu. Select the controller by highlighting it. Once the entry has been selected, click on the **OK** button. If communication is established, the terminal window will open and a colon prompt will be displayed. From the top line of the terminal, commands can be sent to the controller. Command syntax is described in the Command Reference and later sections.

If you are not properly communicating with the controller, the program will pause for 3-15 seconds. The top of the screen will display the message “Status: not connected with Galil motion controller” and the following error will appear: “STOP - Unable to establish communication with the Galil controller. A time-out occurred while waiting for a response from the Galil controller.” If this message appears, you must click OK. Contact Galil if you are unable to communicate with your controller.

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

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

The command, BA is used to select the axes of sinusoidal commutation. For example, BAXZ sets X and Z as axes with sinusoidal commutation.

Notes on Configuring Sinusoidal Commutation:

The command, BA, reconfigures the controller such that it has one less axis of 'standard' control for each axis of sinusoidal commutation. For example, if the command BAX is given to a DMC-1640 controller, the controller will be re-configured to be a DMC-1630 controller. In this case the highest axis is no longer available except to be used for the 2nd phase of the sinusoidal commutation. Note that the highest axis on a controller can never be configured for sinusoidal commutation.

The first phase signal is the motor command signal. The second phase is derived from the highest DACX on the controller. When more than one axis is configured for sinusoidal commutation, the highest sinusoidal commutation axis will be assigned to the highest DAC and the lowest sinusoidal commutation axis will be assigned to the lowest available DAC. Note the lowest axis is the X axis.

Example: Sinusoidal Commutation Configuration using a DMC-1640

BAXY

This command causes the controller to be reconfigured as a DMC-1620 controller. The X and Y axes are configured for sinusoidal commutation. The first phase of the X axis will be the motor command X signal. The second phase of the X axis will be Z signal. The first phase of the Y axis will be the motor command Y signal. The second phase of the Y axis will be the motor command W signal.

Step 7. Make Connections to Amplifier and Encoder.

Once you have established communications between the software and the DMC-1600, you are ready to connect the rest of the motion control system. The motion control system typically consists of an ICM-1900 Interface Module, an amplifier for each axis of motion, and a motor to transform the current from the amplifier into torque for motion. Galil also offers the AMP-19X0 series Interface Modules, which are ICM-1900's equipped with servo amplifiers for brush type DC motors.

If you are using an ICM-1900, connect the 100-pin ribbon cable to the DMC-1600 and to the connector located on the AMP-19x0 or ICM-1900 board. The ICM-1900 provides screw terminals for access to the connections described in the following discussion.

System connection procedures will depend on system components and motor types. Any combination of motor types can be used with the DMC-1600. If sinusoidal commutation is to be used, special attention must be paid to the reconfiguration of axes.

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

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

Step B. Connect the amplifier enable signal.

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

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

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

The amplifier enable signal is used by the controller to disable the motor. This signal is labeled AMPENX for the X axis on the ICM-1900 and should be connected to the enable signal on the amplifier. Note that many amplifiers designate this signal as the INHIBIT signal. Use the command, MO, to disable the motor amplifiers - check to insure that the motor amplifiers have been disabled (often this is indicated by an LED on the amplifier).

This signal changes under the following conditions: the watchdog timer activates, the motor-off command, MO, is given, or the OE1 command (Enable Off-On-Error) is given and the position error exceeds the error limit. As shown in Figure 3-4, AEN can be used to disable the amplifier for these conditions.

The standard configuration of the AEN signal is TTL active high. In other words, the AEN signal will be high when the controller expects the amplifier to be enabled. The polarity and the amplitude can be changed if you are using the ICM-1900 interface board. To change the polarity from active high (5 volts = enable, zero volts = disable) to active low (zero volts = enable, 5 volts = disable), replace the 7407 IC with a 7406. Note that many amplifiers designate the enable input as 'inhibit'.

To change the voltage level of the AEN signal, note the state of the resistor pack on the ICM-1900. When Pin 1 is on the 5V mark, the output voltage is 0-5V. To change to 12 volts, pull the resistor pack and rotate it so that Pin 1 is on the 12 volt side. If you remove the resistor pack, the output signal is an open collector, allowing the user to connect an external supply with voltages up to 24V. The user should provide current limiting resistors in this case.

Step C. Connect the encoders

For stepper motor operation, an encoder is optional.

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

The DMC-1600 accepts single-ended or differential encoder feedback with or without an index pulse. If you are not using the AMP-19x0 or the ICM-1900 you will need to consult the appendix for the encoder pinouts for connection to the motion controller. The AMP-19x0 and the ICM-1900 can accept encoder feedback from a 10-pin ribbon cable or individual signal leads. For a 10-pin ribbon cable encoder, connect the cable to the protected header connector labeled X ENCODER (repeat for each axis necessary). For individual wires, simply match the leads from the encoder you are using to the encoder feedback inputs on the interconnect board. The signal leads are labeled CHA (channel A), CHB (channel B), and INDEX. For differential encoders, the complement signals are labeled CHA-, CHB-, and INDEX-.

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

Step D. Verify proper encoder operation.

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

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

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

Step E. Connect Hall Sensors if available.

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

The Hall effect sensors are connected to the digital inputs of the controller. These inputs can be used with the general use inputs (bits 1-8), the auxiliary encoder inputs (bits 81-96), or the extended I/O inputs of the DMC-1600 controller (bits 17-80). Note: The general use inputs are optoisolated and require a voltage connection at the INCOM point - for more information regarding the digital inputs, see Chapter 3, *Connecting Hardware*.

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

BI ,, 6

or

BIZ = 6

Step 8a. Connect Standard Servo Motors

The following discussion applies to connecting the DMC-1600 controller to standard servo motor amplifiers:

The motor and the amplifier may be configured in the torque or the velocity mode. In the torque mode, the amplifier gain should be such that a 10 Volt signal generates the maximum required current. In the velocity mode, a command signal of 10 Volts should run the motor at the maximum required speed.

Step by step directions on servo system setup are also included on the WSDK (Windows Servo Design Kit) software offered by Galil. See section on WSDK for more details.

Check the Polarity of the Feedback Loop

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

Step A. Set the Error Limit as a Safety Precaution

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

Input the commands:

ER 2000 <CR> Sets error limit on the X axis to be 2000 encoder counts

OE 1 <CR> Disables X axis amplifier when excess position error exists

If the motor runs away and creates a position error of 2000 counts, the motor amplifier will be disabled. **Note:** This function requires the AEN signal to be connected from the controller to the amplifier.

Step B. Set Torque Limit as a Safety Precaution

To limit the maximum voltage signal to your amplifier, the DMC-1600 controller has a torque limit command, TL. This command sets the maximum voltage output of the controller and can be used to avoid excessive torque or speed when initially setting up a servo system.

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

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

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

TL 1 <CR>

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

Step C. Enable Off-On-Error as a safety precaution. To limit the maximum distance the motor will move from the commanded position, enable the Off-On-Error function using the command , OE 1. If the motor runs away due to positive feedback or another systematic problem the controller will disable the amplifier when the position error exceeds the value set by the command, ER.

Step D. Disable motor with the command MO (Motor off).

Step E. Connect the Motor and issue SH

Once the parameters have been set, connect the analog motor command signal (ACMD) to the amplifier input.

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

PR 1000 <CR> Position relative 1000 counts

BGX <CR> Begin motion on X axis

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

Inverting the Loop Polarity

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

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

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

TTX (CR) Tell torque on X

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

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

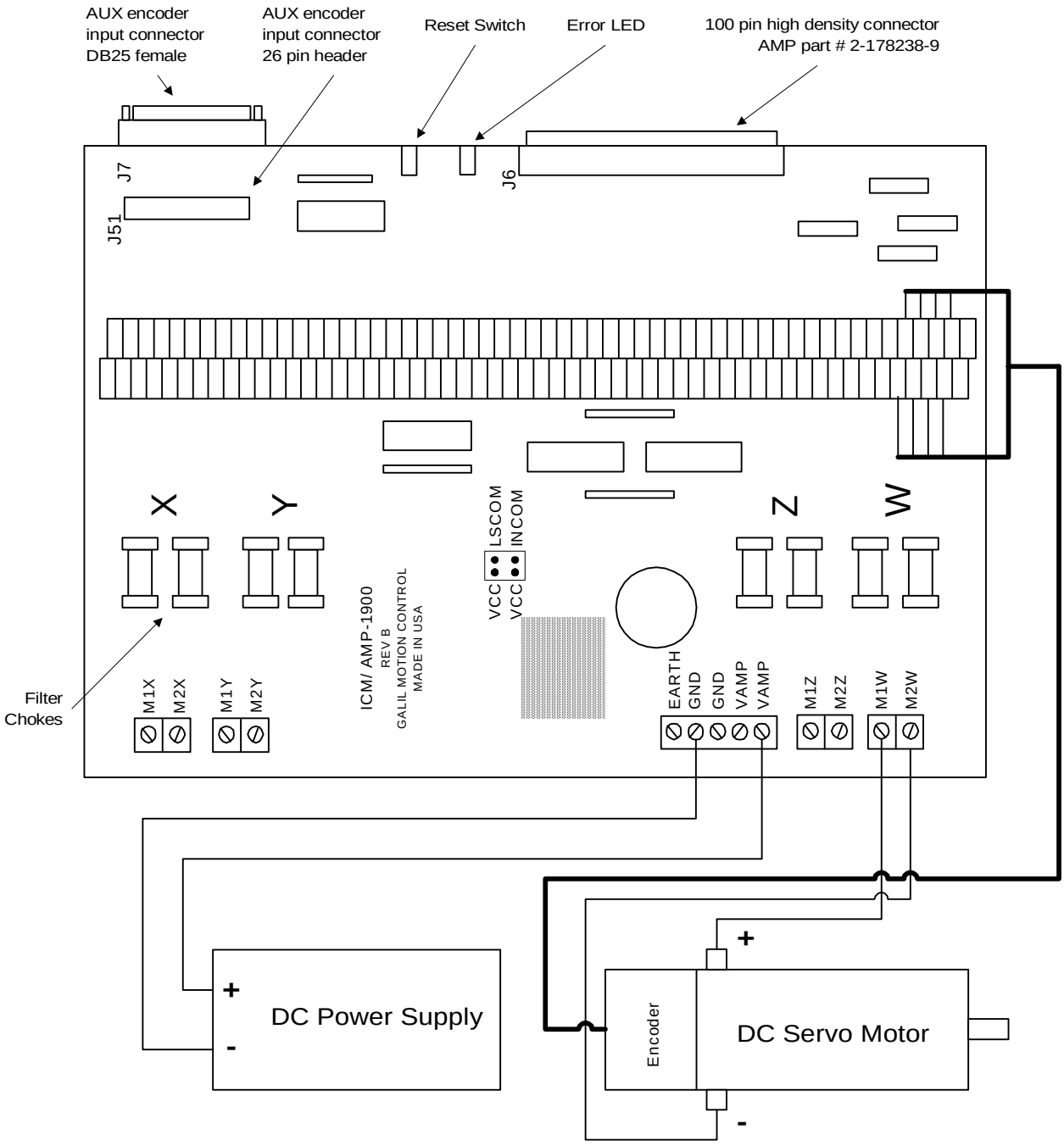


Figure 2-2 - System Connections with the AMP-1900Amplifier. Note: this figure shows a Galil Motor and Encoder which uses a flat ribbon cable for connection to the AMP-1900 unit.

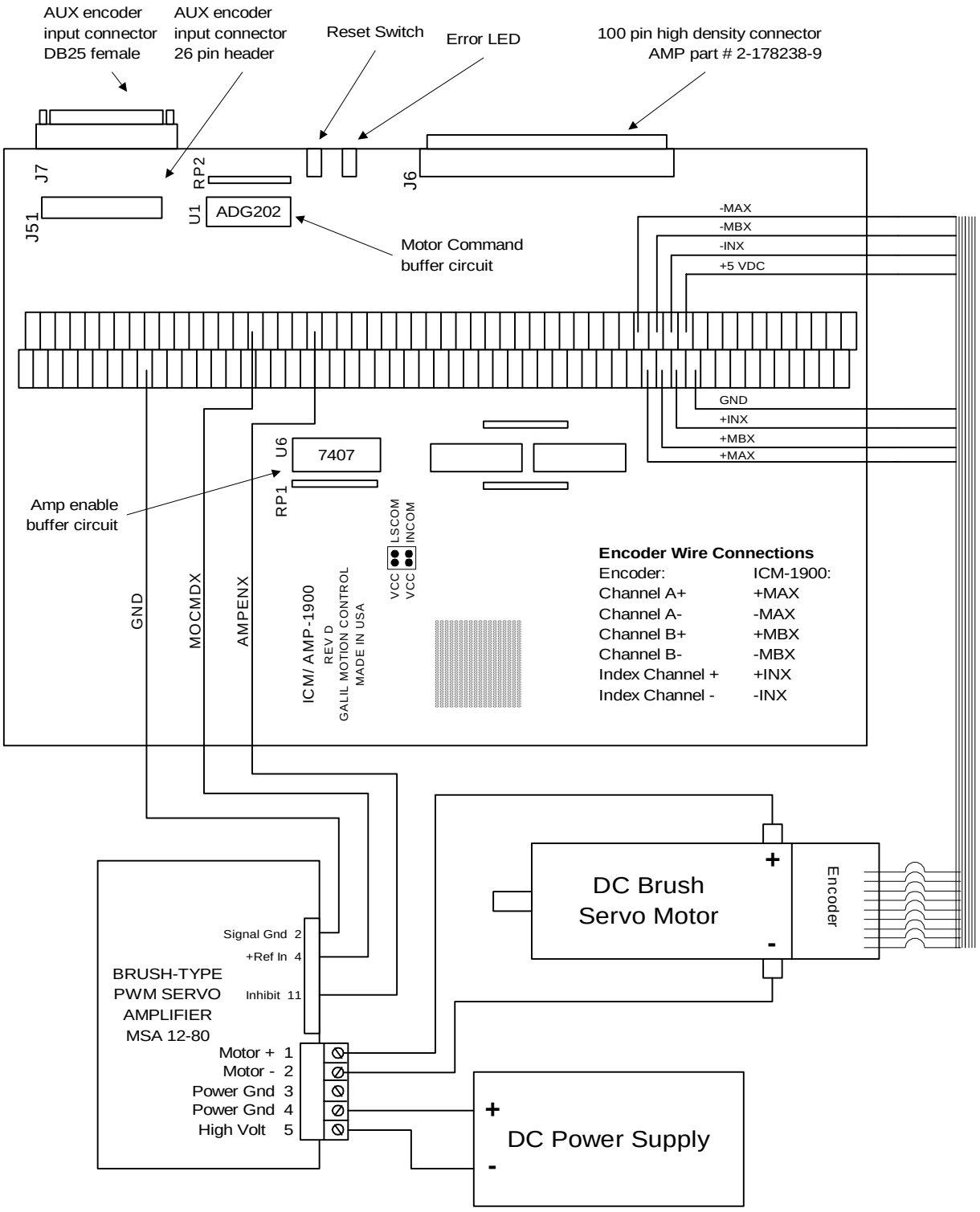


Figure 2-3 System Connections with a separate amplifier (MSA 12-80). This diagram shows the connections for a standard DC Servo Motor and encoder

Step 8b. Connect Sinusoidal Commutation Motors

When using sinusoidal commutation, the parameters for the commutation must be determined and saved in the controller's non-volatile memory. The servo can then be tuned as described in Step 9.

Step A. Disable the motor amplifier

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

Step B. Connect the motor amplifier to the controller.

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

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

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

Example: Sinusoidal Commutation Configuration using a DMC-1640

BAXY

This command causes the controller to be reconfigured as a DMC-1620 controller. The X and Y axes are configured for sinusoidal commutation. The first phase of the X axis will be the motor command X signal. The second phase of the X axis will be the motor command Z signal. The first phase of the Y axis will be the motor command Y signal. The second phase of the Y axis will be the motor command W signal.

Step C. Specify the Size of the Magnetic Cycle.

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

BM 62000

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

BM,, 1333.333

Step D. Test the Polarity of the DACs and Hall Sensor Configuration.

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

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

BSX = 2,700

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

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

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

If Hall Sensors are Available:

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

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

Step E. Save Brushless Motor Configuration

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

If Hall Sensors are Not Available:

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

Step F. Set Zero Commutation Phase

When an axis has been defined as sinusoidally commutated, the controller must have an estimate for commutation phase. When hall sensors are used, the controller automatically estimates this value upon reset of the controller. If no hall sensors are used, the controller will not be able to make this estimate and the commutation phase must be set before enabling the motor.

If Hall Sensors are Not Available:

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

The BZ command argument is a real number, which represents the voltage to be applied to the amplifier during the initialization. When the voltage is specified by a positive number, the initialization process will end in the motor off (MO) state. A negative number causes the process to end in the Servo Here (SH) state.

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

For example, BZ -2

will drive the X axis to zero, using a 2V signal. The controller will then leave the motor enabled. For systems that have external forces working against the motor, such as gravity, the BZ argument must provide a torque 10x the external force. If the torque is not sufficient, the commutation zero may not be accurate.

If Hall Sensors are Available:

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

Method 1. Use the BZ command as described above.

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

SHX	Enable X axis motor
PRX=-1*(_BZX)	Move X motor close to zero commutation phase
BGX	Begin motion on X axis
AMX	Wait for motion to complete on X axis
BZX=-1	Drive motor to commutation phase zero and leave the motor on

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

SHX	Enable X axis motor
BCX	Enable the brushless calibration command
PRX=50000	Command a relative position movement on X axis
BGX	Begin motion on X axis. When the hall sensors detect a phase transition, the commutation phase is re-set.

Step 8c. Connect Step Motors

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

The frequency of the step motor pulses can be smoothed with the filter parameter, KS. The KS parameter has a range between 0.5 and 8, where 8 implies the largest amount of smoothing. *See Command Reference regarding KS*.

The DMC-1600 profiler commands the step motor amplifier. All DMC-1600 motion commands apply such as PR, PA, VP, CR and JG. The acceleration, deceleration, slew speed and smoothing are also used. Since step motors run open-loop, the PID filter does not function and the position error is not generated.

To connect step motors with the DMC-1600 you must follow this procedure:

Step A. Install SM jumpers

Each axis of the DMC-1600 that will operate a stepper motor must have the corresponding stepper motor jumper installed. For a discussion of SM jumpers, see Step 2.

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

from the controller to respective signals on your step motor amplifier. (These signals are labeled PULSX and DIRX for the x-axis on the ICM-1900). Consult the documentation for your step motor amplifier.

Step C. Configure DMC-1600 for motor type using MT command. You can configure the DMC-1600 for active high or active low pulses. Use the command MT 2 for active high step motor pulses and MT -2 for active low step motor pulses. *See description of the MT command in the Command Reference.*

Step 9. Tune the Servo System

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

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

To start, set the integrator to zero with the instruction

KI 0 (CR) Integrator gain

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

KP 1 (CR) Proportional gain

KD 100 (CR) Derivative gain

For more damping, you can increase KD (maximum is 4095). Increase gradually and stop after the motor vibrates. A vibration is noticed by audible sound or by interrogation. If you send the command

TE X (CR) Tell error

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

Next you need to increase the value of KP gradually (maximum allowed is 1023). You can monitor the improvement in the response with the Tell Error instruction

KP 10 (CR) Proportion gain

TE X (CR) Tell error

As the proportional gain is increased, the error decreases.

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

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

TE X (CR)

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

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

Design Examples

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

Example 1 - System Set-up

This example shows various ways to assign parameters.

Instruction	Interpretation
KP10,10,10,10	Set gains for a,b,c,d (or X,Y,Z,W axes)
KP*=10	Alternate method for setting gain on all axes
KPX=10	Alternate method for setting X (or A) axis gain
KPA=10	Alternate method for setting A (or X) axis gain
KP, 20	Set Y axis gain only

Example 2 - Profiled Move

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

Instruction	Interpretation
PR 10000	Distance
SP 20000	Speed
DC 100000	Deceleration
AC 100000	Acceleration
BG X	Start Motion

Example 3 - Multiple Axes

Objective: Move the four axes independently.

Instruction	Interpretation
PR 500,1000,600,-400	Distances of X,Y,Z,W
SP 10000,12000,20000,10000	Slew speeds of X,Y,Z,W
AC 100000,10000,100000,100000	Accelerations of X,Y,Z,W
DC 80000,40000,30000,50000	Decelerations of X,Y,Z,W
BG XZ	Start X and Z motion
BG YW	Start Y and W motion

Example 4 - Independent Moves

The motion parameters may be specified independently as illustrated below.

Instruction	Interpretation
PR ,300,-600	Distances of Y and Z
SP ,2000	Slew speed of Y
DC ,80000	Deceleration of Y
AC, 100000	Acceleration of Y
SP ,,40000	Slew speed of Z
AC ,,100000	Acceleration of Z
DC ,,150000	Deceleration of Z
BG Z	Start Z motion
BG Y	Start Y motion

Example 5 - Position Interrogation

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

Instruction	Interpretation
TP	Tell position all four axes
TP X	Tell position - X axis only
TP Y	Tell position - Y axis only
TP Z	Tell position - Z axis only
TP W	Tell position - W axis only

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

Instruction	Interpretation
TE	Tell error - all axes
TE X	Tell error - X axis only
TE Y	Tell error - Y axis only
TE Z	Tell error - Z axis only
TE W	Tell error - W axis only

Example 6 - Absolute Position

Objective: Command motion by specifying the absolute position.

Instruction	Interpretation
DP 0,2000	Define the current positions of X,Y as 0 and 2000
PA 7000,4000	Sets the desired absolute positions
BG X	Start X motion
BG Y	Start Y motion

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

PA 0,0	Move to 0,0
BG XY	Start both motions

Example 7 - Velocity Control

Objective: Drive the X and Y motors at specified speeds.

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

after a few seconds, command:

JG -40000	New X speed and Direction
TV X	Returns X speed

and then

JG ,20000	New Y speed
TV Y	Returns Y speed

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

ST	Stop

Example 8 - Operation Under Torque Limit

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

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

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

Increase the torque level gradually by instructions such as

Instruction	Interpretation
TL 1.0	Increase torque limit to 1 volt.
TL 9.98	Increase torque limit to maximum, 9.98 Volts.

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

Example 9 - Interrogation

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

Instruction	Interpretation
KP ?	Return gain of X axis.
KP ,,?	Return gain of Z axis.
KP ?,?,?,?	Return gains of all axes.

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

Example 10 - Operation in the Buffer Mode

The instructions may be buffered before execution as shown below.

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

Example 11 - Using the On-Board Editor

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

The instruction

ED	Edit mode

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

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

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

XQ #A	Start the program running

If the ED command is issued from the Galil Windows terminal software (such as DTERM32), the software will open a Windows based editor. From this editor a program can be entered, edited, downloaded and uploaded to the controller.

Example 12 - Motion Programs with Loops

Motion programs may include conditional jumps as shown below.

Instruction	Interpretation
#A	Label

DP 0	Define current position as zero
V1=1000	Set initial value of V1
#Loop	Label for loop
PA V1	Move X motor V1 counts
BG X	Start X motion
AM X	After X motion is complete
WT 500	Wait 500 ms
TP X	Tell position X
V1=V1+1000	Increase the value of V1
JP #Loop,V1<10001	Repeat if V1<10001
EN	End

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

XQ #A	Execute Program #A

Example 13 - Motion Programs with Trippoints

The motion programs may include trippoints as shown below.

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

To start the program, command:

XQ #B	Execute Program #B

Example 14 - Control Variables

Objective: To show how control variables may be utilized.

Instruction	Interpretation
#A;DP0	Label; Define current position as zero
PR 4000	Initial position
SP 2000	Set speed
BGX	Move X
AMX	Wait until move is complete
WT 500	Wait 500 ms
#B	

V1 = _TPX	Determine distance to zero
PR -V1/2	Command X move 1/2 the distance
BGX	Start X motion
AMX	After X moved
WT 500	Wait 500 ms
V1=	Report the value of V1
JP #C, V1=0	Exit if position=0
JP #B	Repeat otherwise
#C	Label #C
EN	End of Program

To start the program, command

XQ #A	Execute Program #A

This program moves X to an initial position of 1000 and returns it to zero on increments of half the distance. Note, _TPX is an internal variable which returns the value of the X position. Internal variables may be created by preceding a DMC-1600 instruction with an underscore, _.

Example 15 - Linear Interpolation

Objective: Move X,Y,Z motors distance of 7000,3000,6000, respectively, along linear trajectory. Namely, motors start and stop together.

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

Example 16 - Circular Interpolation

Objective: Move the XY axes in circular mode to form the path shown on Fig. 2-4. Note that the vector motion starts at a local position (0,0) which is defined at the beginning of any vector motion sequence. See application programming for further information.

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

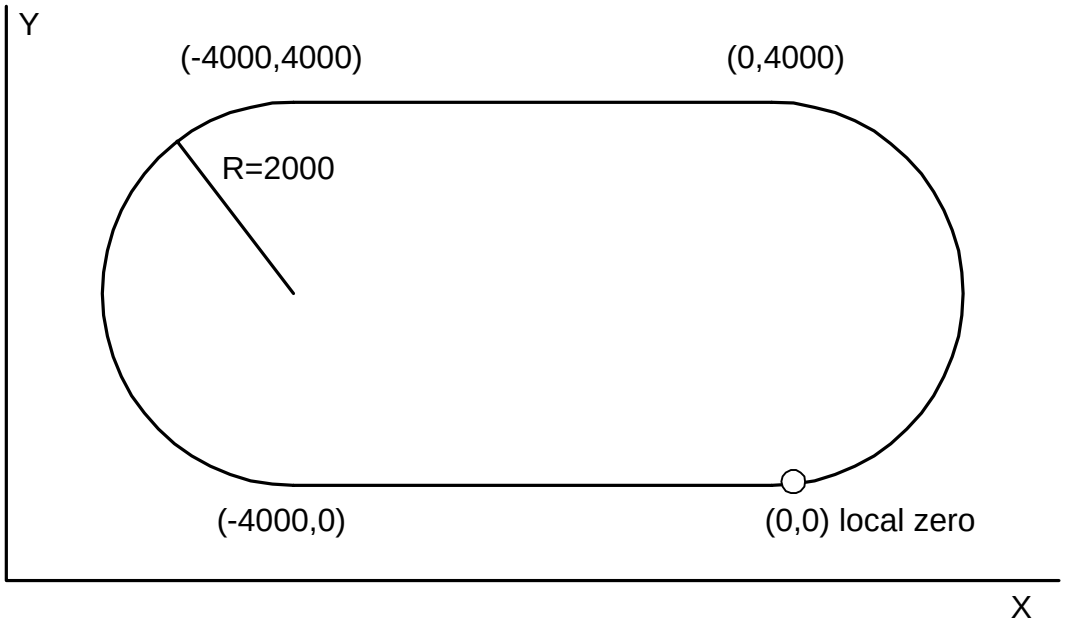


Figure 2-4 Motion Path for Example 16

THIS PAGE LEFT BLANK INTENTIONALLY

Chapter 3 Connecting Hardware

Overview

The DMC-1600 provides optoisolated digital inputs for **forward limit**, **reverse limit, home**, and **abort** signals. The controller also has 8 **optoisolated, uncommitted inputs (**for general use) as well as 8 TTL outputs and 8 analog inputs configured for voltages between +/- 10 volts.

The DMC-1610, 1620, 1630 and 1640 controllers have an additional 64 I/O which can be connected to OPTO 22 racks.

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

If you plan to use the auxiliary encoder feature of the DMC-1600, you must also connect a 26-pin IDC cable from the 26-pin J5 Auxiliary encoder connector on the DMC-1600 to the 26-pin header connector on the AMP-19X0 or ICM-1900. This cable is not shipped unless requested when ordering.

Using Optoisolated Inputs

Limit Switch Input

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

When a forward or reverse limit switch is activated, the current application program that is running will be interrupted and the controller will automatically jump to the #LIMSWI subroutine if one exists. This is a subroutine which the user can include in any motion control program and is useful for executing specific instructions upon activation of a limit switch. Automatic Subroutines are discussed in chapter 6.

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

The operands, _LFx and _LRx, contain the state of the forward and reverse limit switches, respectively (x represents the axis, X,Y,Z,W etc.). The value of the operand is either a ‘0’ or ‘1’ corresponding to the logic state of the limit switch. Using a terminal program, the state of a limit

switch can be printed to the screen with the command, MG _LFx or MG _LFx. This prints the value of the limit switch operands for the 'x' axis. The logic state of the limit switches can also be interrogated with the TS command. For more details on TS see the Command Reference.

Home Switch Input

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

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

There are three homing routines supported by the DMC-1600: Find Edge (FE), Find Index (FI), and Standard Home (HM).

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

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

The Standard Homing routine is initiated by the sequence of commands HMX <return>, BGX <return>. Standard Homing is a combination of Find Edge and Find Index homing. Initiating the standard homing routine will cause the motor to slew until a transition is detected in the logic state of the Home input. The motor will accelerate at the rate specified by the command, AC, up to the slew speed. After detecting the transition in the logic state on the Home Input, the motor will decelerate to a stop at the rate specified by the command, DC. After the motor has decelerated to a stop, it switches direction and approaches the transition point at the speed of 256 counts/sec. When the logic state changes again, the motor moves forward (in the direction of increasing encoder count) at the same speed, until the controller senses the index pulse. After detection, it decelerates to a stop and defines this position as 0. The logic state of the Home input can be interrogated with the command MG _HMX. This command returns a 0 or 1 if the logic state is low or high, respectively. The state of the Home input can also be interrogated indirectly with the TS command.

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

Abort Input

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

NOTE: The response of the abort input is significantly different from the response of an activated limit switch. When the abort input is activated, the controller stops generating motion

commands immediately, whereas the limit switch response causes the controller to make a decelerated stop.

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

All motion programs that are currently running are terminated when a transition in the Abort input is detected. For information on setting the Off-On-Error function, see the Command Reference, OE.

Uncommitted Digital Inputs

The DMC-1600 has 8 opto-isolated inputs. These inputs can be read individually using the function @IN[x] where x specifies the input number (1 thru 8). These inputs are uncommitted and can allow the user to create conditional statements related to events external to the controller. For example, the user may wish to have the x-axis motor move 1000 counts in the positive direction when the logic state of IN1 goes high.

IN9-IN16	INCOM 2
FLE,RLE,HOMEE	LSCOM 2
FLF,RLF,HOMEF	
FLG,RLG,HOMEG	
FLH,RLH,HOMEH	

Note: When using the ICM-1100, the INCOM is different for IN9-IN16 from IN1-IN8.

A logic zero is generated when at least 1mA of current flows from the common to the input. A positive voltage (with respect to the input) must be supplied at the common. This can be accomplished by connecting a voltage in the range of +5V to +28V into INCOM of the input circuitry from a separate power supply.

DMC-1610, 1620, 1630, 1640 controllers have 64 additional TTL I/O. The CO commands configures each set of 8 I/O as inputs or outputs. The DMC-16X8 use two 50 pin headers which connect directly via ribbon cable to an OPTO 22 (24 I/O) or Grayhill Opto rack (32 I/O).

The @ IN1 thru @ IN80 function checks the state of the inputs 1 thru 80.

Wiring the Optoisolated Inputs

Bi-Directional Capability.

All inputs can be used as active high or low - If you are using an isolated power supply you can connect +5V to INCOM or supply the isolated ground to INCOM. Connecting +5V to INCOM configures the inputs for active low. Connecting ground to INCOM configures the inputs for active high.

The optoisolated inputs are configured into groups. For example, the general inputs, IN1-IN8, and the ABORT input are one group. Figure 3.1 illustrates the internal circuitry. The INCOM signal is a common connection for all of the inputs in this group.

The optoisolated inputs are connected in the following groups

Group (Controllers with 1- 4 Axes)	Group (Controllers with 5 - 9 Axes)	Common Signal
IN1-IN8, ABORT	IN1-IN16, ABORT	INCOM
FLX,RLX,HOMEX FLY,RLY,HOMEY FLZ,RLZ,HOMEZ FLW,RLW,HOMEW	FLX,RLX,HOMEX,FLY,RLY,HOMEY FLZ,RLZ,HOMEZ,FLW,RLW,HOMEW FLE,RLE,HOMEE,FLF,RLF,HOMEF FLG,RLG,HOMEG,FLH,RLH,HOMEH	LSCOM

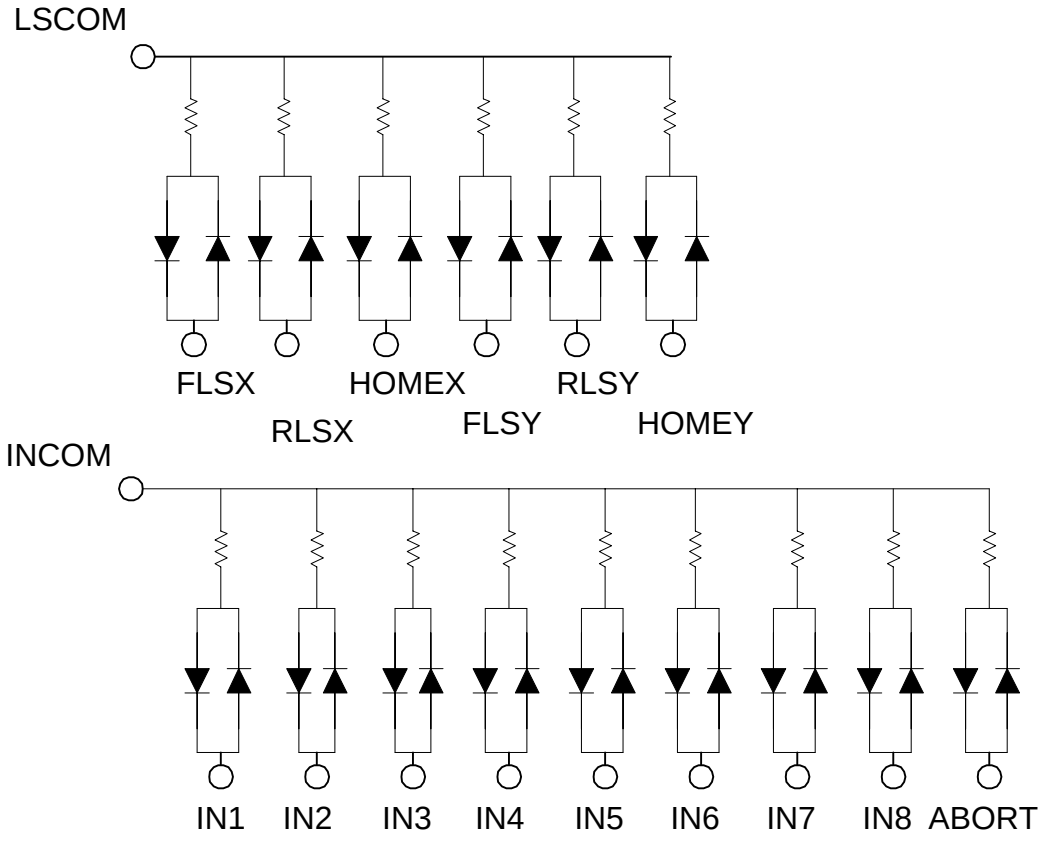


Figure 3-1. The Optoisolated Inputs

Using an Isolated Power Supply

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

$$1 \text{ mA} < \text{V supply}/(R + 2.2\text{K}\Omega) < 15 \text{ mA}$$

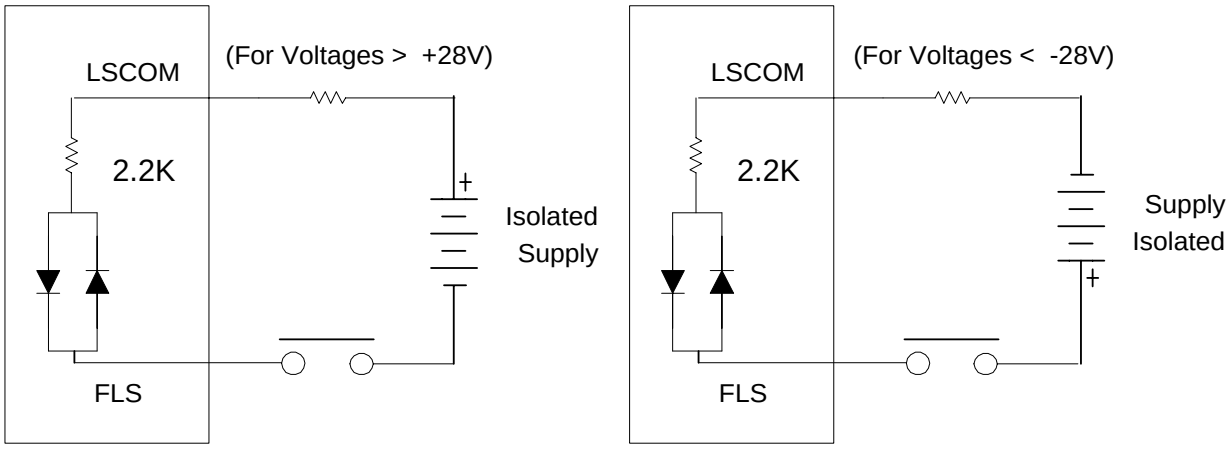


Figure 3-2. Connecting a single Limit or Home Switch to an Isolated Supply

NOTE: As stated in Chapter 2, the wiring is simplified when using the ICM-1900 or AMP-19X0 interface board. This board accepts the signals from the ribbon cables of the DMC-1600 and provides phoenix-type screw terminals. A picture of the ICM-1900 can be seen on pg 18. If an ICM-1900 is not used, an equivalent breakout board will be required to connect signals from the DMC-1600.

Bypassing the Opto-Isolation:

If no isolation is needed, the internal 5 Volt supply may be used to power the switches. This can be done by connecting a jumper between the pins LSCOM or INCOM and 5V, labeled JP3. These jumpers can be added on either the ICM-1900 (J52) or the DMC-1600. This can also be done by connecting wires between the 5V supply and common signals using the screw terminals on the ICM-1900 or AMP-19X0.

To close the circuit, wire the desired input to any ground (GND) terminal or pin out.

Analog Inputs

The DMC-1600 has eight analog inputs configured for the range between -10V and 10V. The inputs are decoded by a 12-bit A/D decoder giving a voltage resolution of approximately .005V. A 16-bit ADC is available as an option. The impedance of these inputs is 10 KΩ. The analog inputs are specified as AN[x] where x is a number 1 thru 8.

Amplifier Interface

The DMC-1600 analog command voltage, AC MD, ranges between +/-10V. This signal, along with GND, provides the input to the power amplifiers. The power amplifiers must be sized to drive the motors and load. For best performance, the amplifiers should be configured for a current mode of operation with no additional compensation. The gain should be set such that a 10 Volt input results in the maximum required current.

The DMC-1600 also provides an amplifier enable signal, AEN. This signal changes under the following conditions: the watchdog timer activates, the motor-off command, MO, is given, or the

OE1command (Enable Off-On-Error) is given and the position error exceeds the error limit. As shown in Figure 3-4, AEN can be used to disable the amplifier for these conditions.

The standard configuration of the AEN signal is TTL active high. In other words, the AEN signal will be high when the controller expects the amplifier to be enabled. The polarity and the amplitude can be changed if you are using the ICM-1900 interface board. To change the polarity from active high (5 volts= enable, zero volts = disable) to active low (zero volts = enable, 5 volts= disable), replace the 7407 IC with a 7406. Note that many amplifiers designate the enable input as 'inhibit'.

To change the voltage level of the AEN signal, note the state of the resistor pack on the ICM-1900. When Pin 1 is on the 5V mark, the output voltage is 0-5V. To change to 12 volts, pull the resistor pack and rotate it so that Pin 1 is on the 12 volt side. If you remove the resistor pack, the output signal is an open collector, allowing the user to connect an external supply with voltages up to 24V. A resistor should be put in line with the 24V supply to regulate current to 15mA.

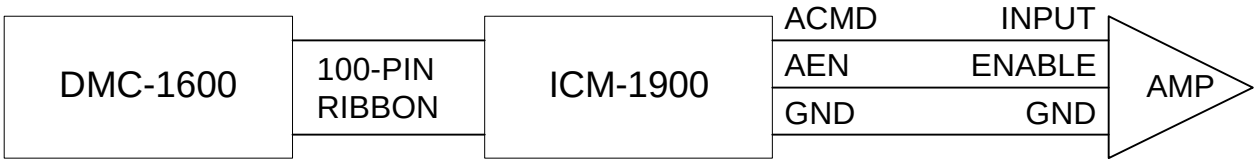


Figure 3-4 - Connecting AEN to an amplifier

TTL Outputs

The DMC-1600 provides eight general use outputs, an output compare and an error signal output.

The general use outputs are TTL and are accessible through the ICM-1900 as OUT1 thru OUT8. These outputs can be turned On and Off with the commands, SB (Set Bit), CB (Clear Bit), OB (Output Bit), and OP (Output Port). For more information about these commands, see the Command Summary. The value of the outputs can be checked with the operand _OP and the function @OUT[] (see Chapter 7, Mathematical Functions and Expressions).

NOTE: For systems using the ICM-1900 interconnect module, the ICM-1900 has an option to provide optoisolation on the outputs. In this case, the user provides an isolated power supply (+5volts to +24volts and ground). For more information, consult Galil.

The output compare signal is TTL and is available on the ICM-1900 as CMP. Output compare is controlled by the position of any of the main encoders on the controller. The output can be programmed to produce an active low pulse (1usec) based on an incremental encoder value or to activate once when an axis position has been passed. For further information, see the command OC in the Command Reference.

The error signal output is available on the interconnect module as ERROR. This is a TTL signal which is low when the controller has an error.

Note: When the error signal is low, the LED on the controller will be on which indicates an error condition one of the following error conditions:

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

Chapter 4 - Software Tools and Communications

Introduction

Galil software is available for PC computers running Microsoft Windows® to communicate with DMC-1600 controllers. Standard Galil communications software utilities are available for Windows operating systems, which includes **SmartTERM** and **WSDK**. These software packages operate under Windows 98SE, ME, NT4.0, 2000, and XP, and include the necessary drivers. In addition, Galil offers software development tools (**DMCWin** and **ActiveX Toolkit)** to allow users to create their own application interfaces using programming environments such as C, C++, Visual Basic, and LabVIEW.

Galil also offers some basic software drivers and utilities for non-Windows environments such as DOS, Linux, and QNX. For users who prefer to develop their own drivers, details are provided in this chapter describing the communication registers.

This chapter is an introduction to the software tools and communication techniques used by Galil. Figure-1 illustrates the software hierarchy that Galil communications software employs. At the application level, SmartTERM and WSDK are the basic programs that the majority of users will need to communicate with the controller, to perform basic setup, and to develop application code (.DMC programs) that is downloaded to the controller. At the Galil API level, Galil provides software tools (ActiveX and API functions) for advanced users who wish to develop their own custom application programs to communicate to the controller. Custom application programs can utilize API function calls directly to our DLL's, or use our ActiveX COM objects, which simplifies programming. At the driver level, we provide fundamental hardware interface information for users who desire to create their own drivers.

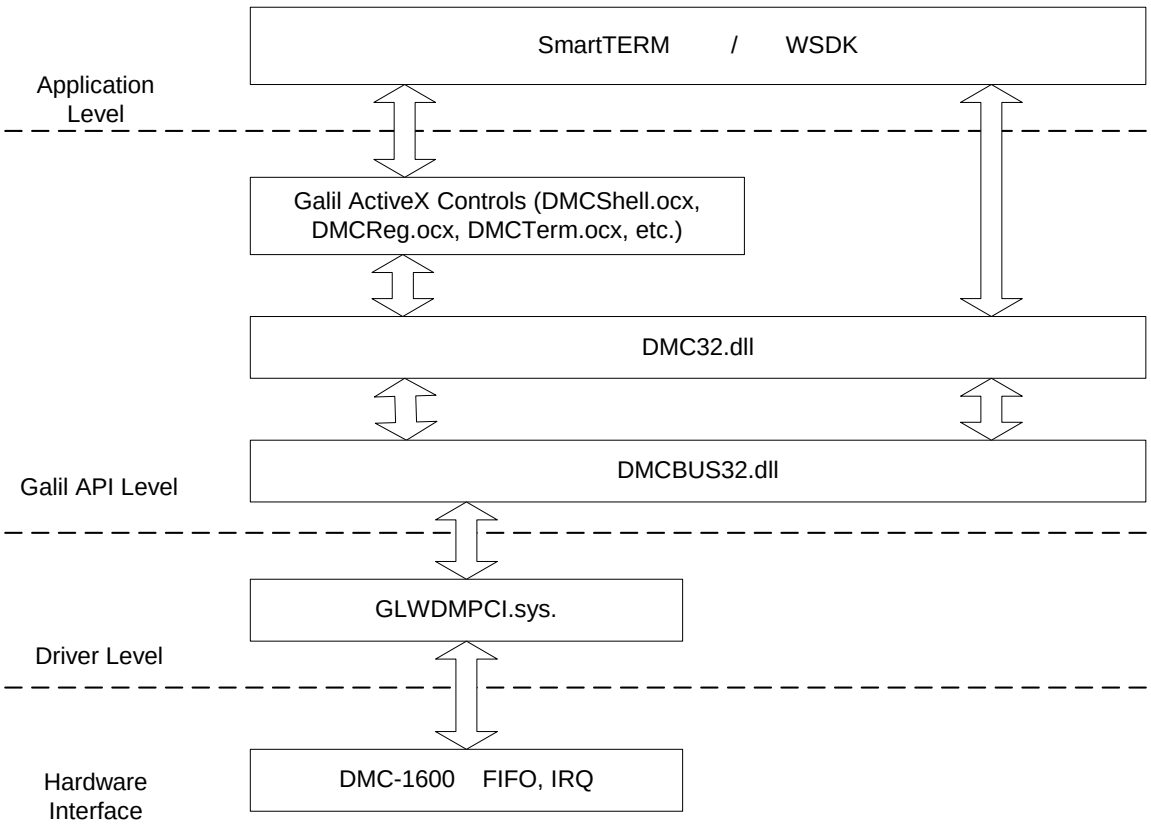


Figure 1 - Software Communications Hierarchy

Galil SmartTERM

SmartTERM is Galil's basic communications utility that allows the user to perform basic tasks such as sending commands directly to the controller, editing, downloading, and executing DMC programs, uploading and downloading arrays, and updating controller firmware. The latest version of SmartTERM can be downloaded from the Galil website at http://www.galilmc.com/support/download.html

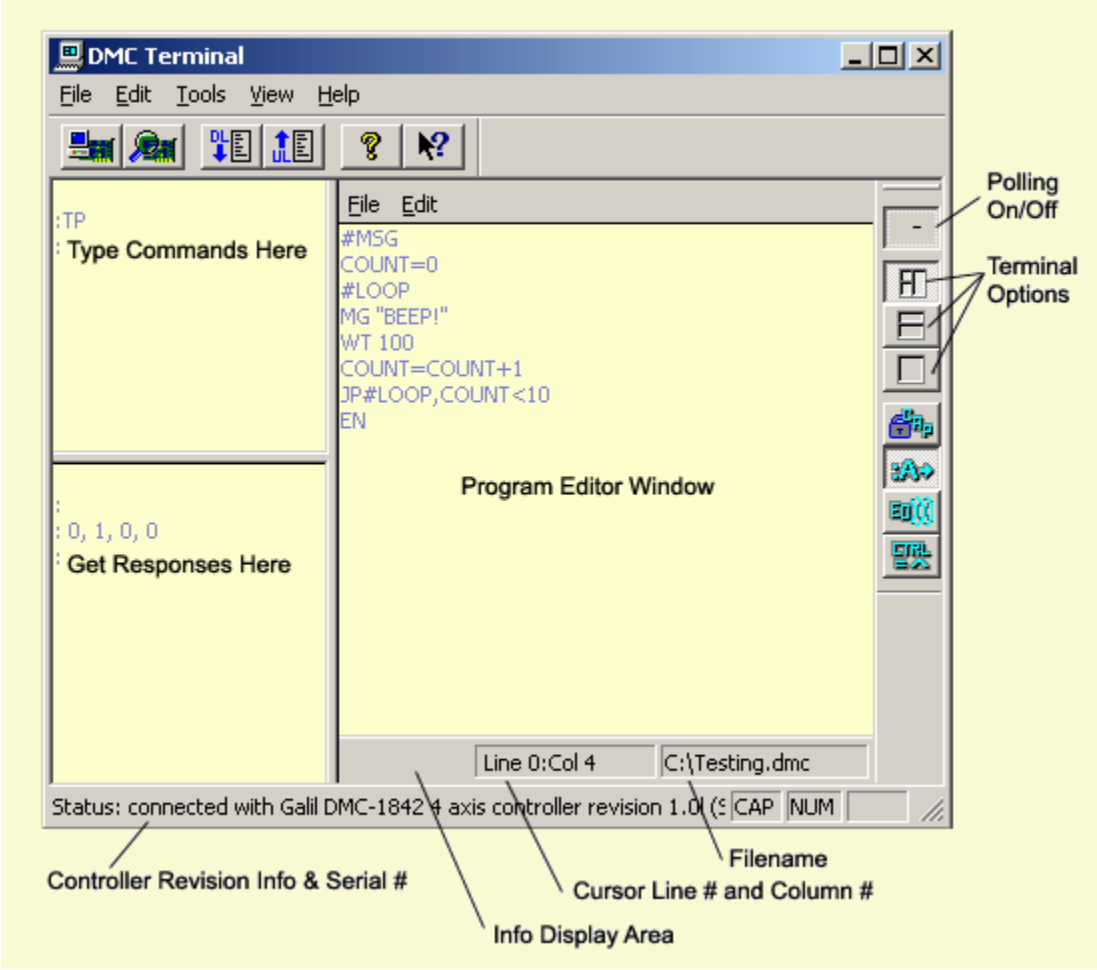


Figure 4.1 - Galil SmartTERM layout

The following SmartTERM **File** menu items describe basic features of the application.

Download File...

Launches a file-open dialog box that selects a file (usually a DMC file) to be downloaded to the controller. This command uses the DL command to download the file, clearing all programs in the controller's RAM.

Upload File...

Opens a file save-as dialog that creates a file for saving the DMC program that is in the controller's RAM. This command uses the UL command to upload the file.

Send File...

Launches a file-open dialog box that selects a file (usually a DMC file) to be sent to the controller. Each line of the file is sent to the controller as a command and is executed immediately.

Download Array...

Opens the "Download Array" dialog box that allows an array in the controller's RAM to be defined and populated with data. The dialog box uses the DMC32.dll 's DMCArrayDownload function to download the array. The controller's firmware must be recent enough to support the QD command. Array values specified in the data file must be comma separated or CRLF deliminated.

Upload Array...

Opens the "Upload Array" dialog box that allows an array in the controller's RAM to be saved to a file on the hard disk. The dialog box uses the DMC32.dll 's DMCArrayUpload function to upload the array. The controller's firmware must be recent enough to support the QU command.

Convert File ASCII to Binary...

Opens a dialog box that allows converting a file containing Galil ASCII language commands to Galil binary commands and saves the result to the specified file name.

Convert File Binary to ASCII...

Opens a dialog box that allows converting a file containing Galil binary language commands to Galil ASCII commands and saves the result to the specified file name.

Send Binary File...

Launches a file-open dialog box that selects a file (usually a DMC file) to be sent to the controller. This file can contain binary commands. Each line of the file is sent to the controller as a command and executed immediately.

The **Tools** menu items described below provide tasks such as updating firmware, diagnostics, accessing the registry editor, and resetting the controller.

Select Controller...

Opens the "Select Controller" dialog box that displays the currently registered Galil Motion Controllers. Selecting a controller from the list and clicking on the OK button or double-clicking a controller will cause the application to close any current connections to a controller and open a new connection to the selected controller. DMCTerminal only connects to a single controller at a time. However, multiple

	instances of the application can be open at once.
Disconnect from Controller	Causes the currently open connection to a Galil Motion Controller to be closed.
Controller Registration...	Opens the "Edit Registry" dialog box, which allows the Galil Registry entries to be edited or new entries for non Plug-and-Play controllers to be created or deleted.
DMC Program Editor...	Causes the terminal to enter "Smart Terminal with Editor" mode. This is the same as clicking on the "Smart Terminal with Editor" mode button on the terminal window's toolbar.
Reset Controller	Offers three "reset" options. "Reset Controller" sends an RS command to the controller. The RS command does not clear burned variables, programs, or parameters. "Master Reset" performs a master reset on the controller. A Master Reset does clear burned saved variables, programs, or parameters. "Clear Controller's FIFO" causes the controller's output FIFO to be cleared of data.
Device Driver	The Device Driver menu selection is available to operating systems and/or controllers that have device drivers that can be stopped and started. This includes drivers on NT4.0 and serial and Ethernet controllers on all operating systems.
Diagnostics	The "Diagnostics" menu allows diagnostics to be stopped and started. It also will load the diagnostics output file specified in the Tools/Options menu to be loaded into the editor window for analysis. The "Test Controller" command tests the current controller with a series of standard communication tests.
Update Firmware...	The "Update Firmware" command allows new firmware to be downloaded to the currently connected controller. Selecting this command will cause a file-open dialog box to open, allowing the user to specify a *.HEX file to be specified for download. The latest firmware files can be downloaded from Galil's website.
Display Data Record	Causes the Data Record dialog box to be

	displayed for the currently connected controller. The dialog automatically configures itself to display the data record for each type of Galil Motion Controller.
Options	The Options menu command causes the Options dialog to be displayed. The Options dialog box allows several application options to be set. These option settings are preserved between uses.

DMC Program Editor Window

The Program Editor Window is used to create application programs (.DMC) that are downloaded to the controller. The editor window is also useful for uploading and editing programs already residing in the controller's memory. This window has basic text editing features such as copy, cut, paste, etc. Also the editor window File menu allows an application program to be downloaded with compression (80 characters wide). This allows the user to write an application program in the editor window that is longer than the normal line limitation (1000 lines) and download it to the controller.

Additionally, dynamic syntax help is available by activating the syntax help button ("**:A->"** icon**)**.

DMC Data Record Display

The DMC SmartTERM utility program includes a "Data Record" display window that is useful for observing the current status of all the major functions of the controller including axis specific data, I/O status, application program status, and general status. The data record is available through a secondary communications channel.

To display the Data Record (shown in Fig 4.2), select **Display Data Record** under the **Tools** menu of DMC SmartTERM.

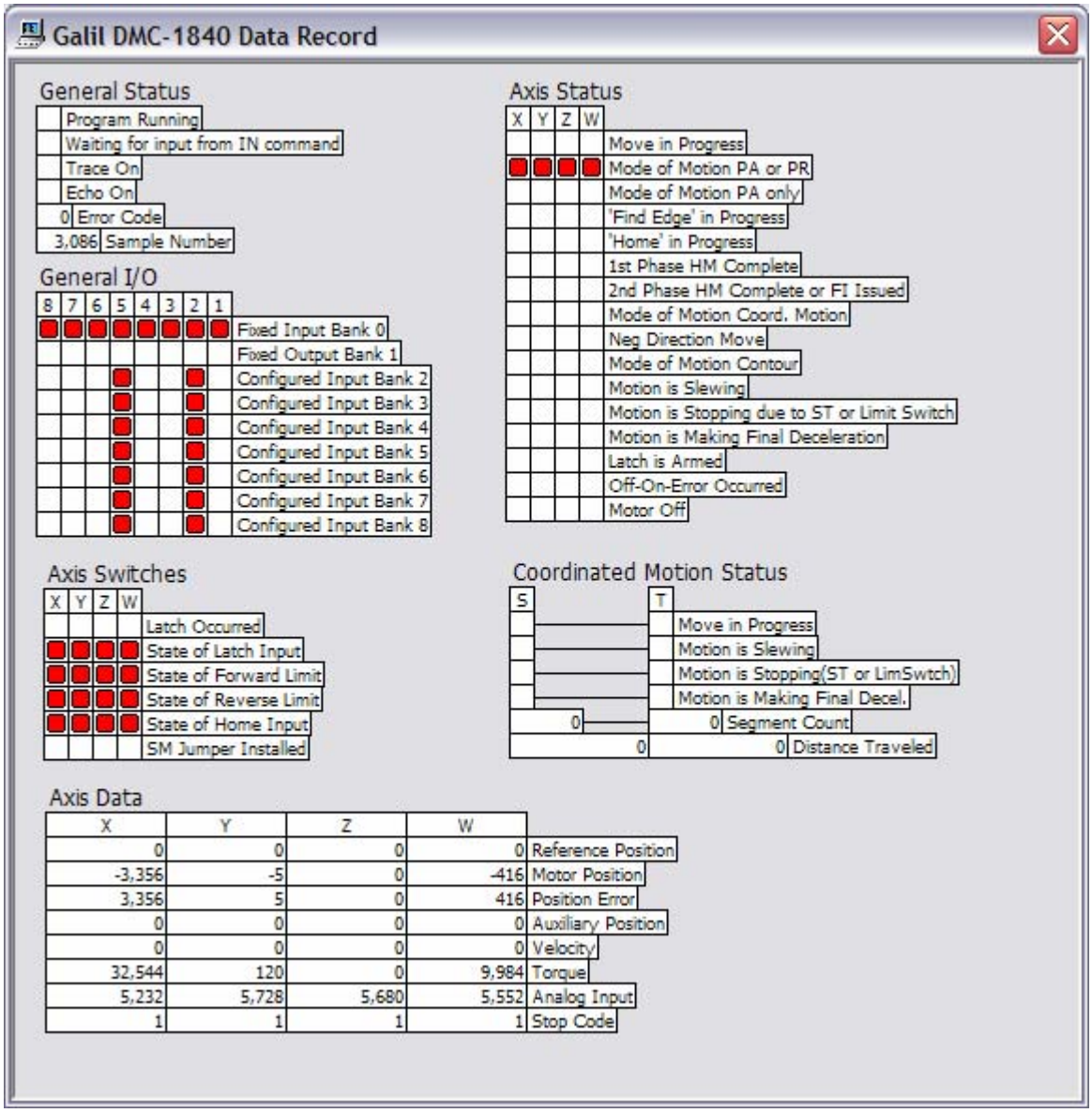


Figure 4.2 - Data Record Display for a DMC-1840

The Data Record display is user customizable so that all, or just parts, of the record can be displayed. To modify the display, right click on an object to access the options. For detailed information about the features of the Galil DMC SmartTERM including the Data Record, please consult **Help Topics** under the **Help** menu.

Communication Settings

The Galil SmartTERM application installation (as well as WSDK, ActiveX, and DMCWIN32 installations) includes the necessary drivers and .DLL files required to communicate with the Galil controller. The drivers are automatically installed and default communications settings are applied

to the device by the driver when a card is installed as per the installation procedure outlined in Ch.2. However, some advanced settings are available to modify the communications methods and data record access. These settings are accessed through the Galil Registry Editor after the card is properly installed.

Galil Registry Editor

The "Edit Registry" dialog box (shown in Fig 4.3) can be accessed by selecting **Controller Registration…** under the **Tools** menu (or by selecting the toolbar icon with the magnifying glass) within DMC SmartTERM. The Edit Registry dialog shows the current controller models installed to the PC along with their associated I/O addresses, interrupt lines, and controller serial numbers. The Galil Registry is part of the DMCReg.ocx ActiveX object (refer to Fig 4.1). This ActiveX control is used to create, maintain, and modify the critical communication parameters, which are discussed next.

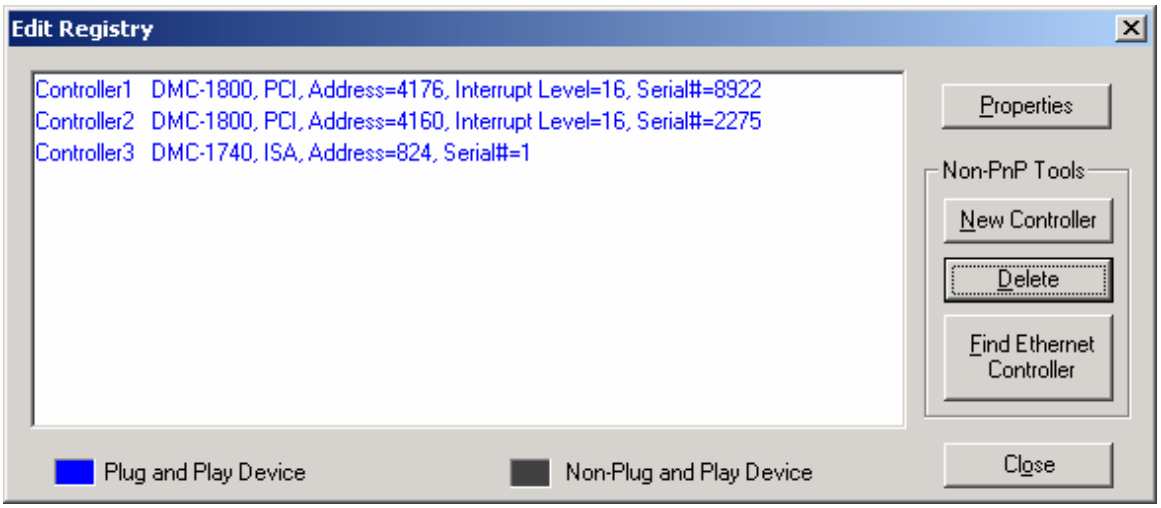


Figure 4.3 - Galil Registry Editor

Setting Communications Parameters and Methods

To access the Controller Communication Parameters dialog, highlight the desired controller in the Galil Registry Editor accessed through SmartTERM and select the **Properties** command button.

The timeout property under the **General Parameters** tab (shown in Fig 4.4) allows the user to select the timeout period that the Galil software waits for a response from the controller before generating an error. If the controller does not reply with the data response and a colon (or just a colon for commands that do not invoke responses), then the Galil software API will generate the timeout error code -1 (A time-out occurred while waiting for a response from the Galil controller). The default setting for the timeout is 5000ms, which should be sufficient for most cases.

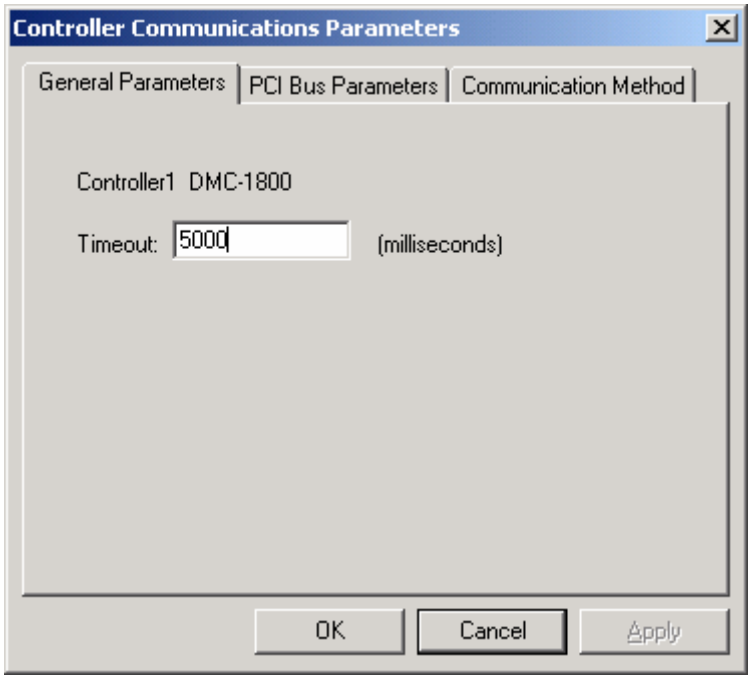


Figure 4.4 - General Communications Parameters Dialog

Advanced communications settings are available under the **Communications Method** tab to allow different methods of communications to be utilized (shown in Fig 4.5). The version 7 (and higher) drivers and .DLL's allow for three different methods of communications: Interrupt, Stall, and Delay.

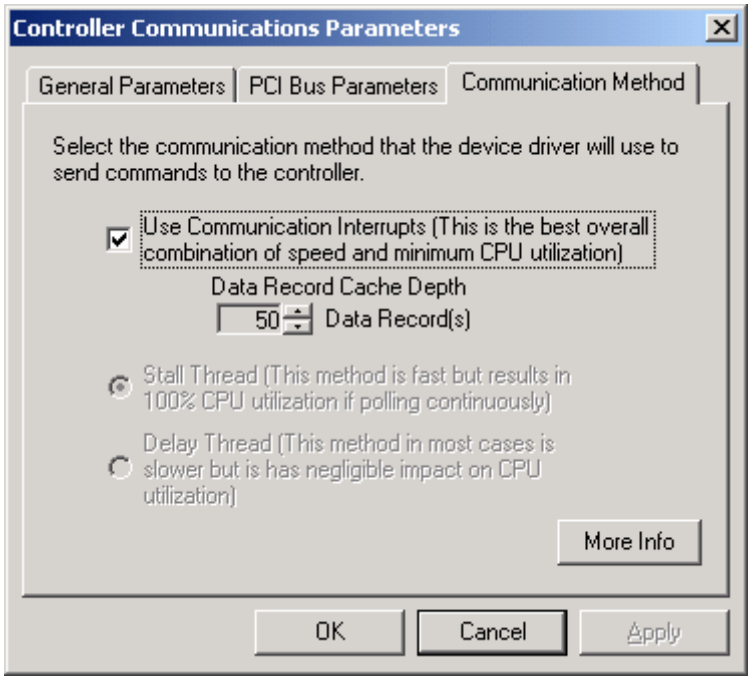


Figure 4.5 - Controller Communications Method Dialog Box

Interrupt Communications Method

The interrupt method overall is the most efficient of the three methods. The software communications method uses a hardware interrupt to notify the application that a response or unsolicited data is available. This allows for greater efficiency and response time, since the drivers do not have to “poll” the buffers for the data. Additionally, the interrupt method allows for data record caching.

The interrupt method uses bus level interrupts (IRQ) from the controller to notify the PC that data is available. This requires that the Controller be configured with a valid interrupt line. For DMC-1600 controllers the interrupt is configured automatically. Firmware version 2.0m (and greater) is required for the “communications interrupt” method to be available. For complete information on the different communications methods, select the **More Info** button on the Communications parameters dialog box.

Data Record Cache Depth

With the “interrupt communications” method enabled, the driver will cache data records for retrieval via API function calls. This makes it possible to not 'miss' any data records, even if the DR command has been configured to refresh the data record every two milliseconds. For example, a program could poll at a relatively long frequency (say every 50 milliseconds), and not miss any data. The cache depth can be set when the interrupt communication method is selected. The data record cache functions like a FIFO. Reading the data records removes them from the cache. If the cache is full and a new data record arrives from the controller, the new data record is placed in the cache and the oldest data record in the cache is discarded. If multiple handles to a controller are open, the first handle to retrieve the data record(s) will possess the only copy available. When an application needs only the most recent data record available, the cache depth should be set to 1.

Stall Thread and Delay Thread Methods

Users can also choose between "Delay" and "Stall" methods. These methods affect how the software "waits" for a response from the controller when a command is sent. If a controller is configured with the "Delay" method, the thread waiting for a command response gives up its time slice, allowing other processes running on the operating system to proceed. This method can slow communication, but results in negligible CPU utilization. The second method, the "Stall" method, uses the opposite strategy. The thread that performs I/O with the controller maintains ownership of the CPU and polls the controller until a response is received. This approach is essentially the same method employed in previous versions (< V7) of the Galil communication DLLs and drivers. While the "Stall" method does not have to wait for its thread to become eligible for execution, it does result in 100% CPU utilization while communicating with the controller.

Data Record Refresh Rate

Under the **PCI Bus Parameters** tab, the rate at which the data record is sent to the software drivers can be configured. The period between refreshes can be set from 2 - 256 ms (assuming the standard TM setting of 1000 is set). The Galil communications .DLL will use this value to send the appropriate DR command to the controller when a communications session is opened. Additionally, for DMC-1700 users, the dialog box (shown in Fig 4.6) allows the user to select between two Data Record Access methods: DMA or Secondary FIFO.

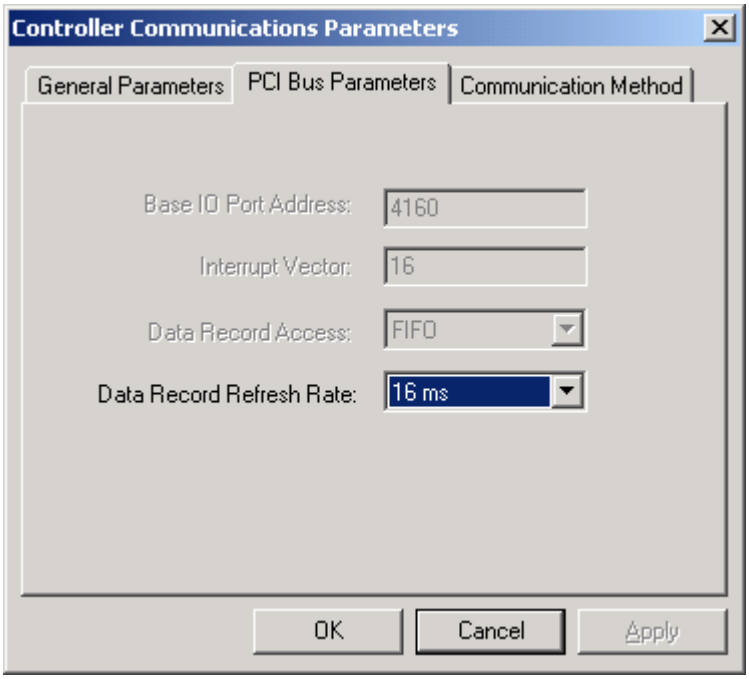


Figure 4.7 - DMC-1600 Data Record Parameters

Windows Servo Design Kit (WSDK)

The Galil Windows Servo Design Kit includes advanced tuning and diagnostic tools that allows the user to maximize the performance of their systems, as well as aid in setup and configuration of Galil controllers. WSDK is recommended for all first time users of Galil controllers. WSDK has an automatic servo tuning function that adjusts the PID filter parameters for optimum performance and displays the resulting system step response. A four-channel storage scope provides a real-time display of the actual position, velocity, error and torque. WSDK also includes impulse, step and frequency response tests, which are useful for analyzing system stability, bandwidth and resonances. WSDK can be purchased from Galil via the web at http://store.yahoo.com/galilmc/wsdk32.html.

Features Include:

- Automatic tuning for optimizing controller PID filter parameters
- Provides impulse, step and frequency response tests of actual hardware
- Four-channel storage scope for displaying real-time position, velocity, error and torque
- Displays X versus Y position for viewing 2-D motion path
- Terminal editor and program editor for easy communication with the controller

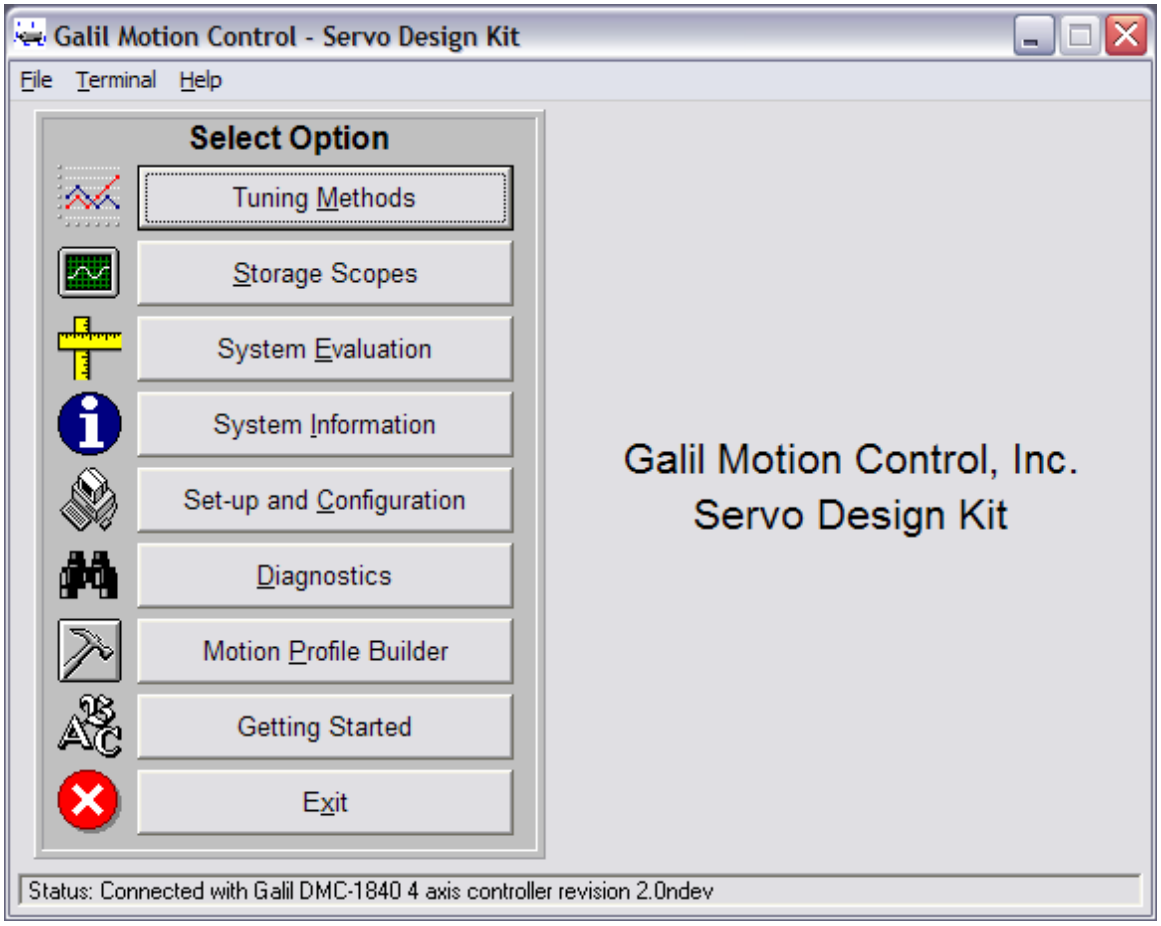


Figure 4.8- WSDK Main Screen

Creating Custom Software Interfaces

Galil provides programming tools so that users can develop their own custom software interfaces to a Galil controller. These tools include the **ActiveX Toolkit** and **DMCWin.**

ActiveX Toolkit

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

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

ActiveX Toolkit Includes:

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

For more detailed information on the ActiveX Toolkit, please refer to the user manual at http://www.galilmc.com/support/manuals/activex.pdf.

DMCWin Programmers Toolkit

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

This software package is free for download and is available at http://www.galilmc.com/support/download.html.

Galil Communications API with C/C++

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

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

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

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

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

```
#include <windows.h>
#include <dmccom.h>
long rc;
HANDLEDMC hDmc;
HWND hWnd;
int main(void)
{
     // Connect to controller number 1
```

```
        rc = DMCOpen(1, hWnd, &hDmc);
        if (rc == DMCNOERROR)
        {
        char szBuffer[64];
        // Move the X axis 1000 counts
        rc = DMCCommand(hDmc, "PR1000;BGX;", szBuffer,
      sizeof(szBuffer));
      // Disconnect from controller number 1 as the last action
        rc = DMCClose(hDmc);
        }
      return 0;
}
```

Galil Communications API with Visual Basic

Declare Functions

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

Sending Commands in VB

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

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

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

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

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

Private Sub Form_Load()
   m_nResponseLength = 256
```

```
    m_nController = 1
    m_nRetCode = DMCOpen(m_nController, 0, m_hDmc)
End Sub

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

Where:

'm_nController' is the number for the controller in the Galil registry.

'm_hDmc' is the DMC handle used to identify the controller. It is returned by DMCOpen.

'm_nRetCode' is the return code for the routine.

'm_nResponseLength' is the response string length which must be set to the size of the response string.

'm_sResponse' is the string containing the controller response to the command.

DOS, Linux, and QNX tools

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

DOS

DOS based utilities & Programming Libraries for Galil controllers, which includes a terminal, utilities to upload and download programs, and source code for BASIC and C programs. Download DMCDOS at http://www.galilmc.com/support/download.html#dos.

Linux

Galil has developed code examples for the Linux operating system. The installation includes sample drivers to establish communication with Galil PCI and ISA controllers. The current version of the software has been tested under Redhat 6.X O.S. Source code for the drivers and other utilities developed for Linux are available to customers upon request. Linux drivers are available for ISA and PCI cards under Kernal 2.2. Drivers are also available for the PCI card only for Kernal 2.4.
For more information on downloading and installing the Linux drivers for Galil controllers, download the Linux manual at: http://www.galilmc.com/support/manuals/lnxmanual.pdf.

QNX

Galil offers sample drivers for ISA and PCI cards for the QNX 4.24 operating system. We also offer drivers and utilities for QNX 6.2 for PCI only. Download at http://www.galilmc.com/support/download.html#linux.

Command Format and Controller Response

Instructions may be sent in Binary or ASCII format. Binary communication allows for faster data processing since the controller does not have to first decode the ASCII characters.

ASCII Command mode
In the ASCII mode, instructions are represented by two characters followed by the appropriate parameters. Each instruction must be terminated by a carriage return or semicolon.

The controller decodes each ASCII character (one byte) one at a time. It takes approximately 350 sec for the controller to decode each command and execute it.

After the instruction is decoded, the controller returns a colon (:) if the instruction was valid or a question mark (?) if the instruction was not valid.

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

An echo function is also provided to enable associating the response with the command sent. The echo is enabled by sending the command EO 1 to the controller.

Binary Command Mode

Some commands have an equivalent binary value for the controllers. These values are listed in the Command Reference next to the command in parentheses in hexadecimal format . Binary communication mode can be executed much faster than ASCII commands since the controller does not have to first decode the ASCII characters. Binary format can only be used when commands are sent from the PC and cannot be embedded in an application program.

Binary Command Format

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

Binary Header Format:

Byte 1 specifies the hexadecimal command number between 80 to FF.

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

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

Byte 3 specifies whether the command applies to coordinated motion on the “S” or “T” axis as follows:

Bit 1 =	T axis coordinated motion movement
Bit 0 =	S axis coordinated motion movement

For example, the command STS commands motion to stop on the S axis vector motion. The third byte for the equivalent binary command would then be 01.

Byte 4 specifies the axis # or data field as follows

Bit 7 = H axis or 8th data field

Bit 6 = G axis or 7th data field

Bit 5 = F axis or 6th data field

Bit 4 = E axis or 5th data field

Bit 3 = D axis or 4th data field

Bit 2 = C axis or 3rd data field

Bit 1 = B axis or 2nd data field

Bit 0 = A axis or 1st data field

Data Fields Format

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

A7 02 00 05 03 E8 xx xx FE 0C

where A7 is the command number for PR

02 specifies 2 bytes for each data field

00 coordinated motion is not active for PR

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

03 E8 represents 1000

xx xx represents inactive data for the B axis (xx xx can be any values since byte 4 was configured to ignore it)

FE OC represents -500

Example

The command "STABC" to stop motion on just axis A, B, and C would be:

A1 00 00 07

where A1 is the command number for ST

00 specifies 0 data fields

00 specifies the command does not apply to the coordinated motion

07 specifies stop A (bit 0), B (bit 1) and C (bit 2) ($2^0+2^1+2^2 =7$)

For more information and a complete list of all Galil binary commands, please refer to the Optima Series Bus-Based Command Reference at http://www.galilmc.com/support/manuals/optcom.pdf.

Controller Event Interrupts and User Interrupts

The DMC-1600 provides a hardware interrupt line that will, when enabled, interrupt the PC bus, which will allow the controller to notify the host application of particular events occurring on the controller. Interrupts free the host from having to poll for the occurrence of certain events such as motion complete or excess position error.
The DMC-1600 uses only one of the PC's interrupts; however, it is possible to interrupt on multiple conditions. For this reason, the controller provides a status byte register that contains a byte designating each condition.
The DMC-1600 provides an interrupt buffer that is 16 deep. This allows for multiple interrupt conditions to be stored in sequence of occurrence without loss of data.
The DMC-1600 provides two command forms of interrupt functionality, EI and UI. Specific interrupt conditions can be enabled using the EI command, or explicit user defined interrupts can be sent using the UI command.

Enabling Event Interrupts (EI command)

To enable certain conditions, use the command EIm,n. Where the first field "m" represents a 16-bit value of conditions described in the table below. For example, to enable interrupts on X and Y motion complete and position error, set EI515 (i.e. $515=2^0+2^1+2^9$). Once the EI command is enabled for a specific condition, an interrupt will occur for every instance of that condition, except for the items marked with an asterisk (*), they must be re-enabled after every occurrence.

Bit Number	Condition
0	X motion complete
1	Y motion complete
2	Z motion complete
3	W motion complete
4	E motion complete
5	F motion complete
6	G motion complete
7	H motion complete
8	All axes motion complete
9	Excess position error*
10	Limit switch
11	Watchdog timer
12	Reserved
13	Application program stopped
14	Command done †
15	Inputs* (uses n for mask)

†Not used when using new version 7 drivers.

The argument "n" enables interrupts for the first 8 general inputs. To enable interrupts for the desired inputs, set bit 15 of the "m" argument, then set the desired inputs using the 8-bit mask for the "n" argument. For example, to enable interrupt on inputs 1-4, set EI32768,15. Note that the input interrupts must be reset for all inputs after any input has caused an interrupt.

Bit number	Input
0	Input 1
1	Input 2
2	Input 3

3	Input 4
4	Input 5
5	Input 6
6	Input 7
7	Input 8

User Interrupts (UI command)

The DMC-1600 also provides 16 User Interrupts which can be sent by executing the command UIn, where n is an integer between 0 and 15. The UI command does not require the EI command. UI commands are useful in DMC programs to let the host application know that certain points within the DMC program have occurred.

Servicing Interrupts

Once an interrupt occurs, the controller sends a Status Byte to the host computer. The Status Byte returned denotes what condition has occurred, as described in the table below.

Status Byte (hex)	Condition
00	No interrupt
D9	Watchdog timer activated
DA	Command done
DB	Application program done
F0 thru FF	User interrupt (UI)
E1 thru E8	Input interrupt
C0	Limit switch occurred
C8	Excess position error
D8	All axis motion complete
D7	H axis motion complete
D6	G axis motion complete
D5	F axis motion complete
D4	E axis motion complete
D3	W axis motion complete
D2	Z axis motion complete
D1	Y axis motion complete
D0	X axis motion complete

The recommended method to utilize the interrupts in a host application is to use a pre-defined interrupt service routine, which on interrupt, will automatically execute and return the Status Byte. For example, when using the ActiveX toolkit DMCShell control with VB, the DMCShell1_DMCInterrupt() event procedure (shown below) will automatically execute and return the StatusByte in the argument. This StausByte can then be used in a case structure as the key to notify the host application of a specific event or condition.

In this VB example below, the event procedure will display a message box every time the X-axis motion is complete, assuming the command EI1 was sent to the controller. Note: the argument is returned as 208 since the status byte is returned as an integer (i.e. D0 hex = 208 decimal).

```
Private Sub DMCShell1_DMCInterrupt(StatusByte As Integer)
      If StatusByte = 208 Then
            MsgBox "X axis complete"
      End If
End Sub
```

Hardware Level Communications

This section of the chapter describes in detail the structures used to communicate with the controller at the register interface level. The information in this section is intended for advanced programmers with extensive knowledge of ISA and PCI bus operation.
For main bi-directional communication, the DMC-1600 features a 512 character write FIFO buffer, and a 512 character read buffer. This permits sending commands at high speeds ahead of their actual processing by the DMC-1600. The DMC-1600 also provides a secondary FIFO, for access to the data record.
Note: This chapter provides an in-depth look at how the controller communicates over the PCI bus at the register interface level. For most users, we recommend using the drivers supplied by Galil to provide the necessary tools for communicating with the controller.

Determining the Base Address

The base address "N" is assigned its value by the BIOS and/or Operating System. The FIFO address N is referenced in the PCI configuration space at BAR2 (offset 18H). The following PCI information (HEX) can be used to identity the DMC-1600 controller:

PCI Device Identification

DEVICE ID	VENDOR ID	SUBSYSTEM ID	SUBSYSTEM VENDOR ID
9050H	10B5H	1640H	1079H

Read, Write, and Control Registers

The DMC-1600 provides four registers used for communication. The main communications FIFO register for sending commands and receiving responses occupies address N. The control register used to monitor the main communications status occupies address N+4. The reset register occupies address N+8 and is used for resetting the controller and/or main read/write FIFO registers as well as retrieving the IRQ status byte. The secondary FIFO for accessing the data record occupies address N+C.

Communication Registers

Register	Address	Read/Write	Description
Main FIFO	N	Read / Write	Send commands and receive responses
CONTROL	N+4	Read / Write	For FIFO status control
IRQ / RESET	N+8	Read / Write	For IRQ status byte and controller reset
Secondary FIFO	N+C	Read only	For data record access

Simplified Communication Procedure

The simplest approach for communicating with the DMC-1800 is to check bits 0 and 2 of the CONTROL register at address N+4. Bit 0 is for WRITE STATUS and bit 2 is for READ STATUS.

Read Procedure - To receive data from the DMC-1800, read the control register at address N+4 and check bit 2. If bit 2 is zero, the DMC-1600 has data to be read in the READ register at address N. Bit 2 must be checked for every character read.

Write Procedure - To send data to the DMC-1800, read the control register at address N+4 and check bit 0. If bit 0 is zero, the DMC-1600 FIFO buffer is not full and a character may be written to the WRITE register at address N. If bit 0 is one, the buffer is full and any additional data will be lost.

Any high-level computer language such as C, Basic, Pascal or Assembly may be used to communicate with the DMC-1600 as long as the READ/WRITE procedure is followed as described above, so long as the base address is known.

FIFO Control Register at N+4

Status Bit	Read/Write	Meaning
7	Read Only	If 1, Secondary FIFO empty
6	Read/Write	IRQ enable: Write 1 to enable IRQ Write 0 to disable IRQ Read 1 = IRQ enabled
5	Read/Write	IRQ status: Write 1 to clear IRQ Read 1 = IRQ pending
4	Read/Write	Freeze Status of Secondary FIFO: Write 1 to freeze 2nd FIFO Write 0 to clear freeze of 2nd FIFO Read 1 = 2nd FIFO frozen
3	Read Only	If 1, Secondary FIFO is busy updating
2	Read Only	If 1, DMC to PC Buffer empty, No data to be read
1	Read Only	If 0, PC to DMC buffer not half full. Can write at least 255 bytes. If 1, buffer is more than half full.
0	Read Only	If 1, PC to DMC Buffer full, Do not write data

Half Full Flag

The Half Full flag (Bit 1 of the control register) can be used to increase the speed of writing large blocks of data to the controller. When the half full bit is zero, the write buffer is less than half full. In this case, up to 255 bytes can be written to the controller at address N without checking the buffer full status (bit 0 of the control register).

Reading the Data Record from the Secondary FIFO

To read the data record from the secondary FIFO, first the "freeze" bit (bit 4 of N+4) of the control register must be set. Then wait for the controller to finish updating the last data record by monitoring the "busy" status bit (bit3 of N+4). When bit 3 is "0" the data record can be read. Since the Secondary FIFO at N+C is 4 bytes wide, data may be read in 1 byte, 2 byte or 4 byte increments. Read the data at N+C until bit 7 of N+4 is 1, signifying that the FIFO is empty. After the data has been read, un-freeze the secondary FIFO by setting bit 4 of N+4 to "0", which allows the controller to continue to refresh the data record at the defined rate specified by the DR command.

Enabling and Reading IRQ's

In order to service interrupts from the IRQ line, the IRQ control register (Status Byte) must first be enabled. This is done by setting bit 6 of the control register (N+4) equal to "1".

When interrupted, first the interrupt routine must verify that the interrupt originated from the DMC-1600 controller. This is done by checking that the IRQ enable and IRQ status bits (bit 5 and 6 of N+4) are high. The Status Byte can then be read by reading the IRQ register at N+8. The returned Status Byte indicates what event generated the interrupt (for more information on specific interrupt events, see the EI and UI commands in the Command Reference or the previous section "Controller Event Interrupts…" in this chapter).

Once the Status Byte has been read, the interrupt must be cleared by writing a "1" to bit-5 of N+4. Note: to preserve values of other bits, the interrupt service routine should read N+4 and write this value back to N+4 to clear the interrupt.

Resetting the PC-to-DMC FIFO - To reset the output FIFO, write data to address N+8 where bit 2 is high and all other bits are low.

Resetting the DMC-to-PC FIFO - To reset the input FIFO, write data to address N+8 where bit 1 is high and all other bits are low.

Resetting the Controller - Clearing the FIFO is useful for emergency resets or Abort. For example, to reset the controller, clear the FIFO, then send the RS command. If the controller is not responding, it may be necessary to provide a hardware reset to the controller. This can be accomplished by writing data to address N+8 where bit 7 is high.

Reset Register at N+8

Status Bit	Purpose	Logic State	Meaning
7	WRITE	1	Reset Controller
2	WRITE	1	Reset PC_to_DMC FIFO
1	WRITE	1	Reset DMC_to_PC FIFO

Secondary FIFO Memory Map

ADDR	TYPE	ITEM
00-01	UW	sample number
02	UB	general input block 0 (inputs 1-8)
03	UB	general input block 1 (inputs 9-16)
04	UB	general input block 2 (inputs 17-24)
05	UB	general input block 3 (inputs 25-32)
06	UB	general input block 4 (inputs 33-40)
07	UB	general input block 5 (inputs 41-48)
08	UB	general input block 6 (inputs 49-56)
09	UB	general input block 7 (inputs 57-64)
10	UB	general input block 8 (inputs 65-72)
11	UB	general input block 9 (inputs 73-80)
12	UB	general output block 0 (outputs 1-8)
13	UB	general output block 1 (outputs 9-16)
14	UB	general output block 2 (outputs 17-24)
15	UB	general output block 3 (outputs 25-32)

16	UB	general output block 4 (outputs 33-40)
17	UB	general output block 5 (outputs 41-48)
18	UB	general output block 6 (outputs 49-56)
19	UB	general output block 7 (outputs 57-64)
20	UB	general output block 8 (outputs 65-72)
21	UB	general output block 9 (outputs 73-80)
22	UB	error code
23	UB	general status
24-25	UW	segment count of coordinated move for S plane
26-27	UW	coordinated move status for S plane
28-31	SL	distance traveled in coordinated move for S plane
32-33	UW	segment count of coordinated move for T plane
34-35	UW	coordinated move status for T plane
36-39	SL	distance traveled in coordinated move for T plane
40-41	UW	x,a axis status
42	UB	x,a axis switches
43	UB	x,a axis stopcode
44-47	SL	x,a axis reference position
48-51	SL	x,a axis motor position
52-55	SL	x,a axis position error
56-59	SL	x,a axis auxiliary position
60-63	SL	x,a axis velocity
64-65	SW	x,a axis torque
66-67	SW	x,a axis analog input
68-69	UW	y,b axis status
70	UB	y,b axis switches
71	UB	y,b axis stopcode
72-75	SL	y,b axis reference position
76-79	SL	y,b axis motor position
80-83	SL	y,b axis position error
84-87	SL	y,b axis auxiliary position
88-91	SL	y,b axis velocity
92-93	SW	y,b axis torque
94-95	SW	y,b axis analog input
96-97	UW	z,c axis status
98	UB	z,c axis switches
99	UB	z,c axis stopcode
100-103	SL	z,c axis reference position
104-107	SL	z,c axis motor position
108-111	SL	z,c axis position error
112-115	SL	z,c axis auxiliary position
116-119	SL	z,c axis velocity
120-121	SW	z,c axis torque
122-123	SW	z,c axis analog input
124-125	UW	w,d axis status

126	UB	w,d axis switches
127	UB	w,d axis stop code
128-131	SL	w,d axis reference position
132-135	SL	w,d axis motor position
136-139	SL	w,d axis position error
140-143	SL	w,d axis auxiliary position
144-147	SL	w,d axis velocity
148-149	SW	w,d axis torque
150-151	SW	w,d axis analog input
152-153	UW	e axis status
154	UB	e axis switches
155	UB	e axis stop code
156-159	SL	e axis reference position
160-163	SL	e axis motor position
164-167	SL	e axis position error
168-171	SL	e axis auxiliary position
172-175	SL	e axis velocity
176-177	SW	e axis torque
178-179	SW	e axis analog input
180-181	UW	f axis status
182	UB	f axis switches
183	UB	f axis stopcode
184-187	SL	f axis reference position
188-191	SL	f axis motor position
192-195	SL	f axis position error
196-199	SL	f axis auxiliary position
200-203	SL	f axis velocity
204-205	SW	f axis torque
206-207	SW	f axis analog input
208-209	UW	g axis status
210	UB	g axis switches
211	UB	g axis stopcode
212-215	SL	g axis reference position
216-219	SL	g axis motor position
220-223	SL	g axis position error
224-227	SL	g axis auxiliary position
228-231	SL	g axis velocity
232-233	SW	g axis torque
234-235	SW	g axis analog input
236-237	UW	h axis status
238	UB	h axis switches
239	UB	h axis stopcode
240-243	SL	h axis reference position
244-247	SL	h axis motor position
248-251	SL	h axis position error

252-255	SL	h axis auxiliary position
256-259	SL	h axis velocity
260-261	SW	h axis torque
262-263	SW	h axis analog input

Note: *UB = Unsigned Byte, UW = Unsigned Word, SW = Signed Word, SL = Signed Long Word*

Explanation of Status Information and Axis Switch Information

General Status Information (1 Byte)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Program Running	N/A	N/A	N/A	N/A	N/A	Trace on	Echo On

Axis Switch Information (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	SM Jumper Installed

Axis Status Information (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 decel.	Latch is armed	Off-On-Error enabled	Motor Off

Coordinated Motion Status Information 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	Motion is making	N/A	N/A	N/A

due to ST or Limit Switch	final decel.

Notes Regarding Velocity, Torque and Analog Input Data

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 +/-32544. Maximum negative torque of –9.9982 V is represented by –32544. Maximum positive torque of 9.9982 V is represented by 32544. Torque information is then scaled linearly as 1v=~3255.

The analog input is stored as a 16-bit value (+/-32768), which represents an analog voltage range of +/-10V.

Chapter 5 Command Basics

Introduction

The DMC-1600 provides over 100 commands for specifying motion and machine parameters. Commands are included to initiate action, interrogate status and configure the digital filter. These commands can be sent in ASCII or binary.

In ASCII, the DMC-1600 instruction set is BASIC-like and easy to use. Instructions consist of two uppercase letters that correspond phonetically with the appropriate function. For example, the instruction BG begins motion, and ST stops the motion. In binary , commands are represented by a binary code ranging from 80 to FF.

ASCII commands can be sent "live" over the bus for immediate execution by the DMC-1600, or an entire group of commands can be downloaded into the DMC-1600 memory for execution at a later time. Combining commands into groups for later execution is referred to as Applications Programming and is discussed in the following chapter. Binary commands cannot be used in Applications programming.

This section describes the DMC-1600 instruction set and syntax. A summary of commands as well as a complete listing of all DMC-1600 instructions is included in the *Command Reference* chapter.

Command Syntax - ASCII

DMC-1600 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 <enter> is used to terminate the instruction for processing by the DMC-1600 command interpreter. Note: If you are using a Galil terminal program, commands will not be processed until an <enter> command is given. This allows the user to separate many commands on a single line and not begin execution until the user gives the <enter> command.

IMPORTANT: All DMC-1600 commands are sent in upper case.

For example, the command

PR 4000 <enter> Position relative

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

For specifying data for the X,Y,Z and W 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. The space between the data and instruction is optional.

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

PR 1000	Specify X only as 1000
PR ,2000	Specify Y only as 2000
PR ,,3000	Specify Z only as 3000
PR ,,,4000	Specify W only as 4000
PR 2000, 4000,6000, 8000	Specify X Y Z and W
PR ,8000,,9000	Specify Y and W only
PR ?,?,?,?	Request X,Y,Z,W values
PR ,?	Request Y value only

The DMC-1600 provides an alternative method for specifying data. Here data is specified individually using a single axis specifier such as X,Y,Z or W. An equals sign is used to assign data to that axis. For example:

PRX=1000	Specify a position relative movement for the X axis of 1000
ACY=200000	Specify acceleration for the Y axis as 200000

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

BG X	Begin X only
BG Y	Begin Y only
BG XYZW	Begin all axes
BG YW	Begin Y and W only
BG	Begin all axes
BG ABCDEFGH	Begin all axes
BG D	Begin D only

Coordinated Motion with more than 1 axis

When requesting action for coordinated motion, the letter S is used to specify the coordinated motion. For example:

BG S	Begin coordinated sequence
BG SW	Begin coordinated sequence and W axis

Command Syntax - Binary

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

Binary Command Format

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

Header Format:

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

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

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

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

00	No coordinated motion movement
01	Coordinated motion movement

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

Byte 4 specifies the axis # or data field as follows

Bit 7 = H axis or 8^{th} data field

Bit 6 = G axis or 7^{th} data field

Bit 5 = F axis or 6^{th} data field

Bit 4 = E axis or 5^{th} data field

Bit 3 = D axis or 4^{th} data field

Bit 2 = C axis or 3^{rd} data field

Bit 1 = B axis or 2^{nd} data field

Bit 0 = A axis or 1^{st} data field

Data fields Format

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

A7 02 00 05 03 E8 FE 0C

where A7 is the command number for PR

02 specifies 2 bytes for each data field

00 S is not active for PR

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

03 E8 represents 1000

FE OE represents -500

Example

The command ST XYZS would be

A1 00 01 07

where A1 is the command number for ST

00 specifies 0 data fields

01 specifies stop the coordinated axes S

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

Binary command table

Command	No.	Command	No.	Command	No.
reserved	80	reserved	ab	reserved	d6
KP	81	reserved	ac	reserved	d7
KI	82	reserved	ad	RP	d8
KD	83	reserved	ae	TP	d9
DV	84	reserved	af	TE	da
AF	85	LM	b0	TD	db
KF	86	LI	b1	TV	dc
PL	87	VP	b2	RL	dd
ER	88	CR	a3	TT	de
IL	89	TN	b4	TS	df
TL	8a	LE, VE	b5	TI	e0
MT	8b	VT	b6	SC	e1
CE	8c	VA	b7	reserved	e2
OE	8d	VD	b8	reserved	e3
FL	8e	VS	b9	reserved	e4
BL	8f	VR	ba	TM	e5
AC	90	reserved	bb	CN	e6
DC	91	reserved	bc	LZ	e7
SP	92	CM	bd	OP	e8
IT	93	CD	be	OB	e9
FA	94	DT	bf	SB	ea
FV	95	ET	c0	CB	eb
GR	96	EM	c1	II	ec
DP	97	EP	c2	EI	ed
DE	98	EG	c3	AL	ee
OF	99	EB	c4	reserved	ef
GM	9a	EQ	c5	reserved	f0

reserved	9b	EC	c6	reserved	f1
reserved	9c	reserved	c7	reserved	f2
reserved	9d	AM	c8	reserved	f3
reserved	9e	MC	c9	reserved	f4
reserved	9f	TW	ca	reserved	f5
BG	a0	MF	cb	reserved	f6
ST	a1	MR	cc	reserved	f7
AB	a2	AD	cd	reserved	f8
HM	a3	AP	ce	reserved	f9
FE	a4	AR	cf	reserved	fa
FI	a5	AS	d0	reserved	fb
PA	a6	AI	d1	reserved	fc
PR	a7	AT	d2	reserved	fd
JG	a8	WT	d3	reserved	fe
MO	a9	WC	d4	reserved	ff
SH	aa	reserved	d5		

Controller Response to DATA

The DMC-1600 returns a : for valid commands.

The DMC-1600 returns a ? for invalid commands.

For example, if the command BG is sent in lower case, the DMC-1600 will return a ?.

:bg <enter>	invalid command, lower case
?	DMC-1600 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 inval1d command response. To request the error code, type the command: TC1 For example:

?TC1 <enter>	Tell Code command
1 Unrecognized command	Returned response

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

Interrogating the Controller

Interrogation Commands

The DMC-1600 has a set of commands that directly interrogate the controller. When the command is entered, the requested data is returned in decimal format on the next line followed by a carriage return and line feed. The format of the returned data can be changed using the Position

Format (PF), Variable Format (VF) and Leading Zeros (LZ) command. See Chapter 7 and the Command Reference.

Summary of Interrogation Commands

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

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

TP X <enter>	Tell position X
0000000000	Controllers Response
TP XY <enter>	Tell position X and Y
0000000000,0000000000	Controllers Response

Interrogating Current Commanded Values.

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

PR 1000	Specify X only as 1000
PR ,2000	Specify Y only as 2000
PR ,,3000	Specify Z only as 3000
PR ,,,4000	Specify W only as 4000
PR 2000,4000,6000,8000	Specify X Y Z and W
PR ,8000,,9000	Specify Y and W only
PR ?,?,?,?	Request X,Y,Z,W values
PR ,?	Request Y value only

The controller can also be interrogated with operands.

Operands

Most DMC-1600 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’

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

```
V=_TPX
```

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

Command Summary

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

THIS PAGE LEFT BLANK INTENTIONALLY

Chapter 6 Programming Motion

Overview

The DMC-1600 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-1610 is a single axis controller and uses X-axis motion only. Likewise, the DMC-1620 uses X and Y, the DMC-1630 uses X,Y and Z, and the DMC-1640 uses X,Y,Z and W. The DMC-1650 uses A,B,C,D, and E. The DMC-1660 uses A,B,C,D,E, and F. The DMC-1670 uses A,B,C,D,E,F and G. The DMC-1680 uses the axes A,B,C,D,E,F,G, and H.

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

EXAMPLE APPLICATION	MODE OF MOTION	COMMANDS
Absolute or relative positioning where each axis is independent and follows prescribed velocity profile.	Independent Axis Positioning	PA,PR SP,AC,DC
Velocity control where no final endpoint is prescribed. Motion stops on Stop command.	Independent Jogging	JG AC,DC,SP ST
Motion Path described as incremental position points versus time.	Contour Mode	CM CD DT WC
2,3 or 4 axis coordinated motion where path is described by linear segments.	Linear Interpolation	LM LI,LE VS,VR VA,VD
2-D motion path consisting of arc segments and linear segments, such as engraving or quilting.	Coordinated Motion	VM VP CR VS,VR VA,VD VE

Third axis must remain tangent to 2-D motion path, such as knife cutting.	Coordinated motion with tangent axis specified	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 GR GM (if gantry)
Master/slave where slave axes must follow a master such as conveyer speed.	Electronic Gearing	GA GR
Moving along arbitrary profiles or mathematically prescribed profiles such as sine or cosine trajectories.	Contour Mode	CM CD DT WC
Teaching or Record and Play Back	Contour Mode with Automatic Array Capture	CM CD DT WC RA RD RC
Backlash Correction	Dual Loop	DV
Following a trajectory based on a master encoder position	Electronic Cam	EA EM EP ET EB EG EQ
Smooth motion while operating in independent axis positioning	Independent Motion Smoothing	IT
Smooth motion while operating in vector or linear interpolation positioning	Vector Smoothing	VT
Smooth motion while operating with stepper motors	Stepper Motor Smoothing	KS
Gantry - two axes are coupled by gantry	Gantry Mode	GR GM

Independent Axis Positioning

In this mode, motion between the specified axes is independent, and each axis follows its own profile. The user specifies the desired absolute position (PA) or relative position (PR), slew speed (SP), acceleration ramp (AC), and deceleration ramp (DC), for each axis. On begin (BG), the DMC-1600 profiler generates the corresponding trapezoidal or triangular velocity profile and position trajectory. The controller determines a new command position along the trajectory every sample period until the specified profile is complete. Motion is complete when the last position command is sent by the DMC-1600 profiler. Note: The actual motor motion may not be complete when the profile has been completed, however, the next motion command may be specified.

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

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

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

Command Summary - Independent Axis

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

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

The DMC-1600 also allows use of single axis specifiers such as PRY=2000

Operand Summary - Independent Axis

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

Example - Absolute Position Movement

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

Example - Multiple Move Sequence

Required Motion Profiles:

X-Axis	500 counts	Position
	10000 count/sec	Speed
	500000 counts/sec^2	Acceleration
Y-Axis	1000 counts	Position
	15000 count/sec	Speed
	500000 counts/sec^2	Acceleration
Z-Axis	100 counts	Position
	5000 counts/sec	Speed
	500000 counts/sec	Acceleration

This example will specify a relative position movement on X, Y and Z axes. The movement on each axis will be separated by 20 msec. Fig. 6.1 shows the velocity profiles for the X,Y and Z axis.

#A	Begin Program
PR 2000,500,100	Specify relative position movement of 1000, 500 and 100 counts for X,Y and Z axes.
SP 15000,10000,5000	Specify speed of 10000, 15000, and 5000 counts / sec
AC 500000,500000,500000	Specify acceleration of 500000 counts / sec^2 for all axes
DC 500000,500000,500000	Specify deceleration of 500000 counts / sec^2 for all axes
BG X	Begin motion on the X axis
WT 20	Wait 20 msec
BG Y	Begin motion on the Y axis
WT 20	Wait 20 msec
BG Z	Begin motion on Z axis
EN	End Program

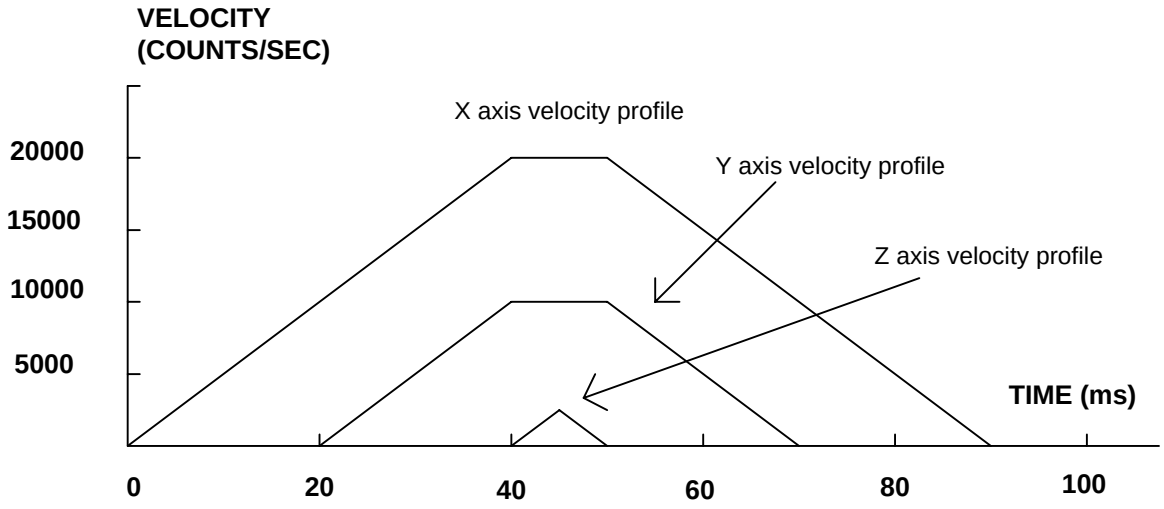


Figure 6.1 - Velocity Profiles of XYZ

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

Independent Jogging

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

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

Note that the controller operates as a closed-loop position controller while in the jog mode. The DMC-1600 converts the velocity profile into a position trajectory and a new position target is generated every sample period. This method of control results in precise speed regulation with phase lock accuracy.

Command Summary - Jogging

COMMAND	DESCRIPTION
AC x,y,z,w	Specifies acceleration rate
BG XYZW	Begins motion

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

Parameters can be set with individual axes specifiers such as JGY=2000 (set jog speed for Y axis to 2000) or ACYH=400000 (set acceleration for Y and H axes to 400000).

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 .25 sec)

Example - Jog in X only

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

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

Example - Joystick Jogging

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

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

Linear Interpolation Mode

The DMC-1600 provides a linear interpolation mode for 2 or more axes. In linear interpolation mode, motion between the axes is coordinated to maintain the prescribed vector speed, acceleration, and deceleration along the specified path. The motion path is described in terms of

incremental distances for each axis. An unlimited number of incremental segments may be given in a continuous move sequence, making the linear interpolation mode ideal for following a piece-wise linear path. There is no limit to the total move length.

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

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

Specifying Linear Segments

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

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

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

It is the responsibility of the user to keep enough LI segments in the DMC-1600 sequence buffer to ensure continuous motion. If the controller receives no additional LI segments and no LE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for LI segments that can be sent to the buffer. 511 returned means the buffer is empty and 511 LI segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional LI segments can be sent at PC bus speeds.

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

Additional Commands

The commands VS n, VA n, and VD n are used to specify the vector speed, acceleration and deceleration. The DMC-1600 computes the vector speed based on the axes specified in the LM mode. For example, LM XYZ designates linear interpolation for the X,Y and Z axes. The vector speed for this example would be computed using the equation:

$VS^2=XS^2+YS^2+ZS^2$, where XS, YS and ZS are the speed of the X,Y and Z axes.

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

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

An Example of Linear Interpolation Motion:

```
#LMOVE          label
DP 0,0          Define position of X and Y axes to be 0
LMXY            Define linear mode between X and Y axes.
LI 5000,0       Specify first linear segment
```

LI 0,5000	Specify second linear segment
LE	End linear segments
VS 4000	Specify vector speed
BGS	Begin motion sequence
AV 4000	Set trippoint to wait until vector distance of 4000 is reached
VS 1000	Change vector speed
AV 5000	Set trippoint to wait until vector distance of 5000 is reached
VS 4000	Change vector speed
EN	Program end

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

Specifying Vector Speed for Each Segment

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

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

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

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

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

As an example, consider the following program.

#ALT	Label for alternative program
DP 0,0	Define Position of X and Y axis to be 0
LMXY	Define linear mode between X and Y axes.
LI 4000,0 <4000 >1000	Specify first linear segment with a vector speed of 4000 and end speed 1000
LI 1000,1000 < 4000 >1000	Specify second linear segment with a vector speed of 4000 and end speed 1000
LI 0,5000 < 4000 >1000	Specify third linear segment with a vector speed of 4000 and end speed 1000
LE	End linear segments
BGS	Begin motion sequence
EN	Program end

Changing Feed Rate:

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

Command Summary - Linear Interpolation

COMMAND	DESCRIPTION
LM xyzw LM abcdefgh	Specify axes for linear interpolation (same) controllers with 5 or more axes
LM?	Returns number of available spaces for linear segments in DMC-1600 sequence buffer. Zero means buffer full. 512 means buffer empty.
LI x,y,z,w < n 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
VT	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-1600 sequence buffer. Zero means buffer full. 512 means buffer empty.
_VPm	Return the absolute coordinate of the last data point along the trajectory. (m=X,Y,Z or W or A,B,C,D,E,F,G or H)

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

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

Example - Linear Move

Make a coordinated linear move in the ZW plane. Move to coordinates 40000,30000 counts at a vector speed of 100000 counts/sec and vector acceleration of 1000000 counts/sec^2.

LM ZW	Specify axes for linear interpolation
LI,,40000,30000	Specify ZW distances
LE	Specify end move

VS 100000	Specify vector speed
VA 1000000	Specify vector acceleration
VD 1000000	Specify vector deceleration
BGS	Begin sequence

Note that the above program specifies the vector speed, VS, and not the actual axis speeds VZ and VW. The axis speeds are determined by the DMC-1600 from:

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

The resulting profile is shown in Figure 6.2.

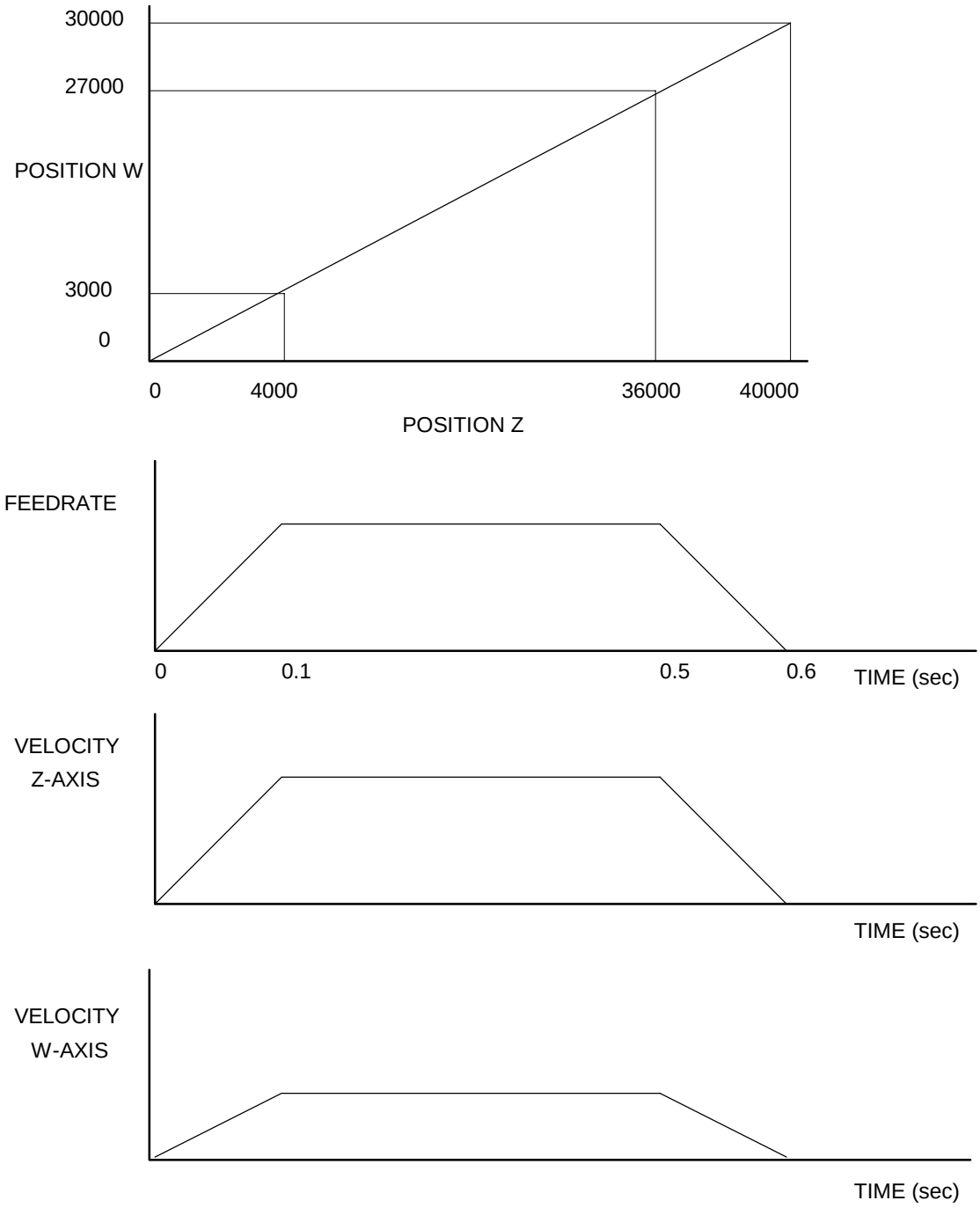


Figure 6.2 - Linear Interpolation

Example - Multiple Moves

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

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

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

Coordinated Motion Sequences

The DMC-1600 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-1600 performs all the complex computations of linear and circular interpolation, freeing the host PC from this time intensive task.

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

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

Specifying the Coordinate Plane

The DMC-1600 allows for 2 separate sets of coordinate axes for linear interpolation mode or vector mode. These two sets are identified by the letters S and T.

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

Specifying Vector Segments

The motion segments are described by two commands; VP for linear segments and CR for circular segments. Once a set of linear segments and/or circular segments have been specified, the sequence is ended with the command VE. This defines a sequence of commands for coordinated motion. Immediately prior to the execution of the first coordinated movement, the controller defines the current position to be zero for all movements in a sequence. Note: This 'local' definition of zero does not affect the absolute coordinate system or subsequent coordinated motion sequences.

The command VP x,y specifies the coordinates of the end points of the vector movement with respect to the starting point. The command CR r,q,d defines a circular arc with a radius r, starting angle of q, and a traversed angle d. The notation for q is that zero corresponds to the positive horizontal direction, and for both q and d, the counter-clockwise (CCW) rotation is positive.

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

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

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

It is the responsibility of the user to keep enough motion segments in the DMC-1600 sequence buffer to ensure continuous motion. If the controller receives no additional motion segments and no VE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for motion segments that can be sent to the buffer. 511 returned means the buffer is empty and 511 segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional segments can be sent at PC bus speeds.

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

Additional commands

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

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

Specifying Vector Speed for Each Segment:

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

VP x,y < n >m

CR r,θ,δ < n >m

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

The second function, > m, requires the vector speed to reach the value m at the end of the segment. Note that the function > m may start the deceleration within the given segment or during

previous segments, as needed to meet the final speed requirement, under the given values of VA and VD.

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

.Changing Feed rate:

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

Compensating for Differences in Encoder Resolution:

By default, the DMC-1600 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 Chapter 11, Command Summary.

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-1600 allows one axis to be specified as the tangent axis. The VM command provides parameter specifications for describing the coordinated axes and the tangent axis.

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

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

Example:

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

#EXAMPLE	Example program
VM XYZ	XY coordinate with Z as tangent
TN 2000/360,-500	2000/360 counts/degree, position -500 is 0 degrees in XY plane
CR 3000,0,180	3000 count radius, start at 0 and go to 180 CCW

VE	End vector
CB0	Disengage knife
PA 3000,0,_TN	Move X and Y to starting position, move Z to initial tangent position
BG XYZ	Start the move to get into position
AM XYZ	When the move is complete
SB0	Engage knife
WT50	Wait 50 msec for the knife to engage
BGS	Do the circular cut
AMS	After the coordinated move is complete
CB0	Disengage knife
MG "ALL DONE"	
EN	End program

Command Summary - Coordinated Motion Sequence

COMMAND	DESCRIPTION.
VM m,n	Specifies the axes for the planar motion where m and n represent the planar axes and p is the tangent axis.
VP m,n	Return coordinate of last point, where m=X,Y,Z or W.
CR r,Θ, ±ΔΘ	Specifies arc segment where r is the radius, Θ is the starting angle and ΔΘ is the travel angle. Positive direction is CCW.
VS s, t	Specify vector speed or feed rate of sequence.
VA s, t	Specify vector acceleration along the sequence.
VD s, t	Specify vector deceleration along the sequence.
VR s, t	Specify vector speed ratio
BGST	Begin motion sequence, S or T.
CSST	Clear sequence, S or T
AV s, t	Trippoint for After Relative Vector distance.
AMST	Holds execution of next command until Motion Sequence is complete.
TN m,n	Tangent scale and offset.
ES m,n	Ellipse scale factor.
VT s, t	S curve smoothing constant for coordinated moves
LM?	Return number of available spaces for linear and circular segments in DMC-1600 sequence buffer. Zero means buffer is full. 512 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-1600 sequence buffer. Zero means buffer is full. 512 means buffer is empty.
_CS	Segment counter - Number of the segment in the sequence, starting at zero.
_VE	Vector length of coordinated move sequence.

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

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

Example:

Traverse the path shown in Fig. 6.3. Feed rate is 20000 counts/sec. Plane of motion is XY

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

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

The value of _AV is 2000

The value of _CS is 0

_VPX and _VPY contain the absolute coordinate of the point A

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

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

The value of _CS is 2

_VPX,_VPY contain the coordinates of the point C

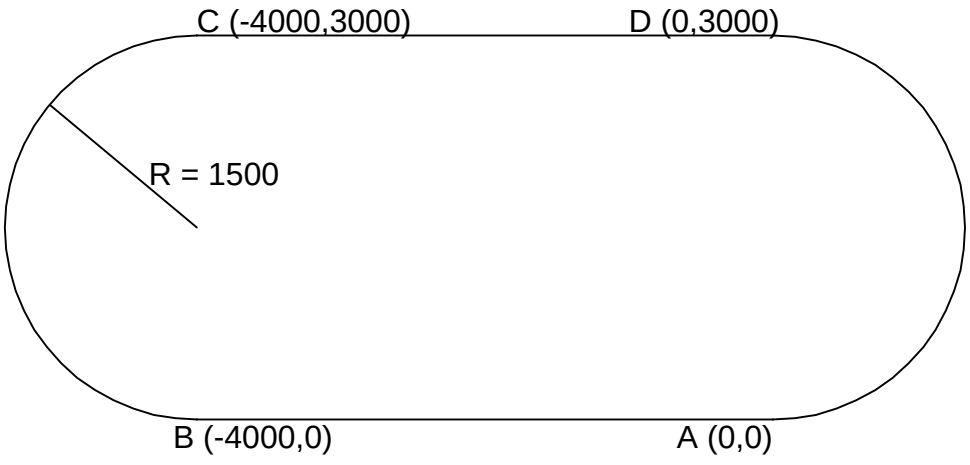


Figure 6.3 - The Required Path

Electronic Gearing

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

The command GAX yzw or GA ABCDEFGH specifies the master axes. GR x,y,z,w specifies the gear ratios for the slaves where the ratio may be a number between +/-127.9999 with a fractional resolution of .0001. There are two modes: standard gearing and gantry mode. The gantry mode is enabled with the command GM. GR 0,0,00 turns off gearing in both modes. A limit switch or ST command disable gearing in the standard mode but not in the gentry mode.

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

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

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

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

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

Command Summary - Electronic Gearing

COMMAND	DESCRIPTION
GA n	Specifies master axes for gearing where: n = X,Y,Z or W or A,B,C,D,E,F,G,H for main encoder as master
	n = CX,CY,CZ or CW or CA, CB, CC, CD, CE, CF,CG,CH for commanded position.
	n = DX,DY,DZ or DW or DA, DB, DC, DD, DE, DF,DG,DH for auxiliary encoders
	n = S vector move as master
GR x,y,z,w	Sets gear ratio for slave axes. 0 disables electronic gearing for specified axis.
GR a,b,c,d,e,f,g,h	Sets gear ratio for slave axes. 0 disables electronic gearing for specified axis.
GM a,b,c,d,e,f,g,h	X = 1 sets gantry mode, 0 disables gantry mode
MR x,y,z,w	Trippoint for reverse motion past specified value. Only one field may be used.
MF x,y,z,w	Trippoint for forward motion past specified value. Only one field may be used.

Example - Simple Master Slave

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

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

BGY	Begin motion

Example - Electronic Gearing

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

Solution: Use a DMC-1630 controller, where the Z-axis is the master and X and Y are the geared axes.

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

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

GR 2	Specify gear ratio for X axis to be 2

Example - Gantry Mode

In applications where both the master and the follower are controlled by the DMC-1600 controller, it may be desired to synchronize the follower with the commanded position of the master, rather than the actual position. This eliminates the coupling between the axes which may lead to oscillations.

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

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

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

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

Under these conditions, this IP command is equivalent to:

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

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

Example - Synchronize two conveyor belts with trapezoidal velocity correction.

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

Electronic Cam

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

The electronic cam is a more general type of electronic gearing which allows a table-based relationship between the axes. It allows synchronizing all the controller axes. For example, the DMC-1680 controller may have one master and up to seven slaves.

To illustrate the procedure of setting the cam mode, consider the cam relationship for the slave axis Y, when the master is X. Such a graphic relationship is shown in Figure 6.8.

Step 1. Selecting the master axis

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

EAp where p = X,Y,Z,W

p is the selected master axis

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

Step 2. Specify the master cycle and the change in the slave axes.

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

EM x,y,z,w

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

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

EM 6000,1500

Step 3. Specify the master interval and starting point.

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

EP m,n

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

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

EP 2000,0

Step 4. Specify the slave positions.

Next, we specify the slave positions with the instruction

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

where n indicates the order of the point.

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

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

This specifies the ECAM table.

Step 5. Enable the ECAM

To enable the ECAM mode, use the command

EB n

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

Step 6. Engage the slave motion

To engage the slave motion, use the instruction

EG x,y,z,w

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

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

Step 7. Disengage the slave motion

To disengage the cam, use the command

EQ x,y,z,w

where x,y,z,w are the master positions at which the corresponding slave axes are disengaged.

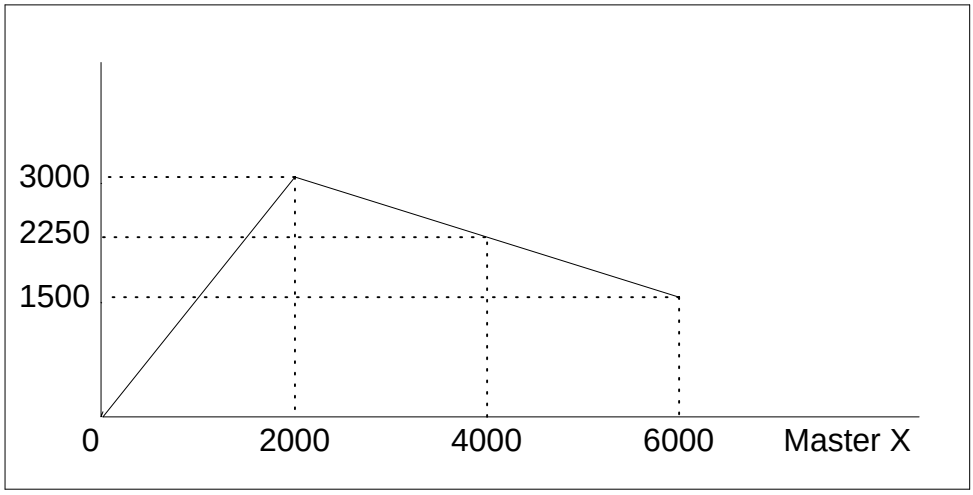


Figure 6.4 - Electronic Cam Example

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

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

the equation:

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

where X is the master, with a cycle of 2000 counts.

The cam table can be constructed manually, point by point, or automatically by a program. The following program includes the set-up.

The instruction EAX defines X as the master axis. The cycle of the master is 2000. Over that cycle, Y varies by 1000. This leads to the instruction EM 2000,1000.

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

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

Instruction	Interpretation
#SETUP	Label
EAX	Select X as master
EM 2000,1000	Cam cycles
EP 20,0	Master position increments
N = 0	Index
#LOOP	Loop to construct table from equation
*P = N*3.6*	Note 3.6 = 0.18*20
*S = @SIN [P] *100*	Define sine position
*Y = N *10+S*	Define slave position
ET [N] =, Y	Define table
N = N+1	
JP #LOOP, N<=100	Repeat the process
EN	

Command Summary – Electronic CAM

COMMAND	DESCRIPTION
EA p	Specifies master axes for electronic cam where: P = X,Y,Z or W or A,B,C,D,E,F,G,H for main encoder as master
EB n	Enables the ECAM
EC n	ECAM counter – sets the index into the ECAM table
EG x,y,z,w	Engages ECAM
EM x,y,z,w	Specifies the change in position for each axis of the CAM cycle.
EP m,n	Defines CAM table entry size and offset
EQ m,n	Disengages ECAM at specified position
ET[n]	Defines the ECAM table entries
EW	Widen segment (see Application Note #2444)

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

This is done with the program:

Instruction	Interpretation
#RUN	Label
EB1	Enable cam
PA,500	starting position
SP,5000	Y speed
BGY	Move Y motor
AM	After Y moved
AI1	Wait for start signal
EG,1000	Engage slave
AI - 1	Wait for stop signal
EQ,1000	Disengage slave
EN	End

The following example illustrates a cam program with a master axis, Z, and two slaves, X and Y.

Instruction	Interpretation
#A;V1=0	Label; Initialize variable
PA 0,0;BGXY;AMXY	Go to position 0,0 on X and Y axes
EA Z	Z axis as the Master for ECAM
EM 0,0,4000	Change for Z is 4000, zero for X, Y
EP400,0	ECAM interval is 400 counts with zero start
ET[0]=0,0	When master is at 0 position; 1st point.
ET[1]=40,20	2nd point in the ECAM table
ET[2]=120,60	3rd point in the ECAM table
ET[3]=240,120	4th point in the ECAM table
ET[4]=280,140	5th point in the ECAM table
ET[5]=280,140	6th point in the ECAM table
ET[6]=280,140	7th point in the ECAM table
ET[7]=240,120	8th point in the ECAM table
ET[8]=120,60	9th point in the ECAM table
ET[9]=40,20	10th point in the ECAM table
ET[10]=0,0	Starting point for next cycle
EB 1	Enable ECAM mode
JGZ=4000	Set Z to jog at 4000
EG 0,0	Engage both X and Y when Master = 0
BGZ	Begin jog on Z axis
#LOOP;JP#LOOP,V1=0	Loop until the variable is set
EQ2000,2000	Disengage X and Y when Master = 2000
MF,, 2000	Wait until the Master goes to 2000
ST Z	Stop the Z axis motion
EB 0	Exit the ECAM mode
EN	End of the program

The above example shows how the ECAM program is structured and how the commands can be given to the controller. The next page provides the results captured by the WSDK program. This shows how the motion will be seen during the ECAM cycles. The first graph is for the X axis, the second graph shows the cycle on the Y axis and the third graph shows the cycle of the Z axis.

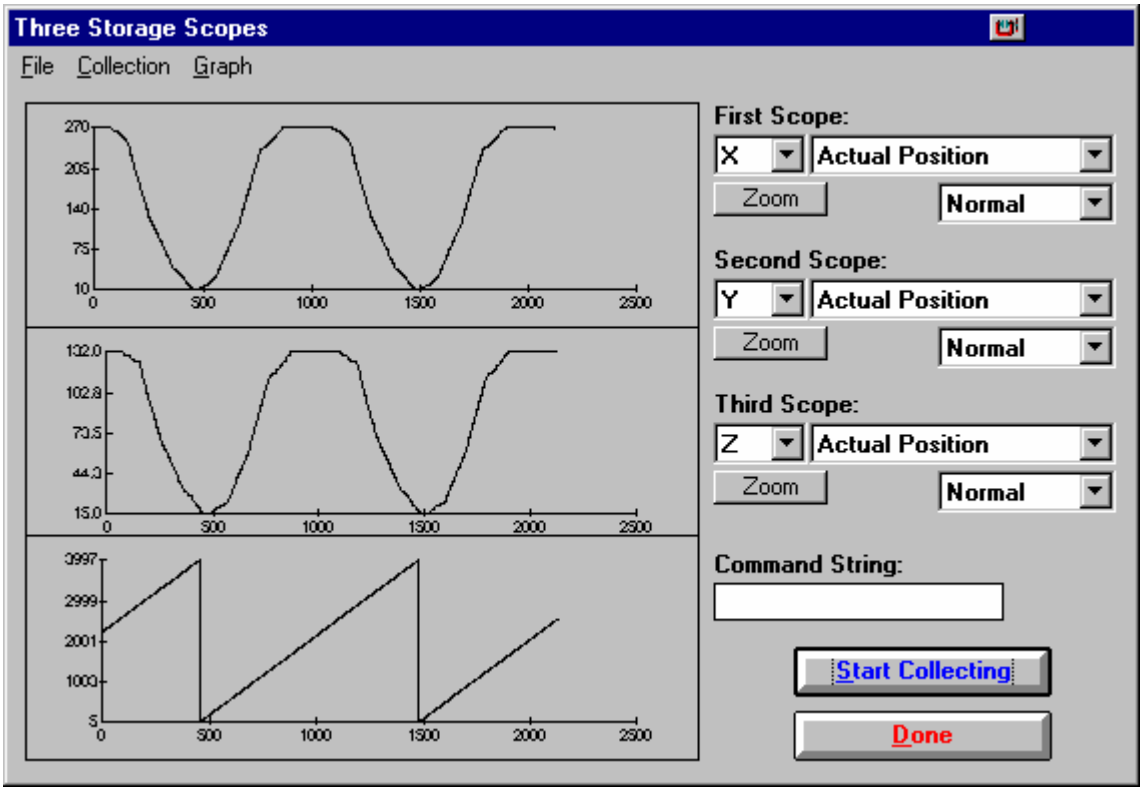


Figure 6.5 – WSDK program results – Three Storage Scopes

Contour Mode

The DMC-1600 also provides a contouring mode. This mode allows any arbitrary position curve to be prescribed for 1 to 8 axes. This is ideal for following computer generated paths such as parabolic, spherical or user-defined profiles. The path is not limited to straight line and arc segments and the path length may be infinite.

Specifying Contour Segments

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

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

Consider, for example, the trajectory shown in Fig. 6.4. The position X may be described by the points:

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

The same trajectory may be represented by the increments

Increment 1	DX=48	Time=4	DT=2
Increment 2	DX=240	Time=8	DT=3
Increment 3	DX=48	Time=16	DT=4

When the controller receives the command to generate a trajectory along these points, it interpolates linearly between the points. The resulting interpolated points include the position 12 at 1 msec, position 24 at 2 msec, etc.

The programmed commands to specify the above example are:

```
#A
```

CMX	Specifies X axis for contour mode
DT 2	Specifies first time interval, 2^2 ms
CD 48;WC	Specifies first position increment
DT 3	Specifies second time interval, 2^3 ms
CD 240;WC	Specifies second position increment
DT 4	Specifies the third time interval, 2^4 ms
CD 48;WC	Specifies the third position increment
DT0;CD0	Exits contour mode

```
EN
```

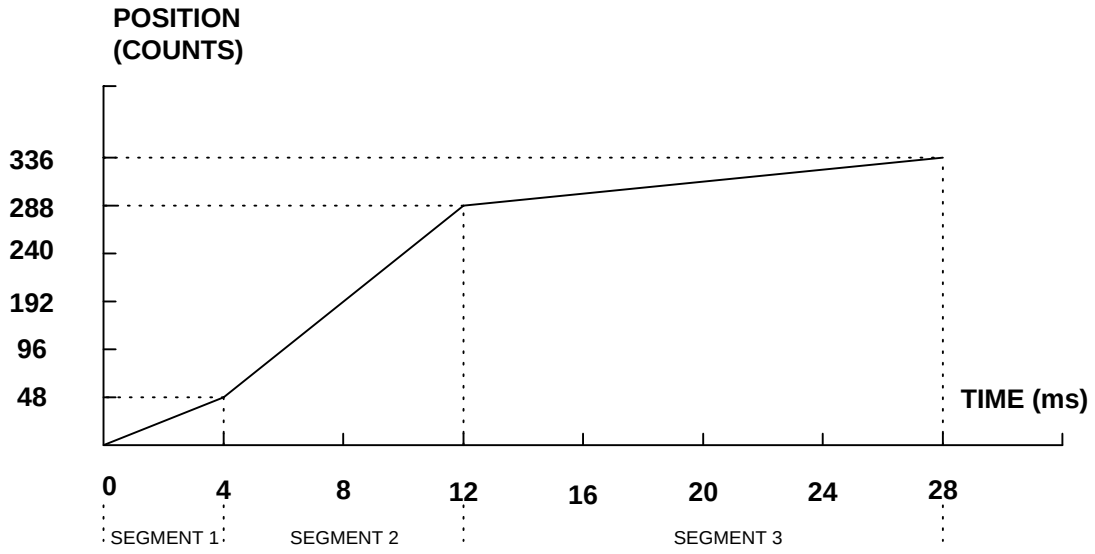


Figure 6.6 - The Required Trajectory

Additional Commands

The command, WC, is used as a trippoint "When Complete". This allows the DMC-1600 to use the next increment only when it is finished with the previous one. Zero parameters for DT followed by zero parameters for CD exit the contour mode.

If no new data record is found and the controller is still in the contour mode, the controller waits for new data. No new motion commands are generated while waiting. If bad data is received, the controller responds with a ?.

Command Summary - Contour Mode

COMMAND	DESCRIPTION
CM XYZW	Specifies which axes for contouring mode. Any non-contouring axes may be operated in other modes.
CM ABCDEFGH	Contour axes for DMC-1680
CD x,y,z,w	Specifies position increment over time interval. Range is +/-32,000. Zero ends contour mode.
CD a,b,c,d,e,f,g,h	Position increment data for DMC-1680
DT n	Specifies time interval 2^n msec 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.
WC	Waits for previous time interval to be complete before next data record is processed.

General Velocity Profiles

The Contour Mode is ideal for generating any arbitrary velocity profiles. The velocity profile can be specified as a mathematical function or as a collection of points.

The design includes two parts: Generating an array with data points and running the program.

Generating an Array - An Example

Consider the velocity and position profiles shown in Fig. 6.7. The objective is to rotate a motor a distance of 6000 counts in 120 ms. The velocity profile is sinusoidal to reduce the jerk and the system vibration. If we describe the position displacement in terms of A counts in B milliseconds, we can describe the motion in the following manner:

$$\omega = \frac{A}{B}\left(1 - \cos(2\pi/B)\right)$$

$$X = \frac{AT}{B} - \frac{A}{2\pi}\sin(2\pi/B)$$

Note: ω is the angular velocity; X is the position; and T is the variable, time, in milliseconds.

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

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

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

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

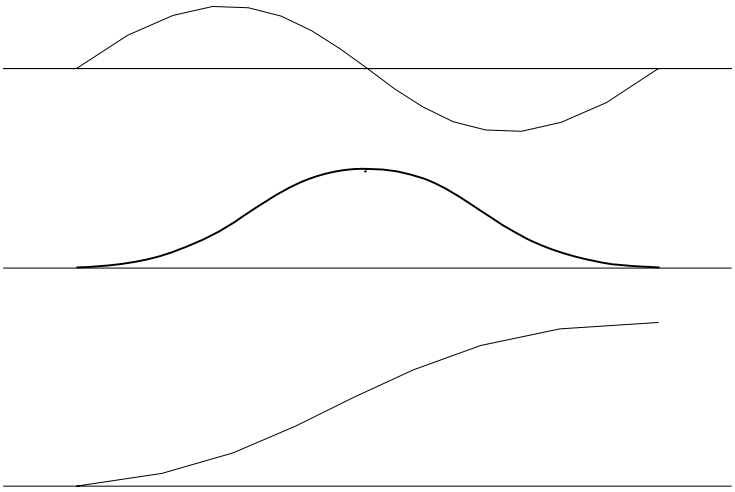


Figure 6.7 - Velocity Profile with Sinusoidal Acceleration

The DMC-1600 can compute trigonometric functions. However, the argument must be expressed in degrees. Using our example, the equation for X is written as:

X = 50T - 955 sin 3T

A complete program to generate the contour movement in this example is given below. To generate an array, we compute the position value at intervals of 8 ms. This is stored at the array POS. Then, the difference between the positions is computed and is stored in the array DIF. Finally the motors are run in the contour mode.

Contour Mode Example

Instruction	Interpretation
#POINTS	Program defines X points
DM POS[16]	Allocate memory
DM DIF[15]	
C=0	Set initial conditions, C is index
T=0	T is time in ms
#A	
V1=50*T	
V2=3*T	Argument in degrees
V3=-955*@SIN[V2]+V1	Compute position
V4=@INT[V3]	Integer value of V3
POS[C]=V4	Store in array POS
T=T+8	
C=C+1	
JP #A,C<16	
#B	Program to find position differences
C=0	
#C	

```
D=C+1
DIF[C]=POS[D]-POS[C]      Compute the difference and store
C=C+1
JP #C,C<15
EN                        End first program
#RUN                      Program to run motor
CMX                       Contour Mode
DT3                       4 millisecond intervals
C=0
#E
CD DIF[C]                 Contour Distance is in DIF
WC                        Wait for completion
C=C+1
JP #E,C<15
DT0
CD0                       Stop Contour
EN                        End the program
```

Teach (Record and Play-Back)

Several applications require teaching the machine a motion trajectory. Teaching can be accomplished using the DMC-1600 automatic array capture feature to capture position data. The captured data may then be played back in the contour mode. The following array commands are used:

DM C[n]	Dimension array
RA C[]	Specify array for automatic record (up to 4 for DMC-1640)
RD _TPX	Specify data for capturing (such as _TPX or _TPZ)
RC n,m	Specify capture time interval where n is 2n msec, m is number of records to be captured
RC? or _RC	Returns a 1 if recording

Record and Playback Example:

```
#RECORD                   Begin Program
DM XPOS[501]              Dimension array with 501 elements
RA XPOS[]                 Specify automatic record
RD _TPX                   Specify X position to be captured
MOX                       Turn X motor off
RC2                       Begin recording; 4 msec interval
#A;JP#A,_RC=1             Continue until done recording
#COMPUTE                  Compute DX
DM DX[500]                Dimension Array for DX
C=0                       Initialize counter
#L                        Label
D=C+1
DELTA=XPOS[D]-            Compute the difference
XPOS[C]
DX[C]=DELTA               Store difference in array
```

C=C+1	Increment index
JP #L,C<500	Repeat until done
#PLAYBCK	Begin Playback
CMX	Specify contour mode
DT2	Specify time increment
I=0	Initialize array counter
#B	Loop counter
CD XPOS[I];WC	Specify contour data I=I+1 Increment array counter
JP #B,I<500	Loop until done
DT 0;CD0	End contour mode
EN	End program

For additional information about automatic array capture, see Chapter 7, Arrays.

Virtual Axis

The DMC-1600 controller has an additional virtual axis designated as the N axis. This axis has no encoder and no DAC. However, it can be commanded by the commands:

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

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

ECAM Master Example

Suppose that the motion of the XY axes is constrained along a path that can be described by an electronic cam table. Further assume that the ecam master is not an external encoder but has to be a controlled variable.

This can be achieved by defining the N axis as the master with the command EAN and setting the modulo of the master with a command such as EMN= 4000. Next, the table is constructed. To move the constrained axes, simply command the N axis in the jog mode or with the PR and PA commands.

For example,

PAN = 2000

BGN

will cause the XY axes to move to the corresponding points on the motion cycle.

Sinusoidal Motion Example

The x axis must perform a sinusoidal motion of 10 cycles with an amplitude of 1000 counts and a frequency of 20 Hz.

This can be performed by commanding the X and N axes to perform circular motion. Note that the value of VS must be

$$VS=2\pi * R * F$$

where R is the radius, or amplitude and F is the frequency in Hz.

Set VA and VD to maximum values for the fastest acceleration

INSTRUCTION	INTERPRETATION
VMXN	Select axes
VA 68000000	Maximum Acceleration

VD 68000000	Maximum Deceleration
VS 125664	VS for 20 Hz
CR 1000, -90, 3600	Ten cycles
VE	
BGS	

Stepper Motor Operation

When configured for stepper motor operation, several commands are interpreted differently than from servo mode. The following describes operation with stepper motors.

Specifying Stepper Motor Operation

In order to command stepper motor operation, the appropriate stepper mode jumpers must be installed. See chapter 2 for this installation.

Stepper motor operation is specified by the command MT. The argument for MT is as follows:

2 specifies a stepper motor with active low step output pulses

-2 specifies a stepper motor with active high step output pulses

2.5 specifies a stepper motor with active low step output pulses and reversed direction

-2.5 specifies a stepper motor with active high step output pulse and reversed direction

Stepper Motor Smoothing

The command, KS, provides stepper motor smoothing. The effect of the smoothing can be thought of as a simple Resistor-Capacitor (single pole) filter. The filter occurs after the motion profiler and has the effect of smoothing out the spacing of pulses for a more smooth operation of the stepper motor. Use of KS is most applicable when operating in full step or half step operation. KS will cause the step pulses to be delayed in accordance with the time constant specified.

When operating with stepper motors, you will always have some amount of stepper motor smoothing, KS. Since this filtering effect occurs after the profiler, the profiler may be ready for additional moves before all of the step pulses have gone through the filter. It is important to consider this effect since steps may be lost if the controller is commanded to generate an additional move before the previous move has been completed. See the discussion below, *Monitoring Generated Pulses vs. Commanded Pulses.*

The general motion smoothing command, IT, can also be used. The purpose of the command, IT, is to smooth out the motion profile and decrease 'jerk' due to acceleration.

Monitoring Generated Pulses vs. Commanded Pulses

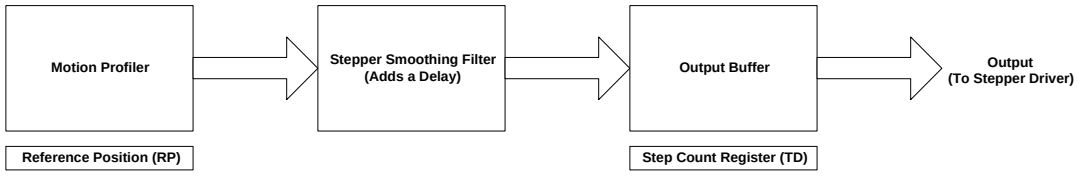
In general, it is necessary to make sure that the controller has completed generating all step pulses before making additional moves. If additional motion commands are given while step motor is already moving, some steps may be missed.* This is most particularly important when moving back and forth. For example, when operating with servo motors, the trippoint AM (After Motion) is used to determine when the motion profiler is complete and is prepared to execute a new motion command. However when operating in stepper mode, the controller may still be generating step pulses when the motion profiler is complete. This is caused by the stepper motor smoothing filter, KS. To understand this, consider the steps the controller executes to generate step pulses:

First, the controller generates a motion profile in accordance with the motion commands.

Second, the profiler generates pulses as prescribed by the motion profile. The pulses that are generated by the motion profiler can be monitored by the command, RP (Reference Position). RP gives the absolute value of the position as determined by the motion profiler. The command, DP, can be used to set the value of the reference position. For example, DP 0, defines the reference position of the X axis to be zero.

Third, the output of the motion profiler is filtered by the stepper smoothing filter. This filter adds a delay in the output of the stepper motor pulses. The amount of delay depends on the parameter which is specified by the command, KS. As mentioned earlier, there will always be some amount of stepper motor smoothing. The default value for KS is 2 which corresponds to a time constant of 6 sample periods.

Fourth, the output of the stepper smoothing filter is buffered and is available for input to the stepper motor driver. The pulses which are generated by the smoothing filter can be monitored by the command, TD (Tell Dual). TD gives the absolute value of the position as determined by actual output of the buffer. The command, DP sets the value of the step count register as well as the value of the reference position. For example, DP 0, defines the reference position of the X axis to be zero.



*Note: Although steps can be missed by over lapping moves, when the last stepper motion has been completed the stepper will be in the correct position.

Motion Complete Trippoint

When used in stepper mode, the MC command will hold up execution of the proceeding commands until the controller has generated the same number of steps out of the step count register as specified in the commanded position. The MC trippoint (Motion Complete) is generally more useful than AM trippoint (After Motion) since the step pulses can be delayed from the commanded position due to stepper motor smoothing.

Using an Encoder with Stepper Motors

An encoder may be used on a stepper motor to check the actual motor position with the commanded position. If an encoder is used, it must be connected to the main encoder input. Note: The auxiliary encoder is not available while operating with stepper motors. The position of the encoder can be interrogated by using the command, TP. The position value can be defined by using the command, DE.

Note: Closed loop operation with a stepper motor is not possible.

Command Summary - Stepper Motor Operation

COMMAND	DESCRIPTION
DE	Define Encoder Position (When using an encoder)
DP	Define Reference Position and Step Count Register
IT	Motion Profile Smoothing - Independent Time Constant
KS	Stepper Motor Smoothing

LC	Low Current Stepper Mode (toggles amp enable line when holding position)
MT	Motor Type (2,-2,2.5 or -2.5 for stepper motors)
RP	Report Commanded Position
TD	Report number of step pulses generated by controller
TP	Tell Position of Encoder

Operand Summary - Stepper Motor Operation

OPERAND	DESCRIPTION
_DEx	Contains the value of the step count register for the 'x' axis
_DPx	Contains the value of the main encoder for the 'x' axis
_ITx	Contains the value of the Independent Time constant for the 'x' axis
_KSx	Contains the value of the Stepper Motor Smoothing Constant for the 'x' axis
_MTx	Contains the motor type value for the 'x' axis
_RPx	Contains the commanded position generated by the profiler for the 'x' axis
_TDx	Contains the value of the step count register for the 'x' axis
_TPx	Contains the value of the main encoder for the 'x' axis

Stepper Position Maintenance Mode (SPM)

The Galil controller can be set into the Stepper Position Maintenance (SPM) mode to handle the event of stepper motor position error. The mode looks at position feedback from the main encoder and compares it to the commanded step pulses. The position information is used to determine if there is any significant difference between the commanded and the actual motor positions. If such error is detected, it is updated into a command value for operator use. In addition, the SPM mode can be used as a method to correct for friction at the end of a microstepping move. This capability provides closed-loop control at the application program level. SPM mode can be used with Galil and non-Galil step drives.

SPM mode is configured, executed, and managed with seven commands. This mode also utilizes the #POSERR automatic subroutine allowing for automatic user-defined handling of an error event.

Internal Controller Commands (user can query):

QS Error Magnitude (pulses)

User Configurable Commands (user can query & change):

OE Profiler Off-On Error

YA Step Drive Resolution (pulses / full motor step)

YB Step Motor Resolution (full motor steps / revolution)

YC Encoder Resolution (counts / revolution)

YR Error Correction (pulses)

YS Stepper Position Maintenance enable, status

A pulse is defined by the resolution of the step drive being used. Therefore, one pulse could be a full step, a half step or a microstep.

When a Galil controller is configured for step motor operation, the step pulse output by the controller is internally fed back to the auxiliary encoder register. For SPM the feedback encoder on the stepper will connect to the main encoder port. Enabling the SPM mode on a controller with YS=1 executes an internal monitoring of the auxiliary and main encoder registers for that axis or axes. Position error is then tracked in step pulses between these two registers (QS command).

$$QS = TD - \frac{TP \times YA \times YB}{YC}$$

Where TD is the auxiliary encoder register(step pulses) and TP is the main encoder register(feedback encoder). Additionally, YA defines the step drive resolution where YA = 1 for full stepping or YA = 2 for half stepping. The full range of YA is up to YA = 9999 for microstepping drives.

Error Limit

The value of QS is internally monitored to determine if it exceeds a preset limit of three full motor steps. Once the value of QS exceeds this limit, the controller then performs the following actions:

1. The motion is maintained or is stopped, depending on the setting of the OE command. If OE=0 the axis stays in motion, if OE=1 the axis is stopped.
2. YS is set to 2, which causes the automatic subroutine labeled #POSERR to be executed.

Correction

A correction move can be commanded by assigning the value of QS to the YR correction move command. The correction move is issued only after the axis has been stopped. After an error correction move has completed and QS is less than three full motor steps, the YS error status bit is automatically reset back to 1; indicating a cleared error.

Example: SPM Mode Setup

The following code demonstrates what is necessary to set up SPM mode for a full step drive, a half step drive, and a $1/64^{th}$ microstepping drive for an axis with a 1.8^{o} step motor and 4000 count/rev encoder. Note the necessary difference is with the YA command.

Full-Stepping Drive, X axis:

```
#SETUP
OE1;          Set the profiler to stop axis upon error
KS16;         Set step smoothing
MT-2;         Motor type set to stepper
YA1;          Step resolution of the full-step drive
YB200;        Motor resolution (full steps per revolution)
YC4000;       Encoder resolution (counts per revolution)
SHX;          Enable axis
```

WT50;	Allow slight settle time
YS1;	Enable SPM mode

Half-Stepping Drive, X axis:

#SETUP	
OE1;	Set the profiler to stop axis upon error
KS16;	Set step smoothing
MT-2;	Motor type set to stepper
YA2;	Step resolution of the half-step drive
YB200;	Motor resolution (full steps per revolution)
YC4000;	Encoder resolution (counts per revolution)
SHX;	Enable axis
WT50;	Allow slight settle time
YS1;	Enable SPM mode

1/64th Step Microstepping Drive, X axis:

#SETUP	
OE1;	Set the profiler to stop axis upon error
KS16;	Set step smoothing
MT-2;	Motor type set to stepper
YA64;	Step resolution of the microstepping drive
YB200;	Motor resolution (full steps per revolution)
YC4000;	Encoder resolution (counts per revolution)
SHX;	Enable axis
WT50;	Allow slight settle time
YS1;	Enable SPM mode

Example: Error Correction

The following code demonstrates what is necessary to set up SPM mode for the X axis, detect error, stop the motor, correct the error, and return to the main code. The drive is a full step drive, with a 1.8^{o} step motor and 4000 count/rev encoder.

#SETUP	
OE1;	Set the profiler to stop axis upon error
KS16;	Set step smoothing
MT-2,-2,-2,-2;	Motor type set to stepper
YA2;	Step resolution of the drive
YB200;	Motor resolution (full steps per revolution)
YC4000;	Encoder resolution (counts per revolution)

```
SHX;                    Enable axis
WT100;                  Allow slight settle time

#MOTION                 Perform motion
SP512;                  Set the speed
PR1000;                 Prepare mode of motion
BGX;                    Begin motion
#LOOP;JP#LOOP;          Keep thread zero alive for #POSERR to run in

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

#POSERR;                Automatic subroutine is called when YS=2
WT100;                  Wait helps user see the correction
spsave=_SPX;            Save current speed setting
JP#RETURN,_YSX<>2;      Return to thread zero if invalid error
SP64;                   Set slow speed setting for correction
MG"ERROR= ",_QSX
YRX=_QSX;               Else, error is valid, use QS for correction
MCX;                    Wait for motion to complete
MG"CORRECTED, ERROR NOW= ",_QSX
WT100;                  Wait helps user see the correction

#RETURN
SPX=spsave;             Return the speed to previous setting
REO;                    Return from #POSERR
```

Example: Friction Correction

The following example illustrates how the SPM mode can be useful in correcting for X axis friction after each move when conducting a reciprocating motion. The drive is a $1/64^{th}$ microstepping drive with a 1.8^{o} step motor and 4000 count/rev encoder.

```
#SETUP;                 Set the profiler to continue upon error
KS16;                   Set step smoothing
MT-2,-2,-2,-2;          Motor type set to stepper
YA64;                   Step resolution of the microstepping drive
YB200;                  Motor resolution (full steps per revolution)
YC4000;                 Encoder resolution (counts per revolution)
SHX;                    Enable axis
WT50;                   Allow slight settle time
YS1;                    Enable SPM mode
```

```
#MOTION;                    Perform motion
SP16384;                    Set the speed
PR10000;                    Prepare mode of motion
BGX;                        Begin motion
MCX
JS#CORRECT;                 Move to correction
#MOTION2
SP16384;                    Set the speed
PR-10000;                   Prepare mode of motion
BGX;                        Begin motion
MCX
JS#CORRECT;                 Move to correction
JP#MOTION
#CORRECT;                   Correction code
spx=_SPX
#LOOP;                      Save speed value
SP2048;                     Set a new slow correction speed
WT100;                      Stabilize
JP#END,@ABS[_QSX]<10;       End correction if error is within defined tolerance
YRX=_QSX;                   Correction move
MCX
WT100;                      Stabilize
JP#LOOP;                    Keep correcting until error is within tolerance
#END;                       End #CORRECT subroutine, returning to code
SPX=spx
EN
```

Dual Loop (Auxiliary Encoder)

The DMC-1600 provides an interface for a second encoder for each axis except for axes configured for stepper motor operation and axis used in circular compare. When used, the second encoder is typically mounted on the motor or the load, but may be mounted in any position. The most common use for the second encoder is backlash compensation, described below.

The second encoder may be a standard quadrature type, or it may provide pulse and direction. The controller also offers the provision for inverting the direction of the encoder rotation. The main and the auxiliary encoders are configured with the CE command. The command form is CE x,y,z,w (or a,b,c,d,e,f,g,h for controllers with more than 4 axes) where the parameters x,y,z,w each equal the sum of two integers m and n. m configures the main encoder and n configures the auxiliary encoder.

Using the CE Command

m=	Main Encoder	n=	Second Encoder
0	Normal quadrature	0	Normal quadrature
1	Pulse & direction	4	Pulse & direction
2	Reverse quadrature	8	Reversed quadrature

3	Reverse pulse & direction	12	Reversed pulse & direction

For example, to configure the main encoder for reversed quadrature, m=2, and a second encoder of pulse and direction, n=4, the total is 6, and the command for the X axis is

```
CE 6
```

Additional Commands for the Auxiliary Encoder

The command, DE x,y,z,w, can be used to define the position of the auxiliary encoders. For example,

```
DE 0,500,-30,300
```

sets their initial values.

The positions of the auxiliary encoders may be interrogated with the command, DE?. For example

```
DE ?,,?
```

returns the value of the X and Z auxiliary encoders.

The auxiliary encoder position may be assigned to variables with the instructions

```
V1= _DEX
```

The command, TD XYZW, returns the current position of the auxiliary encoder.

The command, DV XYZW, configures the auxiliary encoder to be used for backlash compensation.

Backlash Compensation

There are two methods for backlash compensation using the auxiliary encoders:

1. 7Continuous dual loop
2. Sampled dual loop

To illustrate the problem, consider a situation in which the coupling between the motor and the load has a backlash. To compensate for the backlash, position encoders are mounted on both the motor and the load.

The continuous dual loop combines the two feedback signals to achieve stability. This method requires careful system tuning, and depends on the magnitude of the backlash. However, once successful, this method compensates for the backlash continuously.

The second method, the sampled dual loop, reads the load encoder only at the end point and performs a correction. This method is independent of the size of the backlash. However, it is effective only in point-to-point motion systems which require position accuracy only at the endpoint.

Continuous Dual Loop - Example

Connect the load encoder to the main encoder port and connect the motor encoder to the dual encoder port. The dual loop method splits the filter function between the two encoders. It applies

the KP (proportional) and KI (integral) terms to the position error, based on the load encoder, and applies the KD (derivative) term to the motor encoder. This method results in a stable system.

The dual loop method is activated with the instruction DV (Dual Velocity), where

```
DV    1,1,1,1
```

activates the dual loop for the four axes and

```
DV    0,0,0,0
```

disables the dual loop.

Note that the dual loop compensation depends on the backlash magnitude, and in extreme cases will not stabilize the loop. The proposed compensation procedure is to start with KP=0, KI=0 and to maximize the value of KD under the condition DV1. Once KD is found, increase KP gradually to a maximum value, and finally, increase KI, if necessary.

Sampled Dual Loop - Example

In this example, we consider a linear slide which is run by a rotary motor via a lead screw. Since the lead screw has a backlash, it is necessary to use a linear encoder to monitor the position of the slide. For stability reasons, it is best to use a rotary encoder on the motor.

Connect the rotary encoder to the X-axis and connect the linear encoder to the auxiliary encoder of X. Assume that the required motion distance is one inch, and that this corresponds to 40,000 counts of the rotary encoder and 10,000 counts of the linear encoder.

The design approach is to drive the motor a distance, which corresponds to 40,000 rotary counts. Once the motion is complete, the controller monitors the position of the linear encoder and performs position corrections.

This is done by the following program.

Instruction	Interpretation
#DUALOOP	Label
CE 0	Configure encoder
DE0	Set initial value
PR 40000	Main move
BGX	Start motion
#Correct	Correction loop
AMX	Wait for motion completion
V1=10000-_DEX	Find linear encoder error
V2=-_TEX/4+V1	Compensate for motor error
JP#END,@ABS[V2]<2	Exit if error is small
PR V2*4	Correction move
BGX	Start correction
JP#CORRECT	Repeat
#END	
EN	

Motion Smoothing

The DMC-1600 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 and VT Commands (S curve profiling):

When operating with servo motors, motion smoothing can be accomplished with the IT and VT command. These commands filter the acceleration and deceleration functions to produce a smooth velocity profile. The resulting velocity profile, known as S curve, has continuous acceleration and results in reduced mechanical vibrations.

The smoothing function is specified by the following commands:

IT x,y,z,w	Independent time constant
VT n	Vector time constant

The command, IT, is used for smoothing independent moves of the type JG, PR, PA and the command, VT, is used to smooth vector moves of the type VM and LM.

The smoothing parameters, x,y,z,w and n are numbers between 0 and 1 and determine the degree of filtering. The maximum value of 1 implies no filtering, resulting in trapezoidal velocity profiles. Smaller values of the smoothing parameters imply heavier filtering and smoother moves.

The following example illustrates the effect of smoothing. Fig. 6.8 shows the trapezoidal velocity profile and the modified acceleration and velocity.

Note that the smoothing process results in longer motion time.

Example - Smoothing

PR 20000	Position
AC 100000	Acceleration
DC 100000	Deceleration
SP 5000	Speed
IT .5	Filter for S-curve
BG X	Begin

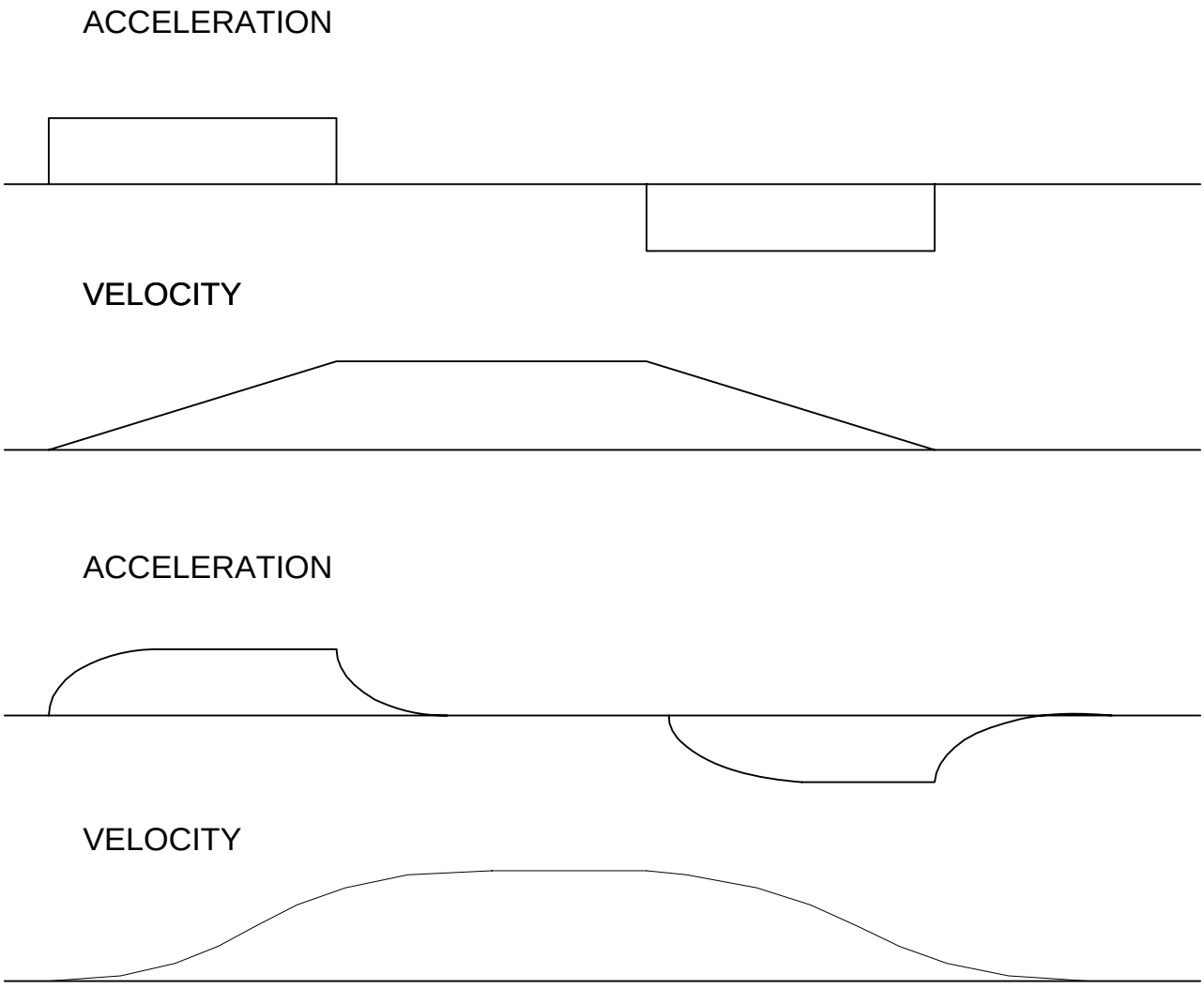


Figure 6.8 - Trapezoidal velocity and smooth velocity profiles

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. The HM command initializes the motor to the encoder index pulse in addition to the Home input. The configure command (CN) is used to define the polarity of the home input.

The Find Edge (FE) instruction is useful for initializing the motor to a home switch. The home switch is connected to the Homing Input. When the Find Edge command and Begin is used, the motor will accelerate up to the slew speed and slew until a transition is detected on the Homing line. The motor will then decelerate to a stop. A high deceleration value must be input before the find edge command is issued for the motor to decelerate rapidly after sensing the home switch.

The Home (HM) command can be used to position the motor on the index pulse after the home switch is detected. This allows for finer positioning on initialization. The command sequence HM and BG causes the following sequence of events to occur.

1. Upon begin, motor accelerates to the slew speed. The direction of its motion is determined by the state of the homing input. A zero (GND) will cause the motor to start in the forward direction; +5V will cause it to start in the reverse direction. The CN command is used to define the polarity of the home input.
2. Upon detecting the home switch changing state, the motor begins decelerating to a stop.
3. The motor then traverses very slowly back until the home switch toggles again.
4. The motor then traverses forward until the encoder index pulse is detected.
5. The DMC-1600 defines the home position (0) as the position at which the index was detected.

Example:

```
#HOME               Label
AC 1000000          Acceleration Rate
DC 1000000          Deceleration Rate
SP 5000             Speed for Home Search
HM X                Home X
BG X                Begin Motion
AM X                After Complete
MG "AT HOME"        Send Message
EN                  End
#EDGE               Label
AC 2000000          Acceleration rate
DC 2000000          Deceleration rate
SP 8000             Speed
FE Y                Find edge command
BG Y                Begin motion
AM Y                After complete
MG "FOUND HOME"     Send message
DP,0                Define position as 0
EN                  End
```

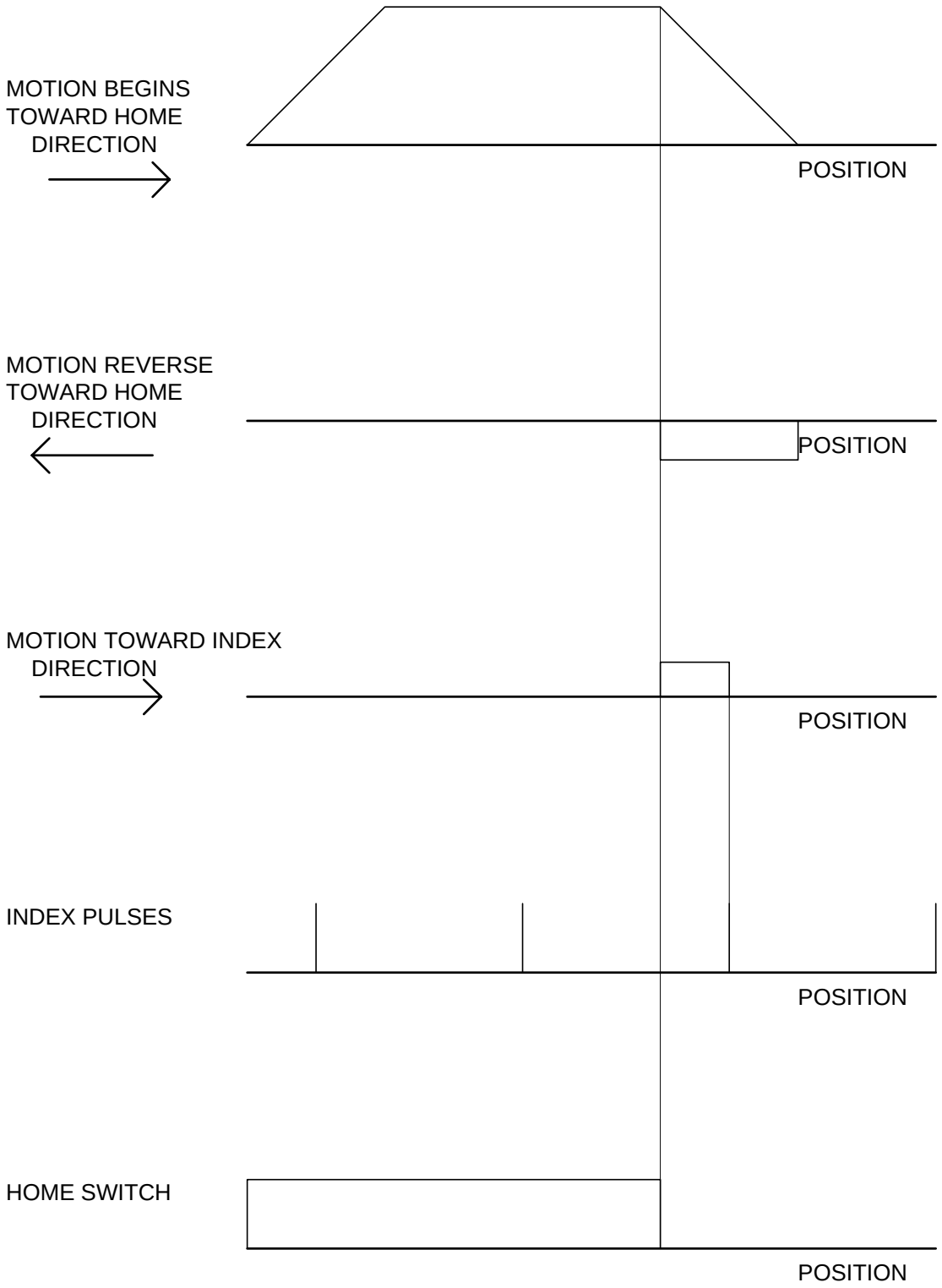


Figure 6.9 - Motion intervals in the Home sequence

High Speed Position Capture (The Latch Function)

Often it is desirable to capture the position precisely for registration applications. The DMC-1600 provides a position latch feature. This feature allows the position of the main or auxiliary encoders of X,Y,Z or W to be captured within 25 microseconds of an external low input signal. Faster latch times are available to <1 usec. Please contact Galil. The general inputs 1 through 4 and 9 thru 12 correspond to each axis.

1 through 4:		9 through 12	
IN1	X-axis latch	IN9	E-axis latch
IN2	Y-axis latch	IN10	F-axis latch
IN3	Z-axis latch	IN11	G-axis latch
IN4	W-axis latch	IN12	H-axis latch

Note: To insure a position capture within 25 microseconds, the input signal must be a transition from high to low.

The DMC-1600 software commands, AL and RL, are used to arm the latch and report the latched position. The steps to use the latch are as follows:

1. Give the AL XYZW command or ABCDEFGH for DMC-1680, to arm the latch for the main encoder and ALSXSYSZSW for the auxiliary encoders.
2. Test to see if the latch has occurred (Input goes low) by using the _AL X or Y or Z or W command. Example, V1=_ALX returns the state of the X latch into V1. V1 is 1 if the latch has not occurred.
3. After the latch has occurred, read the captured position with the RL XYZW command or _RL XYZW.

Note: The latch must be re-armed after each latching event.

Example:

#Latch	Latch program
JG,5000	Jog Y
BG Y	Begin motion on Y axis
AL Y	Arm Latch for Y axis
#Wait	#Wait label for loop
JP #Wait,_ALY=1	Jump to #Wait label if latch has not occurred
Result=_RLY	Set value of variable ‘Result’ equal to the report position of y axis
Result=	Print result
EN	End

Fast Firmware Operation

The DMC-1600 motion controllers can operate in a mode which allows for very fast servo update rates. This mode is known as 'fast mode' and allows the following update rates:

DMC-1610	125 usec
DMC-1620	125 usec
DMC-1630	250 usec
DMC-1640	250 usec
DMC-1650	375 usec

DMC-1660	375 usec
DMC-1670	500 usec
DMC-1680	500 usec

In order to run the DMC-1600 motion controller in fast mode, the fast firmware must be uploaded. This can be done through the Galil terminal software such as DMCTERM and WSDK. The fast firmware is included with the controller utilities. In order to set the desired update rates, use the command TM.

When operating the controller with the fast firmware, some functionality is limited. The following functions are disabled:

Gearing Mode
Ecam Mode
Pole (PL)
Analog Feedback (AF)
Stepper Motor Operation (MT 2,-2,2.5,-2.5)
Trippoints allowed only in thread 0
DMA channel
Tell Velocity Interrogation Command (TV)
Aux Encoders (TD)
Dual Velocity (DV)
Peak Torque Limit (TK)
Notch Filter (NB, NF, NZ)
Second Field of EI

Chapter 7 Application Programming

Overview

The DMC-1600 provides a powerful programming language that allows users to customize the controller for their particular application. Programs can be downloaded into the DMC-1600 memory freeing the host computer for other tasks. However, the host computer can send commands to the controller at any time, even while a program is being executed. Only ASCII commands can be used for application programming.

In addition to standard motion commands, the DMC-1600 provides commands that allow the DMC-1600 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-1600 provides user-defined variables, arrays and arithmetic functions. For example, with a cut-to-length operation, the length can be specified as a variable in a program which the operator can change as necessary.

The following sections in this chapter discuss all aspects of creating applications programs. The program memory size is 80 characters x 1000 lines.

Using the DMC-1600 Editor to Enter Programs

Application programs for the DMC-1600 may be created and edited either locally using the DMC-1600 editor or remotely using another editor and then downloading the program into the controller. (Galil's Terminal and SDK-software software provide an editor and UPLOAD and DOWNLOAD utilities).

The DMC-1600 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-1600 will return a colon.

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

Example:

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

Program Format

A DMC-1600 program consists of DMC-1600 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-1600 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-1600 programs must begin with a label and end with an End (EN) statement. Labels start with the pound (#) sign followed by a maximum of seven characters. The first character must be a letter; after that, numbers are permitted. Spaces are not permitted.

The maximum number of labels which may be defined is 254.

Valid labels

#BEGIN

#SQUARE

#X1

#BEGIN1

Invalid labels

#1Square

#123

A Simple Example Program:

#START	Beginning of the Program
PR 10000,20000	Specify relative distances on X and Y axes
BG XY	Begin Motion
AM	Wait for motion complete
WT 2000	Wait 2 sec
JP #START	Jump to label START
EN	End of Program

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

Special Labels

The DMC-1600 has some special labels, which are used to define input interrupt subroutines, limit switch subroutines, error handling subroutines, and command error subroutines. See section on Auto-Start Routine

The DMC-1600 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 on page on page 132.

#ININT	Label for Input Interrupt subroutine
#LIMSWI	Label for Limit Switch subroutine
#POSERR	Label for excess Position Error subroutine
#MCTIME	Label for timeout on Motion Complete trip point
#CMDERR	Label for incorrect command subroutine

Commenting Programs

Using the command, NO

The DMC-1600 provides a command, NO, for commenting programs. 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
NO 2-D CIRCULAR PATH
VMXY
NO VECTOR MOTION ON X AND Y
VS 10000
NO VECTOR SPEED IS 10000
VP -4000,0
NO BOTTOM LINE
CR 1500,270,-180
NO HALF CIRCLE MOTION
VP 0,3000
NO TOP LINE
CR 1500,90,-180
NO HALF CIRCLE MOTION
VE
NO END VECTOR SEQUENCE
BGS
NO BEGIN SEQUENCE MOTION
EN
NO 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.

Using REM Statements with the Galil Terminal Software.

If you are using Galil software to communicate with the DMC-1600 controller, you may also include REM statements. 'REM' statements begin with the word 'REM' and may be followed by any comments which are on the same line. The Galil terminal software will remove these statements when the program is downloaded to the controller. For example:

```
#PATH
REM 2-D CIRCULAR PATH
VMXY
REM VECTOR MOTION ON X AND Y
VS 10000
```

```
REM VECTOR SPEED IS 10000
VP -4000,0
REM BOTTOM LINE
CR 1500,270,-180
REM HALF CIRCLE MOTION
VP 0,3000
REM TOP LINE
CR 1500,90,-180
REM HALF CIRCLE MOTION
VE
REM END VECTOR SEQUENCE
BGS
REM BEGIN SEQUENCE MOTION
EN
REM END OF PROGRAM
```

These REM statements will be removed when this program is downloaded to the controller.

Executing Programs - Multitasking

The DMC-1600 can run up to 8 independent programs simultaneously. These programs are called threads and are numbered 0 through 7, where 0 is the main thread. Multitasking is useful for executing independent operations such as PLC functions that occur independently of motion.

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

1. Only the main thread, thread 0, may use the input command, IN.

2. When input interrupts are implemented for limit switches, position errors or command errors, the subroutines are executed as thread 0.

To begin execution of the various programs, use the following instruction:

```
XQ #A, n
```

Where n indicates the thread number. To halt the execution of any thread, use the instruction

```
HX n
```

where n is the thread number.

Note that both the XQ and HX commands can be performed by an executing program.

The example below produces a waveform on Output 1 independent of a move.

```
#TASK1          Task1 label
AT0             Initialize reference time
CB1             Clear Output 1
#LOOP1          Loop1 label
AT 10           Wait 10 msec from reference time
SB1             Set Output 1
AT -40          Wait 40 msec from reference time, then initialize reference
CB1             Clear Output 1
JP #LOOP1       Repeat Loop1
```

#TASK2	Task2 label
XQ #TASK1,1	Execute Task1
#LOOP2	Loop2 label
PR 1000	Define relative distance
BGX	Begin motion
AMX	After motion done
WT 10	Wait 10 msec
JP #LOOP2,@IN[2]=1	Repeat motion unless Input 2 is low
HX	Halt all tasks

The program above is executed with the instruction XQ #TASK2,0 which designates TASK2 as the main thread (i.e. Thread 0). #TASK1 is executed within TASK2.

Debugging Programs

The DMC-1600 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.

Data which is output from the controller is stored in an output FIFO buffer. The output FIFO buffer can store up to 512 characters of information. In normal operation, the controller places output into the FIFO buffer. The software on the host computer monitors this buffer and reads information as needed. When the trace mode is enabled, the controller will send information to the FIFO buffer at a very high rate. In general, the FIFO will become full since the software is unable to read the information fast enough. When the FIFO 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.

Break and Single Step Commands

The commands BK and SL can be used to single step through an application program. See the command reference for details.

Error Code Command

When there is a program error, the DMC-1600 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-1600 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-1610 will have a maximum of 8000 array elements in up to 30 arrays. If an array of 100 elements is defined, the command DM ? will return the value 7900 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 of the operands provide information which may be useful in debugging an application program. Below is a list of operands which are particularly valuable for program debugging. To display the value of an operand, the message command may be used. For example, since the operand, _ED contains the last line of program execution, the command MG _ED will display this line number.

_ED contains the last line of program execution. Useful to determine where program stopped.

_DL contains the number of available labels.

_UL contains the number of available variables.

_DA contains the number of available arrays.

_DM contains the number of available array elements.

_AB contains the state of the Abort Input

_LFx contains the state of the forward limit switch for the 'x' axis

_LRx contains the state of the reverse limit switch for the 'x' axis

Debugging Example:

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

:ED	Edit Mode
000 #A	Program Label
001 PR1000	Position Relative 1000
002 BGX	Begin
003 PR5000	Position Relative 5000
004 EN	End
<cntrl> Q	Quit Edit Mode
:XQ #A	Execute #A

?003 PR5000	Error on Line 3
:TC1	Tell Error Code
?7 Command not valid while running.	Command not valid while running
:ED 3	Edit Line 3
003 AMX;PR5000;BGX	Add After Motion Done
<cntrl> Q	Quit Edit Mode
:XQ #A	Execute #A

Program Flow Commands

The DMC-1600 provides instructions to control program flow. The DMC-1600 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-1600 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-1600 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 DMC-1600 can make decisions based on its own status or external events without intervention from a host computer.

DMC-1600 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. 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 DMC-1610, 1720, 1730, 1740. n=1 through 24 for DMC-1650, 1760, 1770, 1780. n=1 through 80 for DMC-17X8.
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	Halts program execution until n msec from reference time. AT 0 sets reference. AT n waits n msec from reference. AT -n waits n msec from reference and sets new reference after elapsed time.
AV n	Halts program execution until specified distance along a coordinated path has occurred.
WT n	Halts program execution until specified time in msec has elapsed.

Event Trigger Examples:

Event Trigger - Multiple Move Sequence

The AM trippoint is used to separate the two PR moves. If AM is not used, the controller returns a ? for the second PR command because a new PR cannot be given until motion is complete.

```
#TWOMOVE        Label
PR 2000         Position Command
BGX             Begin Motion
AMX             Wait for Motion Complete
PR 4000         Next Position Move
BGX             Begin 2nd move
EN              End program
```

Event Trigger - Set Output after Distance

Set output bit 1 after a distance of 1000 counts from the start of the move. The accuracy of the trippoint is the speed multiplied by the sample period.

```
#SETBIT         Label
SP 10000        Speed is 10000
PA 20000        Specify Absolute position
BGX             Begin motion
AD 1000         Wait until 1000 counts
SB1             Set output bit 1
EN              End program
```

Event Trigger - Repetitive Position Trigger

To set the output bit every 10000 counts during a move, the AR trippoint is used as shown in the next example.

```
#TRIP           Label
JG 50000        Specify Jog Speed
BGX;n=0         Begin Motion
#REPEAT         # Repeat Loop
AR 10000        Wait 10000 counts
TPX             Tell Position
SB1             Set output 1
WT50            Wait 50 msec
CB1             Clear output 1
n=n+1           Increment counter
JP #REPEAT,n<5  Repeat 5 times
STX             Stop
EN              End
```

Event Trigger - Start Motion on Input

This example waits for input 1 to go low and then starts motion. Note: The AI command actually halts execution of the program until the input occurs. If you do not want to halt the program sequences, you can use the Input Interrupt function (II) or use a conditional jump on an input, such as JP #GO,@IN[1] = -1.

```
#INPUT          Program Label
AI-1            Wait for input 1 low
PR 10000        Position command
BGX             Begin motion
EN              End program
```

Event Trigger - Set output when at speed

```
#ATSPEED        Program Label
JG 50000        Specify jog speed
AC 10000        Acceleration rate
BGX             Begin motion
ASX             Wait for at slew speed 50000
SB1             Set output 1
 EN             End program
```

Event Trigger - Change Speed along Vector Path

The following program changes the feed rate or vector speed at the specified distance along the vector. The vector distance is measured from the start of the move or from the last AV command.

```
#VECTOR         Label
VMXY;VS 5000    Coordinated path
VP 10000,20000  Vector position
VP 20000,30000  Vector position
VE              End vector
BGS             Begin sequence
AV 5000         After vector distance
VS 1000         Reduce speed
EN              End
```

Event Trigger - Multiple Move with Wait

This example makes multiple relative distance moves by waiting for each to be complete before executing new moves.

#MOVES	Label
PR 12000	Distance
SP 20000	Speed
AC 100000	Acceleration
BGX	Start Motion
AD 10000	Wait a distance of 10,000 counts
SP 5000	New Speed
AMX	Wait until motion is completed
WT 200	Wait 200 ms
PR -10000	New Position
SP 30000	New Speed
AC 150000	New Acceleration
BGX	Start Motion
EN	End

Define Output Waveform Using AT

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

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

Conditional Jumps

The DMC-1600 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 DMC-1600 to make decisions without a host computer. For example, the DMC-1600 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-1600 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-1600 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 parenthesis for proper evaluation by the controller. In addition, the DMC-1600 executes operations from left to right. For further information on Mathematical Expressions and the bit-wise operators '&' and '|', see pg 135.*

For example, using variables named V1, V2, V3 and V4:

JP #TEST, (V1<V2) & (V3<V4)

In this example, this statement will cause the program to jump to the label #TEST if V1 is less than V2 and V3 is less than V4. To illustrate this further, consider this same example with an additional condition:

JP #TEST, ((V1<V2) & (V3<V4)) | (V5<V6)

This statement will cause the program to jump to the label #TEST under two conditions; 1. If V1 is less than V2 and V3 is less than V4. OR 2. If V5 is less than V6.

Using the JP Command:

If the condition for the JP command is satisfied, the controller branches to the specified label or line number and continues executing commands from this point. If the condition is not satisfied, the controller continues to execute the next commands in sequence.

Conditional	Meaning
JP #Loop,COUNT<10	Jump to #Loop if the variable, COUNT, is less than 10
JS #MOVE2,@IN[1]=1	Jump to subroutine #MOVE2 if input 1 is logic level high. After the subroutine MOVE2 is executed, the program sequencer returns to the main program location where the subroutine was called.
JP #BLUE,@ABS[V2]>2	Jump to #BLUE if the absolute value of variable, V2, is greater than 2
JP #C,V1*V7<=V8*V2	Jump to #C if the value of V1 times V7 is less than or equal to the value of V8*V2
JP#A	Jump to #A

Example Using JP command:

Move the X motor to absolute position 1000 counts and back to zero ten times. Wait 100 msec between moves.

#BEGIN	Begin Program
COUNT=10	Initialize loop counter
#LOOP	Begin loop
PA 1000	Position absolute 1000
BGX	Begin move
AMX	Wait for motion complete
WT 100	Wait 100 msec
PA 0	Position absolute 0
BGX	Begin move
AMX	Wait for motion complete
WT 100	Wait 100 msec
COUNT=COUNT-1	Decrement loop counter
JP #LOOP,COUNT>0	Test for 10 times thru loop
EN	End Program

Using If, Else, and Endif Commands

The DMC-1600 provides a structured approach to conditional statements using IF, ELSE and ENDIF commands.

Using the IF and ENDIF Commands

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

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

Using the ELSE Command

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

Nesting IF Conditional Statements

The DMC-1600 allows for IF conditional statements to be included within other IF conditional statements. This technique is known as 'nesting' and the DMC-1600 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
ENDIF	End of 1st conditional statement
#WAIT	Label to be used for a loop
JP#WAIT,(@IN[1]=0) | (@IN[2]=0)	Loop until both input 1 and input 2 are not active
RI0	End Input Interrupt Routine without restoring trippoints

Subroutines

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

Example:

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

#M	Begin Main Program
CB1	Clear Output Bit 1 (pick up pen)
VP 1000,1000;LE;BGS	Define vector position; move pen
AMS	Wait for after motion trippoint
SB1	Set Output Bit 1 (put down pen)
JS #Square;CB1	Jump to square subroutine
EN	End Main Program
#Square	Square subroutine
V1=500;JS #L	Define length of side
V1=-V1;JS #L	Switch direction
EN	End subroutine

#L;PR V1,V1;BGX	Define X,Y; Begin X
AMX;BGY;AMY	After motion on X, Begin Y
EN	End subroutine

Stack Manipulation

It is possible to manipulate the subroutine stack by using the ZS command. Every time a JS instruction, interrupt or automatic routine (such as #POSERR or #LIMSWI) is executed, the subroutine stack is incremented by 1. Normally the stack is restored with an EN instruction. Occasionally it is desirable not to return back to the program line where the subroutine or interrupt was called. The ZS1 command clears 1 level of the stack. This allows the program sequencer to continue to the next line. The ZS0 command resets the stack to its initial value. For example, if a limit occurs and the #LIMSWI routine is executed, it is often desirable to restart the program sequence instead of returning to the location where the limit occurred. To do this, give a ZS command at the end of the #LIMSWI routine.

Auto-Start Routine

The DMC-1600 has a special label for automatic program execution. A program which has been saved into the controllers 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-1600 program sequences. The DMC-1600 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
#AUTOERR	Nonvolatile memory checksum error (use in conjunction with _RS)

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 automatic monitoring 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-1600 must be executing an applications program from memory. This can be a very simple program that does nothing but loop on a statement, such as #LOOP;JP

#LOOP;EN. Motion commands, such as JG 5000 can still be sent from the PC even while the "dummy" applications program is being executed.

:ED	Edit Mode
000 #LOOP	Dummy Program
001 JP #LOOP;EN	Jump to Loop
002 #LIMSWI	Limit Switch Label
003 MG "LIMIT OCCURRED"	Print Message
004 RE	Return to main program
<control> Q	Quit Edit Mode
:XQ #LOOP	Execute Dummy Program
:JG 5000	Jog
:BGX	Begin Motion

Now, when a forward limit switch occurs on the X axis, the #LIMSWI subroutine will be executed.

Notes regarding the #LIMSWI Routine:

1) The RE command is used to return from the #LIMSWI subroutine.

2) The #LIMSWI subroutine will be re-executed if the limit switch remains active.

The #LIMSWI routine is only executed when the motor is being commanded to move.

Example - Position Error

:ED	Edit Mode
000 #LOOP	Dummy Program
001 JP #LOOP;EN	Loop
002 #POSERR	Position Error Routine
003 V1=_TEX	Read Position Error
004 MG "EXCESS POSITION ERROR"	Print Message
005 MG "ERROR=",V1=	Print Error
006 RE	Return from Error
<control> Q	Quit Edit Mode
:XQ #LOOP	Execute Dummy Program
:JG 100000	Jog at High Speed
:BGX	Begin Motion

Now, if the position error on the X axis exceeds that specified by the ER command, the #POSERR routine will execute.

NOTE: The RE command is used to return from the #POSERR subroutine

NOTE: The #POSERR routine will continue to be executed until the position error is cleared (is less than the ER limit).

Example - Input Interrupt:

#A	Label
II1	Input Interrupt on 1
JG 30000,,,60000	Jog
BGXW	Begin Motion
#LOOP;JP#LOOP;EN	Loop

#ININT	Input Interrupt
STXW;AM	Stop Motion
#TEST;JP #TEST, @IN[1]=0	Test for Input 1 still low
JG 30000,,,6000	Restore Velocities
BGXW	Begin motion
RI0	Return from interrupt routine to Main Program and do not re-enable trippoints

Example - Motion Complete Timeout

#BEGIN	Begin main program
TW 1000	Set the time out to 1000 ms
PA 10000	Position Absolute command
BGX	Begin motion
MCX	Motion Complete trip point
EN	End main program
#MCTIME	Motion Complete Subroutine
MG “X fell short”	Send out a message
EN	End subroutine

This simple program will issue the message “X fell short” if the X axis does not reach the commanded position within 1 second of the end of the profiled move.

Example - Command Error

#BEGIN	Begin main program
IN "ENTER SPEED", SPEED	Prompt for speed
JG SPEED;BGX;	Begin motion
JP #BEGIN	Repeat
EN	End main program
#CMDERR	Command error utility
JP#DONE,_ED<>2	Check if error on line 2
JP#DONE,_TC<>6	Check if out of range
MG "SPEED TOO HIGH"	Send message
MG "TRY AGAIN"	Send message
ZS1	Adjust stack
JP #BEGIN	Return to main program
#DONE	End program if other error
ZS0	Zero stack
EN	End program

The above program prompts the operator to enter a jog speed. If the operator enters a number out of range (greater than 8 million), the #CMDERR routine will be executed prompting the operator to enter a new number.

In multitasking applications, there is an alternate method for handling command errors from different threads. Using the XQ command along with the special operands described below allows the controller to either skip or retry invalid commands.

OPERAND	FUNCTION
_ED1	Returns the number of the thread that generated an error

_ED2	Retry failed command (operand contains the location of the failed command)
_ED3	Skip failed command (operand contains the location of the command after the failed command)

The operands are used with the XQ command in the following format:

XQ _ED2 (or _ED3),_ED1,1

Where the ",1" at the end of the command line indicates a restart; therefore, the existing program stack will not be removed when the above format executes.

The following example shows an error correction routine which uses the operands.

Example - Command Error w/Multitasking

```
#A                      Begin thread 0 (continuous loop)
JP#A
EN                      End of thread 0

#B                      Begin thread 1
N=-1                    Create new variable
KP N                    Set KP to value of N, an invalid value
TY                      Issue invalid command
EN                      End of thread 1

#CMDERR                 Begin command error subroutine
IF _TC=6                If error is out of range (KP -1)
N=1                     Set N to a valid number
XQ _ED2,_ED1,1          Retry KP N command
ENDIF
IF _TC=1                If error is invalid command (TY)
XQ _ED3,_ED1,1          Skip invalid command
ENDIF
EN                      End of command error routine
```

Mathematical and Functional Expressions

Mathematical Operators

For manipulation of data, the DMC-1600 provides the use of the following mathematical operators:

OPERATOR	FUNCTION
+	Addition
-	Subtraction
*	Multiplication

/	Division
&	Logical And (Bit-wise)
|	Logical Or (On some computers, a solid vertical line appears as a broken line)
()	Parenthesis

The numeric range for addition, subtraction and multiplication operations is +/- 2,147,483,647.9999. The precision for division is 1/65,000.

Mathematical operations are executed from left to right. Calculations within a parentheses have precedence.

Examples:

SPEED=7.5*V1/2	The variable, SPEED, is equal to 7.5 multiplied by V1 and divided by 2
COUNT=COUNT+2	The variable, COUNT, is equal to the current value plus 2.
RESULT=_TPX-(@COS[45]*40)	Puts the position of X - 28.28 in RESULT. 40 * cosine of 45° is 28.28
TEMP=@IN[1]&@IN[2]	TEMP is equal to 1 only if Input 1 and Input 2 are high

Bit-Wise Operators

The mathematical operators & and | are bit-wise operators. The operator, &, is a Logical And. The operator, |, is a Logical Or. These operators allow for bit-wise operations on any valid DMC-1600 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 +/-.0001)
@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-1600 provides 254 variables. These variables can be numbers or strings. A program can be written in which certain parameters, such as position or speed, are defined as variables. The variables can later be assigned by the operator or determined by program calculations. For example, a cut-to-length application may require that a cut length be variable.

Example:

PR POSX	Assigns variable POSX to PR command
JG RPMY*70	Assigns variable RPMY multiplied by 70 to JG command.

Programmable Variables

The DMC-1600 allows the user to create up to 254 variables. Each variable is defined by a name which can be up to eight characters. The name must start with an alphabetic character, however, numbers are permitted in the rest of the name. Spaces are not permitted. Variable names should not be the same as DMC-1600 instructions. For example, PR is not a good choice for a variable name.

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 (2^{31})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-1600 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

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 Control

The example below reads the voltage of an X-Y joystick and assigns it to variables VX and VY to drive the motors at proportional velocities, where

10 Volts = 3000 rpm = 200000 c/sec

Speed/Analog input = 200000/10 = 20000

#JOYSTIK	Label
JG 0,0	Set in Jog mode
BGXY	Begin Motion
#LOOP	Loop
VX=@AN[1]*20000	Read joystick X
VY=@AN[2]*20000	Read joystick Y
JG VX,VY	Jog at variable VX,VY
JP#LOOP	Repeat
EN	End

Operands

Operands allow motion or status parameters of the DMC-1600 to be incorporated into programmable variables and expressions. Most DMC-1600 commands have an equivalent operand - which are designated by adding an underscore (_) prior to the DMC-1600 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-1600 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.
VAR1=_KPX*2	Assigns value from KPX multiplied by two to variable, VAR1.
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: _KPX=2 is invalid.

Special Operands (Keywords)

The DMC-1600 provides a few additional operands which give access to internal variables that are not accessible by standard DMC-1600 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 Summary, Chapter 11.

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-1600 provides array space for 8000 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.

Defining Arrays

An array is defined with the command DM. The user must specify a name and the number of entries to be held in the array. An array name can contain up to eight characters, starting with an uppercase alphabetic character. The number of entries in the defined array is enclosed in [ ].

Example:

DM POSX[7]	Defines an array names POSX with seven entries
DM SPEED[100]	Defines an array named speed with 100 entries
DM POSX[0]	Frees array space

Assignment of Array Entries

Like variables, each array element can be assigned a value. Assigned values can be numbers or returned values from instructions, functions and keywords.

Array elements are addressed starting at count 0. For example the first element in the POSX array (defined with the DM command, DM POSX[7]) would be specified as POSX[0].

Values are assigned to array entries using the equal sign. Assignments are made one element at a time by specifying the element number with the associated array name.

NOTE: Arrays must be defined using the command, DM, before assigning entry values.

Examples:

DM SPEED[10]	Dimension Speed Array
SPEED[1]=7650.2	Assigns the second element of the array, SPEED the value 7650.2
SPEED[1]=	Returns array element value
POSX[10]=_TPX	Assigns the 11th element of the array POSX the returned value from the tell position command.
CON[2]=@COS[POS]*2	Assigns the third element of the array CON the cosine of the variable POS multiplied by 2.
TIMER[1]=TIME	Assigns the second 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.

For example:

#A	Begin Program
COUNT=0;DM POS[10]	Initialize counter and define array
#LOOP	Begin loop
WT 10	Wait 10 msec
POS[COUNT]=_TPX	Record position into array element
POS[COUNT]=	Report position
COUNT=COUNT+1	Increment counter
JP #LOOP,COUNT<10	Loop until 10 elements have been stored
EN	End Program

The above example records 10 position values at a rate of one value per 10 msec. The values are stored in an array named POS. The variable, COUNT, is used to increment the array element counter. The above example can also be executed with the automatic data capture feature described below.

Uploading and Downloading Arrays to On Board Memory

Arrays may be uploaded and downloaded using the QU and QD commands.

QU array[],start,end,delim

QD array[],start,end

where array is an array name such as A[].

Start is the first element of array (default=0)

End is the last element of array (default=last element)

Delim specifies whether the array data is separated by a comma (delim=1) or a carriage return (delim=0).

The file is terminated using <control>Z, <control>Q, <control>D or \.

Automatic Data Capture into Arrays

The DMC-1600 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 four types of data can be captured and stored in four arrays. The capture rate or time interval may be specified. Recording can done as a one time event or as a circular continuous recording.

Command Summary - Automatic Data Capture

COMMAND	DESCRIPTION
RA n[],m[],o[],p[]	Selects up to four 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
_DEX	2nd encoder position (dual encoder)
_TPX	Encoder position
_TEX	Position error
_SHX	Commanded position
_RLX	Latched position
_TI	Inputs
_OP	Output
_TSX	Switches (only bit 0-4 valid)
_SCX	Stop code
_NOX	Status bits
_TTX	Torque (reports digital value +/-8097)

Note: X may be replaced by Y,Z or W for capturing data on other axes.

Operand Summary - Automatic Data Capture

_RC	Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress
_RD	Returns address of next array element.

Example - Recording into An Array

During a position move, store the X and Y positions and position error every 2 msec.

```
#RECORD                          Begin program
DM XPOS[300],YPOS[300]           Define X,Y position arrays
DM XERR[300],YERR[300]           Define X,Y error arrays
RA XPOS[],XERR[],YPOS[],YERR[]   Select arrays for capture
RD _TPX,_TEX,_TPY,_TEY           Select data types
PR 10000,20000                   Specify move distance
RC1                              Start recording now, at rate of 2 msec
BG XY                            Begin motion
#A;JP #A,_RC=1                   Loop until done
MG "DONE"                        Print message
EN                               End program
#PLAY                            Play back
N=0                              Initial Counter
JP# DONE,N>300                   Exit if done
N=                               Print Counter
X POS[N]=                        Print X position
Y POS[N]=                        Print Y position
XERR[N]=                         Print X error
YERR[N]=                         Print Y error
N=N+1                            Increment Counter
#DONE                            Done
EN
```

Deallocating Array Space

Array space may be deallocated using the DA command followed by the array name. DA*[0] deallocates all the arrays.

Input of Data (Numeric and String)

Input of Data

The command, IN, is used to prompt the user to input numeric or string data. Using the IN command, the user may specify a message prompt by placing a message in quotations. When the controller executes an IN command, the controller will wait for the input of data. The input data is assigned to the specified variable or array element.

An Example for Inputting Numeric Data

```
#A
IN "Enter Length", LENX
EN
```

In this example, the message “Enter Length” is displayed on the computer screen. The controller waits for the operator to enter a value. The operator enters the numeric value which is assigned to the variable, LENX.

Cut-to-Length Example

In this example, a length of material is to be advanced a specified distance. When the motion is complete, a cutting head is activated to cut the material. The length is variable, and the operator is prompted to input it in inches. Motion starts with a start button which is connected to input 1.

The load is coupled with a 2 pitch lead screw. A 2000 count/rev encoder is on the motor, resulting in a resolution of 4000 counts/inch. The program below uses the variable LEN, to length. The IN command is used to prompt the operator to enter the length, and the entered value is assigned to the variable LEN.

```
#BEGIN                          LABEL
AC 800000                       Acceleration
DC 800000                       Deceleration
SP 5000                         Speed
LEN=3.4                         Initial length in inches
#CUT                            Cut routine
AI1                             Wait for start signal
IN "enter Length(IN)", LEN      Prompt operator for length in inches
PR LEN *4000                    Specify position in counts
BGX                             Begin motion to move material
AMX                             Wait for motion done
SB1                             Set output to cut
WT100;CB1                       Wait 100 msec, then turn off cutter
JP #CUT                         Repeat process
EN                              End program
```

Inputting String Variables

String variables with up to six characters may input using the specifier, {Sn} where n represents the number of string characters to be input. If n is not specified, six characters will be accepted. For example, IN "Enter X,Y or Z", V{S} specifies a string variable to be input.

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 Value of KDX is ", _KDX
```

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;BGX;ASX
MG "The Speed is", _TVX {F5.1} {N}
MG "counts/sec"
EN
```

When #A is executed, the above example will appear on the screen as:

```
The speed is 50000 counts/sec
```

Using the MG Command to Configure Terminals

The MG command can be used to configure a terminal. Any ASCII character can be sent by using the format {^n} where n is any integer between 1 and 255.

Example:

```
MG {^07} {^255}
```

sends the ASCII characters represented by 7 and 255 to the bus.

Summary of Message Functions:

FUNCTION	DESCRIPTION
" "	Surrounds text string
{Fn.m}	Formats numeric values in decimal n digits to the right of the decimal point and m digits to the left
{$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

```
#DISPLAY                Label
DM POSX[7]              Define Array POSX with 7 entries
PR 1000                 Position Command
BGX                     Begin
AMX                     After Motion
V1=_TPX                 Assign Variable V1
POSX[0]=_TPX            Assign the first entry
V1=                     Print V1
```

Interrogation Commands

The DMC-1600 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.

Examples:

:DP21	Define position
:TPX	Tell position
0000000021	Default format
:PF4	Change format to 4 places
:TPX	Tell position
0021	New format
:PF-4	Change to hexadecimal format
:TPX	Tell Position
$0015	Hexadecimal value
:PF2	Format 2 places
:TPX	Tell Position
99	Returns 99 if position greater than 99

Removing Leading Zeros from Response to Interrogation Response

The leading zeros on data returned as a response to interrogation commands can be removed by the use of the command, LZ.

Example - Using the LZ command

LZ0	Disables the LZ function
TP	Tell Position Interrogation Command
-0000000009, 0000000005, 0000000000, 0000000007	Response from Interrogation Command (With Leading Zeros)
LZ1	Enables the LZ function
TP	Tell Position Interrogation Command
-9, 5, 0, 7	Response from Interrogation Command (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. For example:

Examples:

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.

:V1=10	Assign V1
:V1=	Return V1
0000000010.0000	Default format
:VF2.2	Change format
:V1=	Return V1
10.00	New format
:VF-2.2	Specify hex format
:V1=	Return V1
$0A.00	Hex value
:VF1	Change format
:V1=	Return V1
9	Overflow

Local Formatting of Variables

PF and VF commands are global format commands that effect 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. For example:

Examples:

: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-1600 position parameters such as PR, PA and VP have units of quadrature counts. Speed parameters such as SP, JG and VS have units of counts/sec. Acceleration parameters such as AC, DC, VA and VD have units of counts/sec^2. 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.

Example:

#RUN	Label
IN "ENTER # OF REVOLUTIONS",N1	Prompt for revs
PR N1*2000	Convert to counts
IN "ENTER SPEED IN RPM",S1	Prompt for RPMs
SP S1*2000/60	Convert to counts/sec
IN "ENTER ACCEL IN RAD/SEC2",A1	Prompt for ACCEL
AC A1*2000/(2*3.14)	Convert to counts/sec2
BG	Begin motion
EN	End program

Programmable Hardware I/O

Digital Outputs

The DMC-1600 has an 8-bit uncommitted output port for controlling external events. The DMC-1650 through DMC-1680 has an additional 8 outputs. The DMC-1600 has an additional 64 I/O (configured as inputs or outputs with CO command). 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).

For example:

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

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	Function
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 8-bit word using the instruction OP (Output Port). This instruction allows a single command to define the state of the entire 8-bit output port, where 2^0 is output 1, 2^1 is output 2 and so on. A 1 designates that the output is on.

For example:

Instruction	Function

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

#OUTPUT	Label
PR 2000	Position Command
BG	Begin
AM	After move
SB1	Set Output 1
WT 1000	Wait 1000 msec
CB1	Clear Output 1
EN	End

Digital Inputs

The DMC-1600 has eight digital inputs for controlling motion by local switches. The @IN[n] function returns the logic level of the specified input 1 through 8.

For example, a Jump on Condition instruction can be used to execute a sequence if a high condition is noted on an input 3. To halt program execution, the After Input (AI) instruction waits until the specified input has occurred.

Example:

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

Solution: Connect panel switch to input 1 of DMC-1600. High on input 1 means switch is in on position.

Instruction	Function
#S;JG 4000	Set speed
AI 1;BGX	Begin after input 1 goes high
AI -1;STX	Stop after input 1 goes low
AMX;JP #S	After motion, repeat
EN;	

Input Interrupt Function

The DMC-1600 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. A 1 designates that input to be enabled for an interrupt, where 2^0 is bit 1, 2^1 is bit 2 and so on. For example, II,,5 enables inputs 1 and 3 ($2^0 + 2^2 = 5$).

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

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

Examples - Input Interrupt

#A	Label #A
II 1	Enable input 1 for interrupt function
JG 30000,-20000	Set speeds on X and Y axes
BG XY	Begin motion on X and Y axes
#B	Label #B
TP XY	Report X and Y axes positions
WT 1000	Wait 1000 milliseconds
JP #B	Jump to #B
EN	End of program
#ININT	Interrupt subroutine
MG "Interrupt has occurred"	Displays the message
ST XY	Stops motion on X and Y axes
#LOOP;JP #LOOP,@IN[1]=0	Loop until Interrupt cleared
JG 15000,10000	Specify new speeds
WT 300	Wait 300 milliseconds
BG XY	Begin motion on X and Y axes
RI	Return from Interrupt subroutine

Analog Inputs

The DMC-1600 provides eight analog inputs. The value of these inputs in volts may be read using the @AN[n] function where n is the analog input 1 through 8. The resolution of the Analog-to-Digital conversion is 12 bits (16-bit ADC is available as an option). Analog inputs are useful for reading special sensors such as temperature, tension or pressure.

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

Example - Position Follower (Point-to-Point)

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

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

Instruction	Interpretation
#Points	Label
SP 7000	Speed

AC 80000;DC 80000	Acceleration
#Loop	
VP=@AN[1]*1000	Read analog input and compute position
PA VP	Command position
BGX	Start motion
AMX	After completion
JP #Loop	Repeat
EN	End

Example - Position Follower (Continuous Move)

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

Instruction	Interpretation
#Cont	Label
AC 80000;DC 80000	Acceleration rate
JG 0	Start job mode
BGX	Start motion
#Loop	
VP=@AN[1]*1000	Compute desired position
VE=VP-_TPX	Find position error
VEL=VE*20	Compute velocity
JG VEL	Change velocity
JP #Loop	Change velocity
EN	End

Example Applications

Wire Cutter

An operator activates a start switch. This causes a motor to advance the wire a distance of 10". When the motion stops, the controller generates an output signal which activates the cutter. Allowing 100 ms for the cutting completes the cycle.

Suppose that the motor drives the wire by a roller with a 2" diameter. Also assume that the encoder resolution is 1000 lines per revolution. Since the circumference of the roller equals 2π inches, and it corresponds to 4000 quadrature, one inch of travel equals:

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

This implies that a distance of 10 inches equals 6370 counts, and a slew speed of 5 inches per second, for example, equals 3185 count/sec.

The input signal may be applied to I1, for example, and the output signal is chosen as output 1. The motor velocity profile and the related input and output signals are shown in Fig. 7.1.

The program starts at a state that we define as #A. Here the controller waits for the input pulse on I1. As soon as the pulse is given, the controller starts the forward motion.

Upon completion of the forward move, the controller outputs a pulse for 20 ms and then waits an additional 80 ms before returning to #A for a new cycle.

Instruction	Function

#A	Label
AI1	Wait for input 1
PR 6370	Distance
SP 3185	Speed
BGX	Start Motion
AMX	After motion is complete
SB1	Set output bit 1
WT 20	Wait 20 ms
CB1	Clear output bit 1
WT 80	Wait 80 ms
JP #A	Repeat the process

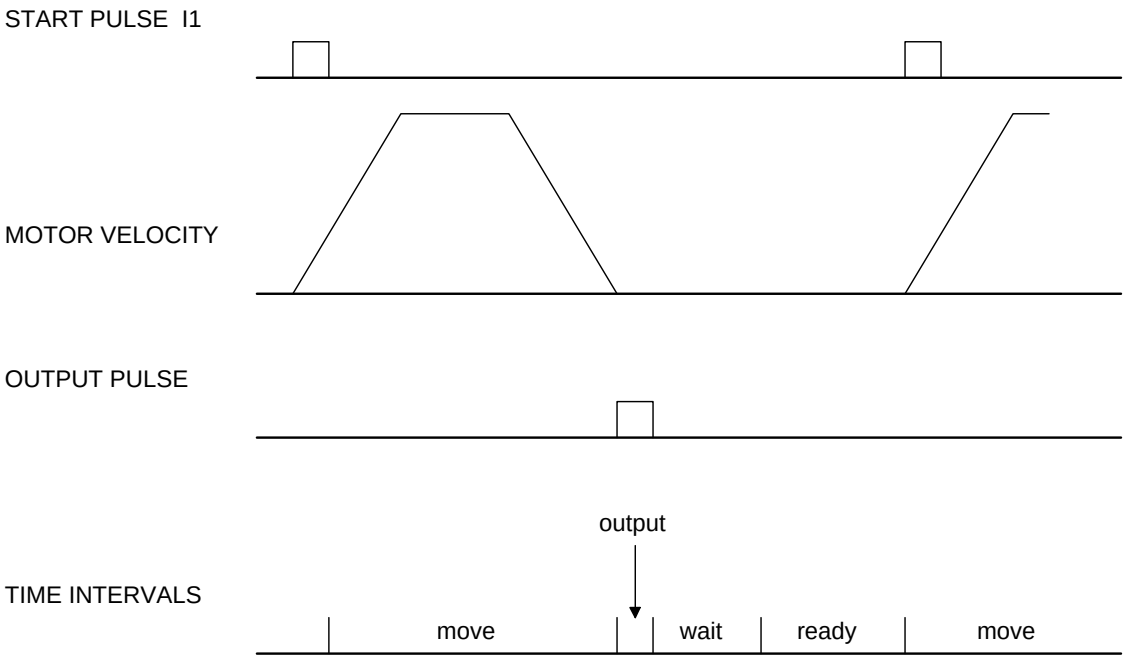


Figure 7.1 - Motor Velocity and the Associated Input/Output signals

X-Y Table Controller

An X-Y-Z system must cut the pattern shown in Fig. 7.2. The X-Y table moves the plate while the Z-axis raises and lowers the cutting tool.

The solid curves in Fig. 7.2 indicate sections where cutting takes place. Those must be performed at a feed rate of 1 inch per second. The dashed line corresponds to non-cutting moves and should be performed at 5 inch per second. The acceleration rate is 0.1 g.

The motion starts at point A, with the Z-axis raised. An X-Y motion to point B is followed by lowering the Z-axis and performing a cut along the circle. Once the circular motion is completed, the Z-axis is raised and the motion continues to point C, etc.

Assume that all of the 3 axes are driven by lead screws with 10 turns-per-inch pitch. Also assume encoder resolution of 1000 lines per revolution. This results in the relationship:

1 inch = 40,000 counts

and the speeds of

1 in/sec = 40,000 count/sec

5 in/sec = 200,000 count/sec

an acceleration rate of 0.1g equals

0.1g = 38.6 in/s2 = 1,544,000 count/s^2

Note that the circular path has a radius of 2" or 80000 counts, and the motion starts at the angle of 270° and traverses 360° in the CW (negative direction). Such a path is specified with the instruction

CR 80000,270,-360

Further assume that the Z must move 2" at a linear speed of 2" per second. The required motion is performed by the following instructions:

Instruction	Function
#A	Label
VM XY	Circular interpolation for XY
VP 160000,160000	Positions
VE	End Vector Motion
VS 200000	Vector Speed
VA 1544000	Vector Acceleration
BGS	Start Motion
AMS	When motion is complete
PR,,-80000	Move Z down
SP,,80000	Z speed
BGZ	Start Z motion
AMZ	Wait for completion of Z motion
CR 80000,270,-360	Circle
VE	
VS 40000	Feed rate
BGS	Start circular move
AMS	Wait for completion
PR,,80000	Move Z up
BGZ	Start Z move
AMZ	Wait for Z completion
PR -21600	Move X
SP 20000	Speed X
BGX	Start X
AMX	Wait for X completion
PR,,-80000	Lower Z
BGZ	
AMZ	
CR 80000,270,-360	Z second circle move
VE	
VS 40000	
BGS	
AMS	
PR,,80000	Raise Z

```
BGZ
AMZ
VP -37600,-16000          Return XY to start
VE
VS 200000
BGS
AMS
EN
```

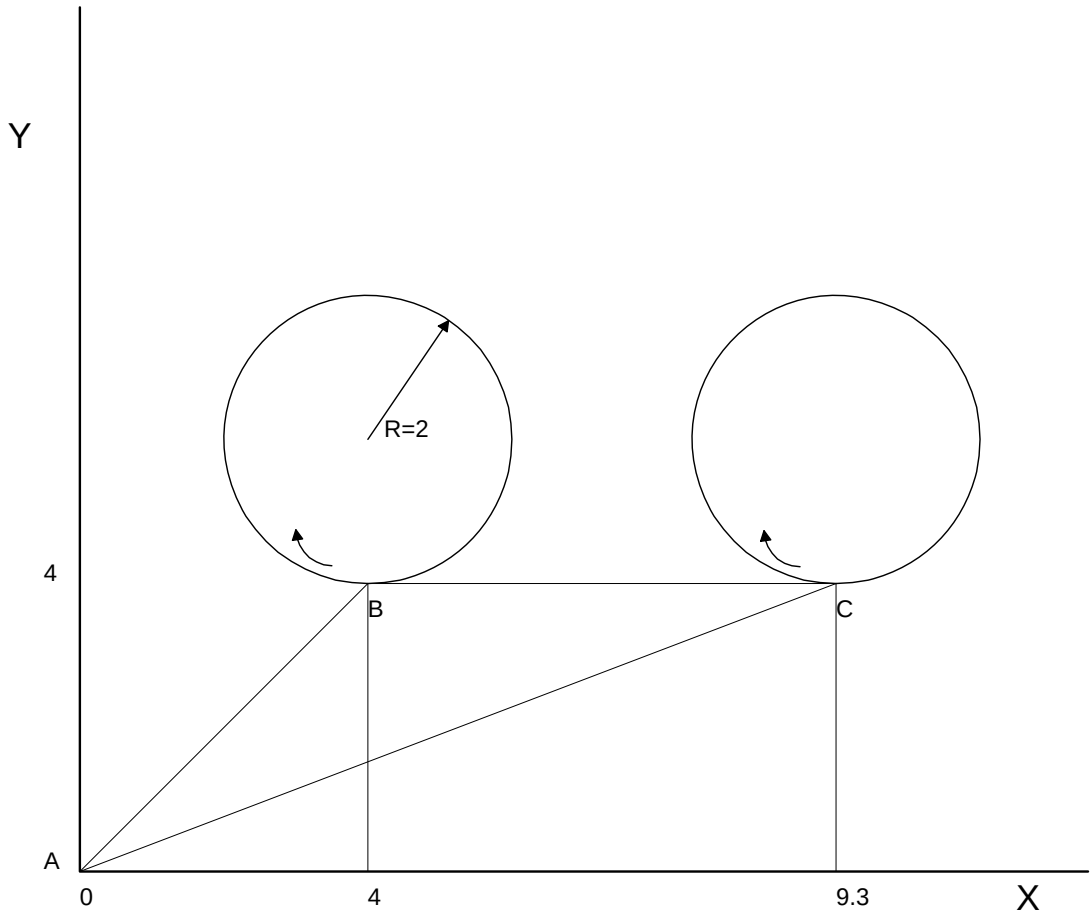


Figure 7.2 - Motor Velocity and the Associated Input/Output signals

Speed Control by Joystick

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

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

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

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

Speed = 20000 x VIN

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

Instruction
#A
JG0
BGX
#B
VIN=@AN[1]
VEL=VIN*20000
JG VEL
JP #B
EN

Position Control by Joystick

This system requires the position of the motor to be proportional to the joystick angle. Furthermore, the ratio between the two positions must be programmable. For example, if the control ratio is 5:1, it implies that when the joystick voltage is 5 Volts, corresponding to 1028 counts, the required motor position must be 5120 counts. The variable V3 changes the position ratio.

Instruction	Function
#A	Label
V3=5	Initial position ratio
DP0	Define the starting position
JG0	Set motor in jog mode as zero
BGX	Start
#B	
V1=@AN[1]	Read analog input
V2=V1*V3	Compute the desired position
V4=V2-_TPX-_TEX	Find the following error
V5=V4*20	Compute a proportional speed
JG V5	Change the speed
JP #B	Repeat the process
EN	End

Backlash Compensation by Sampled Dual-Loop

The continuous dual loop, enabled by the DV1 function is an effective way to compensate for backlash. In some cases, however, when the backlash magnitude is large, it may be difficult to stabilize the system. In those cases, it may be easier to use the sampled dual loop method described below.

This design example addresses the basic problems of backlash in motion control systems. The objective is to control the position of a linear slide precisely. The slide is to be controlled by a rotary motor, which is coupled to the slide by a lead screw. Such a lead screw has a backlash of 4 micron, and the required position accuracy is for 0.5 micron.

The basic dilemma is where to mount the sensor. If you use a rotary sensor, you get a 4 micron backlash error. On the other hand, if you use a linear encoder, the backlash in the feedback loop will cause oscillations due to instability.

An alternative approach is the dual-loop, where we use two sensors, rotary and linear. The rotary sensor assures stability (because the position loop is closed before the backlash) whereas the linear sensor provides accurate load position information. The operation principle is to drive the motor to a given rotary position near the final point. Once there, the load position is read to find the position error and the controller commands the motor to move to a new rotary position which eliminates the position error.

Since the required accuracy is 0.5 micron, the resolution of the linear sensor should preferably be twice finer. A linear sensor with a resolution of 0.25 micron allows a position error of +/-2 counts.

The dual-loop approach requires the resolution of the rotary sensor to be equal or better than that of the linear system. Assuming that the pitch of the lead screw is 2.5mm (approximately 10 turns per inch), a rotary encoder of 2500 lines per turn or 10,000 count per revolution results in a rotary resolution of 0.25 micron. This results in equal resolution on both linear and rotary sensors.

To illustrate the control method, assume that the rotary encoder is used as a feedback for the X-axis, and that the linear sensor is read and stored in the variable LINPOS. Further assume that at the start, both the position of X and the value of LINPOS are equal to zero. Now assume that the objective is to move the linear load to the position of 1000.

The first step is to command the X motor to move to the rotary position of 1000. Once it arrives we check the position of the load. If, for example, the load position is 980 counts, it implies that a correction of 20 counts must be made. However, when the X-axis is commanded to be at the position of 1000, suppose that the actual position is only 995, implying that X has a position error of 5 counts, which will be eliminated once the motor settles. This implies that the correction needs to be only 15 counts, since 5 counts out of the 20 would be corrected by the X-axis. Accordingly, the motion correction should be:

Correction = Load Position Error - Rotary Position Error

The correction can be performed a few times until the error drops below +/-2 counts. Often, this is performed in one correction cycle.

Example motion program:

Instruction	Function
#A	Label
DP0	Define starting positions as zero
LINPOS=0	
PR 1000	Required distance
BGX	Start motion
#B	
AMX	Wait for completion
WT 50	Wait 50 msec
LIN POS = _DEX	Read linear position
ER=1000-LINPOS-_TEX	Find the correction
JP #C,@ABS[ER]<2	Exit if error is small
PR ER	Command correction
BGX	
JP #B	Repeat the process
#C	
EN	

Chapter 8 Hardware & Software Protection

Introduction

The DMC-1600 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-1600 is an integral part of the machine, the engineer should design his overall system with protection against a possible component failure on the DMC-1600. Galil shall not be liable or responsible for any incidental or consequential damages.

Hardware Protection

The DMC-1600 includes hardware input and output protection lines for various error and mechanical limit conditions. These include:

Output Protection Lines

Amp Enable - This signal goes low when the motor off command is given, when the position error exceeds the value specified by the Error Limit (ER) command, or when off-on-error condition is enabled (OE1) and the abort command is given. Each axis amplifier has separate amplifier enable lines. This signal also goes low when the watch-dog timer is activated, or upon reset. *Note: The standard configuration of the AEN signal is TTL active low. Both the polarity and the amplitude can be changed if you are using the ICM-1900 interface board. To make these changes, see section entitled 'Amplifier Interface' pg 36.*

Error Output - The error signal output is available on the interconnect module as ERROR. This is a TTL signal which is low when the controller has an error.

Note: When the error signal is low, the LED on the controller will be on which indicates an error condition one of the following error conditions:

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

Input Protection Lines

Abort - A low input stops commanded motion instantly without a controlled deceleration. For any axis in which the Off-On-Error function is enabled, the amplifiers will be disabled. This could cause the motor to 'coast' to a stop. If the Off-On-Error function is not enabled, the motor will instantaneously stop and servo at the current position. The Off-On-Error function is further discussed in this chapter.

Forward Limit Switch - Low input inhibits motion in forward direction. If the motor is moving in the forward direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the forward direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user). The CN command can be used to change the polarity of the limit switches.

Reverse Limit Switch - Low input inhibits motion in reverse direction. If the motor is moving in the reverse direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the reverse direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user). The CN command can be used to change the polarity of the limit switches.

Software Protection

The DMC-1600 provides a programmable error limit. The error limit can be set for any number between 1 and 32767 using the ER n command. The default value for ER is 16384.

Example:

ER 200,300,400,500	Set X-axis error limit for 200, Y-axis error limit to 300, Z-axis error limit to 400 counts, W-axis error limit to 500 counts
ER,1,,10	Set Y-axis error limit to 1 count, set W-axis error limit to 10 counts.

The units of the error limit are quadrature counts. The error is the difference between the command position and actual encoder position. If the absolute value of the error exceeds the value specified by ER, the DMC-1600 will generate several signals to warn the host system of the error condition. These signals include:

Signal or Function	State if Error Occurs
# POSERR	Jumps to automatic excess position error subroutine
Error Light	Turns on
OE Function	Shuts motor off if OE1
AEN Output Line	Goes low

The Jump on Condition statement is useful for branching on a given error within a program. The position error of X,Y,Z and W can be monitored during execution using the TE command.

Programmable Position Limits

The DMC-1600 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-1600 will not accept position commands beyond the limit. Motion beyond the limit is also prevented.

Example:

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

BG XYZ	Begin

(motion stops at forward limits)

Off-On-Error

The DMC-1600 controller has a built in function which can turn off the motors under certain error conditions. This function is know as 'Off-On-Error". To activate the OE function for each axis, specify 1 for X,Y,Z and W axis. To disable this function, specify 0 for the axes. When this function is enabled, the specified motor will be disabled under the following 3 conditions:

1. The position error for the specified axis exceeds the limit set with the command, ER
2. The abort command is given
3. The abort input is activated with a low signal.

Note: If the motors are disabled while they are moving, they may 'coast' to a stop because they are no longer under servo control.

To re-enable the system, use the Reset (RS) or Servo Here (SH) command.

Examples:

OE 1,1,1,1	Enable off-on-error for X,Y,Z and W
OE 0,1,0,1	Enable off-on-error for Y and W axes and disable off-on-error for W and Z axes

Automatic Error Routine

The #POSERR label causes the statements following to be automatically executed if error on any axis exceeds the error limit specified by ER. The error routine must be closed with the RE command. The RE command returns from the error subroutine to the main program.

NOTE: The Error Subroutine will be entered again unless the error condition is gone.

Example:

#A;JP #A;EN	"Dummy" program
#POSERR	Start error routine on error
MG "error"	Send message
SB 1	Fire relay
STX	Stop motor
AMX	After motor stops
SHX	Servo motor here to clear error
RE	Return to main program

***NOTE:** An applications program must be executing for the #POSERR routine to function.*

Limit Switch Routine

The DMC-1600 provides forward and reverse limit switches which inhibit motion in the respective direction. There is also a special label for automatic execution of a limit switch subroutine. The #LIMSWI label specifies the start of the limit switch subroutine. This label causes the statements following to be automatically executed if any limit switch is activated and that axis motor is moving in that direction. The RE command ends the subroutine.

The state of the forward and reverse limit switches may also be tested during the jump-on-condition statement. The _LR condition specifies the reverse limit and _LF specifies the forward

limit. X,Y,Z, or W following LR or LF specifies the axis. The CN command can be used to configure the polarity of the limit switches.

Limit Switch Example:

```
#A;JP #A;EN              Dummy Program
#LIMSWI                  Limit Switch Utility
V1=_LFX                  Check if forward limit
V2=_LRX                  Check if reverse limit
JP#LF,V1=0               Jump to #LF if forward
JP#LR,V2=0               Jump to #LR if reverse
JP#END                   Jump to end
#LF                      #LF
MG "FORWARD LIMIT"       Send message
STX;AMX                  Stop motion
PR-1000;BGX;AMX          Move in reverse
JP#END                   End
#LR                      #LR
MG "REVERSE LIMIT"       Send message
STX;AMX                  Stop motion
PR1000;BGX;AMX           Move forward
#END                     End
RE                       Return to main program
```

NOTE: *An applications program must be executing for #LIMSWI to function.*

Chapter 9 Troubleshooting

Overview

The following discussion may help you get your system to work.

Potential problems have been divided into groups as follows:

1. Installation
2. Communication
3. Stability and Compensation
4. Operation

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

Installation

SYMPTOM	CAUSE	REMEDY
Motor runs away when connected to amplifier with no additional inputs.	Amplifier offset too large.	Adjust amplifier offset
Same as above, but offset adjustment does not stop the motor.	Damaged amplifier.	Replace amplifier.
Same as above, but offset adjustment does not stop the motor.	Damaged amplifier.	Replace amplifier.
Controller does not read changes in encoder position.	Wrong encoder connections.	Check encoder wiring.
Same as above	Bad encoder	Check the encoder signals. Replace encoder if necessary.
Same as above	Bad controller	Connect the encoder to different axis input. If it works, controller failure. Repair or replace.

Communication

SYMPTOM	CAUSE	REMEDY
Using the Galil provided terminal, cannot communicate with controller.	Address selection in communication does not match jumpers.	Check address jumper positions, and change if necessary. The addresses 1000 or 816 are recommended. Note -- for address 1000, jumper A2 and A4 . For address 816, jumper A7, A6, A3, A2.

Stability

SYMPTOM	CAUSE	REMEDY
Motor runs away when the loop is closed.	Wrong feedback polarity.	Invert the polarity of the loop by inverting the motor leads (brush type) or the encoder.
Motor oscillates.	Too high gain or too little damping.	Decrease KI and KP. Increase KD.

Operation

SYMPTOM	CAUSE	REMEDY
Controller rejects command. Responded with a ?	Anything.	Interrogate the cause with TC or TC1.
Motor does not complete move.	Noise on limit switches stops the motor.	To verify cause, check the stop code (SC). If caused by limit switch noise, reduce noise.
During a periodic operation, motor drifts slowly.	Encoder noise	Interrogate the position periodically. If controller states that the position is the same at different locations it implies encoder noise. Reduce noise. Use differential encoder inputs.
Same as above.	Programming error.	Avoid resetting position error at end of move with SH command.

Chapter 10 Theory of Operation

Overview

The following discussion covers the operation of motion control systems. A typical motion control system consists of the elements shown in Fig 10.1.

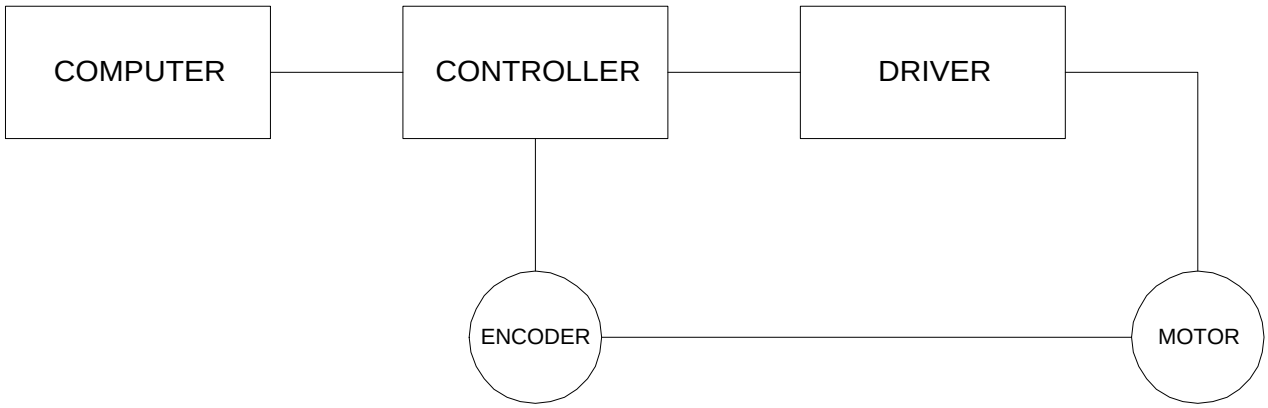


Figure 10.1 - Elements of Servo Systems

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

1. Closing the Loop
2. Motion Profiling
3. Motion Programming

The first level, the closing of the loop, assures that the motor follows the commanded position. This is done by closing the position loop using a sensor. The operation at the basic level of closing the loop involves the subjects of modeling, analysis, and design. These subjects will be covered in the following discussions.

The motion profiling is the generation of the desired position function. This function, R(t), describes where the motor should be at every sampling period. Note that the profiling and the closing of the loop are independent functions. The profiling function determines where the motor should be and the closing of the loop forces the motor to follow the commanded position

The highest level of control is the motion program. This can be stored in the host computer or in the controller. This program describes the tasks in terms of the motors that need to be controlled, the distances and the speed.

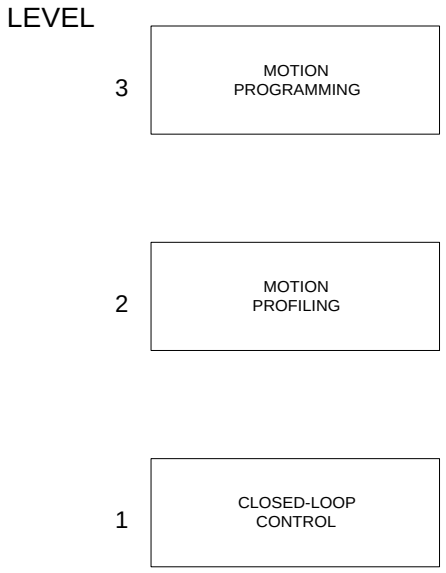


Figure 10.2 - Levels of Control Functions

The three levels of control may be viewed as different levels of management. The top manager, the motion program, may specify the following instruction, for example.

```
PR 6000,4000
SP 20000,20000
AC 200000,00000
BG X
AD 2000
BG Y
EN
```

This program corresponds to the velocity profiles shown in Fig. 10.3. Note that the profiled positions show where the motors must be at any instant of time.

Finally, it remains up to the servo system to verify that the motor follows the profiled position by closing the servo loop.

The following section explains the operation of the servo system. First, it is explained qualitatively, and then the explanation is repeated using analytical tools for those who are more theoretically inclined.

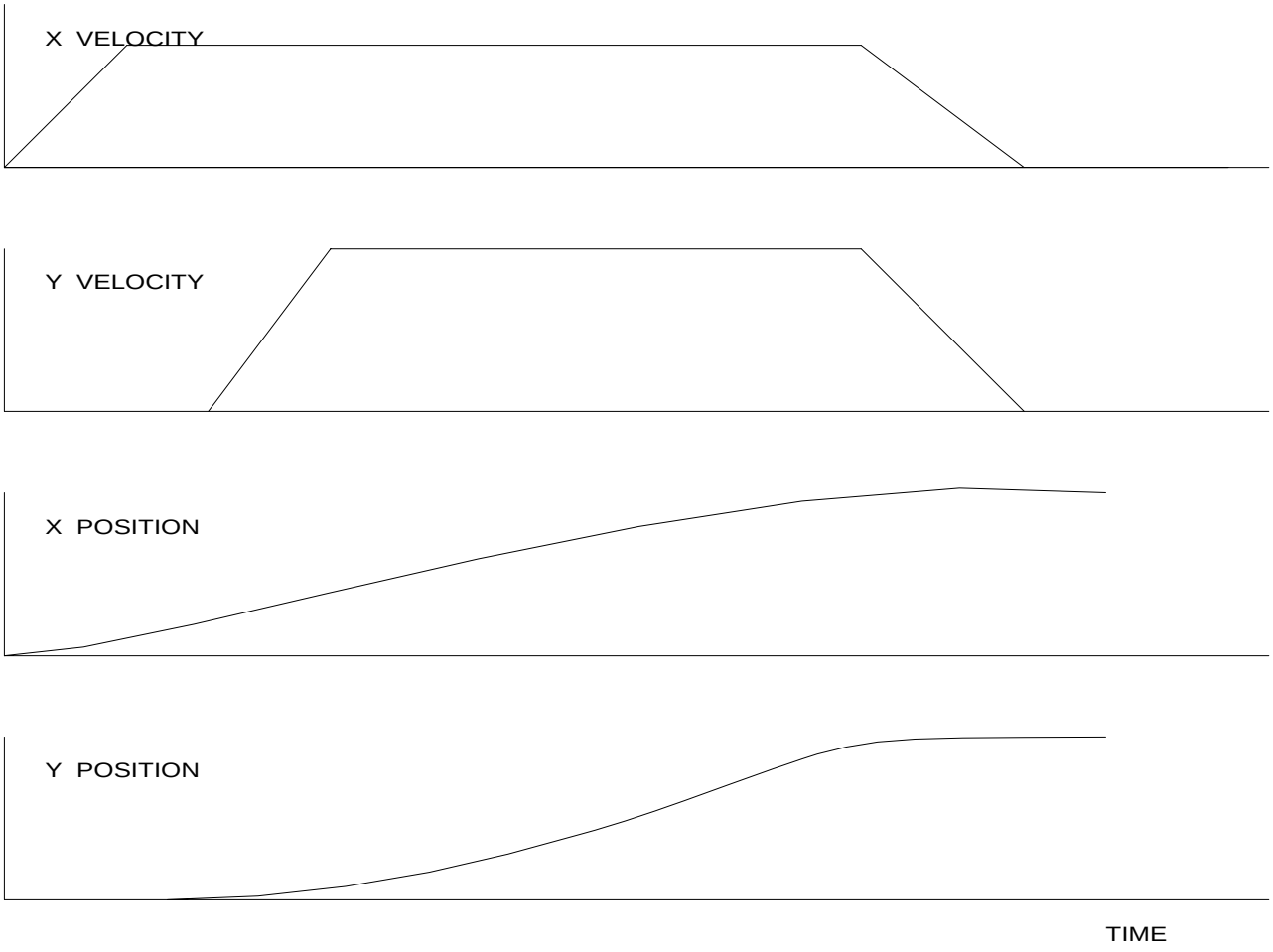


Figure 10.3 - Velocity and Position Profiles

Operation of Closed-Loop Systems

To understand the operation of a servo system, we may compare it to a familiar closed-loop operation, adjusting the water temperature in the shower. One control objective is to keep the temperature at a comfortable level, say 90 degrees F. To achieve that, our skin serves as a temperature sensor and reports to the brain (controller). The brain compares the actual temperature, which is called the feedback signal, with the desired level of 90 degrees F. The difference between the two levels is called the error signal. If the feedback temperature is too low, the error is positive, and it triggers an action which raises the water temperature until the temperature error is reduced sufficiently.

The closing of the servo loop is very similar. Suppose that we want the motor position to be at 90 degrees. The motor position is measured by a position sensor, often an encoder, and the position feedback is sent to the controller. Like the brain, the controller determines the position error, which is the difference between the commanded position of 90 degrees and the position feedback. The controller then outputs a signal that is proportional to the position error. This signal produces a proportional current in the motor, which causes a motion until the error is reduced. Once the error becomes small, the resulting current will be too small to overcome the friction, causing the motor to stop.

The analogy between adjusting the water temperature and closing the position loop carries further. We have all learned the hard way, that the hot water faucet should be turned at the "right" rate. If you turn it too slowly, the temperature response will be slow, causing discomfort. Such a slow reaction is called overdamped response.

The results may be worse if we turn the faucet too fast. The overreaction results in temperature oscillations. When the response of the system oscillates, we say that the system is unstable. Clearly, unstable responses are bad when we want a constant level.

What causes the oscillations? The basic cause for the instability is a combination of delayed reaction and high gain. In the case of the temperature control, the delay is due to the water flowing in the pipes. When the human reaction is too strong, the response becomes unstable.

Servo systems also become unstable if their gain is too high. The delay in servo systems is between the application of the current and its effect on the position. Note that the current must be applied long enough to cause a significant effect on the velocity, and the velocity change must last long enough to cause a position change. This delay, when coupled with high gain, causes instability.

This motion controller includes a special filter which is designed to help the stability and accuracy. Typically, such a filter produces, in addition to the proportional gain, damping and integrator. The combination of the three functions is referred to as a PID filter.

The filter parameters are represented by the three constants KP, KI and KD, which correspond to the proportional, integral and derivative term respectively.

The damping element of the filter acts as a predictor, thereby reducing the delay associated with the motor response.

The integrator function, represented by the parameter KI, improves the system accuracy. With the KI parameter, the motor does not stop until it reaches the desired position exactly, regardless of the level of friction or opposing torque.

The integrator also reduces the system stability. Therefore, it can be used only when the loop is stable and has a high gain.

The output of the filter is applied to a digital-to-analog converter (DAC). The resulting output signal in the range between +10 and -10 Volts is then applied to the amplifier and the motor.

The motor position, whether rotary or linear is measured by a sensor. The resulting signal, called position feedback, is returned to the controller for closing the loop.

The following section describes the operation in a detailed mathematical form, including modeling, analysis and design.

System Modeling

The elements of a servo system include the motor, driver, encoder and the controller. These elements are shown in Fig. 10.4. The mathematical model of the various components is given below.

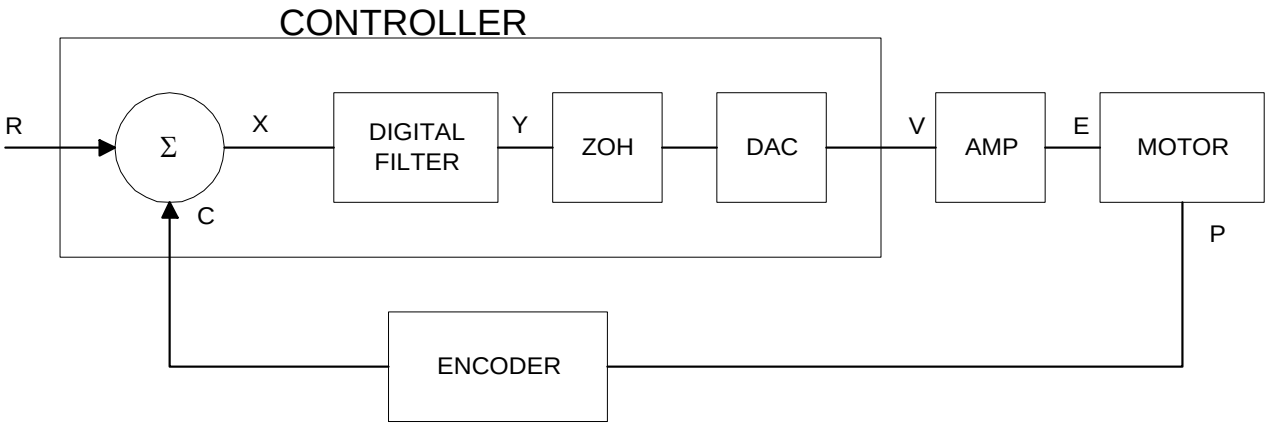


Figure 10.4 - Functional Elements of a Motion Control System

Motor-Amplifier

The motor amplifier may be configured in three modes:

1. Voltage Drive
2. Current Drive
3. Velocity Loop

The operation and modeling in the three modes is as follows:

Voltage Drive

The amplifier is a voltage source with a gain of Kv [V/V]. The transfer function relating the input voltage, V, to the motor position, P, is

$$P/V = K_V / \left[K_t S (ST_m + 1)(ST_e + 1) \right]$$

where

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

and

$$T_e = L/R \quad [s]$$

and the motor parameters and units are

K_t	Torque constant [Nm/A]
R	Armature Resistance Ω
J	Combined inertia of motor and load [kg.m^2]
L	Armature Inductance [H]

When the motor parameters are given in English units, it is necessary to convert the quantities to MKS units. For example, consider a motor with the parameters:

K_t = 14.16 oz - in/A = 0.1 Nm/A

R = 2 Ω

J = 0.0283 oz-in-s^2 = 2.10^{-4} kg . m^2

L = 0.004H

Then the corresponding time constants are

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

and

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

Assuming that the amplifier gain is Kv = 4, the resulting transfer function is

$$P/V = 40/[s(0.04s+1)(0.002s+1)]$$

Current Drive

The current drive generates a current I, which is proportional to the input voltage, V, with a gain of Ka. The resulting transfer function in this case is

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

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

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

If the motor is a DC brushless motor, it is driven by an amplifier that performs the commutation. The combined transfer function of motor amplifier combination is the same as that of a similar brush motor, as described by the previous equations.

Velocity Loop

The motor driver system may include a velocity loop where the motor velocity is sensed by a tachometer and is fed back to the amplifier. Such a system is illustrated in Fig. 10.5. Note that the transfer function between the input voltage V and the velocity ω is:

$$\omega /V = [K_a K_t/Js]/[1+K_a K_t K_g/Js] = 1/[K_g(sT_1+1)]$$

where the velocity time constant, T1, equals

$$T1 = J/K_a K_t K_g$$

This leads to the transfer function

$$P/V = 1/[K_g s(sT1+1)]$$

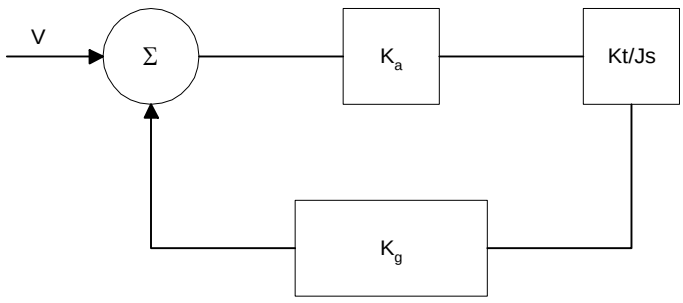
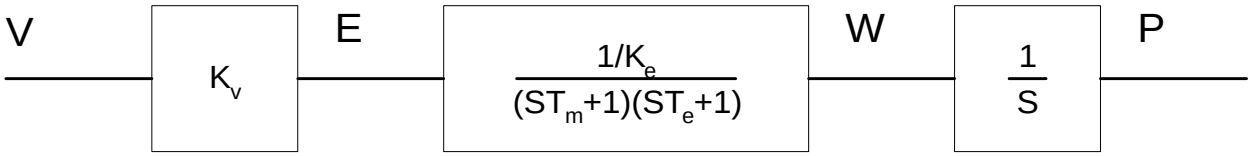


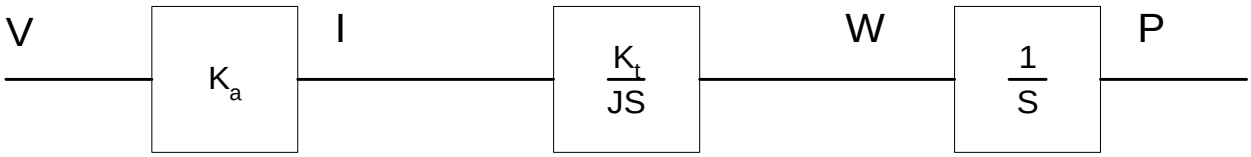
Figure 10.5 - Elements of velocity loops

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

VOLTAGE SOURCE



CURRENT SOURCE



VELOCITY LOOP

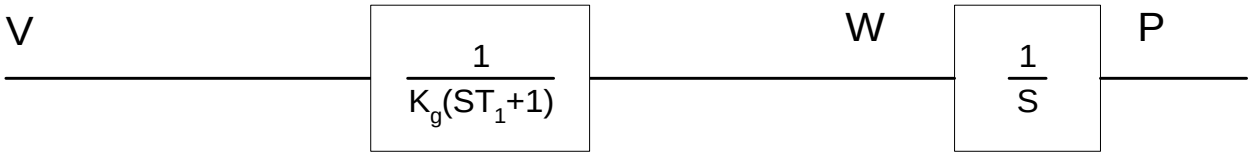


Figure 10.6 - Mathematical model of the motor and amplifier in three operational modes

Encoder

The encoder generates N pulses per revolution. It outputs two signals, Channel A and B, which are in quadrature. Due to the quadrature relationship between the encoder channels, the position resolution is increased to 4N quadrature counts/rev.

The model of the encoder can be represented by a gain of

$K_f = 4N/2\pi$ [count/rad]

For example, a 1000 lines/rev encoder is modeled as

$K_f = 638$

DAC

The DAC or D-to-A converter converts a 16-bit number to an analog voltage. The input range of the numbers is 65536 and the output voltage range is +/-10V or 20V. Therefore, the effective gain of the DAC is

K= 20/65536 = 0.0003 [V/count]

Digital Filter

The digital filter has three element in series: PID, low-pass and a notch filter. The transfer function of the filter. The transfer function of the filter elements are:

PID $D(z) = \frac{K(Z-A)}{Z} + \frac{CZ}{Z-1}$

Low-pass $L(z) = \frac{1-B}{Z-B}$

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

The filter parameters, K, A, C and B are selected by the instructions KP, KD, KI and PL, respectively. The relationship between the filter coefficients and the instructions are:

K = (KP + KD) • 4

A = KD/(KP + KD)

C = KI/2

B = PL

The PID and low-pass elements are equivalent to the continuous transfer function G(s).

G(s) = (P + sD + I/s) ∗ a/(S+a)

P = 4KP

D = 4T • KD

I = KI/2T

a = 1/T ln (1/B)

where T is the sampling period.

For example, if the filter parameters of the DMC-1600 are

KP = 4

KD = 36

KI = 2

PL = 0.75

T = 0.001 s

the digital filter coefficients are

K = 160

A = 0.9

C = 1

a = 250 rad/s

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

$$G(s) = [16 + 0.144s + 1000/s\} * 250/ (s+250)$$

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

The effect of the notch filter is to cancel the resonance affect by placing the complex zeros on top of the resonance poles. The notch poles, P and p, are programmable and are selected to have sufficient damping. It is best to select the notch parameters by the frequency terms. The poles and zeros have a frequency in Hz, selected by the command NF. The real part of the poles is set by NB and the real part of the zeros is set by NZ.

The most simple procedure for setting the notch filter, identify the resonance frequency and set NF to the same value. Set NB to about one half of NF and set NZ to a low value between zero and 5.

ZOH

The ZOH, or zero-order-hold, represents the effect of the sampling process, where the motor command is updated once per sampling period. The effect of the ZOH can be modeled by the transfer function

$$H(s) = 1/(1+sT/2)$$

If the sampling period is T = 0.001, for example, H(s) becomes:

$$H(s) = 2000/(s+2000)$$

However, in most applications, H(s) may be approximated as one.

This completes the modeling of the system elements. Next, we discuss the system analysis.

System Analysis

To analyze the system, we start with a block diagram model of the system elements. The analysis procedure is illustrated in terms of the following example.

Consider a position control system with the DMC-1600 controller and the following parameters:

$K_t = 0.1$	Nm/A	Torque constant
$J = 2.10^{-4}$	$kg.m^2$	System moment of inertia
R = 2	Ω	Motor resistance
$K_a = 4$	Amp/Volt	Current amplifier gain
KP = 12.5		Digital filter gain
KD = 245		Digital filter zero
KI = 0		No integrator
N = 500	Counts/rev	Encoder line density
T = 1	ms	Sample period

The transfer function of the system elements are:

Motor

$$M(s) = P/I = Kt/Js2 = 500/s^2 \ [rad/A]$$

Amp

$K_a = 4$ [Amp/V]

DAC

$K_d = 0.0003$ [V/count]

Encoder

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

ZOH

$2000/(s+2000)$

Digital Filter

KP = 12.5, KD = 245, T = 0.001

Therefore,

$D(z) = 1030\ (z-0.95)/Z$

Accordingly, the coefficients of the continuous filter are:

P = 50

D = 0.98

The filter equation may be written in the continuous equivalent form:

$G(s) = 50 + 0.98s = .098\ (s+51)$

The system elements are shown in Fig. 10.7.

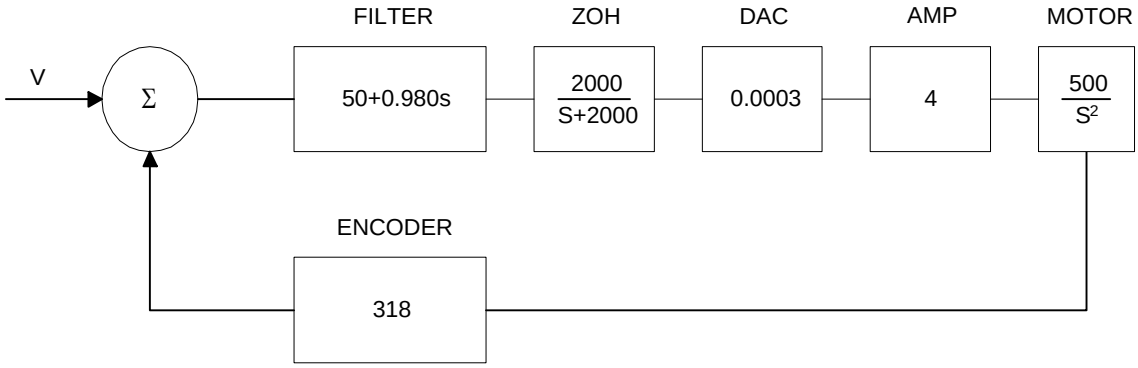


Figure 10.7 - Mathematical model of the control system

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

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

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

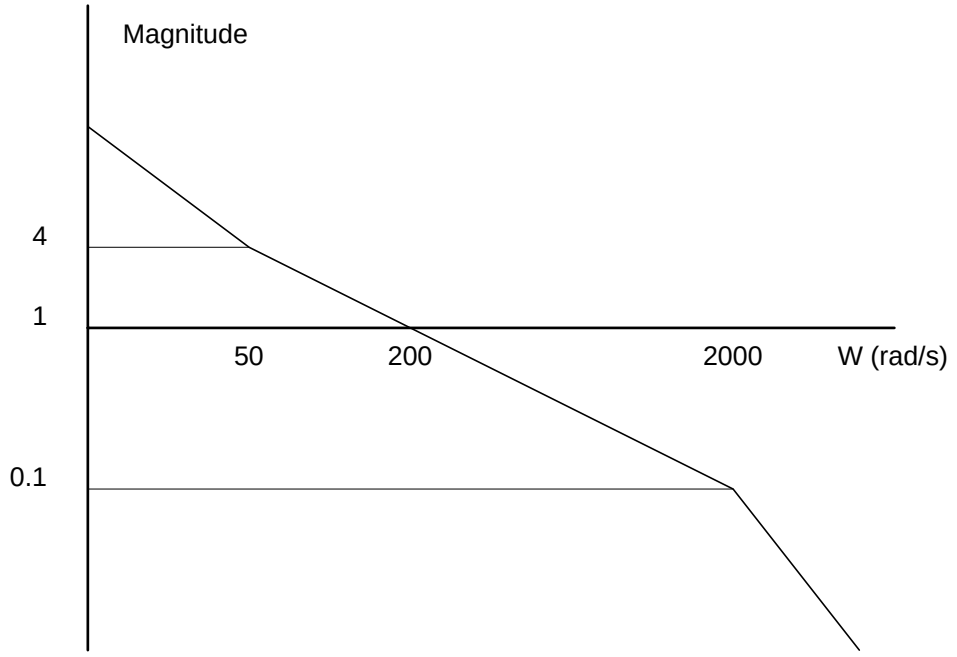


Figure 10.8 - Bode plot of the open loop transfer function

For the given example, the crossover frequency was computed numerically resulting in 200 rad/s.

Next, we determine the phase of A(s) at the crossover frequency.

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

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

$$\alpha = 76° - 180° - 6° = -110°$$

Finally, the phase margin, PM, equals

$$PM = 180° + \alpha = 70°$$

As long as PM is positive, the system is stable. However, for a well damped system, PM should be between 30 degrees and 45 degrees. The phase margin of 70 degrees given above indicated overdamped response.

Next, we discuss the design of control systems.

System Design and Compensation

The closed-loop control system can be stabilized by a digital filter, which is preprogrammed in the DMC-1600 controller. The filter parameters can be selected by the user for the best compensation. The following discussion presents an analytical design method.

The Analytical Method

The analytical design method is aimed at closing the loop at a crossover frequency, ω_c, with a phase margin PM. The system parameters are assumed known. The design procedure is best illustrated by a design example.

Consider a system with the following parameters:

K_t	Nm/A	Torque constant

$J = 2.10^{-4}$	$kg.m^2$	System moment of inertia
$R = 2$	Ω	Motor resistance
$K_a = 2$	Amp/Volt	Current amplifier gain
$N = 1000$	Counts/rev	Encoder line density

The DAC of the DMC-1600 outputs +/-10V for a 14-bit command of +/-8192 counts.

The design objective is to select the filter parameters in order to close a position loop with a crossover frequency of $\omega_c = 500$ rad/s and a phase margin of 45 degrees.

The first step is to develop a mathematical model of the system, as discussed in the previous system.

Motor

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

Amp

$$K_a = 2 \qquad [Amp/V]$$

DAC

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

Encoder

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

ZOH

$$H(s) = 2000/(s+2000)$$

Compensation Filter

$$G(s) = P + sD$$

The next step is to combine all the system elements, with the exception of G(s), into one function, L(s).

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

Then the open loop transfer function, A(s), is

$$A(s) = L(s)\, G(s)$$

Now, determine the magnitude and phase of L(s) at the frequency $\omega_c = 500$.

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

This function has a magnitude of

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

and a phase

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

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

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

$$Arg\ [A(j500)] = -135°$$

However, since

$$A(s) = L(s)\, G(s)$$

then it follows that G(s) must have magnitude of

$$|G(j500)| = |A(j500)/L(j500)| = 160$$

and a phase

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

In other words, we need to select a filter function G(s) of the form

$$G(s) = P + sD$$

so that at the frequency ω_c =500, the function would have a magnitude of 160 and a phase lead of 59 degrees.

These requirements may be expressed as:

$$|G(j500)| = |P + (j500D)| = 160$$

and

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

The solution of these equations leads to:

$$P = 160\cos 59° = 82.4$$

$$500D = 160\sin 59° = 137$$

Therefore,

$$D = 0.274$$

and

$$G = 82.4 + 0.2744s$$

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

$$D(z) = 4KP + 4KD(1-z^{-1})$$

where

$$P = 4 * KP$$

$$D = 4 * KD * T$$

and

$$4 * KD = D/T$$

Assuming a sampling period of T=1ms, the parameters of the digital filter are:

KP = 20.6

KD = 68.6

The DMC-1600 can be programmed with the instruction:

KP 20.6

KD 68.6

In a similar manner, other filters can be programmed. The procedure is simplified by the following table, which summarizes the relationship between the various filters.

Equivalent Filter Form

	DMC-1600
Digital	$D(z) = [K(z-A/z) + Cz/(z-1)] * (1-B)/(Z-B)$
Digital	$D(z) = [4\ KP + 4\ KD(1-z^{-1}) + KI/2(1-z^{-1})] * (1-B)/(Z-B)$
KP, KD, KI, PL	$K = (KP + KD) \cdot 4$
	$A = KD/(KP+KD)$
	$C = KI/2$
	$B = PL$
Continuous	$G(s) = (P + Ds + I/s) * a/s+a$
PID, T	$P = 4\ KP$
	$D = 4\ T*KD$
	$I = KI/2T$
	$a = 1/T\ \ln(1/PL)$

Appendices

Electrical Specifications

Servo Control

ACMD Amplifier Command:	+/-10 Volts analog signal. Resolution 16-bit DAC or .0003 Volts. 3 mA maximum
A+,A-,B+,B-,IDX+,IDX- Encoder and Auxiliary	TTL compatible, but can accept up to +/-12 Volts. Quadrature phase on CHA,CHB. Can accept single-ended (A+,B+ only) or differential (A+,A-,B+,B-). Maximum A,B edge rate: 12 MHz. Minimum IDX pulse width: 80 nsec.

Stepper Control

Pulse	TTL (0-5 Volts) level at 50% duty cycle. 3,000,000 pulses/sec maximum frequency
Direction	TTL (0-5 Volts)

Input/Output

Uncommitted Inputs, Limits, Home, Abort Inputs:	2.2K ohm in series with optoisolator. Active high or low requires at least 2mA to activate. Can accept up to 28 Volts without additional series resistor. Above 28 Volts requires additional resistor.
AN[1] thru AN[8] Analog Inputs:	Standard configuration is +/-10 Volt. 12-Bit Analog-to-Digital converter. 16-bit optional.
OUT[1] thru OUT[8] Outputs:	TTL.

Power Requirement

+5V	750 mA
+12V	40 mA
-12V	40mA

Performance Specifications

	Normal Firmware	Fast Firmware
Minimum Servo Loop Update Time:		
DMC-1610	250 µsec	125 µsec
DMC-1620	250 µsec	125 µsec
DMC-1630	375 µsec	250 µsec
DMC-1640	375 µsec	250 µsec
Position Accuracy:	+/-1 quadrature count	
Velocity Accuracy:		
Long Term	Phase-locked, better than .005%	
Short Term	System dependent	
Position Range:	+/-2147483647 counts per move	
Velocity Range:	Up to 12,000,000 counts/sec servo; 3,000,000 pulses/sec-stepper	
Velocity Resolution:	2 counts/sec	
Motor Command Resolution:	16 bit or 0.0003 V	
Variable Range:	+/-2 billion	
Variable Resolution:	$1 \cdot 10^{-4}$	
Array Size:	8000 elements, 30 arrays	
Program Size:	1000 lines x 80 characters	

Connectors for DMC-1600 Main Board

J1 DMC-1640 (A-D AXES) MAIN;

100-PIN HIGH DENSITY (AMP 2-178238-9):

1	Analog GND	51	nc
2	Ground	52	Ground
3	+5V	53	+5V
4	Error Output	54	Limit common
5	Reset	55	Home W
6	Encoder-Compare Output	56	Reverse limit W
7	Ground	57	Forward limit W
8	Ground	58	Home Z
9	Motor command W	59	Reverse limit Z
10	Sign W / Dir W	60	Forward limit Z
11	PWM W / Step W	61	Home Y
12	Motor command Z	62	Reverse limit Y
13	Sign Z / Dir Z	63	Forward limit Y
14	PWM Z / Step Z	64	Home X
15	Motor command Y	65	Reverse limit X
16	Sign Y/ Dir Y	66	Forward limit X
17	PWM Y/ Step Y	67	Ground
18	Motor command X	68	+5V
19	Sign X/ Dir X	69	Input common
20	PWM X / Step X	70	Latch X
21	Amp enable W	71	Latch Y
22	Amp enable Z	72	Latch Z
23	Amp enable Y	73	Latch W
24	Amp enable X	74	Input 5
25	A+ X	75	Input 6
26	A- X	76	Input 7
27	B+ X	77	Input 8
28	B- X	78	Abort
29	I+ X	79	Output 1
30	I- X	80	Output 2
31	A+ Y	81	Output 3
32	A- Y	82	Output 4
33	B+ Y	83	Output 5
34	B- Y	84	Output 6
35	I+ Y	85	Output 7
36	I- Y	86	Output 8
37	A+ Z	87	+5V
38	A- Z	88	Ground
39	B+ Z	89	Ground
40	B- Z	90	Ground
41	I+ Z	91	Analog In 1
42	I- Z	92	Analog In 2
43	A+ W	93	Analog In 3
44	A- W	94	Analog In 4
45	B+ W	95	Analog In 5
46	B- W	96	Analog In 6
47	I+ W	97	Analog In 7
48	I- W	98	Analog In 8
49	+12V	99	-12V
50	+12V	100	-12V

J5-DMC-1640 AUXILIARY ENCODERS

36-PIN HIGH DENSITY (AMP 178238-5):

1	+5V	19	NC
2	Ground	20	NC
3	A+ Aux X	21	NC
4	A- Aux X	22	NC
5	B+ Aux X	23	NC
6	B- Aux X	24	NC
7	A+ Aux Y	25	NC
8	A- Aux Y	26	NC
9	B+ Aux Y	27	NC
10	B- Aux Y	28	NC
11	A+ Aux Z	29	NC
12	A- Aux Z	30	NC
13	B+ Aux Z	31	NC
14	B- Aux Z	32	NC
15	A+ Aux W	33	NC
16	A- Aux W	34	NC
17	B+ Aux W	35	NC
28	B- Aux W	36	NC

Notes: *X,Y,Z,W are interchangeable designations for A,B,C,D axes.*

For A Description of the Connectors of the Extended I/O, see section below, "Extended I/O of the DMC-1600 Controller".

Pin-Out Description for DMC-1600

Outputs

Analog Motor Command	+/- 10 Volt range signal for driving amplifier. In servo mode, motor command output is updated at the controller sample rate. In the motor off mode, this output is held at the OF command level.
Amp Enable	Signal to disable and enable an amplifier. Amp Enable goes low on Abort and OE1.
PWM/STEP OUT	PWM/STEP OUT is used for directly driving power bridges for DC servo motors or for driving step motor amplifiers. For servo motors: If you are using a conventional amplifier that accepts a +/-10 Volt analog signal, this pin is not used and should be left open. The switching frequency is 16.7 kHz. The PWM output is available in two formats: Inverter and Sign Magnitude. In the Inverter mode, the PWM signal is .2% duty cycle for full negative voltage, 50% for 0 Voltage and 99.8% for full positive voltage. In the Sign Magnitude Mode (Jumper SM), the PWM signal is 0% for 0 Voltage, 99.6% for full voltage and the sign of the Motor Command is available at the sign output.
PWM/STEP OUT	For step motors: The STEP OUT pin produces a series of pulses for input to a step motor driver. The pulses may either be low or high. The pulse width is 50%. Upon Reset, the output will be low if the SM jumper is on. If the SM jumper is not on, the output will be tristate.
Sign/Direction	Used with PWM signal to give the sign of the motor command for servo amplifiers or direction for step motors.
Error	The signal goes low when the position error on any axis exceeds the value specified by the error limit command, ER.
Output 1-Output 8	These 8 TTL outputs are uncommitted and may be designated by the user to toggle relays and trigger external events. The output lines are toggled by Set Bit, SB, and Clear Bit, CB, instructions. The OP instruction is used to define the state of all the bits of the Output port.

Inputs	
Encoder, A+, B+	Position feedback from incremental encoder with two channels in quadrature, CHA and CHB. The encoder may be analog or TTL. Any resolution encoder may be used as long as the maximum frequency does not exceed 12,000,000 quadrature states/sec. The controller performs quadrature decoding of the encoder signals resulting in a resolution of quadrature counts (4 x encoder cycles). Note: Encoders that produce outputs in the format of pulses and direction may also be used by inputting the pulses into CHA and direction into Channel B and using the CE command to configure this mode.
Encoder Index, I+	Once-Per-Revolution encoder pulse. Used in Homing sequence or Find Index command to define home on an encoder index.
Encoder, A-, B-, I-	Differential inputs from encoder. May be input along with CHA, CHB for noise immunity of encoder signals. The CHA- and CHB- inputs are optional.
Auxiliary Encoder, Aux A+, Aux B+, Aux I+, Aux A-, Aux B-, Aux I-	Inputs for additional encoder. Used when an encoder on both the motor and the load is required. Not available on axes configured for step motors.
Abort	A low input stops commanded motion instantly without a controlled deceleration (OE0). Also aborts motion program.
Reset	A low input resets the state of the processor to its power-on condition. The previously saved state of the controller, along with parameter values, and saved sequences are restored.
Forward Limit Switch	When active, inhibits motion in forward direction. Also causes execution of limit switch subroutine, #LIMSWI. The polarity of the limit switch may be set with the CN command.
Reverse Limit Switch	When active, inhibits motion in reverse direction. Also causes execution of limit switch subroutine, #LIMSWI. The polarity of the limit switch may be set with the CN command.
Home Switch	Input for Homing (HM) and Find Edge (FE) instructions. Upon BG following HM or FE, the motor accelerates to slew speed. A transition on this input will cause the motor to decelerate to a stop. The polarity of the Home Switch may be set with the CN command.
Input 1 - Input 8 (Isolated) Input 17 - Input 80 (TTL)	Uncommitted inputs. May be defined by the user to trigger events. Inputs are checked with the Conditional Jump instruction and After Input instruction or Input Interrupt. Input 1 is latch X, Input 2 is latch Y, Input 3 is latch Z and Input 4 is latch W if the high speed position latch function is enabled.
Latch	High speed position latch to capture axis position within 20 nanoseconds on occurrence of latch signal. AL command arms latch. Input 1 is latch X, Input 2 is latch Y, Input 3 is latch Z and Input 4 is latch W. Input 9 is latch E, input 10 is latch F, input 11 is latch G, input 12 is latch H.

Extended I/O of the DMC-1600 Controller

The DMC-1600 controller offers 64 extended I/O points which can be configured as inputs or outputs in 8 bit increments through software. The I/O points are accessed through a single 80 pin High Density connector. Rev A&B DMC-16x0 controllers used a 100 pin HD connector.

Configuring the I/O of the DMC-1600

The 64 extended I/O points of the DMC-1600 series controller can be configured in blocks of 8. The extended I/O is denoted as bits 17-80 and blocks 2-9.

The command, CO, is used to configure the extended I/O as inputs or outputs, in blocks of 8 bits. The CO command has one field:

CO n

n is a decimal value which represents a binary number. Each bit of the binary number represents one block of extended I/O. When set to 1, the corresponding block is configured as an output.

The least significant bit represents block 2 and the most significant bit represents block 9. The decimal value can be calculated by the following formula. $n = n_2 + 2*n_3 + 4*n_4 + 8*n_5 +16* n_6 +32* n_7 +64* n_8 +128* n_9$ where n_x represents the block. If the n_x value is a one, then the block of 8 I/O points is to be configured as an output. If the n_x value is a zero, then the block of 8 I/O points will be configured as an input. For example, if block 4 and 5 are to be configured as outputs, CO 12 is issued.

8-Bit I/O Block	Block	Binary Representation	Decimal Value of Bit
17-24	2	2^0	1
25-32	3	2^1	2
33-40	4	2^2	4
41-48	5	2^3	8
49-56	6	2^4	16
57-64	7	2^5	32
65-72	8	2^6	64
73-80	9	2^7	128

The simplest method for determining n:

Step 1. Choose which 8-bit I/O blocks that should be configured as outputs.

Step 2. From the table, determine the decimal value for each I/O block to be set as an output.

Step 3. Add up all of the values determined in step 2. This is the value to be used for n.

For example, if blocks 1 and 2 are to be outputs, then n is 3 and the command, CO3, should be issued. Note: This calculation is identical to the formula: $n = n_2 + 2*n_3 + 4*n_4 + 8*n_5 +16* n_6 +32* n_7 +64* n_8 +128* n_9$ where n_x represents the block.

Saving the State of the Outputs in Non-Volatile Memory

The configuration of the extended I/O and the state of the outputs can be stored in the EEPROM with the BN command. If no value has been set, the default of CO 0 is used (all blocks are inputs).

Accessing extended I/O

When configured as an output, each I/O point may be defined with the SBn and CBn commands (where n=9 through 56). OBn can also be used with n=9 through 56.

The command, OP, may be used to set the state of output bits. The OP command has 2 parameters. The first parameter sets the values of the main output port of the controller. The second parameter sets the value of the extended I/O configured as outputs. The command syntax for the command is the following:

OP m,a,b,c,d,e

where m is the decimal representation of the bits 1-8 (values from 0 to 255) and a,b,c,d,e represent the extended I/O in consecutive groups of 16 bits. (values from 0 to 65535). Arguments which are given for I/O points which are configured as inputs will be ignored. The following table describes the arguments used to set the state of outputs.

Argument	Blocks	Bits	Description
m	0	1-8	General Outputs
a	2,3	17-32	Extended I/O
b	4,5	33-40	Extended I/O
c	6,7	41-48	Extended I/O
d	8,9	49-56	Extended I/O

When accessing I/O blocks configured as inputs, use the TIn command. The argument 'n' refers to the block to be read (n=1 to 9).

Individual bits can be queried using the @IN[n] command (where n=9 to 80). If the following command is issued;

MG @IN[17]

the controller will return the state of the least significant bit of block 2 (assuming block 2 is configured as an input).

Connector Description:

The extended I/O connector is a single 100 pin High Density Connector used on Revs A&B versions of the DMC-16x0.

J101

100-PIN HIGH DENSITY:

Pin	Signal	Block	Bit @IN[n], @OUT[n]	Bit No
1.	I/O	4	40	7
3.	I/O	4	39	6
5	I/O	4	38	5
7.	I/O	4	37	4
9.	I/O	4	36	3
11.	I/O	4	35	2
13.	I/O	4	34	1

15.	I/O	4	33	0
17.	I/O	3	32	7
19.	I/O	3	31	6
21.	I/O	3	30	5
23.	I/O	3	29	4
25.	I/O	3	28	3
27.	I/O	3	27	2
29.	I/O	3	26	1
31.	I/O	3	25	0
33.	I/O	2	24	7
35.	I/O	2	23	6
37.	I/O	2	22	5
39.	I/O	2	21	4
41.	I/O	2	20	3
43.	I/O	2	19	2
45.	I/O	2	18	1
47.	I/O	2	17	0
49.	+5V	-	-	-
2.	I/O	5	48	0
4.	I/O	5	47	1
6.	I/O	5	46	2
8.	I/O	5	45	3
10.	I/O	5	44	4
12.	I/O	5	43	5
14.	I/O	5	42	6
16.	I/O	5	41	7
18.	GND	-	-	-
20.	GND	-	-	-
22.	GND	-	-	-
24.	GND	-	-	-
26.	GND	-	-	-
28.	GND	-	-	-
30.	GND	-	-	-
32.	GND	-	-	-
34.	GND	-	-	-
36.	GND	-	-	-
38.	GND	-	-	-
40.	GND	-	-	-
42.	GND	-	-	-
44.	GND	-	-	-
46.	GND	-	-	-
48.	GND	-	-	-
50.	GND	-	-	-
51.	I/O	8	72	7
53.	I/O	8	71	6

55	I/O	8	70	5
57.	I/O	8	69	4
59.	I/O	8	68	3
61.	I/O	8	67	2
63.	I/O	8	66	1
65.	I/O	8	65	0
67.	I/O	7	64	7
69.	I/O	7	63	6
71.	I/O	7	62	5
73.	I/O	7	61	4
75.	I/O	7	60	3
77.	I/O	7	59	2
79.	I/O	7	58	1
81.	I/O	7	57	0
83.	I/O	6	56	7
85.	I/O	6	55	6
87.	I/O	6	54	5
89.	I/O	6	53	4
91.	I/O	6	52	3
93.	I/O	6	51	2
95.	I/O	6	50	1
97.	I/O	6	49	0
99.	+5V	-	-	-
52.	I/O	9	80	7
54.	I/O	9	79	6
56.	I/O	9	78	5
58.	I/O	9	77	4
60.	I/O	9	76	3
62.	I/O	9	75	2
64.	I/O	9	74	1
66.	I/O	9	73	0
68.	GND	-	-	-
70.	GND	-	-	-
72.	GND	-	-	-
74.	GND	-	-	-
76.	GND	-	-	-
78.	GND	-	-	-
80.	GND	-	-	-
82.	GND	-	-	-
84.	GND	-	-	-
86.	GND	-	-	-
88.	GND	-	-	-
90.	GND	-	-	-
92.	GND	-	-	-
94.	GND	-	-	-

96.	GND	-	-	-
98.	GND	-	-	-
100.	GND	-	-	-

Note for Interfacing to External I/O Racks

The extended I/O connector can be made compatible with external I/O mounting racks such as Grayhill 70GRCM32-HL and OPTO-22 G4PB24 by using the CB-50-80 and a 80 pin high density cable. By connecting the CB-50-80, the user will be provided with 2 50pin IDC connectors which are directly compatible with specific I/O mounting racks. When using the OPTO-22 G4PB24 I/O mounting rack, the user will only have access to 48 of the 64 I/O points available on the controller. Block 5 and Block 9 must be configured as inputs and will be grounded by the I/O rack.

Jumper Description for DMC-1600

JUMPER	LABEL	FUNCTION (IF JUMPERED)
JP20	SMX	For each axis, the SM jumper selects the SM
	SMY	magnitude mode for servo motors or selects
	SMZ	stepper motors. If you are using stepper
	SMW	motors, SM must always be jumpered. The Analog motor command is not valid with SM jumpered.
	SM E	
	SM F	
	SM G	
	SM H	
	OPT	Reserved
JP21	MRST	Master Reset enable. Returns controller to factory default settings and erases EEPROM. Requires power-on or RESET to be activated.

Accessories and Options

DMC-1610	1- axis motion controller
DMC-1620	2- axes motion controller
DMC-1630	3- axes motion controller
DMC-1640	4- axes motion controller
Cable-1600-1M	100-pin high density cable, 1 meter
Cable-1600-4M	100-pin high density cable, 4 meter
CB-50-100	50-pin to 100-pin converter board, includes two 50-pin ribbon cables
Starter Kit	Includes DMC-1600, ICM-1900 or AMP 19X0, cable, utilities, WSDK software, and manual
16-Bit ADC	Increased resolution for analog inputs
Sinusoidal Commutation Option	Sinusoidal Commutation for brushless motors
ICM-1900	Interconnect module
ICM-1900-Opto	Optoisolated digital outputs
AMP-1910	Interconnect module with 1-axis power amplifier
AMP-1920	Interconnect module with 2-axes power amplifier
AMP-1930	Interconnect module with 3-axes power amplifier
AMP-1940	Interconnect module with 4-axes power amplifier
DMC-1600 Utilities	Utilities for Plug & Play, firmware
WSDK-16	Servo Design Kit for Windows 3.X
WSDK-32	Servo Design Kit for Windows NT or Windows 95, 98, 2000, ME, XP
VBX Tool Kit	Visual Basic™ Tool Kit (includes VBXs and OCXs)
Setup 16	Set-up software for Windows 3.X
Setup 32	Set-up software for Windows NT or Windows 95
CAD-to-DMC	AutoCAD® DXF translator
HPGL	HPGL translator

PC/AT Interrupts and Their Vectors

(These occur on the first 8259)

IRQ	VECTOR	USAGE
0	8 or 08h	Timer chip (DON'T USE THIS!)
1	9 or 09h	Keyboard (DON'T USE THIS!)
2	10 or 0ah	Cascade from second 8259 (DON'T USE THIS!)
3	11 or 0bh	COM2:
4	12 or 0ch	COM1:
5	13 or 0dh	LPT2:
6	14 or 0eh	Floppy (DON'T USE THIS!)
7	15 or 0fh	LPT1:

(These occur on the second 8259)

IRQ	VECTOR	USAGE
8	104 or 70h	Real-time clock (DON'T USE THIS!)
9	105 or 71h	Redirect-cascade (DON'T USE THIS!)
10	106 or 72h	
11	107 or 73h	
12	108 or 74h	Mouse DSR
13	109 or 75h	Math Co-processor exception
14	110 or 76h	Fixed Disk (DON'T USE THIS!)
15	111 or 77h	

ICM-1900 Interconnect Module

The ICM-1900 interconnect module provides easy connections between the DMC-1600 series controllers and other system elements, such as amplifiers, encoders, and external switches. The ICM- 1900 accepts the 100-pin main cable and 25-pin auxiliary cable and breaks them into screw-type terminals. Each screw terminal is labeled for quick connection of system elements. An ICM-1900 is required for each set of 4 axes.

The ICM-1900 is contained in a metal enclosure. A version of the ICM-1900 is also available with servo amplifiers (see AMP-19X0).

Features

- Breaks out DMC-1600 cables into individual screw-type terminals
- Clearly identifies all terminals
- Provides jumper for connecting limit and input supplies to 5 V supply from PC
- Available with on-board servo drives (see AMP-19X0)
- Can be configured for AEN high or low

Terminal #	Label	I/O	Description
1	+AAX	I	X Auxiliary encoder A+

2	-AAX	I	X Auxiliary encoder A-
3	+ABX	I	X Auxiliary encoder B+
4	-ABX	I	X Auxiliary encoder B-
5	+AAY	I	Y Auxiliary encoder A+
6	-AAY	I	Y Auxiliary encoder A-
7	+ABY	I	Y Auxiliary encoder B+
8	-ABY	I	Y Auxiliary encoder B-
9	+AAZ	I	Z Auxiliary encoder A+
10	-AAZ	I	Z Auxiliary encoder A-
11	+ABZ	I	Z Auxiliary encoder B+
12	-ABZ	I	Z Auxiliary encoder B-
13	+AAW	I	W Auxiliary encoder A+
14	-AAW	I	W Auxiliary encoder A-
15	+ABW	I	W Auxiliary encoder B+
16	-ABW	I	W Auxiliary encoder B-
17	GND		Signal Ground
18	+VCC		+ 5 Volts
19	OUTCOM	O	Output Common (for use with the opto-isolated output option)
20	ERROR	O	Error signal
21	RESET	I	Reset
22	CMP	O	Circular Compare output
23	MOCMDW	O	W axis motor command to amp input (w / respect to ground)
24	SIGNW	O	W axis sign output for input to stepper motor amp
25	PWMW	O	W axis pulse output for input to stepper motor amp
26	MOCMDZ	O	Z axis motor command to amp input (w / respect to ground)
27	SIGNZ	O	Z axis sign output for input to stepper motor amp
28	PWMZ	O	Z axis pulse output for input to stepper motor amp
29	MOCMDY	O	Y axis motor command to amp input (w / respect to ground)
30	SIGNY	O	Y axis sign output for input to stepper motor amp
31	PWMY	O	Y axis pulse output for input to stepper motor amp
32	MOCMDX	O	X axis motor command to amp input (w / respect to ground)
33	SIGNX	O	X axis sign output for input to stepper motor amp
34	PWMX	O	X axis pulse output for input to stepper motor amp
35	GND	O	Signal Ground
36	+VCC	O	+ 5 Volts
37	AMPENW	O	W axis amplifier enable
38	AMPENZ	O	Z axis amplifier enable
39	AMPENY	O	Y axis amplifier enable
40	AMPENX	O	X axis amplifier enable
41	LSCOM	I	Limit Switch Common
42	HOMEW	I	W axis home input
43	RLSW	I	W axis reverse limit switch input
44	FLSW	I	W axis forward limit switch input
45	HOMEZ	I	Z axis home input
46	RLSZ	I	Z axis reverse limit switch input

47	FLSZ	I	Z axis forward limit switch input
48	HOMEY	I	Y axis home input
49	RLSY	I	Y axis reverse limit switch input
50	FLSY	I	Y axis forward limit switch input
51	HOMEX	I	X axis home input
52	RLSX	I	X axis reverse limit switch input
53	FLSX	I	X axis forward limit switch input
54	+VCC		+ 5 Volts
55	GND		Signal Ground
56	INCOM	I	Input common (Common for general inputs and Abort input)
57	XLATCH	I	Input 1 (Used for X axis latch input)
58	YLATCH	I	Input 2 (Used for Y axis latch input)
59	ZLATCH	I	Input 3 (Used for Z axis latch input)
60	WLATCH	I	Input 4 (Used for W axis latch input)
61	IN5	I	Input 5
62	IN6	I	Input 6
63	IN7	I	Input 7
64	IN8	I	Input 8
65	ABORT	I	Abort Input
66	OUT1	O	Output 1
67	OUT2	O	Output 2
68	OUT3	O	Output 3
69	OUT4	O	Output 4
70	OUT5	O	Output 5
71	OUT6	O	Output 6
72	OUT7	O	Output 7
73	OUT8	O	Output 8
74	GND		Signal Ground
75	AN1	I	Analog Input 1
76	AN2	I	Analog Input 2
77	AN3	I	Analog Input 3
78	AN4	I	Analog Input 4
79	AN5	I	Analog Input 5
80	AN6	I	Analog Input 6
81	AN7	I	Analog Input 7
82	AN8	I	Analog Input 8
83	+MAX	I	X Main encoder A+
84	-MAX	I	X Main encoder A-
85	+MBX	I	X Main encoder B+
86	-MBX	I	X Main encoder B-
87	+INX	I	X Main encoder Index +
88	-INX	I	X Main encoder Index -
89	GND		Signal Ground
90	+VCC		+ 5 Volts
91	+MAY	I	Y Main encoder A+

92	-MAY	I	Y Main encoder A-
93	+MBY	I	Y Main encoder B+
94	-MBY	I	Y Main encoder B-
95	+INY	I	Y Main encoder Index +
96	-INY	I	Y Main encoder Index -
97	+MAZ	I	Z Main encoder A+
98	-MAZ	I	Z Main encoder A-
99	+MBZ	I	Z Main encoder B+
100	-MBZ	I	Z Main encoder B-
101	+INZ	I	Z Main encoder Index +
102	-INZ	I	Z Main encoder Index -
103	GND		Signal Ground
104	+VCC		+ 5 Volts
105	+MAW	I	W Main encoder A+
106	-MAW	I	W Main encoder A-
107	+MBW	I	W Main encoder B+
108	-MBW	I	W Main encoder B-
109	+INW	I	W Main encoder Index +
110	-INW	I	W Main encoder Index -
111	+12V		+12 Volts
112	-12V		-12 Volts

Specifications

Dimensions: 13.5” x 2.675” x 6.88”

ICM-1900 Drawing

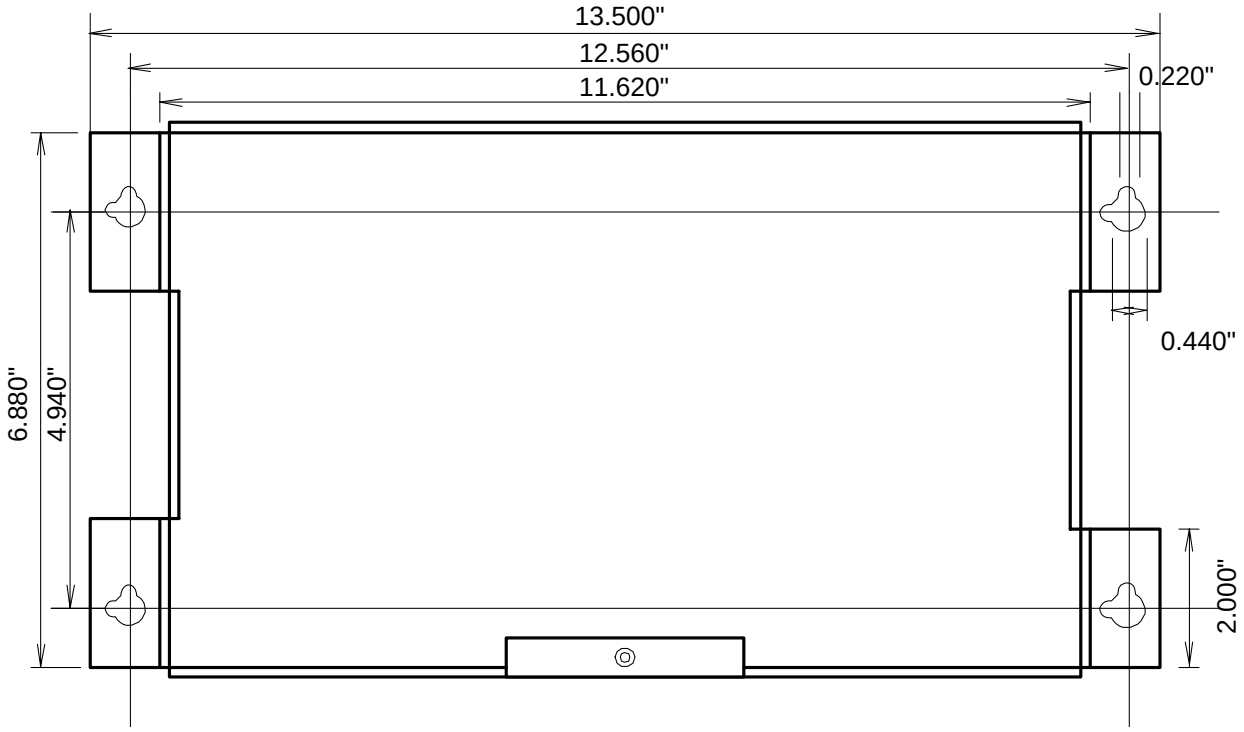


Figure A.1 – ICM-1900 Drawing

AMP-19X0 Mating Power Amplifiers

The AMP-19X0 series are mating, brush-type servo amplifiers for the DMC-1600. The AMP-1910 contains 1 amplifier: the AMP-1920, 2 amplifiers; the AMP-1930, 3 amplifiers; and the AMP-1940, 4 amplifiers. Each amplifier is rated for 7 amps continuous, 10 amps peak at up to 80 V. The gain of the AMP-19X0 is 1 amp/V. The AMP-19X0 requires an external DC supply. The AMP-19X0 connects directly to the DMC-1600 and screw terminals are provided for connection to motors, encoders, and external switches.

Features

- 7 amps continuous, 10 amps peak; 20 to 80V
- Available with 1, 2, 3, or 4 amplifiers
- Connects directly to DMC-1600 series controllers
- Screw-type terminals for easy connection to motors, encoders, and switches
- Steel mounting plate with 1/4" keyholes

Specifications

Minimum motor inductance: 1 mH
PWM frequency: 30 kHz
Ambient operating temperature: 0^o to 70^o C
Dimensions:
Weight:
Mounting: Keyholes -- 1/4"∅
Gain: 1 amp/V

Coordinated Motion - Mathematical Analysis

The terms of coordinated motion are best explained in terms of the vector motion. The vector velocity, Vs, which is also known as the feed rate, is the vector sum of the velocities along the X and Y axes, Vx and Vy.

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

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

The vector velocity is specified independently of the path to allow continuous motion. The path is specified as a collection of segments. For the purpose of specifying the path, define a special X-Y coordinate system whose origin is the starting point of the sequence. Each linear segment is specified by the X-Y coordinate of the final point expressed in units of resolution, and each circular arc is defined by the arc radius, the starting angle, and the angular width of the arc. The zero angle corresponds to the positive direction of the X-axis and the CCW direction of rotation is positive. Angles are expressed in degrees, and the resolution is 1/256th of a degree. For example, the path shown in Fig. A.2 is specified by the instructions:

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

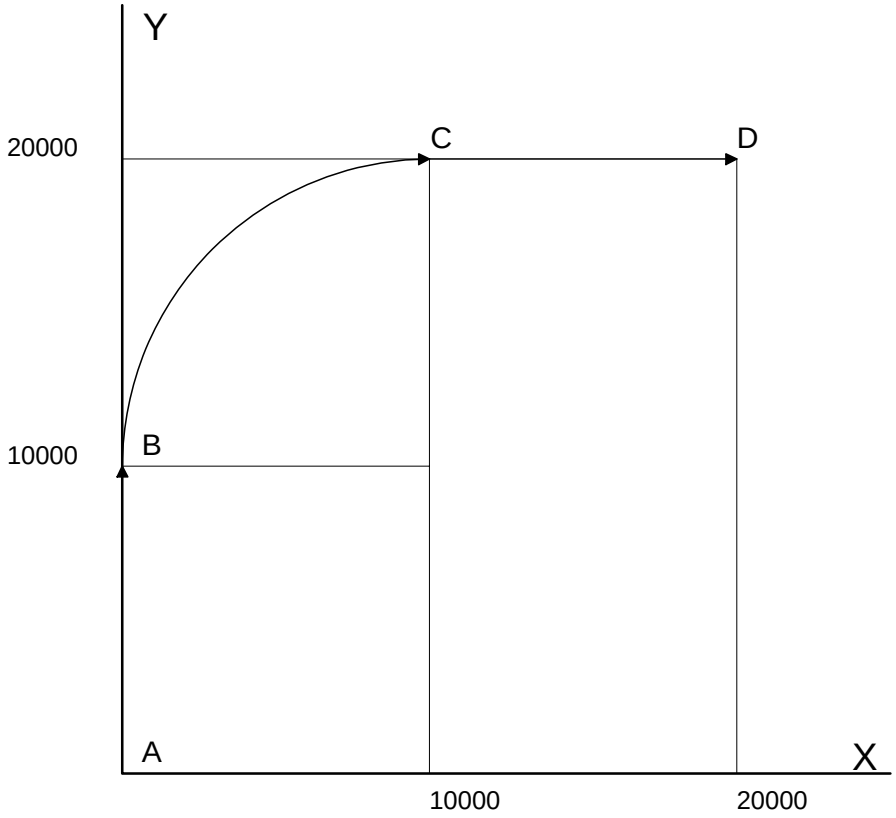


Figure A.2 - X-Y Motion Path

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

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

In general, the length of each linear segment is

$$L_k = \sqrt{Xk^2 + Yk^2}$$

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

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

The total travel distance is given by

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

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

For example, the velocity profile corresponding to the path of Fig. A.2 may be specified in terms of the vector speed and acceleration.

VS 100000

VA 2000000

The resulting vector velocity is shown in Fig. A.3.

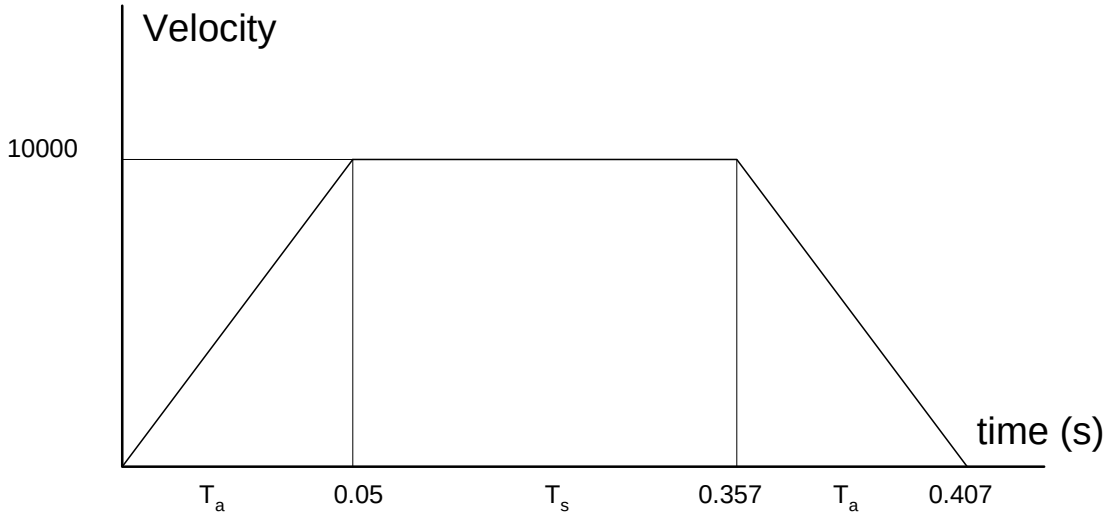


Figure A.3 - Vector Velocity Profile

The acceleration time, T_a, is given by

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

The slew time, Ts, is given by

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

The total motion time, Tt, is given by

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

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

For example, the velocities along the X and Y axes for the path shown in Fig. A.2 are given in Fig. A.4.

Fig. A.4a shows the vector velocity. It also indicates the position point along the path starting at A and ending at D. Between the points A and B, the motion is along the Y axis. Therefore,

Vy = Vs and Vx = 0

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

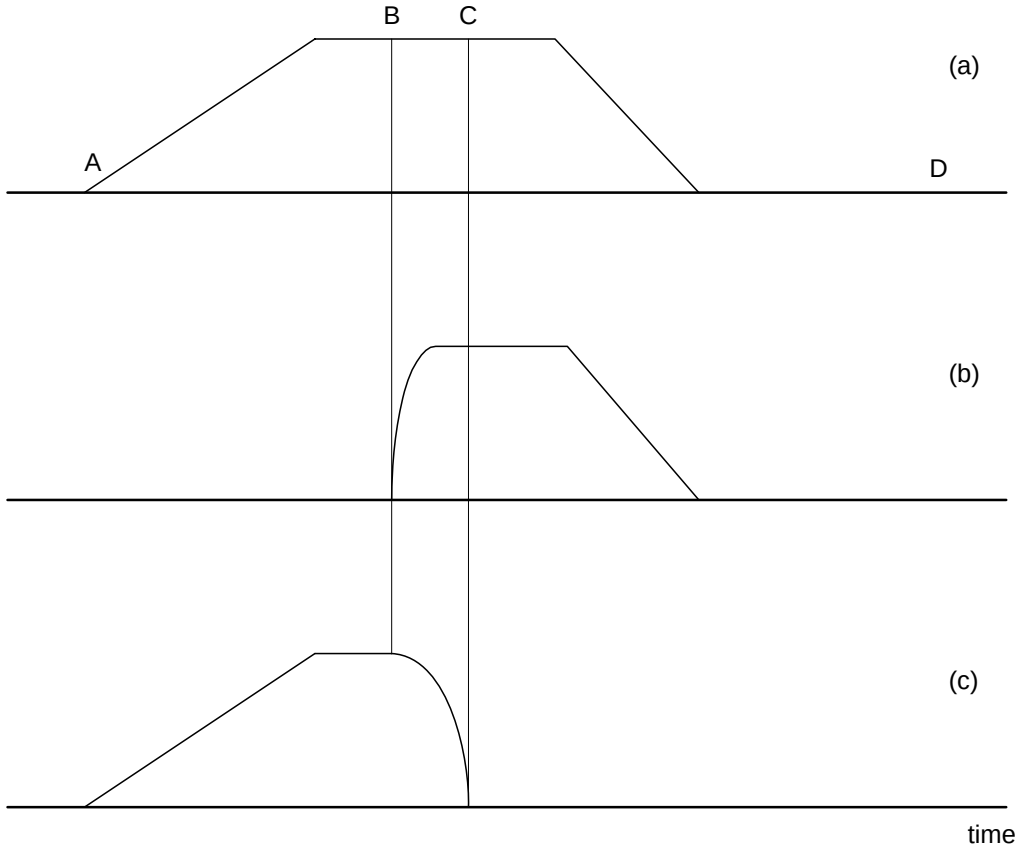


Figure A.4 - Vector and Axes Velocities

DMC-1600/DMC-1000 Comparison

BENEFIT	DMC-1600	DMC-1000
Higher Speed communication Frees host	Two communication channels-MAIN & Secondary FIFO	Only one channel- FIFO
Easy to install – self-configuring	Plug and Play	No Plug and Play
Programs don't have to be downloaded from PC but can be stored on controller	Non-Volatile Program Storage	No storage for programs
Can capture and save array data	Variable storage	No storage for variables
Parameters can be stored	Array storage	No storage for arrays
Firmware can be upgraded in field without removing controller from PC	Flash memory for firmware	EPROM for firmware which must be installed on controller
Faster servo operation – good for very high resolution sensors	12 MHz encoder speed for servos	8 MHz
Faster stepper operation	3 MHz stepper rate	2 MHz
Higher servo bandwidth	62 μsec/axis sample time	125 μsec/axis
Expanded memory lets you store more programs	1000 lines X 80 character program memory	500 line X 40 character
Expanded variables	254 symbolic variables	126 variables
Expanded arrays for more storage—great for data capture	8000 array elements in 30 arrays	1600 elements in 14 arrays
Higher resolution for analog inputs	8 analog inputs with 16-bit ADC option	7 inputs with 12-bit ADC only
Better for EMI reduction	100-pin high density connector	60-pin IDC, 26-pin IDC, 20-pin IDC (x2)
For precise registration applications	Output Position Compare	Available only as a special
More flexible gearing	Multiple masters allowed in gearing mode	One master for gearing
Flexible- Binary mode is higher speed	Binary and ASCII communication modes	ASCII only
Multiple threads	Up to 8 threads	Up to 4 threads

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 200,000 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 15 years, Galil has actively shared its market insights in a no-nonsense way for a world of engineers on the move. In fact, over 10,000 engineers have attended Galil seminars. The tradition continues with three different seminars, each designed for your particular skill set—from beginner to the most advanced.

MOTION CONTROL MADE EASY

WHO SHOULD ATTEND

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

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

ADVANCED MOTION CONTROL

WHO SHOULD ATTEND

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

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

PRODUCT WORKSHOP

WHO SHOULD ATTEND

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

TIME: Two days (8:30 am-5:00 pm)

Contacting Us

Galil Motion Control

270 Technology Way

Rocklin, California 95765

Phone: 916-626-0101

Fax: 916-626-0102

Internet address: support@galilmc.com

URL: www.galilmc.com

FTP: galilmc.com

WARRANTY

All products manufactured by Galil Motion Control are warranted against defects in materials and workmanship. The warranty period for controller boards is 1 year. The warranty period for all other products is 180 days.

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.

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.

Index

D

E

F

G

H

I

J

K

L

M

O

P

Q

R

S

T

U

V

W

Z