USER MANUAL

DMC-2x00

Manual Rev. 1.6

By Galil Motion Control, Inc.

Galil Motion Control, Inc.
3750 Atherton Road
Rocklin, California 95765
Phone: (916) 626-0101
Fax: (916) 626-0102
Internet Address: support@galilmc.com
URL: www.galilmc.com

Rev 05/02

Using This Manual

This user manual provides information for proper operation of the DMC-2x00 controller. A separate supplemental manual, the Command Reference, contains a description of the commands available for use with this controller.

Your DMC-2x00 motion controller has been designed to work with both servo and stepper type motors. Installation and system setup will vary depending upon whether the controller will be used with stepper motors or servo motors. To make finding the appropriate instructions faster and easier, icons will be next to any information that applies exclusively to one type of system. Otherwise, assume that the instructions apply to all types of systems. The icon legend is shown below.

Attention: Pertains to servo motor use.

Attention: Pertains to stepper motor use.

Attention: Pertains to controllers with more than 4 axes.

Please note that many examples are written for the DMC-2x40 four-axes controller or the DMC-2x80 eight axes controller. Users of the DMC-2x30 3-axis controller, DMC-2x20 2-axes controller or DMC-2x10 1-axis controller should note that the DMC-2x30 uses the axes denoted as XYZ, the DMC-2x20 uses the axes denoted as XY, and the DMC-2x10 uses the X-axis only.

Examples for the DMC-2x80 denote the axes as A,B,C,D,E,F,G,H. Users of the DMC-2x50 5-axes controller. DMC-2x60 6-axes controller or DMC-2x70, 7-axes controller should note that the DMC-2x50 denotes the axes as A,B,C,D,E, the DMC-2x60 denotes the axes as A,B,C,D,E,F and the DMC-2x70 denotes the axes as A,B,C,D,E,F,G. The axes A,B,C,D may be used interchangeably with A,B,C,D.

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

Contents

Chapter 1 Overview 1

Introduction....1
Overview of Motor Types....1
- Standard Servo Motor with +/- 10 Volt Command Signal....2
- Brushless Servo Motor with Sinusoidal Commutation....2
- Stepper Motor with Step and Direction Signals....2

Overview of Amplifiers....2
- Amplifiers in Current Mode....2
- Amplifiers in Velocity Mode....3
- Stepper Motor Amplifiers....3

DMC-2x00 Functional Elements....3
- Microcomputer Section....3
- Motor Interface....3
- Communication....4
- General I/O....4
- System Elements....4
- Motor....4
- Amplifier (Driver)....4
- Encoder....5
- Watch Dog Timer....5

Chapter 2 Getting Started 7

The DMC-2x00 Main Board....7
The DMC-2000 Daughter Board....8
The DMC-2200 Daughter Board....9
Elements You Need....10
Installing the DMC-2x00....12
- Step 1. Determine Overall Motor Configuration....12
- Step 2. Install Jumpers on the DMC-2x00....13
- Step 3a. Configure DIP switches on the DMC-2000....14
- Step 3b. Configure DIP switches on the DMC-2100....15
- Step 3c. Configure DIP switches on the DMC-2200....15
- Step 4. Install the Communications Software....16
- Step 5. Connect AC Power to the Controller....16
- Step 6. Establish Communications with Galil Software....17
- Step 7. Determine the Axes to be Used for Sinusoidal Commutation....20
- Step 8. Make Connections to Amplifier and Encoder.....21
- Step 9a. Connect Standard Servo Motors....23
- Step 9b. Connect Sinusoidal Commutation Motors....26
- Step 9c. Connect Step Motors....29

Step 10. Tune the Servo System....29
Design Examples....30
System Set-up....30
Profiled Move....31
Multiple Axes....31
Independent Moves....31
Position Interrogation....31
Absolute Position....32
Velocity Control....32
Operation Under Torque Limit....33
Interrogation....33
Operation in the Buffer Mode....33
Using the On-Board Editor....33
Motion Programs with Loops....34
Motion Programs with Trippoints....34
Control Variables....35
Linear Interpolation....35
Circular Interpolation....36

Chapter 3 Connecting Hardware 37

Overview....37
Using Optoisolated Inputs....37
Limit Switch Input....37
Home Switch Input....38
Abort Input....38
Reset Input....39
Uncommitted Digital Inputs....39
Wiring the Opto-Isolated Inputs....39
The Opto-Isolation Common Point....39
Using an Isolated Power Supply....40
Bypassing the Opto-Isolation:....41
Analog Inputs....41
Amplifier Interface....41
TTL Inputs....42
The Auxiliary Encoder Inputs:....42
TTL Outputs....43
General Use Outputs....43
Output Compare....43
Error Output....43
Extended I/O of the DMC-2x00 Controller....44

Chapter 4 Communication 45

Introduction....45
RS232 Ports....45
RS232 - Main Port {P1} DATATERM....45
RS232 - Auxiliary Port {P2} DATASET....45
*RS422 - Main Port {P1}....46
*RS422 - Auxiliary Port {P2}....46
RS-232 Configuration....46
Ethernet Configuration (DMC-2100/2200 only)....48
Communication Protocols....48
Addressing....49
Communicating with Multiple Devices....50
Multicasting....52

Using Third Party Software 52
Data Record 52
Data Record Map 52
Explanation of Status Information and Axis Switch Information 55
Notes Regarding Velocity and Torque Information 56
QZ Command 56
Controller Response to Commands 57
Unsolicited Messages Generated by Controller 57
Galil Software Tools and Libraries 58

Chapter 5 Command Basics 59

Introduction 59
Command Syntax - ASCII 59
Coordinated Motion with more than 1 axis 60
Command Syntax - Binary 61
Binary Command Format 61
Binary Command Table 62
Controller Response to DATA 63
Interrogating the Controller 64
Interrogation Commands 64
Summary of Interrogation Commands 64
Interrogating Current Commanded Values. 64
Operands 64
Command Summary 65

Chapter 6 Programming Motion 67

Overview 67
Independent Axis Positioning 68
Command Summary - Independent Axis 69
Operand Summary - Independent Axis 69
Examples 70
Independent Jogging 71
Command Summary - Jogging 71
Operand Summary - Independent Axis 72
Examples 72
Linear Interpolation Mode 73
Specifying the Coordinate Plane 73
Specifying Linear Segments 73
Additional Commands 74
Command Summary - Linear Interpolation 75
Operand Summary - Linear Interpolation 75
Example 75
Vector Mode: Linear and Circular Interpolation Motion 78
Specifying the Coordinate Plane 78
Specifying Vector Segments 79
Additional commands 79
Command Summary - Coordinated Motion Sequence 80
Operand Summary - Coordinated Motion Sequence 81
Example 81
Electronic Gearing 83
Command Summary - Electronic Gearing 84
Electronic Cam 85
Command Summary - Electronic CAM 88
Operand Summary - Electronic CAM 89

Example 89
Contour Mode....90
Specifying Contour Segments 90
Additional Commands....91
Command Summary - Contour Mode....92
General Velocity Profiles 92
Example 92
Virtual Axis 95
Ecam master example 95
Sinusoidal Motion Example 95
Stepper Motor Operation....96
Specifying Stepper Motor Operation....96
Stepper Motor Smoothing....96
Monitoring Generated Pulses vs Commanded Pulses 96
Motion Complete Trippoint 97
Using an Encoder with Stepper Motors 97
Command Summary - Stepper Motor Operation 97
Operand Summary - Stepper Motor Operation....98
Dual Loop (Auxiliary Encoder) 98
Additional Commands for the Auxiliary Encoder 98
Backlash Compensation....99
Example 99
Motion Smoothing....100
Using the IT and VT Commands:....100
Example 101
Using the KS Command (Step Motor Smoothing): 102
Homing 103
Example 103
Command Summary - Homing Operation....105
Operand Summary - Homing Operation....105
High Speed Position Capture (The Latch Function)....105
Example 106

Chapter 7 Application Programming 107

Overview....107
Using the DOS Editor to Enter Programs (DMC-2000 only) 107
Edit Mode Commands 108
Example 108
Program Format 109
Using Labels in Programs 109
Special Labels....109
Commenting Programs 110
Executing Programs - Multitasking....111
Debugging Programs 112
Trace Commands (DMC-2100/2200 only) 112
Error Code Command 113
Stop Code Command 113
RAM Memory Interrogation Commands 113
Operands....113
Example 113
Program Flow Commands....114
Event Triggers & Trippoints....114
Conditional Jumps....118
If, Else, and Endif 120
Subroutines 122

Stack Manipulation....122
Auto-Start Routine....123
Automatic Subroutines for Monitoring Conditions....123
Mathematical and Functional Expressions....128
Mathematical Operators....128
Bit-Wise Operators....128
Functions....130
Variables....130
Programmable Variables....131
Operands....132
Special Operands (Keywords)....132
Arrays....133
Defining Arrays....133
Assignment of Array Entries....133
Uploading and Downloading Arrays to On Board Memory....134
Automatic Data Capture into Arrays....134
Deallocating Array Space....136
Input of Data (Numeric and String)....136
Input of Data....136
Operator Data Entry Mode....137
Using Communication Interrupt (For DMC-2000 only)....138
Output of Data (Numeric and String)....139
Sending Messages....140
Displaying Variables and Arrays....141
Interrogation Commands....141
Formatting Variables and Array Elements....143
Converting to User Units....144
Hardware I/O....144
Digital Outputs....144
Digital Inputs....145
The Auxiliary Encoder Inputs....146
Input Interrupt Function....146
Analog Inputs....147
Extended I/O of the DMC-2x00 Controller....148
Configuring the I/O of the DMC-2x00....148
Saving the State of the Outputs in Non-Volatile Memory....149
Accessing Extended I/O....149
Interfacing to Grayhill or OPTO-22 G4PB24....150
Example Applications....150
Wire Cutter....150
A-B Table Controller....151
Speed Control by Joystick....153
Position Control by Joystick....154
Backlash Compensation by Sampled Dual-Loop....154
Introduction....157
Hardware Protection....157
Output Protection Lines....157
Input Protection Lines....157
Software Protection....158
Programmable Position Limits....158
Off-On-Error....159
Automatic Error Routine....159
Limit Switch Routine....160

Chapter 9 Troubleshooting 161

Overview....161
Installation....161
Communication....162
Stability....162
Operation....162

Chapter 10 Theory of Operation 163

Overview....163
Operation of Closed-Loop Systems....165
System Modelling....166
Motor-Amplifier....167
Encoder....169
DAC....170
Digital Filter....170
ZOH....171
System Analysis....172
System Design and Compensation....174
The Analytical Method....174

Appendices 177

Electrical Specifications....177
Servo Control....177
Stepper Control....177
Input / Output....177
Power....178
Performance Specifications....178
Minimum Servo Loop Update Time:....178
Fast Update Rate Mode....179
Connectors for DMC-2x00 Main Board....180
DMC-2x00 Axes A-D High Density Connector....180
DMC-2x00 Axes E-H High Density Connector....181
DMC-2x00 Auxiliary Encoder 36 Pin High Density Connector....182
DMC-2x00 Extended I/O 80 Pin High Density Connector....182
RS-232-Main Port....184
RS-232-Auxiliary Port....184
USB - In USB - Out....184
Ethernet....185
Cable Connections for DMC-2x00....185
Standard RS-232 Specifications....185
DMC-2x00 Serial Cable Specifications....186
Pin-Out Description for DMC-2x00....188
Jumper Description for DMC-2x00....190
Dimensions for DMC-2x00....191
Accessories and Options....192
ICM-2900 Interconnect Module....193
ICM-2900 Drawing:....196
ICM-2908 Interconnect Module....197
ICM-2908 Drawing:....198
PCB Layout of the ICM-2900:....199
ICM-1900 Interconnect Module....200
Features....200
ICM-1900 Drawing:....203
AMP-19x0 Mating Power Amplifiers....203
Features....203

Specifications 204
Opto-Isolated Outputs for ICM-2900 / ICM-1900 / AMP-19x0 204
Standard Opto-Isolation and High Current Opto-isolation: 204
Configuring the Amplifier Enable for ICM-2900 / ICM-1900 205
-LAEN Option: 205
-Changing the Amplifier Enable Voltage Level: 205
IOM-1964 Opto-Isolation Module for Extended I/O 206
Description: 206
Overview 206
Configuring Hardware Banks 207
Digital Inputs 208
High Power Digital Outputs 209
Standard Digital Outputs 210
Electrical Specifications 211
Relevant DMC Commands 212
Screw Terminal Listing 212
CB-50-100 Adapter Board 215
Connectors: 215
CB-50-100 Drawing: 218
CB-50-80 Adapter Board 219
Connectors: 220
CB-50-80 Drawing: 222
TERM-1500 Operator Terminal 224
Features 225
Description 225
Specifications - Hand-Held 225
Specifications - Panel Mount 226
Keypad Maps - Hand-Held 226
Keypad Map - Panel Mount – 6 columns x 5 rows 227
Configuration 228
Function Keys 229
Input/Output of Data – DMC-2x00 Commands 229
Ordering Information 230
Coordinated Motion - Mathematical Analysis 231
Example - Communicating with OPTO-22 SNAP-B3000-ENET 234
DMC-2x00/DMC-1500 Comparison 237
List of Other Publications 238
Training Seminars 238
Contacting Us 239
WARRANTY 240

Index **241**

Chapter 1 Overview

Introduction

The DMC-2x00 Series are Galil's highest performance stand-alone controller. The controller series offers many enhanced features including high speed communications, non-volatile program memory, faster encoder speeds, and improved cabling for EMI reduction.

Each DMC-2x00 provides two communication channels: high speed RS-232 (2 channels up to 115K Baud) and Universal Serial Bus (12Mb/s) for the DMC-2000 or 10BaseT Ethernet for the DMC-2100 and 100BaseT Ethernet for the DMC-2200.

A 4Meg Flash EEPROM provides non-volatile memory for storing application programs, parameters, arrays and firmware. New firmware revisions are easily upgraded in the field.

The DMC-2x00 is available with up to eight axes in a single stand alone unit. The DMC-2x10, 2x20, 2x30, 2x40 are one thru four axes controllers and the DMC-2x50, 2x60, 2x70, 2x80 are five thru eight axes controllers.

Designed to solve complex motion problems, the DMC-2x00 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-2x00 provides continuous vector feed of an infinite number of linear and arc segments. The controller also features electronic gearing with multiple master axes as well as gantry mode operation.

For synchronization with outside events, the DMC-2x00 provides uncommitted I/O, including 8 opto-isolated digital inputs (16 inputs for DMC-2x50 thru DMC-2x80), 8 digital outputs (16 outputs for DMC-2x50 thru DMC-2x80), and 8 analog inputs for interface to joysticks, sensors, and pressure transducers. The DMC-2x00 also has an additional 64 I/O. Further I/O is available if the auxiliary encoders are not being used (2 inputs / each axis). Dedicated optoisolated inputs are provided for forward and reverse limits, abort, home, and definable input interrupts.

Commands can be sent in either Binary or ASCII. Additional software is available for automatic-tuning, trajectory viewing on a PC screen, CAD translation, and program development using many environments such as Visual Basic, C, C++ etc. Drivers for DOS, Linux, Windows 3.1, 95, 98, 2000, ME and NT are available.

Overview of Motor Types

The DMC-2x00 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-2x00 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-2x00 can provide sinusoidal commutation for brushless motors (BLM). In this configuration, the controller generates two sinusoidal signals for connection with amplifiers specifically designed for this purpose.

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

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

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

The DMC-2x00 can control BLMs equipped with Hall sensors as well as without Hall sensors. If hall hall sensors are available, once the controller has been setup, the brushless motor parameters may be saved in non-volatile memory. In this case, the controller will automatically estimates the commutation phase upon reset. This allows the motor to function immediately upon power up. The hall effect sensors also provides a method for setting the precise commutation phase. Chapter 2 describes the proper connection and procedure for using sinusoidal commutation of brushless motors.

Stepper Motor with Step and Direction Signals

The DMC-2x00 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.

Overview of Amplifiers

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

Amplifiers in Current Mode

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

Amplifiers in Velocity Mode

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

Stepper Motor Amplifiers

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

DMC-2x00 Functional Elements

The DMC-2x00 circuitry can be divided into the following functional groups as shown in Figure 1.1 and discussed below.

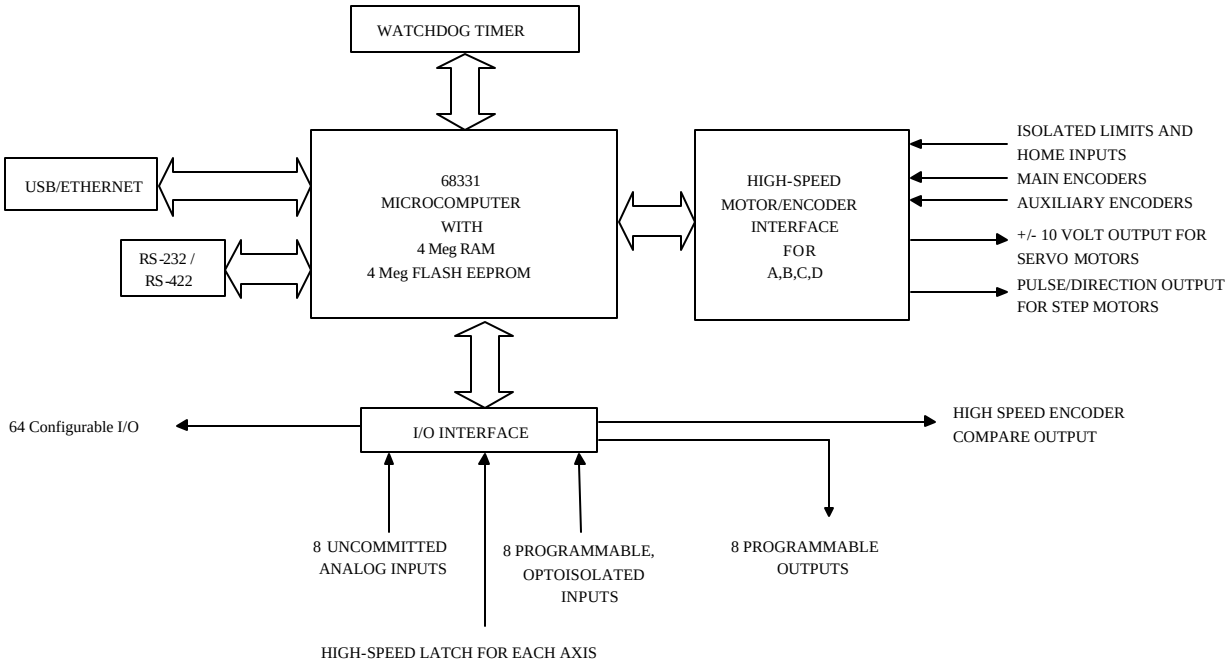


Figure 1.1 - DMC-2x00 Functional Elements

Microcomputer Section

The main processing unit of the DMC-2x00 is a specialized 32-Bit Motorola 68331 Series Microcomputer with 4 Meg RAM and 4 Meg 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-2x00 firmware.

Motor Interface

Galil's GL-1800 custom, sub-micron gate array performs quadrature decoding of each encoder at up to 12 MHz. For standard servo operation, the controller generates a +/-10 Volt analog signal (16 Bit DAC). For sinusoidal commutation operation, the controller uses 2 DACs to generate 2 +/-10Volt analog signals. For stepper motor operation, the controller generates a step and direction signal.

Communication

The communication interface with the DMC-2x00 consists of high speed RS-232 and USB or high speed RS-232 and Ethernet. The USB channel accepts based rates up to 12Mb/sec and the two RS-232 channels can generate up to 115K.

General I/O

The DMC-2x00 provides interface circuitry for 8 bi-directional, optoisolated inputs, 8 TTL outputs and 8 analog inputs with 12-Bit ADC (16-Bit optional). The DMC-2x00 also has an additional 64 I/O and unused auxiliary encoder inputs may also be used as additional inputs (2 inputs / each axis). The general inputs can also be used as high speed latches for each axes. A high speed encoder compare output is also provided.

2x80

The DMC-2x50 through DMC-2x80 controller provides an additional 8 optoisolated inputs and 8 TTL outputs.

System Elements

As shown in Fig. 1.2, the DMC-2x00 is part of a motion control system which includes amplifiers, motors and encoders. These elements are described below.

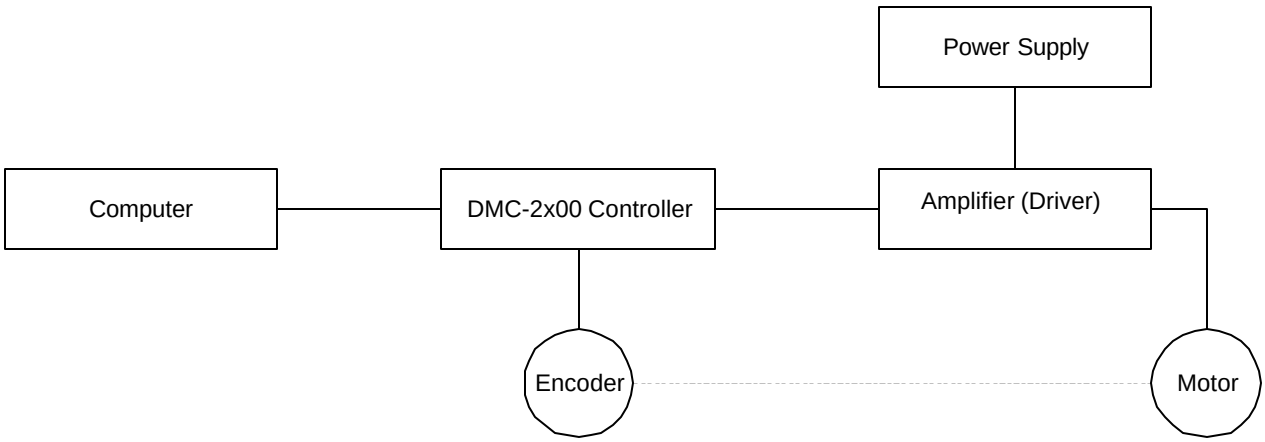


Figure 1.2 - Elements of Servo systems

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. (Ga lil'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-2x00 accepts feedback from either a rotary or linear encoder. Typical encoders provide two channels in quadrature, known as CHA and CHB. This type of encoder is known as a quadrature encoder. Quadrature encoders may be either single-ended (CHA and CHB) or differential (CHA,CHA- and CHB,CHB-). The DMC-2x00 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-2x00 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-2x00. Single-ended 12 Volt signals require a bias voltage input to the complementary inputs).

The DMC-2x00 can accept analog feedback instead of an encoder for any axis.

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 and talk to one of our applications engineers about your particular system requirements.

Watch Dog Timer

The DMC-2x00 provides an internal watch dog timer which checks for proper microprocessor operation. The timer toggles the Amplifier Enable Output (AMPEN) which can be used to switch the amplifiers off in the event of a serious DMC-2x00 failure. The AMPEN output is normally high. During power-up and if the microprocessor ceases to function properly, the AMPEN output will go low. The error light will also turn on at this stage. A reset is required to restore the DMC-2x00 to normal operation. Consult the factory for a Return Materials Authorization (RMA) Number if your DMC-2x00 is damaged.

THIS PAGE LEFT BLANK INTENTIONALLY

Chapter 2 Getting Started

The DMC-2x00 Main Board

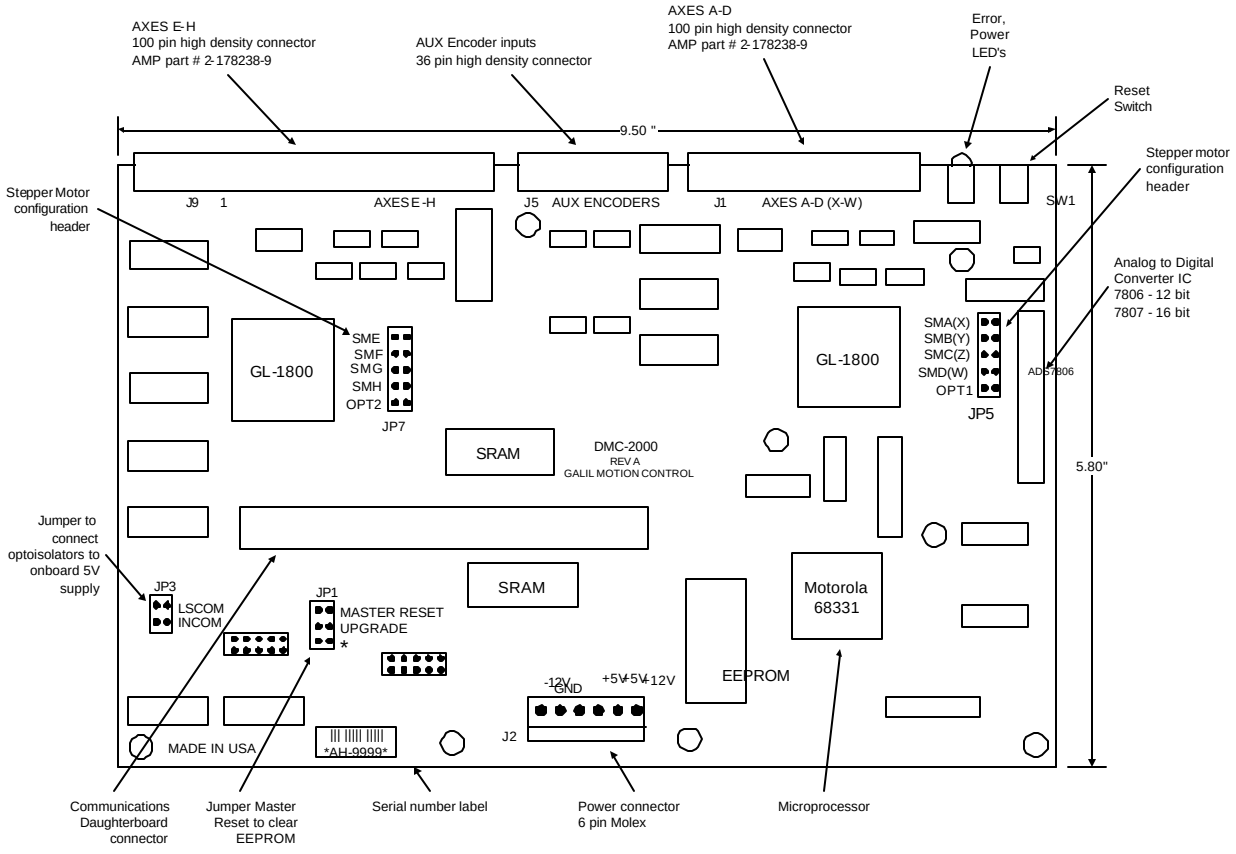


Figure 2-1 - Outline of the main board of the DMC-2x00

The DMC-2000 Daughter Board

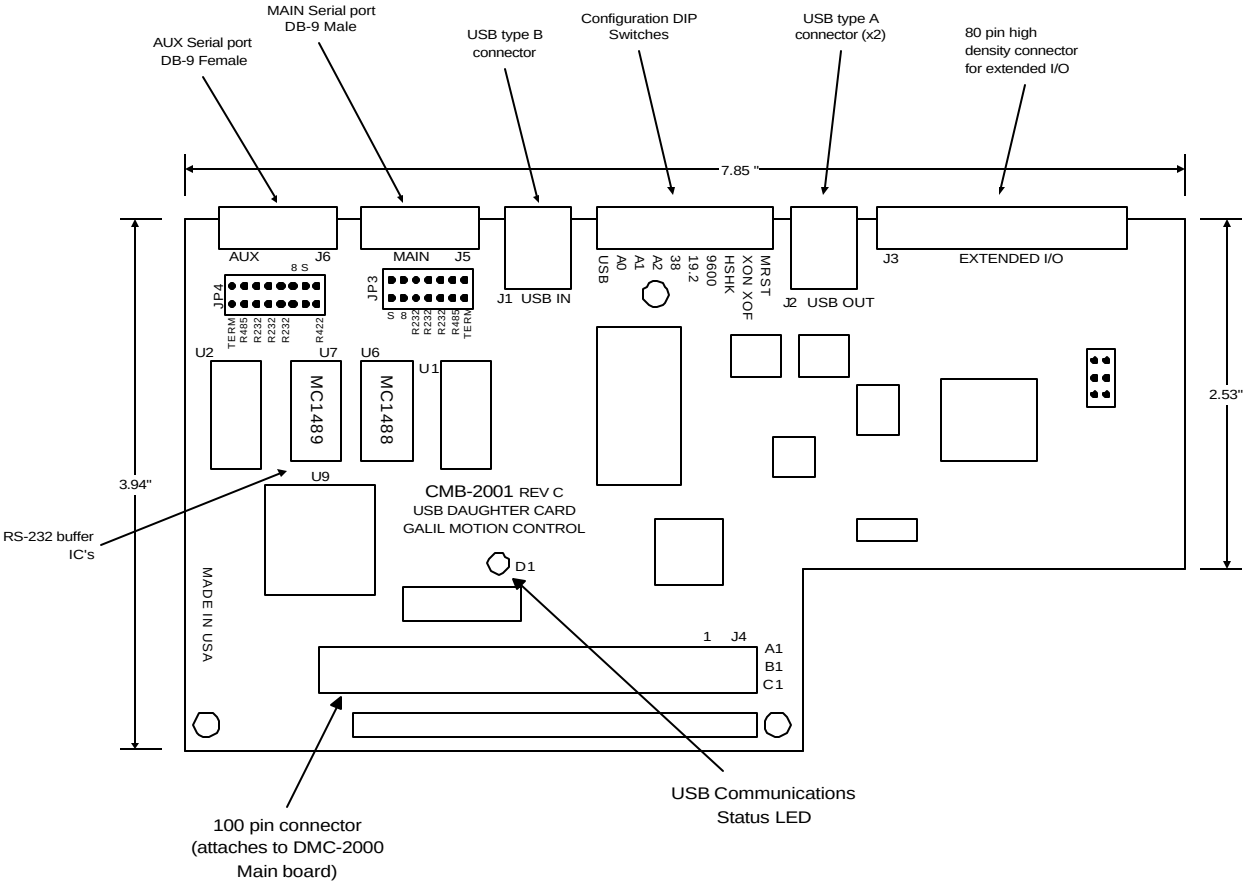


Figure 2-2 - Outline of the DMC-2000 Daughter Board

The DMC-2200 Daughter Board

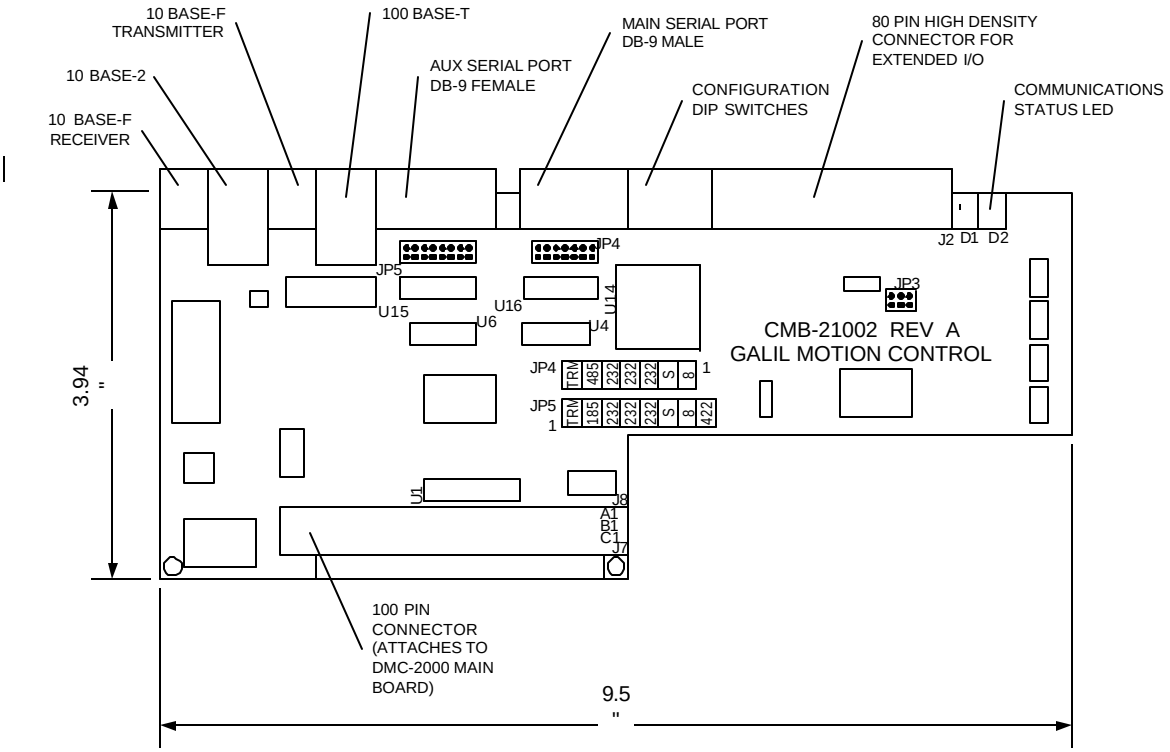


Figure 2-3B - Outline of the DMC-2200 Daughter Board

Elements You Need

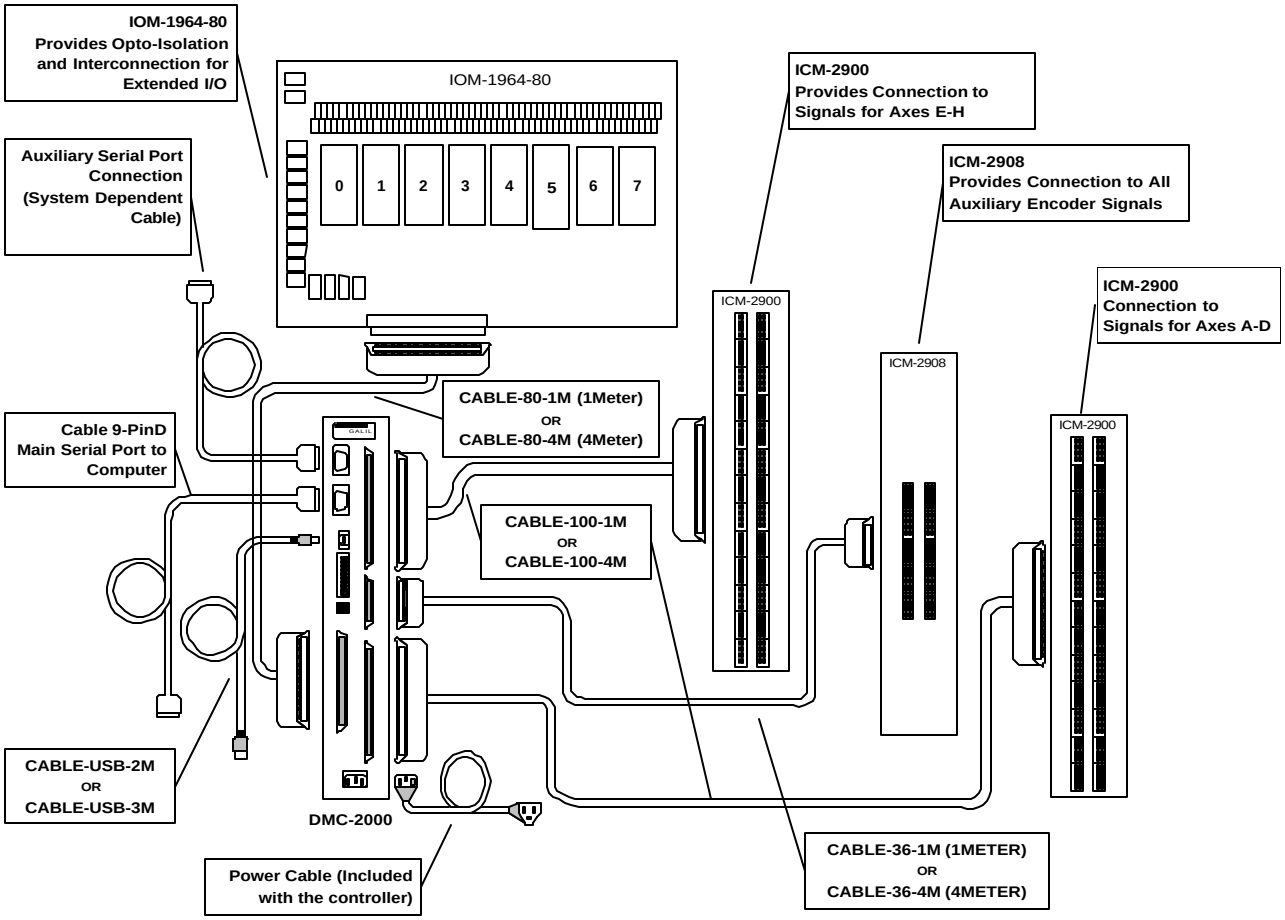


Figure 2-4 Recommended System Elements of DMC-2000

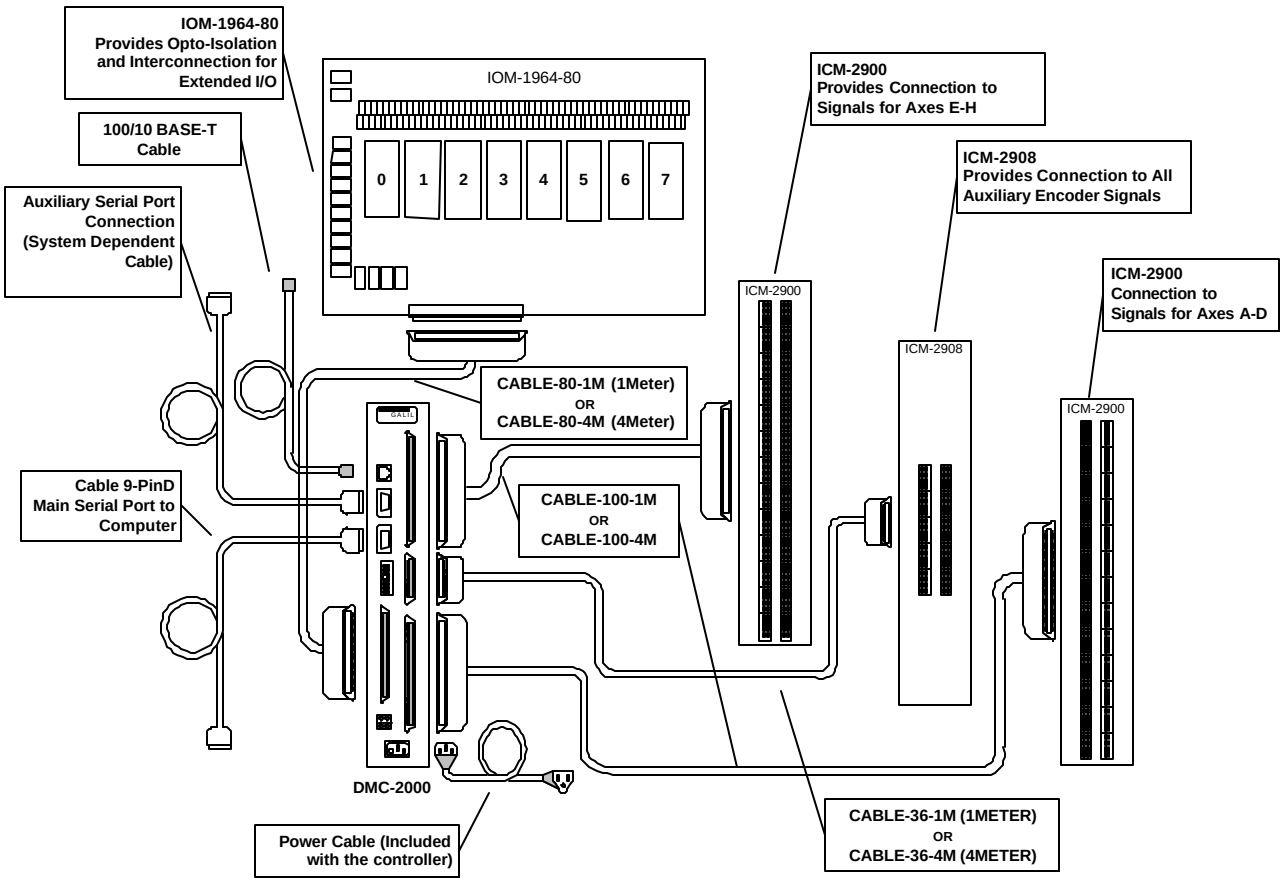


Figure 2-5 Recommended System Elements of DMC-2100/DMC-2200

For a complete system, Galil recommends the following elements:

1a. DMC-2x10, 2x20, 2x30, or DMC-2x40 Motion Controller

or

1b. DMC-2x50, 2x60, 2x70 or DMC-2x80

2a. (1) ICM-2900 and (1) CABLE-100 for controllers DMC-2x10 through DMC-2x40

or

2b. (2) ICM-2900's and (2) CABLE-100's for controllers DMC-2x50 through DMC-2x80.

or

2c. An interconnect board provided by the user.

3. (1) IOM-1964 and (1) CABLE-80 for access to the extended I/O. Only required if extended I/O will be used. The CABLE-80 can also be converted for use with OPTO-22 or Grayhill I/O modules - consult Galil.

4. (1) ICM-2908 and (1) CABLE-36 for access to auxiliary encoders. Only required if auxiliary encoders are needed.

5. Motor Amplifiers.
6. Power Supply for Amplifiers.
7. Brush or Brushless Servo motors with Optical Encoders or stepper motors.
8. PC (Personal Computer - RS232 or USB for DMC-2000 or Ethernet for DMC-2100)

9a. WSDK-16 or WSDK-32 (recommend for first time users.)

or

9b. DMCWIN16, DMCWIN32 or DMCDOS communication software.

The WSDK software is highly recommended for first time users of the DMC-2x00. It provides step-by-step instructions for system connection, tuning and analysis.

Installing the DMC-2x00

Installation of a complete, operational DMC-2x00 system consists of 9 steps.

Step 1. Determine overall motor configuration.

Step 2. Install Jumpers on the DMC-2x00.

Step 3a. Configure the DIP switches on the DMC-2000.

Step 3b. Configure the DIP switches on the DMC-2100.

Step 3c. Configure the DIP switches on the DMC-2200

Step 4. Install the communications software.

Step 5. Connect AC power to controller.

Step 6. Establish communications with the Galil Communication Software.

Step 7. Determine the Axes to be used for sinusoidal commutation.

Step 8. Make connections to amplifier and encoder.

Step 9a. Connect standard servo motors.

Step 9b. Connect sinusoidal commutation motors

Step 9c. Connect step motors.

Step 10. 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-2x00 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-2x00 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-2x00 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-2x40.

Sinusoidal commutation is configured with the command, BA. For example, BAA sets the A axis to be sinusoidally commutated. The second DAC for the sinusoidal signal will be the highest available DAC on the controller. For example: Using a DMC-2x40, the command BAA will configure the A axis to be the main sinusoidal signal and the 'D' axis to be the second sinusoidal signal.

The BA command also reconfigures the controller to indicate that the controller has one less axis of 'standard' control for each axis of sinusoidal commutation. For example, if the command BAA is given to a DMC-2x40 controller, the controller will be re-configured to a DMC-2x30 controller. By definition, a DMC-2x30 controls 3 axes: A,B and C. The 'D' axis is no longer available since the output DAC is being used for sinusoidal commutation.

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

Stepper Motor Operation

To configure the DMC-2x00 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-2x00". Further instruction for stepper motor connections are discussed in Step 9.

Step 2. Install Jumpers on the DMC-2x00

Master Reset and Upgrade Jumpers

JP1 on the main board contains two jumpers, MRST and UPGRD. The MRST jumper is the Master Reset jumper. When MRST is connected, the controller will perform a master reset upon PC power up or upon the reset input going low. The MRST can also be set with the DIP switches on the outside of the controller. 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 opto-isolated. If you are not using an isolated supply, the internal +5V supply from the PC may be used to power the opto-isolators. This is done by installing jumpers on JP3 on main board.

Stepper Motor Jumpers

For each axis that will used for stepper motor operation, the corresponding stepper mode (SM) jumper must be connected. The stepper mode jumpers, labeled JP5 and JP7 are located directly beside the GL-1800 IC's on the main board (see the diagram of the DMC-2x00). The individual jumpers are labeled SMA thru SMH and configure the controller for 'Stepper Motors' for the corresponding axes A-H when installed. Note that the daughter board must be removed to access these jumpers. Contact the Galil factory if stepper motor jumpers should be placed on your controller with each order for a special part number.

(Optional) Motor Off Jumpers

The state of the motor upon power up may be selected with the placement of a hardware jumper on the controller. With a jumper installed at the MO location, the controller will be powered up in the "motor off" state. The SH command will need to be issued in order for the motor to be enabled. With no jumper installed, the controller will immediately enable the motor upon power up. The MO command will need to be issued to turn the motor off.

The MO jumper is always located on the same block of jumpers as the stepper motor jumpers (SM). This feature is only available to newer revision controllers. Please consult Galil for adding this functionality to older revision controllers.

Communications Jumpers for DMC-2000

The Main and Auxiliary Serial Communication Ports are normally connected for RS-232 connection. The jumpers JP3 and JP4 on the DMC-2001 daughter-board allows the DMC-2000 to be configured for RS-422. This can be specified as an option when the unit is purchased or the DMC-2000 may be re-configured by the user, please consult Galil for instructions. Other serial communication protocols, such as RS-485, can be implemented as a special - consult Galil.

Communications Jumpers for DMC-2100/DMC-2200

The main and Auxiliary Serial Commutations Ports are normally connected for RS-232 connection. The jumpers JP4 and JP5 on the DMC-21001 daughter board allows the controller to be configured for RS-422. This can be specified as an option when the unit is purchased or the controller may be re-configured by the user, please consult Galil for instructions. Other serial communications protocols, such as RS-485, can be implemented as a special - consult Galil.

Step 3a. Configure DIP switches on the DMC-2000

Located on the outside of the controller box is a set of 5 DIP switches. When the controller is powered on or reset, the state of the dip switches are read.

Switch 1 - Master Reset

When this switch is on, the controller will perform a master reset upon PC power up. Whenever the controller has a master reset, all programs and motion control parameters stored in EEPROM will be ERASED. During normal operation, this switch should be off.

Switch 2 - XON / XOFF

When on, this switch will enable software handshaking (XON/XOFF) through the main serial port.

Switch 3 - Hardware Handshake Mode

When on, this switch will enable hardware handshaking through the main serial port.

Switch 4,5 and 6 - Main Serial Port Baud Rate

The following table describes the baud rate settings:

9600	19.2	3800	BAUD RATE
ON	ON	OFF	1200
ON	OFF	OFF	9600
OFF	ON	OFF	19200
OFF	OFF	ON	38400
OFF	ON	ON	115200

Switch 10 - USB

When on, the controller will use the USB port as a default port for messages. When off, the controller will use the RS-232 port as default. When the firmware is updated, the controller will send the response (a colon), to the default port setting. If this is not the same port that was used to download the firmware, the Galil software will not return control to the user. In this case, the software will have to be re-started.

Step 3b. Configure DIP switches on the DMC-2100

Switch 1 - Master Reset

When this switch is on, the controller will perform a master reset upon PC power up. Whenever the controller has a master reset, all programs and motion control parameters stored in EEPROM will be ERASED. During normal operation, this switch should be off.

Switch 2 - XON / XOFF

When on, this switch will enable software handshaking (XON/XOFF) through the main serial port.

Switch 3 - Hardware Handshake Mode

When on, this switch will enable hardware handshaking through the main serial port.

Step 3c. Configure DIP switches on the DMC-2200

Switch 1 - Master Reset

When this switch is on, the controller will perform a master reset upon PC power up. Whenever the controller has a master reset, all programs and motion control parameters stored in EEPROM will be ERASED. During normal operation, this switch should be off.

Switch 2 - XON / XOFF

When on, this switch will enable software handshaking (XON/XOFF) through the main serial port.

Switch 3 - Hardware Handshake Mode

When on, this switch will enable hardware handshaking through the main serial port.

Switch 4,5 and 6 - Main Serial Port Baud Rate

The following table describes the baud rate settings:

9600	19.2	3800	BAUD RATE
ON	ON	OFF	1200
ON	OFF	OFF	9600
OFF	ON	OFF	19200
OFF	OFF	ON	38400
OFF	ON	ON	115200

Switch 7-Option

When OFF, the controller will use the auto-negotiate function to set the Ethernet connection speed. When the DIP switch is ON, the controller defaults to 10BaseT.

Switch 8-Ethernet

When ON, the controller will use the Ethernet port as the default port for unsolicited messages. When OFF, the controller will use the RS-232 port as the default. When the firmware is updated, the controller will send the response (a colon) to the default port setting. If this is not the same port that was used to download the firmware, the Galil software will not return control to the user. In this case, the software will have to be re-started.

Step 4. 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 3.x (16 bit versions):

Using the Galil Software CD ROM, go to the directory, DMCWIN16. Run DMCWIN16.exe at the Command prompt and follow the directions.

Using Windows 95,98, NT, ME 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 which can be used for communication.

Step 5. Connect AC Power to the Controller

Before applying power, connect the 100-pin cable between the DMC-2x00 and ICM-2900 interconnect module. The DMC-2x00 requires a single AC supply voltage, single phase, 50 Hz or 60 Hz. from 90 volts to 260 volts.

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

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

Step 6. Establish Communications with Galil Software

Communicating through the Main Serial Communications Port

Connect the DMC-2x00 MAIN serial port to your computer via the Galil CABLE-9PIN-D (RS-232 Cable).

Using Galil Software for DOS (DMC-2000 only)

To communicate with the DMC-2000, type TALK2DMC at the prompt. Once you have established communication, the terminal display should show a colon, :. If you do not receive a colon, press the carriage return. If a colon prompt is not returned, there is most likely an incorrect setting of the serial communications port. The user must ensure that the correct communication port and baud rate are specified when attempting to communicate with the controller. Please note that the serial port on the controller must be set for handshake mode for proper communication with Galil software. The user must also insure that the proper serial cable is being used, see appendix for pin-out of serial cable.

Using Galil Software for Windows

In order for the windows software to communicate with a Galil controller, the controller must be registered in the Windows Registry. To register a controller, you must specify the model of the controller, the communication parameters, and other information. The registry is accessed throught the Galil software, such as WSDK and DTERM (DTERM is installed with DMCWIN and installed as the icon "Galil Terminal"). From WSDK, the registry is accessed under the FILE menu. From the DTERM program, the registry is accessed from the REGISTRY menu.

The registry window is equipped with buttons to **Add, Change,** or **Delete** a controller. Pressing any of these buttons will bring up the Set Registry Information window.

Use the **Add** button to add a new entry to the Registry. You will need to supply the Galil Controller type. The controller model number must be entered and if you are changing an existing controller, this field will already have an entry. Pressing the down arrow to the right of this field will reveal a menu of valid controller types. You then need to choose serial or Ethernet connection. *Remember, a DMC-2000 connected via USB is plug-and play and should be automatically added to the registry upon connection.* The registry information will show a default Comm Port of 1 and a default Comm Speed of 9600 appears. This information should be changed as necessary to reflect the computers Comm Port and the baud rate set by the controller's DIP switches. The registry entry also displays timeout and delay information. These are advanced parameters which should only be modified by advanced users (see software documentation for more information).

Once you have set the appropriate Registry information for your controller, Select OK and close the registry window. You will now be able to communicate with the DMC-2x00. Once the entry has been selected, click on the **OK** button. If the software has successfully established communications with the controller, the registry entry will be displayed at the top of the screen.

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. In this case, there is most likely an incorrect setting of the serial communications port. The user must ensure that the correct communication port and baud rate are specified when attempting to communicate with the controller. Please note that the serial port on the controller must be set for handshake mode for proper communication with Galil software. The user must also insure that the proper serial cable is being used, see appendix for pin-out of serial cable.

Once you establish communications, click on the menu for terminal and you will receive a colon prompt. Communicating with the controller is described in later sections.

Using Non-Galil Communication Software

The DMC-2x00 main serial port is configured as DATASET. Your computer or terminal must be configured as a DATATERM for full duplex, no parity, 8 data bits, one start bit and one stop bit.

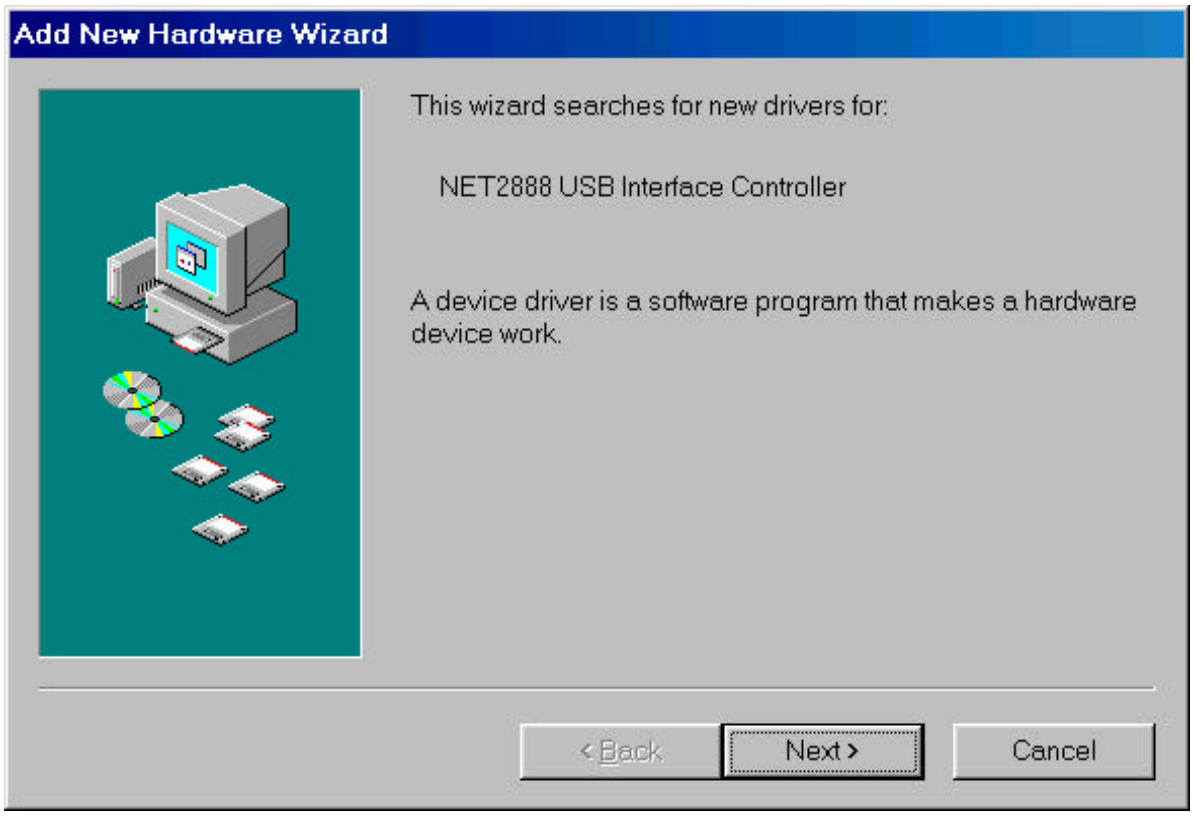
Check to insure that the baud rate switches have been set to the desired baud rate as described above.

Your computer needs to be configured as a "dumb" terminal which sends ASCII characters as they are typed to the DMC-2x00.

Communicating through the Universal Serial Bus (USB)

NOTE: Galil Software only supports the use of the USB port under Windows 98, ME and 2000.

Connect the USB cable from the computer to the USB IN port on the controller. Since the controller has been powered on in the previous step, the computer will recognize the first connection to a Galil USB controller. The computer will identify the USB port as a NET2888 device. The computer will then display the following screen:



Choose NEXT. Then choose "Search for the Best Driver for your device". Search the Galil CD-ROM and go to the directory INSTALL\DMC2000. The computer will recognize the file GALILUSB.INF. This driver will allow for the computer to communicate through USB with any DMC-2000 controllers.

Once the USB driver has been installed, new DMC-2000 controllers that are connected via USB, they will automatically be entered into the Galil Registry. Each controller is differentiated by it's unique serial number. If the Galil Registry entry is deleted, it will be re-instated when the controller is re-connected to the computer.

NOTE: If additional USB devices are connected to the DMC-2000 by using one of the two USB OUT ports, the computer will require the USBHUB.SYS file to be installed from the Windows Installation Disk.

Communicating through the Ethernet

Using Galil Software for Windows

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

From WSDK, the registry is accessed under the FILE menu. From DTERM it is accessed under the REGISTRY menu. Use the ***Add*** button to add a new entry in the registry. Choose DMC-2100 or DMC-2200 as the controller type. Enter the IP address obtained from your system administrator. Select the button corresponding to the UDP or TCP protocol in which you wish to communicate with the controller. If the IP address has not been already assigned to the controller, click on **ASSIGN IP ADDRESS.**

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

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

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

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

Sending Test Commands to the Terminal:

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

Now type

TPA <return>

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

0000000

Step 7. Determine the Axes to be Used for Sinusoidal Commutation

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

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

Notes on Configuring Sinusoidal Commutation:

The command, BA, reconfigures the controller such that it has one less axis of 'standard' control for each axis of sinusoidal commutation. For example, if the command BAA is given to a DMC-2x40 controller, the controller will be re-configured to be a DMC-2x30 controller. In this case the highest axis is no longer available except to be used for the 2nd phase of the sinusoidal commutation. Note that the highest axis on a controller can never be configured for sinusoidal commutation.

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

Example: Sinusoidal Commutation Configuration using a DMC-2x70

BAAC

This command causes the controller to be reconfigured as a DMC-2x50 controller. The A and C axes are configured for sinusoidal commutation. The first phase of the A axis will be the motor command

A signal. The second phase of the A axis will be F signal. The first phase of the C axis will be the motor command C signal. The second phase of the C axis will be the motor command G signal.

Step 8. Make Connections to Amplifier and Encoder.

Once you have established communications between the software and the DMC-2x00, you are ready to connect the rest of the motion control system. The motion control system typically consists of an ICM-2900 Interface Module, an amplifier for each axis of motion, and a motor to transform the current from the amplifier into torque for motion.

If you are using an ICM-2900, connect it to the DMC-2x00 via the 100-pin high density cable. The ICM-2900 provides screw terminals for access to the connections described in the following discussion.

2x80

Motion Controllers with more than 4 axes require a second ICM-2900 and 100-pin cable.

System connection procedures will depend on system components and motor types. Any combination of motor types can be used with the DMC-2x00. 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 AMPENA for the A axis on the ICM-2900 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. AMPEN can be used to disable the amplifier for these conditions.

The standard configuration of the AMPEN signal is TTL active high. In other words, the AMPEN 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-2900 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 AMPEN signal, note the state of the resistor pack on the ICM-2900. 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.

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-2x00 accepts single-ended or differential encoder feedback with or without an index pulse. If you are not using the ICM-2900 you will need to consult the appendix for the encoder pinouts for connection to the motion controller. The ICM-2900 accepts encoder feedback via individual signal leads. 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 A encoder first. Once it is connected, turn the motor shaft and interrogate the position with the instruction TPA <return>. The controller response will vary as the motor is turned.

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

1. The encoder connections are incorrect - check the wiring as necessary.
2. The encoder has failed - using an oscilloscope, observe the encoder signals. Verify that both channels A and B have a peak magnitude between 5 and 12 volts. Note that if only one encoder channel fails, the position reporting varies by one count only. If the encoder failed, replace the encoder. If you cannot observe the encoder signals, try a different encoder.
3. There is a hardware failure in the controller - connect the same encoder to a different axis. If the problem disappears, you 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-2x00 controller (bits 17-80).

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

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

BI ,, 6 or

BIC = 6

Step 9a. Connect Standard Servo Motors

The following discussion applies to connecting the DMC-2x00 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.

Step A. Check the Polarity of the Feedback Loop

It is assumed that the motor and amplifier are connected together and that the encoder is operating 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 A axis as an examples.

Step B. Set the Error Limit as a Safety Precaution

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

Input the commands:

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

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

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

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

Step C. Set Torque Limit as a Safety Precaution

To limit the maximum voltage signal to your amplifier, the DMC-2x00 controller has a torque limit command, TL. This command sets the maximum voltage output of the controller and can be used to avoid excessive torque or speed when initially setting up a servo system.

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

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

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

TL 1 <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 D. Connect the Motor

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

BGA <CR> Begin motion on A axis

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

Inverting the Loop Polarity

When the polarity of the feedback is incorrect, the user must invert the loop polarity and this may be accomplished by several methods. If you are driving a brush-type DC motor, the simplest way is to invert the two motor wires (typically red and black). For example, switch the M1 and M2 connections going from your amplifier to the motor. When driving a brushless motor, the polarity reversal may be done with the encoder. If you are using a single-ended encoder, interchange the signal 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:

TTA <return> Tell torque on A

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

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

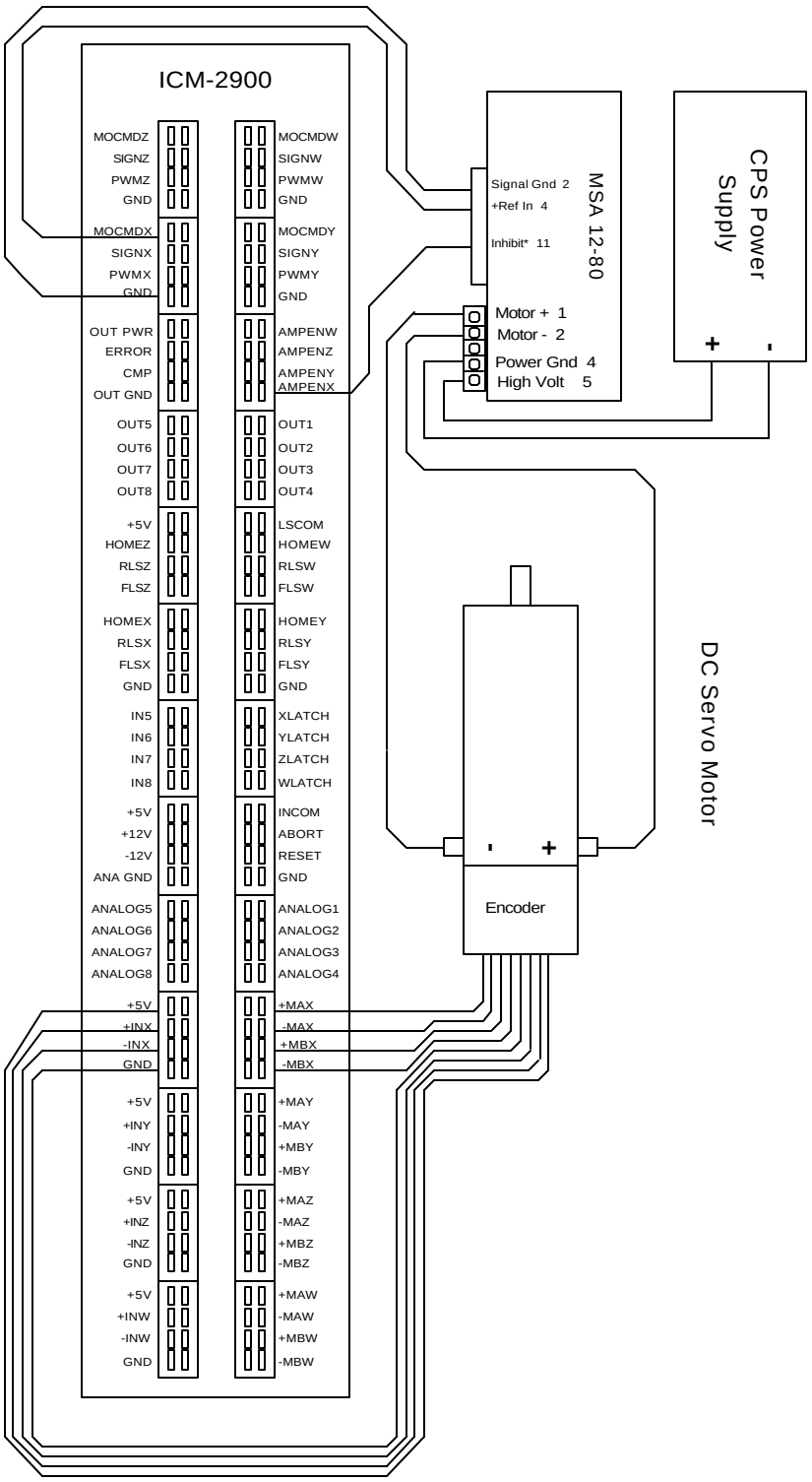


Figure 2-2 System Connections with a separate amplifier (MSA 12-80). This diagram shows the connections for a standard DC Servo Motor and encoder

Step 9b. Connect Sinusoidal Commutation Motors

When using sinusoidal commutation, the parameters for the commutation must be determined and saved in the controllers non-volatile memory. The setup for sinusoidal commutation is different when using Hall Sensors. Each step which is affected by Hall Sensor Operation is divided into two parts, part 1 and part 2. After connecting sinusoidal commutation motors, the servos must be tuned as described in Step 10.

Step A. Disable the motor amplifier

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

Step B. Connect the motor amplifier to the controller.

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

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

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

Example: Sinusoidal Commutation Configuration using a DMC-2x70

BAAC

This command causes the controller to be reconfigured as a DMC-2x50 controller. The A and C axes are configured for sinusoidal commutation. The first phase of the A axis will be the motor command A signal. The second phase of the A axis will be the motor command F signal. The first phase of the C axis will be the motor command C signal. The second phase of the C axis will be the motor command G signal.

Step C. Specify the Size of the Magnetic Cycle.

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

BM 62000

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

BM,, 1333.333

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

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

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

BSA = 2,700

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

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

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

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

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

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

Step E. Save Brushless Motor Configuration

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

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

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

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

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

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

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

BZ –2, 0,1

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

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

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

Method 1. Use the BZ command as described above.

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

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

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

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

Step 9c. Connect Step Motors

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

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-2x00 profiler commands the step motor amplifier. All DMC-2x00 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-2x00 you must follow this procedure:

Step A. Install SM jumpers

Each axis of the DMC-2x00 that will operate a stepper motor must have the corresponding stepper motor jumper installed. For a discussion of SM jumpers, see section *Step* 2. Install Jumpers on the DMC-2x00.

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 A-axis on the ICM-2900). Consult the documentation for your step motor amplifier.

Step C. Configure DMC-2x00 for motor type using MT command. You can configure the DMC-2x00 for active high or active low pulses. Use the command MT 2 for active low step motor pulses and MT -2 for active high step motor pulses. *See description of the MT command in the Command Reference.*

Step 10. Tune the Servo System

Adjusting the tuning parameters 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 <return> Integrator gain

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

KP 1 <return> Proportional gain

KD 100 <return> Derivative gain

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

TE A <return> Tell error

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

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 <return> Proportion gain

TE A <return> Tell error

As the proportional gain is increased, the error decreases.

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

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

TE A <return>

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

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.

System Set-up

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

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

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

Profiled Move

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

Instruction	Interpretation
PR1000	Distance
SP20000	Speed
DC 100000	Deceleration
AC 100000	Acceleration
BG A	Start Motion

Multiple Axes

Objective: Move the four axes independently.

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

Independent Moves

The motion parameters may be specified independently as illustrated below.

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

Position Interrogation

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

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

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

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

Absolute Position

Objective: Command motion by specifying the absolute position.

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

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

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

Velocity Control

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

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

after a few seconds, command:

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

and then

JG ,20000	New B speed
TV B	Returns B speed

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

ST	Stop

Operation Under Torque Limit

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

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

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

Increase the torque level gradually by instructions such as

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

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

Interrogation

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

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

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

Operation in the Buffer Mode

The instructions may be buffered before execution as shown below.

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

Using the On-Board Editor

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

The instruction

ED	Edit mode

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

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

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

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

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

Motion Programs with Loops

Motion programs may include conditional jumps as shown below.

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

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

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

Motion Programs with Trippoints

The motion programs may include trippoints as shown below.

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

AP ,50000	Wait until position B=50000
SP ,10000	Change speed of B
EN	End program

To start the program, command:

XQ #B	Execute Program #B

Control Variables

Objective: To show how control variables may be utilized.

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

To start the program, command

XQ #A	Execute Program #A

This program moves A to an initial position of 1000 and returns it to zero on increments of half the distance. Note, _TPA is an internal variable which returns the value of the A position. Internal variables may be created by preceding a DMC-2x00 instruction with an underscore, _.

Linear Interpolation

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

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

Circular Interpolation

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

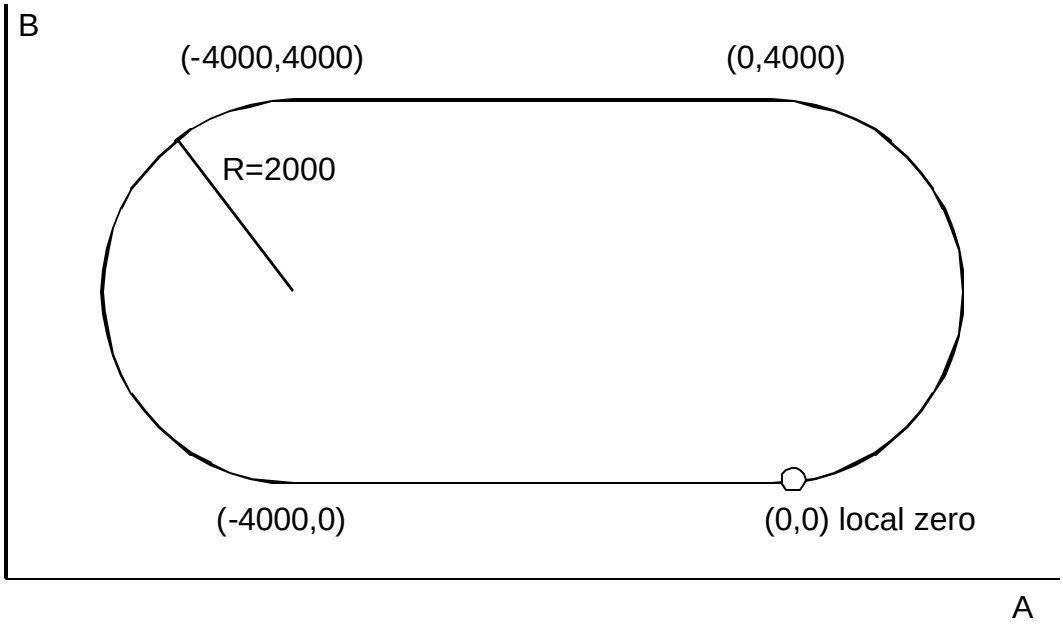


Figure 2-5 Motion Path for Circular Interpolation Example

Chapter 3 Connecting Hardware

Overview

The DMC-2x00 provides opto-isolated digital inputs for **forward limit**, **reverse limit, home**, and **abort** signals. The controller also has 8 **opto-isolated, uncommitted inputs (**for general use) as well as 8 TTL outputs and 8 analog inputs configured for voltages between +/- 10 volts.

2x80

Controllers with 5 or more axes have an additional 8 opto-isolated inputs and an additional 8 TTL outputs.

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

If you plan to use the auxiliary encoder feature of the DMC-2x00, you will require a separate encoder cable and breakout - contact Galil Motion control

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, A,B,C,D 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-2x00: Find Edge (FE), Find Index (FI), and Standard Home (HM).

The Find Edge routine is initiated by the command sequence: FEA <return>, BGA <return>. The Find Edge routine will cause the motor to accelerate, then slow 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. 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: FIA <return>, BGA <return>. Find Index will cause the motor to accelerate to the user-defined slew speed 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 slew speed and direction in which the motor will move is designated by the JG command. 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 HMA <return>, BGA <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 _HMA. 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.

Reset Input

When this input is pulled low (to 0 Volts), the controller will reset. This is equivalent to pushing the reset button on the front of the DMC-2x00.

Uncommitted Digital Inputs

The DMC-2x00 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 IN1goes high.

2x80

Controllers with more than 4 axes have 16 optoisolated inputs which are denoted as Inputs 1 thru 16.

Wiring the Opto-Isolated Inputs

The Opto-isolation inputs have a bi-directional capability. To activate an input, at least 1mA of current must flow from the input common through the input (see figure 3.1). This can be accomplished by 2 methods:

Method 1: Connect a positive voltage in the range of +5V to +24V (with respect to the input) at the input common point. Each input is connected to ground to activate the input.

Method 2: Connect ground to the input common point. Each input is activated by connecting a positive voltage between +5V and +24 Volts.

The Opto-Isolation Common Point

The opto-isolated inputs are configured into 2 groups. The general inputs, IN1-IN8, and the ABORT input are in one group. The signal, INCOM, is a common connection for all inputs in this group. The limit switches and home switches are in the second group. The signal, LSCOM, is a common connection for all inputs in this group. Figure 3.1 illustrates the internal circuitry.

Group (Controllers with 1- 4 Axes)	Group (Controllers with 5 - 9 Axes)	Common Signal
IN1-IN8, ABORT	IN1-IN16, ABORT	INCOM
FLA,RLA,HOMEA FLB,RLB,HOMEB FLC,RLC,HOMEC FLD,RLD,HOMED	FLA,RLA,HOMEA,FLB,RLB,HOMEB FLC,RLC,HOMEC,FLD,RLD,HOMED FLE,RLE,HOMEE,FLF,RLF,HOMEF FLG,RLG,HOMEG,FLH,RLH,HOMEH	LSCOM

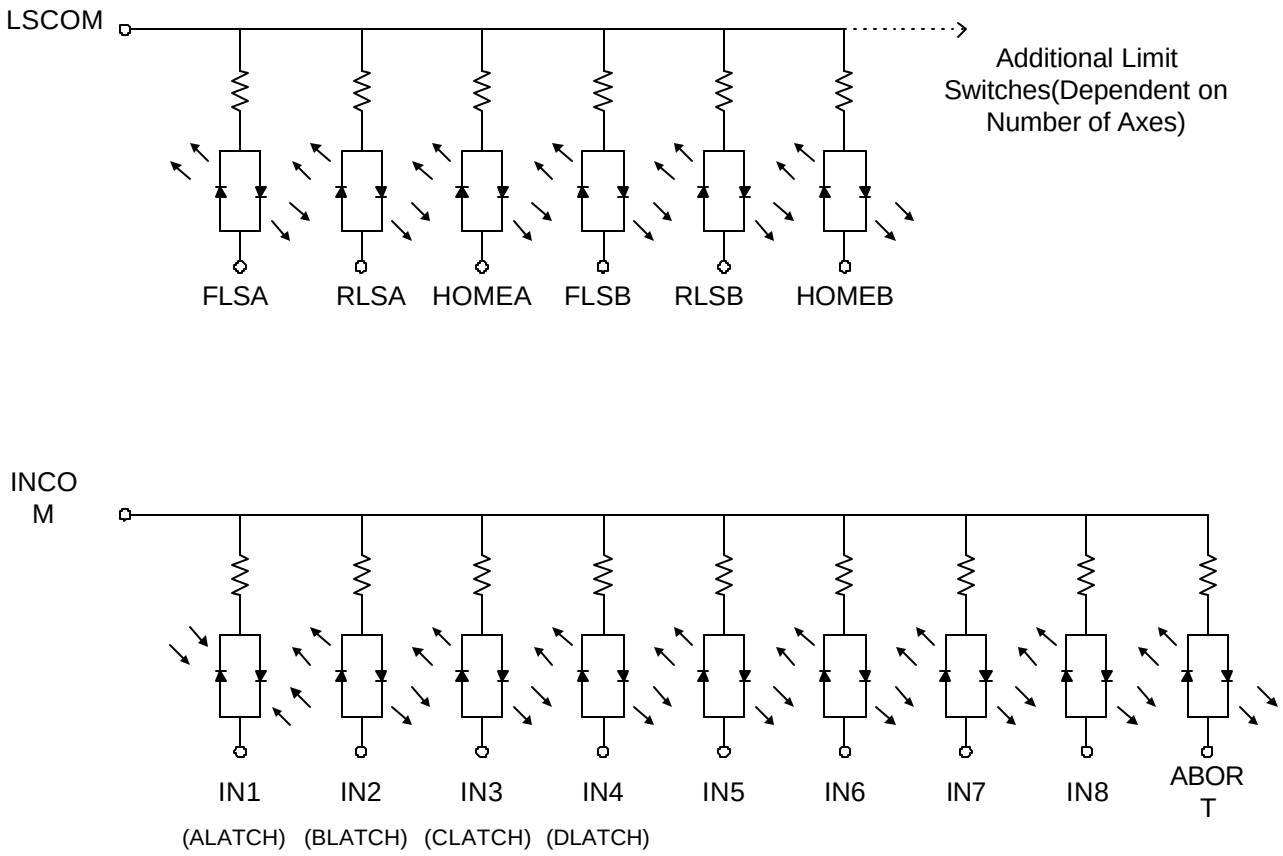


Figure 3-1. The Optoisolated Inputs.

NOTE: Controllers with 5 or axes have IN9 through IN16 also connected to INCOM.

Using an Isolated Power Supply

To take full advantage of opto-isolation, an isolated power supply should be connected to the input common. 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 24 Volts may be applied directly (see Figure 3-2). For voltages greater than 24 Volts, a resistor, R, is needed in series with the input such that

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

The DMC-2100/2200 provides three levels of Modbus communication. The first level allows the user to create a raw packet and receive raw data. It uses the MBh command with a function code of –1. The format of the command is

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

array[] is the array with the data

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

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

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

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

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

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

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

To view an example procedure for communicating with an OPTO-22 rack, refer to the appendix.

Which devices receive what information from the controller depends on a number of things. If a device queries the controller, it will receive the response unless it explicitly tells the controller to send it to another device. If the command that generates a response is part of a downloaded program, the response will route to whichever port is specified as the default (unless explicitly told to go to another port) with the ENET switch ("ON" designates Ethernet in which case it goes to the last handle to communicate with the controller, "OFF" designates main RS232). To designate a specific destination for the information, add {Eh} to the end of the command. (Ex. MG{EC}"Hello" will send the message "Hello" to handle #3. TP,,?{EF} will send the z axis position to handle #6.)

Multicasting

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

Using Third Party Software

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

The DMC-2100 can communicate with a host computer through any application that can send TCP/IP or UDP/IP packets. A good example of this is Telnet, a utility that comes with most Windows systems.

Data Record

The DMC-2x00 can provide a block of status information with the use of a single command, QR. This command, along with the QZ command can be very useful for accessing complete controller status. The QR command will return 4 bytes of header information and specific blocks of information as specified by the command arguments: QR ABCDEFGHST

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

Data Record Map

DATA TYPE	ITEM	BLOCK
UB	1st byte of header	Header
UB	2nd byte of header	Header
UB	3rd byte of header	Header
UB	4rth byte of header	Header
UW	sample number	I block

UB	general input 0	I block
UB	general input 1	I block
UB	general input 2	I block
UB	general input 3	I block
UB	general input 4	I block
UB	general input 5	I block
UB	general input 6	I block
UB	general input 7	I block
UB	general input 8	I block
UB	general input 9	I block
UB	general output 0	I block
UB	general output 1	I block
UB	general output 2	I block
UB	general output 3	I block
UB	general output 4	I block
UB	general output 5	I block
UB	general output 6	I block
UB	general output 7	I block
UB	general output 8	I block
UB	general output 9	I block
UB	error code	I block
UB	general status	I block
UW	segment count of coordinated move for S plane	S block
UW	coordinated move status for S plane	S block
SL	distance traveled in coordinated move for S plane	S block
UW	segment count of coordinated move for T plane	T block
UW	coordinated move status for T plane	T block
SL	distance traveled in coordinated move for T plane	T block
UW	a axis status	A block
UB	a axis switches	A block
UB	a axis stopcode	A block
SL	a axis reference position	A block
SL	a axis motor position	A block
SL	a axis position error	A block
SL	a axis auxiliary position	A block
SL	a axis velocity	A block
SW	a axis torque	A block
UW	b axis status	B block
UB	b axis switches	B block
UB	b axis stopcode	B block
SL	b axis reference position	B block
SL	b axis motor position	B block
SL	b axis position error	B block
SL	b axis auxiliary position	B block
SL	b axis velocity	B block

SW	b axis torque	B block
UW	c axis status	C block
UB	c axis switches	C block
UB	c axis stopcode	C block
SL	c axis reference position	C block
SL	c axis motor position	C block
SL	c axis position error	C block
SL	c axis auxiliary position	C block
SL	c axis velocity	C block
SW	c axis torque	C block
UW	d axis status	D block
UB	d axis switches	D block
UB	d axis stopcode	D block
SL	d axis reference position	D block
SL	d axis motor position	D block
SL	d axis position error	D block
SL	d axis auxiliary position	D block
SL	d axis velocity	D block
SW	d axis torque	D block
UW	e axis status	E block
UB	e axis switches	E block
UB	e axis stopcode	E block
SL	e axis reference position	E block
SL	e axis motor position	E block
SL	e axis position error	E block
SL	e axis auxiliary position	E block
SL	e axis velocity	E block
SW	e axis torque	E block
UW	f axis status	F block
UB	f axis switches	F block
UB	f axis stopcode	F block
SL	f axis reference position	F block
SL	f axis motor position	F block
SL	f axis position error	F block
SL	f axis auxiliary position	F block
SL	f axis velocity	F block
SW	f axis torque	F block
UW	g axis status	G block
UB	g axis switches	G block
UB	g axis stopcode	G block
SL	g axis reference position	G block
SL	g axis motor position	G block
SL	g axis position error	G block
SL	g axis auxiliary position	G block
SL	g axis velocity	G block

SW	g axis torque	G block
UW	h axis status	H block
UB	h axis switches	H block
UB	h axis stopcode	H block
SL	h axis reference position	H block
SL	h axis motor position	H block
SL	h axis position error	H block
SL	h axis auxiliary position	H block
SL	h axis velocity	H block
SW	h axis torque	H block

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

Explanation of Status Information and Axis Switch Information

Header Information - Byte 0, 1 of Header:

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

Bytes 2, 3 of Header:

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

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

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

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	Waiting for input from IN command	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 (2 Byte)

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 or Limit Switch	Motion is making final decel.	Latch is armed	Off-On-Error occurred	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 due to ST or Limit Switch	Motion is making final decel.	N/A	N/A	N/A

Notes Regarding Velocity and Torque Information

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

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

QZ Command

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

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

Controller Response to Commands

Most DMC-2x00 instructions are represented by two characters followed by the appropriate parameters. Each instruction must be terminated by a carriage return or semicolon.

Instructions are sent in ASCII, and the DMC-2x00 decodes each ASCII character (one byte) one at a time. It takes approximately .5 msec for the controller to decode each command. However, the PC can send data to the controller at a much faster rate because of the FIFO buffer.

After the instruction is decoded, the DMC-2x00 returns a response to the port from which the command was generated. If the instruction was valid, the controller returns a colon (:) or a question mark (?) if the instruction was not valid. For example, the controller will respond to commands which are sent via the USB port back through the USB port, to commands which are sent via the main RS-232 port back through the RS-232 port, and to commands which are sent via the Ethernet port back through the Ethernet port.

For instructions that return data, such as Tell Position (TP), the DMC-2x00 will return the data followed by a carriage return, line feed and : .

It is good practice to check for : after each command is sent to prevent errors. An echo function is provided to enable associating the DMC-2x00 response with the data sent. The echo is enabled by sending the command EO 1 to the controller.

Unsolicited Messages Generated by Controller

When the controller is executing a program, it may generate responses which will be sent via the USB port (DMC-2000), main RS-232 port, or Ethernet ports (DMC-2100/2200). This response could be generated as a result of messages using the MG or IN command OR as a result of a command error. These responses are known as unsolicited messages since they are not generated as the direct response to a command.

Messages can be directed to a specific port using the specific Port arguments - see MG and IN commands described in the Command Reference. If the port is not explicitly given, unsolicited messages will be sent to the default port. The default port is determined by the state of the USB/Ethernet dip switch when the system is reset.

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

When handshaking is used (hardware and/or software handshaking) characters which are generated by the controller are placed in a FIFO buffer before they are sent out of the controller. This size of the USB buffer is 64 bytes and the size of the RS-232 buffer is 128 bytes. When this buffer becomes full, the controller must either stop executing commands or ignore additional characters generated for output. The command CW,1 causes the controller to ignore all output from the controller while the

FIFO is full. The command, CW ,0 causes the controller to stop executing new commands until more room is made available in the FIFO. This command can be very useful when hardware handshaking is being used and the communication line between controller and terminal will be disconnected. In this case, characters will continue to build up in the controller until the FIFO is full. For more information, see the CW command in the Command Reference.

Galil Software Tools and Libraries

API (Application Programming Interface) software is available from Galil. The API software is written in C and is included in the Galil COMM disks. They can be used for development under DOS and Windows environments (16 and 32 bit Windows). With the API's, the user can incorporate already existing library functions directly into a C program.

Galil has also developed a Visual Basic Toolkit. This provides VBXs, 16-bit OCX's and 32-bit OCXs for handling all of the DMC-2x00 communications including support of interrupts. These objects install directly into Visual Basic and are part of the run-time environment. For more information, contact Galil.

Chapter 5 Command Basics

Introduction

The DMC-2x00 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-2x00 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-2x00, or an entire group of commands can be downloaded into the DMC-2x00 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-2x00 instruction set and syntax. A summary of commands as well as a complete listing of all DMC-2x00 instructions is included in the *Command Reference* chapter.

Command Syntax - ASCII

DMC-2x00 instructions are represented by two ASCII upper case characters followed by applicable arguments. A space may be inserted between the instruction and arguments. A semicolon or <return> is used to terminate the instruction for processing by the DMC-2x00 command interpreter.

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

IMPORTANT: All DMC-2x00 commands are sent in upper case.

For example, the command

PR 4000 <return> Position relative

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

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

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

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

The DMC-2x00 provides an alternative method for specifying data. Here data is specified individually using a single axis specifier such as A,B,C or D. An equals sign is used to assign data to that axis. For example:

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

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

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

2x80

For controllers with 5 or more axes, the axes are referred to as A,B,C,D,E,F,G,H.

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 and T are used to specify coordinated motion planes. For example:

BG S	Begin coordinated sequence, S
BG TW	Begin coordinated sequence, T, and D 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 is followed by data fields. The 4 bytes are specified in hexadecimal format.

Header Format:

Byte 1

Specifies the command number between 80 to FF. The complete binary command number table is listed below.

Byte 2

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

00	No 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 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

Datafields Format

Datafields 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 ABCS 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	I I	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-2x00 returns a : for valid commands and a ? for invalid commands.

For example, if the command BG is sent in lower case, the DMC-2x00 will return a ?.

:bg <return>	invalid command, lower case
?	DMC-2x00 returns a ?

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

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

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

Interrogating the Controller

Interrogation Commands

The DMC-2x00 has a set of commands that directly interrogate the controller. When the command is entered, the requested data is returned in decimal format on the next line followed by a carriage return and line feed. The format of the returned data can be changed using the Position Format (PF), Variable Format (VF) and Leading Zeros (LZ) command. See Chapter 7 and the Command Reference.

Summary of Interrogation Commands

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

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

TP A <return>	Tell position A
0000000000	Controllers Response
TP AB <return>	Tell position A and B
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 ?,?,?,?	Request A,B,C,D values
PR ,?	Request B value only

The controller can also be interrogated with operands.

Operands

Most DMC-2x00 commands have corresponding operands that can be used for interrogation. Operands must be used inside of valid DMC expressions. For example, to display the value of an operand, the user could use the command:

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

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

```
V=_TPA
```

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

Command Summary

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

THIS PAGE LEFT BLANK INTENTIONALLY

Chapter 6 Programming Motion

Overview

The DMC-2x00 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-2x10 is a single axis controller and uses A-axis motion only. Likewise, the DMC-2x20 uses A and B, the DMC-2x30 uses A,B and C, and the DMC-2x40 uses A,B,C and D. The DMC-2x50 uses A,B,C,D, and E. The DMC-2x60 uses A,B,C,D,E, and F. The DMC-2x70 uses A,B,C,D,E,F and G. The DMC-2x80 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 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-2x00 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-2x00 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. ABC or D 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 BG.

NOTE: If the motor is not moving, the IP command is equivalent to the PR and BG command combination.

Command Summary - Independent Axis

COMMAND	DESCRIPTION
PR A,B,C,D	Specifies relative distance
PA A,B,C,D	Specifies absolute position
SP A,B,C,D	Specifies slew speed
AC A,B,C,D	Specifies acceleration rate
DC A,B,C,D	Specifies deceleration rate
BG ABCD	Starts motion
ST ABCD	Stops motion before end of move
IP A,B,C,D	Changes position target
IT A,B,C,D	Time constant for independent motion smoothing
AM ABCD	Trippoint for profiler complete
MC ABCD	Trippoint for "in position"

The DMC-2x00 also allows use of single axis specifiers such as PRB=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

Examples

Absolute Position Movement

Instruction	Interpretation
PA 10000,20000	Specify absolute A,B position
AC 1000000,1000000	Acceleration for A,B
DC 1000000,1000000	Deceleration for A,B
SP 50000,30000	Speeds for A,B
BG AB	Begin motion

Multiple Move Sequence

Required Motion Profiles:

A-Axis	500 counts	Position
	10000 count/sec	Speed
	500000 counts/sec^2	Acceleration
B-Axis	1000 counts	Position
	15000 count/sec	Speed
	500000 counts/sec^2	Acceleration
C-Axis	100 counts	Position
	5000 counts/sec	Speed
	500000 counts/sec	Acceleration

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

Instruction	Interpretation
#A	Begin Program
PR 2000,500,100	Specify relative position movement of 1000, 500 and 100 counts for A,B and C 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 A	Begin motion on the A axis
WT 20	Wait 20 msec
BG B	Begin motion on the B axis
WT 20	Wait 20 msec
BG C	Begin motion on C axis
EN	End Program

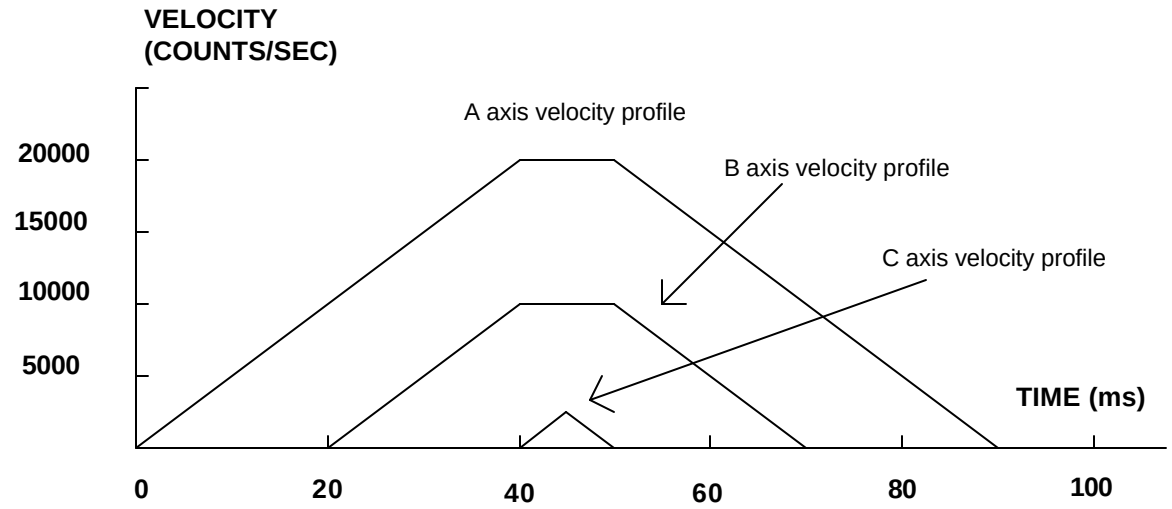


Figure 6.1 - Velocity Profiles of ABC

Notes on fig 6.1: The A and B axis have a 'trapezoidal' velocity profile, while the C axis has a 'triangular' velocity profile. The A and B axes accelerate to the specified speed, move at this constant speed, and then decelerate such that the final position agrees with the command position, PR. The C axis accelerates, but before the specified speed is achieved, must begin deceleration such that the axis will stop at the commanded position. All 3 axes have the same acceleration and deceleration rate, 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-2x00 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 A,B,C,D	Specifies acceleration rate
BG ABCD	Begins motion
DC A,B,C,D	Specifies deceleration rate
IP A,B,C,D	Increments position instantly
IT A,B,C,D	Time constant for independent motion smoothing
JG +/-A,B,C,D	Specifies jog speed and direction
ST ABCD	Stops motion

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

Operand Summary - Independent Axis

OPERAND	DESCRIPTION
_ACx	Return acceleration rate for the axis specified by 'x'
_DCx	Return deceleration rate for the axis specified by 'x'
_SPx	Returns the jog speed for the axis specified by 'x'
_TVx	Returns the actual velocity of the axis specified by 'x' (averaged over .25 sec)

Examples

Jog in X only

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

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

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.

Instruction	Interpretation
#JOY	Label
JG0	Set in Jog Mode
BGA	Begin motion
#B	Label for loop
vl =@AN[1]	Read analog input
vel=v1*50000/10	Compute speed
JG vel	Change JG speed
JP #B	Loop

Linear Interpolation Mode

The DMC-2x00 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 BC selects only the B and C 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 the Coordinate Plane

The DMC-2x00 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 Linear Segments

The command 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-2x00 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-2x00 computes the vector speed based on the axes specified in the LM mode. For example, LM ABC designates linear interpolation for the A,B and C axes. The vector speed for this example would be computed using the equation:

$VS^2=AS^2+BS^2+CS^2$, where AS, BS and CS are the speed of the A,B and C axes.

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

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.

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 a,b,c,d < n >m

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

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

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

As an example, consider the following program.

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

Changing Feedrate:

The command VR n allows the feedrate, 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 feedrate 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 abcdefgh	Specify axes for linear interpolation
LM?	Returns number of available spaces for linear segments in DMC-2x00 sequence buffer. Zero means buffer full. 512 means buffer empty.
LI a,b,c,d,e,f,g,h < n	Specify incremental distances relative to current position, and assign vector speed n.
VS n	Specify vector speed
VA n	Specify vector acceleration
VD n	Specify vector deceleration
VR n	Specify the vector speed ratio
BGS	Begin Linear Sequence
CS	Clear sequence
LE	Linear End- Required at end of LI command sequence
LE?	Returns the length of the vector (resets after 2147483647)
AMS	Trippoint for After Sequence complete
AV n	Trippoint for After Relative Vector distance, n
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-2x00 sequence buffer. Zero means buffer full. 512 means buffer empty.
_VPm	Return the absolute coordinate of the last data point along the trajectory. (m= A,B,C,D,E,F,G or H)

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

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

Example

Linear Interpolation Motion

In this example, the AB 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.

Instruction	Interpretation
#LMOVE	Label

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

Linear Move

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

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

Note that the above program specifies the vector speed, VS, and not the actual axis speeds VC and VD. The axis speeds are determined by the DMC-2x00 from:

$$VS = \sqrt{VC^2 + VD^2}$$

The resulting profile is shown in Figure 6.2.

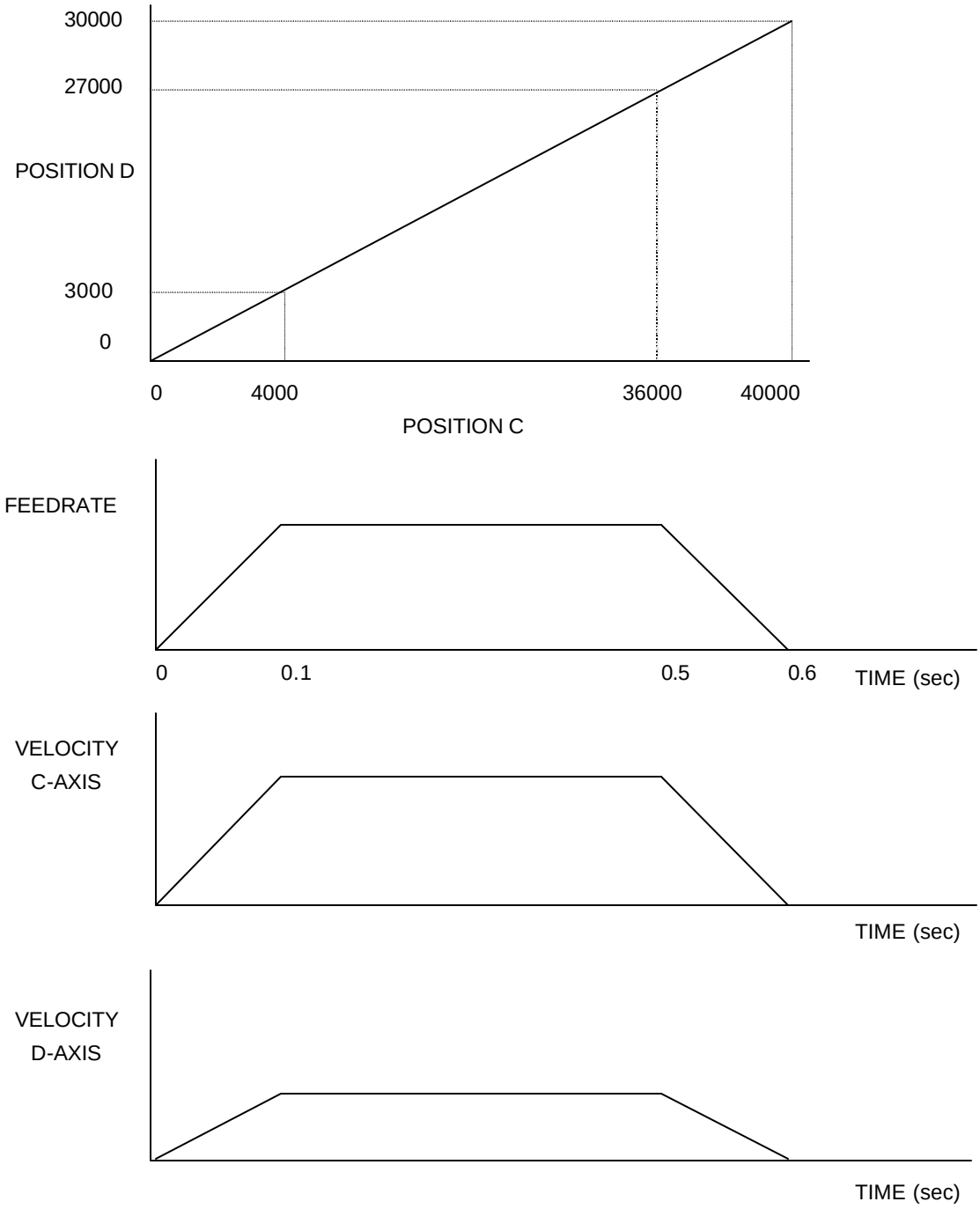


Figure 6.2 - Linear Interpolation

Multiple Moves

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

Instruction	Interpretation
#LOAD	Load Program
DM VA [750],VB [750]	Define Array
count=0	Initialize Counter
n=0	Initialize position increment
#LOOP	LOOP
VA [count]=n	Fill Array VA
VB [count]=n	Fill Array VB
n=n+10	Increment position
count = count +1	Increment counter
JP #LOOP, count <750	Loop if array not full
#A	Label
LM AB	Specify linear mode for AB
count =0	Initialize array counter
#LOOP2;JP#LOOP2,_LM=0	If sequence buffer full, wait
JS#C, count =500	Begin motion on 500th segment
LI VA[count],VB[count]	Specify linear segment
count = count +1	Increment array counter
JP #LOOP2, count <750	Repeat until array done
LE	End Linear Move
AMS	After Move sequence done
MG "DONE"	Send Message
EN	End program
#C;BGS;EN	Begin Motion Subroutine

Vector Mode: Linear and Circular Interpolation Motion

The DMC-2x00 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-2x00 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 ABC selects the AD axes for coordinated motion and the C-axis as the tangent.

Specifying the Coordinate Plane

The DMC-2x00 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 xy specifies the coordinates of the end points of the vector movement with respect to the starting point. Non-sequential axes do not require comma delimitation. The command, CR r,q,d define a circular arc with a radius r, starting angle of q, and a traversed angle d. The notation for q is that zero corresponds to the positive horizontal direction, and for both q and d, the counter-clockwise (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-2x00 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 a,b < 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 Feedrate:

The command VR n allows the feedrate, 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 feedrate 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-2x00 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-2x00 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 A,B,C or D p=N turns off tangent axis

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

Command Summary - Coordinated Motion Sequence

Command	Description
VM m,n	Specifies the axes for the planar motion where m and n represent the planar axes and p is the tangent axis.
VP m,n	Return coordinate of last point, where m=A,B,C or D.
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 n	Specify vector speed or feedrate of sequence.

VA n	Specify vector acceleration along the sequence.
VD n	Specify vector deceleration along the sequence.
VR n	Specify vector speed ratio
BGS	Begin motion sequence.
CS	Clear sequence.
AV n	Trippoint for After Relative Vector distance, n.
AMS	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 curve smoothing constant for coordinated moves
LM?	Return number of available spaces for linear and circular segments in DMC-2x00 sequence buffer. Zero means buffer is full. 512 means buffer is empty.

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-2x00 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 _VPA and _VPB can be used to return the coordinates of the last point specified along the path.

Example

Tangent Axis

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

Instruction	Interpretation
#EXAMPLE	Example program
VM ABC	AB coordinate with C as tangent
TN 2000/360,-500	2000/360 counts/degree, position -500 is 0 degrees in AB plane
CR 3000,0,180	3000 count radius, start at 0 and go to 180 CCW
VE	End vector
CB0	Disengage knife

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

Coordinated Motion

Traverse the path shown in Fig. 6.3. Feedrate is 20000 counts/sec. Plane of motion is AB

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

_VPA and _VPB contain the absolute coordinate of the point A

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

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

The value of _CS is 2

_VPA,_VPB contain the coordinates of the point C

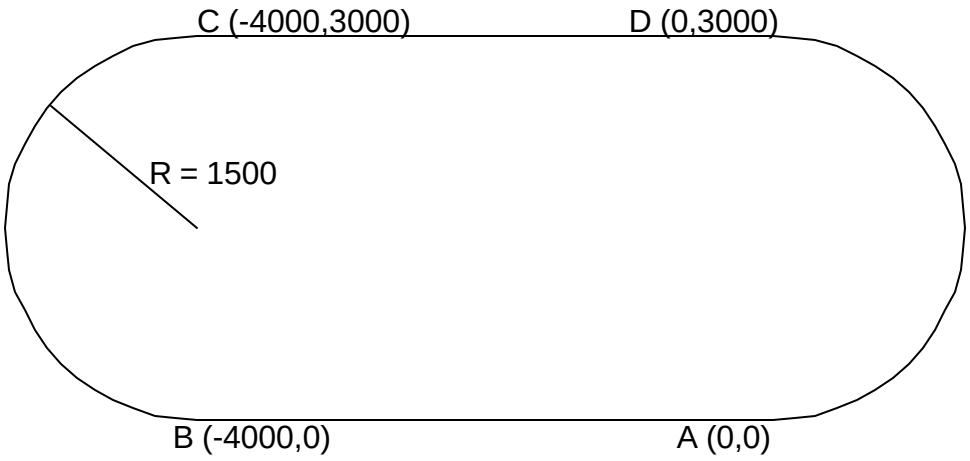


Figure 6.3 - The Required Path

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 GA ABCDEFGH specifies the master axes. GR a,b,c,d 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 0,0,0,0 turns off gearing in both modes. A limit switch or ST command disable gearing in the standard mode but not in the gantry mode.

The command GM a,b,c,d 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, GACA indicates that the gearing is the commanded position of A.

An alternative gearing method is to synchronize the slave motor to the commanded vector motion of several axes performed by GAS. For example, if the A and B motor form a circular motion, the C axis may move in proportion to the vector move. Similarly, if A,B and C perform a linear interpolation move, 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 = A,B,C,D,E,F,G,H for main encoder as master.
	n = CA,CB,CC,CD,CE,CF,CG,CH for commanded position.
	n = DA, DB, DC, DD, DE, DF,DG,DH for auxiliary encoders.
	n = S or T for gearing to coordinated motion.
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	1 sets gantry mode, 0 disables gantry mode.
MR a,b,c,d	Trippoint for reverse motion past specified value. Only one field may be used.
MF a,b,c,d	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. B is defined as the master. A,C,D are geared to master at ratios of 5,-.5 and 10 respectively.

Instruction	Interpretation
GA B,,B,B	Specify master axes as B
GR 5,,-.5,10	Set gear ratios
PR ,10000	Specify B position
SP ,100000	Specify B speed
BGB	Begin motion

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-2x30 controller, where the C-axis is the master and A and B are the geared axes.

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

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

GR 2	Specify gear ratio for A axis to be 2

Gantry Mode

In applications where both the master and the follower are controlled by the DMC-2x00 controller, it may be desired to synchronize the follower with the commanded position of the master, rather than the actual position. This eliminates the coupling between the axes which may lead to oscillations.

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

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

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

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

Under these conditions, this IP command is equivalent to:

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

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

Synchronize two conveyor belts with trapezoidal velocity correction.

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

Electronic Cam

The electronic cam is a motion control mode which enables the periodic synchronization of several axes of motion. Up to 7 axes 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-2x80 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 B, when the master is A. Such a graphic relationship is shown in Figure 6.4.

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 = A,B,C,D

p is the selected master axis

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

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

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 a and b are redefined as zero. To specify the master cycle and the slave cycle change, we use the instruction EM.

```
EM a,b,c,d
```

where a,b,c,d 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]= a,b,c,d
```

where n indicates the order of the point.

The value, n, starts at zero and may go up to 256. The parameters A,B,C,D 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 a,b,c,d

where a,b,c,d 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 a,b,c,d

where a,b,c,d are the master positions at which the corresponding slave axes are disengaged.

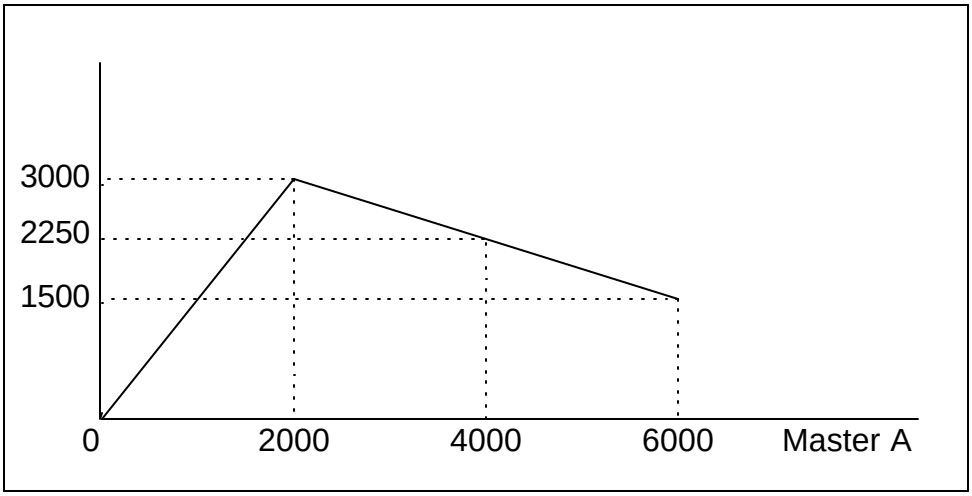


Figure 6.4: Electronic Cam Example

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

Step 8. Create program to ***generate*** ECAM table

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

the equation:

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

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

The cam table can be constructed manually, point by point, or automatically by a program. The following program includes the set-up. The instruction EAA defines A as the master axis. The cycle of the master is 2000. Over that cycle, A varies by 1000. This leads to the instruction EM 2000,1000.

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

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

Instruction	Interpretation
#SETUP	Label
EAA	Select A as master
EM 2000,1000	Cam cycles
EP 20,0	Master position increments
n = 0	Index
#LOOP	Loop to construct table from equation
p = n*3.6	Note 3.6 = 0.18*20
s = @SIN [P] *100	Define sine position
b = n *10+s	Define slave position
ET [n] =, b	Define table
n = n+1	Update Counter
JP #LOOP, n<=100	Repeat the process
EN	End Program

Step 9. Create program to ***run*** ECAM mode

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

This is done with the program:

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

Command Summary - Electronic CAM

Command	Description
EA p	Specifies master axes for electronic cam where:
EB n	Enables the ECAM
EC n	ECAM counter - sets the index into the ECAM table
EG a,b,c,d	Engages ECAM
EM a,b,c,d	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

Operand Summary - Electronic CAM

command	description
_EB	Contains State of ECAM
_EC	Contains current ECAM index
_EGa	Contains ECAM status for each axis
_EM	Contains size of cycle for each axis
_EP	Contains value of the ECAM table interval
_EQx	Contains ECAM status for each axis

Example

Electronic CAM

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

Instruction	Interpretation
#A;vl=0	Label; Initialize variable
PA 0,0;BGAB;AMAB	Go to position 0,0 on A and B axes
EA C	C axis as the Master for ECAM
EM 0,0,4000	Change for C is 4000, zero for A, B
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
JGC=4000	Set C to jog at 4000
EG 0,0	Engage both A and B when Master = 0
BGC	Begin jog on C axis
#LOOP;JP#LOOP,vl=0	Loop until the variable is set
EQ2000,2000	Disengage A and B when Master = 2000
MF,, 2000	Wait until the Master goes to 2000
ST C	Stop the C axis motion
EB 0	Exit the ECAM mode
EN	End of the program

The above example shows how the ECAM program is structured and how the commands can be given to the controller. 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 A axis, the second graph shows the cycle on the B axis and the third graph shows the cycle of the C axis.

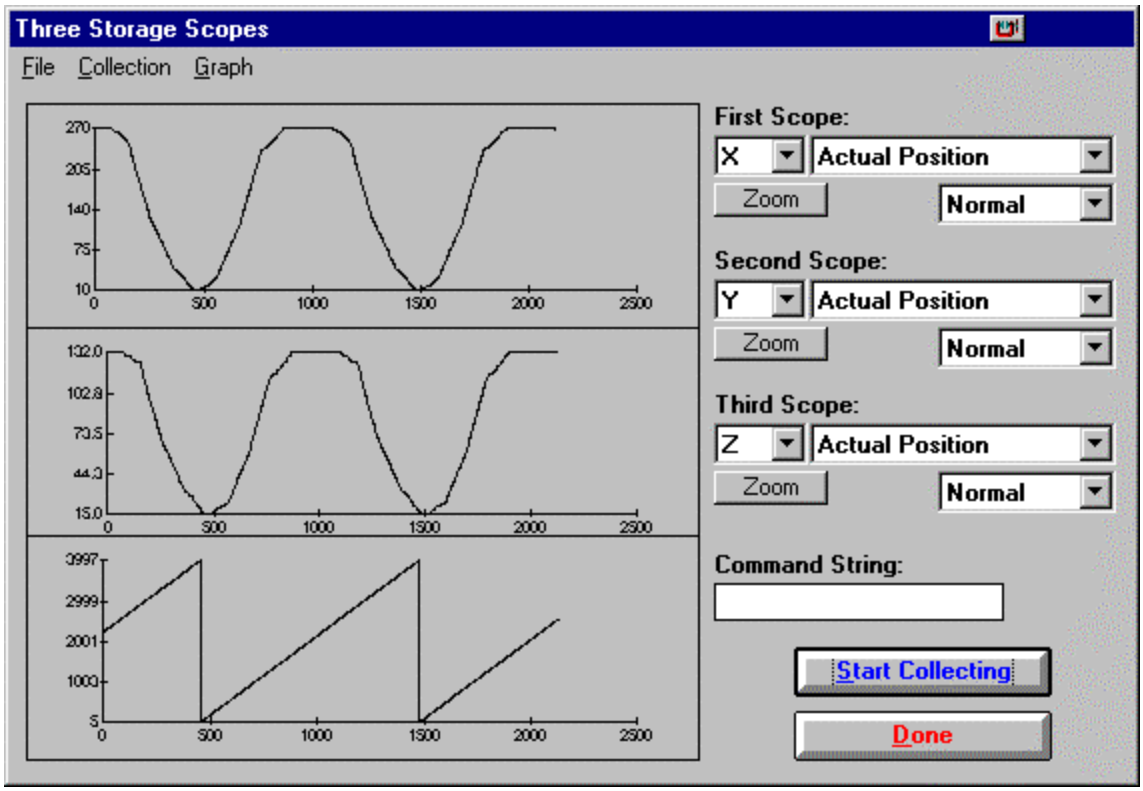


Figure 6.5 - Velocity Profiles of XYZ

Contour Mode

The DMC-2x00 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, CMAC specifies contouring on the A and C 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 a,b,c,d 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 A may be described by the points:

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

The same trajectory may be represented by the increments

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

When the controller receives the command to generate a trajectory along these points, it 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:

Instruction	Interpretation
#A	Label
CMA	Specifies A 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	

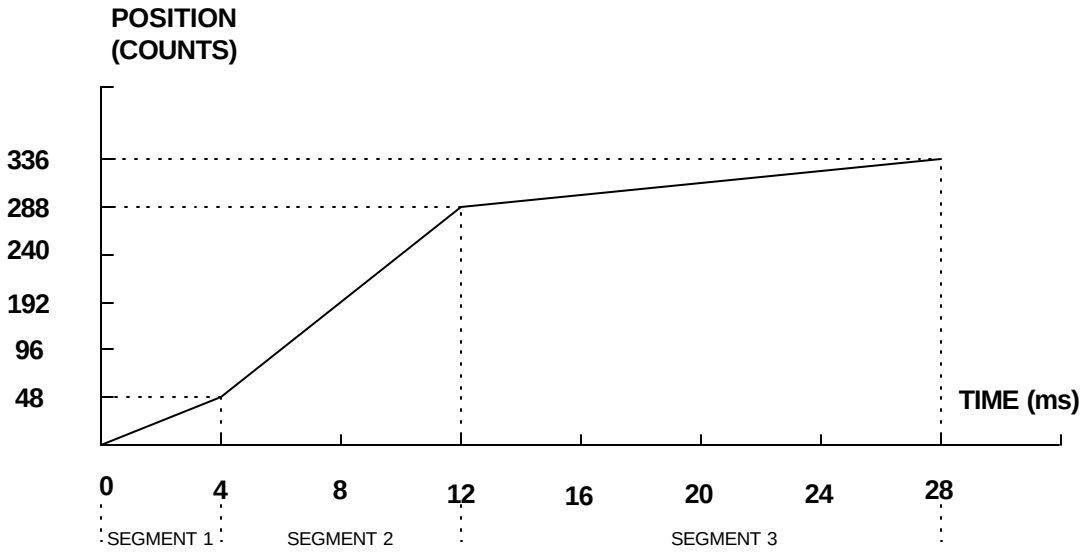


Figure 6.6 - The Required Trajectory

Additional Commands

The command, WC, is used as a trippoint "When Complete". This allows the DMC-2x00 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 ABCDEFGH	Specifies which axes for contouring mode. Any non-contouring axes may be operated in other modes.
CD a,b,c,d,e,f,g,h	Specifies position increment over time interval. Range is +/-32,000. (Zero ends contour mode, when issued following DT0)
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.

Example

Generating an Array

Consider the velocity and position profiles shown in Fig. 6.5. 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:

$$w = \frac{\mathrm{A}}{\mathrm{B}}\left(1 - \cos(2p\mathrm{T}/\mathrm{B})\right)$$

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

NOTE: ω is the angular velocity; A 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:

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

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

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

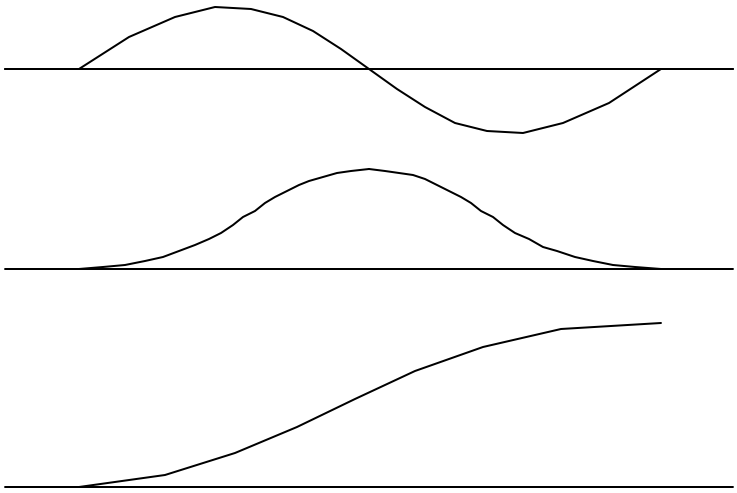


Figure 6.7 - Velocity Profile with Sinusoidal Acceleration

The DMC-2x00 can compute trigonometric functions. However, the argument must be expressed in degrees. Using our example, the equation for A is written as:

A = 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 dir. Finally the motors are run in the contour mode.

Contour Mode

Instruction	Interpretation
#POINTS	Program defines A points
DM pos[16]	Allocate memory
DM dir[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	
c=c+1	
dir[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
CMA	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-2x00 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-2x40)
RD _TPA	Specify data for capturing (such as _TPA or _TPC)
RC n,m	Specify capture time interval where n is 2^n samples, m is number of records to be captured
RC? or _RC	Returns a 1 if recording

Record and Playback Example

Instruction	Interpretation
#RECORD	Begin Program
DP0	Define position for A axis to be 0
DA*[ ]	De-allocate all arrays
DM xpos [501]	Dimension 501 element array called xpos
RA xpos [ ]	Record Elements into xpos array
RD_TPA	Element to be recorded is encoder position of A axis
MOA	Motor off for A axis
RC2	Begin Recording with a sample rate of 2^2 msec
#LOOP1;JP#LOOP1,_RC=1	Loop until all elements have been recorded
#COMPUTE	Routine to determine the difference between consecutive points
DM dx [500]	Dimension a 500 element array to hold contour points
i = 0	Set loop counter
#LOOP2	Loop to calculate the difference
DX[I]= xpos [i+1]- xpos [i]	Calculate difference
i=i+1	Update loop counter
JP#LOOP2,i<500	Continue looping until dx is full
#PLAYBK	Routine to play back motion that was recorded

SHA	Servo Here
WT1000	Wait 1 sec (1000 msec)
CMA	Specify contour mode on A axis
DT2	Set contour data rate to be 2^2 msec
i=0	Set array index to 0
#LOOP3	Subroutine to execute contour points
CD dx[i];WC	Contour data command; Wait for next contour point
i=i+1	Update index
JP#LOOP3,i<500	Continue until all array elements have been executed
DT0	Set contour update rate to 0
CD0	Disable the contour mode (combination of DT0 and CD0)
EN	End program

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

Virtual Axis

The DMC-2x00 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, EA.

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

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

For example,

```
PAN = 2000
BGN
```

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

Sinusoidal Motion Example

The 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 A and N axes to perform circular motion. Note that the value of VS must be

$$VS = 2p * R * F$$

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

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

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

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

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

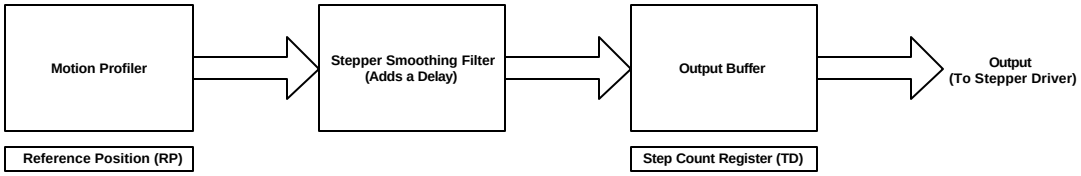


Figure 6.8 - Velocity Profiles of ABC

Motion Complete Trippoint

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

Using an Encoder with Stepper Motors

An encoder may be used on a stepper motor to check the actual motor position with the commanded position. If an encoder is used, it must be connected to the main encoder input.

NOTE: The auxiliary encoder is not available while operating with stepper motors. The position of the encoder can be interrogated by using the command, TP. The position value can be defined by using the command, DE.

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

Command Summary - Stepper Motor Operation

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

MT	Motor Type (2,-2,2.5 or -2.5 for stepper motors)
RP	Report Commanded Position
TD	Report number of step pulses generated by controller
TP	Tell Position of Encoder

Operand Summary - Stepper Motor Operation

operand	Description
_DEa	Contains the value of the step count register for the ‘a’ axis
_DPa	Contains the value of the main encoder for the ‘a’ axis
_ITa	Contains the value of the Independent Time constant for the 'a' axis
_KSa	Contains the value of the Stepper Motor Smoothing Constant for the 'a' axis
_MTa	Contains the motor type value for the 'a' axis
_RPa	Contains the commanded position generated by the profiler for the ‘a’ axis
_TDa	Contains the value of the step count register for the ‘a’ axis
_TPa	Contains the value of the main encoder for the ‘a’ axis

Dual Loop (Auxiliary Encoder)

The DMC-2x00 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 a,b,c,d (or a,b,c,d,e,f,g,h for controllers with more than 4 axes) where the parameters a,b,c,d each equal the sum of two integers m and n. m configures the main encoder and n configures the auxiliary encoder.

NOTE: This operation is not available for axes configured for stepper motors.

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 A axis is

CE 6

Additional Commands for the Auxiliary Encoder

The command, DE a,b,c,d 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 A and C auxiliary encoders.

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

```
V1= _DEA
```

The command, TD a,b,c,d, returns the current position of the auxiliary encoder.

The command, DV a,b,c,d, configures the auxilliary encoder to be used for backlash compensation.

Backlash Compensation

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

1. Continuous dual loop
2. Sampled dual loop

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

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

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

Example

Continuous Dual Loop

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

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 A-axis and connect the linear encoder to the auxiliary encoder of A. 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
BGA	Start motion
#CORRECT	Correction loop
AMA	Wait for motion completion
v1=10000-_DEA	Find linear encoder error
v2=-_TEA/4+v1	Compensate for motor error
JP#END,@ABS[v2]<2	Exit if error is small
PR v2*4	Correction move
BGA	Start correction
JP#CORRECT	Repeat
#END	
EN	

Motion Smoothing

The DMC-2x00 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:

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, has continuous acceleration and results in reduced mechanical vibrations.

The smoothing function is specified by the following commands:

IT a,b,c,d	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, a,b,c,d 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. 6.6 shows the trapezoidal velocity profile and the modified acceleration and velocity.

Note that the smoothing process results in longer motion time.

Example

Instruction	Interpretation
PR 20000	Position
AC 100000	Acceleration
DC 100000	Deceleration
SP 5000	Speed
IT .5	Filter for smoothing
BG A	Begin

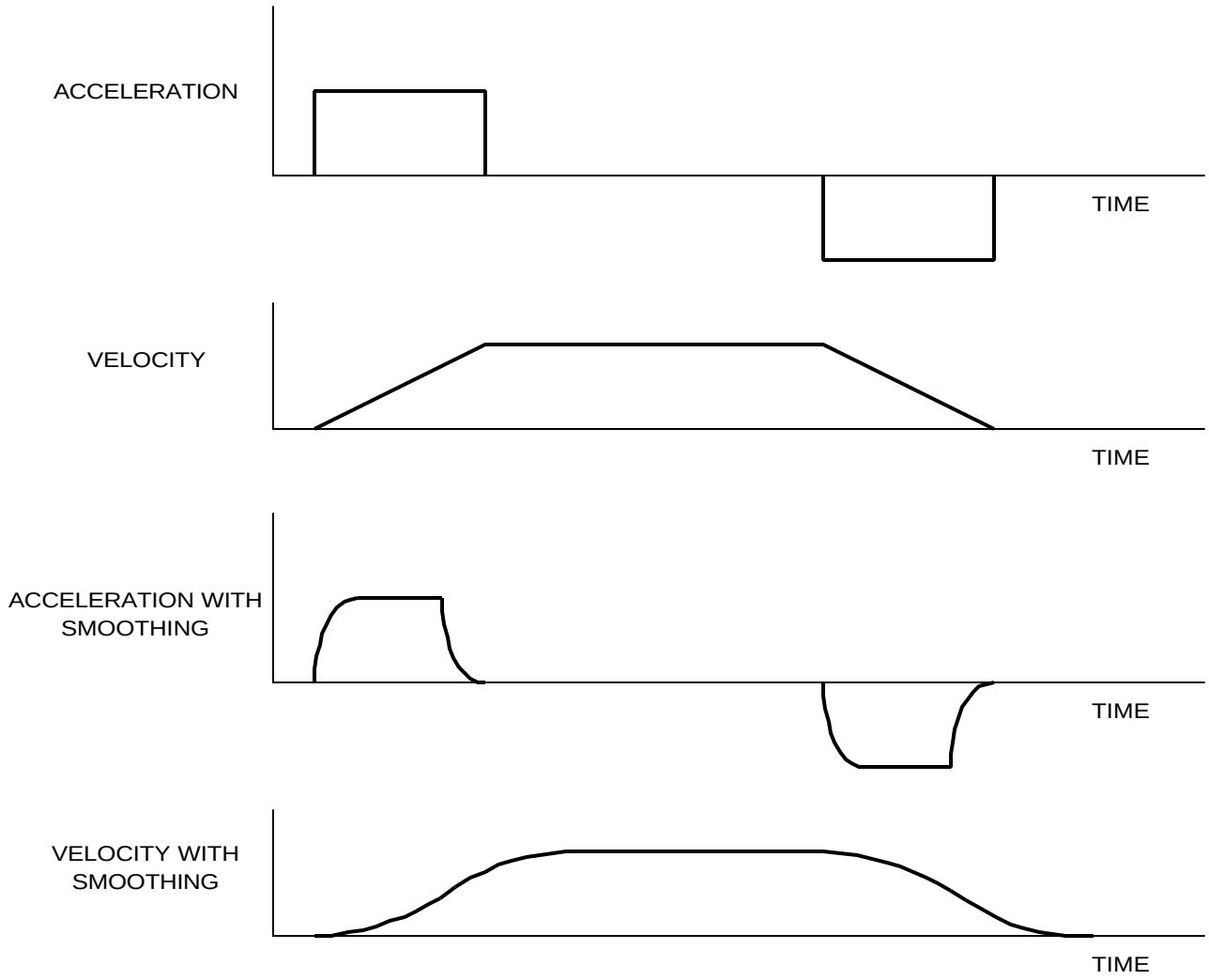


Figure 6.9 - Trapezoidal velocity and smooth velocity profiles

Using the KS Command (Step Motor Smoothing):

When operating with step motors, motion smoothing can be accomplished with the command, KS. The KS command smoothes the frequency of step motor pulses. Similar to the commands, IT and VT, this produces a smooth velocity profile.

The step motor smoothing is specified by the following command:

KS a,b,c,d	where a,b,c,d is an integer from 0.5 to 8 and represents the amount of smoothing

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, a,b,c,d and n are numbers between 0.5 and 8 and determine the degree of filtering. The minimum value of 0.5 implies no filtering, resulting in trapezoidal velocity profiles. Larger values of the smoothing parameters imply heavier filtering and smoother moves.

Note that KS is valid only for step motors.

Homing

The Find Edge (FE) and Home (HM) instructions may be used to home the motor to a mechanical reference. This reference is connected to the Home input line. The HM command initializes the motor to the encoder index pulse in addition to the Home input. The configure command (CN) is used to define the polarity of the home input.

The Find Edge (FE) instruction is useful for initializing the motor to a home switch. The home switch is connected to the Homing Input. When the Find Edge command and Begin is used, the motor will accelerate up to the slew speed and slew until a transition is detected on the Homing line. The motor will then decelerate to a stop. A high deceleration value must be input before the find edge command is issued for the motor to decelerate rapidly after sensing the home switch. The velocity profile generated is shown in Fig. 6.7.

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-2x00 defines the home position (0) as the position at which the index was detected.

Example

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

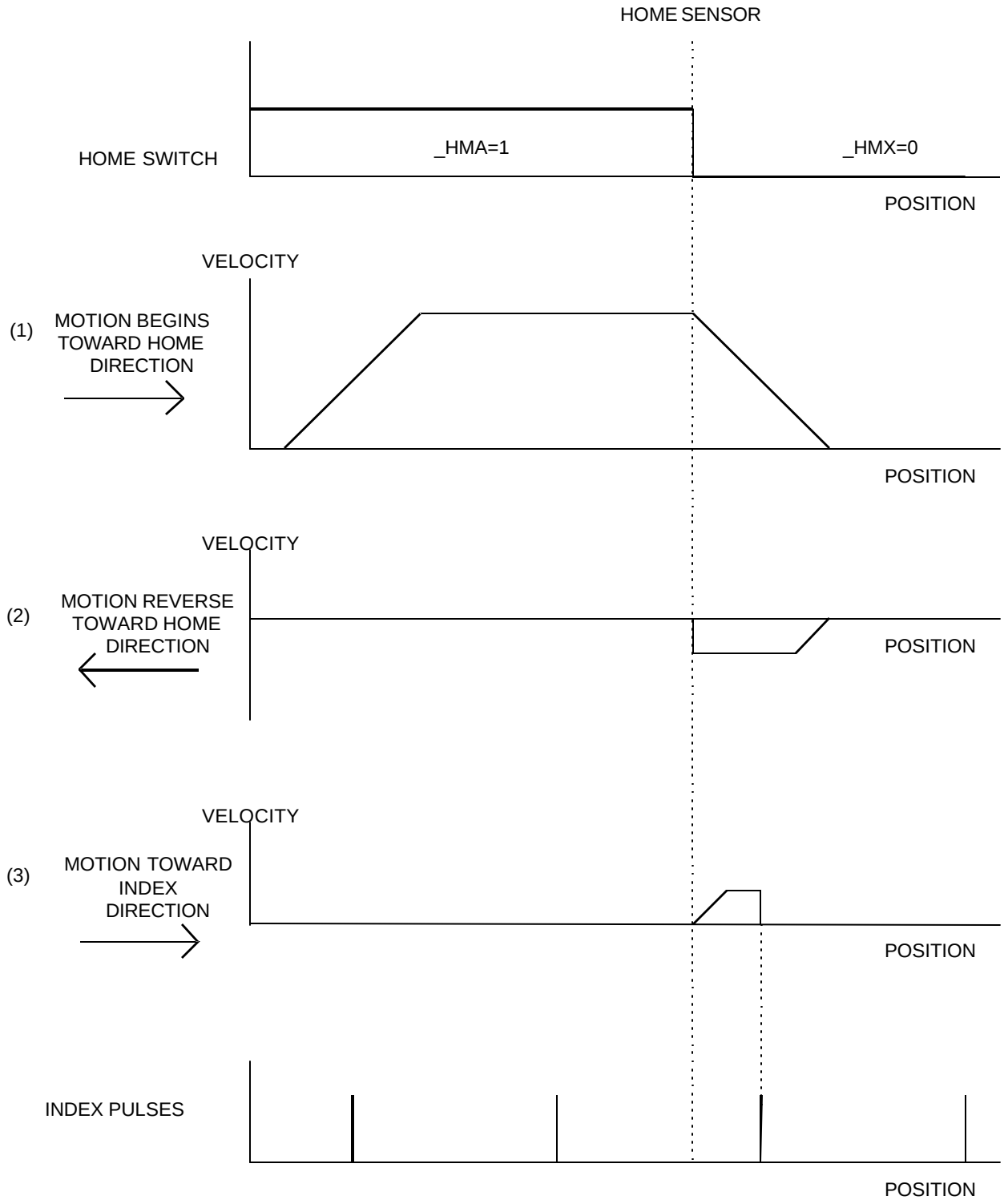


Figure 6.10 - Motion intervals in the Home sequence

Command Summary - Homing Operation

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

Operand Summary - Homing Operation

Operand	Description
_HMa	Contains the value of the state of the Home Input
_SCa	Contains stop code
_TSa	Contains status of switches and inputs

High Speed Position Capture (The Latch Function)

Often it is desirable to capture the position precisely for registration applications. The DMC-2x00 provides a position latch feature. This feature allows the position of the main or auxiliary encoders of A,B,C or D to be captured when the latch input changes state. This function can be setup such that the position is captured when the latch input goes high or low. When the latch function is enabled for active low operation, the position will be captured within 12 microseconds. When the latch function is enabled for active high operation, the position will be captured within 35 microseconds. Each axis has one general input associated to the axis for position capture:

Input	Function	Input	Function
IN1	A Axis Latch	IN9	E Axis Latch
IN2	B Axis Latch	IN10	F Axis Latch
IN3	C Axis Latch	IN11	G Axis Latch
IN4	D Axis Latch	IN12	H Axis Latch

The DMC-2x00 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 ABCD command to arm the latch for the main encoder and ALSASBSCSD for the auxiliary encoders.
2. Test to see if the latch has occurred (Input goes low) by using the _AL A or B or C or D command. Example, V1=_ALA returns the state of the A 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 ABCD command or _RL ABCD.

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

Example

Instruction	Interpretation
#LATCH	Latch program
JG,5000	Jog B
BG B	Begin motion on B axis
AL B	Arm Latch for B axis
#WAIT	#Wait label for loop
JP #WAIT,_ALB=1	Jump to #Wait label if latch has not occurred
Result=_RLB	Set ‘Result’ equal to the reported position of y axis
Result=	Print result
EN	End

Chapter 7 Application Programming

Overview

The DMC-2x00 provides a powerful programming language that allows users to customize the controller for their particular application. Programs can be downloaded into the DMC-2x00 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-2x00 provides commands that allow the DMC-2x00 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-2x00 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 DOS Editor to Enter Programs (DMC-2000 only)

The DMC-2000 has an internal editor which may be used to create and edit programs in the controller's memory. The internal editor is opened by the command ED. Note that the command ED will not open the internal editor if is sued from Galil's Window based software - in this case, a Windows based editor will be automatically opened. The Windows based editor provides much more functionality and ease-of-use, therefore, the internal editor is most useful when using a simple terminal with the controller and a Windows based editor is not available.

Once the ED command has been given, 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.

NOTE: ED command only accepts a parameter (such as #BEGIN) in DOS Window. For general purposes, the editing features in this section are not applicable when not in DOS mode.

Instruction	Interpretation
: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 labelled 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-2000 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

NOTE: Editor is not available for DMC-2100, however, any terminal may be used (ie. Telnet)

Program Format

A DMC program consists of DMC-2x00 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-2x00 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-2x00 programs must begin with a label and end with an End (EN) statement. Labels start with the pound (#) sign followed by a maximum of seven characters. The first character must be a letter; after that, numbers are permitted. Spaces are not permitted.

The maximum number of labels which may be defined is 510, for firmware 1.0c and higher.

Valid labels

#BEGIN

#SQUARE

#X1

#BEGIN1

Invalid labels

#1Square

#123

Example

Instruction	Interpretation
#START	Beginning of the Program
PR 10000,20000	Specify relative distances on A and B axes
BG AB	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 A and B 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-2x00 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-2x00 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 on page 123.

#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
#COMINT	Label for communication interrupt on the aux. serial port
#TCPERR	Label for TCP/IP communication error (2100 and 2200 only)

Commenting Programs

There are two methods for commenting programs. The first method uses the NO command and allows for programs to be embedded into Galil programs. The second method used the REM statement and requires the use of Galil software.

NO Command

The DMC-2x00 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:

Instruction	Interpretation
#PATH	Label
NO 2-D CIRCULAR PATH	Comment - No Operation
VMAB	Vector Mode
NO VECTOR MOTION ON A AND B	Comment - No Operation
VS 10000	Vector Speed
NO VECTOR SPEED IS 10000	Comment - No Operation
VP -4000,0	Vector Position
NO BOTTOM LINE	Comment - No Operation
CR 1500,270,-180	Circle Motion
NO HALF CIRCLE MOTION	Comment - No Operation
VP 0,3000	Vector Position
NO TOP LINE	Comment - No Operation
CR 1500,90,-180	Circle
NO HALF CIRCLE MOTION	Comment - No Operation
VE	Vector End
NO END VECTOR SEQUENCE	Comment - No Operation
BGS	Begin Sequence
NO BEGIN SEQUENCE MOTION	Comment - No Operation
EN	End of Program
NO END OF PROGRAM	Comment - No Operation

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

HINT: *Some users annotate their programs using the word "NOTE:"; everything after the "NO" is a comment.*

REM Command

If you are using Galil software to communicate with the DMC-2x00 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
VMAB
REM VECTOR MOTION ON A AND B
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-2x00 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 automatic subroutines are implemented for limit switches, position errors or command errors, they are executed in 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.

Instruction	Interpretation
#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, 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 (ie. Thread 0). #TASK1 is executed within TASK2.

Debugging Programs

The DMC-2x00 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 (DMC-2100/2200 only)

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.

Error Code Command

When there is a program error, the DMC-2x00 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-2x00 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-2x10 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 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

_FLa contains the state of the forward limit switch for the 'a' axis

_RLa contains the state of the reverse limit switch for the 'a' axis

Example

The following program has an error. It attempts to specify a relative movement while the A-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:

Instruction	Interpretation
:ED	Edit Mode
000 #A	Program Label
001 PR1000	Position Relative 1000
002 BGA	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;BGA	Add After Motion Done
<cntrl> Q	Quit Edit Mode
:XQ #A	Execute #A

Program Flow Commands

The DMC-2x00 provides instructions to control program flow. The DMC-2x00 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-2x00 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-2x00 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-2x00 can make decisions based on its own status or external events without intervention from a host computer.

NOTE: Not recommended for DMC-2100/2200. The buffer becomes filled easily when using event triggers which would halt communications between the host and the controller.

DMC-2x00 Event Triggers

Command	Function
AM A B C D E FG H or S	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 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 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 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 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 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 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 A,B,C,D sets timeout to declare an error if not in position. If timeout occurs, then the trippoint will clear and the stopcode 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-2x10, 2x20, 2x30, 2x40. n=1 through 24 for DMC-2x50, 2x60, 2x70, 2x80. n=1 through 80 for DMC-2x80.
AS 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.

Example- 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.

Instruction	Interpretation
#TWOMOVE	Label
PR 2000	Position Command
BGA	Begin Motion
AMA	Wait for Motion Complete
PR 4000	Next Position Move
BGA	Begin 2nd move
EN	End program

Example- 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.

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

Example- 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.

Instruction	Interpretation
#TRIP	Label
JG 50000	Specify Jog Speed
BGA;n=0	Begin Motion
#REPEAT	# Repeat Loop
AR 10000	Wait 10000 counts
TPA	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
STA	Stop
EN	End

Example - 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.

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

Example - Set Output when At Speed

Instruction	Interpretation
#ATSPEED	Program Label
JG 50000	Specify jog speed
AC 10000	Acceleration rate
BGA	Begin motion
ASA	Wait for at slew speed 50000
SB1	Set output 1
EN	End program

Example - Change Speed along Vector Path

The following program changes the feedrate 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.

Instruction	Interpretation
#VECTOR	Label
VMAB;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

Example - Multiple Move with Wait

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

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

Example- 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.

Instruction	Interpretation
#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-2x00 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-2x00 to make decisions without a host computer. For example, the DMC-2x00 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-2x00 numeric operand, including variables, array elements, numeric values, functions, keywords, and arithmetic expressions. If no conditional statement is given, the jump will always occur.

Number	V1=6
Numeric Expression	V1=V7*6
	@ABS[V1]>10
Array Element	V1<Count[2]
Variable	V1<V2
Internal Variable	_TPA=0
	_TVA>500
I/O	V1>@AN[2]
	@IN[1]=0

Multiple Conditional Statements

The DMC-2x00 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 paranthesis for proper evaluation by the controller. In addition, the DMC-2x00 executes operations from left to right. For further information on Mathematical Expressions and the bit-wise operators '&' and '|', see pg 7- 128.

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.

Examples

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.

Instruction	Interpretation
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

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

Instruction	Interpretation
#BEGIN	Begin Program
count=10	Initialize loop counter
#LOOP	Begin loop
PA 1000	Position absolute 1000
BGA	Begin move
AMA	Wait for motion complete
WT 100	Wait 100 msec
PA 0	Position absolute 0
BGA	Begin move
AMA	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

If, Else, and Endif

The DMC-2x00 provides a structured approach to conditional statements using IF, ELSE and ENDIF commands.

Using the IF and ENDIF Commands

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

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

Using the ELSE Command

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

Nesting IF Conditional Statements

The DMC-2x00 allows for IF conditional statements to be included within other IF conditional statements. This technique is known as 'nesting' and the DMC-2x00 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.

Instruction	Interpretation
#TEST	Begin Main Program "TEST"
II,,3	Enable 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 executed if 1st IF conditional true
MG "INPUT 1 AND INPUT 2 ARE ACTIVE"	Message executed if 2nd IF is true

ELSE	ELSE command for 2nd IF statement
MG "ONLY INPUT 1 IS ACTIVE	Message executed if 2nd IF is false
ENDIF	End of 2nd conditional statement
ELSE	ELSE command for 1st IF statement
MG"ONLY INPUT 2 IS ACTIVE"	Message executed if 1st IF 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 Input 1& 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.

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.

Instruction	Interpretation
#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;BGA	Define A,B; Begin A
AMA;BGB;AMB	After motion on A, Begin B
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-2x00 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-2x00 program sequences. The DMC-2x00 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 (see page 122 and 158)
#ININT	Input specified by II goes low (see page 123)
#POSERR	Position error exceeds limit specified by ER (see page 122 & 157)
#MCTIME	Motion Complete timeout occurred. Timeout period set by TW command (see page 123)
#CMDERR	Bad command given (see page 124)
#COMINT	Communication Interrupt Routine (see page 125, 135 and 136)
#TCPERR	TCP/IP communication error (2100 and 2200 only)

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.

NOTE: Communication interrupt does not apply to DMC-2100.

Example - Limit Switch:

This program prints a message upon the occurrence of a limit switch. Note, for the #LIMSWI routine to function, the DMC-2x00 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.

Instruction	Interpretation
: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

:BGA	Begin Motion

Now, when a forward limit switch occurs on the A 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

Instruction	Interpretation
:ED	Edit Mode
000 #LOOP	Dummy Program
001 JP #LOOP;EN	Loop
002 #POSERR	Position Error Routine
003 v1=_TEA	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, when a forward limit switch occurs on the A 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 - Input Interrupt

Instruction	Interpretation
#A	Label
II1	Input Interrupt on 1
JG 30000,,,60000	Jog
BGAD	Begin Motion
#LOOP;JP#LOOP;EN	Loop
#ININT	Input Interrupt
STAD;AM	Stop Motion
#TEST;JP #TEST, @IN[1]=0	Test for Input 1 still low
JG 30000,,,6000	Restore Velocities
BGAD	Begin motion
RI0	Return from interrupt routine to Main Program and do not re-enable trippoints

Example - Motion Complete Timeout

Instruction	Interpretation
#BEGIN	Begin main program
TW 1000	Set the time out to 1000 ms
PA 10000	Position Absolute command
BGA	Begin motion
MCA	Motion Complete trip point
EN	End main program
#MCTIME	Motion Complete Subroutine
MG “A fell short”	Send out a message
EN	End subroutine

This simple program will issue the message “A fell short” if the A axis does not reach the commanded position within 1 second of the end of the profiled move.

Example - Command Error

Instruction	Interpretation
#BEGIN	Begin main program
IN "ENTER SPEED", speed	Prompt for speed
JG speed;BGA	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

Instruction	Interpretation
#A	Begin thread 0 (continuous loop)
JP#A	
EN	End of thread 0
#B	Begin thread 1
n=-1	Create new variable
KP n	Set KP to value of N, an invalid value
TY	Issue invalid command
EN	End of thread 1
#CMDERR	Begin command error subroutine
IF _TC=6	If error is out of range (KP -1)
N=1	Set N to a valid number
XQ _ED2,_ED1,1	Retry KP N command
ENDIF	
IF _TC=1	If error is invalid command (TY)
XQ _ED3,_ED1,1	Skip invalid command
ENDIF	
EN	End of command error routine

Example - Communication Interrupt

A DMC-2x10 is used to move the A axis back and forth from 0 to 10000. This motion can be paused, resumed and stopped via input from an auxiliary port terminal.

Instruction	Interpretation
#BEGIN	Label for beginning of program
CC 9600,0,0,0	Setup communication configuration for auxiliary serial port
CI 2	Setup communication interrupt for auxiliary serial port
MG {P2}"Type 0 to stop motion"	Message out of auxiliary port
MG {P2}"Type 1 to pause motion"	Message out of auxiliary port
MG {P2}"Type 2 to resume motion"	Message out of auxiliary port
rate=2000	Variable to remember speed
SPA=rate	Set speed of A axis motion
#LOOP	Label for Loop
PAA=10000	Move to absolute position 10000
BGA	Begin Motion on A axis
AMA	Wait for motion to be complete
PAA=0	Move to absolute position 0

BGA	Begin Motion on A axis
AMA	Wait for motion to be complete
JP #LOOP	Continually loop to make back and forth motion
EN	End main program
#COMINT	Interrupt Routine
JP #STOP,P2CH="0"	Check for S (stop motion)
JP #PAUSE,P2CH="1"	Check for P (pause motion)
JP #RESUME,P2CH="2"	Check for R (resume motion)
EN1,1	Do nothing
#STOP	Routine for stopping motion
STA;ZS;EN	Stop motion on A axis; Zero program stack; End Program
#PAUSE	Routine for pausing motion
RATE=_SPA	Save current speed setting of A axis motion
SPA=0	Set speed of A axis to zero (allows for pause)
EN1,1	Re-enable trip-point and communication interrupt
#RESUME	Routine for resuming motion
SPA=RATE	Set speed on A axis to original speed
EN1,1	Re-enable trip-point and communication interrupt

For additional information, see section on page.

Example – Ethernet Communication Error

This simple program executes in the DMC-2100/2200 and indicates (via the serial port) when a communication handle fails. By monitoring the serial port, the user can re-establish communication if needed.

Instruction	**Interpretation**
#LOOP	Simple program loop
JP#LOOP	
EN	
#TCPERR	Ethernet communication error auto routine
MG {P1}_IA4	Send message to serial port indicating which handle did not receive proper acknowledgment.
RE	

Mathematical and Functional Expressions

Mathematical Operators

For manipulation of data, the DMC-2x00 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.

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=_TPA-(@COS[45]*40)	Puts the position of A - 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-2x00 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:

Instruction	Interpretation
#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, resolution of 1/64000 degrees, max +/- 4 billion)
@COS[n]	Cosine of n (n in degrees, resolution of 1/64000 degrees, max +/- 4 billion)
@TAN[n]	Tangent of n (n in degrees, resolution of 1/64000 degrees, max ±4 billion)
@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]	2's Compliment 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.

Instruction	Interpretation
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-2x00 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.

Instruction	Interpretation
PR posa	Assigns variable posa to PR command
JG rpmb*70	Assigns variable rpmb multiplied by 70 to JG command.

Programmable Variables

The DMC-2x00 allows the user to create up to 254 variables. Each variable is defines 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 can be upper or lowercase, or any combination. Variables are case sensitive; SPEEDC?speedC. Variable names should not be the same as DMC-2x00 instructions. For example, PR is not a good choice for a variable name.

Examples of valid and invalid variable names are:

Valid Variable Names

```
POSA
pos1
speedC
```

Invalid Variable Names

```
REALLONGNAME      ; Cannot have more than 8 characters
123               ; Cannot begin variable name with a number
SPEED C           ; 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-2x00 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.

Instruction	Interpretation
posA=_TPA	Assigns returned value from TPA command to variable posA
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 GN or PR.

```
PR v1              Assign v1 to PR command
SP _VSS*2000       Assign _VSS*2000 to SP command
```

Displaying the value of variables at the terminal

Variables may be sent to the screen using the format, variable=. For example, v1= , returns the value of the variable v1.

Example - Using Variables for Joystick

The example below reads the voltage of an A-B joystick and assigns it to variables VA and VB to drive the motors at proportional velocities, where

10 Volts = 3000 rpm = 200000 c/sec

Speed/Analog input = 200000/10 = 20000

Instruction	Interpretation
#JOYSTIK	Label
JG 0,0	Set in Jog mode
BGAB	Begin Motion
#LOOP	Loop
va=@AN[1]*20000	Read joystick A
vb=@AN[2]*20000	Read joystick B
JG va,vb	Jog at variable va,vb
JP#LOOP	Repeat
EN	End

Operands

Operands allow motion or status parameters of the DMC-2x00 to be incorporated into programmable variables and expressions. Most DMC-2x00 commands have an equivalent operand - which are designated by adding an underscore (_) prior to the DMC-2x00 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-2x00 registers. The axis designation is required following the command.

Instruction	Interpretation
posA=_TPA	Assigns value from Tell Position A to the variable posA.
JP #LOOP,_TEA>5	Jump to #LOOP if the position error of A 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: _TPA=2 is invalid.

Special Operands (Keywords)

The DMC-2x00 provides a few additional operands which give access to internal variables that are not accessible by standard DMC-2x00 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)

LRn	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.

v1=_LFA	Assign v1 the state of the Forward Limit Switch on the A-axis
v3=TIME	Assign v3 the current value of the time clock
v4=_HMD	Assign v4 the logical state of the Home input on the D-axis

Arrays

For storing and collecting numerical data, the DMC-2x00 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 [ ].

DM posA[7]	Defines an array names posA with seven entries
DM speed[100]	Defines an array named speed with 100 entries
DM posA[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 posA array (defined with the DM command, DM posA[7]) would be specified as posA[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.

DM speed[10]	Dimension Speed Array
speed[1]=7650.2	Assigns the first element of the array the value 7650.2
speed[1]=	Returns array element value
posXA[10]=_TPA	Assigns the 10th element the position of A
con[2]=@COS[POS]*2	Assigns the 2nd element of the array the cosine of POS * 2.
timer[1]=TIME	Assigns the 1st element of the array TIME

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.

Instruction	Interpretation
#A	Begin Program
coimt=0;DM POS[10]	Initialize counter and define array
#LOOP	Begin loop
WT 10	Wait 10 msec
POS[count]=_TPA	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 seperated 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-2x00 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 be 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
_DEA	2nd encoder position (dual encoder)
_TPA	Encoder position
_TEA	Position error
_SHA	Commanded position
_RLA	Latched position
_TI	Inputs
_OP	Output
_TSA	Switches (only bit 0-4 valid)
_SCA	Stop code
_NOA	Status bits
_TTA	Torque (reports digital value +/-32544)
_AFA	Analog Input (Letter corresponds to input, eg. AFA = 1st Analog In, AFB=2nd Analog In.)

NOTE: A may be replaced by B,C,D,E,F,G, or H 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

Instruction	Interpretation
#RECORD	Begin program
DM apos[300],bpos[300]	Define A,B position arrays
DM aerr[300],berr[300]	Define A,B error arrays
RA apos [],aerr[],bpos[],berr[]	Select arrays for capture
RD _TPA,_TEA,_TPB,_TEB	Select data types

PR 10000,20000	Specify move distance
RC1	Start recording now, at rate of 2 msec
BG AB	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
apos [n]=	Print X position
bpos [n]=	Print Y position
aerr[n]=	Print X error
berr[n]=	Print Y error
n=n+1	Increment Counter
#DONE	Done
EN	End Program

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.

Example- Inputting Numeric Data

```
#A
IN "Enter Length",lenA
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, lenA. (**NOTE:** Do not include a space between the comma at the end of the input message and the variable name.)

Example- Cut-to-Length

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.

Instruction	Interpretation
#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
BGA	Begin motion to move material
AMA	Wait for motion done
SB1	Set output to cut
WT100;CB1	Wait 100 msec, then turn off cutter
JP #CUT	Repeat process
EN	End program

Operator Data Entry Mode

The Operator Data Entry Mode provides for un-buffered data entry through the auxiliary RS-232 port. In this mode, the DMC-2x00 provides a buffer for receiving characters. This mode may only be used when executing an applications program.

The Operator Data Entry Mode may be specified for Port 2 only. This mode may be exited with the \ or <escape> key.

NOTE: Operator Data Entry Mode cannot be used for high rate data transfer.

Set the third field of the CC command to zero to set the Operator Data Entry Mode.

To capture and decode characters in the Operator Data Mode, the DMC-2x00 provides special the following keywords:

Keyword	Function
P2CH	Contains the last character received
P2ST	Contains the received string
P2NM	Contains the received number
P2CD	Contains the status code: -1 mode disabled 0 nothing received 1 received character, but not <enter> 2 received string, not a number 3 received number

NOTE: The value of P2CD returns to zero after the corresponding string or number is read.

These keywords may be used in an applications program to decode data and they may also be used in conditional statements with logical operators.

Example

Instruction	Interpretation
JP #LOOP,P2CD< >3	Checks to see if status code is 3 (number received)
JP #P,P1CH="V"	Checks if last character received was a V
PR P2NM	Assigns received number to position
JS #XAXIS,P1ST="X"	Checks to see if received string is X

Using Communication Interrupt (For DMC-2000 only)

The DMC-2x00 provides a special interrupt for communication allowing the application program to be interrupted by input from the user. The interrupt is enabled using the CI command. The syntax for the command is CI n:

n = 0	Don't interrupt Port 2
n = 1	Interrupt on <enter> Port 2
n = 2	Interrupt on any character Port 2
n = -1	Clear any characters in buffer

The #COMINT label is used for the communication interrupt. For example, the DMC-2x00 can be configured to interrupt on any character received on Port 2. The #COMINT subroutine is entered when a character is received and the subroutine can decode the characters. At the end of the routine the EN command is used. EN,1 will re-enable the interrupt and return to the line of the program where the interrupt was called, EN will just return to the line of the program where it was called without re-enabling the interrupt. As with any automatic subroutine, a program must be running in thread 0 at all times for it to be enabled.

Example

A DMC-2x00 is used to jog the A and B axis. This program automatically begins upon power-up and allows the user to input values from the main serial port terminal. The speed of either axis may be changed during motion by specifying the axis letter followed by the new speed value. An S stops motion on both axes.

Instruction	Interpretation
#AUTO	Label for Auto Execute
speedA=10000	Initial A speed
speedB=10000	Initial B speed
CI 2	Set Port 2 for Character Interrupt
JG speedA, speedB	Specify jog mode speed for A and B axis
BGXY	Begin motion
#PRINT	Routine to print message to terminal
MG{P2}"TO CHANGE SPEEDS"	Print message
MG{P2}"TYPE A OR B"	
MG{P2}"TYPE S TO STOP"	
#JOGLOOP	Loop to change Jog speeds
JG speedA, speedB	Set new jog speed
JP #JOGLOOP	
EN	End of main program

```
#COMINT                          Interrupt routine
JP #A,P2CH="A"                   Check for A
JP #B,P2CH="B"                   Check for B
JP #C,P2CH="S"                   Check for S
ZS1;CI2;JP#JOGLOOP               Jump if not X,Y,S
#A;JS#NUM
speedX=val                       New X speed
ZS1;CI2;JP#PRINT                 Jump to Print
#B;JS#NUM
speedY=val                       New Y speed
ZS1;CI2;JP#PRINT                 Jump to Print
#C;ST;AMX;CI-1                   Stop motion on S
MG{^8}, "THE END"
ZS;EN,1                          End-Re-enable interrupt
#NUM                             Routine for entering new jog speed
MG "ENTER",P2CH{S},"AXIS SPEED"  Prompt for value
{N}
#NUMLOOP; CI-1                   Check for enter
#NMLP                            Routine to check input from terminal
JP #NMLP,P2CD<2                  Jump to error if string
JP #ERROR,P2CD=2                 Read value
val=P2NM
EN                               End subroutine
#ERROR;CI-1                      Error Routine
MG "INVALID-TRY AGAIN"           Error message
JP #NMLP
EN                               End
```

Inputting String Variables

String variables with up to six characters may be input using the specifier, {Sn} where n represents the number of string characters to be input. If n is not specified, six characters will be accepted. For example, IN "Enter A,B or C", V{S} specifies a string variable to be input.

The DMC-2x00, stores all variables as 6 bytes of information. When a variable is specified as a number, the value of the variable is represented as 4 bytes of integer and 2 bytes of fraction. When a variable is specified as a string, the variable can hold up to 6 characters (each ASCII character is 1 byte). When using the IN command for string input, the first input character will be placed in the top byte of the variable and the last character will be placed in the lowest significant byte of the fraction. The characters can be individually separated by using bit-wise operations, see section Bit-wise Operators.

Output of Data (Numeric and String)

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

Sending Messages

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

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

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

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

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

Specifying the Port for Messages:

By default, messages will be sent through the port specified by the USB/Ethernet Dip Switch - the state of this switch upon power up will determine if messages will be sent to USB port (DMC-2000), or Ethernet (DMC-2100/2200) the Main Serial Port. However, the port can be specified with the specifier, {P1} for the main serial port {P2} for auxiliary serial port, {U} for the USB port , or {E} for the Ethernet port.

MG {P2} "Hello World" Sends message to Auxiliary Port

Formatting Messages

String variables can be formatted using the specifier, {Sn} where n is the number of characters, 1 thru 6. For example:

```
MG STR {S3}
```

This statement returns 3 characters of the string variable named STR.

Numeric data may be formatted using the {Fn.m} expression following the completed MG statement. {$n.m} formats data in HEX instead of decimal. The actual numerical value will be formatted with n characters to the left of the decimal and m characters to the right of the decimal. Leading zeros will be used to display specified format.

For example::

```
MG "The Final Value is", result {F5.2}
```

If the value of the variable result is equal to 4.1, this statement returns the following:

```
The Final Value is 00004.10
```

If the value of the variable result is equal to 999999.999, the above message statement returns the following:

```
The Final Value is 99999.99
```

The message command normally sends a carriage return and line feed following the statement. The carriage return and the line feed may be suppressed by sending {N} at the end of the statement. This is useful when a text string needs to surround a numeric value.

Example:

```
#A
JG 50000;BGA;ASA
MG "The Speed is", _TVA {F5.1} {N}
MG "counts/sec"
```

```
EN
```

When #A is executed, the above example will appear on the screen as:

```
The speed is 50000 counts/sec
```

Using the MG Command to Configure Terminals

The MG command can be used to configure a terminal. Any ASCII character can be sent by using the format {^n} where n is any integer between 1 and 255.

Example:

```
MG {^07} {^255}
```

sends the ASCII characters represented by 7 and 255 to the bus.

Summary of Message Functions

function	description
" "	Surrounds text string
{Fn.m}	Formats numeric values in decimal n digits to the left of the decimal point and m digits to the right
{P1}, {P2}, {U} or {E}	Send message to Main Serial Port, Auxiliary Serial Port, USB Port or Ethernet Port
{$n.m}	Formats numeric values in hexadecimal
{^n}	Sends ASCII character specified by integer n
{N}	Suppresses carriage return/line feed
{Sn}	Sends the first n characters of a string variable, where n is 1 thru 6.

Displaying Variables and Arrays

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

Example - Printing a Variable and an Array element

Instruction	Interpretation
#DISPLAY	Label
DM posA[7]	Define Array POSA with 7 entries
PR 1000	Position Command
BGX	Begin
AMX	After Motion
v1=_TPA	Assign Variable v1
posA[1]=_TPA	Assign the first entry
v1=	Print v1

Interrogation Commands

The DMC-2x00 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 ch 5.

Using the PF Command to Format Response from Interrogation Commands

The command, PF, can change format of the values returned by theses interrogation commands:

BL ?	LE ?
DE ?	PA ?
DP ?	PR ?
EM ?	TN ?
FL ?	VE ?
IP ?	TE
TP	

The numeric values may be formatted in decimal or hexadecimal* with a specified number of digits to the right and left of the decimal point using the PF command.

Position Format is specified by:

PF m.n

where m is the number of digits to the left of the decimal point (0 thru 10) and n is the number of digits to the right of the decimal point (0 thru 4) A negative sign for m specifies hexadecimal format.

Hex values are returned preceded by a $ and in 2's complement. Hex values should be input as signed 2's complement, where negative numbers have a negative sign. The default format is PF 10.0.

If the number of decimal places specified by PF is less than the actual value, a nine appears in all the decimal places.

Example

Instruction	Interpretation
:DP21	Define position
:TPA	Tell position
0000000021	Default format
:PF4	Change format to 4 places
:TPA	Tell position
0021	New format
:PF-4	Change to hexadecimal format
:TPA	Tell Position
$0015	Hexadecimal value
:PF2	Format 2 places
:TPA	Tell Position
99	Returns 99 if position greater than 99

Removing Leading Zeros from Response to Interrogation Commands

The leading zeros on data returned as a response to interrogation commands can be removed by the use of the command, LZ.

LZ0	Disables the LZ function
TP	Tell Position Interrogation Command
-0000000009, 0000000005	Response (With Leading Zeros)

LZ1	Enables the LZ function
TP	Tell Position Interrogation Command
-9, 5	Response (Without Leading Zeros)

Local Formatting of Response of Interrogation Commands

The response of interrogation commands may be formatted locally. To format locally, use the command, {Fn.m} or {$n.m} on the same line as the interrogation command. The symbol F specifies that the response should be returned in decimal format and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal.

TP {F2.2}	Tell Position in decimal format 2.2
-05.00, 05.00, 00.00, 07.00	Response from Interrogation Command
TP {$4.2}	Tell Position in hexadecimal format 4.2
FFFB.00,$0005.00,$0000.00,$0007.00	Response from Interrogation Command

Formatting Variables and Array Elements

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

VF m.n

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

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

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

Instruction	Interpretation
v1=10	Assign v1
v1=	Return v1
:0000000010.0000	Response - Default format
VF2.2	Change format
v1=	Return v1
:10.00	Response - New format
vF-2.2	Specify hex format
v1=	Return v1
$0A.00	Response - Hex value
VF1	Change format
v1=	Return v1
:9	Response - Overflow

Local Formatting of Variables

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

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

The local format is also used with the MG* command.

Converting to User Units

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

The DMC-2x00 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.

Instruction	Interpretation
#RUN	Label
IN "ENTER # OF REVOLUTIONS",n1	Prompt for revs
PR n1*2000	Convert to counts
IN "ENTER SPEED IN RPM",s1	Prompt for RPMs
SP s1*2000/60	Convert to counts/sec
IN "ENTER ACCEL IN RAD/SEC2",a1	Prompt for ACCEL
AC a1*2000/(2*3.14)	Convert to counts/sec2
BG	Begin motion
EN	End program

Hardware I/O

Digital Outputs

The DMC-2x00 has an 8-bit uncommitted output port and an additional 64 I/O which may be configured as inputs or outputs with the CO command for controlling external events. The DMC-2x50 through DMC-2x80 has an additional 8 outputs. Each bit on the output port may be set and cleared with the software instructions SB (Set Bit) and CB(Clear Bit), or OB (define output bit).

Example- Set Bit and Clear Bit

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

Example- Output Bit

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

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

The output port can be set by specifying an 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.

Example- Output Port

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

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

Example - Turn on output after move

Instruction	Interpretation
#OUTPUT	Label
PR 2000	Position Command
BG	Begin
AM	After move
SB1	Set Output 1
WT 1000	Wait 1000 msec
CB1	Clear Output 1
EN	End

Digital Inputs

The general digital inputs for are accessed by using the @IN[n] function or the TI command. The @IN[n] function returns the logic level of the specified input, n, where n is a number 1 through 96..

Example - Using Inputs to control program flow

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

Example - Start Motion on Switch

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

Solution: Connect panel switch to input 1 of DMC-2x00. High on input 1 means switch is in on position.

Instruction	Interpretation
#S;JG 4000	Set speed
AI 1;BGA	Begin after input 1 goes high
AI -1;STA	Stop after input 1 goes low
AMA;JP #S	After motion, repeat
EN;	

The Auxiliary Encoder Inputs

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

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

Example:

A DMC-2x10 has one auxiliary encoder. This encoder has two inputs (channel A and channel B). Channel A input is mapped to input 81 and Channel B input is mapped to input 82. To use this input for 2 TTL signals, the first signal will be connected to AA+ and the second to AB+. AA- and AB- will be left unconnected. To access this input, use the function @IN[81] and @IN[82].

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

Input Interrupt Function

The DMC-2x00 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 command (not EN) to return from the #ININT subroutine.

Example - Input Interrupt

Instruction	Interpretation
#A	Label #A
II 1	Enable input 1 for interrupt function
JG 30000,-20000	Set speeds on A and B axes
BG AB	Begin motion on A and B axes
#B	Label #B
TP AB	Report A and B axes positions
WT 1000	Wait 1000 milliseconds
JP #B	Jump to #B
EN	End of program
#ININT	Interrupt subroutine
MG "Interrupt has occurred"	Displays the message
ST AB	Stops motion on A and B axes
#LOOP;JP #LOOP,@IN[1]=0	Loop until Interrupt cleared
JG 15000,10000	Specify new speeds
WT 300	Wait 300 milliseconds
BG AB	Begin motion on A and B axes
RI	Return from Interrupt subroutine

Analog Inputs

The DMC-2x00 provides eight analog inputs. The value of these inputs in volts may be read using the @AN[n] function where n is the analog input 1 through 8. The resolution of the Analog-to-Digital conversion is 12 bits (16-bit ADC is available as an option). Analog inputs are useful for reading special sensors such as temperature, tension or pressure.

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

Example - Position Follower (Point-to-Point)

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

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

Instruction	Interpretation
#POINTS	Label
SP 7000	Speed
AC 80000;DC 80000	Acceleration
#LOOP	
VP=@AN[1]*1000	Read and analog input, compute position

PA VP	Command position
BGA	Start motion
AMA	After completion
JP #LOOP	Repeat
EN	End

Example - Position Follower (Continuous Move)

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

Instruction	Interpretation
#CONT	Label
AC 80000;DC 80000	Acceleration rate
JG 0	Start job mode
BGX	Start motion
#LOOP	
vp=@AN[1]*1000	Compute desired position
ve=vp-_TPA	Find position error
vel=ve*20	Compute velocity
JG vel	Change velocity
JP #LOOP	Change velocity
EN	End

Extended I/O of the DMC-2x00 Controller

The DMC-2x00 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 1 80 pin high density connector.

Configuring the I/O of the DMC-2x00

The 64 extended I/O points of the DMC-2x00 series controller can be configured in blocks of 8. The extended I/O is denoted as blocks 2-9 or bits 17-80.

The command, CO, is used to configure the extended I/O as inputs or outputs. The CO command has one field:

CO n

where n is a decimal value which represents a binary number. Each bit of the binary number represents one block of extended I/O. When set to 1, the corresponding block is configured as an output.

The least significant bit represents block 2 and the most significant bit represents block 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 is to be configured as an output, CO 12 is issued.

8-Bit I/O Block	Block	Binary Representation	Decimal Value for Block
17-24	2	2^0	1
25-32	3	$2^1$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. Determine which 8-bit I/O blocks to be configured as outputs.

Step 2. From the table, determine the decimal value for each I/O block to be set as an output.

Step 3. Add up all of the values determined in step 2. This is the value to be used for n.

For example, if blocks 2 and 3 are to be outputs, then n is 3 and the command, CO3, should be issued. **NOTE:** This calculation is identical to the formula: $n = n_2 + 2*n_3 + 4*n_4 + 8*n_5 +16* n_6 +32* n_7 +64* n_8 +128* n_9$ 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=1 through 8 and 17 through 80). Outputs may also be defined with the conditional command, OBn (where n=1 through 8 and 17 through 80).

The command, OP, may also be used to set output bits, specified as blocks of data. The OP command accepts 5 parameters. The first parameter sets the values of the main output port of the controller (Outputs 1-8, block 0). The additional parameters set the value of the extended I/O as outlined:

OP m,a,b,c,d

where m is the decimal representation of the bits 1-8 (values from 0 to 255) and a,b,c,d represent the extended I/O in consecutive groups of 16 bits. (values from 0 to 65535). Arguments which are given for I/O points which are configured as inputs will be ignored. The following table describes the arguments used to set the state of outputs.

Argument	Blocks	Bits	Description
m	0	1-8	General Outputs
a	2,3	17-32	Extended I/O
b	4,5	33-48	Extended I/O
c	6,7	49-64	Extended I/O
d	8,9	65-80	Extended I/O

For example, if block 8 is configured as an output, the following command may be issued:

OP 7,,,,7

This command will set bits 1,2,3 (block 0) and bits 65,66,67 (block 8) to 1. Bits 4 through 8 and bits 68 through 80 will be set to 0. All other bits are unaffected.

When accessing I/O blocks configured as inputs, use the TIn command. The argument 'n' refers to the block to be read (n=0,2,3,4,5,6,7,8 or 9). The value returned will be a decimal representation of the corresponding bits.

Individual bits can be queried using the @IN[n] function (where n=1 through 8 or 17 through 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).

Interfacing to Grayhill or OPTO-22 G4PB24

The DMC-2x00 controller uses one 80 Pin high density connector which requires connection to a 80 pin high density cable (Galil CABLE-80). This cable can be converted to 2 50 pin IDC connectors which are compatible with I/O mounting racks such as Grayhill 70GRCM32-HL and OPTO-22 G4PB24. To convert the 80 pin cable, use the CB-50-80 adapter from Galil. The 50 pin ribbon cables which connect to the CB-50-80 connect directly into the 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.

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:

$4000/2\pi = 637$ 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	Interpretation
#A	Label
AI1	Wait for input 1
PR 6370	Distance
SP 3185	Speed
BGA	Start Motion
AMA	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

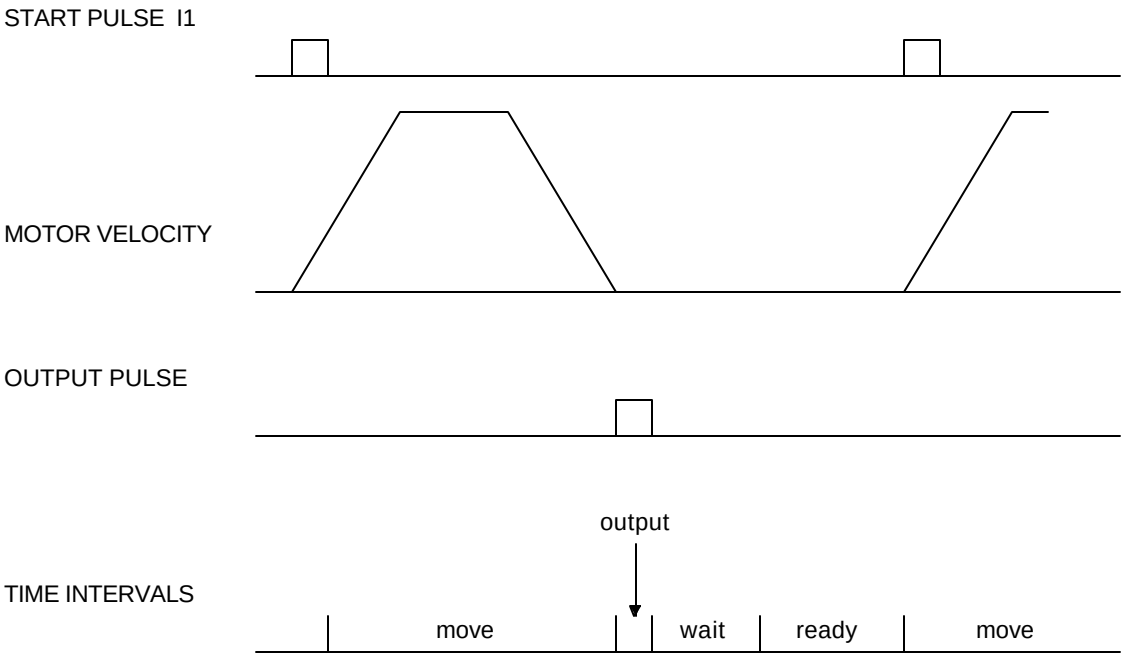


Figure 7.1 - Motor Velocity and the Associated Input/Output signals

A-B Table Controller

An A-B-C system must cut the pattern shown in Fig. 7.2. The A-B table moves the plate while the C-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 feedrate 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 C-axis raised. An A-B motion to point B is followed by lowering the C-axis and performing a cut along the circle. Once the circular motion is completed, the C-axis is raised and the motion continues to point C, etc.

Assume that all of the 3 axes are driven by lead screws with 10 turns-per-inch pitch. Also assume encoder resolution of 1000 lines per revolution. This results in the relationship:

1 inch = 40,000 counts

and the speeds of

1 in/sec = 40,000 count/sec

5 in/sec = 200,000 count/sec

an acceleration rate of 0.1g equals

$0.1g = 38.6 \text{ in/s2} = 1{,}544{,}000 \text{ count/s}^2$

Note that the circular path has a radius of 2" or 80000 counts, and the motion starts at the angle of 270° and traverses 360° in the CW (negative direction). Such a path is specified with the instruction

CR 80000,270,-360

Further assume that the C must move 2" at a linear speed of 2" per second. The required motion is performed by the following instructions:

Instruction	Interpretation
#A	Label
VM AB	Circular interpolation for AB
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 C down
SP,,80000	C speed
BGC	Start C motion
AMC	Wait for completion of C motion
CR 80000,270,-360	Circle
VE	
VS 40000	Feedrate
BGS	Start circular move
AMS	Wait for completion
PR,,80000	Move C up
BGC	Start C move
AMC	Wait for C completion
PR -21600	Move A
SP 20000	Speed A
BGA	Start A
AMA	Wait for A completion
PR,,-80000	Lower C
BGC	
AMC	
CR 80000,270,-360	C second circle move
VE	
VS 40000	
BGS	
AMS	

```
PR,,80000              Raise C
BGC
AMC
VP -37600,-16000       Return AB to start
VE
VS 200000
BGS
AMS
EN
```

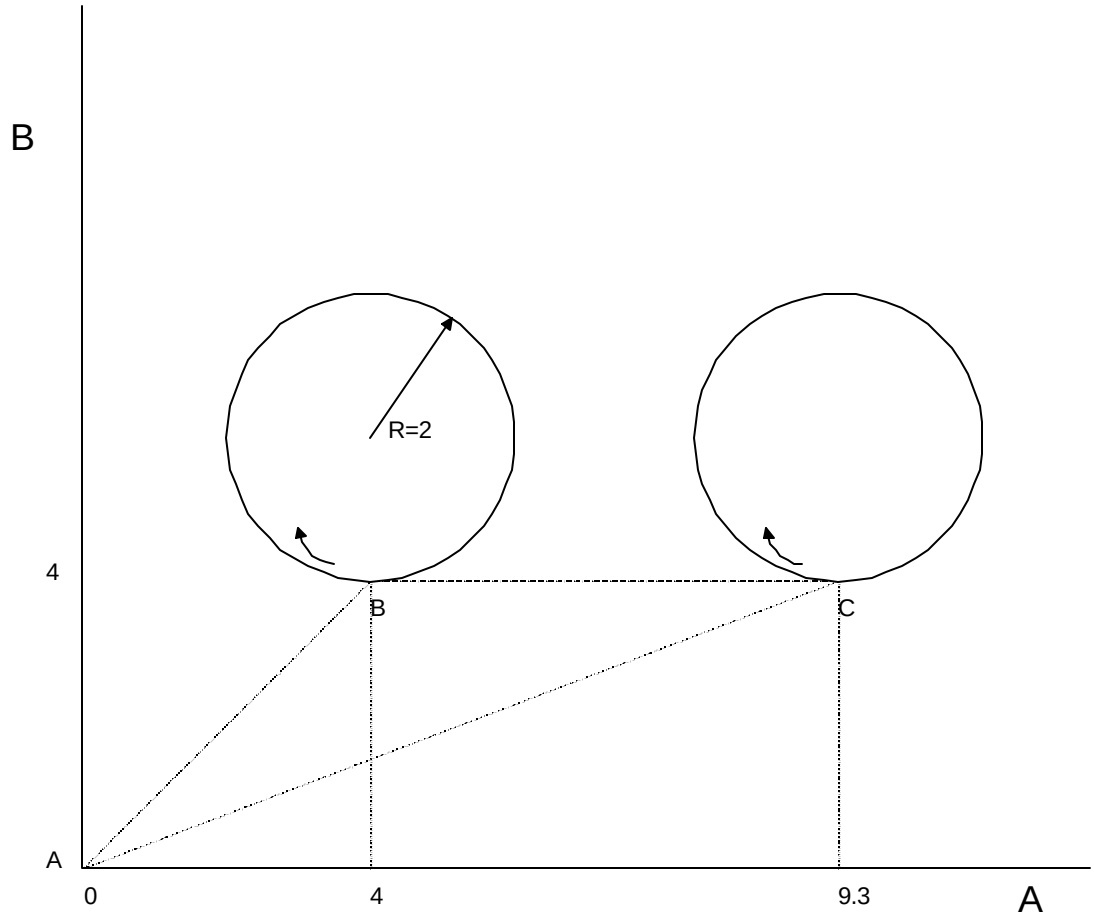


Figure 7.2 - Motor Velocity and the Associated Input/Output signals

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
BGA
#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 1024 counts, the required motor position must be 5120 counts. The variable V3 changes the position ratio.

Instruction	Interpretation
#A	Label
v3=1024	Initial position ratio
DP0	Define the starting position
JG0	Set motor in jog mode as zero
BGA	Start
#B	
v1=@AN[1]	Read analog input
v2=v1*v3	Compute the desired position
v4=v2-_TPA-_TEA	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 leadscrew. Such a leadscrew 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.

Instruction	Interpretation
#A	Label
DP0	Define starting positions as zero
linpos=0	
PR 1000	Required distance
BGA	Start motion
#B	
AMA	Wait for completion
WT 50	Wait 50 msec
linpos = _DEA	Read linear position
er=1000- linpos -_TEA	Find the correction
JP #C,@ABS[er]<2	Exit if error is small
PR er	Command correction
BGA	
JP #B	Repeat the process
#C	
EN	

THIS PAGE LEFT BLANK INTENTIONALLY

Chapter 8 Hardware & Software Protection

Introduction

The DMC-2x00 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-2x00 is an integral part of the machine, the engineer should design his overall system with protection against a possible component failure on the dmc-2x00. Galil shall not be liable or responsible for any incidental or consequential damages.

Hardware Protection

The DMC-2x00 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-2900 interface board. To make these changes, see section entitled 'Amplifier Interface' pg 3-25.

Error Output - The error output is a TTL signal which indicates on error condition in the controller. This signal is available on the interconnect module as ERROR. When the error signal is low, this indicates on of the following error conditions.

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

Input Protection Lines

General Abort - A low input stops commanded motion instantly without a controlled deceleration. For any axis in which the Off-On-Error function is enabled, the amplifiers will be disabled. This could cause the motor to 'coast' to a stop. If the Off-On-Error function is

not enabled, the motor will instantaneously stop and servo at the current position. The Off-On-Error function is further discussed in this chapter.

Selective Abort - The controller can be configured to provide an individual abort for each axis. Activation of the selective abort signal will act the same as the Abort Input but only on the specific axis. To configure the controller for selective abort, issue the command CN,,,1. This configures the inputs 5,6,7,8,13,14,15,16 to act as selective aborts for axes A,B,C,D,E,F,G,H respectively.

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-2x00 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.

ER 200,300,400,500	Set A-axis error limit for 200, B-axis error limit to 300, C-axis error limit to 400 counts, D-axis error limit to 500 counts
ER,1,,10	Set B-axis error limit to 1 count, set D-axis error limit to 10 counts.

The units of the error limit are quadrature counts. The error is the difference between the command position and actual encoder position. If the absolute value of the error exceeds the value specified by ER, the DMC-2x00 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 A,B,C and D can be monitored during execution using the TE command.

Programmable Position Limits

The DMC-2x00 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-2x00 will not accept position commands beyond the limit. Motion beyond the limit is also prevented.

Example

Instruction	Interpretation

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

(motion stops at forward limits)

Off-On-Error

The DMC-2x00 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 A,B,C and D 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.

Example

OE 1,1,1,1	Enable off-on-error for A,B,C and D
OE 0,1,0,1	Enable off-on-error for B and D axes, Disable off-on-error for A and C

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

Instruction	Interpretation
#A;JP #A;EN	"Dummy" program
#POSERR	Start error routine on error
MG "error"	Send message
SB 1	Fire relay
STA	Stop motor
AMA	After motor stops
SHA	Servo motor here to clear error
RE	Return to main program

NOTE: An applications program must be executing for the #POSERR routine to function.

Limit Switch Routine

The DMC-2x00 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. A,B,C, or D following LR or LF specifies the axis. The CN command can be used to configure the polarity of the limit switches.

Example

Instruction	Interpretation
#A;JP #A;EN	Dummy Program
#LIMSWI	Limit Switch Utility
v1=_LFA	Check if forward limit
v2=_LRA	Check if reverse limit
JP#LF,v1=0	Jump to #LF if forward
JP#LR,v2=0	Jump to #LR if reverse
JP#END	Jump to end
#LF	#LF
MG "FORWARD LIMIT"	Send message
STX;AMA	Stop motion
PR-1000;BGA;AMA	Move in reverse
JP#END	End
#LR	#LR
MG "REVERSE LIMIT"	Send message
STX;AMA	Stop motion
PR1000;BGA;AMA	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 terminal emulator, cannot communicate with controller.	Selected comm. port incorrect	Try another comport
Same as above	Selected baud rate incorrect	Check to be sure that baud rate same as dip switch settings on controller, change as necessary.

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.

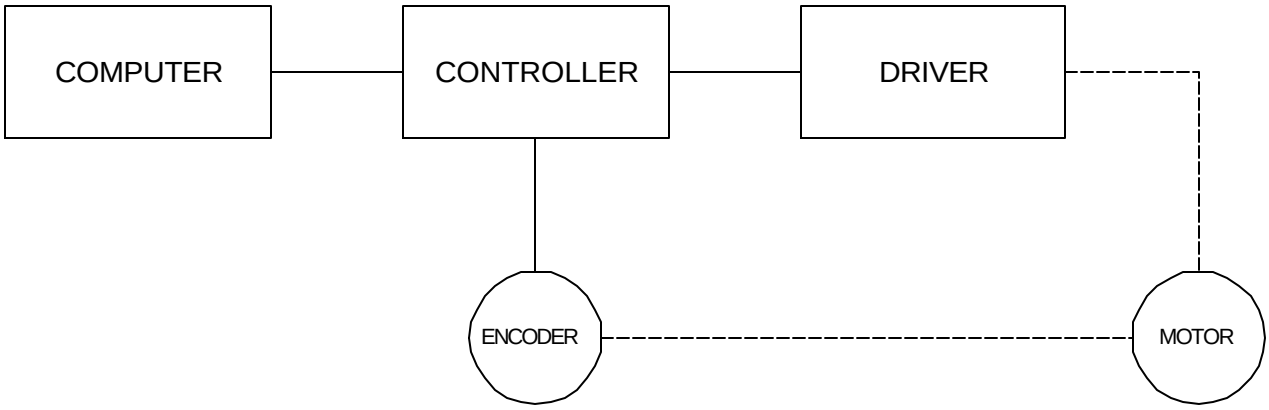


Figure 10.1 - Elements of Servo Systems

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.

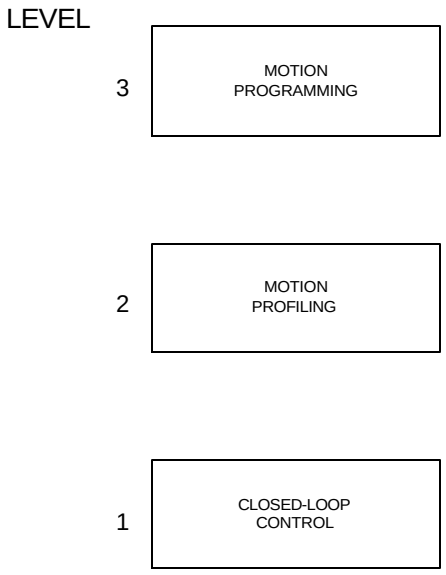


Figure 10.2 - Levels of Control Functions

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 A
AD 2000
BG B
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.

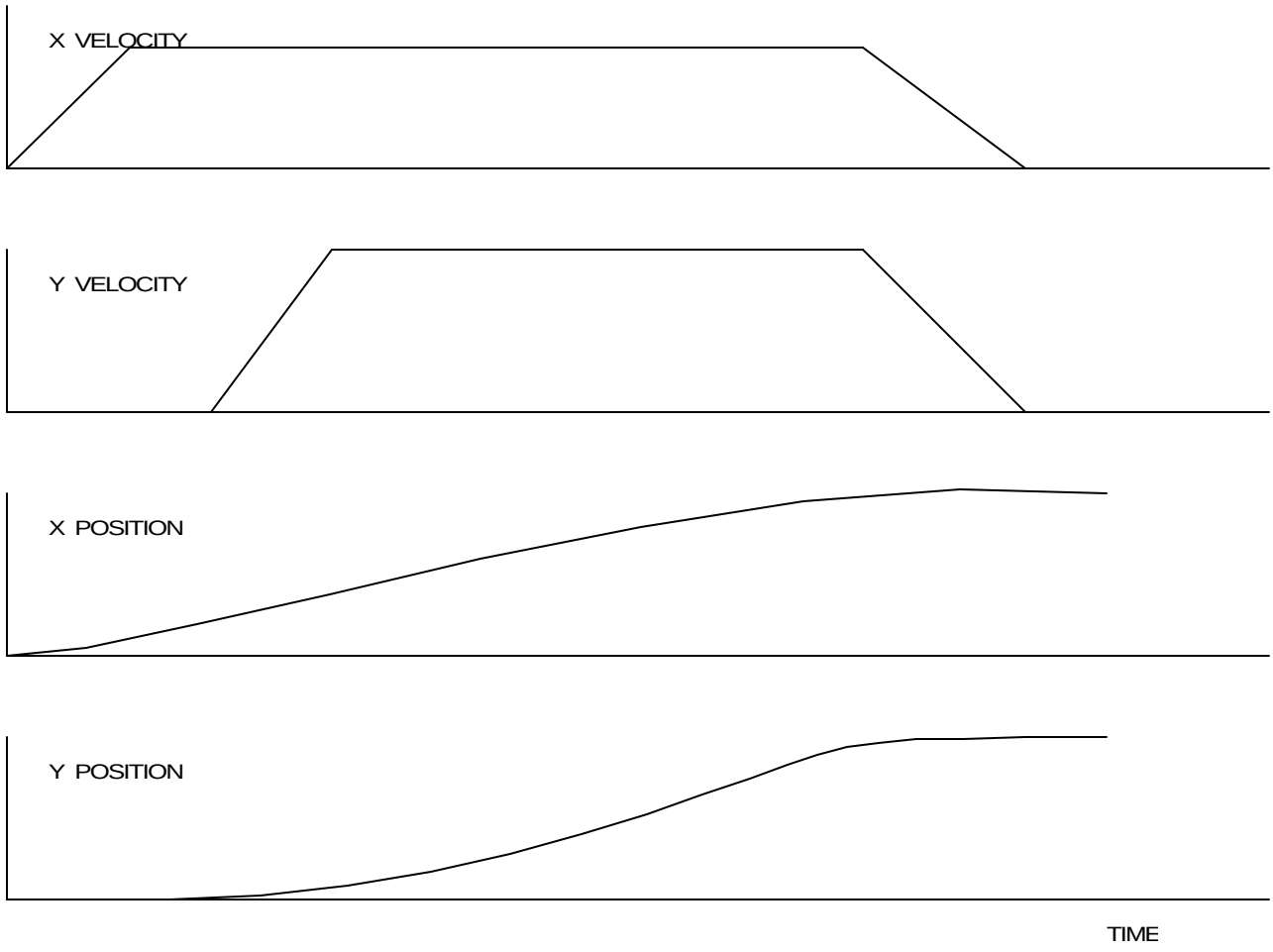


Figure 10.3 - Velocity and Position Profiles

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 Modelling

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.

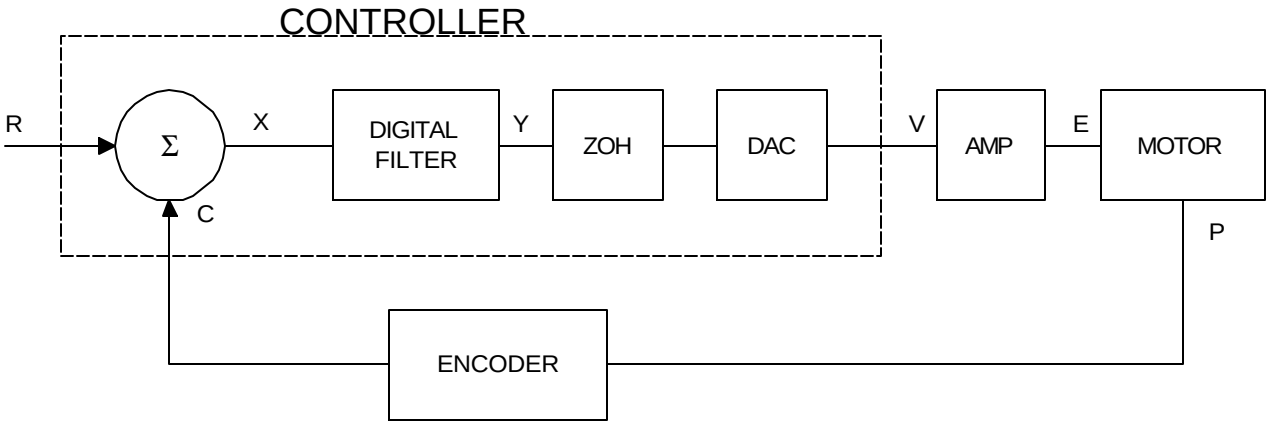


Figure 10.4 - Functional Elements of a Motion Control System

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 sec

and

T_e = 0.002 sec

Assuming that the amplifier gain is Kv = 4, the resulting transfer function is

P/V = 40/[s(0.04s+1)(0.002s+1)]

Current Drive

The current drive generates a current I, which is proportional to the input voltage, V, with a gain of Ka. The resulting transfer function in this case is

P/V = 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:

$$P/V = 1000/s^2 \qquad [rad/V]$$

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

$$P/V = 1/[K_g s(sT1+1)]$$

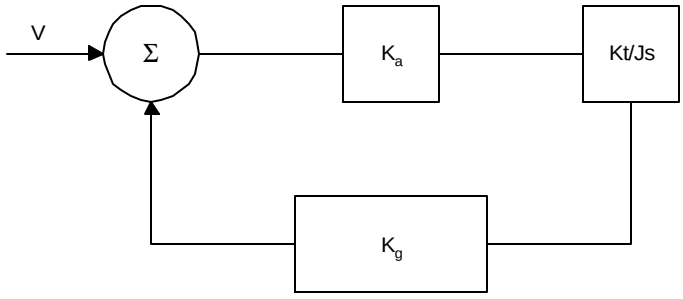
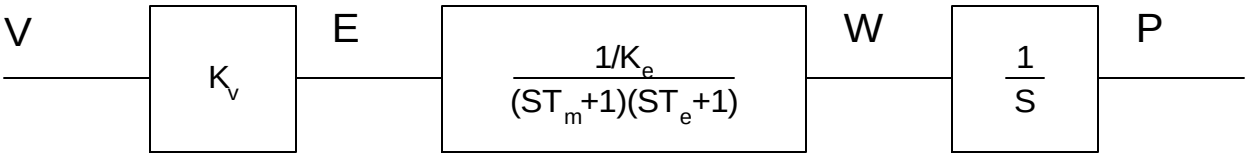


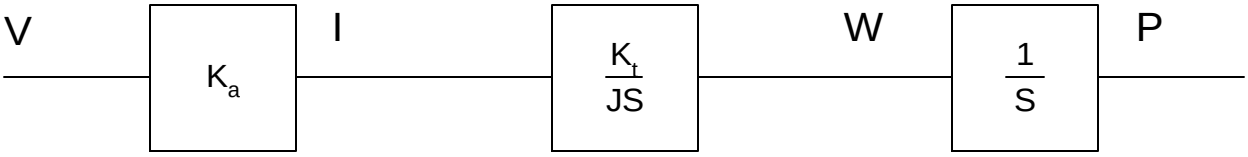
Figure 10.5 - Elements of velocity loops

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

VOLTAGE SOURCE



CURRENT SOURCE



VELOCITY LOOP

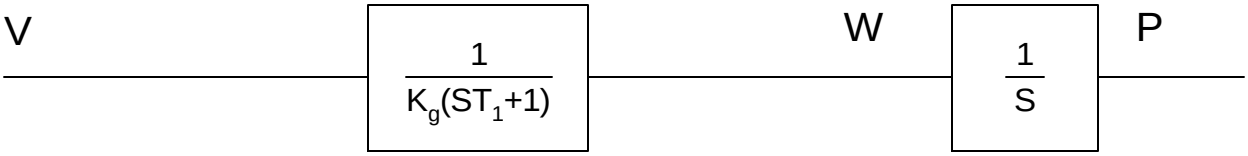


Figure 10.6 - Mathematical model of the motor and amplifier in three operational modes

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 modelled 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-2x00 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-2x00 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

$$M(s) = P/I = Kt/Js2 = 500/s^2 \text{ [rad/A]}$$

Amp

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

DAC

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

Encoder

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

ZOH

$$2000/(s+2000)$$

Digital Filter

$$KP = 12.5,\ KD = 245,\ T = 0.001$$

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.

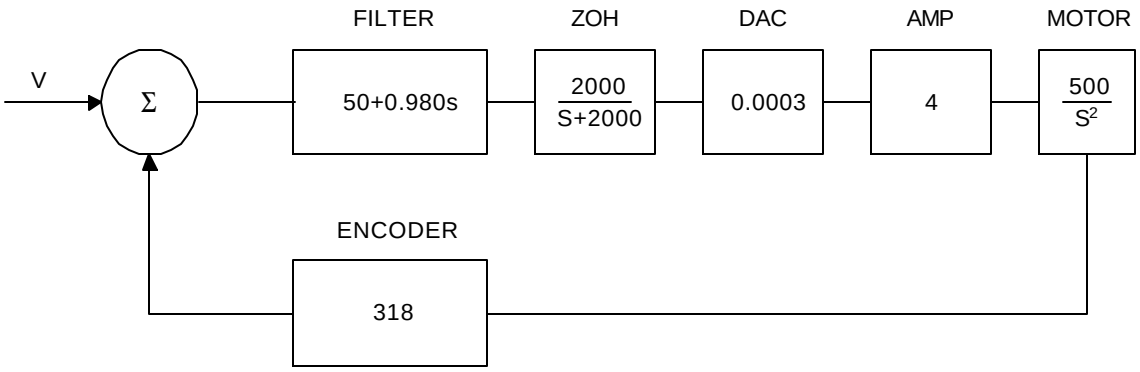


Figure 10.7 - Mathematical model of the control system

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 ω_c) equals one. This can be done by the Bode plot of A(j ω_c), as shown in Fig. 10.8.

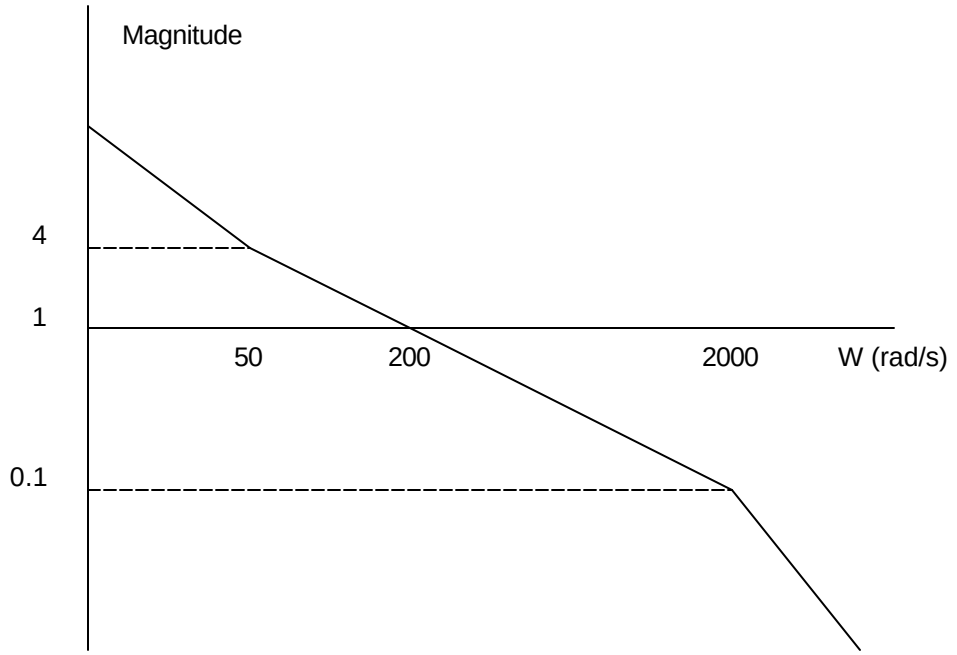


Figure 10.8 - Bode plot of the open loop transfer function

For the given exa mple, the crossover frequency was computed numerically resulting in 200 rad/s.

Next, we determine the phase of A(s) at the crossover frequency.

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

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

$$\alpha = 76° - 180° - 6° = -110°$$

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-2x00 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-2x00 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 $\omega_c = 500$, the function would have a magnitude of 160 and a phase lead of 59 degrees.

These requirements may be expressed as:

$$|G(j500)| = |P + (j500D)| = 160$$

and

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

The solution of these equations leads to:

$$P = 160\cos 59° = 82.4$$

$$500D = 160\sin 59° = 137$$

Therefore,

$$D = 0.274$$

and

$$G = 82.4 + 0.2744s$$

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

$$D(z) = 4KP + 4KD(1-z^{-1})$$

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-2x00 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-2x00
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

Limit Switch Inputs, Home Inputs.	2.2K ohm in series with opto-isolator. Active high or low requires at least 1mA to activate. Once activated, the input requires the current to go below 0.5ma. All Limit Switch and Home inputs use one common voltage (LSCOM) which can accept up to 24 Volts. Voltages above 24 Volts requires additional resistor.
IN[1] thru IN[8] Uncommitted Inputs and Abort Input	
IN[9] thru IN[16] Uncommitted Inputs (DMC-2x50 through DMC-2x80 only)	
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
OUT[9] thru OUT[16] Outputs:	TTL
(DMC-2x50 through DMC-2x80 only)	
IN[81], IN[82]	Auxiliary Encoder Inputs for A (X) axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 Volts.

IN[83], IN[84] (DMC-2x20 through DMC-2x80 only)	Auxiliary Encoder Inputs for B (Y) axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 Volts.
IN[85], IN[86] (DMC-2x30 through DMC-2x80 only)	Auxiliary Encoder Inputs for C (Z) axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 Volts.
IN[87], IN[88] (DMC-2x40 through DMC-2x80 only)	Auxiliary Encoder Inputs for D (W) axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 Volts.
IN[89], IN[90] (DMC-2x50 through DMC-2x80 only)	Auxiliary Encoder Inputs for E axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 Volts.
IN[91], IN[92] (DMC-2x60 through DMC-2x80 only)	Auxiliary Encoder Inputs for F axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 Volts.
IN[93], IN[94] (DMC-2x70 through DMC-2x80 only)	Auxiliary Encoder Inputs for G axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 Volts.
IN[95], IN[96] (DMC-2x80 only)	Auxiliary Encoder Inputs for H axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 Volts.

Power

+5V	1.1 A
+12V	40 mA
-12V	40 mA

Performance Specifications

Minimum Servo Loop Update Time:

	Normal	Fast Firmware
DMC-2x10	250 µsec	125 µsec
DMC-2x20	250 µsec	125 µsec
DMC-2x30	375 µsec	250 µsec
DMC-2x40	375 µsec	250 µsec
DMC-2x50	500 µsec	375 µsec
DMC-2x60	500 µsec	375 µsec
DMC-2x70	625 µsec	500 µsec
DMC-2x80	625 µsec	500 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

Fast Update Rate Mode

The DMC-2x00 can operate with much faster servo update rates. This mode is known as 'fast mode' and allows the controller to operate with the following update rates:

DMC-2x10, DMC-2x20	125 usec
DMC-2x30, DMC-2x40	250 usec
DMC-2x50, DMC-2x60	375 usec
DMC-2x70, DMC-2x80	500 usec

In order to run the DMC-2x00 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 original DMC-2x00 utilities. To set the update rate use command TM.

When the controller is operating with the fast firmware, the following functions are **disabled:**

Gearing mode

Ecam mode

Pole (PL)

Analog Feedback (AF)

Stepper Motor Operation (MT 2,-2,2.5,-2.5)

Trippoints in thread 2-8

DMA channel

Tell Velocity Interrogation Command (TV)

Connectors for DMC-2x00 Main Board

DMC-2x00 Axes A-D High Density Connector

1 Analog Ground	51 nc
2 gnd	52 gnd
3 5v	53 5v
4 error output	54 limit common
5 reset	55 home W
6 encoder-compare output	56 reverse limit W
7 gnd	57 forward limit W
8 gnd	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 Y	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 gnd
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 pos
38 A-Z	88 gnd
39 B+Z	89 gnd
40 B-Z	90 gnd
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

DMC-2x00 Axes E-H High Density Connector

1 nc	51 nc
2 gnd	52 gnd
3 5v	53 5v
4 error output	54 limit common
5 reset	55 home H
6 encoder-compare output	56 reverse limit H
7 gnd	57 forward limit H
8 gnd	58 home G
9 motor command H	59 reverse limit G
10 sign H / dir H	60 forward limit G
11 pwm H / step H	61 home F
12 motor command G	62 reverse limit F
13 sign G / dir G	63 forward limit F
14 pwm G / step G	64 home E
15 motor command F	65 reverse limit E
16 sign F / dir F	66 forward limit E
17 pwm F / step F	67 gnd
18 motor command E	68 5v
19 sign E / dir E	69 input common
20 pwm E / step E	70 latch E
21 amp enable H	71 latch F
22 amp enable G	72 latch G
23 amp enable F	73 latch H
24 amp enable E	74 input 13
25 A+E	75 input 14
26 A- E	76 input 15
27 B+E	77 input 16
28 B-E	78 abort
29 I+E	79 output 9
30 I-E	80 output 10
31 A+F	81 output 11
32 A-F	82 output 12
33 B+F	83 output 13
34 B-F	84 output 14
35 I+F	85 output 15
36 I-F	86 output 16
37 A+G	87 5v
38 A-G	88 gnd
39 B+G	89 gnd
40 B-G	90 gnd
41 I+G	91 nc
42 I-G	92 nc

43 A+H	93 nc
44 A-H	94 nc
45 B+H	95 nc
46 B-H	96 nc
47 I+H	97 nc
48 I-H	98 nc
49 +12V	99 -12v
50 +12V	100 -12v

DMC-2x00 Auxiliary Encoder 36 Pin High Density Connector

1 5v	19 5v
2 gnd	20 gnd
3 +aaX	21 +aaE
4 -aaX	22 -aaE
5 +abX	23 +abE
6 -abX	24 -abE
7 +aaY	25 +aaF
8 -aaY	26 -aaF
9 +abY	27 +abF
10 -abY	28 -abF
11 +aaZ	29 +aaG
12 -aaZ	30 -aaG
13 +abZ	31 +abG
14 -abZ	32 -abG
15 +aaW	33 +aaH
16 -aaW	34 -aaH
17 +abW	35 +abH
18 -abW	36 -abH

DMC-2x00 Extended I/O 80 Pin High Density Connector

Pin	Signal	Block	Bit @IN[n], @OUT[n]	Bit No
1	I/O	8	72	7
2	I/O	9	73	0
3	I/O	8	71	6
4	I/O	9	74	1
5	I/O	8	70	5
6	I/O	9	75	2
7	I/O	8	69	4
8	I/O	9	76	3
9	I/O	8	68	3
10	I/O	9	77	4
11	I/O	8	67	2
12	I/O	9	78	5
13	I/O	8	66	1
14	I/O	9	79	6
15	I/O	8	65	0
16	I/O	9	80	7
17	I/O	7	64	7
18	GND	--	--	GND

19	I/O	7	63	6
20	GND	--	--	GND
21	I/O	7	62	5
22	GND	--	--	GND
23	I/O	7	61	4
24	GND	--	--	GND
25	I/O	7	60	3
26	GND	--	--	GND
27	I/O	7	59	2
28	GND	--	--	GND
29	I/O	7	58	1
30	GND	--	--	GND
31	I/O	7	57	0
32	I/O	6	56	7
33	I/O	6	55	6
34	I/O	6	54	5
35	I/O	6	53	4
36	I/O	6	52	3
37	I/O	6	51	2
38	I/O	6	50	1
39	I/O	6	49	0
40	+5V	--	--	+5V
41	I/O	4	40	7
42	I/O	5	41	0
43	I/O	4	39	6
44	I/O	5	42	1
45	I/O	4	38	5
46	I/O	5	43	2
47	I/O	4	37	4
48	I/O	5	44	3
49	I/O	4	36	3
50	I/O	5	45	4
51	I/O	4	35	2
52	I/O	5	46	5
53	I/O	4	34	1
54	I/O	5	47	6
55	I/O	4	33	0
56	I/O	5	48	7
57	I/O	3	32	7
58	GND	--	--	GND
59	I/O	3	31	6
60	GND	--	--	GND
61	I/O	3	30	5
62	GND	-	--	GND
63	I/O	3	29	4
64	GND	--	--	GND
65	I/O	3	28	3
66	GND	--	--	GND
67	I/O	3	27	2
68	GND	--	--	GND
69	I/O	3	26	1
70	GND	--	--	GND

71	I/O	3	25	0
72	I/O	2	24	7
73	I/O	2	23	6
74	I/O	2	22	5
75	I/O	2	21	4
76	I/O	2	20	3
77	I/O	2	19	2
78	I/O	2	18	1
79	I/O	2	17	0
80	+5V	--	--	+5V

RS-232-Main Port

Standard connector and cable, 9Pin

Pin	Signal
1	CTS – OUTPUT
2	Transmit data-output
3	Receive data-input
4	RTS – input
5	Gnd
6	CTS – output
7	RTS – input
8	CTS – output
9	Nc

RS-232-Auxiliary Port

Standard connector and cable, 9Pin

Pin	Signal
1	CTS – input
2	Transmit data-input
3	Receive data-output
4	RTS – output
5	Gnd
6	CTS – input
7	RTS – output
8	CTS – input
9	5v

USB - In

Series B, 4 pos

Connector: Amp # 787780-1

USB - Out

Series A, 8 pos

Connector: Amp # 787617-1

Ethernet

100 BASE-T/10 BASE-T - Kycon GS-NS-88-3.5

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

10 BASE-2- AMP 227161-7

10 BASE-F- HP HFBR-1414 (TX, Transmitter)

HP HFBR-2416 (RX, Receiver)

Cable Connections for DMC-2x00

The DMC-2x00 requires the transmit, receive, and ground for slow communication rates. (ie 1200 baud) For faster rates the handshake lines are required. The connection tables below contain the handshake lines. These descriptions and tables are for RS-232 only. RS-422 is available on request.

Standard RS-232 Specifications

25 pin Serial Connector (Male, D-type)

This table describes the pinout for standard serial ports found on most comp uters.

Pin Number	Function
1	NC
2	Transmitted Data
3	Received Data
4	Request to Send
5	Clear to Send
6	Data Set Ready
7	Signal Ground
8	Carrier Detect
9	+Transmit Current Loop Return
10	NC
11	-Transmit Current Loop Data
12	NC
13	NC
14	NC

15	NC
16	NC
17	NC
18	+Receive Current Loop Data
19	NC
20	Data Terminal Ready
21	NC
22	Ring Indicator
23	NC
24	NC
25	-Receive Current Loop Return

9 Pin Serial Connector (Male, D-type)

Standard serial port connections found on most computers.

PIN NUMBER	FUNCTION
1	Carrier Detect
2	Receive Data
3	Transmit Data
4	Data Terminal Ready
5	Signal Ground
6	Data Set Ready
7	Request to Send
8	Clear to Send
9	Ring Indicator

DMC-2x00 Serial Cable Specifications

Cable to Connect Computer 25 pin to Main Serial Port

25 Pin (Male - computer)	9 Pin (female - controller)
8 (Carrier Detect)	1
3 (Receive Data)	2
2 (Transmit Data)	3
20 (Data Terminal Ready)	4
7 (Signal Ground)	5
Controller Ground	9

Cable to Connect Computer 9 pin to Main Serial Port Cable (9 pin)

9 Pin (FEMALE - Computer)	9 Pin (FEMALE - Controller)
1 (Carrier Detect)	1
2 (Receive Data)	2
3 (Transmit Data)	3

4 (Data Terminal Ready)	4
5 (Signal Ground)	5
Controller Ground	9

Cable to Connect Computer 25 pin to Auxiliary Serial Port Cable (9 pin)

25 Pin (Male - terminal)	9 Pin (male - controller)
20 (Data Terminal Ready)	1
2 (Transmit Data)	2
3 (Receive Data)	3
8 (Carrier Detect)	4
7 (Signal Ground)	5
Controller +5V	9

Cable to Connect Computer 9 pin to Auxiliary Serial Port Cable (9 pin)

9 Pin (FEMALE - terminal)	9 Pin (MALE - Controller)
4 (Data Terminal Ready)	1
3 (Transmit Data)	2
2 (Receive Data)	3
1 (Carrier Detect)	4
5 (Signal Ground)	5
Controller +5V	9

Pin-Out Description for DMC-2x00

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 stepmotors: 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 Output 9-Output 16 (DMC-2x50 thru 2x80	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. 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 9 - Input 16 isolated	Uncommitted inputs. May be defined by the user to trigger events. Inputs are checked with the Conditional Jump instruction and After Input instruction or Input Interrupt. Input 1 is latch A, Input 2 is latch B, Input 3 is latch C and Input 4 is latch D if the high speed position latch function is enabled.
Latch	High speed position latch to capture axis position within 20 nano seconds on occurrence of latch signal. AL command arms latch. Input 1 is latch A, Input 2 is latch B, Input 3 is latch C and Input 4 is latch D. Input 9 is latch E, input 10 is latch F, input 11 is latch G, input 12 is latch H.

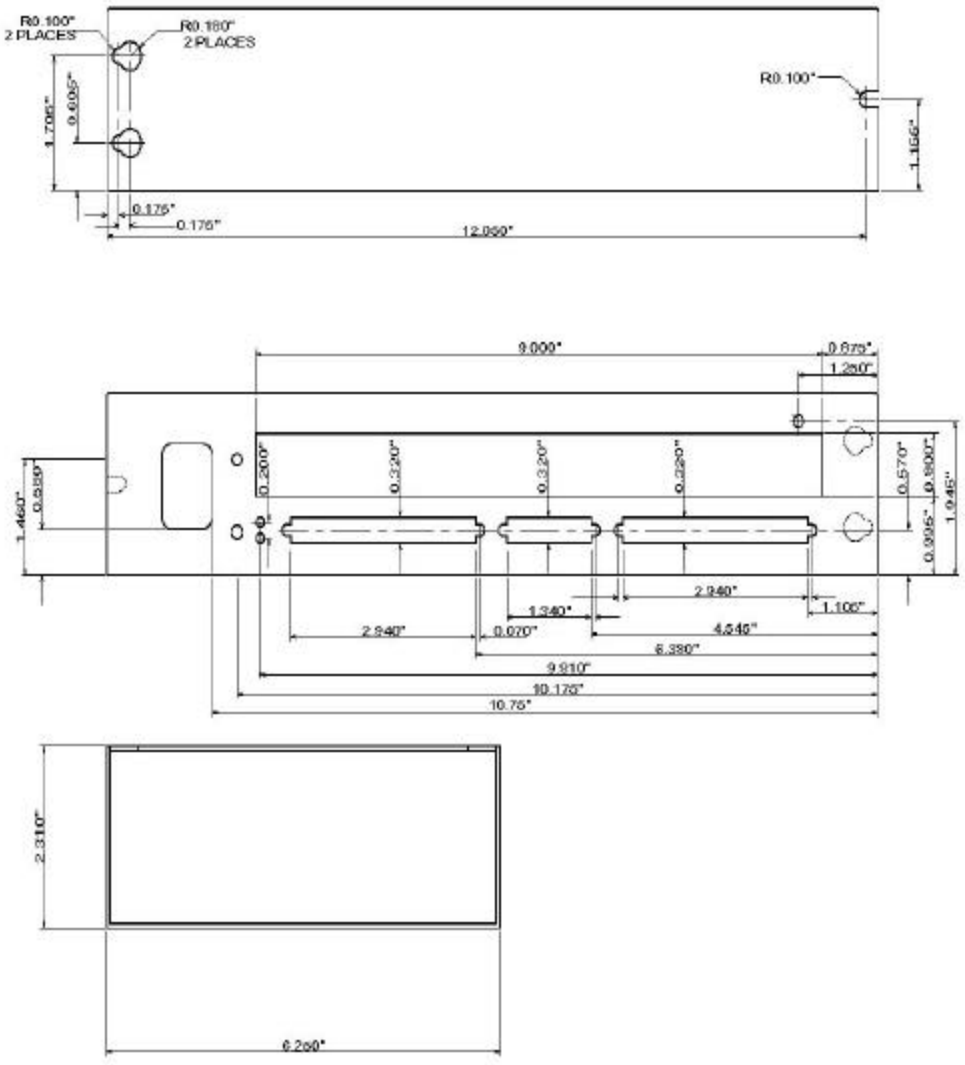
Jumper Description for DMC-2x00

Jumper	Label	Function (If jumpered)
JP5 MB	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 command is not valid with SM jumpered.
JP7 MB	SM E	
	SM F	
	SM G	
	SM H	
	OPT	Reserved
JP1 MB	MRST	Master Reset enable. Returns controller to factory default settings and erases EEPROM. Requires power-on or RESET to be activated.
JP 3 DB for DMC-2000 JP4 DB for DMC-2100/2200	UPGRADE	Used to upgrade controller firmware when resident firmware is corrupt.
JP4 DB for DMC-2000 JP 5 for DMC-2100/2200	AUX	Serial Port Configuration for RS-232/RS-422
JP3	MAIN	Main Serial Port configuration for RS-232/RS-422

NOTE: MB denotes motherboard. DB, daughter board.

Dimensions for DMC-2x00

DMC-2080 MOUNTING DIMENSIONS
Overall Dimensions: 12.0" x 2.3" x 6.25"



Accessories and Options

DMC-20x0	1- 8 axis motion controllers where x specifies the number of axes
-16	16-Bit ADC Option for analog inputs
CABLE-100-1M	100-pin high density cable, 1 meter
CABLE-100-4M	100-pin high density cable, 4 meter
CABLE-80-1M	80-pin high density cable, 1 meter
CABLE-80-4M	80-pin high density cable, 4 meter
CABLE-36-1M	36-pin high density cable, 1 meter
CABLE-36-4M	36-pin high density cable, 4 meter
CABLE-USB-2M	USB cable, 2 meter
CABLE-USB-3M	USB cable, 3 meter
CB-50-100	50-pin to 100-pin converter board, includes two 50-pin ribbon cables
CB-50-80	50-pin to 80-pin converter board, includes two 50-pin ribbon cables
ICM-1900	Interconnect module
-LAEN	Option for ICM-1900 Provides Active Low Amplifier Enable Signal
-OPTO	Option for ICM-1900 Provides Opto0isolation for digital outputs
-OPTOHC	Option for ICM-1900 Provides High Current Opto-isolation for digital outputs
AMP-19x0	Interconnect module with 1 - 4 brush motor amplifiers where x specifies the number of amplifiers.
-OPTO	Option for AMP-19x0 Provides Opto0isolation for digital outputs
-OPTOHC	Option for AMP-19x0 Provides High Current Opto-isolation for digital outputs
ICM-2900	Interconnect module with detachable screw terminal
-LAEN	Option for ICM-2900 Provides Active Low Amplifier Enable Signal
-FL	Option for ICM-2900 where the ICM-2900 includes flanges for rack mounting
-ST	ICM-2900 module with screw terminal
-OPTO	Option for AMP-19x0 Provides Opto0isolation for digital outputs
-OPTOHC	Option for AMP-19x0 Provides High Current Opto-isolation for digital outputs
Galil CD-ROM / Utilities. Includes the following: DMCWIN16 DMCWIN32 SETUP16 SETUP32 C KIT	 Windows 3.x Terminal Windows 95 / 98 / NT Terminal Setup Utility for Window 3.x Setup Utility for Windows 95/98/NT C-Programmers Kit
WSDK-16	Servo Design Kit for Windows 3.x
WSDK-32	Servo Design Kit for Windows 95 / 98 / NT
VBX Tool Kit	Visual Basic™ Tool Kit (includes VBXs and OCXs)
CAD-to-DMC	AutoCAD® DXF translator
MCS	Motion Control Selector. Utility for motor / amplifier sizing.
HPGL	HPGL translator

ICM-2900 Interconnect Module

The ICM-2900 interconnect module provides easy connections between the Optima series controllers and other system elements, such as amplifiers, encoders, and external switches. The ICM-2900 accepts the 100-pin main cable and provides terminal blocks for connections. Each terminal is labeled for quick connection of system elements. The ICM-2900 provides access to the signals for up to 4 axes (Two required for 5 or more axes).

Block (4 PIN)	Label	I/O	Description
1	MOCMDZ	O	Z axis motor command to amp input (w / respect to ground)
1	SIGNZ	O	Z axis sign output for input to stepper motor amp
1	PWMZ	O	Z axis pulse output for input to stepper motor amp
1	GND	O	Signal Ground
2	MOCMDW	O	W axis motor command to amp input (w / respect to ground)
2	SIGNW	O	W axis sign output for input to stepper motor amp
2	PWMW	O	W axis pulse output for input to stepper motor amp
2	GND	O	Signal Ground
3	MOCMDX	O	X axis motor command to amp input (w / respect to ground)
3	SIGNX	O	X axis sign output for input to stepper motor amp
3	PWMX	O	X axis pulse output for input to stepper motor amp
3	GND	O	Signal Ground
4	MOCMDY	O	Y axis motor command to amp input (w / respect to ground)
4	SIGNY	O	Y axis sign output for input to stepper motor amp
4	PWMY	O	Y axis pulse output for input to stepper motor amp
4	GND	O	Signal Ground
5	OUT PWR	I	Isolated Power In for Opto-Isolation Option
5	ERROR	O	Error output
5	CMP	O	Circular Compare Output
5	OUT GND	O	Isolated Ground for Opto-Isolation Option
6	AMPENW	O	W axis amplifier enable
6	AMPENZ	O	Z axis amplifier enable
6	AMPENY	O	Y axis amplifier enable
6	AMPENX	O	X axis amplifier enable
7	OUT5	O	General Output 5
7	OUT6	O	General Output 6
7	OUT7	O	General Output 7
7	OUT8	O	General Output 8
8	OUT1	O	General Output 1
8	OUT2	O	General Output 2
8	OUT3	O	General Output 3

8	OUT4	O	General Output 4
9	+5V	O	+ 5 Volts
9	HOMEZ	I	Z axis home input
9	RLSZ	I	Z axis reverse limit switch input
9	FLSZ	I	Z axis forward limit switch input
10	LSCOM	I	Limit Switch Common Input
10	HOMEW	I	W axis home input
10	RLSW	I	W axis reverse limit switch input
10	FLSW	I	W axis forward limit switch input
11	HOMEX	I	X axis home input
11	RLSX	I	X axis reverse limit switch input
11	FLSX	I	X axis forward limit switch input
11	GND	O	Signal Ground
12	HOMEY	I	Y axis home input
12	RLSY	I	Y axis reverse limit switch input
12	FLSY	I	Y axis forward limit switch input
12	GND	O	Signal Ground
13	IN5	I	Input 5
13	IN6	I	Input 6
13	IN7	I	Input 7
13	IN8	I	Input 8
14	XLATCH	I	Input 1 (Used for X axis latch input)
14	YLATCH	I	Input 2 (Used for Y axis latch input)
14	ZLATCH	I	Input 3 (Used for Z axis latch input)
14	WLATCH	I	Input 4 (Used for W axis latch input)
15	+5V	O	+ 5 Volts
15	+12V	O	+12 Volts
15	-12V	O	-12 Volts
15	ANA GND	O	Isolated Analog Ground for Use with Analog Inputs
16	INCOM	I	Input Common For General Use Inputs
16	ABORT	I	Abort Input
16	RESET	I	Reset Input
16	GND	O	Signal Ground
17	ANALOG5	I	Analog Input 5
17	ANALOG6	I	Analog Input 6
17	ANALOG7	I	Analog Input 7
17	ANALOG8	I	Analog Input 8
18	ANALOG1	I	Analog Input 1
18	ANALOG2	I	Analog Input 2
18	ANALOG3	I	Analog Input 3
18	ANALOG4	I	Analog Input 4

19	+5V	O	+ 5Volts
19	+INX	I	X Main encoder Index +
19	-INX	I	X Main encoder Index -
19	GND	O	Signal Ground
20	+MAX	I	X Main encoder A+
20	-MAX	I	X Main encoder A-
20	+MBX	I	X Main encoder B+
20	-MBX	I	X Main encoder B-
21	+5V	O	+ 5Volts
21	+INY	I	Y Main encoder Index +
21	-INY	I	Y Main encoder Index -
21	GND	O	Signal Ground
22	+MAY	I	Y Main encoder A+
22	-MAY	I	Y Main encoder A-
22	+MBY	I	Y Main encoder B+
22	-MBY	I	Y Main encoder B-
23	+5V	O	+ 5Volts
23	+INZ	I	Z Main encoder Index +
23	-INZ	I	Z Main encoder Index -
23	GND	O	Signal Ground
24	+MAZ	I	Z Main encoder A+
24	-MAZ	I	Z Main encoder A-
24	+MBZ	I	Z Main encoder B+
24	-MBZ	I	Z Main encoder B-
25	+5V	O	+ 5Volts
25	+INW	I	W Main encoder Index +
25	-INW	I	W Main encoder Index -
25	GND	O	Signal Ground
26	+MAW	I	W Main encoder A+
26	-MAW	I	W Main encoder A-
26	+MBW	I	W Main encoder B+
26	-MBW	I	W Main encoder B-

ICM-2900 Drawing:

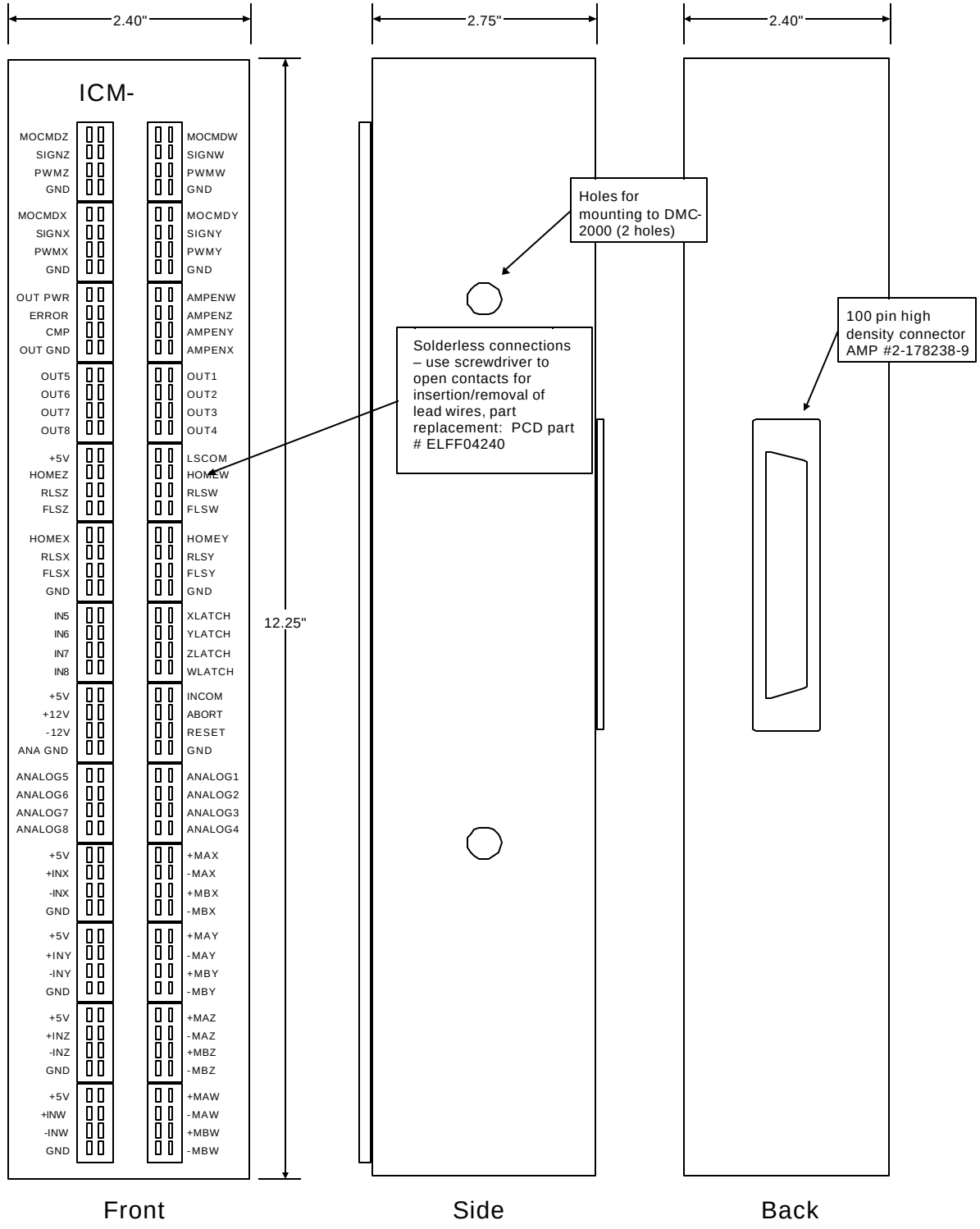


Figure A-1

ICM-2908 Interconnect Module

The ICM-2908 interconnect module provides easy connections between the auxiliary encoder connections of the DMC-2x00 series controller and other system elements. The ICM-2908 accepts the 36 pin high density cable (CABLE-36) from the controller and provides terminal blocks for easy access. Each terminal is labeled for quick connection. One ICM-1900 provides access to all of the auxiliary encoders on a DMC-2x00 (up to 8 axes).

ICM-2908 Drawing:

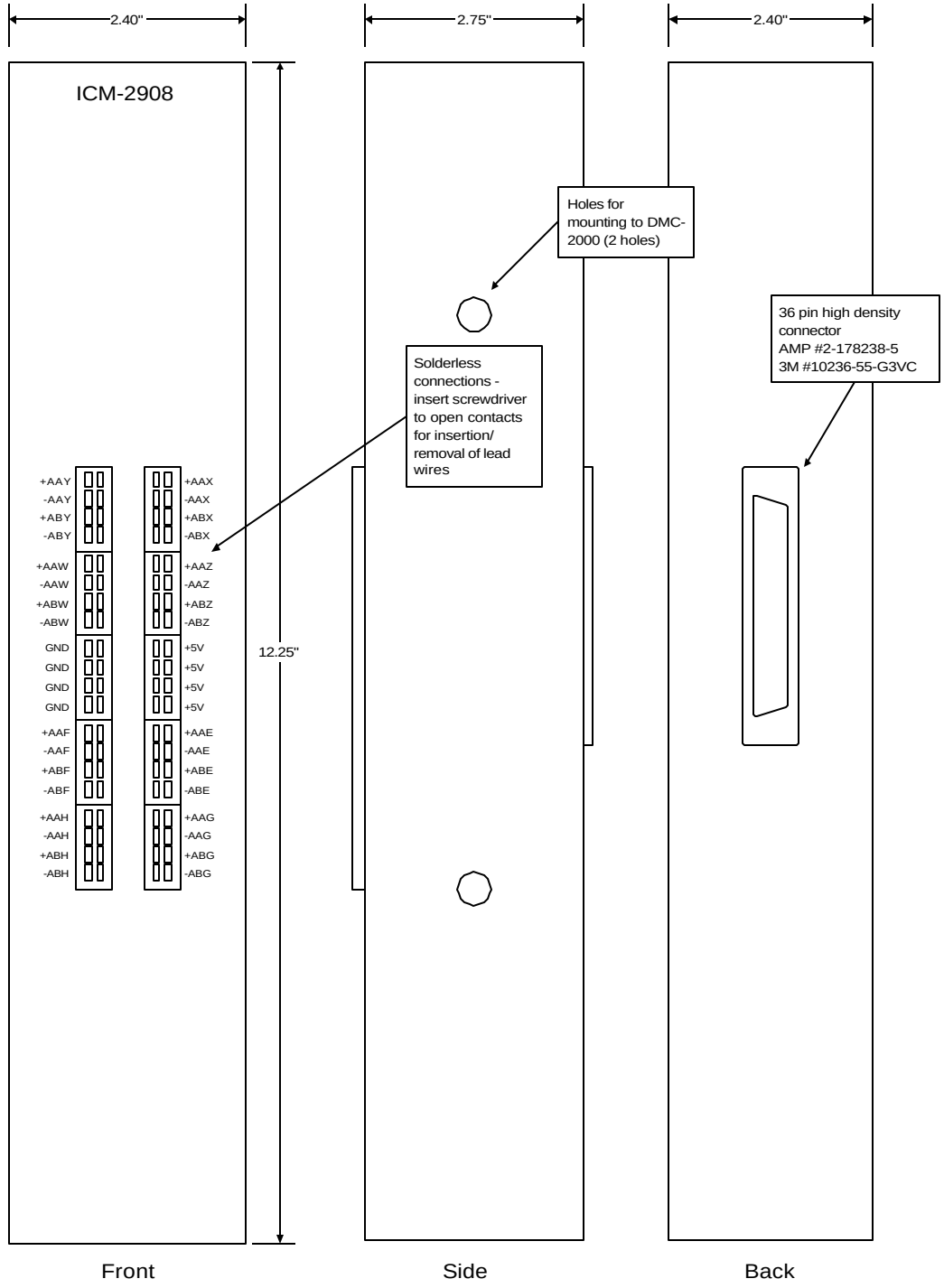
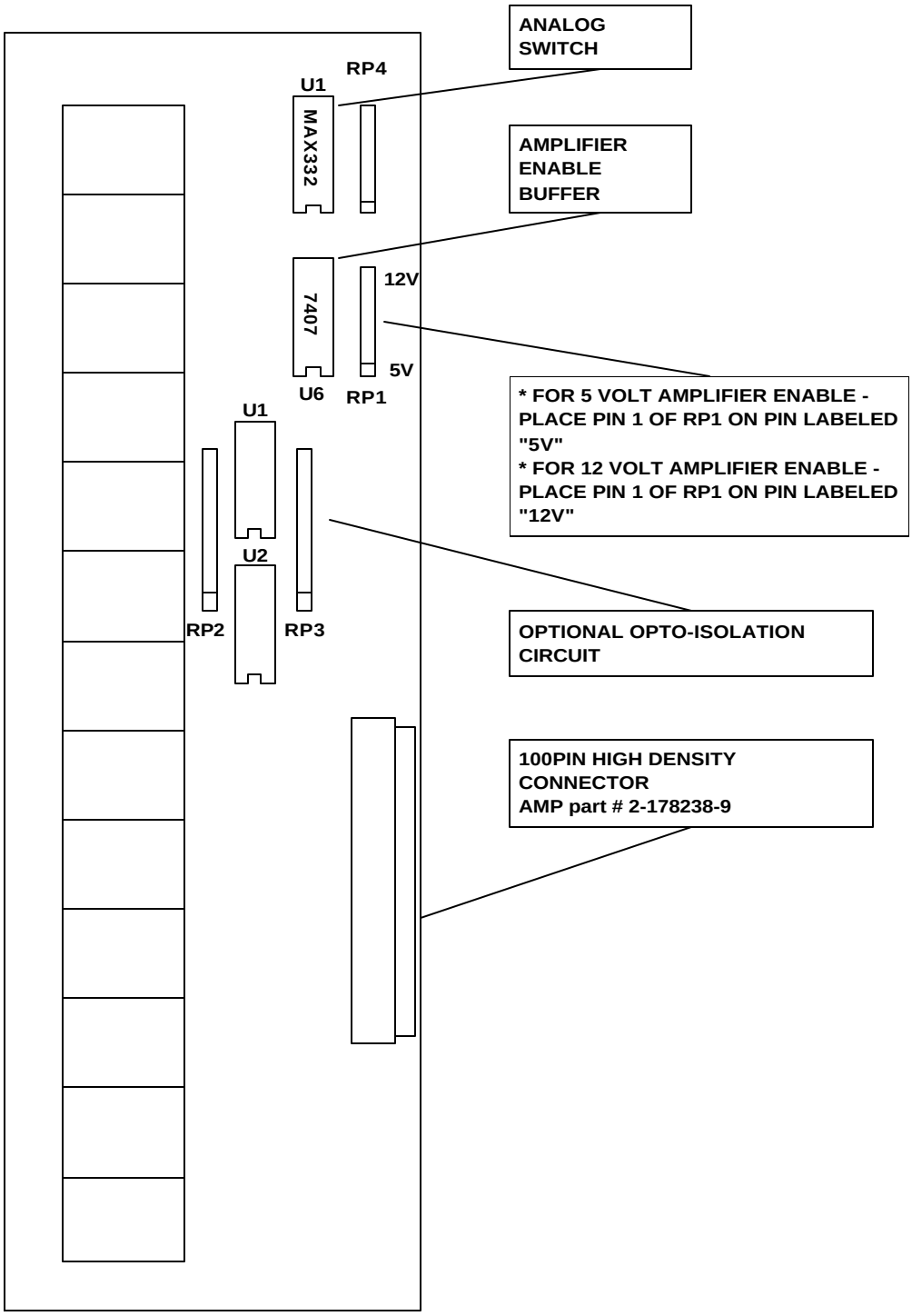


Figure A-2

PCB Layout of the ICM-2900:



ICM-2900 BOARD LAYOUT

ICM-1900 Interconnect Module

The ICM-1900 interconnect module provides easy connections between the DMC-2x00 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. (Two required for DMC-2x50 thru DMC-2x80).

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

- Separate DMC-2x00 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

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

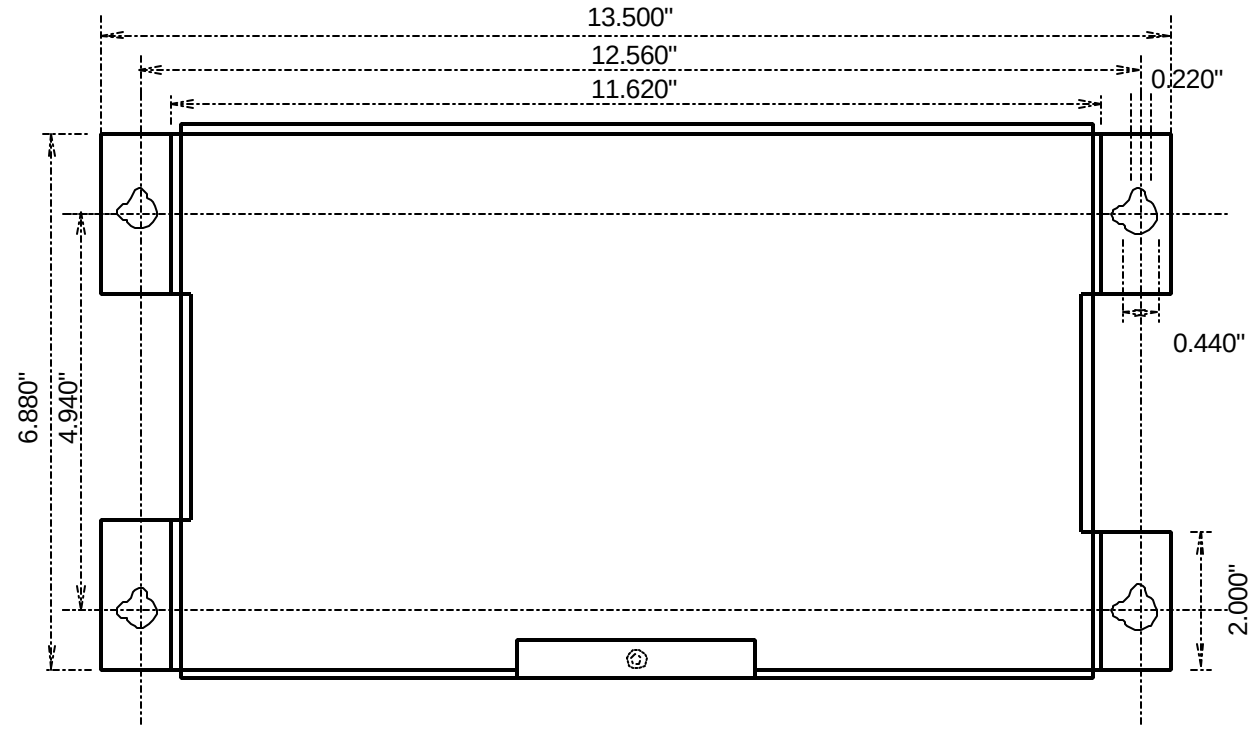
Terminal	Label	I/O	Description
1	+AAX	I	X Auxiliary encoder A+
2	-AAX	I	X Auxiliary encoder A-
3	+ABX	I	X Auxiliary encoder B+
4	-ABX	I	X Auxiliary encoder B-
5	+AAY	I	Y Auxiliary encoder A+
6	-AAY	I	Y Auxiliary encoder A-
7	+ABY	I	Y Auxiliary encoder B+
8	-ABY	I	Y Auxiliary encoder B-
9	+AAZ	I	Z Auxiliary 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	ISO OUT POWER	O	Isolated Output Power(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	ISO OUT GND	O	Isolated Output 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	ANA GND		Analog 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

ICM-1900 Drawing:



F

Figure A-3

AMP-19x0 Mating Power Amplifiers

The AMP-19X0 series are mating, brush-type servo amplifiers for the DMC-2x00. 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-2x00, and screwtype 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-2x00 series controllers

- Screw-type terminals for easy connection to motors, encoders, and switches
- Steel mounting plate with ¼" keyholes

Specifications

Minimum motor inductance: 1 mH

PWM frequency: 30 Khz

Ambient operating temperature: 0^o to 70^o C

Dimensions:

Weight:

Mounting: Keyholes – ¼" ∅

Gain: 1 amp/V

Opto-Isolated Outputs for ICM-2900 / ICM-1900 / AMP-19x0

The ICM/AMP 1900 and ICM-2900 modules from Galil have an option for opto-isolated outputs.

Standard Opto-Isolation and High Current Opto-isolation:

The Opto-isolation option on the ICM-1900 has 2 forms: -opto (standard) and -optohc (high current). The standard version provides outputs with 4ma drive current / output with approximately 2 usec response time. The high current version provides 25ma drive current / output with approximately 400 usec response time.

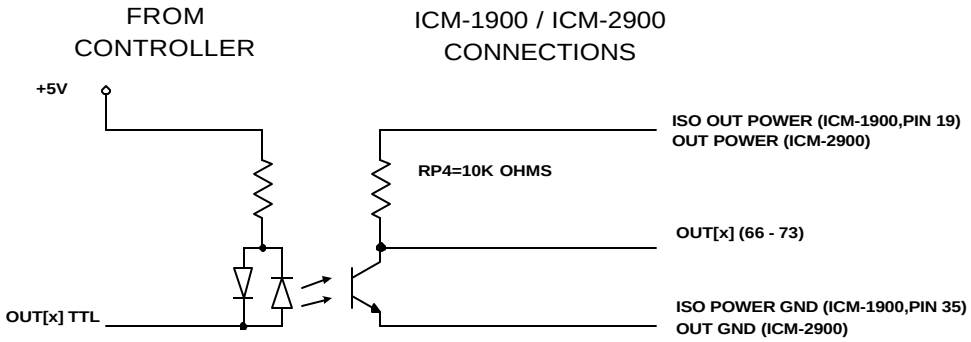


Figure A-4

The ISO OUT POWER (OUT POWER ON ICM-2900) and ISO POWER GND (OUT GND ON ICM-2900) signals should be connected to an isolated power supply. This power supply should be used only to power the outputs in order to obtain isolation from the controller. The signal "OUT[x]" is one of the isolated digital outputs where X stands for the digital output terminals.

The default configuration is for active high outputs. If active low outputs are desired, reverse RP3 in it's socket. This will tie RP3 to GND instead of VCC, inverting the sense of the outputs.

NOTE: If power is applied to the outputs with an isolated power supply but power is not applied to the controller, the outputs will float high (unable to sink current). This may present a problem when using active high logic and care should be taken. Using active low logic should avoid any problems associated with the outputs floating high.

Configuring the Amplifier Enable for ICM-2900 / ICM-1900

The ICM-1900 and ICM-2900 modules can be configured to provided an active low signal to enable external amplifiers. These modules can also be configured for voltage levels other than TTL.

-LAEN Option:

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 can be changed when using a Galil Interconnect Module. To change the polarity from active high (5 volts = enable, zero volts = disable) to active low (zero volts = enable, 5 volts = disable), replace the socketed IC, 7407, with a 7406. These IC's are labeled U6 on the ICM-1900 and U2 on the ICM-2900 and can be accessed by removing the cover. This option can be requested when ordering the unit by specifying the -LAEN option.

-Changing the Amplifier Enable Voltage Level:

To change the voltage level of the AEN signal, note the state of the resistor pack, labeled RP1 on the ICM-1900 / ICM-2900. 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.

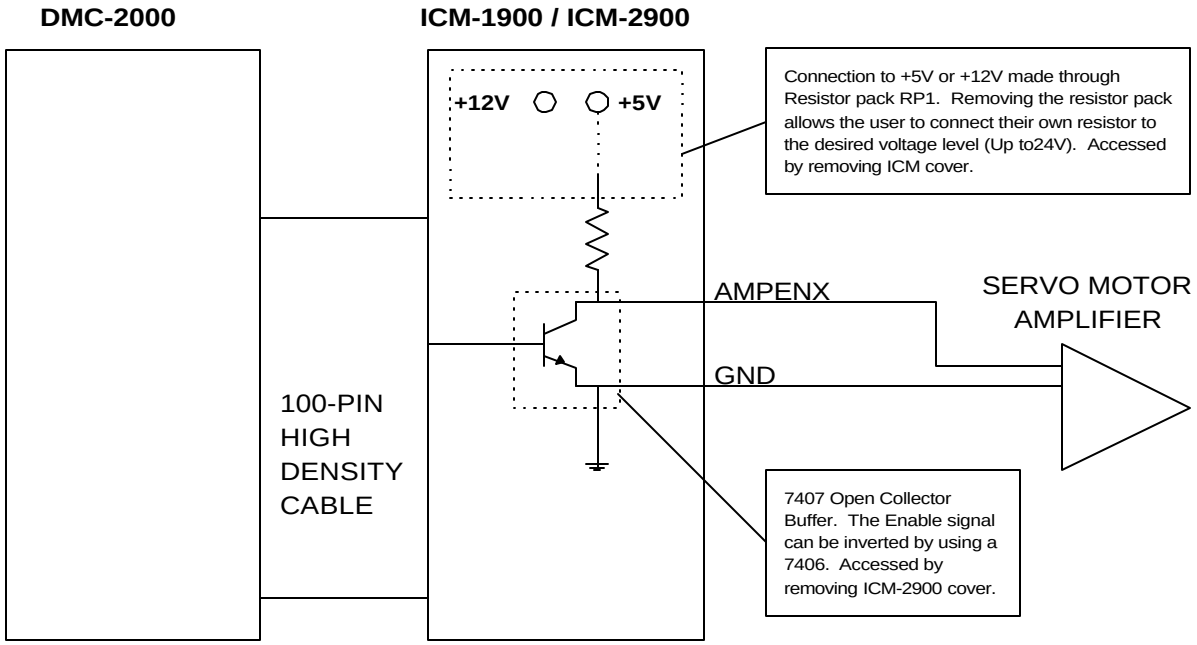


Figure A-5

IOM-1964 Opto-Isolation Module for Extended I/O

Description:

- Provides 64 optically isolated inputs and outputs, each rated for 2mA at up to 28 VDC
- Configurable as inputs or outputs in groups of eight bits
- Provides 16 high power outputs capable of up to 500mA each
- Connects to controller via 80 pin shielded cable
- All I/O points conveniently labeled
- Each of the 64 I/O points has status LED
- Dimensions 6.8” x 11.4”

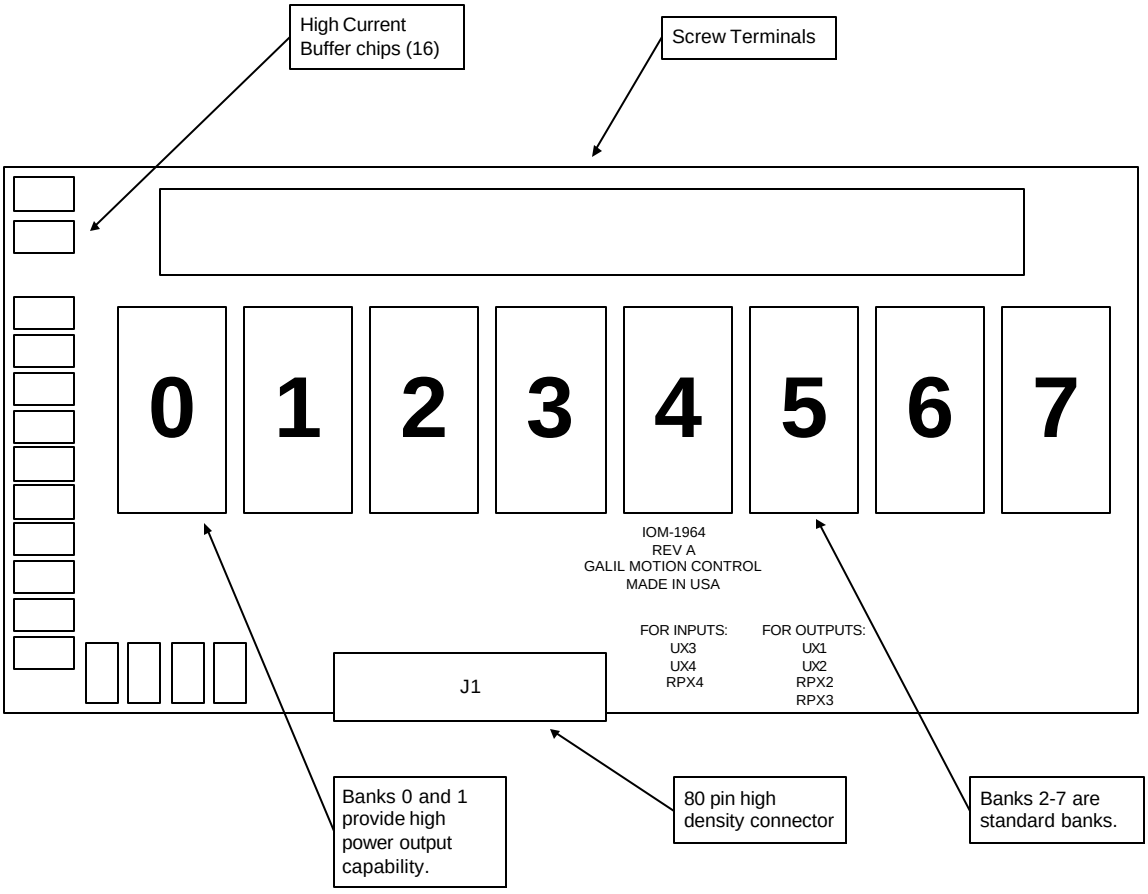


Figure A-6

Overview

The IOM-1964 is an input/output module that connects to the motion controller cards from Galil, providing optically isolated buffers for the extended inputs and outputs of the controller. The IOM-1964 also provides 16 high power outputs capable of 500mA of current per output point. The IOM-1964 splits the 64 I/O points into eight banks of eight I/O points each, corresponding to the eight banks

of extended I/O on the controller. Each bank is individually configured as an input or output bank by inserting the appropriate integrated circuits and resistor packs. The hardware configuration of the IOM-1964 must match the software configuration of the controller card.

All DMC-2x00 series controllers have general purpose I/O connections. On a DMC-2x10, -2x20, -2x30, and -2x40 the standard uncommitted I/O consists of: eight optically isolated digital inputs, eight TTL digital outputs, and eight analog inputs.

The **DMC-2x00**, however, has an additional 64 digital input/output points than the 16 described above for a total of 80 input/output points. An 80 pin shielded cable connects from the 80 pin connector of the DMC-2x00 to the 80 pin high density connector on the IOM-1964 (J1). Illustrations for this connection can be found on pages 10 and 11.

Configuring Hardware Banks

The extended I/O on the **DMC-2x00** is configured using the CO command. The banks of buffers on the IOM-1964 are configured to match by inserting the appropriate IC's and resistor packs. The layout of each of the I/O banks is identical.

For example, here is the layout of bank 0:

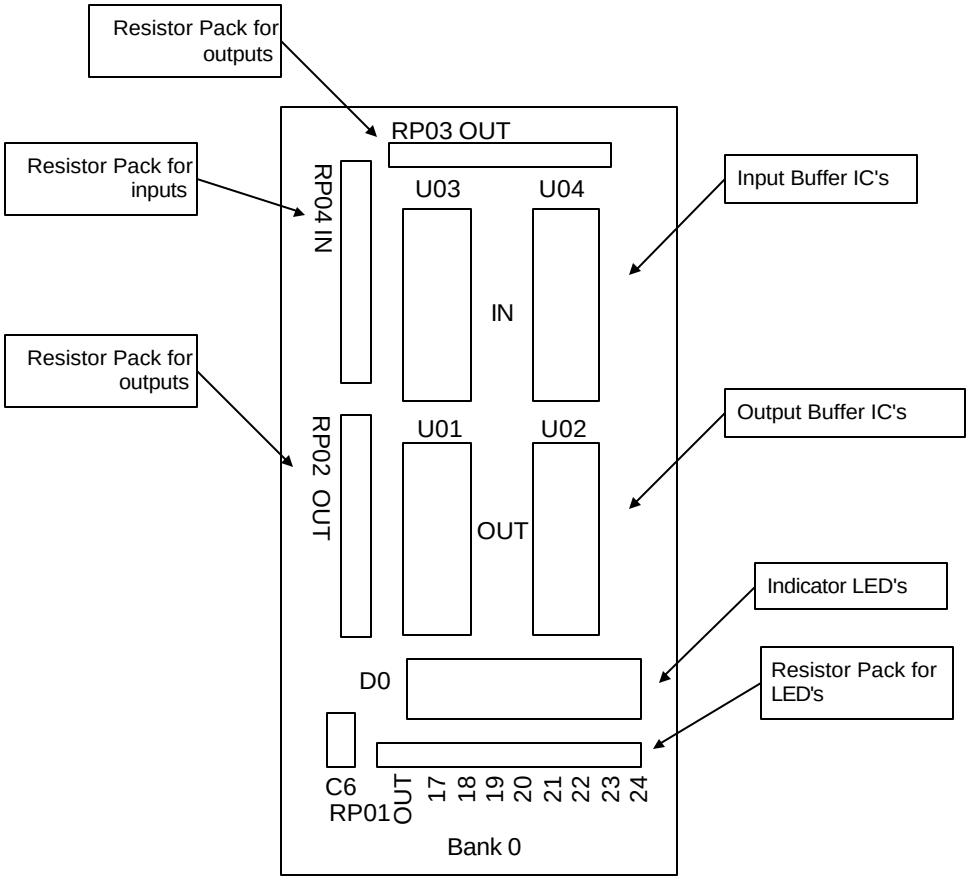


Figure A-7

All of the banks have the same configuration pattern as diagrammed above. For example, all banks have Ux1 and Ux2 output optical isolator IC sockets, labeled in bank 0 as U01 and U02, in bank 1 as U11 and U12, and so on. Each bank is configured as inputs or outputs by inserting optical isolator IC's and resistor packs in the appropriate sockets. A group of eight LED's indicates the status of each

I/O point. The numbers above the Bank 0 label indicate the number of the I/O point corresponding to the LED above it.

Digital Inputs

Configuring a bank for inputs requires that the Ux3 and Ux4 sockets be populated with NEC2505 optical isolation integrated circuits. The IOM-1964 is shipped with a default configuration of banks 2-7 configured as inputs. The output IC sockets Ux1 and Ux2 must be empty. The input IC's are labeled Ux3 and Ux4. For example, in bank 0 the IC's are U03 and U04, bank 1 input IC's are labeled U13 and U14, and so on. Also, the resistor pack RPx4 must be inserted into the bank to finish the input configuration.

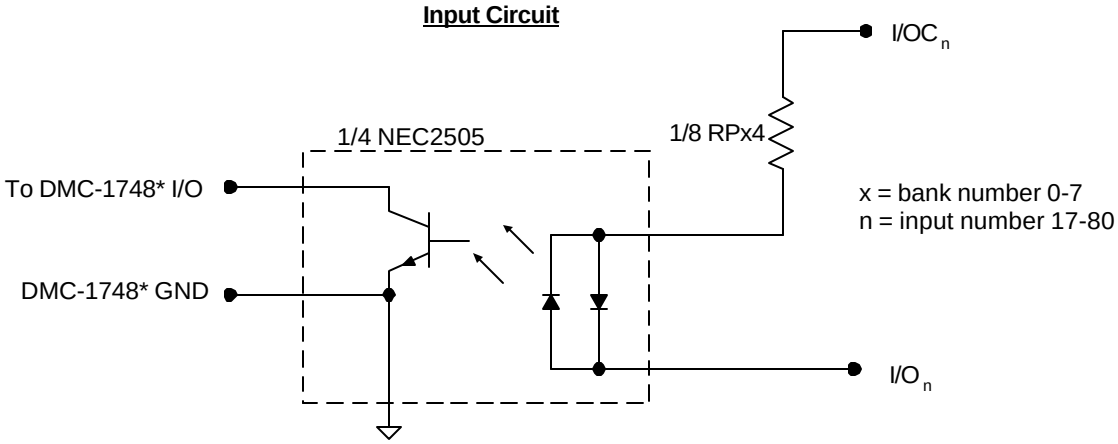


Figure A-8

Connections to this optically isolated input circuit are done in a sinking or sourcing configuration, referring to the direction of current. Some example circuits are shown below:

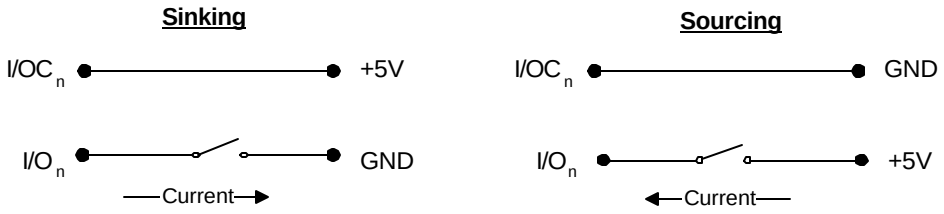


Figure A-9

There is one I/OC connection for each bank of eight inputs. Whether the input is connected as sinking or souring, when the switch is open no current flows and the digital input function @IN[n] returns 1. This is because of an internal pull up resistor on the DMC-2x40*. When the switch is closed in either circuit, current flows. This pulls the input on the DMC-2x40 to ground, and the digital input function @IN[n] returns 0. Note that the external +5V in the circuits above is for example only. The inputs are optically isolated and can accept a range of input voltages from 4 to 28 VDC.

Active outputs are connected to the optically isolated inputs in a similar fashion with respect to current. An NPN output is connected in a sinking configuration, and a PNP output is connected in the sourcing configuration.

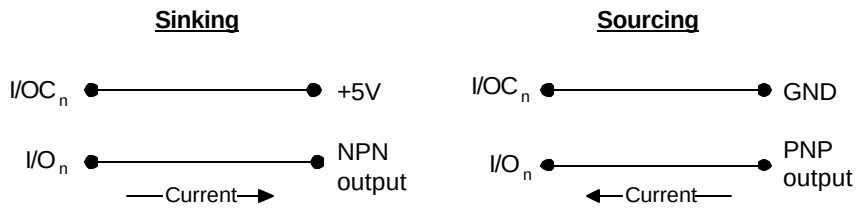


Figure A-10

Whether connected in a sinking or sourcing circuit, only two connections are needed in each case. When the NPN output is 5 volts, then no current flows and the input reads 1. When the NPN output goes to 0 volts, then it sinks current and the input reads 0. The PNP output works in a similar fashion, but the voltages are reversed i.e. 5 volts on the PNP output sources current into the digital input and the input reads 0. As before, the 5 volt is an example, the I/OC can accept between 4-28 volts DC.

Note that the current through the digital input should be kept below 3 mA in order to minimize the power dissipated in the resistor pack. This will help prevent circuit failures. The resistor pack RPx4 is standard 1.5k ohm which is suitable for power supply voltages up to 5.5 VDC. However, use of 24 VDC for example would require a higher resistance such as a 10k ohm resistor pack.

*The 1-4 axis models of the **DMC-2x00** all work with the IOM-1964, all have identical extended I/O features.

High Power Digital Outputs

The first two banks on the IOM-1964, banks 0 and 1, have high current output drive capability. The IOM-1964 is shipped with banks 0 and 1 configured as outputs. Each output can drive up to 500mA of continuous current. Configuring a bank of I/O as outputs is done by inserting the optical isolator NEC2505 IC's into the Ux1 and Ux2 sockets. The digital input IC's Ux3 and Ux4 are removed. The resistor packs RPx2 and RPx3 are inserted, and the input resistor pack RPx4 is removed.

Each bank of eight outputs shares one I/OC connection, which is connected to a DC power supply between 4 and 28 VDC. A 10k ohm resistor pack should be used for RPx3. Here is a circuit diagram:

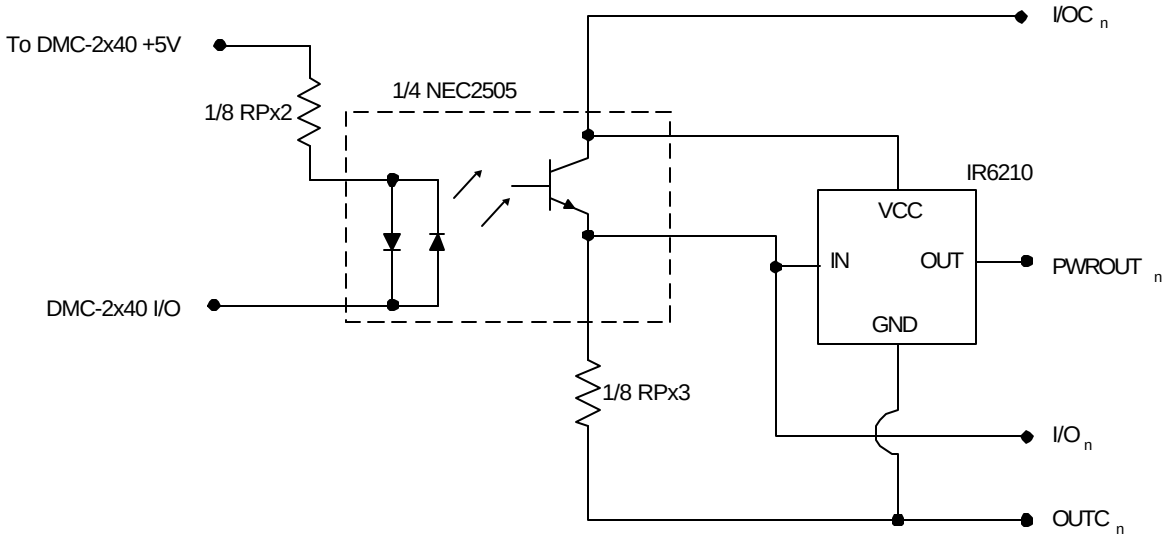


Figure A-11

The load is connected between the power output and output common. The I/O connection is for test purposes, and would not normally be connected. An external power supply is connected to the I/OC and OUTC terminals, which isolates the circuitry of the DMC-2x40 controller from the output circuit.

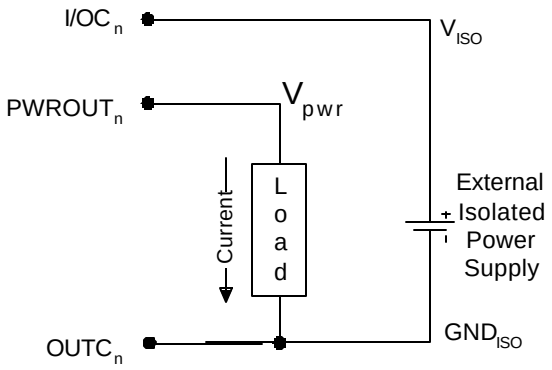


Figure A-12

The power outputs must be connected in a driving configuration as shown on the previous page. Here are the voltage outputs to expect after the Clear Bit and Set Bit commands are given:

Output Command	**Result**
CB_n	$V_{pwr} = V_{iso}$
SB_n	$V_{pwr} = GND_{iso}$

Standard Digital Outputs

The I/O banks 2-7 can be configured as optically isolated digital outputs, however these banks do not have the high power capacity as in banks 0-1. In order to configure a bank as outputs, the optical isolator chips Ux1 and Ux2 are inserted, and the digital input isolator chips Ux3 and Ux4 are removed. The resistor packs RPx2 and RPx3 are inserted, and the input resistor pack RPx4 is removed.

Each bank of eight outputs shares one I/OC connection, which is connected to a DC power supply between 4 and 28 VDC. The resistor pack RPx3 is optional, used either as a pull up resistor from the output transistor's collector to the external supply connected to I/OC or the RPx3 is removed resulting in an open collector output. Here is a schematic of the digital output circuit:

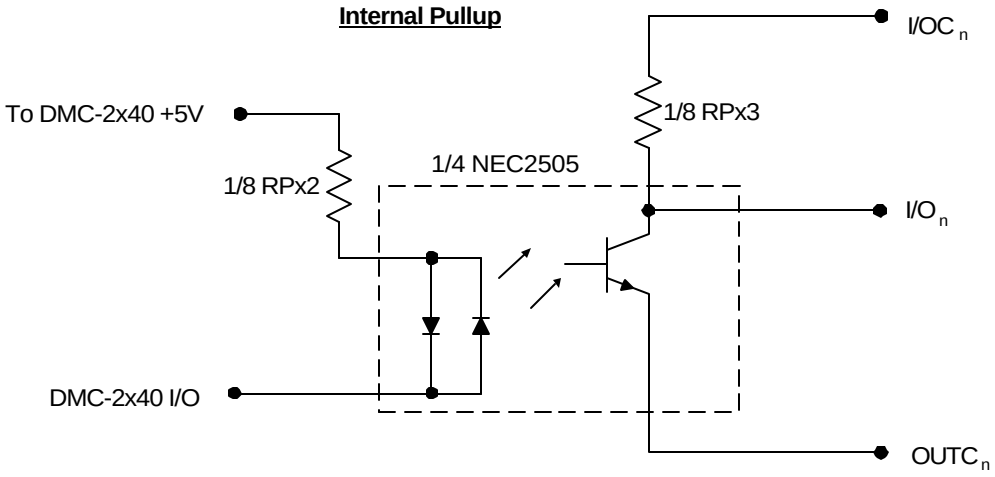


Figure A-13

The resistor pack RPx3 limits the amount of current available to source, as well as affecting the low level voltage at the I/O output. The maximum sink current is 2mA regardless of RPx3 or I/OC voltage, determined by the NEC2505 optical isolator IC. The maximum source current is determined by dividing the external power supply voltage by the resistor value of RPx3.

The high level voltage at the I/O output is equal to the external supply voltage at I/OC. However, when the output transistor is on and conducting current, the low level output voltage is determined by three factors. The external supply voltage, the resistor pack RPx3 value, and the current sinking limit of the NEC2505 all determine the low level voltage. The sink current available from the NEC2505 is between 0 and 2mA. Therefore, the maximum voltage drop across RPx3 is calculated by multiplying the 2mA maximum current times the resistor value of RPx3. For example, if a 10k ohm resistor pack is used for RPx3, then the maximum voltage drop is 20 volts. The digital output will never drop below the voltage at OUTC, however. Therefor a 10k ohm resistor pack will result in a low level voltage of .7 to 1.0 volts at the I/O output for an external supply voltage between 4 and 21 VDC. If a supply voltage greater than 21 VDC is used, a higher value resistor pack will be required.

Output Command	**Result**
CB_n	$V_{out} = GND_{iso}$
SB_n	$V_{out} = V_{iso}$

The resistor pack RPx3 is removed to provide open collector outputs. The same calculations for maximum source current and low level voltage applies as in the above circuit. The maximum sink current is determined by the NEC2505, and is approximately 2mA.

Open Collector

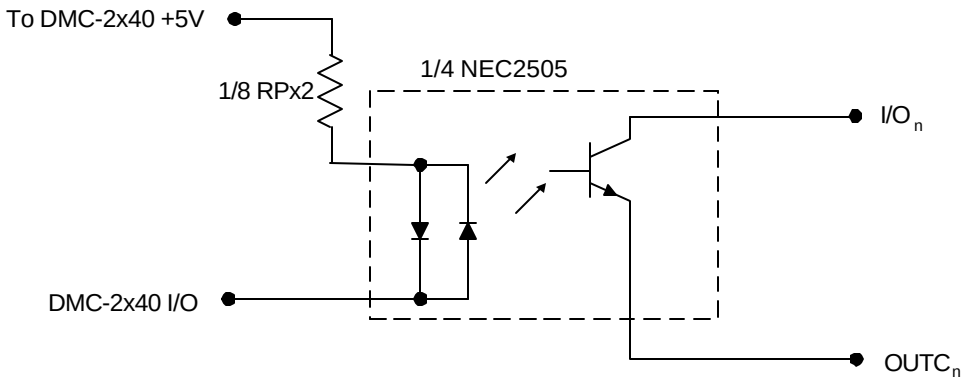


Figure A-14

Electrical Specifications

- I/O points, configurable as inputs or outputs in groups of 8

Digital Inputs

- Maximum voltage: 28 VDC
- Minimum input voltage: 4 VDC
- Maximum input current: 3 mA

High Power Digital Outputs

- Maximum external power supply voltage: 28 VDC
- Minimum external power supply voltage: 4 VDC
- Maximum source current, per output: 500mA
- Maximum sink current: sinking circuit inoperative

Standard Digital Outputs

- Maximum external power supply voltage: 28 VDC
- Minimum external power supply voltage: 4 VDC
- Maximum source current: limited by pull up resistor value
- Maximum sink current: 2mA

Relevant DMC Commands

CO n	Configures the 64 bits of extended I/O in 8 banks of 8 bits each. $N = n_2 + 2*n_3 + 4*n_4 + 8*n_5 + 16*n_6 + 32*n_7 + 64*n_8 + 128*n_9$ where n_x is a 1 or 0, 1 for outputs and 0 for inputs. The x is the bank number
OP m,n,o,p,q	m = 8 standard digital outputs n = extended I/O banks 0 & 1, outputs 17-32 = extended I/O banks 2 & 3, outputs 33-48 p = extended I/O banks 4 & 5, outputs 49-64 q = extended I/O banks 6 & 7, outputs 65-80
SB n	Sets the output bit to a logic 1, n is the number of the output from 1 to 80.
CB n	Clears the output bit to a logic 0, n is the number of the output from 1 to 80.
OB n,m	Sets the state of an output as 0 or 1, also able to use logical conditions.
TI n	Returns the state of 8 digital inputs as binary converted to decimal, n is the bank number +2.
_TI n	Operand (internal variable) that holds the same value as that returned by TI n.
@IN[n]	Function that returns state of individual input bit, n is number of the input from 1 to 80.

Screw Terminal Listing

TERM	LABEL	DESCRIPTION	BANK
1	GND	Ground pins of J1	N/A
2	5V	5V DC out from J1	N/A
3	GND	Ground pins of J1	N/A
4	5V	5V DC out from J1	N/A
5	I/O80	I/O bit 80	7
6	I/O79	I/O bit 79	7
7	I/O78	I/O bit 78	7
8	I/O77	I/O bit 77	7
9	I/O76	I/O bit 76	7
10	I/O75	I/O bit 75	7

11	I/O74	I/O bit 74	7
12	I/O73	I/O bit 73	7
13	OUTC73-80	Out common for I/O 73-80	7
14	I/OC73-80	I/O common for I/O 73-80	7
15	I/O72	I/O bit 72	6
16	I/O71	I/O bit 71	6
17	I/O70	I/O bit 70	6
18	I/O69	I/O bit 69	6
19	I/O68	I/O bit 68	6
20	I/O67	I/O bit 67	6
21	I/O66	I/O bit 66	6
22	I/O65	I/O bit 65	6
23	OUTC65-72	Out common for I/O 65-72	6
24	I/OC65-72	I/O common for I/O 65-72	6
25	I/O64	I/O bit 64	5
26	I/O63	I/O bit 63	5
27	I/O62	I/O bit 62	5
28	I/O61	I/O bit 61	5
29	I/O60	I/O bit 60	5
30	I/O59	I/O bit 59	5
31	I/O58	I/O bit 58	5
32	I/O57	I/O bit 57	5
33	OUTC57-64	Out common for I/O 57-64	5
34	I/OC57-64	I/O common for I/O 57-64	5
35	I/O56	I/O bit 56	4
36	I/O55	I/O bit 55	4
37	I/O54	I/O bit 54	4
38	I/O53	I/O bit 53	4
39	I/O52	I/O bit 52	4
40	I/O51	I/O bit 51	4
41	I/O50	I/O bit 50	4
42	I/O49	I/O bit 49	4
43	*OUTC49-56	Out common for I/O 49-56	4
44	I/OC49-56	I/O common for I/O 49-56	4
45	I/O48	I/O bit 48	3
46	I/O47	I/O bit 47	3
47	I/O46	I/O bit 46	3
48	I/O45	I/O bit 45	3
49	I/O44	I/O bit 44	3
50	I/O43	I/O bit 43	3
51	I/O42	I/O bit 42	3
52	I/O41	I/O bit 41	3

53	OUTC41-48	Out common for I/O 41-48	3
54	I/OC41-48	I/O common for I/O 41-48	3
55	I/O40	I/O bit 40	2
56	I/O39	I/O bit 39	2
57	I/O38	I/O bit 38	2
58	I/O37	I/O bit 37	2
59	I/O36	I/O bit 36	2
60	I/O35	I/O bit 35	2
61	I/O34	I/O bit 34	2
62	I/O33	I/O bit 33	2
63	OUTC33-40	Out common for I/O 33-40	2
64	I/OC33-40	I/O common for I/O 33-40	2
65	I/O32	I/O bit 32	1
66	I/O31	I/O bit 31	1
67	I/O30	I/O bit 30	1
68	I/O29	I/O bit 29	1
69	I/O28	I/O bit 28	1
70	I/O27	I/O bit 27	1
71	I/O26	I/O bit 26	1
72	I/O25	I/O bit 25	1
73	OUTC25-32	Out common for I/O 25-32	1
74	*I/OC25-32	I/O common for I/O 25-32	1
75	*OUTC25-32	Out common for I/O 25-32	1
76	I/OC25-32	I/O common for I/O 25-32	1
77	PWROUT32	Power output 32	1
78	PWROUT31	Power output 31	1
79	PWROUT30	Power output 30	1
80	PWROUT29	Power output 29	1
81	PWROUT28	Power output 28	1
82	PWROUT27	Power output 27	1
83	PWROUT26	Power output 26	1
84	PWROUT25	Power output 25	1
85	I/O24	I/O bit 24	0
86	I/O23	I/O bit 23	0
87	I/O22	I/O bit 22	0
88	I/O21	I/O bit 21	0
89	I/O20	I/O bit 20	0
90	I/O19	I/O bit 19	0
91	I/O18	I/O bit 18	0
92	I/O17	I/O bit 17	0
93	OUTC17-24	Out common for I/O 17-24	0
94	*I/OC17-24	I/O common for I/O 17-24	0

95	*OUTC17-24	Out common for I/O 17-24	0
96	I/OC17-24	I/O common for I/O 17-24	0
97	PWROUT24	Power output 24	0
98	PWROUT23	Power output 23	0
99	PWROUT22	Power output 22	0
100	PWROUT21	Power output 21	0
101	PWROUT20	Power output 20	0
102	PWROUT19	Power output 19	0
103	PWROUT18	Power output 18	0
104	PWROUT17	Power output 17	0

- Silkscreen on Rev A board is incorrect for these terminals.

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

CB-50-100 Adapter Board

The CB-50-100 adapter board can be used to convert the CABLE-100 to (2) 50 Pin Ribbon Cables. The 50 Pin Ribbon Cables provide a versatile method of accessing the controller signals without the use of a Galil Interconnect Module.

Connectors:

JC8 50 PIN IDC	J9 100 PIN HIGH DENSITY CONNECTOR
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20

21	21
22	22
23	23
24	24
25	25
26	26
27	27
28	28
29	29
30	30
31	31
32	32
33	33
34	34
35	35
36	36
37	37
38	38
39	39
40	40
41	41
42	42
43	43
44	44
45	45
46	46
47	47
48	48
49	49
50	50

JC6 50 PIN IDC	J9 100 PIN HIGH DENSITY CONNECTOR
1	51
2	52
3	53
4	54
5	55
6	56
7	57
8	58
9	59
10	60
11	61
12	62
13	63
14	64
15	65
16	66
17	67
18	68
19	69
20	70
21	71
22	72
23	73
24	74
25	75
26	76
27	77
28	78
29	79
30	80
31	81
32	82
33	83
34	84
35	85
36	86
37	87
38	88
39	89
40	90
41	91
42	92

43	93
44	94
45	95
46	96
47	97
48	98
49	99
50	100

CB-50-100 Drawing:

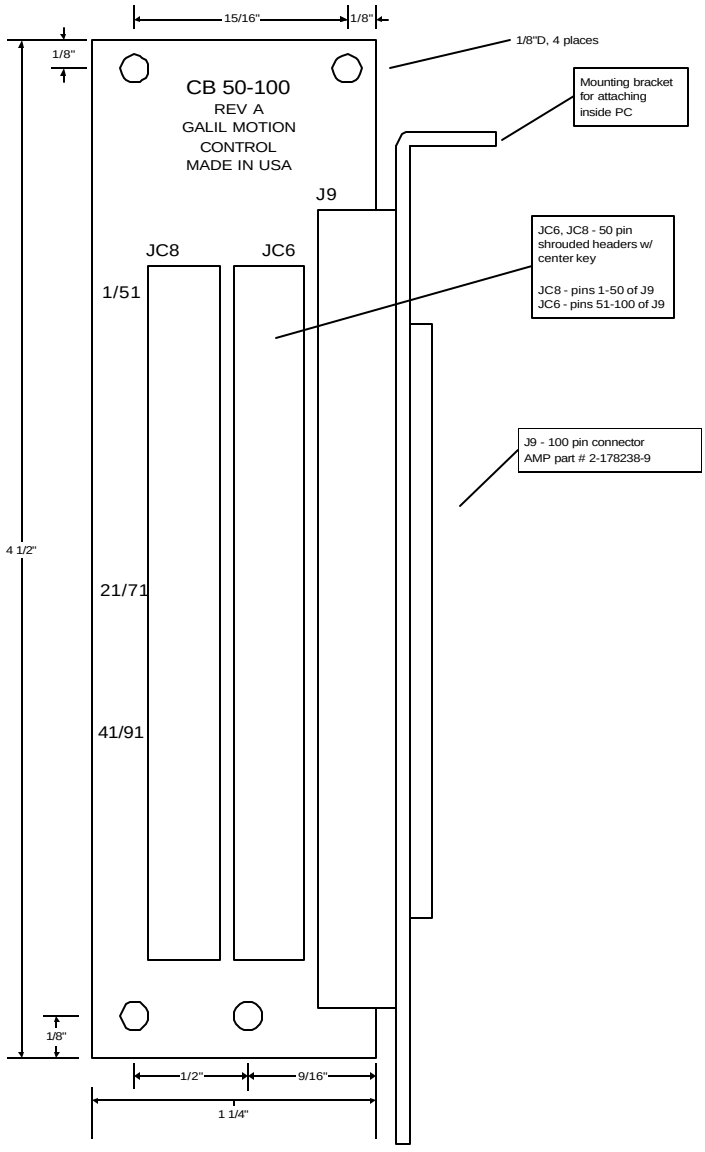


Figure A-15

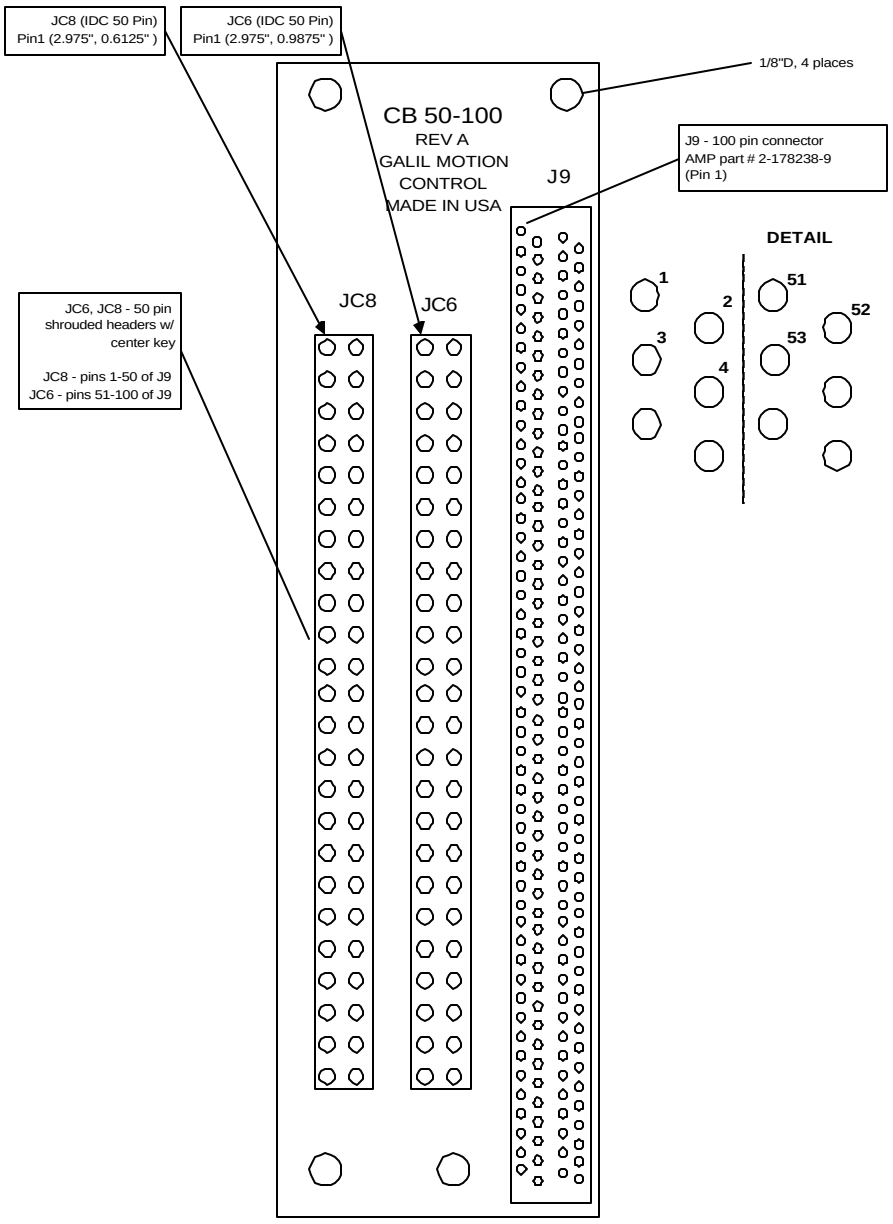


Figure A-16

CB-50-80 Adapter Board

The CB-50-80 adapter board can be used to convert the CABLE-80 to (2) 50 Pin Ribbon Cables. The 50 Pin Ribbon Cables provide a versatile method of accessing the extended I/O signals without the use of the Galil IOM-1964.

The ribbon cables provided by the CB-50-80 are compatible with I/O mounting racks such as Grayhill 70GRCM32-HL and OPTO-22 G4PB24.

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.

Connectors:

JC8 and JC6: 50 Pin Male IDC

J9: 80 Pin High Density Connector, AMP PART #3-178238-0

JC8	J9	JC8	J9
1	1	38	GND
2	2	39	35
3	3	40	GND
4	4	41	36
5	5	42	GND
6	6	43	37
7	7	44	GND
8	8	45	38
9	9	46	GND
10	10	47	39
11	11	48	GND
12	12	49	+5V
13	13	50	GND
14	14		
15	15		
16	16		
17	17		
18	GND		
19	19		
20	GND		
21	21		
22	GND		
23	23		
24	GND		
25	25		
26	GND		
27	27		
28	GND		
29	29		
30	GND		
31	31		
32	GND		
33	32		
34	GND		
35	33		
36	GND		
37	34		

JC6	J9 (Continued)
1	41
2	42
3	43
4	44
5	45
6	46
7	47
8	48
9	49
10	50
11	51
12	52
13	53
14	54
15	55
16	56
17	57
18	GND
19	59
20	GND
21	61
22	GND
23	63
24	GND
25	65
26	GND
27	67
28	GND
29	69
30	GND
31	71
32	GND
33	72
34	GND
35	73
36	GND
37	74
38	GND
39	75
40	GND
41	76
42	GND
43	77
44	GND
45	78
46	GND
47	79
48	GND
49	+5V
50	GND

CB-50-80 Drawing:

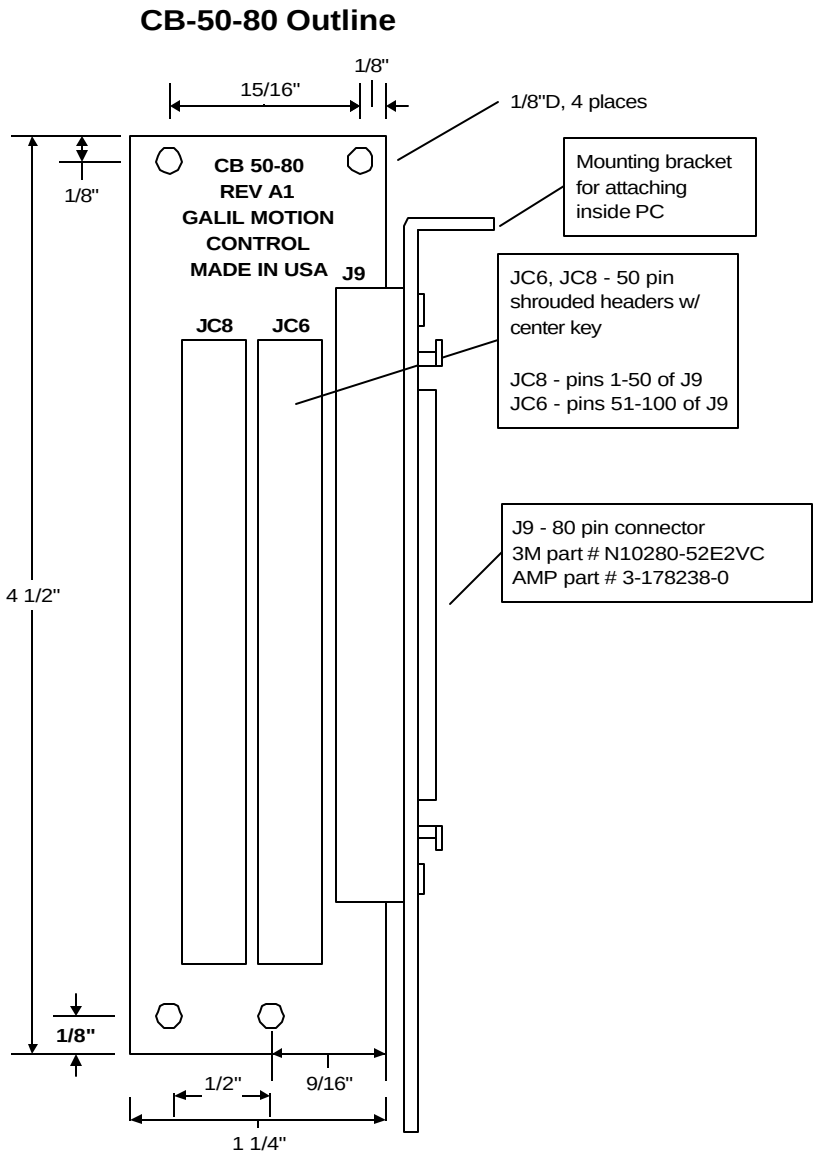


Figure A-17

CB-50-80 Layout

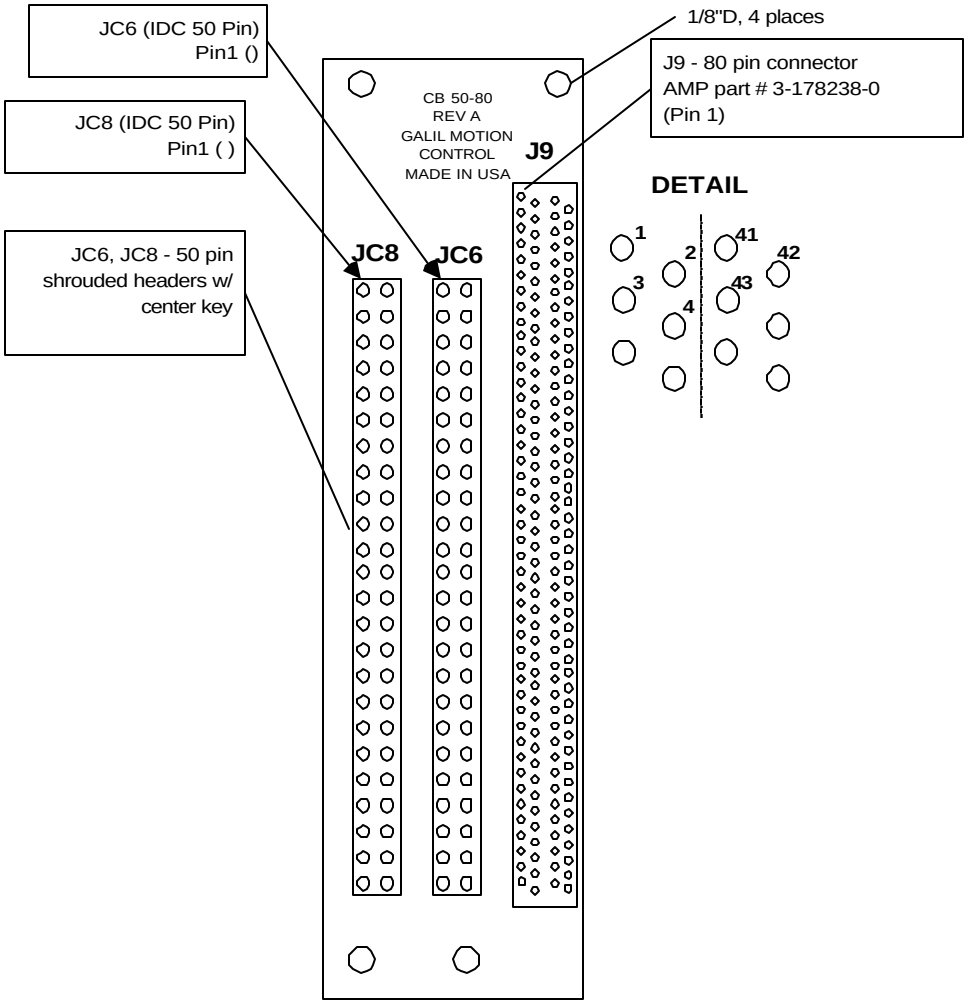


Figure A-18

TERM-1500 Operator Terminal

Two types of terminals are offered from Galil; the hand-held unit and the panel mount unit. Both have the same programming characteristics.

Hand held unit is shown below:

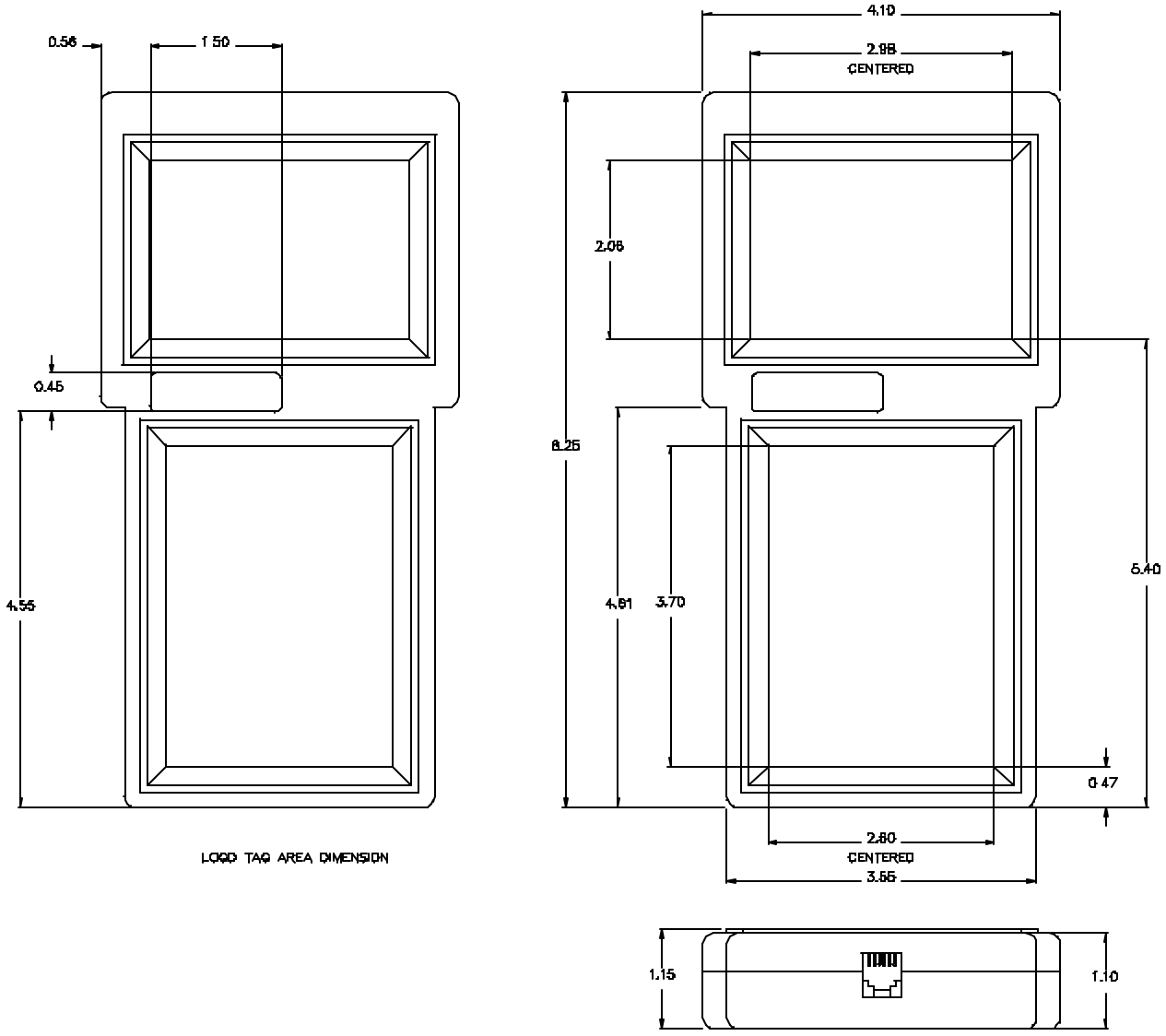


Figure A-19

The panel mount terminal is shown below:

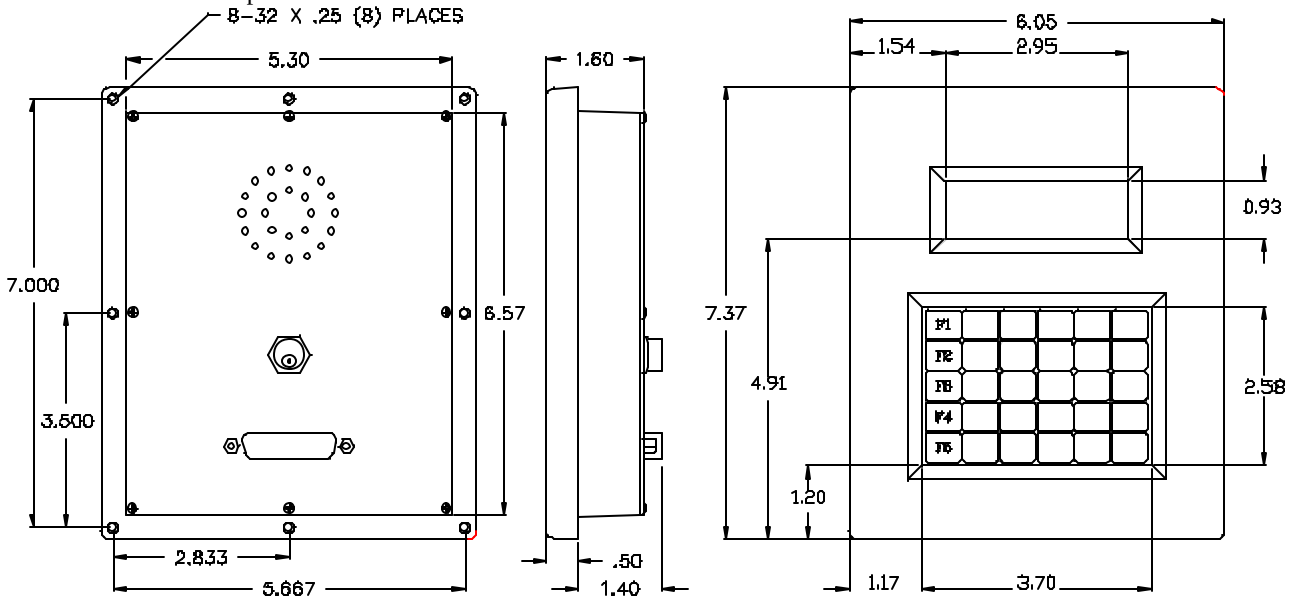


Figure A-20

Features

- For easy data entry to DMC-2x00 motion controller
- 4 line x 20 character Liquid Crystal Display
- Full numeric keypad
- Five programmable function keys
- Available in Hand-held or Panel Mount
- No external power supply required
- Connects directly to RS232 port P2 via coiled cable

Description

The TERM-2000 is a compact ASCII terminal for use with the DMC-2x00 motion controller. Its numeric keypad allows easy data entry from an operator. The TERM-1500 is available with a male adapter for connection to P2 (Dataset).

NOTE: Since the TERM-1500 requires +5V on pin 9 of RS-232, it can only work with port 2 of the DMC-2x00.

Specifications - Hand-Held

Keypad	Key Tactile 4 row x 5character
Display	LCD with 5 by 7 character font
Power	5 Volts, 30mA (from DMC-2x00)

Specifications - Panel Mount

Keypad	30-Key; 5 rows x 6 columns ; 5x7 font
Display	4 row x 20 character LCD
Power	5 Volts, 30mA

Keypad Maps - Hand-Held

30 Keys: 5 keys across, 6 down

Single Key Output

6	F1 (22)	F2 (23)	F3 (24)	F4 (25)	F5 (26)
5		1	2	3	
4		4	5	6	
3		7	8	9	
2			0		
1	CTRL	SHIFT	SPACE	BKSPC	ENTER

Shift Key Output

6	A	B	C	D	E
5	F	G	H	I	J
4	K	L	M	N	O
3	P	Q	R	S	T
2	U	V	W	X	Y
1	CTRL	SHIFT	Z	,	?

CTRL Key Output

6	(18)	(16)	(9)	(4)	(17)
5	(19)	(2)	!	“	%
4	*	+	/	$	;
3	<	>	\	[	]
2	^	-	@	{	}
1	CTRL	SHIFT	ESC	=	#

NOTE: Values in parentheses are ASCII decimal values. Key locations are represented by [m,n] where m is element column, n is element row.

Example:

U is <Xhift>[1,2]

is <Cntrl>[5,1]

Keypad Map - Panel Mount – 6 columns x 5 rows

Single Key Output

5	F1		1	2	3	
4	F2		4	5	6	
3	F3		7	8	9	
2	F4		-	0	.	
1	F5	CTRL	SHIFT	SPACE	BKSPC	ENTER

Shift Key Output

5	A	F	G	H	I	J
4	B	K	L	M	N	O
3	C	P	Q	R	S	T
2	D	U	V	W	X	Y
1	E	CTRL	SHIFT	Z	,	?

CTRL Key Output

5	(18)	(19)	(2)	!	“	%
4	(16)	*	+	/	$	;
3	(9)	<	>	\	[	]
2	(4)	^	-	@	{	}
1	(17)	CTRL	SHIFT	ESC	=	#

NOTE: Values in parentheses are ASCII decimal values. Key locations are represented by [m,n] where m is element column, n is element row.

Escape Commands

Escape codes can be used to control the TERM-1500 display, cursor style, and position, and sound settings.

NOTE: The escape character (hex 1B) can be sent through port 2 of the DMC-2x00 with special syntax {^27}:

Example: MG {P2}{^27},”H” Sends escape H to the terminal from port 2

Cursor Movement Commands

ESC A	Cursor Up
ESC B	Cursor Down
ESC C	Cursor Right
ESC D	Cursor Left

Erasing Display

ESC E	Clear Display and Home
ESC I	Clear Display
ESC J	Cursor to End of Display
ESC K	Cursor to End of Line
ESC M	Line Containing Cursor

Sounds

ESC T	Short Bell
ESC L	Long Bell
ESC P	Click
ESC Q	Alert

Cursor Style

ESC F	Underscore Cursor On
ESC G	Underscore Cursor Off
ESC R	Blinking Cursor On
ESC S	Blinking Cursor Off

Key Clicks (audible sounds from terminal)

ESC U	Key Click Enable
ESC V	Key Click Disable

Identify (sends "TT!" then terminal firmware version)

ESC Z	Send Terminal ID

Cursor Position

ESC Y	Pr Pc

In the above sequence, Pr is the row number and Pc is te column number of the target cursor location. These parameters are formed by adding hexadecimal 1F to the row and column nubmers. Row and column numbers are absolute, with row 1, column 1 (Pr = H20, Pc = H20) representing the upper left corner of the display.

Configuration

<CNTRL><SHIFT>F1 Allows user to configure terminal; Follow prompts on display to change configuration

Default Configuration:

Baud Rate	9600
Data bits	7
Parity	Ignore PE
Display	enabled
Repeat	Fast
Echo	Disabled
Handshake	Disabled
Self Test	Disabled
Key Click - Disabled	<Ctrl>Space <Shift> [2,2]
Key Click - Enabled	<Ctrl>Space <Shift> [1,2]
Clear Display and Home	<Ctrl>Space <Shift> [5,6]

Function Keys

<CNTRL><SHIFT>F3 Allows function keys to be configured; Follow prompts on display to change function keys

Default Function Keys

F1	22 decimal
F2	23 decimal
F3	24 decimal
F4	25 decimal
F5	26 decimal

Input/Output of Data – DMC-2x00 Commands

Refer to Chapter 7 in this manual for Data Communication commands.

When using Port 2, use CC command to configure P2.

Example:

CC 9600,0,0,1	Configures P2
MG{P2} “Hello There”, V1{F2.1}	Send message to P2
IN{P2} “Enter Value”, NUM	Prompts operator for value

Example:

#A	
CI 0;CC 9600,0,0,1	#A Interrupt on any key; Configure P2
MG {P2} “press F1 to start X”	Print Message to P2
MG {P2} “Press F2 to start Y”	Print Message to P2

#B; JP#B;EN	End Program
#COMINT	Interrupt Routine
JS #XMOVE,P2CH=F1	Jump to X move if F1
JS #YMOVE,P2CH=F2	Jump to Y move if F2
EN1,1	End, Re-enable comm interrupt & restore trippoint
#XMOVE;PR1000;BGX;EN	Move X routine
#YMOVE;PR,1000;BGY;EN	Move Y routine

NOTE: F1 through F5 are used as dedicated keywords for testing function keys. Do not use these as variables.

6-Pin Modular Connector

1	+5 Volts
2	Handshake in
3	Handshake out
4	Data in
5	Data out
6	Ground

9-Pin D Adaptor - Male (For P2)

1	CTS input
2	Transmit Data - input
3	Receive Data - output
4	RTS - output
5	Ground
6	CTS - input
	RTS - output
	CTS - input
	5V or no connect or sample clock with jumpers

NOTE: Out and in are referenced to the terminal.

Ordering Information

TERM-1500H-P2	Hand-held terminal with female adapter
TERM-1500P-P2	Panel Mount terminal with female adapter

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 $1/256^{th}$ of a degree. For example, the path shown in Fig. 12.2 is specified by the instructions:

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

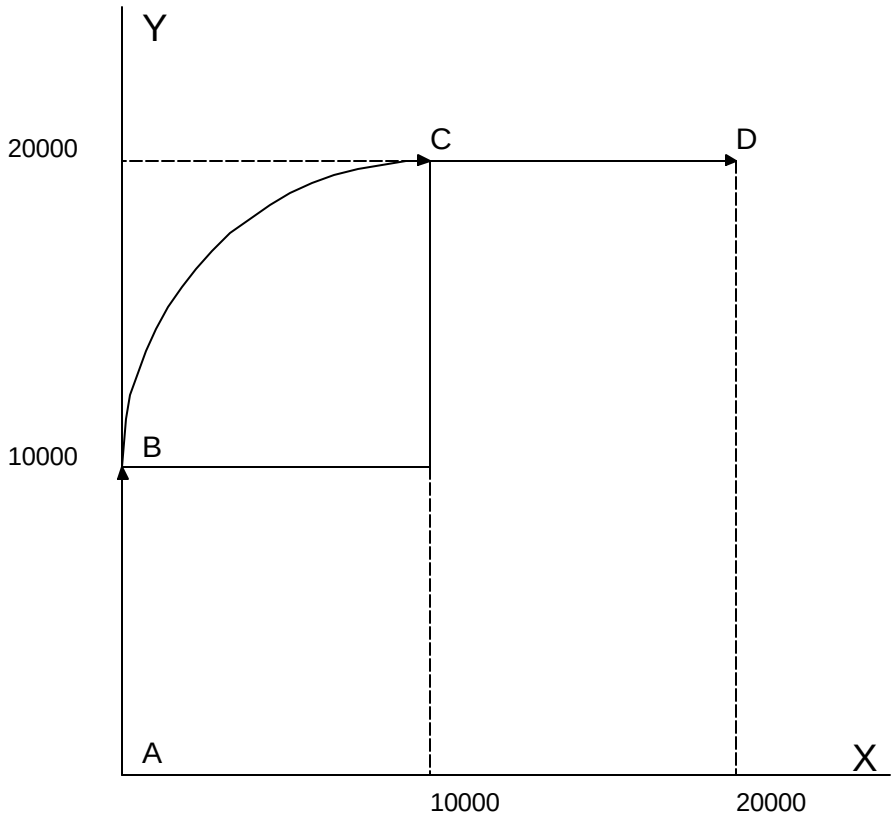


Figure A-21 - X-Y Motion Path

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	$\frac{R\lvert\Delta q\rvert 2p}{360}$ = 15708
C-D	Linear	1000
	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

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

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. 12.2 may be specified in terms of the vector speed and acceleration.

VS 100000

VA 2000000

The resulting vector velocity is shown in Fig. 12.3.

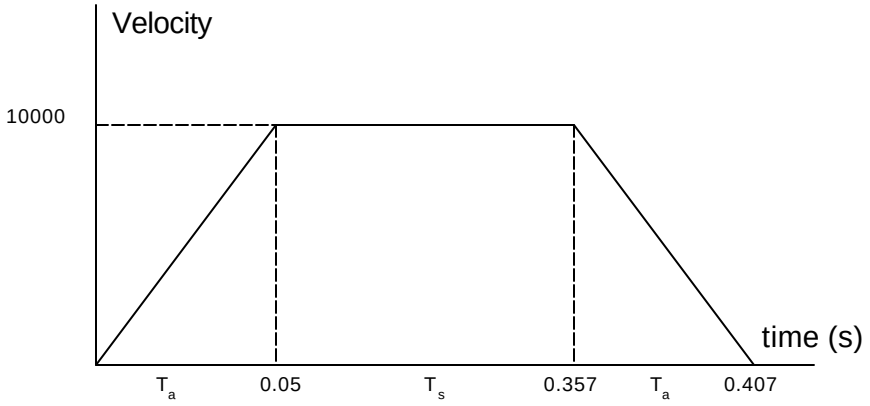


Figure A-22 - Vector Velocity Profile

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. 12.2 are given in Fig. 12.4.

Fig. 12.4a 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.

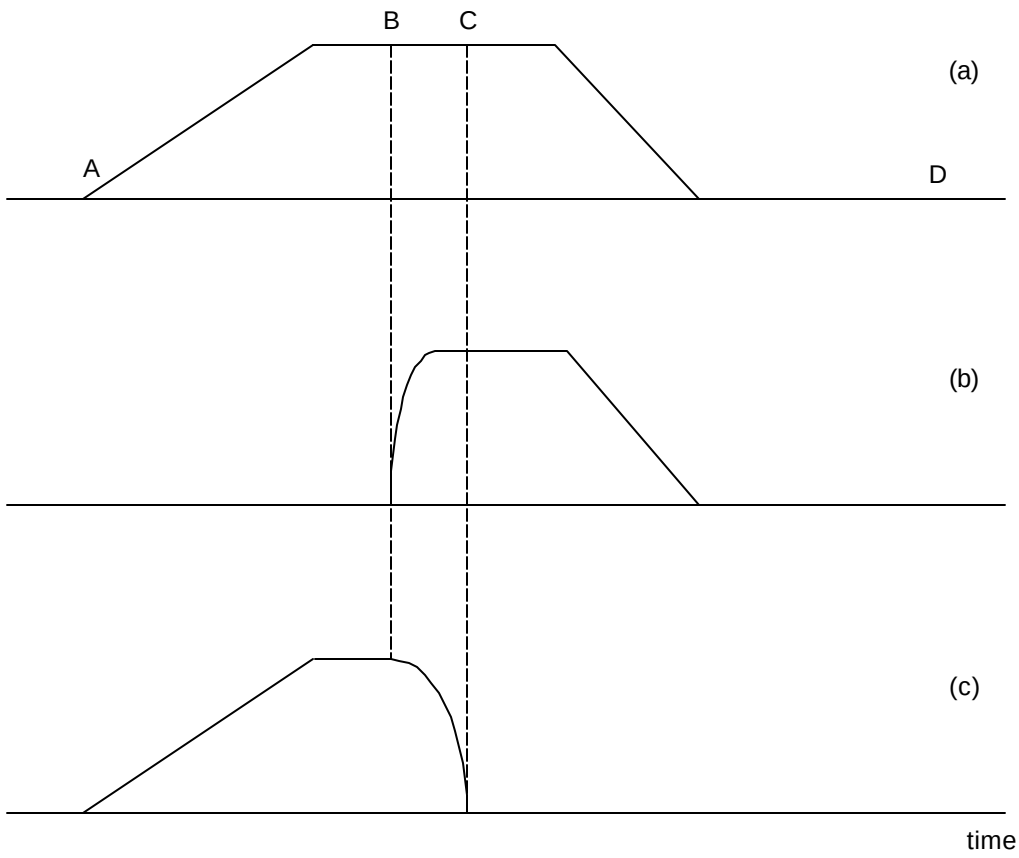


Figure A-23 - Vector and Axes Velocities

Example- Communicating with OPTO-22 SNAP-B3000-ENET

Controller is connected to OPTO-22 via handle F. The OPTO-22's IP address is 131.29.50.30. The Rack has the following configuration:

Digital Inputs	Module 1
Digital Outputs	Module 2
Analog Outputs (+/-10V)	Module 3
Analog Inputs (+/-10V)	Module 4

Instruction	Interpretation
#CONFIG	Label
IHF=131,29,50,30<502>2	Establish connection
WT10	Wait 10 milliseconds
JP #CFGERR,_IHF2=0	Jump to subroutine
JS #CFGDOUT	Configure digital outputs
JS #CFGAOUT	Configure analog outputs
JS #CFGAIN	Configure analog inputs
MBF = 6,6,1025,1	Save configuration to OPTO-22

```
EN                                  End

#CFGDOUT                            Label
MODULE=2                            Set variable
CFGVALUE=$180                       Set variable
NUMOFIO=4                           Set variable
JP #CFGJOIN                         Jump to subroutine

#CFGAOUT                            Label
MODULE=3                            Set variable
CFGVALUE=$A7                        Set variable
NUMOFIO=2                           Set variable
JP #CFGJOIN                         Jump to subroutine

#CFGAIN                             Label
MODULE=5                            Set variable
CFGVALUE=12                         Set variable
NUMOFIO=2                           Set variable
JP#CFGJOIN                          Jump to subroutine

#CFGJOIN                            Label
DM A[8]                             Dimension array
I=0                                 Set variable
#CFGLOOP                            Loop subroutine
A[I]=0                              Set array element
I=I+1                               Increment
A[I]=CFGVALUE                       Set array element
I=I+1                               Increment
JP #CFGLOOP,I<(2*NUMOFIO)           Conditional statement
MBF=6,16,632+(MODULE*8),NU          Configure I/O using Modbus function code 16 where the starting
MOFIO*2,A[]                         register is 632+(MODULE*8), number of registers is
                                    NUMOFIO*2 and A[] contains the data.
EN                                  end

#CFERR                              Label
MG"UNABLE TO ESTABLISH              Message
CONNECTION"
EN                                  End
```

Using the equation

I/O number = (Handlenum*1000) + ((Module-1)*4) + (Bitnum-1)

MG @IN[6001] display level of input at handle 6, module 1, bit 2

SB 6006 set bit of output at handle 6, module 2, bit 3

or to one

OB 6006,1

AO 608,3.6 set analog output at handle 6, module 53, bit 1 to 3.6 volts

MG @AN[6017] display voltage value of analog input at handle6, module 5, bit 2

DMC-2x00/DMC-1500 Comparison

BENEFIT	DMC-2x00	DMC-1500
Access to parameters – real time data processing & recording	Data Record - Block Data Transfer	No DMA channel
Easy to install – USB is self configuring	Plug and Play	USB not available
Can capture and save array data	Variable storage	Option
Parameters can be stored	Array storage	Option
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
Higher resolution for analog inputs	8 analog inputs with 16-bit ADC option	7 inputs with 16-Bit option
Improved EMI	100-pin high density connector	60-pin IDC, 26-pin IDC, 20-pin IDC (x2)
For precise registration applications	Output Position Compare	Available as a special
More flexible gearing	Multiple masters allowed in gearing mode	One master for gearing
Binary command mode	Binary and ASCII communication modes	ASCII only
Gearing	Multiple Gearing Masters Accepted	Single Gearing Master Accepted
Coordinated Motion	2 Sets of Coordinated Motion Accepted	Single set of coordinated motion only

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 their market insights in a no-nonsense way for a world of engineers on the move. In fact, over 10,000 engineers have attended Galil seminars. The tradition continues with three different seminar, each designed for your particular skillset-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

3750 Atherton Road

Rocklin, CA 95765

Phone: 916-626-0101

Fax: 916-626-0102

E-Mail Address: support@galilmc.com

URL: www.galilmc.com

FTP: www.galilmc.com/ftp

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.

COPYRIGHT (3-97)

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

Index

Abort 37–38, 73, 79, 155, 157, 186–87
  Off-On-Error 21, 38, 41, 155, 157
  Stop Motion..... 73, 79, 124, 158
Absolute Position.....68–70, 115–16, 120
Absolute Value..... 86, 120, 129, 156
Acceleration.....117, 136, 143, 146–51, 231–32
Accessories 189
Address..... 133–34, 190–97, 190–97, 238
  Jumpers..... 45
Ampflier Gain 5
Amplifier Enable 41, 155
Amplifier Gain 165, 170, 172
Analog Input…….. .....4, 37, 41, 72, 129–31, 132, 139, 146–47, 153, 175
Analysis
  SDK 107
Arithmetic Functions..... 107, 119, 127, 130, 143
Arm Latch 106
Array .....…….3, 68, 78, 92–94, 107, 113, 119, 127, 131–42, 144, 177
Automatic Subroutine 109, 110, 122
  CMDERR..... 110, 123, 125
  LIMSWI.....37, 110, 122–23, 156–58
  MCTIME..... 110, 115, 123, 125
  POSERR.....110, 122–24, 156–57
Auxiliary Encoder..... …….37, 84, 96–100, 96–100, 96–100, 187, 192, 197, 199
  Dual Encoder..... 64, 100, 134
Backlash 68, 98–100, 153–54
Backlash Compensation
  Dual Loop 68, 96–100, 96–100, 96–100, 153
Baud Rate 18, 46
Begin Motion..... …….109–12, 116–17, 123–24, 131, 135–36, 143, 146
Binary..... 1, 59, 62
Bit-Wise..... 119, 127
Burn
  EEPROM 3
Bypassing Optoisolation..... 41
Capture Data
  Record 68, 92, 94, 132, 134
Circle 150–51
Circular Interpolation.....78–81, 83, 133, 150–51
Clear Bit 143
Clear Sequence..... 73, 75, 79, 81
  Electronic Cam..... 67–68, 85, 87
Echo 47–52, 57
Edit Mode..... 114, 123
Editor..... 107–8
EEPROM 3, 14, 15
Electronic Cam..... 67–68, 85, 87
Electronic Gearing..... 67–68, 83–85
Ellipse Scale 81
Enable
  Amplifer Enable 41, 155
Encoder
  Auxiliary Encoder……, 96–100, 96–100, 96–100, 187, 192, 197, 199
  Differential..... 22, 24, 160
  Dual Encoder..... 64, 100, 134
  Index Pulse..... 22, 38, 103
  Quadrature 5, 98, 143, 149, 156, 167
Error Code..... 123, 131, 134–35, 149–50, 152–54
Error Handling.....37, 110, 122–23, 156–58
Error Limit 21, 23, 41, 123, 155–57
  Off-On-Error 21, 38, 41, 155, 157
Example
  Wire Cutter..... 149
Feedrate 74, 80, 117, 150–51
FIFO..... 57
Filter Parameter
  Damping..... 160, 164
  Gain 131, 139
  Integrator..... 164
  PID 24, 164, 174
  Proportional Gain 164
  Stability 99–100, 154, 159–60, 164, 171
Find Edge 38, 103
Formatting.....139, 140–42
Frequency.....5, 102, 171–73
Function…….....38, 59, 73, 92–93, 100–101, 105, 107, 111–15, 117, 119, 123, 127–32, 139–40

Functions
Arithmetic 107, 119, 127, 130, 143
Gain……131, 139
Proportional 164
Gear Ratio 83–84
Gearing 67–68, 83–85
Halt……74, 111–15, 117–18
Abort 37–38, 73, 79, 155, 157, 186–87
Off-On-Error 21, 38, 41, 155, 157
Stop Motion 73, 79, 124, 158
Hardware 37, 143, 155
Address 133–34, 190–97, 190–97, 238
Amplifier Enable 41, 155
Clear Bit 143
Jumper 41, 160
Offset Adjustment 159
Output of Data 138
Set Bit 143
TTL 5, 37, 41, 155
Home Input 38, 103, 132
Homing 38, 103
Find Edge 38, 103
I/O
Amplifier Enable 41, 155
Analog Input 72
Clear Bit 143
Digital Input 37, 39, 129, 144
Digital Output 129, 143
Home Input 38, 103, 132
Output of Data 138
Set Bit 143
TTL 5, 37, 41, 155
ICM-1100 ..... 21, 41, 155
Independent Motion
Jog……..71–72, 83, 89, 106, 116–17, 123–25, 131, 153, 157
Index Pulse 22, 38, 103
ININT 110, 123–24, 145–46
Input
Analog 72
Input Interrupt 109, 117, 123–24, 145–46
ININT 110, 123–24, 145–46
Input of Data 135
Inputs
Analog……4, 37, 41, 129–31, 132, 139, 146–47, 153, 175
Interconnect Module 17
Installation 159
Integrator 164
Interconnect Module 17
ICM-1100 ..... 21, 41, 155
Interface
Terminal 59
Internal Variable 119, 130
Interrogation 64–65, 75, 82, 138, 140
Interrupt 109–11, 117, 122–24, 145–46
Invert 98, 160
Jog……..... 71–72, 83, 89, 106, 116–17, 123–25, 131, 153, 157
Joystick 72, 131, 152–53
Jumper 41, 160
Jumpers 45
Keyword 119, 127, 130, 131–32
TIME 132–33
Label……..41, 72–74, 78, 88–89, 100, 103, 106, 107–13, 115–24, 131, 136, 140, 143–47, 151, 153–54, 157
LIMSWI 156–58
POSERR 156–57
Special Label 109, 158
Latch 64, 105
Arm Latch 106
Data Capture 133–34
Position Capture 105
Record 68, 92, 94, 132, 134
Teach 94
Limit
Torque Limit 23
Limit Switch…….. ... 37–38, 109–11, 122–23, 132, 156–58, 160
LIMSWI 37, 110, 122–23, 156–58
Linear Interpolation 67, 73–74, 77, 83, 91
Clear Sequence 73, 75, 79, 81
Logical Operator 118
Logical Operators 136
Masking
Bit-Wise 119, 127
Math Function
Absolute Value 86, 120, 129, 156
Bit-Wise 119, 127
Cosine 68, 127–29, 133
Logical Operator 118
Sine 68, 88, 129
Math Functions
Logical Operators 136
Mathematical Expression 119, 127, 129
MCTIME 110, 115, 123, 125
Memory 59, 93, 107, 113, 118, 123, 131, 133
Array……..... 3, 68, 78, 92–94, 107, 113, 119, 127, 131–42, 144, 177
Download 59, 107, 133
Upload 107
Message……..78, 103, 113, 123–25, 128, 135–39, 146, 157–58
Messages 46
Modelling 161, 164–65, 169
Motion Complete
MCTIME 110, 115, 123, 125
Motion Smoothing 68, 101, 102
S-Curve 74, 101
Motor Command 24, 169

Moving
Acceleration....117, 136, 143, 146–51, 231–32
Begin Motion..... ……..109–12, 116–17, 123–24, 131, 135–36, 143, 146
Circular....78–81, 83, 133, 150–51
Multitasking....111
Halt....74, 111–15, 117–18
OE
Off-On-Error....155, 157
Off-On-Error....21, 38, 41, 155, 157
Offset Adjustment....159
Operand
Internal Variable....119, 130
Operators
Bit-Wise....119, 127
Optoisolation....37, 39–40
Home Input....38, 103, 132
Output
Amplifier Enable....41, 155
ICM-1100....21, 41
Motor Command....24, 169
Output of Data....138
Clear Bit....143
Set Bit....143
Outputs....45
Interconnect Module....17
PID……. 24, 164, 174
Play Back....68, 135
POSERR....110, 122–24, 156–57
Position Error……. 23, 110, 122–24, 131, 133–34, 147, 154
Position Capture....105
Latch....64, 105
Teach....94
Position Error……. 21, 23, 41, 100, 110, 122–24, 131, 133–34, 147, 154, 155–57, 160, 163
POSERR....110, 122–24
Position Follow....146–47
Position Limit....157
Program Flow....109, 114
Interrupt....109–11, 117, 122–24, 145–46
Stack....122, 125, 146
Programmable....130–31, 143, 153, 156
EEPROM....3
Programming
Halt....74, 111–15, 117–18
Proportional Gain....164
Protection
Error Limit....21, 23, 41, 123, 155–57
Torque Limit....23
PWM....5
Quadrature....5, 98, 143, 149, 156, 167
Quit
Abort....37–38, 73, 79, 155, 157, 186–87
Stop Motion....73, 79, 124, 158
Record....68, 92, 94, 132, 134
Latch....64, 105
Position Capture....105
Teach....94
Register....131
Reset....37, 118, 155, 157
SB
Set Bit....143
Scaling
Ellipse Scale....81
S-Curve....74, 101
Motion Smoothing....68, 101, 102
SDK……10 7
Selecting Address....133–34, 190–97, 190–97, 238
Serial Port....17, 18
Servo Design Kit
SDK....107
Set Bit....143
Sine……. 68, 88, 129
Single-Ended....5, 22, 24
Slew…… 68, 84, 103, 115, 117, 149
Smoothing....68, 74, 75, 79, 81, 101–2
Software
SDK....107
Terminal....59
Special Label....109, 158
Specification....74, 80
Stability....99–100, 154, 159–60, 164, 171
Stack....122, 125, 146
Zero Stack....125, 146
Status....59, 64, 75, 113–14, 131, 134
Interrogation....64–65, 75, 82, 138, 140
Stop Code....64, 134, 160
Tell Code....63
Step Motor....102
KS, Smoothing....68, 74, 75, 79, 81, 101–2
Stop
Abort....37–38, 73, 79, 155, 157, 186–87
Stop Code……. 64, 123, 131, 134–35, 134, 149–50, 152–54, 160
Stop Motion....73, 79, 124, 158
Subroutine....37, 78, 110, 118–24, 145–46, 156–57
Automatic Subroutine....109, 110, 122
Synchronization....5, 45, 85
Syntax....59–60
Tangent....68, 78, 80
Teach....94
Data Capture....133–34
Latch....64, 105
Play-Back....68, 135
Position Capture....105
Record....68, 92, 94, 132, 134
Tell Code....63
Tell Error....64
Position Error……. 23, 110, 122–24, 131, 133–34, 147, 154

Tell Position 57, 64
Tell Torque 64
Term-1500 45
Terminal 37, 41, 59, 107, 130, 140
Theory 161
  Damping 160, 164
  Digital Filter 59, 168–70, 172–74
  Modelling 161, 164–65, 169
  PID 24, 164, 174
  Stability 99–100, 154, 159–60, 164, 171
Time
  Clock 132
TIME 132–33
Time Interval 90–92, 94, 133
Timeout 17, 110, 115, 123, 124
  MCTIME 110, 115, 123, 125
Torque Limit 23
Trigger 107, 114, 163
Trippoint 69, 74–75, 80–81, 92, 115–16, 121, 122
Troubleshooting 159
TTL 5, 37, 41, 155
Tuning
  SDK 107
  Stability 99–100, 154, 159–60, 164, 171
Upload 107
User Unit 143
Variable
  Internal 119, 130
Vector Acceleration 75–76, 80, 151
Vector Deceleration 75–76, 80
Vector Mode
  Circle 150–51
  Circular Interpolation 78–81, 83, 133, 150–51
  Clear Sequence 73, 75, 79, 81
  Ellipse Scale 81
  Feedrate 74, 80, 117, 150–51
  Tangent 68, 78, 80
Vector Speed 73–79, 80, 117, 151
Wire Cutter 149
Zero Stack 125, 146