KOLLMORGEN®

*Because Motion Matters™*

# This is a Discontinued Product

Contact Kollmorgen Customer Support at 1-540-633-3545 or email us at support.kollmorgen.com if assistance is required.

PRICE: $50.00

# SLO-SYN® MODEL MX2000 PROGRAMMABLE MULTI-AXIS MOTION CONTROLLER (with VERSION 4.0 SOFTWARE) INSTALLATION AND OPERATION MANUAL

## ENGINEERING CHANGES

Superior Electric reserves the right to make engineering refinements on all its products. Such refinements may affect information given in instructions, Therefore, **USE ONLY THE INSTRUCTIONS THAT ARE PACKED WITH THE PRODUCT.**

| RECORD OF REVISION | | |
|---|---|---|
| Revision | Date | Description |
| A | 4/30/99 | Initial Release |
| B | 6/08/00 | Revise corporate identity |
| | | |

The MX2000-2 and MX2000-6 are UL recognized components, File No. E146240.

# Table of Contents

Following CONTINUED

# List of Illustrations

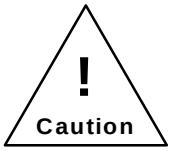
# Section 1
# Important
# Safety Information

## 1.1 – Cautions and Warnings

Before installing and operating your MX2000 motion control product, it is extremely important both to you and us that you read this section very
thoroughly and carefully. Your Slo-Syn product will deliver years of reliable, trouble-free, and most importantly, safe operation if you heed the cautions and warnings outlined in this section, and follow the subsequent instructions in the remainder of this manual.

Throughout this section, and the remainder of this manual, two very important symbols will be used to identify hazardous and potentially dangerous situations. These symbols are the electrical shock indicator and the exclamation point. Both are always surrounded by a triangle as shown.


**The electrical shock symbol shown to the left is used to indicate situations where ELECTRICAL SHOCK hazards may exist. These warnings must be followed to ensure that YOU avoid electrocution that could result in serious injury or death.**

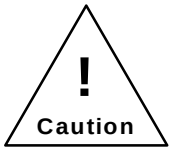



**The exclamation point symbol shown to the left is used to indicate situations other than electrical hazards that may be potentially dangerous to either YOU or to the product. Follow these warnings carefully to avoid injury to you and damage to the product.**

The following is a partial list of precautions that must be followed to ensure safe operation of the unit. Other more specific precautions are indicated in the appropriate sections of this manual. As you read through the manual, pay particularly close attention to these cautions and warnings as they could **save your life**.

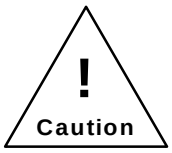

**High voltages are present inside this unit. An Electrical shock hazard exist that may cause serious injury or death if this unit is operated without its protective cover in place.**


**Be certain the power has been removed for a minimum of 5 minutes before any service work or circuit board configuration changes are performed. This assures that the power supplies are at zero.**

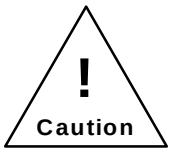



**Do not exceed the voltage or current ratings of the various inputs and outputs; Please read the electrical specification in Section 5. This will protect the circuitry and components from accidental damage.**

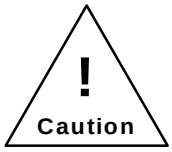

**In order to provide the correct level of protection in the unit, replacement fuses must be the same exact style and ratings as those originally installed in the unit.**



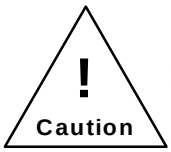

**Secure mounting and proper grounding of both the MX2000 controller and the motors are essential for proper operation of the system.**



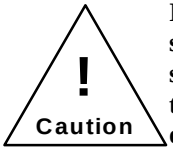

**Be sure to mount the unit so there is adequate space around it for cooling airflow, and observe the environmental limitations for temperature and humidity.**



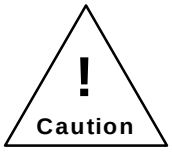

**The 24-volt dc power supply is limited to a total current output of 0.75 amperes. Do not exceed this rating, or the Controller may shut down or work erratically as the power supply's current limiting circuitry operates to protect the unit from overload.**



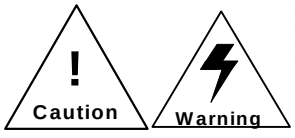

**Please follow good wiring practices and keep low-level signal lines away from power and motor wiring. It is best to use shielded, twisted-pair cables for signal lines, being sure to ground the shields at one end. Doing this will help to avoid electrical noise interference problems.**



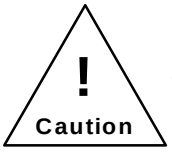

**If the unit is opened or disassembled, be sure to treat the circuit cards as static-sensitive components to avoid damage due to electrostatic discharge (ESD). Work only in ESD protected areas, and it is best not to touch the circuit conductors or components unless you are wearing an ESD-protective grounding strap.**




**It is your responsibility to follow the appropriate federal, state, and local electrical and occupational safety codes in the application of this product.**




**NEVER wire the unit with the power on! Serious injury as well as damage to the unit may result.**




**NONE of the inputs to the unit are to be used as an EMERGENCY STOP in ANY application. Although activation of certain inputs will discontinue motion or disable motor current, these are NOT designed as fail-safe E-STOP inputs. Relying exclusively on inputs to the unit to cease motion that could cause dangerous conditions is a violation of Machine Safety Code (ref, IEC204-1). Other measures such as mechanical stops and fail-safe brakes must be used in these situations.**

# Section 2
# Introduction

## 2.1 - How To Use This Manual

Congratulations on the purchase of your new MX2000 motion control product! Your programmable motion controller is a full-featured and flexible product, yet it is fairly simple to apply it to your machine control application. This manual is designed to guide and assist you through the installation, programming, and operation of the controller. If you're reading this, you understand the importance of familiarizing yourself with how this product should be installed and operated. We strongly recommend that you read through this manual until you are comfortable with electrical connections and operating concepts of this unit.

Section 1, **Important Safety Information**, has cautions and warnings information. This section should be read first and the cautions and warnings should be followed.

Section 2, **Introduction**, has user prerequisite requirements, conventions used in the manual and Applications Assistance information.

Section 3, **Quick Start Installation Guide**, contains the minimum steps necessary to get up and running. The references to the appropriate manual sections where further details can be found are included.

Section 4, **Overview of System Operation**, contains features and functions, along with a general overview of the MX2000 system.

Section 5, **Specification and Equivalent Circuits**, has specifications, setup requirements, connection diagrams, and equivalent circuits for each board in the MX2000 system.

Section 6, **Motion Control Programming Interface**, contains general Programming information. PC software installation and execution, communications with the MX2000 controller, and user project creation.

Section 7, **Software Reference Guide**, contains the basic command conventions used and a listing with descriptions of the Basic Program commands and Host Commands.

Section 8, **Following**, contains detailed information on following, description of follower commands, application examples, and listing and descriptions of the individual follower commands.

Section 9, **Servo Drive**, contains general information on servos, tuning, testing performance. Also a listing and descriptions of the individual servo commands.

Section 10, **Stepper Drive**, contains general information on stepper drives, closed and open loop stepper setups. Also a listing and descriptions of the individual stepper commands.

Section 11, **Data Logging**, describes how to data log MX2000 parameters.

Section 12, **Debug Environment**, describes how to debug a user's task.

Section 13, **Application Examples**, contains descriptions of applications with example programs for them.
Section 14, **Troubleshooting Guide**, has helpful hints on troubleshooting problems.

Section 15, **Glossary**, contains a glossary of terms used in the manual.

## 2.2 – What you need to know first

This manual is written in a simple and easy to follow format that should be suitable for both new and experienced motion control users. In order to get the most out of your SLO-SYN Programmable Motion Controller, we assume the user will be knowledgeable in the following areas:

Basic electrical and electronics skills, including preparing and following an equipment wiring diagram or schematic.
The basics of motion control system applications, such as torque, speed, move distance, and how to structure a motion task into move segments and input/output control.
Some familiarity with elementary computer programming, including defining the problem to be solved and coding it in a computer language.

## 2.3 – Conventions used in this manual

Motor rotation direction (CW and CCW) is properly oriented when viewing the motor from the end opposite the mounting flange.

Please refer to the Glossary section for detailed descriptions of terms such as sink and source I/O, various motion terms, etc.

## 2.4 - Applications Assistance

Although this manual represents a detailed compilation of information regarding your SLO-SYN control product, sometimes questions may arise which will require that you contact us, You now have a few options available to you when you need information regarding your product or its application.

**On the Internet at www.danahermotion.com.** Our multimedia enabled web site offers you information such as:
Free Software
TechFax fax on demand documents (1-800-234-3369)
HTML Product Selector, HTML Brand Selector
Product News and Links
Sales and Distribution Information
Product information and specifications
Many more features

2. **By Phone**. You may reach us by phoning our Motion Control Application Engineering Department at telephone (800)787-3532 ext. 4751. Or call our main number at (860)585-4510. Both may be reached between the hours of 8:00 AM and 5:00 PM (Eastern Time), Monday through Friday. Technical personnel are available to assist you in getting your application up and running.

# Section 3
# Quick Start Installation Guide

## 3.1 – Switch and Jumper Settings

Before mounting and installing the MX controller, it is best to set the switches and internal jumpers that govern various operating features.

## 3.1.1 – Serial Communication Baud Rate switches

The "BAUD" DIP switch located on the 32 bit DSP controller panel needs to be set to match the baud rate of the host computer or terminal to which it is connected. The factory default is 9600 baud; if this is not what is desired, then set the switches toward one of the appropriate values shown on the label. Valid selections are "9.6" (9600), "19.2" (19,200), and "38.4" (38,400). If all switches are "off" (toward the right), then the baud rate is set to 4800. These switches are only read at power-up, hence changing the baud rate requires a power-down, power-up cycle before the change takes effect.

Although the controller's serial ports are configurable for up to 38.4K baud, the serial communications may be limited by the PC. A PC may not be able to receive data from the controller at baud rates above 9600. This limitation is due to the PC's inability, at the higher baud rates, to read the received character in time, before another character is received. If this happens, an OVERRUN ERROR will occur. This problem will not exist if the serial port's UART has hardware buffering. The following is a list of UART' s commonly used on PC serial port cards. The UART' s marked with an * are buffered.

UART' s: 8250, 16450, 16451, 16452, *16550, *16552

The following is a table of controller operations vs. Maximum PC receiver baud rate.

| OPERATION | UART (no buffer) | UART (buffer) |
|---|---|---|
| Load operating system | 38400* | 38400 |
| Load user program | 38400* | 38400 |
| Extract source code | 9600 | 38400 |
| Host commands | 9600 | 38400 |

* The unbuffered UART will perform the first two operations at the higher baud rate, since during these operations, the controller does not transmit multiple characters in succession.

## 3.1.2 – Host RS232/RS485

The RS232/RS485 switch located on the 32 bit DSP controller panel needs to be set to match the communication protocol of the host device, RS-232 or RS-485, to which the Controller is connected. The factory default is for "232" (RS-232); if RS-485 is desired, set the switch toward "485".

Serial communications format for the host port is "N-8-1", or No parity, 8 data bits, and 1 stop bit.

## 3.1.3 – Auxiliary RS232/RS485

The communication protocol (RS232 or RS485) for the Auxiliary Serial Port on the 32 bit DSP Card is selected via a jumper on the card, immediately behind the port connector. The factory default is for RS485; if RS232 is desired, the DSP Card must be removed and the jumper setting changed to the RS232 setting. Baud rate for this port is set at 9600; if another Baud rate is desired, select it via software using the "SETCOM" command. See Section 7 for further details on this command.

Serial communications format for the auxiliary port is "N-8-1", or No parity, 8 data bits, and 1 stop bit.

## 3.1.4 – Unit ID switch

The Controller is capable of being operated in a "daisy-chain" fashion, with up to 9 units connected to a single host. A connection diagram is depicted in Section 5.6.2 of this manual. Each unit in the chain requires a unique identification number (ID #); this value is selected by the "UNIT ID" selector switch on the DSP controller board. The unit is shipped from the factory with 1 selected (the first unit in the chain). If needed, set the selector switch to a different value by using a small screwdriver. Set the pointer on the switch to the desired value, 1 to 9.

The controller scans the Unit ID switch during power up or when a system Reset command is issued.

## 3.1.5 – 32 bit DSP Board Inputs

The four optically isolated inputs can either be sinking or sourcing. A jumper on the card controls the selection, located behind the DSP input connector. The factory setting is sinking; if sourcing is desired, the DSP card must be removed and the jumper setting changed to the Source position.

**Hint: A CLR to COM jumper is required for motion to occur**.

## 3.1.6 – Dual Axis Inputs settings

The eight dedicated inputs on the Dual Axis Interface Card can either be sinking or sourcing. A jumper on the card controls the selection, located behind the AXIS I/O connector. The factory setting is sinking; if sourcing is desired, the Dual Axis card must be removed and the jumper setting changed to the Source position.

## 3.1.6.1 – Dual Axis I.D. switches

Each axis card must be assigned a unique ID (1-4) and ID 1 must always be assigned to one of the boards. The axis ID switch settings assign this ID. A table for the different assignments is illustrated in Section 5.5 of this manual.

## 3.1.7 – Digital I/O settings

The digital I/O board inputs and outputs can either be sinking or sourcing. Two jumpers on the card controls the selection, located in the lower left corner of the board. The factory setting is sinking; if sourcing is desired, the Digital I/O card must be removed and the jumpers setting changed to the Source position.

## 3.2 - Step-by-Step Start-Up Procedure

The MX2000 stepper/servo motor positioning system is a sophisticated and versatile product. Setting up the system, however, can be simple and straight-forward if the proper steps are followed. Please use the step-by-step set up guide below.

## 3.2.1 - Bench Set Up.

Before connecting your MX2000 and motors to your mechanical system or machine, we recommend that you "bench test" the system. This will allow you to become familiar with the wiring, programming and operation of the system before installing it into your machine. This may also prevent inadvertent damage to your mechanical system if you make programming errors that cause unexpected motion. The bench set up can be used to perform simple motions with an unloaded motor. To perform a bench test, do the following:

1) **Wire it up**. Connect the servo drives as illustrated in section 5.5.2, connect the stepper drives as illustrated in section 5.5.1, connect the AC power, I/O and other required signals per the wiring diagrams and instructions in section 5. **BE SAFE!!** Do not apply AC power to the unit until you are sure of all connections. Initially, there is no need to connect all of the wiring of your system together. Wire the AC line input, drives, motors and HOST communication ports. This will be all you need to establish communications to the unit and perform simple motion.

   **HINT:** Don't forget to wire the **Enable and Ready** signals to the servo drive, see section 5.5.6 **Analog Drive Connector**.

2) **Load Software.** You will need to use a PC to program the unit according to your requirements. First you must load the MCPI software onto the PC from the floppy disks provided with your unit. Simply insert disk #1 and run the file SETUP.EXE. Once the software is loaded, run it by double clicking on the MCPI icon. See Section 6.3.1 for more details on the MCPI installation process.

3) **Create your Project**. You can now create your new **Project.** Your Project will contain **Configuration** information for your particular system, and also your program **Task's** that holds the user program written in BASIC-like language. Read section 6 of this manual, and then step through the Configuration folders and enter the appropriate data for your system, saving the configuration when you are done. Don't forget to set up the serial port for your PC to the correct port number and baud rate.

   **HINT:** The **Drive type** for each axis must be selected in the **System** folder. Now the axes must be assigned to a specific task. The **Task assignment** item in the **System** folder is used for this purpose.

   **HINT:** If the axis is a servo drive or closed loop stepper the **line count** of the encoder must be entered into the **Encoder** folder.

   **HINT:** Motion is commanded in User Units**.** The **User Units per motor revolution** item in the **System** folder allows you to enter the value. Initially, it is easiest to set this to 1. This will mean that move distances are in motor revolutions (e.g. move=1 moves one revolution), speeds will be in revs/sec, and accelerations will be in revs/sec/sec. Later this can be changed (e.g. to allow programming in inches on a lead screw) to allow ease of programming once the motor is installed into the mechanical system. **All** move distances, speeds, and accelerations (or decelerations), and encoder information are provided in User Units, so be sure you understand this before continuing.

4) **Compile and Download** the project into the controller using the command buttons of the MCPI. Note that initially, you can leave the Task blank and command motion using the **Host Commands.** Host commands are entered in **Terminal Mode** from the MCPI. Enter the terminal mode by clicking on the **Terminal** command button on you screen. If your system consists of stepper drives only go to step 8. See Section 6.3.8.2 of this manual.

5) **Tune the Servo axes**. Before running the motor, the controller compensation parameters (gains) must be set. To aid in this task an automatic servo tuning procedure is available. To enter the servo tuning screen click on the **servo tuning** button. The default values for auto-tuning procedure should work fine for now. The motor may be tuned on the bench with no load. Ensure that the motor is properly secured to your work surface (bench). **Note: Do not clamp the motor anywhere except at the mounting flange.**

   Begin the auto-tuning process by selecting the servo axis you desire to tune and then by clicking on the **Auto Tune** button. A screen with the default values will appear. Click on the **OK** button to use these settings. Next, click on the **Measure System Gain** button. The motor should bump, then the System Gain value should update on the screen. Now click on the **Calculate Servo Gains** button and the calculated servo gain values will be displayed on the screen. Click the **Update Gains** button, the servo should now be locked in position. Verify this by manually trying to turn the motor shaft. The servo should fight to stay in position.
   It's now time to try a test move by entering profile parameters. First click the **Motion Setup** button and enter the desired Acceleration, Deceleration, Speed and Move Distance in user units (e.g. revolutions by

default). When finished click the **Done** button. Now make the motor move by clicking the **Move Response** button. The motor should complete the programmed profile and the position error plot should appear on the screen. You may have to adjust the display time in order to see the whole move.

**6)** Repeat step 5 for all servo axes. Then click on the **Exit** command button and **OK** when save parameter screen appears.

**7)** **Compile and Download** the project into the unit by clicking on the **Compile** and then the **Download** command buttons of the MCPI. This will save the new servo parameters to the MX2000 controller. Note that initially, you can leave the Task blank and command motion using the **HOST Commands**. Host commands are entered in the **Terminal Mode** from the MCPI. Enter the terminal mode by clicking on the **Terminal** command button.

**8)** **Make it move**! Now that you have compiled and downloaded your project into the unit, you are ready to make the motor move. First you must enter the speed at which you wish the motor to turn, such as 1 rev/sec. Do this by typing speed(axis)=1<CR> (<CR> means the Return or Enter key). Now enter the acceleration, for example 50 revs/sec/sec by typing accel(axis)= 50<CR>. Set the deceleration to match by typing decel(axis)=50<CR>. Make sure to connect **CLR** to **COM** for sinking I/O or **CLR** to **+24V** for sourcing I/O on the DSP board or no motor motion will occur. With the motor secured to the bench, you can now command a move. If the axis you want to move is a servo drive you must enable the drive first. This is accomplished by typing wndgs(axis)=1<CR>. To command an incremental move of 10 revolutions type move(axis)=10<CR>. The designated axis motor should now move 10 revolutions. If it does not, check your wiring. Also verify your configuration settings. In addition, check the motor direction to insure it meets your requirements. The motor direction can be reversed in the System folder if necessary.

**Note: Axis is the desired axis you want to address.**

**9)** **Write a BASIC Program**. Now that you have made a simple move, you are ready to write your Task in the MCPI BASIC-like language. Refer to Section 7 for a complete description of all of the **Program Commands.** You can start by opening your Task and entering the commands. First, let's enter the exact same commands that you used in the Terminal HOST mode. Enter speed(axis)=1<CR>, accel(axis)= 50<CR>, decel(axis)=50<CR>, and move(axis)=10 <CR> commands as you did in step 8). If the axis is a servo drive enter the WNDGS(axis)= 1<CR> command before the move command as you did in step 8). You must enter two more commands to tell the unit that the program is done after it performs the move. Type WAITDONE(axis)<CR> and END<CR> as the last lines of the program. Since your program has changed, you must compile and download it into the unit again for the changes to take effect. If you receive compilation errors, check your spelling and syntax with the information in Section 7.

**10)** **Execute the Program**. From the Terminal Host Mode, click on the **RUN** button to make the motor move 10 revolutions. If desired you can now add lines to the program to perform more sophisticated motion. For example, type x=10 <CR>. This assigns the REAL variable "x" a value of 10. Change the MOVE(axis)=10 line to MOVE(axis)=x. Now the motor will move the designated axis whatever distance has been assigned to x. Recompile and download your program, then run it. It should operate the same as before, but now the program is now using x as the move distance in place of 10 as before. Change the value of x to different distance values to verify that it works correctly.

**11)** **Expand the Program and Debug it**. Now that you have written a simple program, you can add more complexity by adding more commands. You can do complex looping, access I/O, and motion functions as required. It will be helpful now to use the **DEBUG** feature of the MCPI environment. Again, refer to Section 12 for a detailed description of the debug mode. If you compile your program in Debug Mode, you can enter the debug screen as your program runs and step through your code to verify proper operation. Once the code is functioning correctly, you should re-compile in Release Mode as this will speed up program execution.

## 3.2.2 - Installation into Mechanical System

Once you have tested everything out in a controlled environment, you may complete the installation into your system. This will require making all the necessary wiring connections for limit switches, additional I/O, analog inputs, encoder, etc. The first thing that must be done is to retune your servo axes, repeat steps 5 to 7. **Start simple!!** Just as you started with a simple move on the bench, you should start simple here as well, slowly adding complexity as you debug your code and gain more confidence in programming. You may use the Debug Mode to help in this process. See Section 12 **Debug Environment** for more information.

## 3.3 - Installation

It is important to select a mounting location for you controller that will meet the environmental specifications listed in Section 5.2. Avoid locations that expose the unit to extremes of temperature, humidity, dirt/dust, or vibration.

Also, it is best to avoid areas with high "electrical noise." This will help to prevent misoperation due to electromagnetic interference. Please refer to Section 3.4.1 for general guidelines on selecting a location for your controller where it will be less susceptible to EMI/RFI problems.

When mounting the unit near other apparatus, such as inside an electrical cabinet or enclosure, please leave at least 2 inches of space on all sides for proper cooling. Mounting brackets are supplied to attach the controller to a vertical surface. The MX2000-8 can also be mounted in a standard 19 inch rack configuration by removing the mounting brackets and rotating them 180E. Please refer to section 5.12, 5.13, and 5.14 for overall dimensions and mounting hole locations for the MX2000-2, -2A, -6, -6A, and -8 respectively.

## 3.4 - WIRING THE CONTROLLER FOR OPERATION

Section 5 **Specifications and Equivalent Circuits** shows how to wire up the individual connectors, depicts equivalent circuits for each connector, describes connector labels, defines connector signal characteristics, defines AC electrical ratings of the System, and defines mechanical and environmental specifications. Be sure to observe the listed electrical ratings of the ac input and the various I/O circuits; this will ensure proper, reliable operation of your controller.

## 3.4.1 – General Wiring Guidelines

SLO-SYN 2000 controls and drives use modern solid-state digital electronics to provide the features needed for advanced motion control applications. Some user equipment may produce electromagnetic interference (EMI, or electrical noise) that can cause inappropriate operation of the digital logic used in the control, drive, or other computer-type equipment in the user=s system.

In general, any equipment that causes arcs or sparks or that switches voltage or current at high frequencies can cause interference. In addition, ac utility lines are often polluted with electrical noise from sources outside a user=s control (such as equipment in the factory next door). Some of the more common causes of electrical interference are:

- power from the utility ac line
- relays, contactors and solenoids
- light dimmers
- arc welders
- motors and motor starters
- induction heaters
- radio controls or transmitters
- switch-mode power supplies
- computer-based equipment
- high frequency lighting equipment
- dc servo and stepper motors and drives

**The following wiring practices should be used to reduce noise interference**.

1) **Solid grounding of the system is essential.** Be sure that there is a solid connection to the ac system earth ground. Bond the drive case to the system enclosure. Use a single-point grounding system for all related components of a system (a Ahub and spokes@ arrangement). Keep the ground connections short and direct.

2) **Keep signal and power wiring well separated**. If possible, use separate conduit or ducts for each. If the wires must cross, they should do so at right angles to minimize coupling.

   Note: Power wiring includes ac wiring, motor wires, etc. Signal wiring is inputs and outputs (I/O), encoder wiring, serial communications (RS232 lines), etc.

3) **Use shielded, twisted-pair cables** for the drive to motor wiring. BE SURE TO GROUND THE SHIELD AT THE DRIVE END.

4) **Suppress all relays** to prevent noise generation. Typical suppressors are capacitors or MOV=s. (See manufacturer=s literature for complete information). Whenever possible, use solid-state relays instead of mechanical contact types to minimize noise generation.

   In some extreme cases of interference, it may be necessary to **add external filtering** to the ac line(s) feeding affected equipment, or to **use isolation transformers** to supply their ac power.

   NOTE: We make a wide range of ac power line conditioners that can help solve electrical interference problems. Contact 1-800-SUP-ELEC (1-800-787-3532) for further assistance.

(This page intentionally left blank)

# SECTION 4
# OVERVIEW OF SYSTEM OPERATION

## 4.1 – Features and Functions

The controller is based on the Texas Instruments TMS320C31 32 bit, 33MHZ Digital Signal Processor (DSP). It can control from 2 to 8 stepper or servo drives, plus 350 I/O points. Each pair of axes is supervised by a powerful Application Specific Integrated Circuit (ASIC) that is custom programmed for the controller. This state-of-the-art computer hardware gives the controller plenty of processing power to coordinate motion and simultaneously execute multi-tasks up to seven complex motion and input-output (I/O) user tasks. The basic two-axis system consists of three major circuit cards that communicate via a passive back plane and are housed in a rugged enclosure.

### MX2 or MX6 system

- 90 to 265 VAC 50/60 Hz input.
- Built-in AC line filter and MOV's.
- Power-on LED.
- Built in 24-volt dc @ 750 ma. supply for I/O.
- 50-pin header for interfacing to as many as 24 OPTO-22 style I/O, or up to 4 BCD switch banks.

### MX2A or MX6A system

- 90 to 265 VAC 50/60 Hz input.
- Built-in AC line filter and MOV's.
- Power-on LED.
- Built in 24-volt dc @ 750 ma. supply for I/O..
- 16 optically isolated inputs.
- 8 optically isolated outputs.

### MX8 system

- Dual Ac voltage range.
- 90 to 132 VAC 50/60 Hz input.
- 175 to 264 VAC 50/60 Hz input.
- Built-in AC line filter and MOV' s
- Power-On LED
- Built in 24-volt dc @ 750 ma. supply for I/O.

### DSP Controller Card

- 256 Kbytes of Flash memory available for user program storage.
- Two serial ports configurable as an RS232 or RS485 device.
- 4 optically isolated inputs.

### Dual Axis Card

- 2 analog outputs capable of a ±10 volt DC swing.
- 4 analog inputs capable of a ±10 volt DC swing.
- 8 dedicated optically isolated inputs for limits and triggers.
- 2 servo or stepper drive interfaces.
- 2 encoder interfaces.

### Digital I/O Card

- 24 optically isolated inputs.
- 16 optically isolated inputs.
- Removable connectors with screw terminals.
- 24 volt power supply access.

### Expansion I/O Card

- 50-pin header for interface to as many as 48 OPTO-22 style I/O, or up to 8 BCD switch banks.

### Programming Features

- English language, BASIC-like coding.
- Full math capability, including trig functions, logs, and square root.
- Boolean logic functions (and, or, xor, not).
- Complex motions (arc, path, line).
- Simple Motions (move, jog).
- Trigger motions (movehome, movereg).
- Position Following.
- Changing Velocity during motion.
- Position Capture from a trigger.
- Subroutines , nested up to 16 levels.
- Multi-tasking of up to 7 concurrent tasks.
- String manipulation (for message handling).
- Program control functions (for-next, if-then-else if-else, goto, do-while, etc).
- Macro substitution (#define) for user-friendly text naming of I/O, etc.
- Complex expressions (using parentheses).
- Multi dimension Arrays.
- 2 Timers per task.
- Complete error handling and warning messages.

## 4.2 - General Overview

The Programmable Motion Controller is a powerful, DSP-based machine controller that is capable of far more than simply moving motors. This section is intended to give the user an overview of the controller's many capabilities including all the functions and features users expect for controlling motion. There are a wide variety of inputs and outputs and software features that, in many cases, allow the controller to operate an entire sophisticated machine. Figure 4.1 shows a typical 2-axis application. Section 5 has details on setting up and wiring the unit.

Of special note is the ease of communication with either "intelligent" or "dumb" operator interfaces. The controller does not require the use of any operator interface panel or host computer to operate as a stand-alone system. Simple switch interfaces via axis I/O or expansion I/O will often suffice for controlling a machine that does not need extensive interaction with the operator for setup information or message display. BCD switches are often used to enter numeric data for simple setup. However, using a panel with a keypad and display gives more flexibility and sometimes easier and more "user-friendly" machine operation.

### 4.2.1 - Serial Communications

Communication with the MX2000 controller is via two serial ports on the DSP Card. These serial ports can be operated as an RS232 or RS485 device. The Host port is used for programming and operating the unit. The Auxiliary Port is used to communicate with an external serial device during program execution. Use of these ports is covered in more detail in Sections 5.6.1 and 5.6.2.

### 4.2.2 - Shutdown Input & Program Select Inputs

The 32 bit DSP interface has four optically isolated inputs. One of these inputs is used as a system shutdown or "motion clear" input. The 3 remaining inputs allow selection of any one of up to seven user programs that will be executed at power-up or when a Reset command is issued. These inputs can be sinking or sourcing. See Section 5.6.3 for more details.

### 4.2.3 - Expansion I/O - BCD Port

An expansion I/O port is provided on the MX2 or MX6 Power Supply or optional Expansion I/O board. The I/O is designed to interface to industry-standard "OPTO-22" style high-power inputs and outputs.

Alternatively, this port can be used to read BCD switches (seven digits plus sign per bank). We provide standard switch banks for use with the controller. Users may also combine BCD's and expansion I/O. See Sections 5.7 and 5.9 for more details.

### 4.2.4 – Digital I/O

A digital I/O port is provided on the MX2A or MX6A Power Supply or optional Digital I/O board. The I/O is designed to operate with switches and relays. These inputs and outputs can be sinking or sourcing. See Sections 5.8 and 5.10 for more details.

### 4.2.5 - Stepper Interface

Standard pulse and direction signals are provided on the Axis Card for controlling most types of stepper drives. Signals are compatible with drives up to 50,000 pulses per revolution (1/250 micro-stepping), since the maximum pulse rate is 1.99 MHz. See Sections 5.5.1 and 5.5.5 for more details.

It is important to note that the controller can be easily programmed in user units, such as inches or revolutions, based on the motor/drive resolution and the machine's characteristics. This is possible because of the controller's extensive math functions. See Section 7 **Software Reference Guide** for more details.

### 4.2.6 – Analog Drive

The analog outputs can be used as the torque command for a servo drive. In addition a pair of drive enable output and drive ready inputs have been provided.

### 4.2.7 - Encoder Interface

Inputs from two incremental encoders are provided on the Axis Card. The maximum count rate is 2 MHz. There is 5Vdc power available on this connector to power the encoders. Wiring to this port is covered in Section 5.5.3.

### 4.2.8 - Axis I/O and Analog I/O

Inputs are provided on the Axis Card for two axes worth of limit switches, home switches, and mark registration sensors. (The latter two are connected to the "Event 1" and "Event 2" pins.) These can be configured for sink or source operation. Also, there are two sets of analog inputs that can be read under program control. These inputs may be used for reading various types of sensors (temperature, pressure, etc.) and then controlling index distance or motor speed based on the value read. See section 5.5.4 for more details.

## 4.3 - Use of the Serial Ports, "HOST" and "AUXILIARY"

The controller has two serial ports, which are identified as "HOST" and "AUXILIARY". The "HOST" port, as its name implies, is typically connected to a host computer such as an IBM PC or compatible. The "AUXILIARY" port is intended for use with an operator interface panel.

The "HOST" port is used for downloading the user's application program and for direct control of the controller. When using the MCPI programmable Interface, all communication with the controller is via the "HOST" port. In addition, all on-line debugging is accomplished using this port. The "HOST" port also has the capability to "DAISY CHAIN" to other controllers; this requires only one serial port on a user's host computer to communicate to multiple controllers. While the user's program could use the "HOST" port for communication with any device that has a serial port, it is recommended that the "HOST" port be reserved for debugging the user's program and for communication with the host computer.

The "AUXILIARY" port, while intended for use with an operator interface panel (O.I.P.), can in fact communicate with any device that has a serial port, such as counter units, etc. The "AUXILIARY" port can send and receive standard ASCII characters. The user's application program can transmit a prompt or message using the "PRINT" statement and wait for a response using the "INPUT" statement.

Example:

```
PRINT #2,"Enter 6 digit part number"
INPUT #2, PART$
```

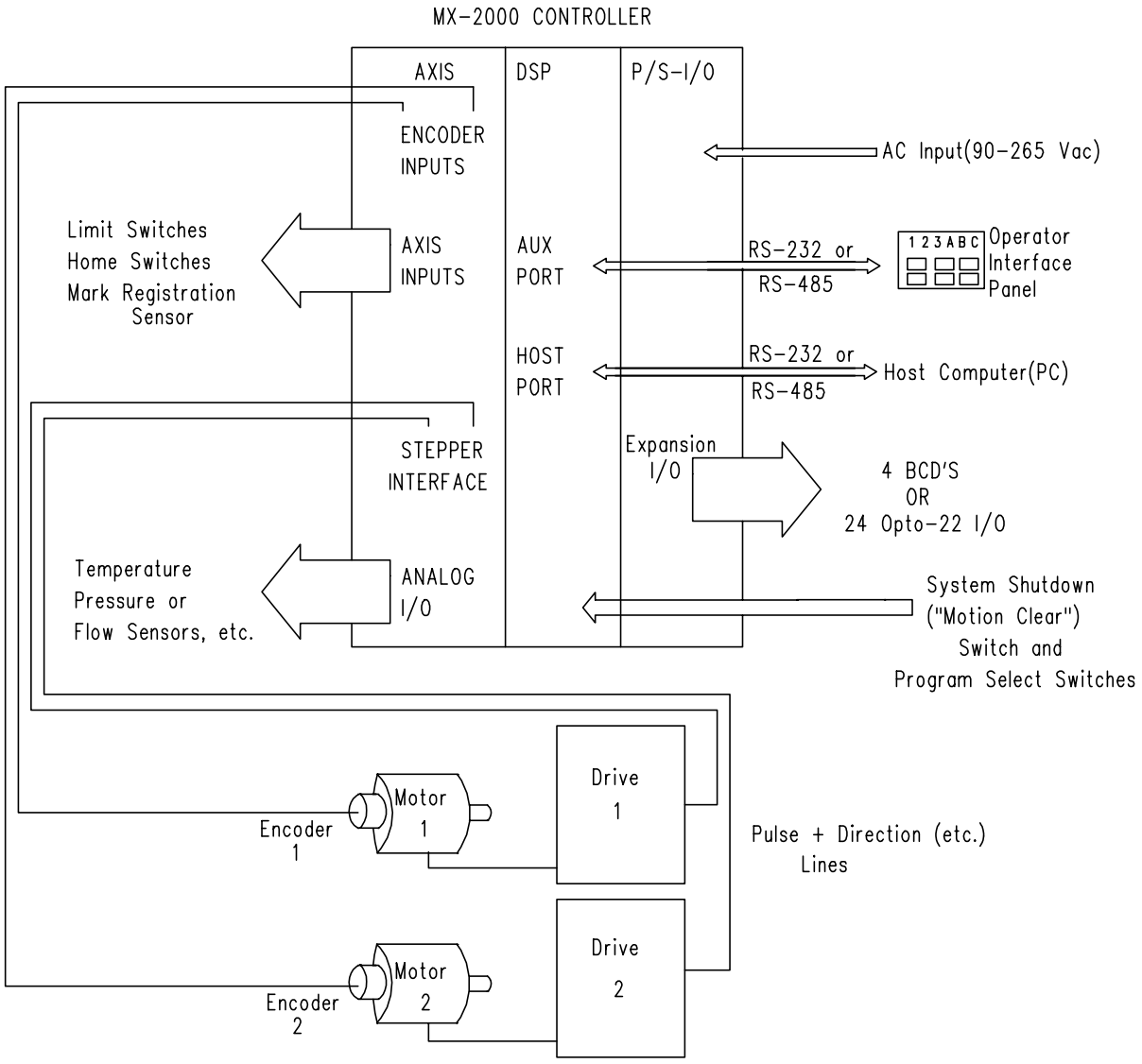
A message is displayed on the OIP screen prompting the machine operator to enter a part number. The string variable PART$ can now be examined (by the controller program) to determine what type of process to perform. The information provided by the operator can then be used to control the process flow, ie. move distance, velocity, dwell, etc., for the desired part number that the machine is processing.

While the process is in operation, messages can also be sent back to the OIP, telling the operator the status of the process. For example,

```
PRINT #2, "Coarse grind"
PRINT #2, "Finish grind"
```

will display the indicated messages on the OIP regarding the grinding operation that is occurring.

**Figure 4.1, General Application Overview**



This page left intentionally blank

# Section 5
# Specifications
# And
# Equivalent Circuits

## 5.1 – Mechanical Specification

MX2000-2
- Size: 5.34” X 10.63” X 7.48”
  135.6 mm X 270 mm X 190 mm
- Weight: 8.25 lbs
  3.75 Kg

MX2000-6
- Size: 9.34” X 10.63” X 7.48”
  237.3 mm X 270 mm X 190 mm
- Weight: 11.0 lbs
  5.0 Kg

MX2000-8
- Size: 19.0” X 10.63” X 7.54”
  482.6 mm X 270 mm X 191.6 mm
- Weight: 12.0 lbs
  5.45 Kg

## 5.2 – Environmental Specification

| | |
|---|---|
| Operating Temperature: | +32° F to +122° F |
| | 0° C to +50° C |
| Storage Temperature: | -40° F to +167° F |
| | -40° C to +75° C |
| Humidity: | 95% max. non-condensing |
| Altitude: | 10,000 feet maximum |
| | 3048 meters maximum |

## 5.3 – Input Power

MX2000-2 or MX2000-2A
- voltage: 90 to 265 VAC, 50/60 hz
- current: < 0.5 Amps @ 115 VAc
- fuse: 2 Amp (normal blow), 250VAC, 3AG type (2 required)

MX2000-6 or MX2000-6A
- voltage: 90 to 265 VAC, 50/60 hz
- current: < 0.5 Amps @ 115 VAc
- fuse: 2 Amp (normal blow), 250VAC, 3AG type (2 required)

MX2000-8
- voltage: 90 to 132 VAC, 50/60 hz
  175 to 254 VAC, 50/60 hz
- current: < 3 Amps @ 115 VAc
- fuse: 3 Amp (slow blow), 250VAC

## 5.4 – MX2000 System

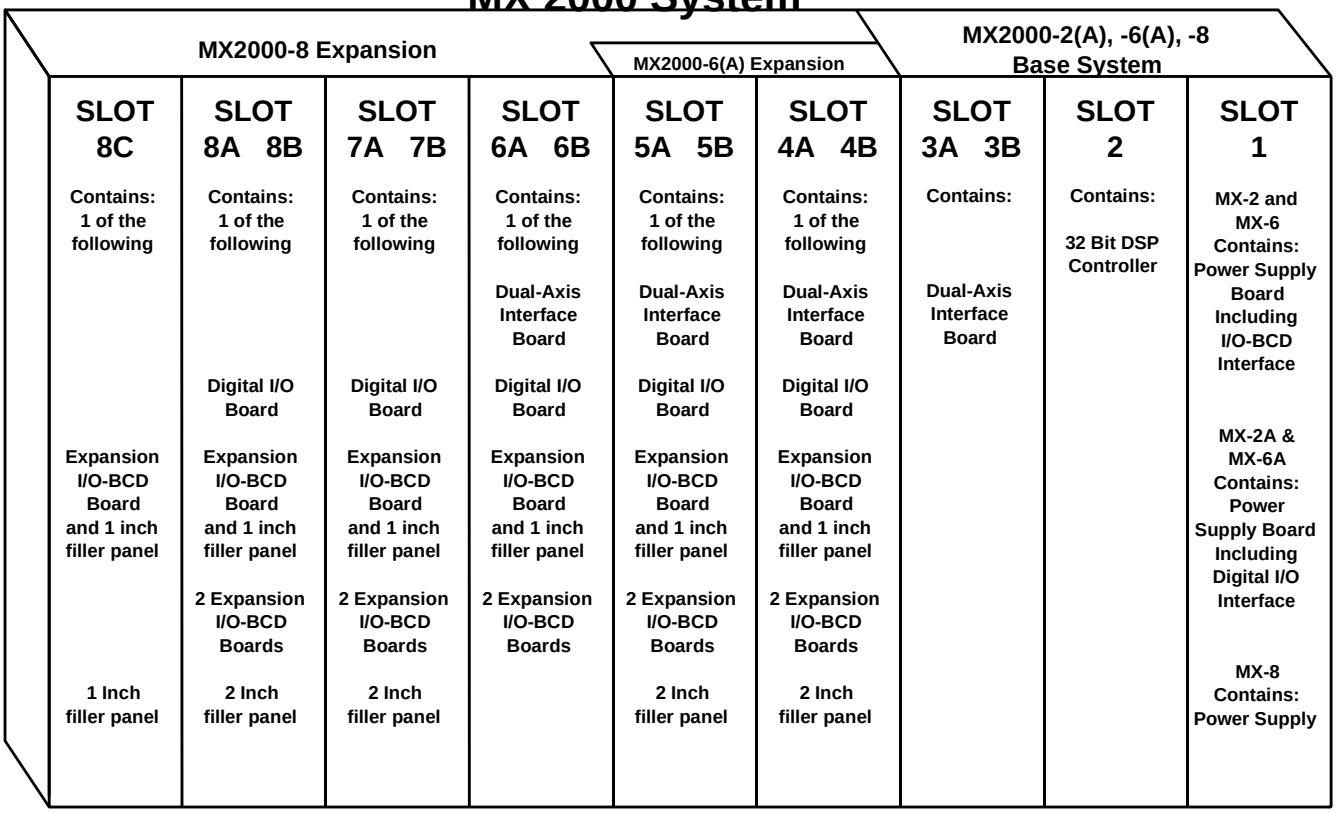
When ordering an MX2000 system a number of factors must be taken into account. The number of axes, number of I/O points and whether the I/O requires optical isolation. The MX-2 and MX-6 power supplies have 24 expansion I/O points that are not optically isolated but can be interfaced to an OPTO-22 rack module. The MX-2A and MX-6A power supply has 16 optically isolated inputs and 8 optically isolated outputs. The MX-8 power supply has no I/O on it.

An Isolated 24 volt supply has been provided which has a maximum current capability of 750 ma.

Another consideration for the I/O connections is the connector style. The MX-2, MX-6 and any additional expansion I/O cards have 50 pin mass termination connections and are not optically isolated. The MX-2A, MX-6A and any additional digital I/O card have plug-in screw terminations and are optically isolated.

A System Block diagram with all the different combination to make up an MX2000 controller has been provided on the next page.

# MX 2000 System




Notes: 1) Up to 4 Dual Axis Interfaces Boards allowed in the System.

2) Up to 4 Expansion I/O-BCD Board allowed in the System. This includes the Expansion I/O-BCD section on the Power Supply Board in an MX-2 or MX-6 system.

3) Up to 4 Digital I/O Boards allowed in the System. This includes the Digital I/O section on the Power Supply Board in an MX-2A or MX-6A system.

A list of the part numbers for the discrete part of the system has been provided for your convenience.

| | |
|---|---|
| Dual-Axis Interface board<br>(2 axis stepper and or servo interface with dedicated I/O) | 222420-001 |
| Digital I/O board<br>(24 inputs, 16 outputs optically isolated) | 222421-001 |
| Expansion I/O-BCD board<br>(48 I/O points non-optically isolated) | 222642-001 |

## 5.5 -Dual Axis Interface Card

This card contains the interfaces necessary to connect 2 motor drives to the MX2000 controller. A stepper drive or servo drive can be interfaced to the controller. In addition 4 dedicated inputs and up to 2 analog inputs can be interfaced to each axis.

Up to four Axis cards can be plugged into an MX2000-8 back plane. Each axis card must be assigned a different id (1-4) and id 1 must always be assigned to one of the boards. The factory setting is board Id 1. For proper operation, a Dual Axis board must always be plugged into the MX2000 controller. The ID switches are located behind the Analog output connector on the Dual Axis card.

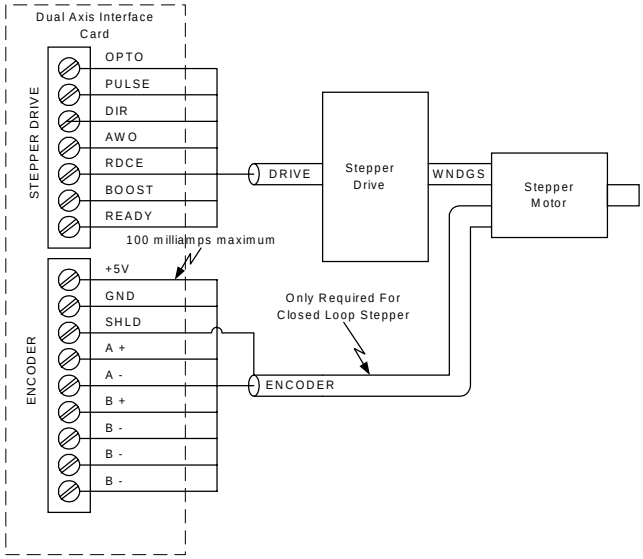
The 4 dedicated inputs for each axis are optically isolated and can be either sinking or sourcing inputs. This selection is made on the Dual Axis Interface card by plugging the select jumper into the desired position. See the card layout diagram to locate the jumper. The factory setting is Sink.

| Board Id | SW1 switch positions | | | Axes Assigned | MX2000-2 System | MX2000-6 System | MX2000-8 System |
|---|---|---|---|---|---|---|---|
| | C | B | A | | | | |
| 1 | On | On | On | 1 & 2 | Yes | Yes | Yes |
| 2 | On | On | Off | 3 & 4 | Not available | Yes | Yes |
| 3 | On | Off | On | 5 & 6 | Not available | Yes | Yes |
| 4 | On | Off | Off | 7 & 8 | Not available | Not available | Yes |

**Note: The "A" side is the odd axis connector and the "B" side is the even axis connector.**

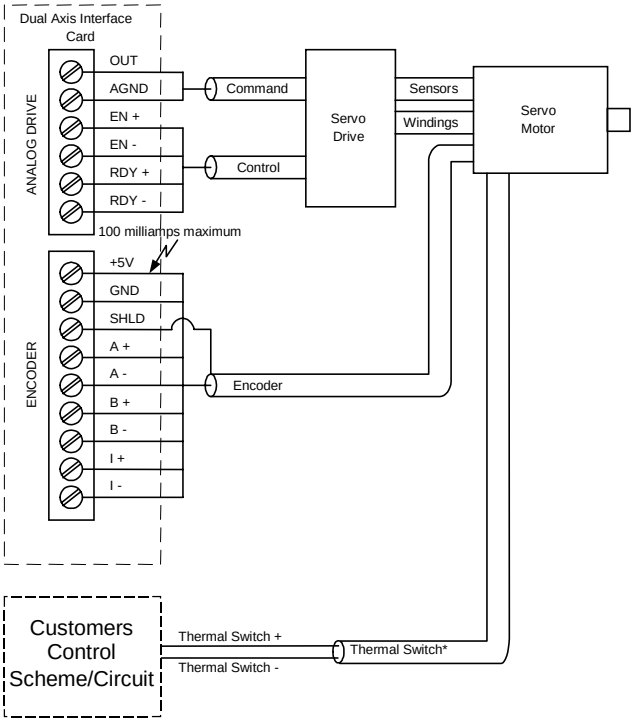
## 5.5.1 – Stepper Drive Connections

The stepper drive connections are made to the STEPPER DRIVE connector. If an encoder is attached to the stepper motor the Encoder connections are made to the ENCODER connector. An illustration of this is provided. The signal descriptions and equivalent circuits of each connector will be covered later on in this section.




## 5.5.2 – Servo Drive Connections

The servo drive connections are made to the ANALOG DRIVE connector. The encoder connections from the motor are made to the ENCODER connector. An illustration of this is provided. The signal descriptions and equivalent circuits of each connector will be covered later on in this section.

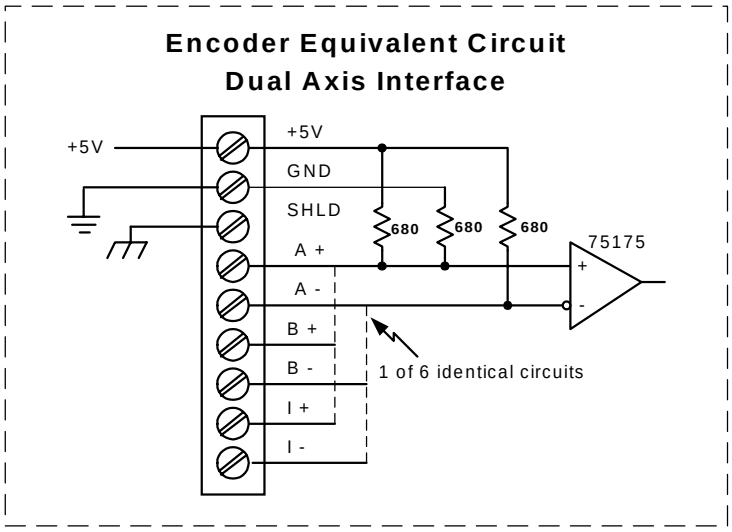


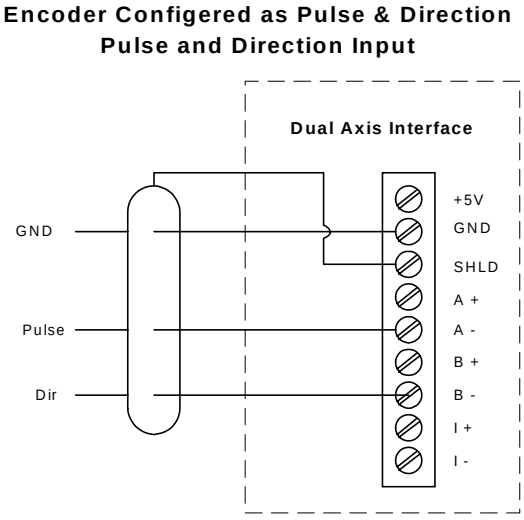

* If the Servo Motor has thermal switches included it is recommended that these connections be made to a control circuit (Stop) to indicate when a Motor Overtemp condition exists.

## 5.5.3 –Encoder Connectors

The Encoder connector provides a means to interface an encoder or pulse and direction input to the controller. There are two identical connectors provided, one for each axis. The encoder 5 volt output is restricted to 100 milli-amps of current per axis. We highly recommend the use of twisted-pair (approximately 6 twists per foot) shielded cable for all encoder wiring to minimize interference problems. The following signal should be twisted together A+ with A-, B+ with B-, I+ with I- and +5V with GND.

| Encoder Connector | | |
|---|---|---|
| Signal Name | Description | Electrical Specification |
| +5V | +5 volts for the encoder | 5 ±0.2 volts @ 100 ma per encoder |
| GND | Signal ground for encoder | Not applicable |
| SHLD | Connection to shield | Not applicable |
| A+ | Encoder channel A+ input | 7.3 ma @ +5 volts<br>7.3 ma @ 0 volts |
| A- | Encoder channel A- input | 0 ma @ +5 volts<br>7.3 ma @ 0 volts |
| B+ | Encoder channel B+ input | 7.3 ma @ +5 volts<br>7.3 ma @ 0 volts |
| B- | Encoder channel B- input | 0 ma @ +5 volts<br>7.3 ma @ 0 volts |
| I+ | Encoder channel I+ input | 7.3 ma @ +5 volts<br>7.3 ma @ 0 volts |
| I- | Encoder channel I- input | 0 ma @ +5 volts<br>7.3 ma @ 0 volts |

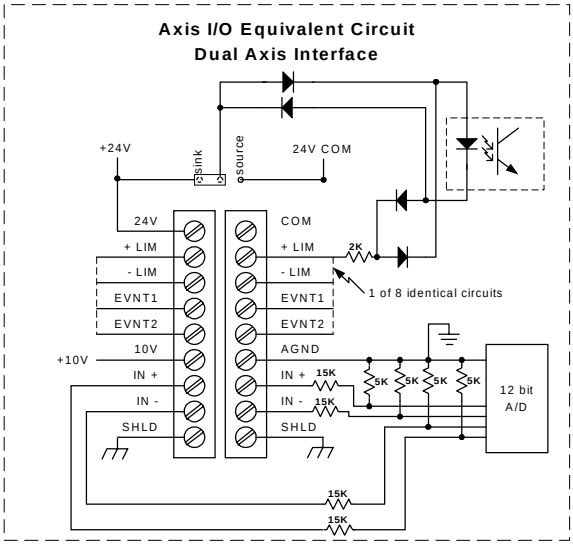


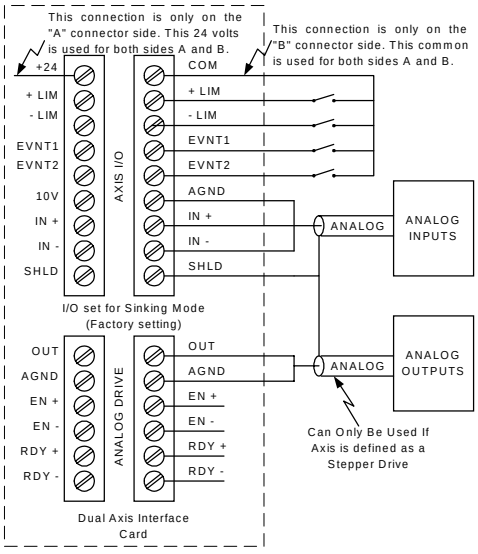



## 5.5.4 – Axis I/O Connectors

The Axis I/O connectors provides a means of interfacing 4 dedicated digital signals and an analog input for each axis. The dedicated inputs are opto-isolated and can be selected as sinking or sourcing on the axis card. The dedicated inputs are labeled +LIM, -LIM, EVNT1 and EVNT2. The two limit inputs can be configured as hard limits or general purpose inputs. The two event inputs are used for mechanical home and mark registration triggers. The COM and 24V terminals have been provided as return paths for the optical isolator circuits. The COM terminal is used for sinking and the +24V terminal is used for sourcing. The analog input terminals, IN+ and IN-, can be configured as single ended or differential input. If configured as single ended these signal become independent inputs with the AGND signal as a signal common. The voltage range for the analog input is ±10 volts. The 10V, AGND and SHLD terminals are intended to be used with the analog inputs. The 10V terminal provides a +10 volt reference output signal for the analog inputs.

| Signal Name | Description | Electrical Specification |
|---|---|---|
| COM | Return for input sinking mode. | Not Applicable |
| 24V | Return for input sourcing mode. | Not Applicable |
| SHLD | Connection to Shield . | Not applicable |
| +LIM | Positive travel limit or general purpose input. | Sink mode: 10.5 ma @ 0v, 0v to +3v on state, 24v off state. Source mode: 10.5 ma @ 24v, 4.5 ma @ 12v, +12v to +24v on state, 0v to +3v off state. |
| -LIM | Negative travel limit or general purpose input. | Sink mode: 10.5 ma @ 0v, 0v to +3v on state, 24v off state. Source mode: 10.5 ma @ 24v, 4.5 ma @ 12v, +12v to +24v on state, 0v to +3v off state. |
| EVNT1 | Home, mark registration or general purpose input. | Sink mode: 10.5 ma @ 0v, 0v to +3v on state, 24v off state. Source mode: 10.5 ma @ 24v, 4.5 ma @ 12v, +12v to +24v on state, 0v to +3v off state. |
| EVNT2 | Home, mark registration or general purpose input. | Sink mode: 10.5 ma @ 0v, 0v to +3v on state, 24v off state. Source mode: 10.5 ma @ 24v, 4.5 ma @ 12v, +12v to +24v on state, 0v to +3v off state. |
| 10V | +10 volt reference | 10 ± .07 volt @ 20 ma maximum load. |
| AGND | Analog Ground | Not applicable |
| IN+ | Analog non-inverting differential or single ended input | 12 bit resolution, 1950 samples/sec, ± 10 volt range, 20K ohms input impedance, ±0.1 volt full scale accuracy, ±0.035 volt zero input accuracy. |
| IN- | Analog inverting differential or single ended input | 12 bit resolution, 1950 samples/sec, ± 10 volt range, 20K ohms input impedance, ±0.1 volt full scale accuracy, ±0.035 volt zero input accuracy. |

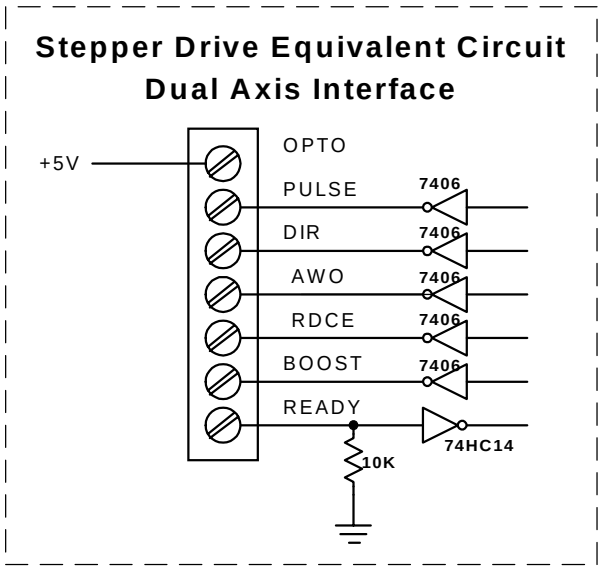


## 5.5.5 – Stepper Drive Connectors

The stepper drive connector provides a means of connecting the controller to a stepper drive. There are two identical connectors provided for this means, one for 1st axis (Side "A") and one for the 2nd axis (Side "B").

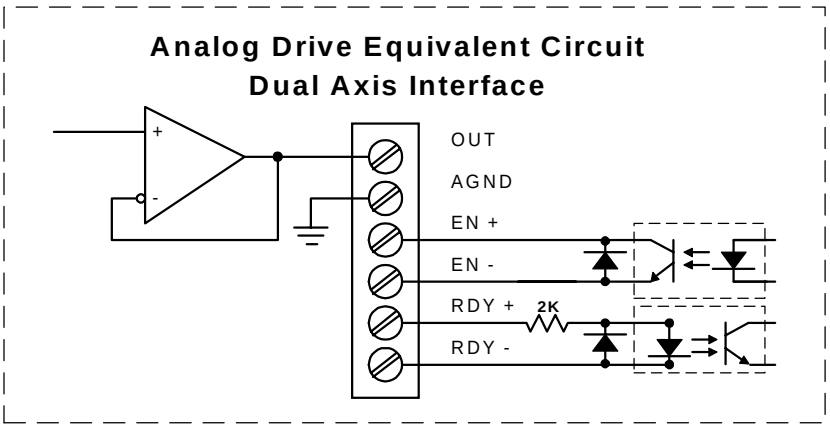
| Signal Name | Description | Electrical Specification |
|---|---|---|
| OPTO | 5 volt source for opto-isolators. | 100 ma maximum load |
| PULSE | 0 to 1.99 Mhz square wave when motion is commanded. | Open drain output, +30v maximum high level voltage, +0.7v @ 40 ma low level voltage. |
| DIR | Motor direction control. | Open drain output, +30v maximum high level voltage, +0.7v @ 40 ma low level voltage. |
| RDCE | Reduce motor current by 50% at standstill. A Low level output reduces the motor current. | Open drain output, +30v maximum high level voltage, +0.7v @ 40 ma low level voltage. |
| BOOST | Increase motor current by 50% when running. A low level output increases motor current during motion. | Open drain output, +30v maximum high level voltage, +0.7v @ 40 ma low level voltage. |
| AWO | Turns winding off when at standstill. A low level output turns windings off. | Open drain output, +30v maximum high level voltage, +0.7v @ 40 ma low level voltage. |
| READY | Indicates the status of drive. A high level input indicates a drive ready condition. | Input loading 10K ohms, low level voltage 0v to +0.9v, high level voltage +3.5v to +5.0v |

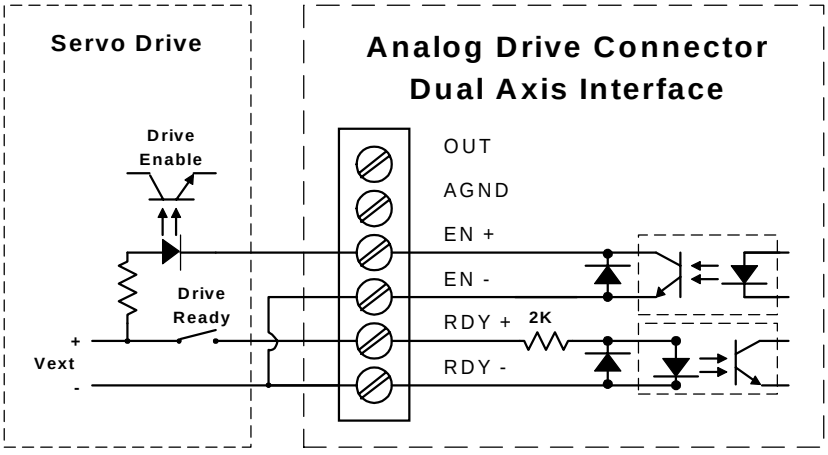


## 5.5.6 – Analog Drive Connector

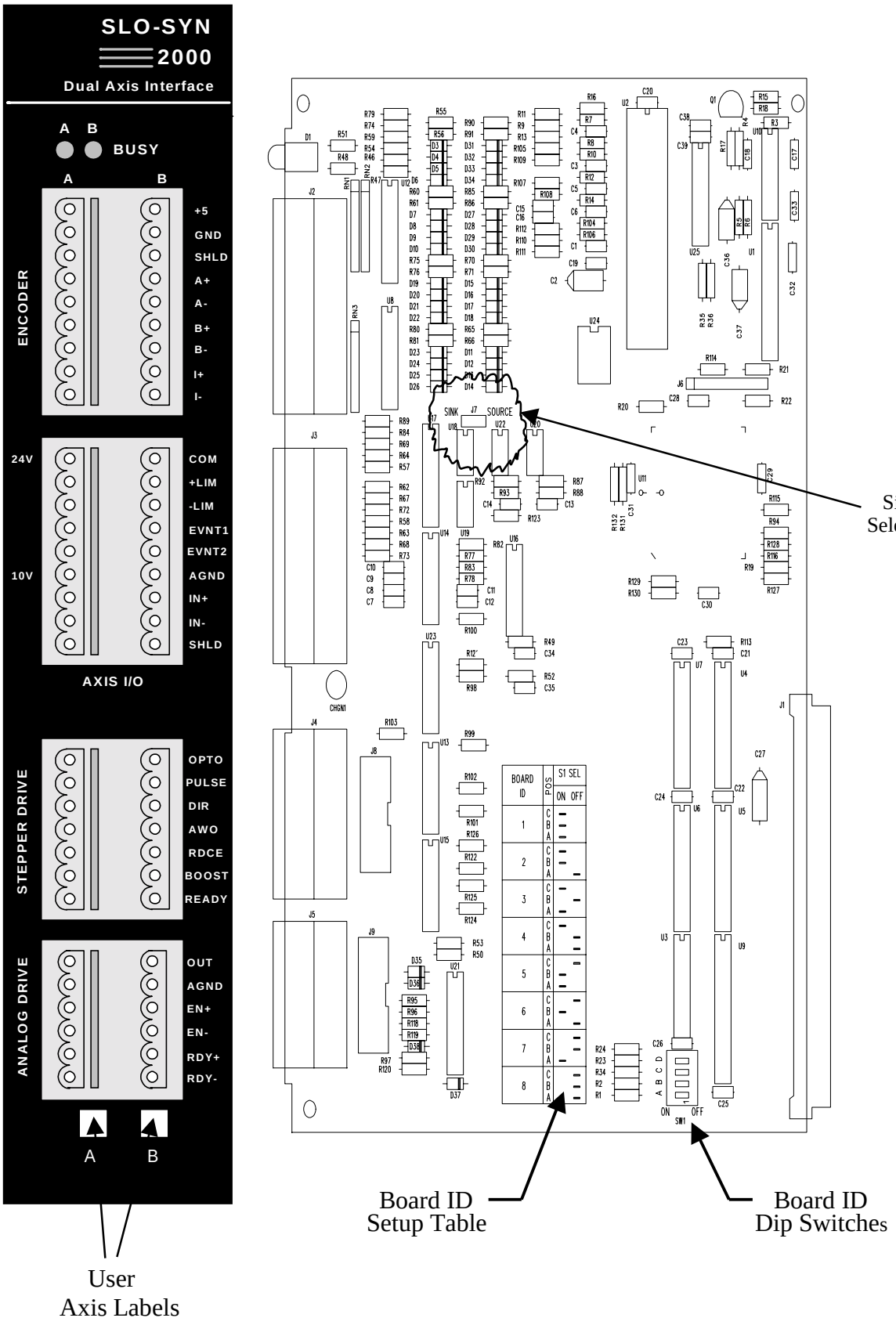
The Analog drive connector provides a means of connecting a servo drive to the controller. There are two identical connectors provided for this means, one for 1st axis (Side "A") and one for the 2nd axis (Side "B").

| Signal Name | Description | Electrical Specification |
|---|---|---|
| OUT | Analog output voltage that can be used as a torque command for a servo drive or an analog output if axis is not a servo drive. | ± 10v output @ 5 ma maximum (2K ohm load). Output accuracy: 0v output ±0.03v, Full scale error ±0.11v. |
| AGND | Analog Ground. | Not applicable |
| EN+ | Collector output of an optically isolated device. When servo drive is enabled the opto-isolator transistor is conducting. | Maximum Collector–Emitter voltage 70 volts, Saturation voltage .4v @ 15ma |
| EN- | Emitter output of an optically isolated device. When servo drive is enabled the opto-isolator transistor is conducting. | Maximum Collector–Emitter voltage 70 volts, Saturation voltage .4v @ 15ma |
| RDY+ | + side of an opto-isolated input. The drive is ready when current flows through opto-isolator. | On current 11.5 ma @ 24 volts (2 K ohm load), Off current < 40 ua @ 0 to 1.3 volts |
| RDY- | - side of an opto-isolated input. The drive is ready when current flows through opto-isolator. | On current 11.5 ma @ 24 volts (2 K ohm load), Off current < 40 ua @ 0 to 1.3 volts |



## 5.5.7 – Dual Axis Interface Card



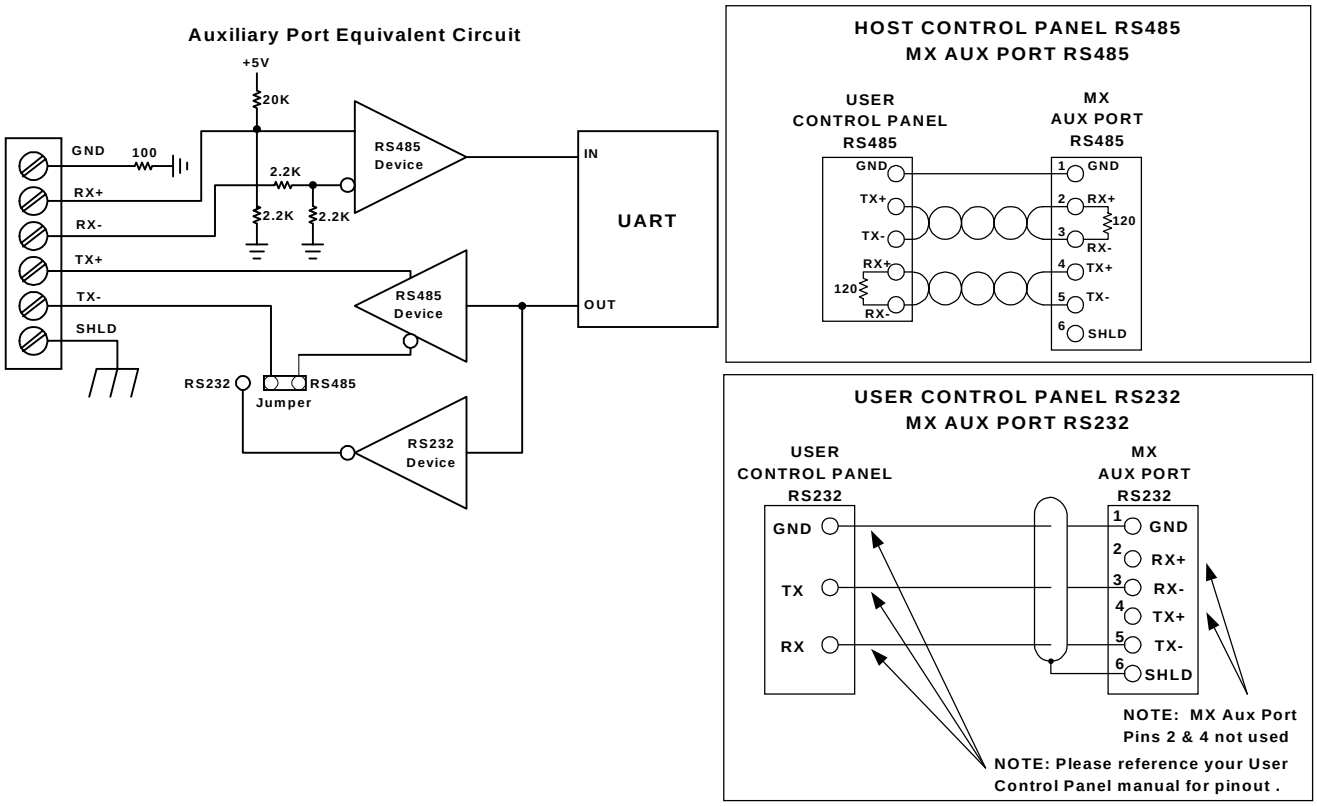
## 5.6 – 32 bit DSP Controller Card

This card contains the interfaces to serially communicate with a host port and auxiliary port, and controls the entire set of cards plugged into the MX2000 system.

## 5.6.1 – Auxiliary Serial Port

This port can communicate serially with an external port as an RS232 or RS485 serial device. The type of device is jumper selectable (J10) on the board and the factory setting is RS485. The serial protocol for this port can be modified by the SETCOM command. The default setting is 9600 baud, no parity, 8 bit and 1 stop bit. This port is referred to as Port #2 when used in the user generated program.

| Signal Name | Description |
|---|---|
| GND | Signal Ground reference |
| RX+ | Differential receiver non-inverting input. |
| RX- | Differential receiver inverting input or RS232 receiver input. |
| TX+ | Differential transmitter non-inverting output. |
| TX- | Differential transmitter inverting output or RS232 transmitter output. |
| SHLD | Connection to Shield |



## 5.6.2 – Host Serial Port

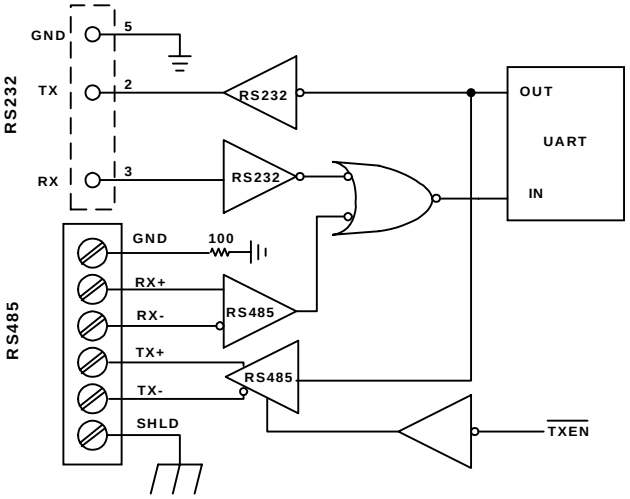
This serial port is used to program the unit or communicate with the host device. There are two serial interfaces for the host port, RS232 and RS485. The RS232 interface uses a 9-pin D female connector. The RS485 interface connection is provided on a 6-position removable terminal strip. The device that communicates with the host computer can be either RS232 or RS485. This selection is made with the 232/485 dip switch on the front panel. MX2000 units can be daisy chained using the RS485 interface. The Unit ID switch is used to set a unique number for each unit for this purpose. This port is referred to as Port #1 when used in a user generated program.

The data format for serial communications is no parity, 8 data bits and 1 stop bit. The baud rate is switch selectable for 4800, 9600, 19200 or 38400. The baud rate switches and host device selection are only read on power turn-on or when a RESET command is issued.

| Dip switch Setting | | | | |
|---|---|---|---|---|
| 232/485 switch | 9.6 switch | 19.2 switch | 38.4 switch | Comments |
| 232 | ON | OFF | OFF | Host communicates RS232, Daisy chaining is RS485, 9600 baud |
| 232 | OFF | ON | OFF | Host communicates RS232, Daisy chaining is RS485, 19200 baud |
| 232 | OFF | OFF | ON | Host communicates RS232, Daisy chaining is RS485, 38400 baud |
| 232 | OFF | OFF | OFF | Host communicates RS232, Daisy chaining is RS485, 4800 baud |
| 485 | ON | OFF | OFF | Host communicates RS485, Daisy chaining is RS485, 9600 baud |
| 485 | OFF | ON | OFF | Host communicates RS485, Daisy chaining is RS485, 19200 baud |
| 485 | OFF | OFF | ON | Host communicates RS485, Daisy chaining is RS485, 38400 baud |
| 485 | OFF | OFF | OFF | Host communicates RS485, Daisy chaining is RS485, 4800 baud |

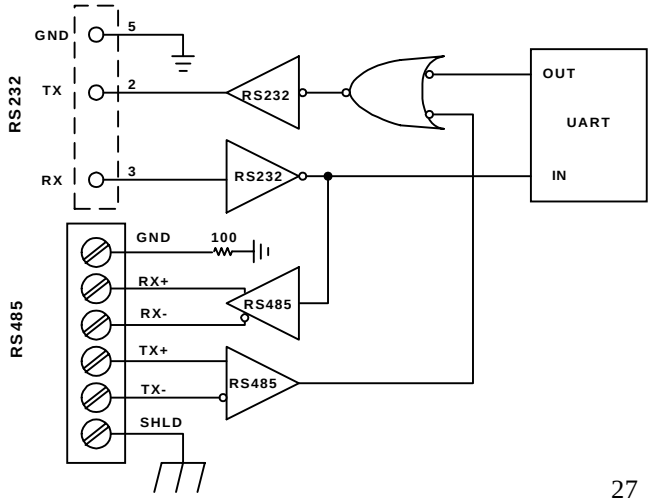
| Host RS485 Connector | |
|---|---|
| Signal Name | Description |
| GND | Signal Ground reference |
| RX+ | Differential receiver non-inverting input. |
| RX- | Differential receiver inverting input. |
| TX+ | Differential transmitter non-inverting output. |
| TX- | Differential transmitter inverting output |
| SHLD | Connection to Shield |

**Host Serial Port Equivalent Circuit**
**RS485 Position selected**

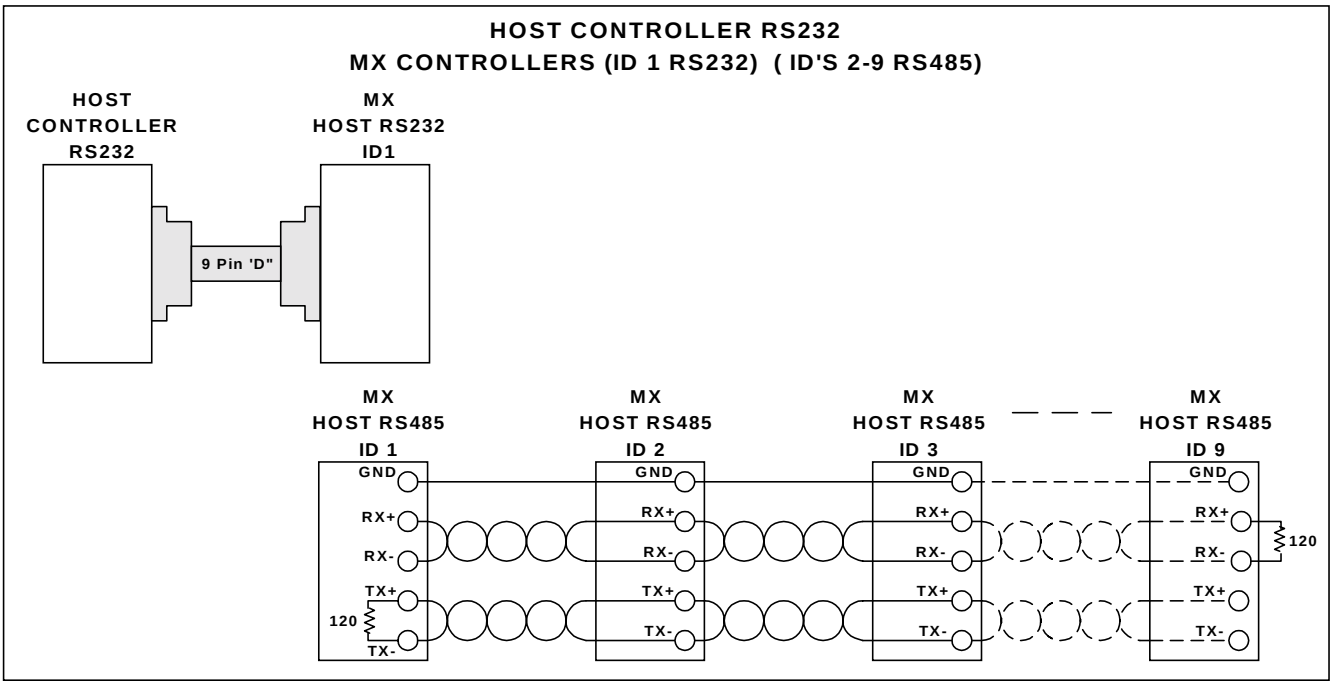


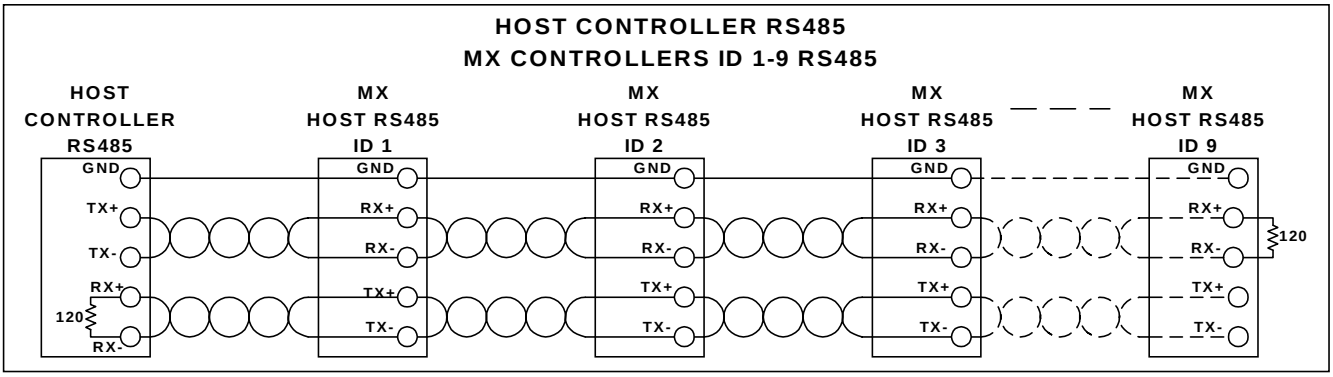

| Host RS232 Connector | | |
|---|---|---|
| Pin | Signal Name | Description |
| 1 | GND | Signal Ground reference |
| 2 | TX | MX2000 Transmitter terminal |
| 3 | RX | MX2000 Receiver terminal |
| 4 | GND | Signal Ground reference |
| 5 | GND | Signal Ground reference |
| 6-9 | NC | No connection |

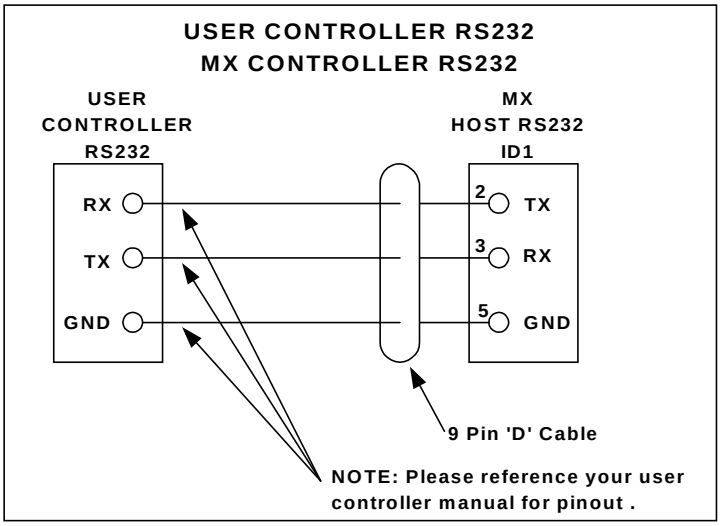
**Host Serial Port Equivalent Circuit**
**RS232 Position selected**



# Daisy Chaining MX2000 Controllers



## 5.6.3 – DSP Card Inputs

There are four optically isolated inputs to the DSP card. These inputs can either be sinking or sourcing and the selection are controlled by a jumper (J11) on the DSP board. The factory setting is sinking.

The CLR terminal is a fail safe input that will terminate program execution and or motion if open circuited. **This input must be active to allow auto-execution to occur on power turn-on or if a RESET command is issued**.

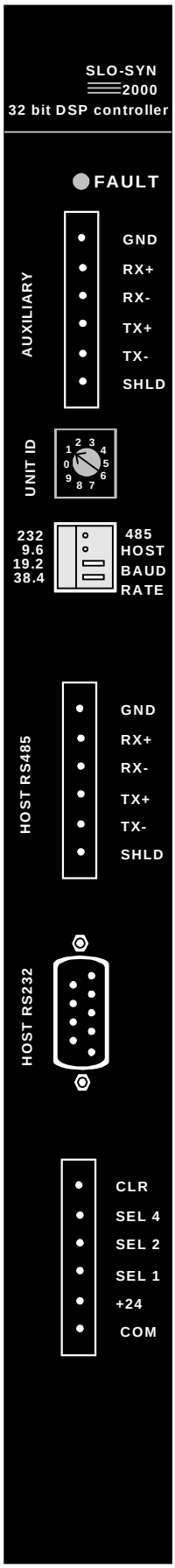
The program that will be auto-executed is selected by the input states of the SEL 4, SEL 2 and SEL 1 inputs.

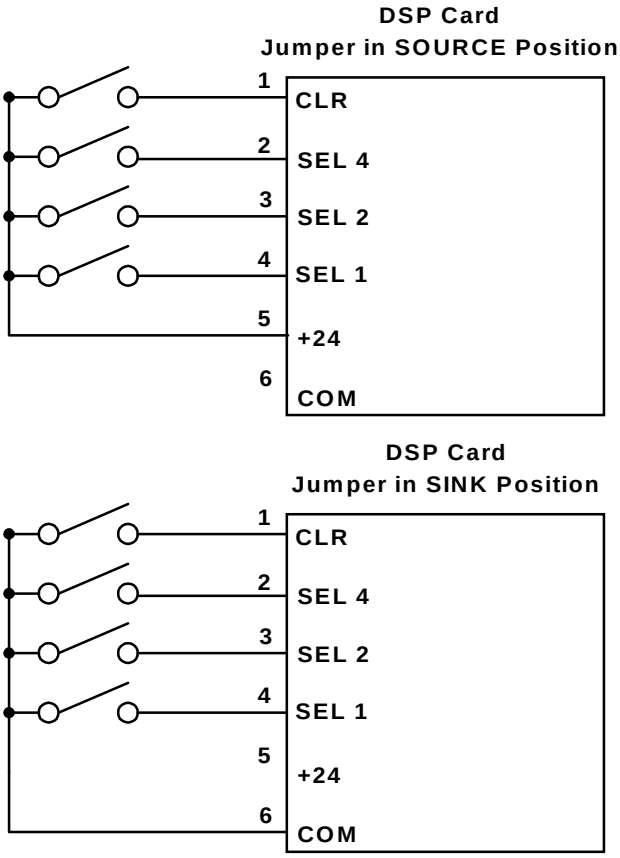
**Note: In Order for motion to occur, a CLR to COM jumper is required.**

Projects are loaded sequentially into the MX2000 controller memory after an ERASE DIR command is issued and are labeled projects 0 to 6.

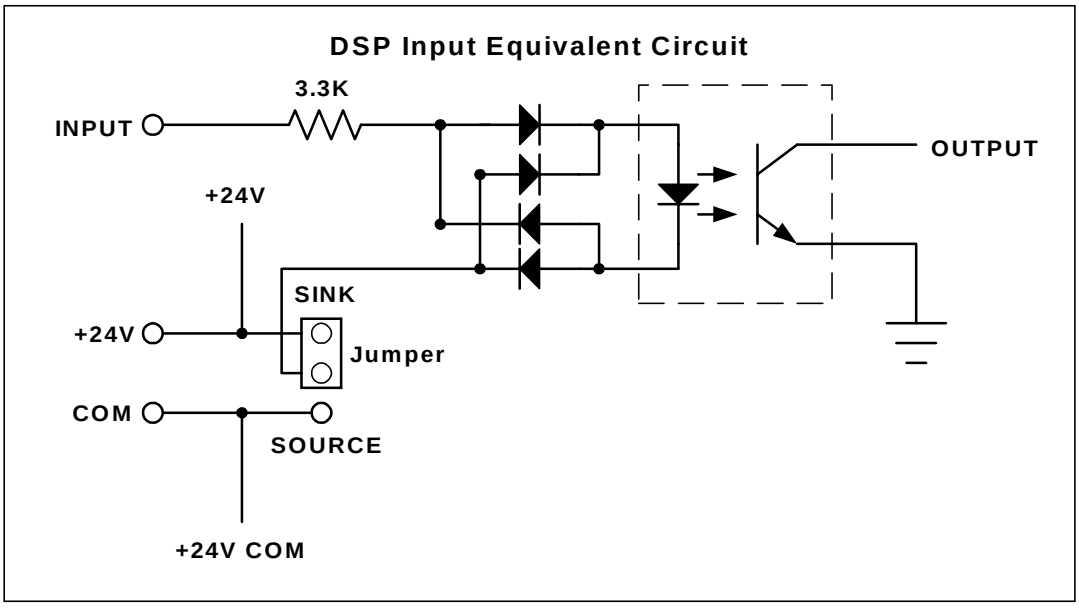
| Auto-Execute selection Chart | | | |
|---|---|---|---|
| Project selected | SEL 4 | SEL 2 | SEL 1 |
| 1st Project (Project 0) | Inactive | Inactive | Inactive |
| 2nd Project (Project 1) | Inactive | Inactive | Active |
| 3rd Project (Project 2) | Inactive | Active | Inactive |
| 4th Project (Project 3) | Inactive | Active | Active |
| 5th Project (Project 4) | Active | Inactive | Inactive |
| 6th Project (Project 5) | Active | Inactive | Active |
| 7th Project (Project 6) | Active | Active | Inactive |
| 1st Project (Project 0) | Active | Active | Active |

| Signal Name | Description | Electrical Specification |
|---|---|---|
| CLR | Stop program execution and motion if open circuited. | Sink mode: On state 0 to +12 volts, 3.2 ma @ +12v, 6.8 ma @ 0v<br>Source mode: On state +12 to +24 volts, 3.2 ma @ +12v, 6.8 ma @ +24v |
| SEL 4 | Auto-execute program select | Sink mode: On state 0 to +12 volts, 3.2 ma @ +12v, 6.8 ma @ 0v<br>Source mode: On state +12 to +24 volts, 3.2 ma @ +12v, 6.8 ma @ +24v |
| SEL 2 | Auto-execute program select | Sink mode: On state 0 to +12 volts, 3.2 ma @ +12v, 6.8 ma @ 0v<br>Source mode: On state +12 to +24 volts, 3.2 ma @ +12v, 6.8 ma @ +24v |
| SEL 1 | Auto-execute program select | Sink mode: On state 0 to +12 volts, 3.2 ma @ +12v, 6.8 ma @ 0v<br>Source mode: On state +12 to +24 volts, 3.2 ma @ +12v, 6.8 ma @ +24v |
| +24 | Return for input sourcing mode | Not applicable |
| COM | Return for input sinking mode | Not applicable |


SLO-SYN
2000
32 bit DSP controller
FAULT
AUXILIARY
GND
RX+
RX-
TX+
TX-
SHLD
UNIT ID
232
9.6
19.2
38.4
485
HOST
BAUD
RATE
HOST RS485
GND
RX+
RX-
TX+
TX-
SHLD
HOST RS232
CLR
SEL 4
SEL 2
SEL 1
+24
COM


DSP Card
Jumper in SOURCE Position
1
CLR
2
SEL 4
3
SEL 2
4
SEL 1
5
+24
6
COM


DSP Card
Jumper in SINK Position
1
CLR
2
SEL 4
3
SEL 2
4
SEL 1
5
+24
6
COM


DSP Input Equivalent Circuit
3.3K
INPUT
OUTPUT
+24V
SINK
+24V
Jumper
COM
SOURCE
+24V COM

## 5.7 – Expansion I/O Board

The Expansion I/O-BCD board has been designed to interface to BCD switches and/or to an OPTO 22 module rack. There are two ports on each board and each port has 24 bi-direction I/O points. The odd pins 1-47 on the 50-pin header are signal pins. The even pins 2-50 are signal grounds.

If the MX2000 controller is an MX2 or MX6 the first expansion I/O board is on the Power supply board and there are only 24 I/O points available (100-124). Up to 4 boards can be interfaced to an MX2000-8 controller. The ID for each board is selected using two dip switches.

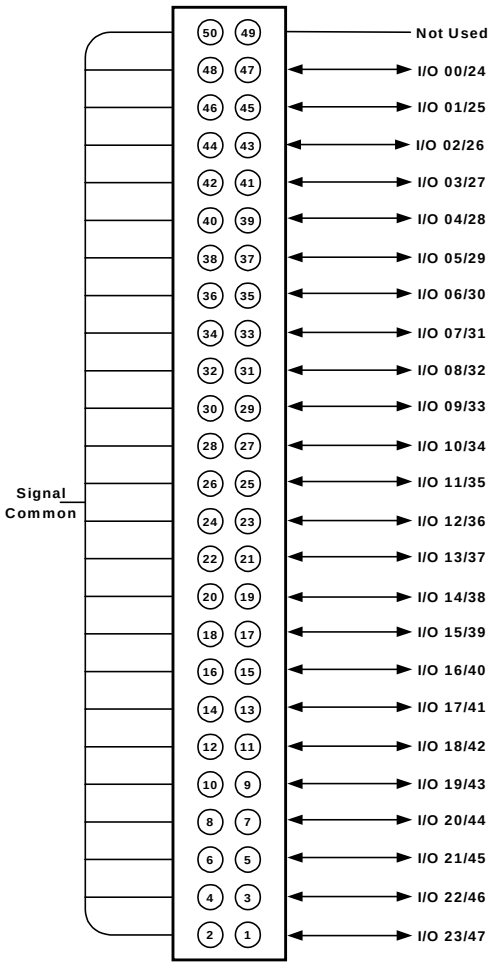
| I/O | Pin | Connector | | I/O | Pin | Connector |
|---|---|---|---|---|---|---|
| B00 | 47 | Top | | B24 | 47 | Bottom |
| B01 | 45 | Top | | B25 | 45 | Bottom |
| B02 | 43 | Top | | B26 | 43 | Bottom |
| B03 | 41 | Top | | B27 | 41 | Bottom |
| B04 | 39 | Top | | B28 | 39 | Bottom |
| B05 | 37 | Top | | B29 | 37 | Bottom |
| B06 | 35 | Top | | B30 | 35 | Bottom |
| B07 | 33 | Top | | B31 | 33 | Bottom |
| B08 | 31 | Top | | B32 | 31 | Bottom |
| B09 | 29 | Top | | B33 | 29 | Bottom |
| B10 | 27 | Top | | B34 | 27 | Bottom |
| B11 | 25 | Top | | B35 | 25 | Bottom |
| B12 | 23 | Top | | B36 | 23 | Bottom |
| B13 | 21 | Top | | B37 | 21 | Bottom |
| B14 | 19 | Top | | B38 | 19 | Bottom |
| B15 | 17 | Top | | B39 | 17 | Bottom |
| B16 | 15 | Top | | B40 | 15 | Bottom |
| B17 | 13 | Top | | B41 | 13 | Bottom |
| B18 | 11 | Top | | B42 | 11 | Bottom |
| B19 | 9 | Top | | B43 | 9 | Bottom |
| B20 | 7 | Top | | B44 | 7 | Bottom |
| B21 | 5 | Top | | B45 | 5 | Bottom |
| B22 | 3 | Top | | B46 | 3 | Bottom |
| B23 | 1 | Top | | B47 | 1 | Bottom |

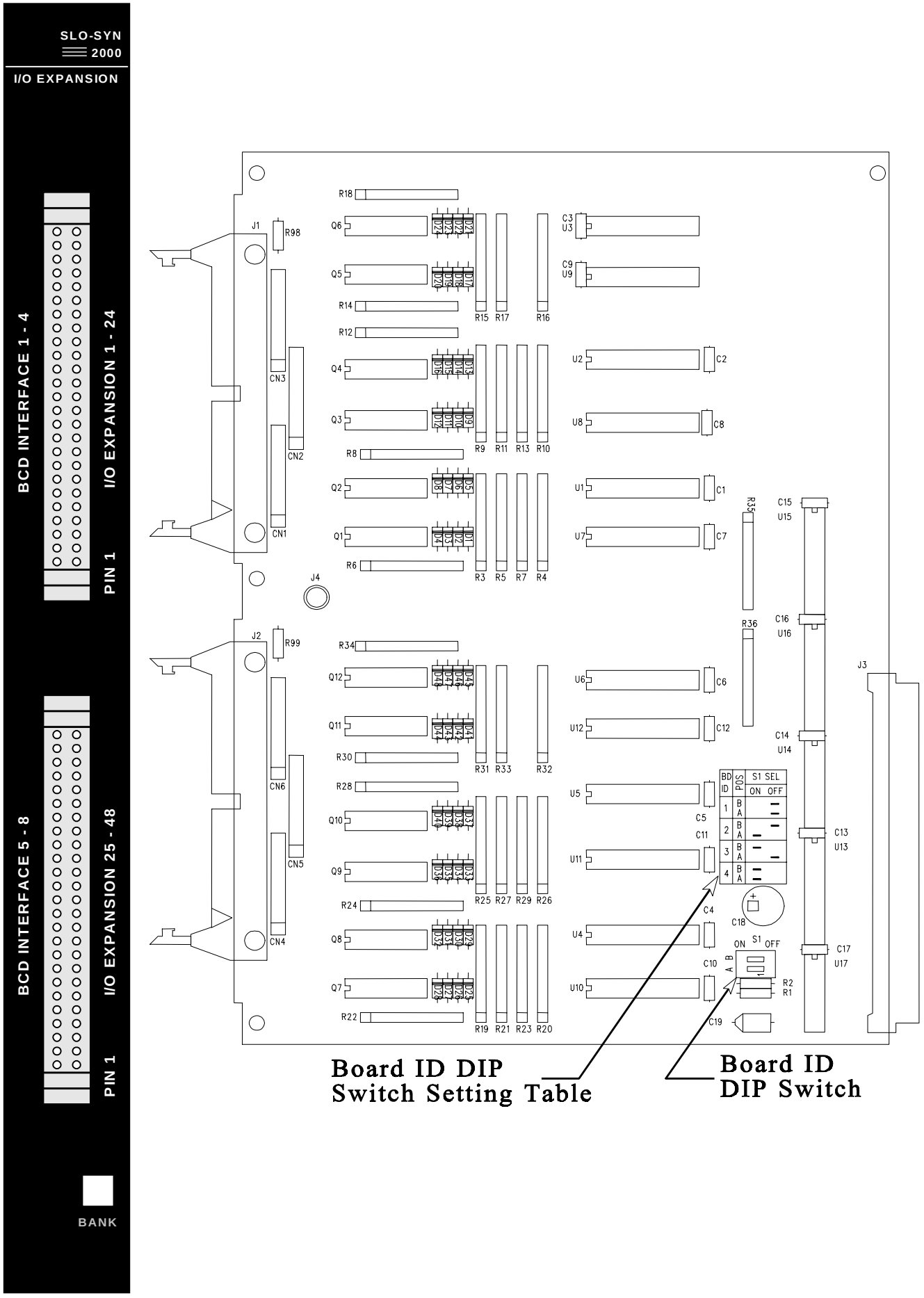
Where: “B” is the board number, 1 through 4.

## 5.7.1 – EXIN/EXOUT assignments

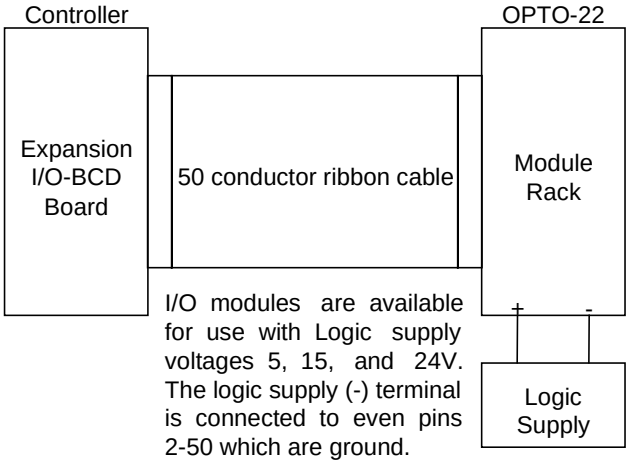
The EXIN and EXOUT commands can be used to access the expansion board I/O. Up to 48 I/O pins can be accessed with these commands. The pin assignment and connector assignment for each I/O point is depicted in the following table.

**Expansion I/O-BCD Connector PinOuts**




SLO-SYN
2000
I/O EXPANSION
BCD INTERFACE 1 - 4
I/O EXPANSION 1 - 24
PIN 1
BCD INTERFACE 5 - 8
I/O EXPANSION 25 - 48
PIN 1
BANK
J1
J2
J3
J4
R98
R99
CN1
CN2
CN3
CN4
CN5
CN6
Board ID DIP
Switch Setting Table
Board ID
DIP Switch
S1 SEL
ON OFF
BD ID
POS
S1
ON
OFF

### Expansion I/O-BCD Port
### Connection to OPTO-22 Module Rack




I/O modules are available for use with Logic supply voltages 5, 15, and 24V. The logic supply (-) terminal is connected to even pins 2-50 which are ground.

| OPTO-22 | |
|---|---|
| Manufacturer | Part Number |
| Crydom | PB-24 |
| Gordos | PB-24 |
| Grayhill | 7ORCK24, 7OMRCQ24 series |
| Potter & Brumfield | ZIO24, ZIOM24 series |
| OPTO-22 | PB-24, PB-24Q, PB-24HQ |

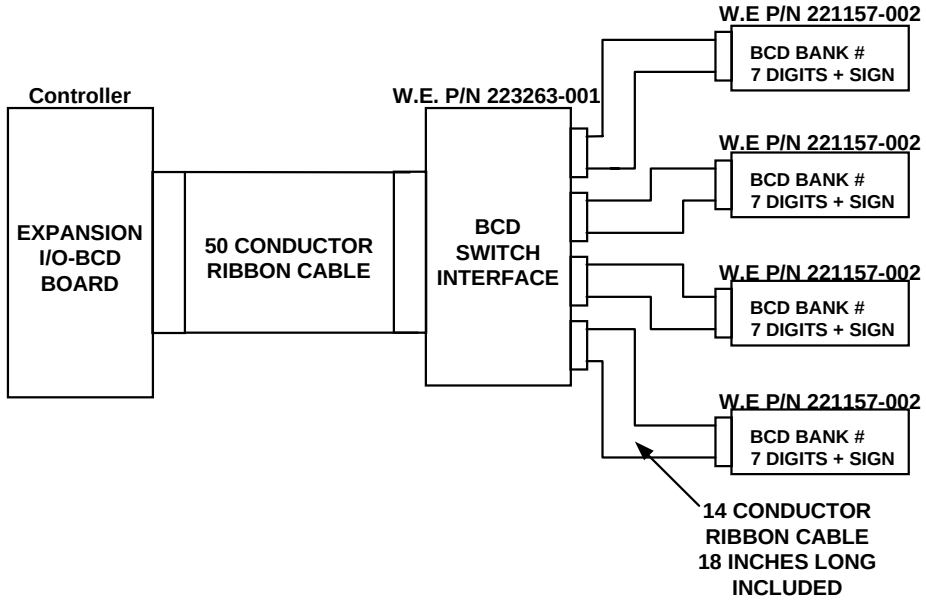
## 5.7.2 - BCD assignments

The BCD commands can be used to access the expansion board BCD switches. Each BCD switch can have up to 7 digits with a sign. Up to 8 sets of BCD switches can be accessed with these commands. The pin assignments and connector assignment for each BCD switch are defined in the following table.

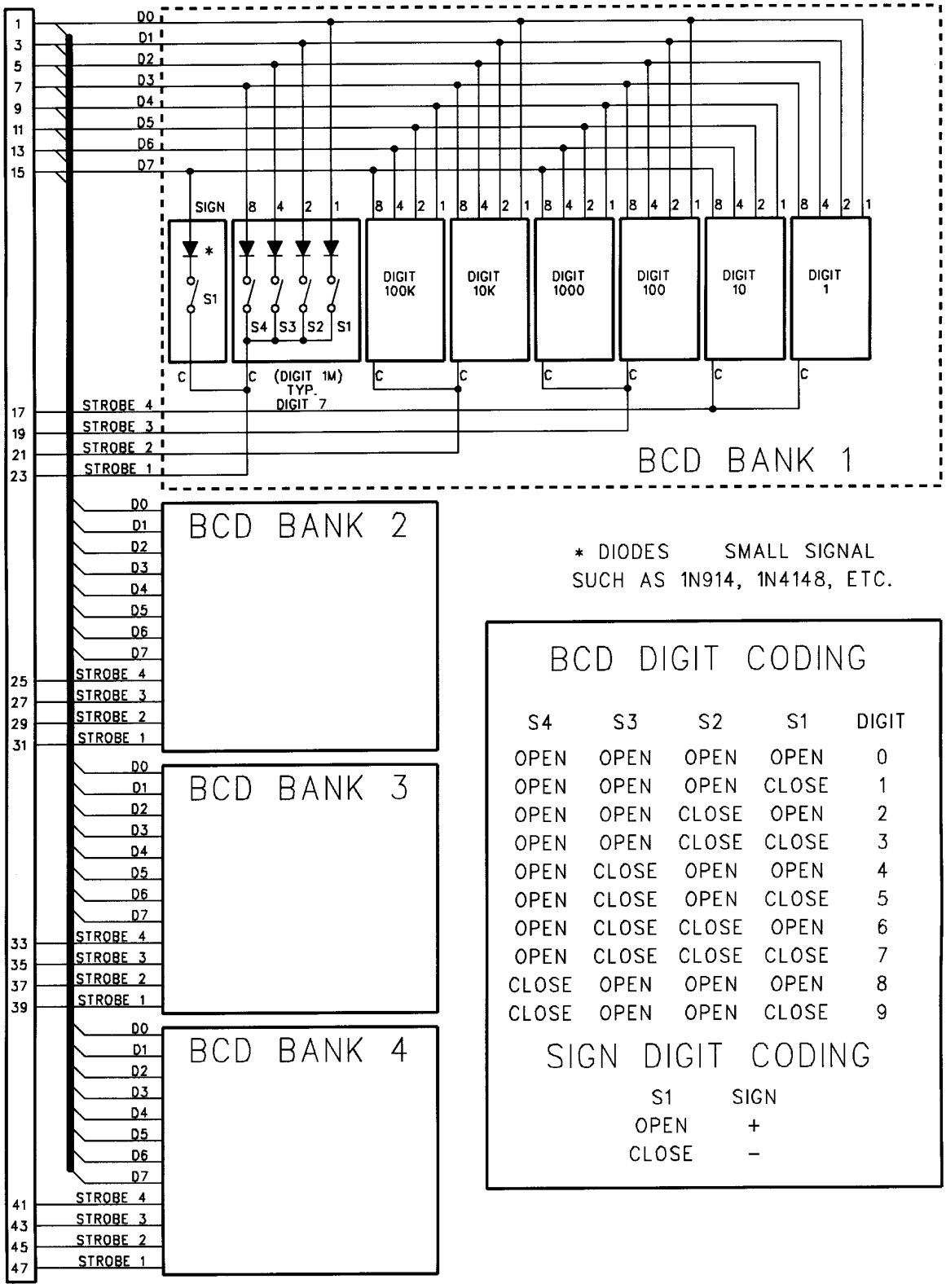
| BCD | Pin | Connector | | BCD | Pin | Connector |
|---|---|---|---|---|---|---|
| B04 | 47,45,43,41 | Top | | B08 | 47,45,43,41 | Bottom |
| B03 | 39,37,35,33 | Top | | B07 | 39,37,35,33 | Bottom |
| B02 | 31,29,27,25 | Top | | B06 | 31,29,27,25 | Bottom |
| B01 | 23,21,19,17 | Top | | B05 | 23,21,19,17 | Bottom |
| B01-B04 | 15,13,11,9,7,5,3,1 | Top | | B05-B08 | 15,13,11,9,7,5,3,1 | Bottom |

Where: “B” is the board number, 1 through 4.

### EXPANSION I/O - BCD PORT
### CONNECTION TO BCD SWITCH BANKS



# EXPANSION I/O-BCD PORT
CONNECTION TO BCD SWITCH BANKS
(USING USER SUPPLIED COMPONENTS)




* DIODES SMALL SIGNAL
SUCH AS 1N914, 1N4148, ETC.

## BCD DIGIT CODING

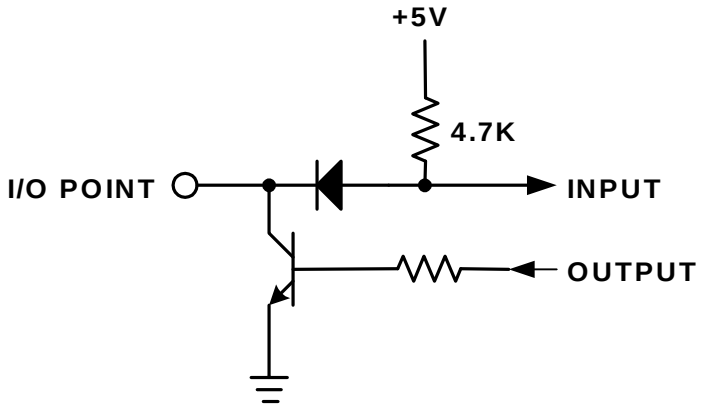
| S4 | S3 | S2 | S1 | DIGIT |
|---|---|---|---|---|
| OPEN | OPEN | OPEN | OPEN | 0 |
| OPEN | OPEN | OPEN | CLOSE | 1 |
| OPEN | OPEN | CLOSE | OPEN | 2 |
| OPEN | OPEN | CLOSE | CLOSE | 3 |
| OPEN | CLOSE | OPEN | OPEN | 4 |
| OPEN | CLOSE | OPEN | CLOSE | 5 |
| OPEN | CLOSE | CLOSE | OPEN | 6 |
| OPEN | CLOSE | CLOSE | CLOSE | 7 |
| CLOSE | OPEN | OPEN | OPEN | 8 |
| CLOSE | OPEN | OPEN | CLOSE | 9 |

## SIGN DIGIT CODING

| S1 | SIGN |
|---|---|
| OPEN | + |
| CLOSE | – |

| I/O Expansion Board | | |
|---|---|---|
| Pins | Description | Specification |
| 41,43,45,47 | BCD 4 or BCD 8 strobes<br>EXIN/EXOUT B03/27 to B00/24 | Input: On state 0 to +1.5 volts, 1 ma @ 0v,<br>Off state +2.9 to +30 volts.<br>Output: open circuit +30 volt maximum,<br>saturation voltage +0.5 volts @ 15 ma. |
| 33,35,37,39 | BCD 3 or BCD 7 strobes<br>EXIN/EXOUT B07/31 to B04/28 | Input: On state 0 to +1.5 volts, 1 ma @ 0v,<br>Off state +2.9 to +30 volts.<br>Output: open circuit +30 volt maximum,<br>saturation voltage +0.5 volts @ 15 ma. |
| 25,27,29,31 | BCD 2 or BCD 6 strobes<br>EXIN/EXOUT B11/35 to B08/32 | Input: On state 0 to +1.5 volts, 1 ma @ 0v,<br>Off state +2.9 to +30 volts.<br>Output: open circuit +30 volt maximum,<br>saturation voltage +0.5 volts @ 15 ma. |
| 17,19,21,23 | BCD 1 or BCD 5 strobes<br>EXIN/EXOUT B15/39 to B12/36 | Input: On state 0 to +1.5 volts, 1 ma @ 0v,<br>Off state +2.9 to +30 volts.<br>Output: open circuit +30 volt maximum,<br>saturation voltage +0.5 volts @ 15 ma. |
| 1,3,5,7,9,11,13,15 | BCD 1-4 or BCD 5-8 data bus<br>EXIN/EXOUT B23/47 to B16/40 | Input: On state 0 to +1.5 volts, 1 ma @ 0v,<br>Off state +2.9 to +30 volts.<br>Output: open circuit +30 volt maximum,<br>saturation voltage +0.5 volts @ 15 ma. |

**Expansion I/O BCD Equivalent Circuit**



## 5.8 – Digital I/O Board

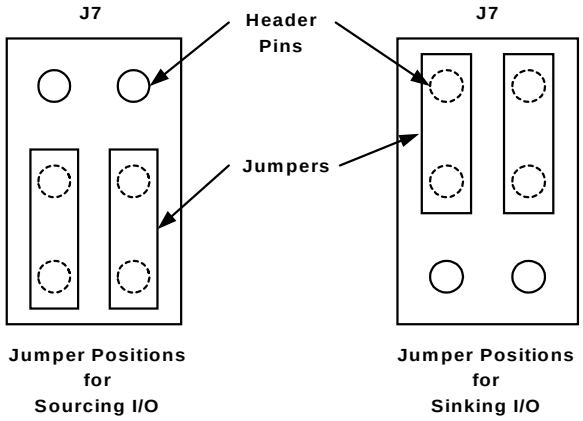
The Digital I/O-BCD board has been designed to interface to switches and/or relays. There are 24 optically isolated inputs and 16 optically isolated outputs per board.

If the MX2000 controller is an MX2A or MX6A the first digital I/O board is the Power supply board and there are 16 inputs and 8 outputs available (101-116). Up to 4 boards can be interfaced to an MX2000-8 controller. The ID for each board is selected via 2 dip switches located on the board.

## 5.8.1 – Input Connector

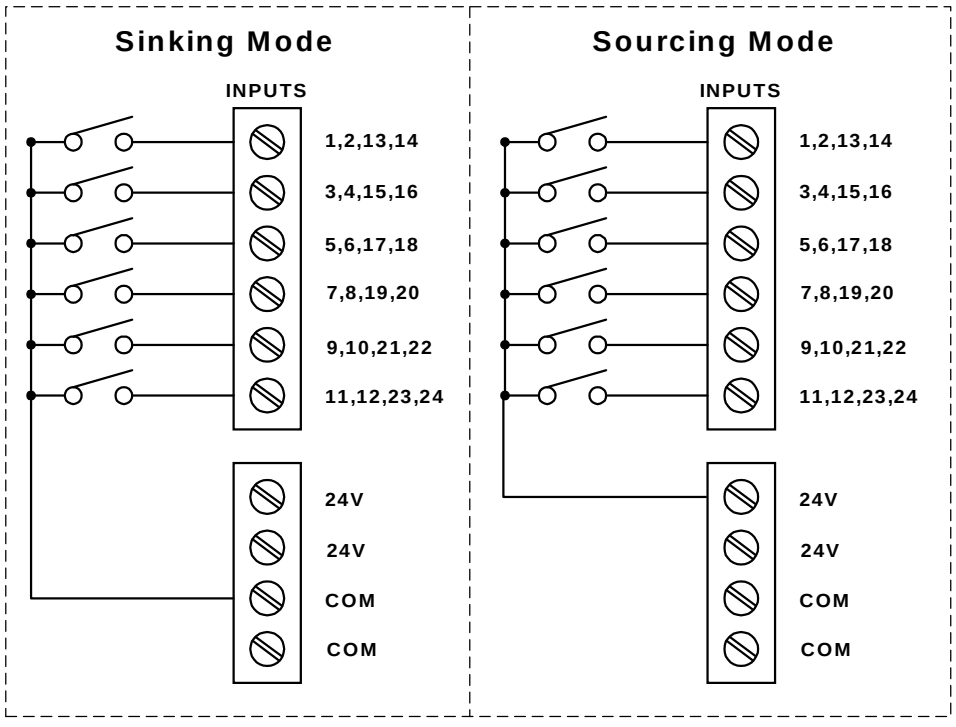
This connector has 24 optically isolated inputs that can be configured for current sinking or sourcing. The sinking/sourcing selection is done on the digital I/O card using jumpers. The factory setting is sinking.

**Note: The movement of both jumpers is required for proper operation. If only one jumper is moved the digital inputs and outputs modes will be different.**




| Inputs | | |
|---|---|---|
| Signal Name | Description | Electrical Specification |
| B01 to B24 | IN (B01-B24) | Sink Mode: On state 0 to +12 volts, On state current 2.3 ma @ +12v, 6.5 ma @ 0v. Source Mode: On state +10 to +24 volts, On state current 2.3 ma @ +10 volts, 6.5 ma @ +24 volts. |

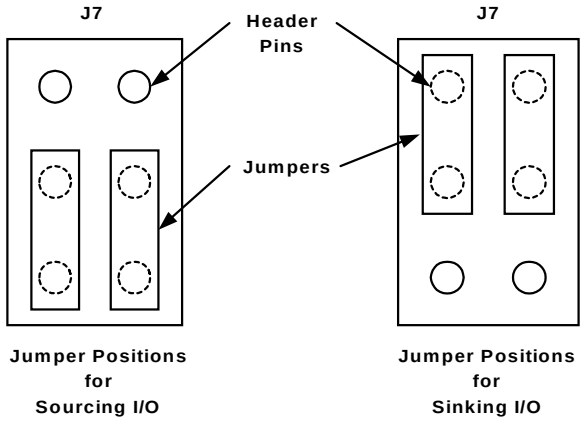
Where: B is the board number, 1 through 4.



## 5.8.2 – Output Connector

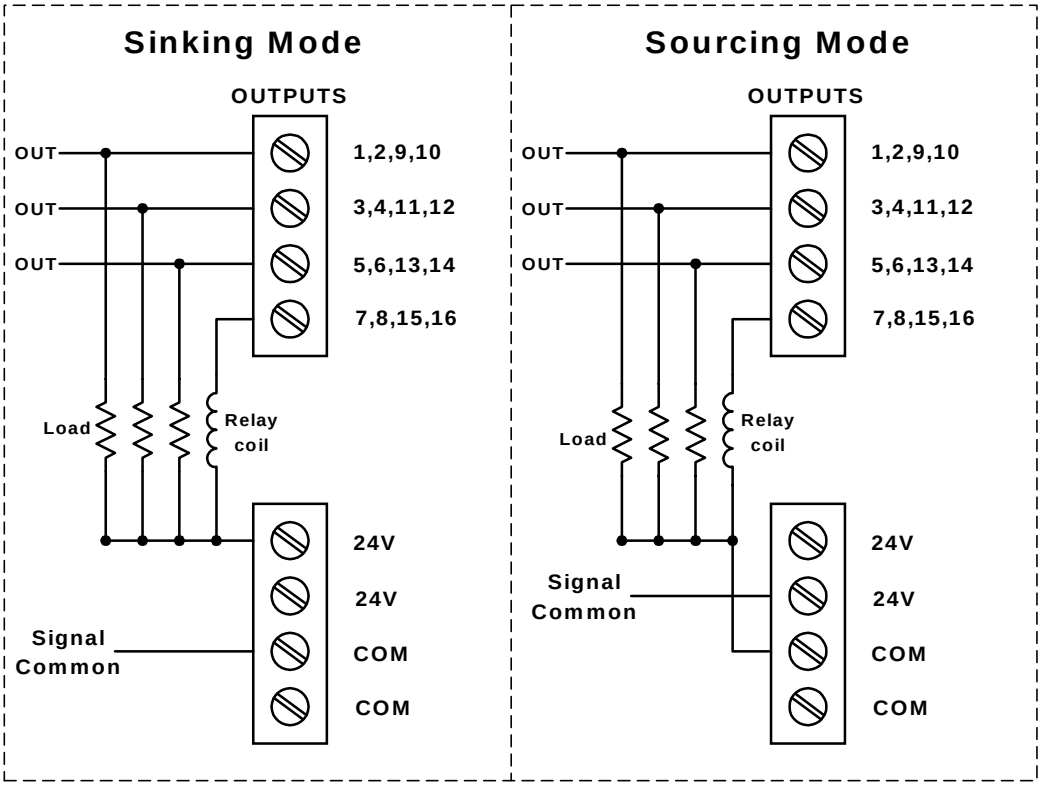
This connector has 16 optically isolated outputs that can be configured for current sinking or sourcing. The sinking/sourcing selection is done on the digital I/O card using a jumper. The factory setting is sinking.

**Note: The movement of both jumpers is required for proper operation. If only one jumper is moved the digital inputs and outputs modes will be different.**




| Outputs | | |
|---|---|---|
| Signal Name | Description | Electrical Specification |
| B01 to B16 | OUT (B01-B16) | Sink Mode: Voltage rating 24 volts, On state 0 to +2 volts @ 50 ma, Off state leakage 0.6 ma maximum @ +24v. Source Mode: Voltage rating 24 volts, On state +20 to +24 volts @ 50 ma, Off state leakage 0.6 ma maximum. |

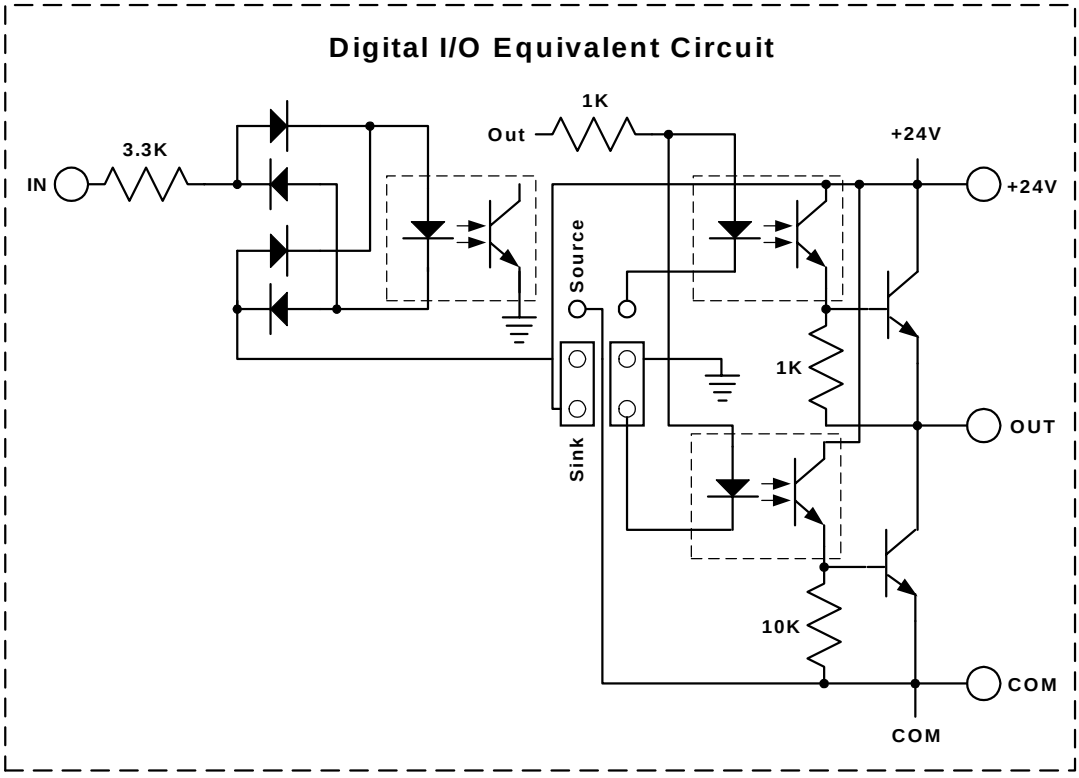
Where: B is the board number.

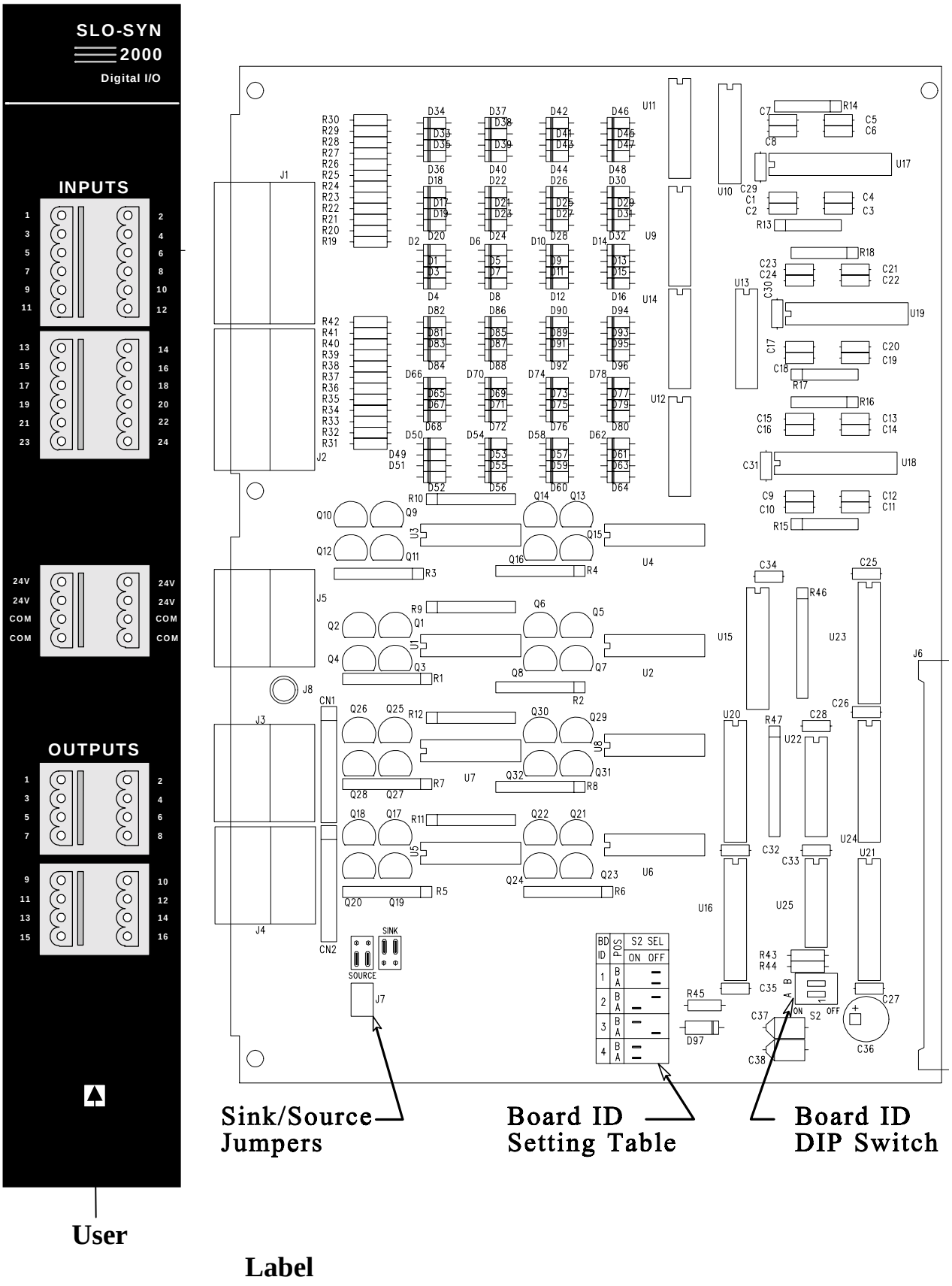


## 5.8.3 – Internal Power Supply

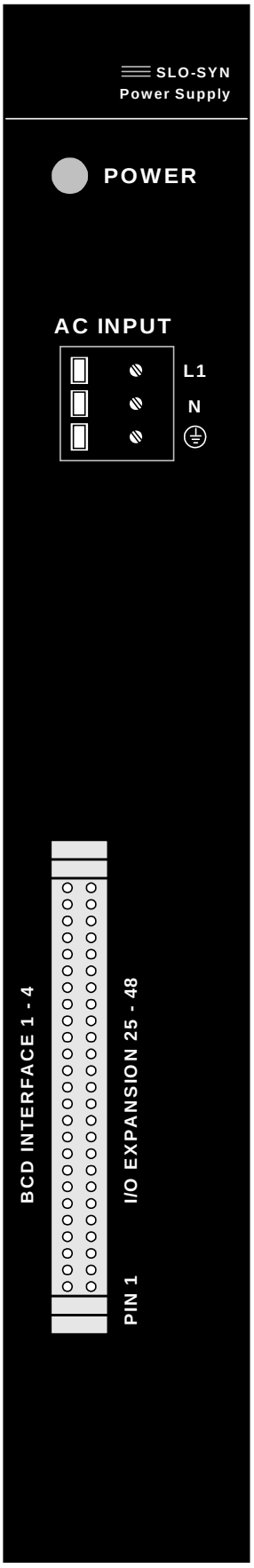
This connector has 4 terminals for +24V @ .75 amps and 4 terminals for COM. These terminals are used as signal returns for inputs and signal common for outputs.

| Internal Power Supply | |
|---|---|
| Mode | Description |
| Sink | The COM terminals are used as the return source for the inputs and the signal common for the outputs. |
| Source | The +24 terminals are used as the return source for the inputs and the signal common for the outputs. |




SLO-SYN
2000
Digital I/O
INPUTS
OUTPUTS
24V
COM
Sink/Source
Jumpers
Board ID
Setting Table
Board ID
DIP Switch
User
Label

## 5.9 – MX2 and MX6 Power Supply Board




This board contains the AC input terminals and interface for 24 non-isolated bi-directional I/O. This board is assigned Expansion I/O board #1, EXIN(100-123) and EXOUT(100,123) . The range for the AC input is 90 to 265 VAC at 50/60 hz.

## 5.9.1 – AC Input

The AC input is connected to a terminal strip.

| Terminal | Description | Lead Color North America Standard | Lead Color European Standard |
|---|---|---|---|
| L1 | Line or Hot | Black | Brown |
| N | Common or Neutral | White | Blue |
| ⏚ | Ground | Green | Green with Yellow Stripe |

## 5.9.2 – EXIN/EXOUT assignments

The EXIN and EXOUT commands can be used to access the expansion board I/O. Up to 24 I/O pins can be accessed with these commands.

| I/O | Pin | | I/O | Pin |
|---|---|---|---|---|
| 100 | 47 | | 112 | 23 |
| 101 | 45 | | 113 | 21 |
| 102 | 43 | | 114 | 19 |
| 103 | 41 | | 115 | 17 |
| 104 | 39 | | 116 | 15 |
| 105 | 37 | | 117 | 13 |
| 106 | 35 | | 118 | 11 |
| 107 | 33 | | 119 | 9 |
| 108 | 31 | | 120 | 7 |
| 109 | 29 | | 121 | 5 |
| 110 | 27 | | 122 | 3 |
| 111 | 25 | | 123 | 1 |

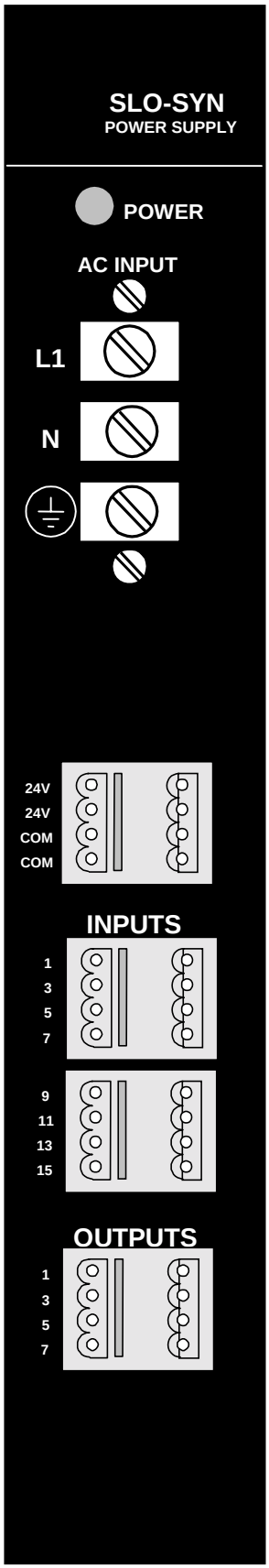
See Section 5.7.1 for more details.

## 5.9.3 – BCD assignments

The BCD command can be used to access the expansion BCD switches. Each BCD switch can have up to 7 digits with a sign. Up to 4 sets of BCD switches can be accessed with this command.

| BCD | Pin |
|---|---|
| 104 | 47,45,43,41 |
| 103 | 39,37,35,33 |
| 102 | 31,29,27,25 |
| 101 | 23,21,19,17 |
| 101-104 | 15,13,11,9,7,5,3,1 |

See Section 5.7.2 for more details.




## 5.10 – MX2A and MX6A Power Supply Board

This board contains the AC input terminals, interface for 16 optically isolated input and 8 optically isolated outputs. This board is assigned digital I/O board #1, IN(101-116) and OUT(101,108) . The range for the AC input is 90 to 265 VAC at 50/60 hz.

## 5.10.1 – AC Input

The AC input is connected to a terminal strip.

| Terminal | Description | Lead Color North America Standard | Lead Color European Standard |
|---|---|---|---|
| L1 | Line or Hot | Black | Brown |
| N | Common or Neutral | White | Blue |
| ⏚ | Ground | Green | Green with Yellow Stripe |

## 5.10.2 – Input Connector

This connector has 16 optically isolated inputs.

| Inputs | | |
|---|---|---|
| Signal Name | Description | Electrical Specification |
| 101 to 116 | IN (101-116) | Sink Mode: On state 0 to +12 volts, On state current 2.3 ma @ +12v, 6.5 ma @ 0v.<br>Source Mode: On state +10 to +24 volts, On state current 2.3 ma @ +10 volts, 6.5 ma @ +24 volts. |

See Section 5.8.1 for more details.

## 5.10.3 – Output Connector

This connector has 8 optically isolated outputs.

| Outputs | | |
|---|---|---|
| Signal Name | Description | Electrical Specification |
| 101 to 108 | OUT (101-108) | Sink Mode: Voltage rating 24 volts, On state 0 to +2 volts @ 50 ma,<br>Off state leakage 0.6 ma maximum @ +24v.<br>Source Mode: Voltage rating 24 volts, On state +20 to +24 volts @ 50 ma, Off state leakage 0.6 ma maximum. |

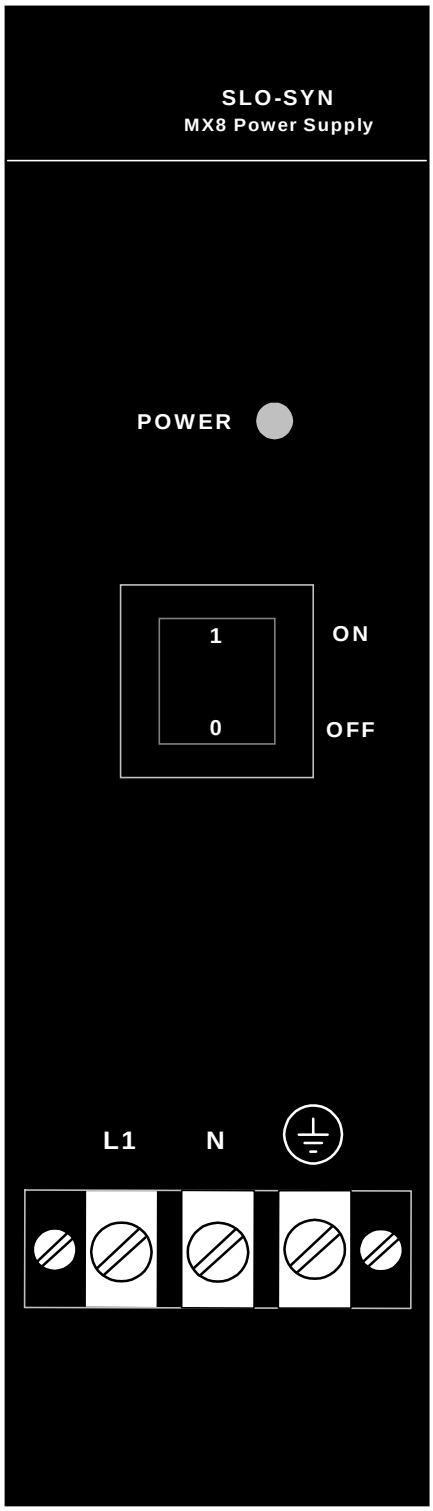
See Section 5.8.2 for more details.

## 5.10.4 – Internal Power Supply

This connector has 4 terminals for +24V and 4 terminals for COM. These terminals are used as signal returns for inputs and signal common for outputs.

| Internal Power Supply | |
|---|---|
| Mode | Description |
| Sink | The COM terminals are used as the return source for the inputs and the signal common for the outputs. |
| Source | The +24 terminals are used as the return source for the inputs and the signal common for the outputs. |

See Section 5.8.3 for more details.




## 5.11 – MX8 Power Supply Board

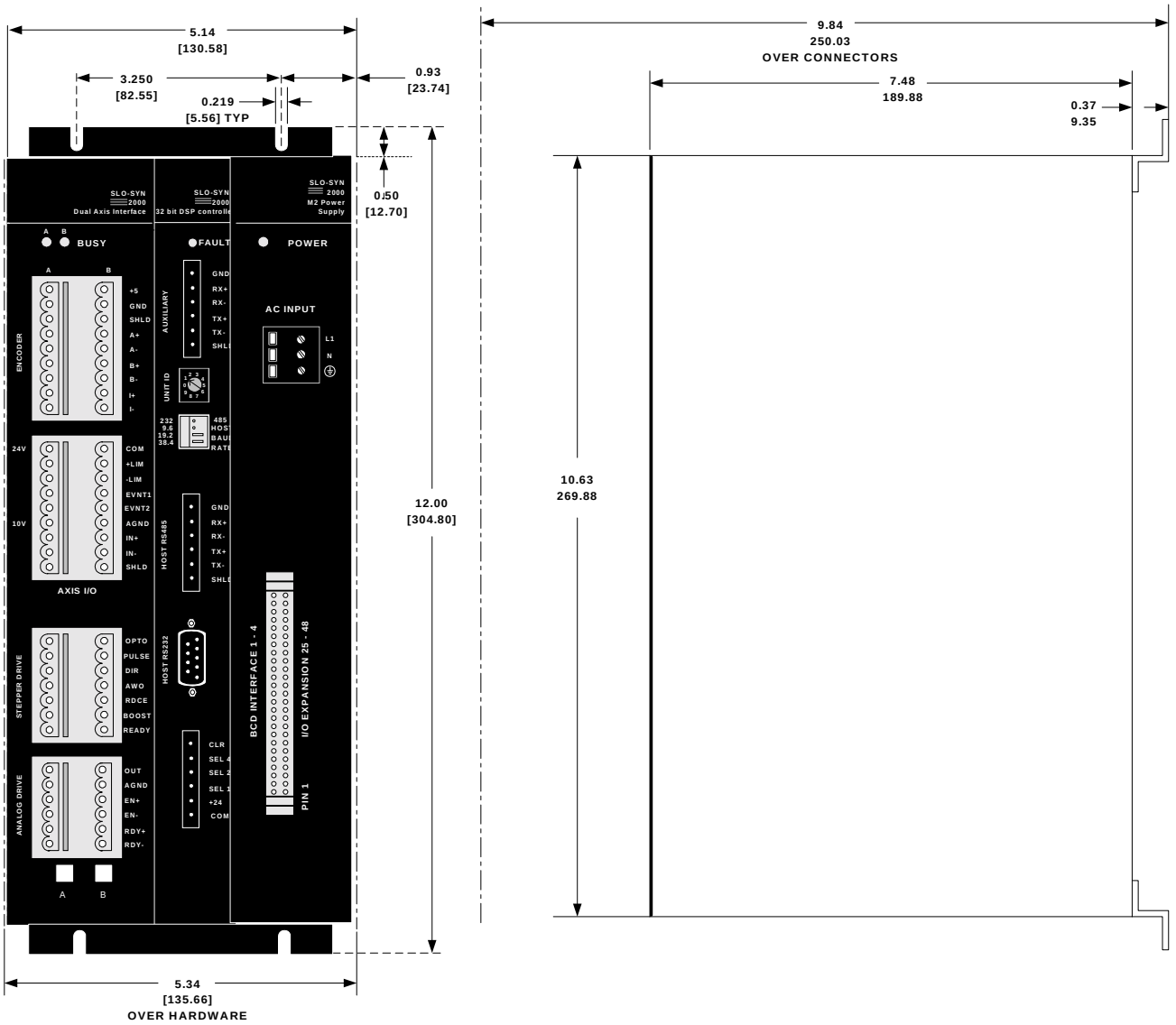
This board contains the AC input terminals with no I/O connections. The input voltage range is 90 to 132 VAC or 175 to 264 VAC 50/60 hz. The MX8 will not operate correctly if the input voltage is not within the two ranges. No operator action is required, the MX8 automatically senses the input voltage and configures itself to operate at either AC input voltage range.

## 5.11.1 – AC Input

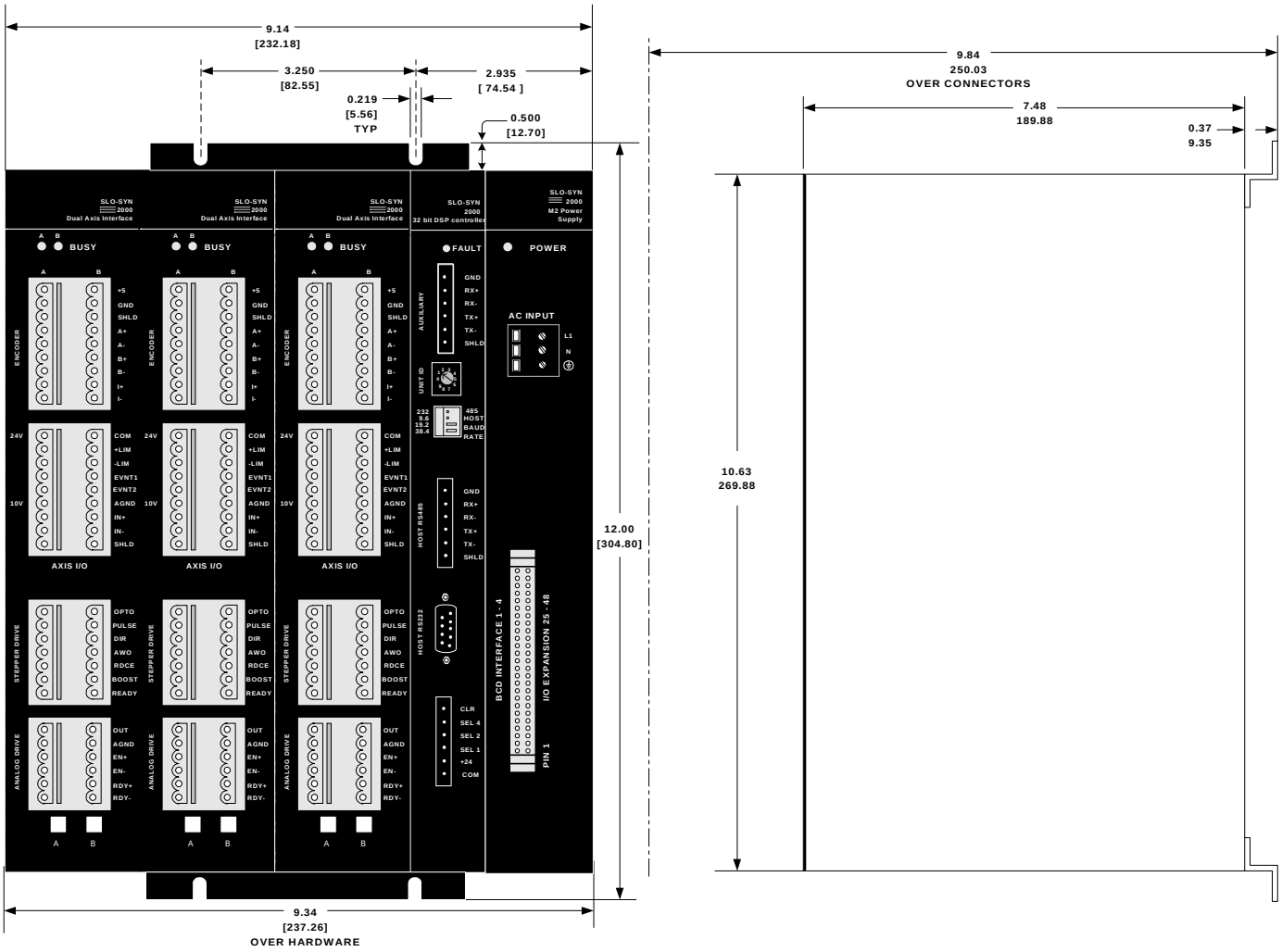
The AC input is connected to a terminal strip.

| Terminal | Description | Lead Color North America Standard | Lead Color European Standard |
|---|---|---|---|
| L1 | Line or Hot | Black | Brown |
| N | Common or Neutral | White | Blue |
| ⏚ | Ground | Green | Green with Yellow Stripe |

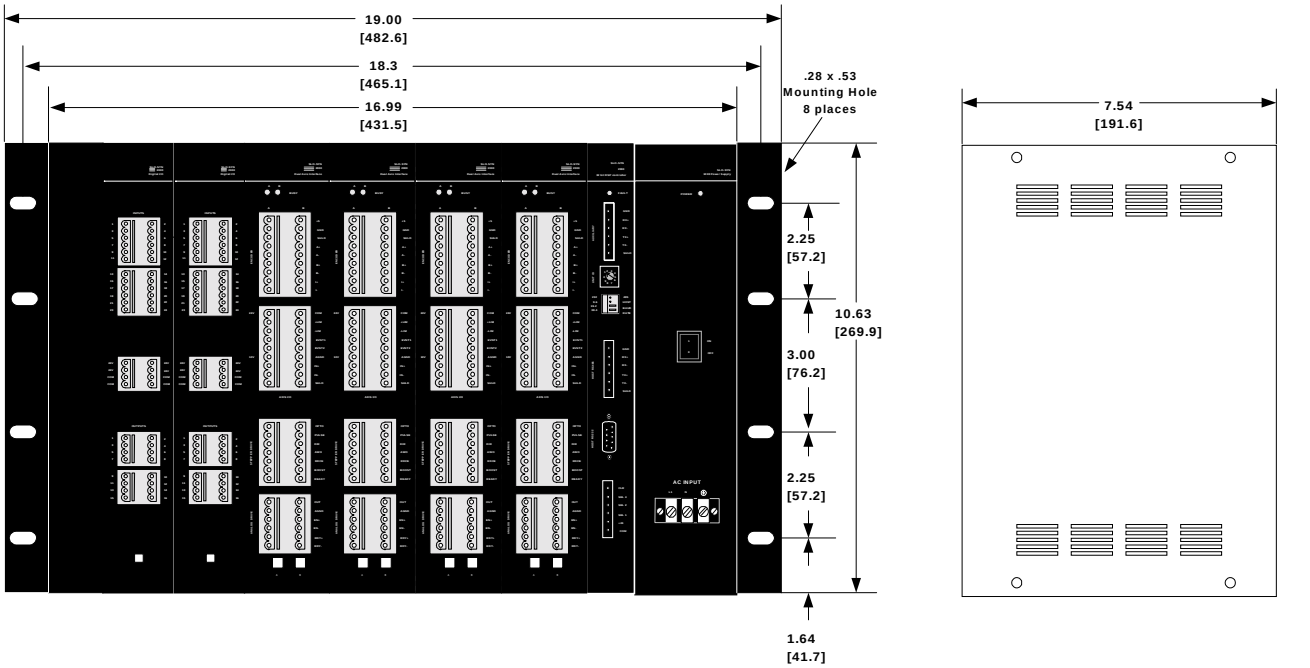
# 5.12 – MX2 Outline



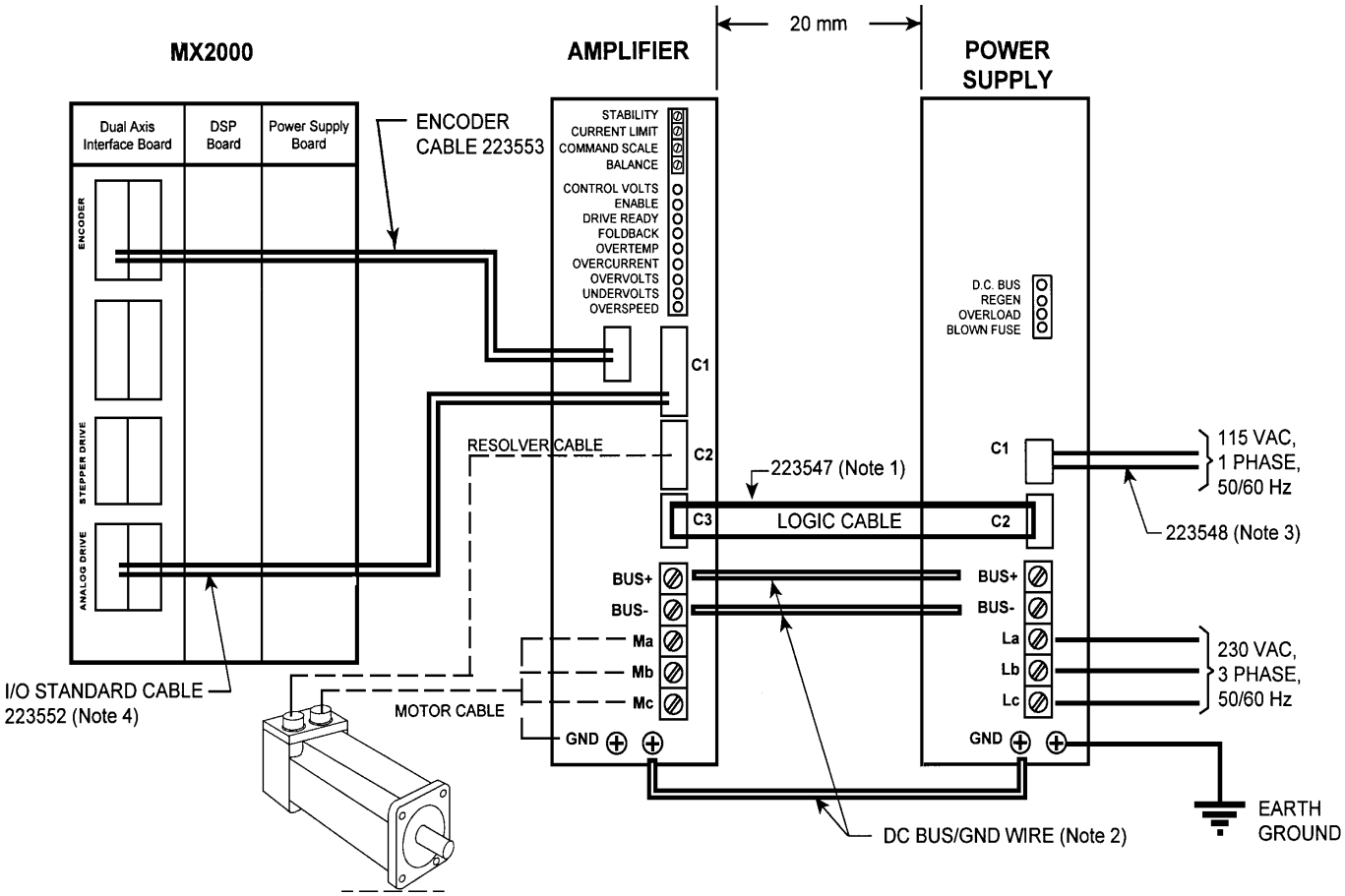
## 5.13 – MX6 Outline




## 5.14 – MX8 Outline



# 5.15 – MX & SERVO AMPLIFIER CONNECTION DIAGRAM



This page left intentionally blank

# SECTION 6
# Motion Controller Programming Interface

# 6.1 - Programming

## 6.1.1 - General Description of Programming

This section provides an overview to the process of programming a Controller. Once the "logic" behind the various commands are understood, programming your Controller will be seen as a straightforward process.

Programming of any sort requires planning and forethought. Programming your Controller is no exception. This section will provide aids to facilitate your planning process. Be patient! Allow time for mistakes, adjustments ("debugging"), and experimentation.

### 6.1.1.1 - What is Programming?

At its most basic level, a computer program is a means of using electronic digital signals (simple ON and OFF) to produce certain results from a machine. A line of code, or "command string," is built up from the presence (On) or absence (Off) of electrical signals. On or Off signals, called "Bits," are bunched together to form "Bytes", or groupings that are coded into what we recognize as alphabetical characters or numbers. (This character coding is accomplished via the ASCII code - see Glossary Section for further details.

A program is a list of discrete lines or command strings that, taken together in sequence, provide the information needed to get a machine to perform your predetermined sequence of instructions. These instructions can, in the case of a Programmable Motion Controller, cause the motor to move at certain speeds and for given distances, read various inputs or set outputs, or send and receive messages from an operator interface panel, all used to accomplish different machine-related tasks.

### 6.1.1.2 – What's in a Program?

A program consists of many individual lines organized in a prescribed sequence. The Controller uses an English language, BASIC-type computer programming language ("SEBASIC"). This makes it easy and intuitive to write and read machine control programs. The language we have designed supports many higher-level-language features, such as statement labels, subroutines, for-next and do-while loops for program flow control. This makes it easy to write concise, well-organized, easily debugged programs. Also, there are built-in mathematical, Boolean, array, and trigonometry functions to perform complex calculations. The rich string-handling functions allow easy data input and message writing when using external operator interface panels. Finally, the motion, I/O, and timing commands are easy to understand, remember, and apply.

In addition to program lines, the controller needs and stores (separate from the commands) a series of set-up parameters in a "header file". The MCPI compiler program automatically creates this file.

### 6.1.1.3 - How is the Controller Programmed?

There are two primary ways to set up and program your Controller. Both involve the use of a personal computer (PC). One is a programming environment called "MCPI", and is supplied on diskette with your unit. Section 6.3 of this manual gives detailed instructions on installing and using this tool to develop your application. A second way is to create your program using any standard text editor or word processor. Write the SEBASIC commands, save the file as an ASCII format then use the MCPI to compile your code and download it.

The types of commands your Controller can accept are pre-set. Thus each command is assigned a "name". These commands are explained in detail in the Software Reference Section of this Manual.

Commands are performed via the statement lines in your program. The program is a sequence of commands that control the motor and motion-related events you want to happen in a particular period of time. Thus, the sequence of commands is critical to the proper operation of your system.

## 6.1.2 - What are "Host Commands"?

There are also "Host Commands" available for certain programming needs. These commands go straight from your input device (computer or terminal, for example), to the Controller, and override the normal sequence of operation directed by your program. These are useful for manually controlling a machine that normally operates under program control.

## 6.1.3 – Memory Types and Usage

A program is stored in Memory. There are two kinds of memory. RAM (Random Access Memory) is called "Volatile Memory" because when power is removed from the Controller, all the electrical signals in that memory are lost, and accordingly, the information stored in that memory is lost. The Controller, for example, stores some transient information in RAM.

The second kind of memory is "Non-Volatile Memory", such as Flash memory, EEPROM (Electrically Programma-

ble Read-Only Memory), or a BBRAM (Battery Backed RAM). The electrical codes stored in this type of memory are not lost when external power is removed from the Controller. The Controller uses a battery backed RAM for storing NVR variables (1-2048). The controller stores the operating system as well as user programs in Flash memory. This memory is located on the DSP Controller card.

A program in your Controller can have hundreds, or even thousands of program lines. Because of the wide variety of program commands, and the variable line lengths allowed, it is impossible to state how many lines of code can be stored in the controller. However, the user memory available is 2044 sectors of 128 bytes per sector, for a total of 261,632 bytes of program space. The FREE command may be used to determine how much memory is available; see Section 7 for details on using this.

## 6.1.4 - References

Newcomers to programming are encouraged to obtain a copy of an elementary text on computer programming. Since your Controller uses a modified form of the familiar "BASIC" computer language, you may refer to a book on using BASIC. There are a great number of such books available in the technical or computer section of your local library or bookstore. We have found that books by SAM's, Microsoft Press particularly ARunning MS DOS QBASIC,@ by Michael

Haverson & David Rygmyr, and those by the Waite Group are among the most helpful.

### Section 6.2 - Multi-Tasking Operations

A single computer can only do one thing at a time. However, a complex motion control system needs to have many tasks done, all at once. An effective way to do this is with a very fast microprocessor (or DSP), running a preemptive multi-tasking operating system. This causes a single computer to appear to be doing several things simultaneously. The computer works on one program for a while, then switches to another program for a while, and after all programs have been serviced, goes back to the first, and repeats the cycle.

With a fast computer, the time slice for each program can be small and the outward appearance is that a separate computer is running each program.

The Controller uses this approach to give the user up to 7 "virtual" motion controllers in a single package. An additional advantage of multi-tasking is that information can be easily shared among the 7 virtual controllers. The Controller runs 1 system task and up to 7 user tasks. Task 0 is a system task, which always runs. It processes commands received over the Host serial port. Up to 7 user, SEBASIC, BASIC programs (Task 1 - Task 7) may be running in addition to Task 0.

Every 256 microseconds, task execution is interrupted in order to perform the time-critical functions associated with motion control. Execution of the next task is resumed upon completion of the interrupt routine. The execution sequences for a 1-user-task system and for a 7-user-task system are shown below.

If an application uses 7 tasks, then each task will be serviced once every 2.048 ms. If fewer tasks are used, then the service time decreases. A one-task system would be serviced every 512 us. The service time can be calculated by the following formula:

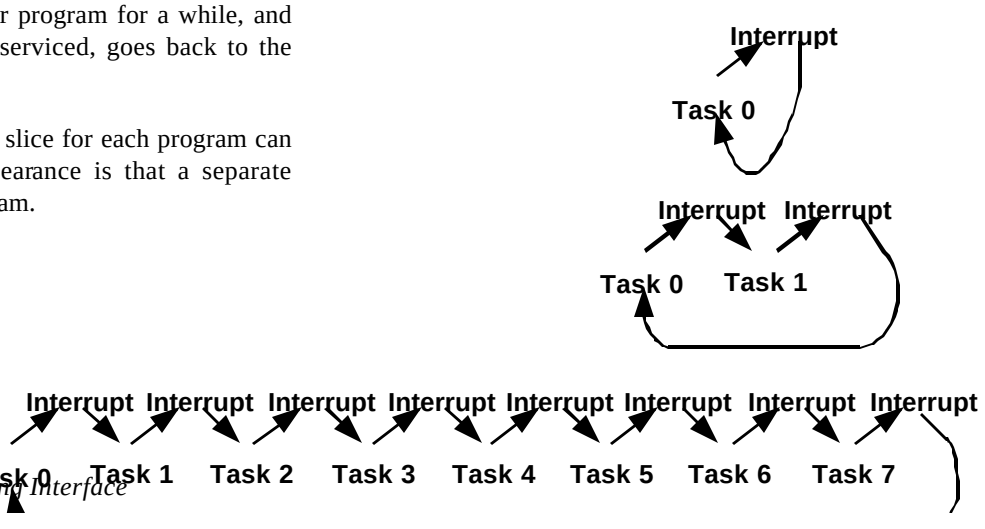
$T_{service} = (n + 1) \times 256$ microseconds
where n is the number of user tasks.

Large, complicated applications typically consist of several independent operations occurring simultaneously. Multi-tasking allows the user to program the application as a collection of several smaller and hence simpler applications.

A typical example of the use of tasks is to break up the system functions into logical groups. For example, control of a large machine might assign functions to tasks as follows:

Task 1 - Motion on axes 1 and 2
Task 2 - Handling all inputs and outputs
Task 3 - Communicating with operator interface panel

Using tasks and multi-tasking allows programs to be more modular, hence they are easier to write, debug, and maintain.



## 6.2.1 – Multi-Tasking timing

The Tasks are switched by the interrupt routine every 256 micro-seconds. The number of tasks being switched is dependent on the number of tasks that are loaded for the user project and on project execution. If program execution is not taking place only Task 0, Host command execution is taking place. If program execution is taking place the active user project tasks and Task 0 will be switched by the interrupt every 256 micro-seconds. The diagrams on the right illustrate the timing for no program execution, single task program execution and a seven-task project being executed. If a task is stopped during program execution, that task will no longer be serviced.

When it is required to poll I/O through out program execution, either dedicate a task to accomplish this or poll the I/O in a program loop.

# 6.3 - Motion Controller Programming Interface (MCPI)

## 6.3.1 - Software Installation

The Motion Controller Programming Interface (MCPI) provides the means by which an application can be fully developed and the controller can be operated using a personal computer (PC). The application can be written, compiled and downloaded to the controller, using the Motion Controller Programming Interface. In addition, a ATerminal Mode@ is provided for operating the controller from your computer.

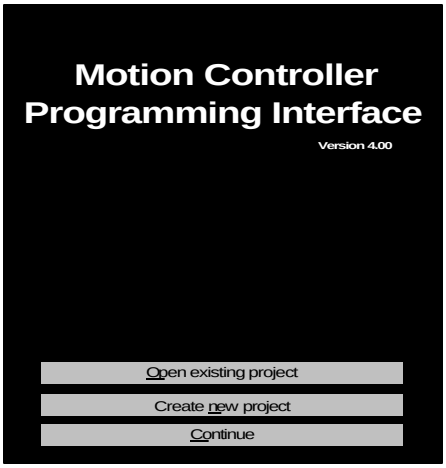
**Installation Instructions**

1) If Windows® is not already running, type **WIN** at the DOS prompt, and press **ENTER**.
2) Insert the **MCPI** Program Disk into drive A: (or B:).
3) For Windows 3.1 Click on the **FILE** menu in the Program Manager.
   For Windows 95/98 Click on the **Start** button on the desk top.
4) Select **RUN**... to display the Run Dialog box.
5) Type **A:setup** (or B:setup) and click **OK**.
6) The installation program will display the MCPI File Manager Setup screen. Follow the prompts on the screen to complete the installation.
7) After the program files have been installed, the installation will create a new Windows group.
8) Remove the installation disk. This concludes the installation.

## 6.3.2 - Starting the MCPI Environment

1) If Windows is not already running, type **WIN** at the DOS prompt, and press **ENTER**.
2) Double click on the **MCPI** Icon.
3) The opening screen will appear.

## 6.3.2.1 – The MCPI opening screen




**Open existing project** opens up an existing project.

**Create new project** creates a new project.

**Continue** enters the MCPI with no selection.

## 6.3.3 - Setting Communication Parameters

The MCPI uses the computer serial port to communicate with the Controller. The MCPI supports the use of four serial ports, (Com 1, Com 2, Com 3 or Com 4). To communicate, an XON - XOFF protocol is used. This protocol needs only three wires to establish a communication link between the computer and the controller. These wires should be connected to transmit (TX), receive (RX) and common (V0) as follows:

| Computer | | Controller |
|---|---|---|
| TX | ---------- | RX |
| RX | ---------- | TX |
| V0 | ---------- | V0 |

**Note 1: The 9-conductor cable supplied in the Controller accessory kit (shipped with your unit) should allow easy connection to your PC's serial port. A 25-to-9 pin adapter is required (user supplied) if the PC port is a 25-pin style.**

**Note 2: Consult your computer manual for the correct pin out of it's serial port.**

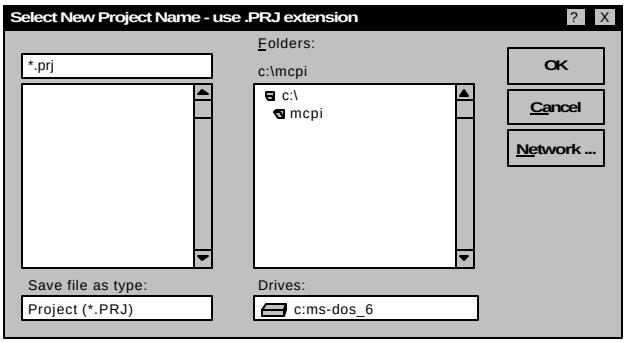
The MCPI supports four-baud rates: 4800, 9600, 19200 and 38400. To set up the serial port, baud rate and Terminal Emulation Mode used for communications, select the **Configure Com Port** item under the **System** menu. The serial word length, parity, and number of stop bits are fixed at 8, none, 1 respectively. The baud rate for the Controller can be set via switches on the front panel. Both the Controller and the MCPI are set to default to a baud rate of 9600 when shipped. The MCPI will also default to Com 1.

**Note 3: The Terminal Emulation Mode should be set to TTY on the Configure Com Port screen.**

Selecting the Terminal item under the Utility menu allows testing of the serial communications to the Controller. Simply click on the **Software Revision** command button and the Controller will return the software revision information, which will be displayed on the terminal screen.

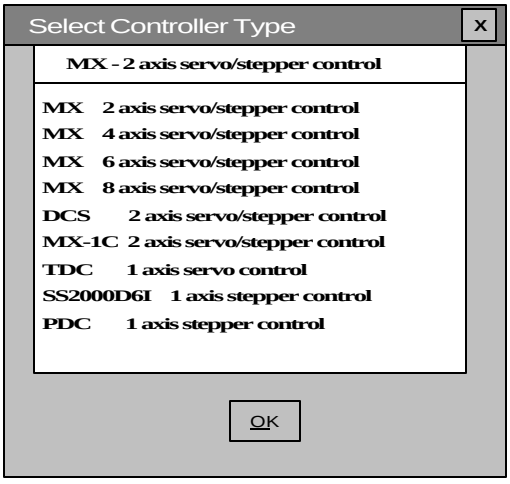
## 6.3.4 – Creating a new project

To create a new project either click on the **Create new project** command button on the Opening screen or the **New** item on the **Project** pull down menu.

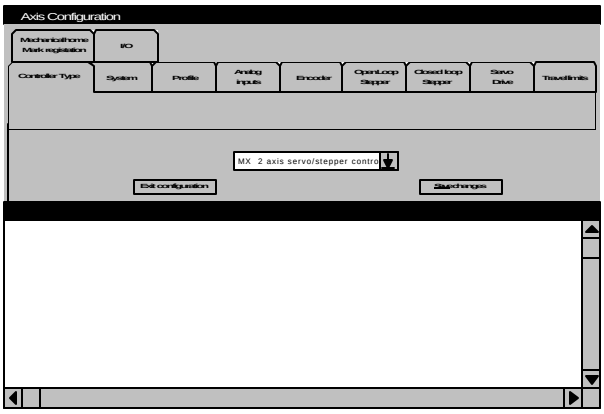



Enter the name of the project with a **.prj extension**. The directory of the project can also be selected at this time. To accept the name and directory click on the **OK** command button.

The controller type can now be selected by clicking on the desired selection and then clicking on the **OK** command button.




The controller type folder screen is now accessed. This screen allows access to the project folders by clicking on the desired folder tab.

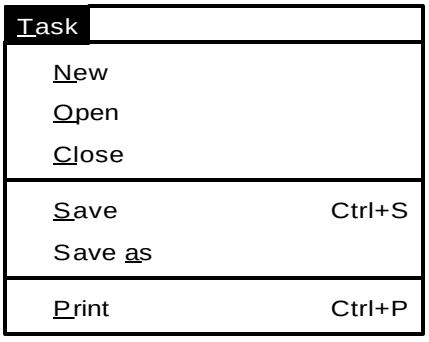



Save each folder that is changed by clicking on the **Save changes** command button. After completing all the changes to the configuration click on the **Exit configuration** command button.

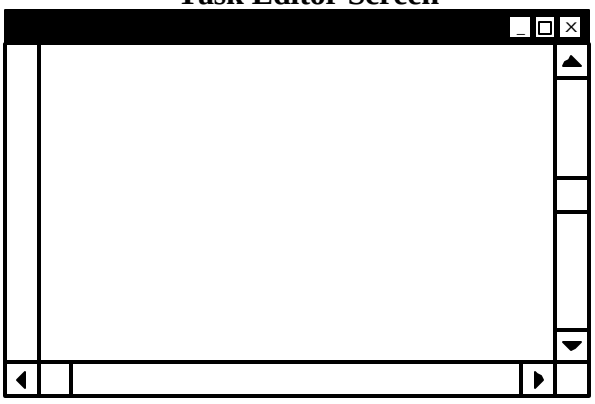
## 6.3.5 – The Task Editor

The Project program is created and edited using the Task Editor. To select the project to be edited click on the **Task** menu and either the **New** or **Open** item. The **New** selection allows a new task to be developed. The **Open** selection allows a previously developed task to be edited.

**Task Menu Screen**



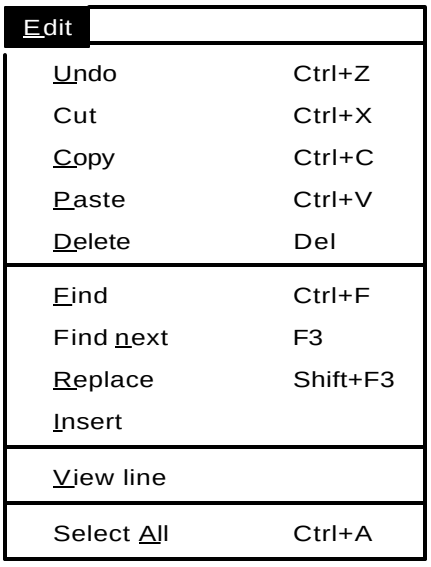

**Task Editor Screen**



Clicking on the Edit menu and then clicking on the desired item can access the Edit functions. The

Items and Actions for the **Edit** menu are listed below.

## Edit Menu




On the menu above:

**Undo (Ctrl+Z)** undoes the latest deletion

**Cut (Ctrl+X)** cuts the selected text and place it on the clip board.

**Copy (Ctrl+C)** copies the selected text and place it on the clip board.

**Paste (Ctrl+V)** pastes the contents of the clip board into the file.

**Delete (Del)** deletes the selected text.

**Find (Ctrl+F)** finds the occurrence of the selected text in the file.

**Find next (F3)** finds the next occurrence of the selected text in the file.

**Replace (shift+F3)** replaces one set of text with another set of text .

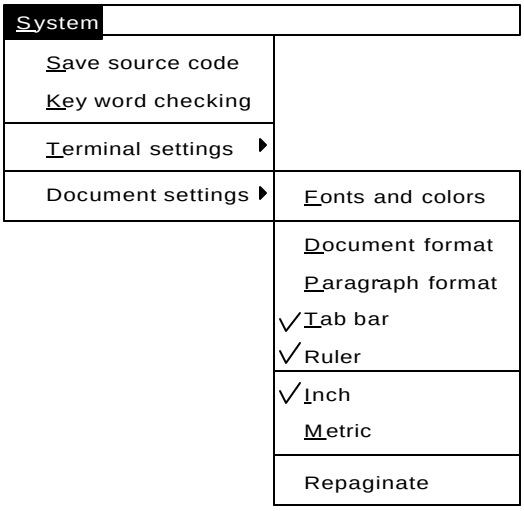
**Insert** Insert a selected file at the current position.

**View line** go to the selected line number.

**Select all (Ctrl+A)** selects all text.

## 6.3.5.1 – Document settings

Clicking on the **System Menu** and then selecting the **Document setting** item can modify the document settings for the task editor.




**Save source code** allows the users to save the project in the MX2000 controller. When checked the user project text will be compressed and sent to the controller during a download project sequence. This text can now be retrieved from the controller.

**Keyword checking** enables or disables Keyword checking. If Enabled it Capitalizes keywords such as program commands and uses the selected colors for keywords and comments.

**Fonts and colors** selects the Font name, Font Style, Font size, background color, foreground color, Keyword color and Comment color. Some of these functions can be duplicated on the Editor Tool Box.

**Document format** selects the document width, height, margins and Tab spacing. Some of these settings can be duplicated on the Editor Tool Box.

**Paragraph format** selects the document margins, Alignment, line spacing, Tabulator type and Tab spacing. These settings can be duplicated on the Editor Tool Box.

**Tab Bar** displays the Tab bar when checked.

**Ruler** displays the ruler when checked.

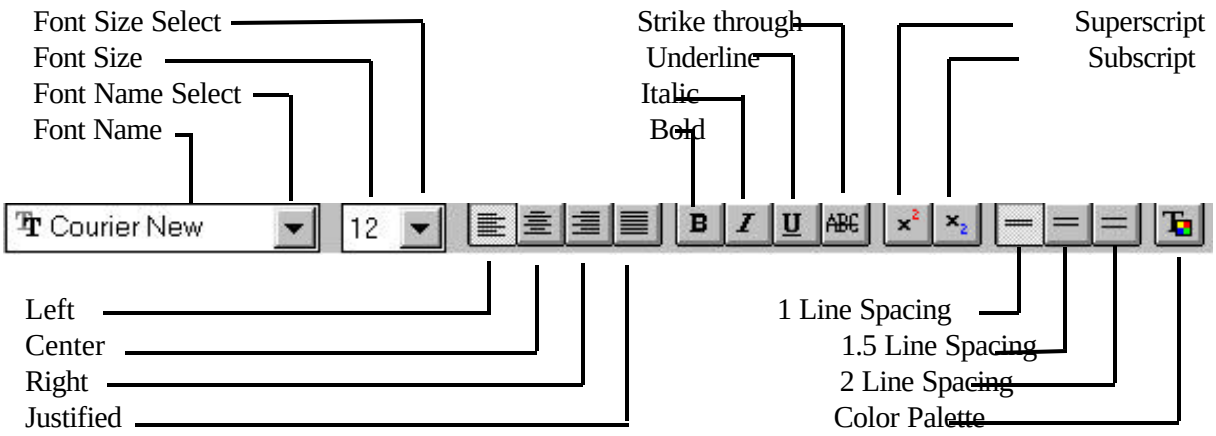
**Inch** selects the inch ruler when checked.

**Metric** selects the metric ruler when checked.

**Repaginate** repaginates the current task.

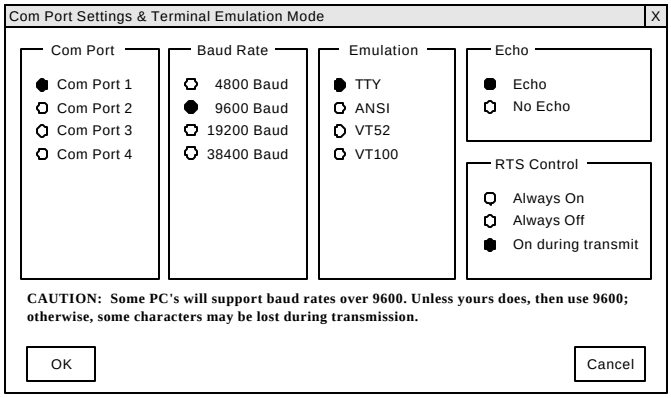
## 6.3.5.2 - Editor Tool Box

The Editor Tool Box can be used to modify the text on the Editor Screen. The Font, Type, line spacing and text color can be modified using the Editor Tool Box.



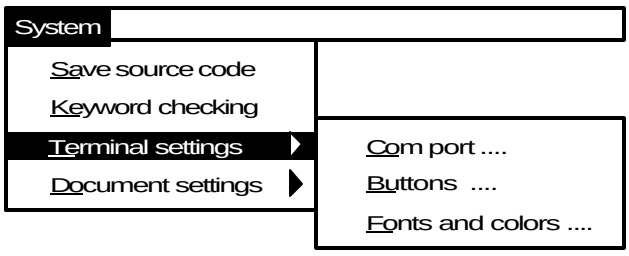

## 6.3.6 – Terminal Emulation

Before entering the Terminal Emulation environment, set up the communication port parameters by clicking on the **System** menu, **Terminal settings** item and then the **Com port** item. Choose the appropriate Com port, baud rate, terminal emulation, echo mode and RTS control from the Com Port Screen by clicking on one of the circles in each section.

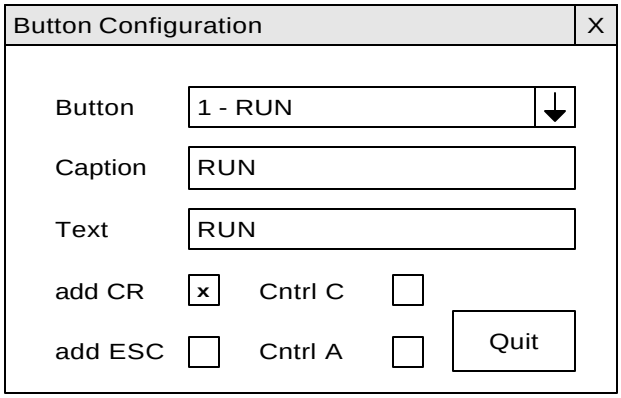



## 6.3.6.1 – Configuring Buttons

To program the buttons on the Terminal Emulation screen, Click on the **System** Menu and then on the **Terminal settings** item.




1) Click on the **Buttons** item.

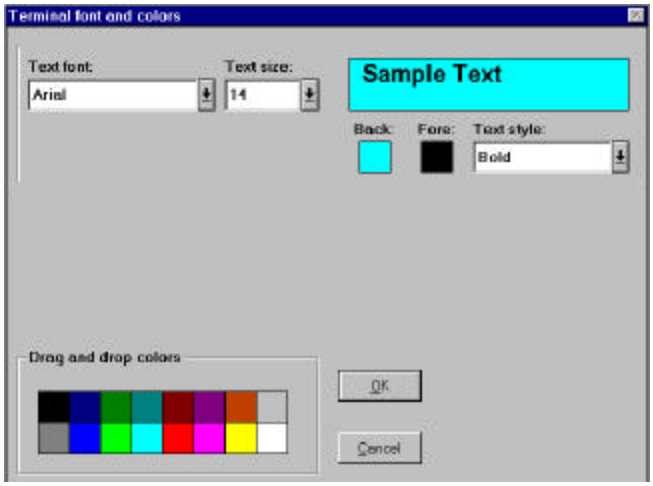



2) Click on drop list arrow and select button number.
3) Click on **Caption** text box and enter the button caption text.
4) Click on **Text box** and enter the command line text, command sent.
5) If motion and program execution is to be stopped after the button's command is executed, click on the **Cntrl C** or **Cntrl A** check box. See the Host Command section of this manual for a more detailed description of **Cntrl A** and **Cntrl C**.
6) If command is to be allowed during program execution click on **Add ESC** check box.
7) Click on **Add CR** check box if not a Cntrl C or Cntrl A command.

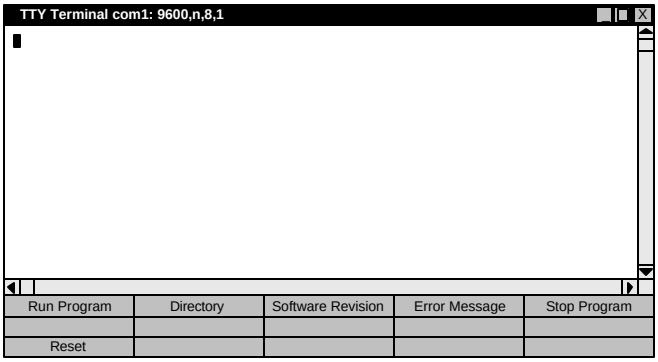
## 6.3.6.2 – Configuration Fonts & colors

To select the Font and Colors for the Terminal Emulation screen Click on the **System** Menu and then on the **Termi-**

**nal settings** item. Click on the **Fonts and colors** item. Select the desired Font, Style, Font size, Background color and Foreground color for the Terminal Emulation environment. When finished, click on the O.K. button.




To enter the Terminal Emulation environment click on the **Terminal** command button.




## 6.3.7 Configuration & Setup Folders

Clicking on the **Configuration** command button accesses the folders for the configuration & setup screens. These folders allow project setup conditions to be programmed. Clicking on its folder tab can access a specific folder.

**Note: Clicking on the Save changes command button saves the current folder data.**

**Clicking on the Exit Configuration command button can be used on any folder to exit the Configuration setup. If any of the items in the folder have been changed, a query will occur which will give the user the option of saving the folder data.**

**Clicking on another folder tab will allow changes to the newly selected folder. The changes, which have already been made, will not be affected. This allows you to click between folders, set up the necessary parameters, and save only once before exiting the configuration screen. A description of each folder follows.**

## 6.3.7.1 – Controller Type Folder

This folder allows the controller type to be defined for the user program. The choices are: MX 2 axis servo/stepper control, MX 4 axis servo/stepper control, MX 6 axis servo/stepper control, MX 8 axis servo/stepper control, DCS 2 axis servo/stepper control, MX1C 2 axis servo/stepper control, TDC 1 axis servo control, and SS2000D6I 1 axis stepper control.

## 6.3.7.2 – System Folder

This folder defines the axis assignments for a task, Drive type, motor direction for a + motion and the units per motor revolution.

System

| | Task assignment | Drive Type | Motor Direction | Units per motor resolution |
|---|---|---|---|---|
| Axis 1 | c:\mcpi\name.ts | open loop stepper | + = cw motor direct | 1.0 |
| Axis 2 | c:\mcpi\name.tsk | open loop stepper | + = cw motor direction | 1.0 |

The **Task assignment** allows an axis to be assigned to a project task.

The **Drive Type** defines the type of drive operation. The choices are: **open loop stepper**, **closed loop stepper** and **servo.** Open loop steppers do not have encoders.

The **Motor Direction** sets the motor direction for a + move. The choices are: **+= cw motor direction** or **+= ccw motor direction**. The motor direction is as viewed from the rear of the motor.

The desired **Units per motor revolution** value should be entered. A unit is the method of measurement to be used, i.e inches, mm, degrees, etc. This sets the number of user units for one motor revolution. Move distances and position values are in units, Speeds are in units/second and Acceleration and Deceleration values are in units/second$^2$.

**Example:**
If a motor is directly coupled to a lead screw, which has a 0.8" pitch, the units per motor revolution should be set to 0.8. The user may now write his program with distances in inches.

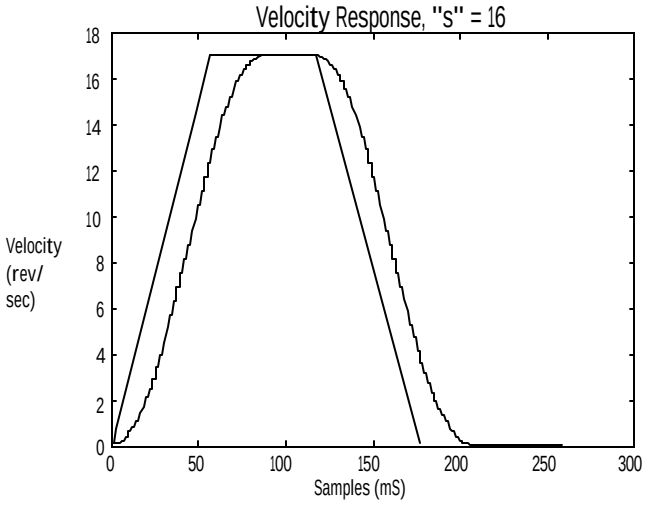
## 6.3.7.3 – Profile Folder

This folder selects the motion profile, maximum acceleration rate, maximum speed and Delay after motion. The Speed, Acceleration and Deceleration item are program execution default values that can be altered with basic commands during program execution.

Profile

| | Motion profile | Speed (units/sec) | Acceleration (units/sec²) | Deceleration (units/sec²) | Max. accel (units/sec²) | Max. speed (units/sec) | Delay after motion(sec) |
|---|---|---|---|---|---|---|---|
| Axis 1 | trapezoidal | 20.0 | 100.0 | 100.0 | 200.0 | 124.0 | 0.05 |
| Axis 2 | trapezoidal | 20.0 | 100.0 | 100.0 | 200.0 | 124.0 | 0.05 |

**Motion Profile** determines how the motor's speed changes. Speed changes require a period of acceleration/deceleration to increase/decrease the motor's speed. The "Motion Profile" determines how this rate is applied. There are 32 choices, and a profile setting of 1 results in a "Trapezoidal" profile, this profile yields the minimum move time. Settings 2 - 32 yields "S-curve" profiles with varying degrees of smoothing. The higher the profile setting, the more "S" like the profile becomes. Move times with profile settings 2 - 32 are from 2 to 62 ms longer respectively than those executed with a setting of 1. The "S-curve" profiles usually results in smoother motion at the expense of longer move times. Move times can be shortened, however, by raising the acceleration, deceleration, and/or speed of the move.




**Speed** sets the non-coordinated speed of an axis in units/sec.

**Acceleration** sets the acceleration rate of an axis in units/sec$^2$.

**Deceleration** sets the deceleration rate of an axis in units/sec$^2$.

**Max. accel** sets the maximum allowed acceleration or deceleration rate in units/sec$^2$. This value is also used to decelerate motion to a stop when a fault such as a travel limit occurs.

**Max. Speed** sets the maximum allowed target speed in units/second. Speed, Acceleration and Deceleration values can be reset within a program as long as the value used is less than or equal to the max speed and max accel respectively.

**Delay after motion** sets the minimum time, in seconds, between two moves.

## 6.3.7.4 – Analog Inputs Folder

This folder defines the analog input configuration and its filter time constant for an axis.

Analog inputs

| | Input type | Filter 1 time constant (sec) | Filter 2 time constant (sec) |
|---|---|---|---|
| Axis 1 | differential | 0.005 | 0.005 |
| Axis 2 | differential | 0.005 | 0.005 |

**Input type** defines the analog input configuration of an axis. The choices are differential or single ended.

**Filter 1 time constant** defines the filter time constant for analog input 1 of the axis.

**Filter 2 time constant** defines the filter time constant for analog input 2 of the axis.

## 6.3.7.5 – Encoder Folder

This folder allows the Encoder for a Servo drive or closed loop stepper drive. This folder is described in further detail in the Servo Drive and Stepper Drive sections of the manual.

## 6.3.7.6 – Open Loop Stepper Folder

This folder sets up the parameters used by an open loop stepper and is described in more detail in the Stepper Drive section of this manual.

## 6.3.7.7 – Closed Loop Stepper Folder

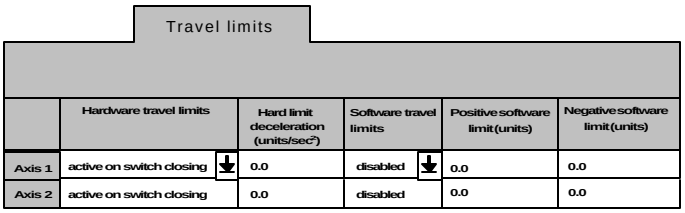
This folder sets up the parameters used by a closed loop stepper and is described in more detail in the Stepper Drive section of this manual.

## 6.3.7.8 – Servo Drive Folder

This folder sets up the parameters used by a servo drive and is described in more detail in the Servo Drive section of this manual.

## 6.3.7.9 - Travel Limit Folder

The hardware limits and software limits are controlled from this folder.




**Hardware travel limits** choices are disabled, active on switch closing and active on switch opening. Hard limit inputs are used to stop the motor before it runs into a physical end of travel, thus avoiding damage to the mechanical system.

**Activating the +limit input stops the motor if it is rotating in the + direction. Activating the –limit input stops the motor if it is rotating in the – direction. A fault condition is a result of the hardware travel limit activation. See the ERR command in Section 7 to create an error handling routine.**

**Hard limit deceleration**, if non-zero, specifies the axis deceleration if a hard limit is activated. If the value is zero than the Maximum Acceleration value will be used to stop the motor.

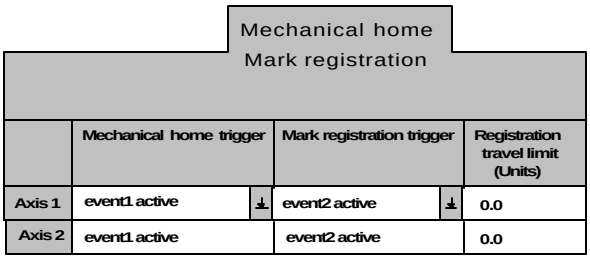
**Software travel limits** can be enabled or disabled.

**Positive software limit** specifies the programmable position limit for a positive motion. An error is generated when this position is exceeded.

**Negative software limit** specifies the programmable position limit for a negative motion. An error is generated when this position is exceeded.

## 6.3.7.10 – Mechanical Home & Mark Registration Folder

This folder specifies the trigger for the mechanical home (MOVEHOME), mark registration cycle (MOVEREG) and specifies the maximum distance allowed for a mark registration cycle.



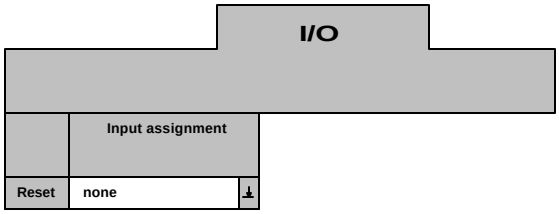

**Mechanical Home trigger** & **Mark Registration trigger** specifies the trigger for the cycle. There are two trigger inputs **EVENT1 and EVENT2** that can be used as a trigger.

The trigger combination for mechanical home and Mark registration are: event 1 active**,** event 1 inactive**,** event 1 active & encoder marker**,** event 1 inactive & encoder marker**,** encoder marker active**,** encoder marker inactive**,** event 2 active and event 2 inactive**.**

**Registration travel limit** specifies the maximum distance, in units, allowed for a mark registration cycle. If the value is zero a travel limit is not limited.

## 6.3.7.11 – I/O Folder

This folder allows an external input to generate a controller system reset.




**Input assignment** allows an external input to generate a controller system reset. The choices are none, Expansion board 1 – input 1 and Digital board 1 – input 1.

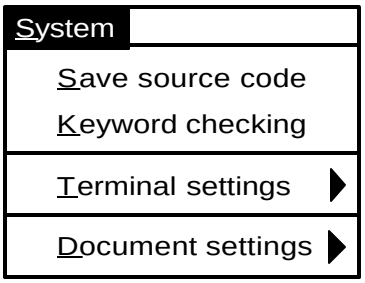
## 6.3.8 – Preparing User Project for Execution

In order to execute a project program it must first be Compiled and then Downloaded to the controller. The project source code can be recovered from the controller as well if the save source option is utilized.

## 6.3.8.1 – Project Source code

The Project Source Code is the English version of the user=s program. If the user=s program needs to be uploaded from the controller at any time, A Save Source Code @ must be enabled. The Source code of a project can be saved in the controller. However, the source code uses up program memory in the controller. The selection for source code saving is accessed by clicking on the **System** menu. Clicking on the Save source code item can toggle the Save source code setting. A check mark will appear when the source code is to be saved.

**Note: Saving the source code in the controller requires a lot of program memory. If the user=s program is extremely long it may not be possible to save the source code. See the FREEMEM command for more information.**



## 6.3.8.2 – Compiling a Project

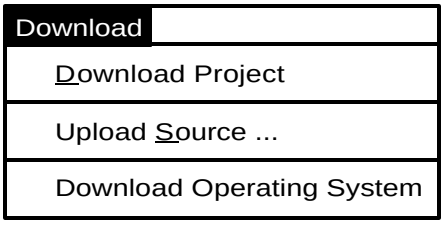
Whether the project is new, or changes have been made to the task or configuration, it MUST be compiled BEFORE DOWNLOADING for it to be stored and implemented in the controller. Compiling converts the users task and configuration to machine code that the controller can understand. A project can be compiled by clicking on the **Compile** Command button or on the **Compile** menu and then the **Compile project** item.

## 6.3.8.3 – Downloading a Project

A project can be downloaded with or without its source code by clicking on the **Download** command button or clicking on the **Download** menu and then the **Download project** item.

## 6.3.8.4 – Uploading Source Code

The projects source code can be uploaded from the controller to the PC by selecting the **Upload Source** item from the **Download menu**. A project can be uploaded from the controller ONLY if it had previously been saved in the controller. See section 6.3.8.1.

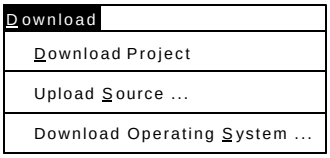


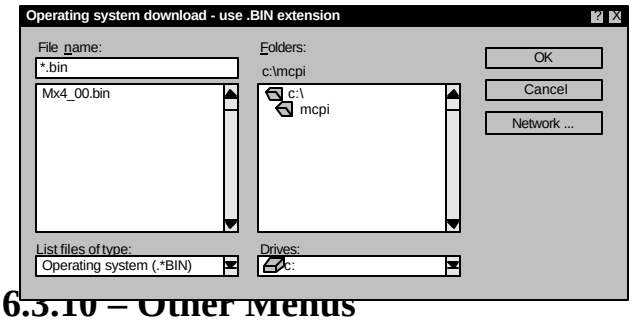

## 6.3.9 – Downloading an Operating System

Although the unit comes with an operating system installed. Clicking on the **Download menu** and then the **Download Operating System** item will download new operating system software.

The operating system file, with an extension .**bin**, can now be selected by clicking on the desired file name. To start the operating system download procedure click on the **OK** command button.

**Note**: The file names for the different controllers start with the following letters: **mx** for the MX2000 controller, **dcs** for the DCS controller , **tdc** for the TDC controller and d**xI** for the SS2000D6i controller.



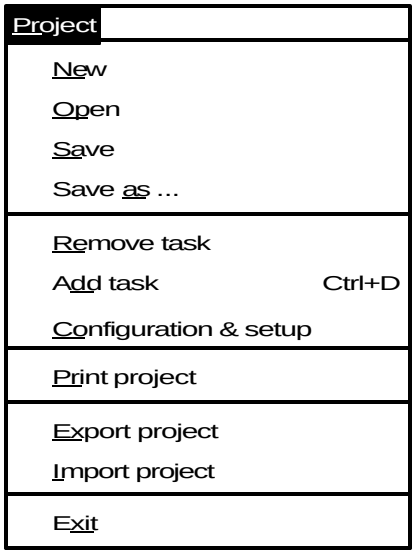




## 6.3.10 – Other Menus

The MCPI menus are pull down menus. Clicking on a menu shows an itemized list of operations allowed for that menu. The menus are Project, Task, Edit, Compile, Download, Utility, System, Window and Help.

## 6.3.10.1 – Project Menu

This menu allows you to create a new project, open an existing project, save a current project, add or remove a task from a project, open the configuration & setup environment, print the current project, or exit the MCPI programming environment.




**New** is used to create a new project.

**Open** is used to open up an existing Project.

**Save** is used to save the current project.

**Save as** is used to save the current project under a new name.

**Remove task** is used to remove a task file from an open project.
**Add task** is used to add a file to a current project. Up to seven tasks may be added to one project.

**Configuration & setup** is used to edit the Configuration & setup folders.

**Print project** is used to print a current project's information.

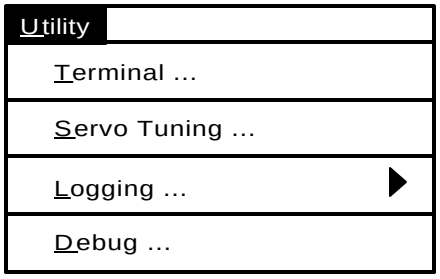
**Export project** is used to export a current project to another drive or directory.

**Import project** is used to import a selected project from another drive or directory into the MCPI Environment.

**Exit** is used to exit the MCPI programming Environment.

## 6.3.10.2 – Utility Menu

This menu allows reselection of terminal mode emulation, data logging, servo tuning, or program debugging.




**Terminal** starts terminal emulation mode. This allows direct communication with the controller.

**Servo Tuning** allows the tuning of a servo system.

**Logging** allows data logging of specific parameters by the controller.

**Debug** starts program task debugging.

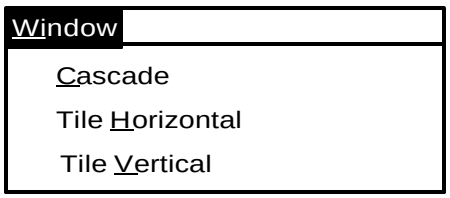
## 6.3.10.3 – Window Menu

This menu selects the windows format for the open windows.

**Cascade** cascades the open windows.

**Tile Horizontal** tiles the open windows Horizontally.

**Tile Vertical** tiles the open windows Vertically.




## 6.3.10.4 – Help Menu

This menu provides help on program commands, technical assistance and displays the MCPI software version.

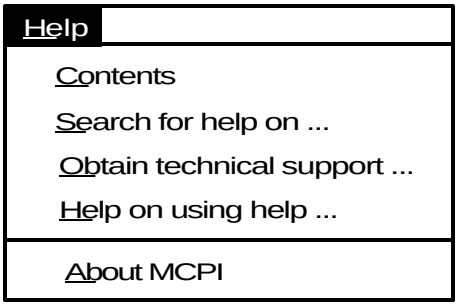
**Contents** list the help topics.

**Search for help on** lists the help items and descriptions.

**Obtaining technical support** provides application assistance telephone numbers.

**Help on using help** provides help on how to use Help.

**About MCPI** provides the MCPI version number.




## 6.3.11 – Project Command Buttons

The MCPI command buttons allow the selection of the Configuration & Setup folders, compilation of a current project, downloading of a current user project, selecting the Terminal Emulation environment, selecting the servo tuning environment, or selecting the program debugger environment.

**Configuration** enters the configuration & setup environment.

**Compile project** compiles the current project.

**Download project** downloads the current project.

**Terminal** enters the Terminal emulation environment.

**Servo tuning** allows the tuning of a servo motor.

**Debug** enters the Program Debugger Environment.

# Section 7.0
# Software Reference Guide

## 7.1 SEBASIC Conventions

A basic-like language conforms to most of the rules and conventions of modern implementations of the BASIC programming Language, such as "QuickBasic", etc. Following is a summary of the considerations to be used in writing your programs.

### 7.1.1 Arithmetic Operators

The SEBASIC arithmetic operators are listed in order of precedence.

| Operator | Function |
|---|---|
| - | Negation. |
| * , / | Multiplication and division. |
| + , - | Addition and subtraction. |

Parentheses changes the order in which arithmetic operations are performed. Operations within parentheses are performed first. Inside parentheses, the usual order of operation is maintained.

**Note: Squaring and exponentiation are not supported; use multiplication to perform these operations.**

Example: to calculate $X^3$, use X*X*X.

### 7.1.2 Logical Operators

Logical operators perform test on multiple relations, bit manipulations or Boolean operations and return a "true" (one) or "false" (zero) value used in making a decision.

These operators are used in Boolean expressions. The logical operators in SEBASIC, listed in order of precedence, are as follows:

| Operator | Use |
|---|---|
| NOT | NOT <term> a false term, results in the Boolean expression being true. |
| AND | <term AND <term> if both terms are true, results in the Boolean expression being true. |
| OR | <term> OR <term> if either term is true, results in the Boolean expression being true. |

### 7.1.3 Relationship Operators

Relationship operators are used to compare two values. The result of the comparison is either "true" (one) or "false" (zero). This result can then be used to make a decision regarding program flow.

| Operator | Relation | Expression |
|---|---|---|
| = | Equality * | X = Y |
| <> | Inequality | X <> Y |
| < | Less than | X < Y |
| > | Greater than | X > Y |
| <= | Less than or equal to | X <= Y |
| >= | Greater than or equal to | X >= Y |

* The equal sign (=) is also used to assign a value to a variable.

### 7.1.4 Basic Data Types

Two basic data types exist: floating point values (REAL) and string values. All values are assumed to be floating point unless a $ suffix is used.

| | |
|---|---|
| x | x is a floating point variable. |
| x$ | x$ is a string variable. |

**Note: All variable names and program labels must begin with a letter A-Z.**

### 7.1.5 Case Sensitivity in Statements & Commands

Some programming statements and commands are case-sensitive; others are not. The following table defines case sensitivity in SEBASIC:

| Basic Language Element | Case Sensitive? | Max. Length (characters) |
|---|---|---|
| Label | No | 80 |
| Variable name | No | 80 |
| String constant | Yes | 80 |
| Basic Keyword | No | N/A |

The Host commands are not case sensitive; that is, upper and lower case letters can be used interchangeably.

## 7.1.6 Program Limits

When writing an MX program please observe the following maximum values; otherwise the project will not compile.

| Item | Maximum Allowed |
|---|---|
| Line labels | 100 total per task |
| Local Variables | 100 total per task |
| Common Variables | 100 total per project |
| Literal' s | 100 total per project |
| DATA Elements | 100 total per task |
| Nested FOR loops | 100 total per task |
| Nester DO loops | 100 total per task |

## 7.1.7 Numeric Formats and Range

Numeric data may be represented in standard format or in scientific notion format. The following are illustration of each type of format.

Standard format:

1234567
-1.234567
0.1234567
1234.567
etc

Scientific notation format

| | |
|---|---|
| 2e6 | (2,000,000) |
| 2.0456e4 | (20,456) |
| -3.14159e0 | (-3.14159) |
| 6.78e-2 | (0.0678) |
| etc | |

The largest number that can be used is 3.4e38. The smallest number that can be used is 2.9e-39. The numeric resolution is 1 part in 8,388,608 or 1.2e-7.

## 7.1.8 Program Comments

An apostrophe (') in a program line prevents a line from executing and allows program comments /documentation. All text to the right of the ' to the end of line is not considered part of the command during execution.

**Examples:**
'MOVE=10 The program will not execute this line
MOVE=100 'The program will execute this line

## 7.1.9 Axis Related Command Syntax

The syntax for programming commands has numerous choices. Some are I/O related and some are axis related. The I/O related syntax's are covered in detail for that specific command. However, the axis related commands are so numerous that they will be covered in this section of the manual.

### 7.1.9.1 Definitions Used in Syntax Descriptions

**COMMAND** represents an arbitrary axis command.

**Expression**
One or more constants, variables or commands that return data operated on by mathematical operators or mathematical functions. Expressions can be very simple (as in the case of a single constant or variable) or quite complex (as shown in the last example below). The compiler will indicate an error if an expression is too complex. For all practical purposes it is not possible to write an expression that is too complex.

The following are all valid expressions:

| | |
|---|---|
| 5.2 | (single constant) |
| X | (single variable) |
| SPEED(2) | (command) |
| X+3 | (addition of variable and constant) |
| SIN(X-3) | (sine function of the difference of a variable and constant) |
| (X+3)*SIN(Y)/(Z+SQRT(X)) | (complicated expression) |

**axis**
Specifies the axis on which the command is performed. Axis is specified as an expression that evaluates to the desired axis number. If the expression evaluates to a non-whole number, then the nearest whole number less than the expression value is used. Axis is most commonly specified as a constant.

Example: COMMAND(x+2)
with x = 1.5 is equivalent to COMMAND(3)

**[text]**
Denotes 0 or more occurrences of what is enclosed by the brackets. The brackets are not part of the syntax.

### 7.1.9.2 Syntax Descriptions

COMMAND(axis) = expression
Execute the command, on the specified axis, using the data supplied by the expression.

Syntax example: MOVE(2) = 10
performs an index motion of 10 units on axis 2

COMMAND = expression list
Execute the command, on the 1 or more specified axes, using the data supplied by the expression list.

expression list
1 to N expressions. Commas follow each expression in the list except for the last one. (N = number of axes in the controller: 2,4,6 or 8) The first expression in the list is for axis 1, the next for axis 2, etc. To skip an axis, simply enter a comma for that axis. Although these examples use con-

stants any expression regardless of complexity can be used.
The following are all valid expression lists:

5.3 no comma, since a comma does not follow the last expression in the list (axis1 value is 5.3)

,,,5.2,,6 1st comma - skip axis 1, 2nd comma – skip axis 2, 3rd comma - skip axis 3, 4th comma follows expression for axis 4, 5th comma - skip axis 5 (axis 4 value is 5.2, axis 6 value is 6)

syntax example:

ABSPOS = ,0,0,0
sets ABSPOS of axes 2-4 to 0

**COMMAND(n [,m] ) = expression [, expression]**
Execute the command, on the 1 or more specified axes, using the data supplied by the 1 or more expressions.

**n**
Axis number, which is a whole number constant in the range 1 to N, where N is the number of axes in the controller: 2,4,6 or 8.

**[,m]**
0 or more occurrences of axis number preceded with a comma.

**[,expression]** 0 or more occurrences of an expression preceded with a comma.

For each axis number there is a corresponding expression. The 1st (left most) expression is associated with the lowest axis number, the 2nd with the next lowest, etc.

COMMAND(2,3,7)=1.1,2.2,3.3 does the same as COMMAND=,1.1,2.2,,,,3.3 but is less prone to errors since there is no need to put all those commas in the right place.

Syntax example:

ACCEL(3,4) = 100,200
sets ACCEL for axis 3 to 100 units and axis 4 to 200 units.

**COMMAND(axis)**
Execute the command on the specified axis. This command syntax returns data and is used in an expression.

Syntax example:
X = DECEL(1)
Sets X equal to the DECEL value of axis 1.

## COMMAND(n [,m] )

Execute the command, on the 1 or more specified axes.

### n

axis number, which is a whole number constant in the range 1 to N, where N is the number of axes in the controller: 2,4,6 or 8.

**[,m]**
0 or more occurrences of axis number preceded with a comma.

Syntax examples:

WAITDONE(2,3)
waits for axis 2 and 3 to finish motion
STOP(1,3)
stops motion on axes 1 & 3

## 7.2 Programming Commands Grouped By Functions

| **Bitwise Operator** | | **Page** |
|---|---|---|
| & | Returns the bitwise AND of the expression. | 78 |
| \| | Returns the bitwise inclusive OR of the expression. | 78 |
| ^ | Returns the bitwise exclusive OR of the expression. | 78 |
| >> | Returns the bitwise shift right of the argument. | 79 |
| << | Returns the bitwise shift left of the argument. | 79 |

| **Boolean Operator** | | |
|---|---|---|
| AND | The logical AND operator is used in Boolean expressions. | 83 |
| NOT | The logical NOT operator is used in Boolean expressions. | 137 |
| OR | The logical OR operator is used in Boolean expressions. | 140 |

| **Following Parameter** | | |
|---|---|---|
| FOLACCDIST | Specifies the master distance traveled for the follower to catch the master velocity after follower motion begins. | 107 |
| FOLDCCDIST | Specifies the master distance traveled for the follower to attain a velocity of zero from the current velocity. | 107 |
| FOLINPUT | This command specifies the follower axes & the master velocity source. | 108 |
| FOLJOG | Requests a Following axis Jog cycle. | 108 |
| FOLMAXRATIO | Sets or returns the maximum allowable following axis speed during an offset advance cycle. | 109 |
| FOLMINRATIO | Sets or returns the minimum allowable following axis speed during an offset recede cycle. | 109 |
| FOLMOVE | Request a Following axis move. | 109 |
| FOLMOVEREG | Request a following axis move registration cycle. | 110 |
| FOLOFFSET | Defines a following incremental offset distance from the current position. | 110 |
| FOLOFFSETDIST | Sets or returns the master distance traveled for a FOLOFFSET command. | 110 |
| FOLRATIO | Sets or returns the ratio of the following axis to the master. | 111 |
| FOLRATIOINC | Specifies the acceleration rate for a FOLRATIO change during motion. | 111 |
| FOLSTARTDIST | Specifies a master distance which is used as a delay distance for starting following motion. | 111 |
| FOLSYNC | Returns the following sync status of a follower axis. | 112 |
| FOLSYNCDIST | Specifies a master distance for the follower to travel in synchronization with the master when a FOLOFFSET command is issued. | 112 |
| FOLTRIG | Defines the follower starting trigger for motion. | 112 |

| **I/O Function** | | |
|---|---|---|
| ANALOG | Sets or returns a numeric value representation on the analog port. | 82 |
| BCD | Returns the BCD switches value connected to an Expansion I/O port. | 86 |
| EXIN | Returns the state of the specified expansion I/O inputs. | 105 |
| EXOUT | Sets or returns the state of the specified expansion I/O outputs. | 106 |
| GETCHAR | Waits for a character on the selected serial port and returns the ASCII code of the character. | 115 |
| IN | Returns the state's of the specified digital I/O inputs. | 121 |
| INCHAR | Returns the ASCII code of a character from the designated serial port. | 121 |
| INPUT | Reads a line of data from the designated serial port. | 122 |

| | | |
|---|---|---|
| OUT | Sets or returns the condition of a specified digital output. | 141 |

## 7.3 Programming Command Summary (alphabetical list)

## 7.4 Alphabetical List of Programming Commands with Syntax and Examples

# & Bitwise Operator

**ACTION:** Returns the bitwise AND of the expressions

**PROGRAM SYNTAX:** result=expression1 & expression2

**REMARK:** A 24 bit binary AND is performed on the two arguments.

**EXAMPLES:**
X=10 & 2
0000 0000 0000 0000 0000 1010 (10)
0000 0000 0000 0000 0000 0010 (2)
0000 0000 0000 0000 0000 0010 (result= 2)
returns a 2 to X.

# | Bitwise Operator

**ACTION:** Returns the bitwise inclusive OR of the expressions.

**PROGRAM SYNTAX:** result=expression1 | expression2

**REMARK:** A 24 bit binary OR is performed on the two arguments.

**EXAMPLES:**
X=10 | 4
0000 0000 0000 0000 0000 1010 (10)
0000 0000 0000 0000 0000 0100 (4)
0000 0000 0000 0000 0000 1110 (result= 14)
returns a 14 to X.

# ^ Bitwise Operator

**ACTION:** Returns the bitwise eXclusive OR of the expression

**PROGRAM SYNTAX:** result=expression1 ^ expression2

**REMARK:** A 24 bit binary eXclusive OR is performed on the two arguments.

If a binary bit in expression2 is a 1 the resulting bit will be inverted in expression1.

**EXAMPLES:**
X= 10 ^ 6
0000 0000 0000 0000 0000 1010 (10)
0000 0000 0000 0000 0000 0110 (6)
0000 0000 0000 0000 0000 1100 (result= 12)
returns a 12 to X.

## >> Bitwise Operator

**ACTION:** Returns the bitwise shift right of the argument.

**PROGRAM SYNTAX:** result= expression >> expression1

**REMARK:** expression is shifted right by the value in expression1 and the resulting value is returned.

A 24 bit binary shift right is performed on the argument. 0's are shifted in starting at the MSB bit.

**EXAMPLES:**

```
X=10
 X=X>>1
 0000 0000 0000 0000 0000 1010     (10)
 0000 0000 0000 0000 0000 0101    (result= 5)
 returns a 5 to X.
```

## << Bitwise Operator

**ACTION:** Returns the bitwise shift left of the argument.

**PROGRAM SYNTAX:** result= expression << expression1

**REMARK:** expression is shifted left by the value in expression1 and the resulting value is returned.

A 24 bit binary shift left is performed on the argument. 0's are shifted in starting at the LSB bit.

**EXAMPLES:**

```
X=10
 X=X<<1
 0000 0000 0000 0000 0000 1010     (10)
 0000 0000 0000 0000 0001 0100    (result= 20)
 returns a 20 to X.
```

## ABS Mathematics Function

**ACTION:** Returns the absolute value of an expression.

**PROGRAM SYNTAX:** ABS(expression) - used in an expression

**REMARK:** The absolute value is the unsigned magnitude of the expression.

**EXAMPLES:**

```
X = -57.5
 A=ABS(X)              ' returns a 57.5 to A
```

# ABSPOS Trajectory Parameter

**ACTION:** Sets or returns the commanded absolute position of an axis.

**PROGRAM SYNTAX:**
ABSPOS(axis)=expression
ABSPOS=expression1, ... , expression8
ABSPOS(axis, ... , axis)=expression, ... , expression
ABSPOS(axis) - used in an expression

**REMARK:** The axis specifies the number of the axis (1-8).

The expression specifies the absolute position in units.

ABSPOS represents the commanded motor position, **and can only be set while no motion is occurring**. Setting ABSPOS during motion, causes the program to be trapped at the ABSPOS instruction until the motion is completed. Setting ABSPOS also sets ENCPOS (encoder position) to the same value. ABSPOS and ENCPOS are initialized to 0 at power up or system reset. ABSPOS is set equal to ENCPOS when the servo drive is enabled by the WNDGS command. ABSPOS is also set at the end of a MOVEHOME command. Reading ABSPOS returns the actual commanded position in user units.

**WARNING!** Care should be taken when setting ABSPOS during program execution on a servo axis. Because the encoder position, ENCPOS, is set to ABSPOS when the command is executed, any position error at that time will be lost. If done repeatedly this may cause an inadvertent accumulative error over time. To prevent this error, only set ABSPOS once in the program, and make sure it is not included in a loop of statements. Incremental moves will increment ABSPOS, and there is no need to be concerned about rollover of the position counter, since the counter will wraparound and continue operating without loss of position.

**EXAMPLES:**
ABSPOS(3)=2
sets the absolute position of axis 3 to 2 units.

ABSPOS=1,,3
sets the absolute position of axis 1 to 1 unit, axis 2 no change and axis 3 to 3 units.

ABSPOS(1,3)=1,3
sets the absolute position of axis 1 to 1 unit and axis 3 to 3 units.

# ACCEL — Trajectory Parameter

**ACTION:** Sets or returns the acceleration value of the motor.

**PROGRAM SYNTAX:**
ACCEL(axis)=expression
ACCEL=expression1, ... , expression8
ACCEL(axis, ... , axis)=expression, ... , expression
ACCEL(axis) - used in an expression

**REMARK:** The axis specifies the number of the axis (1-8).

The expression is the acceleration rate in units/sec$^2$.

The rate at which the motor speed is increased. Specifying a value greater than "Max Accel" set in the system *Configuration and Setup* will result in ACCEL being set to the "Max Accel". ACCEL can be set during motion, but the new setting will not be used until the next motion. Reading ACCEL returns the most recent setting.

**EXAMPLES:**
ACCEL(3)=200
sets the acceleration of axis 3 to 200 units/sec$^2$.

ACCEL=100,,200
sets the acceleration rate of axis 1 to 100 units/sec$^2$ and axis 3 to 200 units/sec$^2$.

ACCEL(1,3)=100,200
sets the acceleration rate of axis 1 to 100 units/sec$^2$ and axis 3 to 200 units/sec$^2$.

# ACTSPD — Trajectory Parameter

**ACTION:** Returns the current commanded velocity of an axis in Units/second.

**PROGRAM SYNTAX:** ACTSPD(axis) - used in an expression

**REMARK:** This command can be used in conjunction with a FOLINPUT command to specify the master source. It can also be used to monitor the current commanded velocity of an axis.

**EXAMPLES:**
FOLINPUT(1,3)=ACTSPD(2)
Sets the current commanded velocity of axis 2 as the master source. Axis 1 and axis 3 are follower axes.

axspd=ACTSPD(2)
Sets variable axspd to the current commanded velocity of axis 2.

# ANALOG I/O Function

**ACTION:** Sets or returns a numeric value representation of the voltage on the selected analog port.

**PROGRAM SYNTAX:** ANALOG(b0n) - used in an expression
ANALOG(b0n)=expression

**REMARK:** The b specifies the axis board number (1-4).

The n specifies the analog input (1-4) or output(1-2).

ANALOG(b0n)
Returns the present value of the specified analog input. The range is +10.0 volts to -10.0 volts.

ANALOG(b0n)=expression
Sets the analog output voltage equal to the expression. The range is +10.0 volts to -10.0 volts.

Board Analog Configuration

| b0n value | A input differential B input differential | A input single ended B input differential | A input differential B input single ended | A input single ended B input single ended |
|---|---|---|---|---|
| 101 | board 1 A+ & A- | board 1 A+ | board 1 A+ & A- | board 1 A+ |
| 102 | board 1 B+ & B- | board 1 B+ & B- | board 1 B+ | board 1 B+ |
| 103 | | board 1 A- | | board 1 A- |
| 104 | | | board 1 B- | board 1 B- |
| 201 | board 2 A+ & A- | board 2 A+ | board 2 A+ & A- | board 2 A+ |
| 202 | board 2 B+ & B- | board 2 B+ & B- | board 2 B+ | board 2 B+ |
| 203 | | board 2 A- | | board 2 A- |
| 204 | | | board 2 B- | board 2 B- |
| 301 | board 3 A+ & A- | board 3 A+ | board 3 A+ & A- | board 3 A+ |
| 302 | board 3 B+ & B- | board 3 B+ & B- | board 3 B+ | board 3 B+ |
| 303 | | board 3 A- | | board 3 A- |
| 304 | | | board 3 B- | board 3 B- |
| 401 | board 4 A+ & A- | board 4 A+ | board 4 A+ & A- | board 4 A+ |
| 402 | board 4 B+ & B- | board 4 B+ & B- | board 4 B+ | board 4 B+ |
| 403 | | board 4 A- | | board 4 A- |
| 404 | | | board 4 B- | board 4 B- |

**EXAMPLES:** X=ANALOG(102)
Returns the current voltage on axis board 1 input 2.

ANALOG(102)=2.5
Sets the voltage on axis board 1 output 2 to +2.5 volts.

# AND — Boolean Operator

**ACTION:** The logical AND operator is used in Boolean expressions.

**PROGRAM SYNTAX:** expression1 AND expression 2

**REMARK:** The AND operator uses this "truth table".

| expression1 | expression2 | Condition Result |
|---|---|---|
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

The result is true if both expressions are true.

**EXAMPLE:** IF (x>2) AND (y<3) THEN GOTO INDEX
The controller checks to see if x>2 and y<3. If both conditions are true the program goes to a label called INDEX.

# ARC Motion Parameter

**ACTION:** Initiates a coordinated motion to move in an arc.

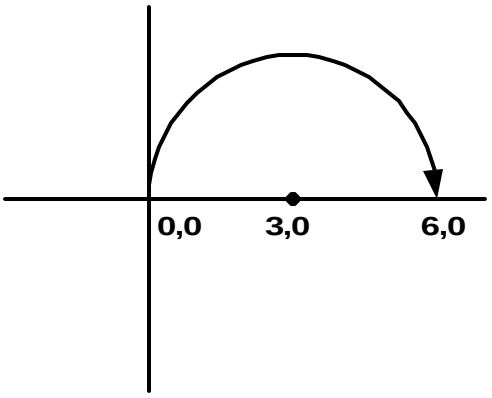
**PROGRAM SYNTAX:**
ARC=x, y, xcenter, ycenter, ±angle (normal)
ARC=xcenter, ycenter, ±angle (in a path)

**REMARK:** The x specifies the axis number for one of the coordinated axes, and the y specifies the axis number for the other axis. The lower numbered axis is considered the master and its parameters SPEED, ACCEL, DECEL and PROFILE are used. The SPEED of the master axis can be used to control the speed of the ARC during motion.
**Note: x and y are not required in a path, since the PATH command defines the axes used.**

The xcenter specifies the x axis coordinate of the arc center, and the ycenter specifies the y axis coordinate of the arc center.

The angle specifies the direction of rotation as well as the arc angle to be executed. The angle is specified in degrees and Clockwise rotation is indicated by a positive sign.

**EXAMPLE:**
ARC=1,2,3,0,+180
'Initiates a 180° clockwise arc rotation, using axis 1 and 2, with a 3 unit radius.




```
PATH=1,2
        ARC=3,0,+180
        statements
PATH END
```
'Initiates a 180° clockwise arc rotation, using axis 1 and 2, with a 3 unit radius when the PATH END command is executed.

In both examples the radius is 3 if:
The controller is in Incremental positioning mode or in Absolute positioning mode and the present position is 0,0.

# ASC — String Manipulation

**ACTION:** Returns the ASCII code for the first character in a string.

**PROGRAM SYNTAX:** ASC(n$)

**REMARK:** The ASCII code returned is for the first character in the string variable n$. If the string is a null then a 0 will be returned.

**EXAMPLE:**

```
a$="part#"
X=ASC(a$)          ' sets x=112 'p'
```

# ATN — Mathematics Function

**ACTION:** Returns the angle (in radians) whose tangent is x.

**PROGRAM SYNTAX:** ATN(x) - used in an expression

**REMARK:** The arctangent returns an angle in the range -π/2 to π/2 radians. π/2 radians equals 90 degrees.

To convert values from degrees to radians, multiply the angle (in degrees) by π/180 (or 0.017453).

To convert a radian value to degrees, multiply it by 180/π ( or 57.295779).

**EXAMPLE:**

```
X=ATN(1)          ' returns .785398 radians, which is 45 degrees.
```

# ATN2 — Mathematics Function

**ACTION:** Returns the angle (in radians) whose tangent is y/x.

**PROGRAM SYNTAX:** ATN2(y,x) - used in an expression

**REMARK:** The arctangent returns an angle in the range -π to π radians. π radians equals 180 degrees.

To convert values from degrees to radians, multiply the angle (in degrees) time π/180 (or .017453).

To convert a radian value to degrees, multiply it by 180/π ( or 57.295779).

**EXAMPLE:**

```
X=ATN2(2.5,3)     ' returns .694727 radians which is 39.8 degrees
```

# BCD — I/O Function

**ACTION:** Returns the value of the BCD switches connected to an Expansion I/O port.

**PROGRAM SYNTAX:** BCD(b0n) - used in an expression

**REMARK:** The b specifies the Expansion I/O board number (1-4).

The n specifies the BCD switch bank number (1-8).

BCD(b0n)
Evaluates and returns the number set on the BCD board switch bank "n", connected to Expansion I/O board "b".

**EXAMPLE:** X=BCD(101)
Sets X equal to the value read on board 1, BCD switch bank 1.

X=BCD(405)
Sets X equal to the value read on board 4, BCD switch bank 5.

# BOOST — Motion Parameter

**ACTION:** Enables or disables the Boost Current feature or returns the boost enable status for the specified axis. When enabled the **stepper drive** BOOST output turns on during motion. This causes the **stepper drive** to boost the motor current by 50%.

**PROGRAM SYNTAX:** BOOST(axis)=expression
BOOST=expression1, ... , expression8
BOOST(axis, ... , axis)=expression, ... , expression
BOOST(axis) - used in an expression

**REMARK:** For more detail refer to Section 10 **Stepper Drive** of this manual.

# BUSY — Motion Parameter

**ACTION:** Returns the motion status of the specified axis. An axis is busy if motion is taking place.

**PROGRAM SYNTAX:** BUSY(axis) - used in an expression

**REMARK:** The axis specifies the number of the axis (1-8).

If the commanded motion is incomplete a true (+1) is return otherwise a false (0) is returned. BUSY is the complement of command DONE.

**EXAMPLE:**
```
DO
    statements
LOOP UNTIL BUSY(1)=0
```

# CAPPOS Miscellaneous Command

**ACTION:** Returns the last captured absolute position of an axis from a MOVEHOME, MOVEREG or CAPTURE cycle.

**PROGRAM SYNTAX:** CAPPOS(axis) - used in an expression

**REMARK:** The axis specifies the number of the axis (1-8).

This command can be used in conjunction with a MOVEHOME, MOVEREG or CAPTURE command to specify the last captured absolute position in Units. The captured position is the position where the trigger occurred during a MOVEHOME, MOVEREG or CAPTURE cycle.

**EXAMPLE:**

```
POSMODE(1)=1                ‘set absolute position mode
ABSPOS(1)=0                 ‘set absolute position to zero
CAPTURE(1)=0                ‘set capture trigger to Event 1 active
JOG(1)=1                    ‘start Jog cycle
DO
LOOP UNTIL CAPTURE(1)=1 ‘ wait for capture to occur
STOP(1)                     ‘stop Jog cycle
WAITDONE(1)                 ‘wait for motion to stop
MOVE=CAPPOS(1)              ‘move to capture position
WAITDONE(1)                 ‘wait for motion to stop
END
```

# CAPTURE Miscellaneous Command

**ACTION:** Sets the position capture trigger condition or returns the position capture status.

**PROGRAM SYNTAX:**
CAPTURE(axis)=expression
CAPTURE=expression1, … , expression8
CAPTURE(axis, … , axis)=expression, … , expression
CAPTURE(axis) – used in an expression

**REMARK:** The axis specifies the number of the axis (1-8).

The expression selects the trigger condition (0-7).

Setting the capture condition arms the position capture function. When the trigger condition is met the position capture occurs. The captured position can be read via the CAPPOS function. If the axis is configured as an open loop device the Absolute Position is captured. If the axis is configured as a closed loop device the Encoder Position is captured. The following trigger conditions can be set:

| Trigger value | Trigger description |
|---|---|
| 0 | Event 1 active |
| 1 | Event 1 inactive |
| 2 | Event 1 active & encoder marker |
| 3 | Event 1 inactive & encoder marker |
| 4 | Encoder marker active |
| 5 | Encoder marker inactive |
| 6 | Event 2 active |
| 7 | Event 2 inactive |

When reading the CAPTURE status a zero indicates no capture has occurred. While a 1 indicates that a capture has occurred.

**EXAMPLE:**

```
POSMODE(1)=1              'set absolute position mode
ABSPOS(1)=0               'set absolute position to zero
CAPTURE(1)=0              'set capture trigger to Event 1 active
JOG(1)=1                  'start Jog cycle
DO
LOOP UNTIL CAPTURE(1)=1 ' wait for capture to occur
STOP(1)                   'stop Jog cycle
WAITDONE(1)               'wait for motion to stop
MOVE=CAPPOS(1)            'move to capture position
WAITDONE(1)               'wait for motion to stop
END
```

## CHR$ String Manipulation

**ACTION:** Returns a one character string whose ASCII code is the argument.

**PROGRAM SYNTAX:** CHR$(code)

**REMARK:** CHR$ is commonly used to send a special character to the serial port.

**EXAMPLE:**
```
PRINT#1,"Input Accel",CHR$(27)
```
Transmits "Input Accel <ESC>" to the host serial port.

## COMMON Miscellaneous Command

**ACTION:** Allows variables to be shared by other tasks.

**PROGRAM SYNTAX:** COMMON variable[,variable][,variable]

**REMARK:** If a variable defined in one task is to be used in another task, the variable name must be declared by the COMMON statements in both tasks, COMMON statements should be placed at the start of the task.

**EXAMPLE:**
```
----------TASK 1----------
COMMON X          'shared variable

----------TASK2----------
COMMON X          'shared variable
```

## COS Mathematics Function

**ACTION:** Returns the cosine of the angle x, where x is in radians.

**PROGRAM SYNTAX:** COS(x) - used in an expression

**REMARK:** To convert values from degrees to radians, multiply the angle (in degrees) by $\pi/180$ (or 0.017453).

To convert a radian value to degrees, multiply it by $180/\pi$ (or 57.295779).

**EXAMPLE:**
```
PI=3.141593
A=COS(PI/3)       ' sets A=0.5, which is the cosine of 60°
```

# DATA — Miscellaneous Command

**ACTION:** Stores the numeric constants used by the READ statement.

**PROGRAM SYNTAX:** DATA constant, constant, etc

**REMARK:** The constant is a numeric constant.

**EXAMPLE:** DATA 1,2,3,4,5,6
Also see the example for the READ command.

# DECEL — Trajectory Parameter

**ACTION:** Sets or returns the deceleration value of an axis.

**PROGRAM SYNTAX:** DECEL(axis)=expression
DECEL=expression1, ... ,expression8
DECEL(axis, ... ,axis)=expression, ... , expression
DECAL(axis) - used in an expression

**REMARK:** The axis specifies the number of the axis (1-8).

The expression defines the deceleration rate is in units/sec$^2$.

The rate at which the motor speed is decreased. Specifying a value greater than "Max Accel" (set in *Configuration and Setup*) will result in DECEL being set to "Max Accel". DECEL can be set during motion, but the new setting will not be used until the next move. Reading DECEL returns the most recent setting.

**EXAMPLE:** DECEL(2)=50
Sets the deceleration rate for axis 2 to 50 units/sec$^2$.

DECEL=50,,75
Sets the deceleration rate for axis 1 to 50 units/sec$^2$ and axis 3 to 75 units/sec$^2$.

DECEL(1,3)=50,75
Sets the deceleration rate for axis 1 to 50 units/sec$^2$ and axis 3 to 75 units/sec$^2$.

# #DEFINE Miscellaneous Command

**ACTION:** Defines a symbolic name to be a particular string of characters.

**PROGRAM SYNTAX:**
#DEFINE name@1, ... , @10 replacement text
#DEFINE replacement text

**REMARK:** The name has the same form as a variable name: a sequence of letters and digits that begins with a letter. The name is case sensitive. Typically upper case is used for the name.

The @1, ... , @10 are the program command substitution arguments for the replacement text.

The replacement text can be any sequence of letters or characters.

Any occurrence of the name in the program, not in quotes and not as part of another name, will be replaced by the corresponding replacement text when the program is compiled.

**EXAMPLE:**
#DEFINE TRUE 1
Substitutes a 1 when the name TRUE is encountered.

#DEFINE FALSE 0
Substitutes a 0 when the name FALSE is encountered.

#DEFINE SENDPOS @1,@2 PRINT#@1,ABSPOS(@2)
Sends the absolute position of axis @2 via port @1.

SENDPOS 1,2
Sends the absolute position of axis 2 via port #1. The 1 is substituted for the @1 argument and 2 is substituted for @2 argument.

```
#DEFINE CLR   PRINT#2,CHR$(12);
#DEFINE LOCATE @1,@2     PRINT#@,CHR$(27);"[@1;@2H";

CLR                ' clear display
LOCATE 1,2         ' locate cursor at row 1 column 2
```

# DELTACAPPOS Miscellaneous Command

**ACTION:** Returns the difference between the current captured position and the previously captured position.

**PROGRAM SYNTAX:** DELTACAPPOS(axis) – used in an expression

**REMARK:** The axis specifies the number of the axis (1-8).

The current captured position can be read with the CAPPOS function. The value returned by DELTACAPPOS is not valid until at least two captures have occurred.

**EXAMPLE:**

```
POSMODE(1)=1                ‘set absolute position mode
ABSPOS(1)=0                 ‘set absolute position to zero
CAPTURE(1)=0                ‘ set trigger to EVENT 1 active
JOG(1)=1                    ‘start Jog cycle
DO
LOOP UNTIL CAPTURE(1)=1  ‘wait for capture trigger
STOP(1)                     ‘stop Jog cycle
WAITDONE(1)                 ‘wait for motion to be completed
CAPTURE(1)=0                ‘ set trigger to EVENT 1 active
JOG(1)=1                    ‘start Jog cycle
DO
LOOP UNTIL CAPTURE(1)=1  ‘wait for capture trigger
STOP(1)                     ‘stop Jog cycle
WAITDONE(1)                 ‘wait for motion to be completed
PRINT#1,”Delta capture position”, DELTACAPPOS(1)
‘print the difference between captured positions
END
```

# DIM Miscellaneous Command

**ACTION:** Declares an array variable and allocates storage space.

**PROGRAM SYNTAX:** DIM variable(dimension,dimension,etc)
DIM variable$(dimension,dimension,etc)

**REMARK:** The "option base zero" for array notation is used, in which the first element of each array dimension is annotated as element "0". Therefore, the total number of elements in the array is: (dimension1 + 1)*(dimension2 + 1) * ... *(dimension n +1).

Example notation for a two-dimensional array:

| | $Y_0$ | $Y_1$ | $Y_2$ | → | $Y_n$ |
|---|---|---|---|---|---|
| $X_0$ | | | | → | |
| $X_1$ | | | | → | |
| $X_2$ | | | | → | |
| ↓ | ↓ | ↓ | ↓ | ↘ | ↓ |
| $X_n$ | | | | → | |

**EXAMPLE:** DIM x(10,10,10)
The variable x is three-dimensional array with 11*11*11 , or 1331 elements.

DIM a$(3,3,3)
The variable string a$ is a three-dimensional array with 4*4*4, or 64 elements.

```
DIM A(3,3,3)          ' 4 * 4 * 4 or 64 elements
A (3,1,2)=5.0
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **0,0,0** | **0,0,1** | **0,0,2** | **0,0,3** | **0,1,0** | **0,1,1** | **0,1,2** | **0,1,3** |
| **0,2,0** | **0,2,1** | **0,2,2** | **0,2,3** | **0,3,0** | **0,3,1** | **0,3,2** | **0,3,3** |
| **1,0,0** | **1,0,1** | **1,0,2** | **1,0,3** | **1,1,0** | **1,1,1** | **1,1,2** | **1,1,3** |
| **1,2,0** | **1,2,1** | **1,2,2** | **1,2,3** | **1,3,0** | **1,3,1** | **1,3,2** | **1,3,3** |
| **2,0,0** | **2,0,1** | **2,0,2** | **2,0,3** | **2,1,0** | **2,1,1** | **2,1,2** | **2,1,3** |
| **2,2,0** | **2,2,1** | **2,2,2** | **2,2,3** | **2,3,0** | **2,3,1** | **2,3,2** | **2,3,3** |
| **3,0,0** | **3,0,1** | **3,0,2** | **3,0,3** | **3,1,0** | **3,1,1** | **3,1,2 5.0** | **3,1,3** |
| **3,2,0** | **3,2,1** | **3,2,2** | **3,2,3** | **3,3,0** | **3,3,1** | **3,3,2** | **3,3,3** |

# DIST — Trajectory Parameter

**ACTION:** Returns the distance moved from the start of the last commanded motion.

**PROGRAM SYNTAX:** DIST(axis) - used in an expression

**REMARK:** The axis specifies the number of the axis (1-8).

Returns a positive number, regardless of the move direction.

**EXAMPLE:**
```
x=DIST(2)
```
Returns the last incremental distance moved in axis 2.

# DO ... LOOP — Program Flow Command

**ACTION:** Repeats a block of statement while a condition is true or until a condition becomes true.

**PROGRAM SYNTAX1:**
```
DO {UNTIL | WHILE} [condition]
    [statement block]
    [EXIT DO]
    [statement block]
LOOP
```

**PROGRAM SYNTAX2:**
```
DO
    [statement block]
    [EXIT DO]
    [statement block]
LOOP {UNTIL | WHILE} [condition]
```

**PROGRAM SYNTAX3:**
```
DO
    [statement block]
    [EXIT DO]
    [statement block]
{UNTIL | WHILE} [condition]
```

**REMARK:** Syntax1 allows the condition to be tested at the top of the loop. Syntax 2 and 3 allows the condition to be tested at the bottom of the loop therefore the loop will always execute at least once.

EXIT DO is an alternate exit from a DO ... LOOP.

EXIT DO transfers control to the statement below the above syntax's and can only be used in a DO ... LOOP statement.

**EXAMPLE:**
```
DO
    statements
WHILE EVENT1(1) <>1  'continue the loop while event1 does not
                       equal 1
```

# DONE Motion Parameter

**ACTION:** Returns the motion status of the designated axis. DONE means motion is completed.

**PROGRAM SYNTAX:** DONE(axis) - used in an expression

**REMARK:** The axis specifies the number of the axis (1-8).

If the commanded motion of an axis is complete a True (1) is returned otherwise a False(0) is returned. DONE is the complement of BUSY.

**EXAMPLE:** X=DONE(1)
The motion status of axis 1 is returned to variable X.

```
DO
statements
UNTIL DONE(1)      ' execute do loop statements until axis 1 commanded motion is completed.
```

# DRVREADY Motion Parameter

**ACTION:** Enables or disables the checking of the drive (READY) signal on the axis card.

**PROGRAM SYNTAX:**
DRVREADY(axis)=expression
DRVREADY=expression1, … , expression8
DRVREADY(axis, … ,axis)=expression, … , expression
DRVREADY(axis) - used in an expression

**REMARK:** Axis specifies the number of the axis (1-8).

The expression sets the enable/disable checking of the Drive READY signal. A 0 enables checking of the axis Drive Ready signal and a 1 disables the signal checking.

Each axis has a hardware Drive Ready input and a software DRVREADY flag. The software flag is cleared during the process of running a project. If motion is commanded and the Drive READY input is not active or the DRVREADY flag is not set then an error will be signaled.

The DRVREADY flag is set using the DRVREADY command, once set the state of the Drive Ready input doesn't matter. The DRVREADY command also returns the DRVREADY status which is the logical OR of the Drive Ready input and the DRVREADY flag.

**EXAMPLE:**
DRVREADY(3)=1
Bypasses the Drive Ready signal checking for axis 3.

DRVREADY=1,,1
Bypasses the Drive Ready signal checking for axis 1 and axis 3.

DRVREADY(1,3)=1,1
Bypasses the Drive Ready signal checking for axis 1 and axis3.

# ENCBAND — Motion Parameter

**ACTION:** Sets or returns the maximum position error allowed when motion is stopped, referred to herein as "position error band."

**COMMAND SYNTAX:**

ENCBAND(axis) =expression
ENCBAND=expression1, … , expression8
ENCBAND(axis, … , axis)=expression, … , expression
ENCBAND(axis) - Used in an expression

**Note: STOPERR can be substituted for ENCBAND.**

**REMARKS**: STOPERR is a stepper drive and servo drive parameter.

STOPERR is defined in detail in both Section 9 Servo Drive and Section 10 Stepper Drive.

# ENCERR — Trajectory Parameter

**ACTION:** Returns the positional error of the designated axis.

**COMMAND SYNTAX:** ENCERR(axis) – used in an expression.

**Note: POSERR can be used in place of ENCERR**

**REMARKS:** Axis specifies the number of the axis(1-8).

Position error is the difference between the absolute position and the encoder position. (ABSPOS – ENCPOS)

**EXAMPLES:**

```
X=ENCERR(1)
IF X > 10 THEN
        PRINT#1,"Large Error"
END IF
```

# ENCFOL — Motion Parameter

**ACTION:** Sets or returns the maximum positional error ("following error") allowed during motion.

**COMMAND SYNTAX:**

ENCFOL(axis) =expression
ENCFOL=expression1, … , expression8
ENCFOL(axis, … , axis)=expression, … , expression
ENCFOL(axis) - Used in an expression

**Note: FOLERR can be substituted for ENCFOL.**

**REMARKS:** FOLERR is a stepper drive and servo drive parameter.

FOLERR is described in detail in both Section 9 Servo Drive and Section 10 Stepper Drive.

## ENCMODE — Motion Parameter

**ACTION:** Sets or returns the operating mode of a closed loop stepper axis.

**PROGRAM SYNTAX:**
ENCMODE(axis)=expression
ENCMODE=expression1, ... , expression8
ENCMODE(axis, ... ,axis)=expression, expression
ENCMODE(axis) - used in an expression

**REMARK:** ENCMODE is defined in detail in the Stepper Drive Section of this manual.

## ENCPOS — Trajectory Parameter

**ACTION:** Returns the encoder position of an axis.

**PROGRAM SYNTAX:** ENCPOS(axis) - used in an expression

**REMARK:** The axis specifies the number of the axis (1-8).

The actual position of the motor. Reading the ENCPOS returns the actual position in user units. ENCPOS is initialized to 0 at power up. Setting ABSPOS sets ENCPOS to the same value.

**EXAMPLE:**
```
X=ENCPOS(1)          'returns the encoder value of axis 1.
```

## ENCSPD — Trajectory Parameter

**ACTION:** Returns the current encoder speed in units/second.

**PROGRAM SYNTAX:** ENCSPD(axis) - used in an expression

**REMARK:** The axis specifies the number of the axis (1-8).

The encoder speed is monitored at the sample rate selected for the axis. This results in an encoder count/sample time value that is converted to units/second. Since this value is digital and not a filtered velocity, deviations will result.

**EXAMPLE:**
X=ENCSPD(2)
Sets variable X to the current encoder speed of axis 2.

```
outputspd=0                          ' initial value
FOR x=1 TO 10                        ' number of samples
   outputspd=outputspd+ENCSPD(1)     ' sample update
   wait=.001                         ' sample time
NEXT x
```

```
outputspd=outputspd/10                    ‘ filtered value
```

# END Miscellaneous Command

**ACTION:** Signifies the end of a program.

**PROGRAM SYNTAX:** END

**REMARK:** If motion is occurring when this command is encountered the controller will set a WARNING number and stop motion on the applicable axis.

**EXAMPLE:**

```
statements
...
END
```

# ERR Miscellaneous Command

**ACTION:** Returns the MX controller error/warning number for this task.

**PROGRAM SYNTAX:** ERR=error number,severity
ERR - used in an expression

**REMARKS:** If an error occurs while the program is running, the program jumps to label ERROR_HANDLER if it is present, otherwise it ends. The fault LED blinks the ERROR code or WARNING code.

If an error or warning has occurred, the axis which caused the error can be obtained by issuing an ERRAXIS command.

The predefined error codes are:

| | |
|---|---|
| 1 | Axis +Limit Input activated while moving in the + direction. |
| 2 | Axis -Limit Input activated while moving in the - direction. |
| 3 | Axis Soft Limit exceeded while moving in the + direction. |
| 4 | Axis Soft Limit exceeded while moving in the - direction. |
| 5 | Closed Loop Correction attempts exceeded. |
| 6 | Position Error exceeded during motion. |
| 7 | Distance after Movereg Trigger is insufficient to decelerate. |
| 8 | Motion was attempted when the Drive is not ready. |
| 9 | Servo axis motion attempted with the Drive disabled. |
| 10 | Program Area out of memory |
| 26 | Excessive Duty Cycle Shutdown (IXT error) |
| 27-99 | User defined |

**Notes: The error trap is only enabled if the error code is 0. This can be accomplished by programming an ERR=0 statement in the ERROR_HANDLER routine. The error code is set to 0 when the program is started by a RUN command or auto started on power on.**

The predefined Warning codes are:

| | |
|---|---|
| 11 | Command axis is not in task group. |
| 12 | Analog I/O selected is out of range. |
| 13 | BCD selected is out of range. |
| 14 | Expansion Input selected is out of range. |
| 15 | Expansion Output selected is out of range. |
| 16 | Digital Input selected is out of range. |
| 17 | Digital Output selected is out of range. |
| 18 | LOG command argument is zero or negative |
| 19 | SQRT command argument is negative. |
| 20 | NVR element is out of range. |
| 21 | READ command is out of data arguments. |
| 22 | MAXSPD command is out of range. |
| 23 | Motion occurring at program end. |
| 24 | RS232 Configuration Error. |
| 25 | Servo Parameter is out of range. |

ERR - used in an expression
Returns the last task error/warning number.

If an error or warning occurs during program execution the fault LED will blink the error code. If the error code is >= 10 the fault LED blinks on 0.25 seconds and off 0.5 seconds for each ten's digit. The LED goes off for 1.25 seconds. If the LSB digit is 0 the LED stays on for 1 second and then goes off for 2.5 seconds. Otherwise, the fault LED blinks on 0.25 seconds and off 0.5 seconds for each unit digit then goes off for 2.5 seconds.

ERR=error number,severity
Used as a user defined error to set an error number and severity of the error. If the error number is a 0 the fault LED will be turned off. The severity levels are:

| | |
|---|---|
| 1 | Stop motors in task at the maximum Deceleration rate. |
| 2 | Stop motors in task at the deceleration rate of each axis. |
| 4 | Stop motors in task at the maximum Deceleration rate. |
| 16 | Create an Error Trap if no other error is set. |

**EXAMPLES:**

x = ERR
sets variable x equal to the present error number or warning number for this task.

ERR = 0,0
clears the error number and enables the error trap.

ERR = nn,1
sets the error number at "nn" and stops all motors in task at the maximum deceleration rate.

ERR = nn,2
sets the error number at "nn" and stops all motors in task at the deceleration rate of each axis.

**ERROR HANDLING:**

The MX2000 Controller will handle errors in one of two ways. The first method is to stop execution of the task in which the error occurred; all remaining tasks continue executing. This is the default method of handling errors. The second method of handling errors is for the user to write an error handler routine. The routine must use the label **ERROR_HANDLER.** This routine will be called whenever an error occurs. The user may then evaluate the condition that invoked the error handling routine. To re-arm the error trap requires the error to be set to 0 (ERR=0,0).

**EXAMPLES:**

ERR=26,17
When this line of code is executed, an error trap condition is created. If an error handling routine has been written execution will resume at the **ERROR_HANDLER** routine. The error number is set to 26 and all motors in this task will stop at the maximum deceleration rate.

**ERROR_HANDLER:**

This label defines the start of the code that is to be executed if an error occurs. The last statement in the **ERROR_HANDLER** code should be an **END** or **GOTO label**.

# ERRAXIS — Miscellaneous Command

**ACTION:** Returns the controller axis number which created the error/warning for the task.

**PROGRAM SYNTAX:** ERRAXIS - used in an expression

**REMARKS:** If a zero is returned then the error was not axis related or there is no actual error.

To determine the error/warning use the **ERR** command.

**EXAMPLE:**

```
ERROR_HANDLER:
Axis = ERRAXIS        ‘ returns the axis number which created the
                        error trap.
Error = ERR           ‘ sets error to error trap number
   statements
ERR = 0,0             ‘ clears error number
GOTO ERROR_EXIT
```

# ERRTRAP — Miscellaneous Command

**ACTION:** Sets an Error Trap in the designated task.

**PROGRAM SYNTAX:** ERRTRAP = ”Task name”, Error number

**REMARKS:** Task name specifies the task to error trap in. The task name must be enclosed in quotes. Only the name of the task is required; the complete path and file extension is not required.

Error number sets the error number in the designated task.

**EXAMPLE:**

```
errorflag=0
ERRORCHECK:
If  IN(101)=1 AND errorflag=0 THEN      ‘error condition occurred
        ERRTRAP=”Motion”,55
        ‘ error trap task “Motion” and set error code 55 in Task
        “Motion”.
        errorflag=1
ELSE IF  IN(101)=0 THEN
        errorflag=0
END IF
GOTO ERRORCHECK
```

# EVENT1 Motion Parameter

**ACTION:** Returns the state of the trigger input labeled EVNT1 for the selected axis or sets the trigger polarity and enable, which are used in MOVEHOME, MOVEREG and FOLMOVEREG cycles.

**PROGRAM SYNTAX:**

EVENT1(axis)=expression
EVENT1=expression1, ... , expression 8
EVENT1(axis, ... , axis)=expression, expression
EVENT1(axis) - used in an expression

**REMARKS:** Axis specifies the number of the axis (1-8).

The EVENT1 command is used to select the effects of the hardware signal at the EVNT1 input on the axis card. This input is typically wired to a switch or sensor. It may be used as a home positioning trigger during a MOVEHOME cycle. It may also be used as a position mark registration trigger during a MOVEREG or FOLMOVEREG cycle. When used for mark registration, a trigger on EVNT1 will initiate the index portion of the MOVEREG or FOLMOVEREG cycle.

The EVENT1 triggering for a MOVEHOME or MOVEREG cycle may be combined with an encoder index pulse input, and is assigned in the user program *Configuration and Setup*.

For a **MOVEHOME** cycle, the EVENT1 command may be used to set the polarity of the move home trigger. If the expression of the command is **positive** the home trigger occurs when the EVNT1 input becomes active. If the expression of the command is **negative** the home trigger occurs when the EVNT1 input becomes inactive. A Home cycle trigger EVENT1 cannot be disabled using this command.

For a **MOVEREG** cycle, the EVENT1 command may be used to set the polarity of the registration trigger. If the expression of the command is **positive** the registration trigger occurs when the EVNT1 input becomes active. If the expression of the command is **negative** the registration trigger occurs when the EVNT1 input becomes inactive.

The EVENT1 trigger for a registration cycle may be **disabled** by setting EVENT1=0. A registration trigger may be enabled to either polarity during a move. It may not, however, be disabled once the cycle has begun.

**EXAMPLES:**

```
EVENT1(2)=0     'disables Event1 as a MOVEREG trigger on axis 2.
EVENT1(2)=1     'enables Event1 as an active input trigger on axis 2.
EVENT1(2)=-1    'enables Event1 as an inactive input trigger on axis 2.
```

# EVENT2 Motion Parameter

**ACTION:** Returns the state of the trigger input labeled EVNT2 for the selected axis or sets the trigger polarity and enable, which are used in MOVEHOME, MOVEREG and FOLMOVEREG cycles.

**PROGRAM SYNTAX:**

EVENT2(axis)=expression
EVENT2=expression1, ... , expression 8
EVENT2(axis, ... , axis)=expression, expression
EVENT2(axis) - used in an expression

**REMARKS:** Axis specifies the number of the axis (1-8).

The EVENT2 command is used to select the effects of the hardware signal at the EVNT2 input on the axis card. This input is typically wired to a switch or sensor. It may be used as a home positioning trigger during a MOVEHOME cycle. It also may be used as a position mark registration trigger during a MOVEREG or FOLMOVEREG cycle. When used for mark registration, a trigger on EVNT2 will initiate the index portion of the MOVEREG or FOLMOVEREG cycle.

The EVENT2 triggering for a MOVEHOME or MOVEREG cycle may be combined with an encoder index pulse input, and is assigned in the user program *Configuration and Setup*.

For a **MOVEHOME** cycle, the EVENT2 command may be used to set the polarity of the move home trigger. If the of the command is **positive** the home trigger occurs when the EVNT2 input becomes active. If the expression of the command is **negative** the home trigger occurs when the EVNT2 input becomes inactive. An EVENT2 home trigger cannot be disabled using this command.

For a **MOVEREG** cycle, the EVENT2 command may be used to set the polarity of the registration trigger. If the expression of the command is **positive** the registration trigger occurs when the EVNT2 input becomes active. If the expression of the command is **negative** the registration trigger occurs when the EVNT2 input becomes inactive.

The EVENT2 trigger for a registration cycle may be **disabled** by setting EVENT2=0. A registration trigger may be enabled to either polarity during a move. It may not, however, be disabled once the cycle has begun.

**EXAMPLES:**

```
EVENT2(2)=0  'disables Event2 as a MOVEREG trigger on axis 2.

EVENT2(2)=1  'enables Event2 as an active input trigger on axis 2.

EVENT2(2)=-1  'enables Event2 as an inactive input trigger on axis 2.
```

# EXIN I/O Function

**ACTION:** Returns the state of the specified expansion I/O inputs.

**PROGRAM SYNTAX:** EXIN(nnn) - used in expression
EXIN(nnn,len) - used in expression

**REMARKS:** The nnn is the I/O terminal point.

| | **board1** | **board2** | **board3** | **board4** |
|---|---|---|---|---|
| nnn= | (100-147) or | (200-247) or | (300-347) or | (400-447) |

len is the number of I/O points. len range is 1-24.

**SINGLE INPUT SYNTAX:** EXIN(nnn)
returns the state (1 or 0) of the designated input.

**EXAMPLE:** x=EXIN(207) ' returns the state of board 2 input 7

**MULTIPLE INPUT SYNTAX:** EXIN(nnn,len)
returns a number corresponding to the states of multiple inputs calculated from the binary weighting of inputs nnn to (nnn+len-1) nnn is the first input and the len is the number of inputs.

**EXAMPLES:** EXIN(nnn,len) is equivalent to:
EXIN(nnn),+2*EXIN(nnn+1) ,+ 4*EXIN(nnn+2) + , ... , $+2^{len-1}$ *EXIN(nnn+len-1)

EXIN(207,3) is equivalent to:
EXIN(207) + 2*EXIN(208) + 4*EXIN(209) depending on the state of inputs 207-209 (EXIN(207,3) will return a number between 0 and 7. So, if the inputs are: 207=0ff, 208=on and 209=on. The resulting value returned would be 6. EXIN(207,3)= 110(binary)

# EXOUT I/O Function

**ACTION:** Sets or returns the state of the specified expansion I/O outputs.

**PROGRAM SYNTAX:**
EXOUT(nnn) - used in expression
EXOUT(nnn,len) - used in expression
EXOUT(nnn)=expression
EXOUT(nnn,len)=expression

**REMARKS:** The nnn is the I/O terminal point.

| board1 | board2 | board3 | board4 |
|---|---|---|---|
| nnn= (100-147) | or (200-247) | or (300-347) | or (400-447) |

len is the number of I/O points. len range is 1-24.

**SET SINGLE OUTPUT SYNTAX:**
EXOUT(nnn)=expression
"expression" turns output nnn on (expression is non-zero) or off (expression=0).

**EXAMPLES:**
```
EXOUT(207)=-3      'turns output 7 on board 2 on
EXOUT(207)=0       'turns output 7 on board 2 off
```

**READ SINGLE OUTPUT SYNTAX:**
EXOUT(nnn) - used in expression
returns the last output commanded (1 or 0) for this I/O pin.
Note: this is different from the state of the I/O pin.

**EXAMPLES:**
```
EXOUT(207)=1       'board 2 output 7 is turned on.
A=EXOUT(207)       'A is set to 1 (last commanded output for 207).
```

**SET MULTIPLE OUTPUTS SYNTAX:**
EXOUT(nnn,len)=expression
The expression is evaluated and converted to an integer value. The least significant "len" bits of the binary representation are then used to set outputs "nnn" to "nnn+len-1" respectively.

**EXAMPLES:**
```
EXOUT(207,3)=6.2   'sets outputs 207-209
```
6.2 is converted to integer 6, the binary representation of 6 is 110. Thus output 207=off, output 208=on and output 209=on.

**READ MULTIPLE OUTPUTS SYNTAX:**
EXOUT(nnn,len) - used in an expression
Evaluates to a number corresponding to the last outputs commanded (1or 0) for these I/O pins. The number is the binary weighted sum of the last commanded outputs nnn to (nnn+len-1). Note: this is different from the state of the I/O pins.

**EXAMPLES:**
```
EXOUT(207,3)=4     'output 209=on, output 208=off and output
                    207=off.

A=EXOUT(208,2)     'A=2 since output 209=on and output 208=off.
```

# FEEDRATE — Trajectory Parameter

**ACTION:** Sets a feed rate override during Path execution.

**PROGRAM SYNTAX:** FEEDRATE = expression

**REMARKS:** The expression range is .01 to 10.0 (1% to 1000%). This value scales the commanded velocity to obtain a target velocity.

This command is only honored during PATH or ARC execution.

**EXAMPLE:**

```
PATH 1,2
    FEEDRATE=.5
    LINE=expression1,expression2
    statements
PATH END
```

# FOLACCDIST — Following Parameter

**ACTION:** Specifies the master distance traveled for the follower to catch the master velocity after follower motion begins.

**PROGRAM SYNTAX:**

```
FOLACCDIST(axis)=expression
FOLACCDIST=expression1, ... , expression8
FOLACCDIST(axis, ... ,axis)=expression, ... , expression
FOLACCDIST(axis) - used in an expression
```

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLDCCDIST — Following Parameter

**ACTION:** Specifies the master distance traveled for the follower to attain a velocity of zero from the current velocity.

**PROGRAM SYNTAX:**

```
FOLDCCDIST(axis)=expression
FOLDCCDIST=expression1, ... , expression8
FOLACCDIST(axis, ... ,axis)=expression, ... , expression
FOLDCCDIST(axis) - used in an expression
```

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLERR — Motion Parameter

**ACTION:** Sets or returns the maximum position error allowed during motion, herein referred to as "following error."

**COMMAND SYNTAX:**
FOLERR(axis)=expression
FOLERR=expression1, number2, . . . , number8
FOLERR(axis, … , axis)=expression, … , expression
FOLERR (axis) - Used in an expression

**Note: ENCFOL can be substituted for FOLERR.**

**REMARKS:** This command is defined in more detail in Section 9 Servo Drive and Section 10 Stepper Drive.

# FOLINPUT — Following Parameter

**ACTION:** This command specifies the follower axes and the master velocity source.

**PROGRAM SYNTAX:** FOLINPUT(axis, ... ,axis)= expression

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLJOG — Following Parameter

**ACTION:** Requests a Following axis jog cycle.

**PROGRAM SYNTAX:**
FOLJOG(axis)=expression
FOLJOG=expression1 , ... , expression8
FOLJOG(axis, ... , axis)= expression, ... , expression

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLMAXRATIO — Following Parameter

**ACTION:** Sets or returns the maximum allowable following axis speed during an offset advance cycle.

**PROGRAM SYNTAX:**
FOLMAXRATIO(axis)=expression
FOLMAXRATIO=expression1 , ... , expression8
FOLMAXRATIO(axis, ... ,axis)=expression, ... , expression
FOLMAXRATIO(axis) - used in an expression

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLMINRATIO — Following Parameter

**ACTION:** Sets or returns the minimum allowable following axis speed during a recede offset cycle.

**PROGRAM SYNTAX:**
FOLMINRATIO(axis)=expression
FOLMINRATIO=expression1 , ... , expression8
FOLMINRATIO(axis, ... ,axis)=expression, ... , expression
FOLMINRATIO(axis) - used in an expression

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLMOVE — Following Parameter

**ACTION:** Request a Following axis move.

**PROGRAM SYNTAX:**
FOLMOVE(axis)=expression
FOLMOVE=expression1 , ... , expression8
FOLMOVE(axis, ... , axis)=expression, ... ,expression

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLMOVEREG — Following Parameter

**ACTION:** Request a Following axis move registration cycle.

**PROGRAM SYNTAX:**
FOLMOVEREG(axis)=expression
FOLMOVEREG=expression1 , ... , expression8
FOLMOVEREG(axis, ... , axis)=expression, ... , expression

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLOFFSET — Following Parameter

**ACTION:** Defines a following incremental offset distance to advance or recede from the master .

**PROGRAM SYNTAX:**
FOLOFFSET(axis)=expression
FOLOFFSET=expression1, ... , expression8

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLOFFSETDIST — Following Parameter

**ACTION:** Sets or returns the master distance traveled for a FOLOFFSET command.

**PROGRAM SYNTAX:**
FOLOFFSETDIST(axis)=expression
FOLOFFSETDIST=expression1, ... , expression8
FOLOFFSETDIST(axis, ... , axis)=expression, ... , expression
FOLOFFSETDIST(axis) - used in an expression

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLRATIO — Following Parameter

**ACTION:** Sets the ratio of the following axis to the master. A value of 1 represents 100% of the master.

**PROGRAM SYNTAX:**
FOLRATIO(axis)=expression
FOLRATIO=expression1, ... , expression8
FOLRATIO(axis, ... ,axis)=expression, ... , expression
FOLRATIO(axis) - used in an expression

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLRATIOINC — Following Parameter

**ACTION:** Specifies the acceleration rate for a folratio change during motion in ratio increment per second.

**PROGRAM SYNTAX:**
FOLRATIOINC(axis)=expression
FOLRATIOINC=expression1, ... , expression8
FOLRATIOINC(axis, ... , axis)=expression, ... , expression
FOLRATIOINC(axis) - used in an expression

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLSTARTDIST — Following Parameter

**ACTION:** Specifies a master distance which is used as a delay distance for starting motion. The distance delay starts when the specified starting trigger of a FOLTRIG command occurs.

**PROGRAM SYNTAX:**
FOLSTARTDIST(axis)=expression
FOLSTARTDIST=expression1, ... , expression8
FOLSTARTDIST(axis) - used in an expression
FOLSTARTDIST(axis, ... .axis)=expression, ... , expression

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLSYNC — Following Parameter

**ACTION:** Returns the following sync status of the specified axis.

**PROGRAM SYNTAX:** FOLSYNC(axis) - used in an expression

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLSYNCDIST — Following Parameter

**ACTION:** Specifies a master distance to travel when a FOLOFFSET command is issued. This distance will be traveled before execution of the FOLOFFSET command .

**PROGRAM SYNTAX:**
FOLSYNCDIST(axis)=expression
FOLSYNCDIST(axis)=expression1, ... , expression8
FOLSYNCDIST(axis) - used in an expression
FOLSYNCDIST(axis, ... ,axis)=expression, ... ,expression

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FOLTRIG — Following Parameter

**ACTION:** Defines the follower starting trigger for motion.

**PROGRAM SYNTAX:**
FOLTRIG(axis)=expression
FOLTRIG=expression1, ... , expression8
FOLTRIG(axis, ... ,axis)=expression, ... , expression
FOLTRIG(axis) - used in an expression

**REMARKS:** This command is defined in more detail in Section 8 Following.

# FORMAT Miscellaneous Command

**ACTION:** Enables or disables the formatting of the STR$ returned string.

**PROGRAM SYNTAX:** FORMAT=m,n,d

**REMARKS:** This command is used in conjunction with the STR$ command to set the format of the returned string.

The m specifies the format mode.

0 disable format
1 leading and trailing 0=s will be returned in the string.
2 sign followed by leading spaces and trailing 0's will be returned in the string.

The n specifies the number of whole digits to be returned in the string. This number does not include the sign of the returned string. If the sign is positive a space will be inserted in place of the sign. If this value is 0 the whole number value will be ignored.

The d specifies the number of decimal digits to be returned in the string. If this value is 0 no decimal point will be returned and the fractional portion of the variable will be ignored.

FORMAT=0,n,d
Disables the format mode. No leading or trailing characters are inserted in the string.

If the number converted is outside the whole number digit the returned string will have * substitutions for the numbers.

**EXAMPLES:**

```
FORMAT=1,4,2              › sign leading & trailing 0'S
ABSPOS(1)= -200.254
A$=STR$(ABSPOS(1))        › A$ =A-0200.25"

FORMAT=2,4,4              › sign leading space trailing 0's
ABSPOS(1)= -200.254
A$=STR$(ABSPOS(1))        › A$ =A- 200.2540"

FORMAT=1,2,4              ›Sign leading & trailing 0's
ABSPOS(1)= -200.254
A$=STR$(ABSPOS(1))        › A$ =A-**.****"

FORMAT=0,1,1              ›Disable formatting
```

# FOR ... NEXT ... STEP — Program Flow Control

**ACTION:** Repeats a block of statements a specified number of times.

**PROGRAM SYNTAX:**

```
FOR counter=start# TO end# [STEP increment]
    statements
    [EXIT FOR]
    statements
NEXT [counter]
```

**REMARKS:**

*Counter* is a variable used as the loop counter.

*Start#* is the initial value of the counter.

*End#* is the end value of the counter.

*Increment* is the amount the counter is changed each time through the loop. If STEP is not specified, *increment* defaults to one.

If *end#* is greater than *start#* then *increment* must be positive. If *start#* is greater than *end#* then *increment* must be negative. If these conditions are not met the loop will not execute, control is transferred to the next statement following the NEXT statement. If *start#* equals *end#* then the loop will execute once regardless of the *increment* value. If *increment* equals zero the loop will execute indefinitely.

EXIT FOR is an alternate exit from a FOR ... NEXT loop.

EXIT FOR transfers control to the statement following the NEXT statement. When used within nested FOR ... NEXT statements, EXIT FOR transfers out of the immediate enclosing loop. EXIT FOR can be used only in a FOR ... NEXT statement.

**EXAMPLES:**

```
FOR x=1 TO 8 STEP 1
    statements
NEXT x

A=2.4
FOR X =1 to A STEP 1
     Statements
NEXT X        ' This loop will execute 2 times (X=1 & X=2)
```

# GETCHAR I/O Command

**ACTION:** Waits for a character on the selected serial port and returns the ASCII code of the character.

**PROGRAM SYNTAX:** GETCHAR(n) - used in an expression

**REMARKS:** The n specifies the serial port number (1 or 2). Port 1 is the Host Port and Port 2 is the Auxiliary Port.

Program execution is suspended while GETCHAR waits for a character to be received by the designated port. If a character is already in the receiver buffer the ASCII code of the character is returned immediately.

**EXAMPLES:**

```
a=GETCHAR(1)       ' sets a to the ASCII code of Host Port character
b=GETCHAR(2)       ' sets b to the ASCII code of Aux Port character
a$=a$ + CHR$(a)    ' add Host character to a$
b$=b$ + CHR$(b)    ' add Auxiliary character to b$
```

# GOSUB ... RETURN Program Flow Control

**ACTION:** Branches to, and returns from, a subroutine.

**PROGRAM SYNTAX:** GOSUB line label

**REMARKS:** You can call a subroutine any number of times in a program. You can call a subroutine from within another subroutine, this is called nesting.

How deeply you can nest is limited only by the available stack area. Subroutines that call themselves (recursive subroutines) can easily run out of stack space.

The execution of the RETURN statement cause program execution to continue with the line immediately following the line that called the subroutine.

Subroutines can appear anywhere in the program, but it is good programming practice to make them readily distinguishable from the main program.

**EXAMPLES:**

```
GOSUB GET_CHAR
    statements
END

GET_CHAR:
    statements
RETURN
```

# GOTO — Program Flow Control

**ACTION:** Branches Unconditionally to the specified label.

**PROGRAM SYNTAX:** GOTO label

**REMARKS:** The GOTO statement provides a mean for branching unconditionally to another label.

It is good programming practice to use subroutines or structured control statements ( DO ... UNTIL, FOR ... NEXT, IF ... THEN ... ELSE IF ... ELSE) instead of GOTO statements, because a program with many GOTO statements can be difficult to read and debug. **Try to avoid using GOTO!**

**EXAMPLES:**

```
IF x=1 THEN GOTO COOLANT_OFF
statements

COOLANT_OFF:
statements
```

# HARDLIMIT Over Travel Limit

**ACTION:** Enables or Disables Hard Limit Switches or returns the current Hard limit Enable/disable state of an axis.

**PROGRAM SYNTAX:**
HARDLIMIT(axis)=expression
HARDLIMIT=expression1, ... , expression8
HARDLIMIT(axis, ... ,axis)=expression, ... , expression
HARDLIMIT(axis) - used in an expression

**REMARKS:** The axis specifies the axis (1-8)

Hard limit inputs are used to stop the motor before it runs into a physical end of travel, thus avoiding damage to the mechanical system. A separate hard limit input is provided for + and - motor rotation on each axis. Activating the trigger level of the + limit input stops the motor if it is traveling in the + direction. Activating the trigger level of the - limit input stops the motor if it is traveling in the - direction.

A True(1) is returned is the axis HARDLIMIT is enabled. A False(0) is returned is the axis HARDLIMIT is disabled.

**EXAMPLES:**
HARDLIMIT(2)=1
Enables the hard limits for axis 2.

HARDLIMIT=1,,0
Enables the hard limits on axis 1 and disables the hard limits on axis 3.

HARDLIMIT(1,3)=1,0
Enables the hard limits on axis 1 and disables the hard limits on axis 3.

# HARDLIMNEG — Over Travel Limit

**ACTION:** Returns the - Limit hardware state for the selected axis.

**PROGRAM SYNTAX:** HARDLIMNEG(axis) - used in an expression

**REMARKS:** The axis is the specified axis (1-8).

A false(0) is returned if the designated axis - limit input is inactive. Otherwise, a true(1) will be returned which indicates that the - Limit input is active.

**EXAMPLES:**

```
IF HARDLIMNEG(1)=1 THEN    ' execute statements if - limit
                             active
        statements
END IF
```

# HARDLIMPOS — Over Travel Limit

**ACTION:** Returns the + Limit hardware state for the selected axis.

**PROGRAM SYNTAX:** HARDLIMPOS(axis) - used in an expression

**REMARKS:** The axis is the specified axis (1-8).

A false(0) is returned if the designated axis + limit input is inactive. Otherwise, a true(1) will be returned which indicates that the + Limit input is active.

**EXAMPLES:**

```
IF HARDLIMPOS(1)=1 THEN    ' execute statements if + limit
                             active
        statements
END IF
```

# HEX$ — String Manipulating

**ACTION:** Returns the hex string equivalent of an argument.

**PROGRAM SYNTAX:** HEX$(expression) - used with a string array

**REMARKS:** The expression must be an integer value.

**EXAMPLES:**

```
a$=HEX$(255)        ' a$="FF"
```

# HVAL — String Manipulation

**ACTION:** Returns the decimal value of a hexadecimal string.

**PROGRAM SYNTAX:** HVAL(A$) - used in an expression

**REMARKS:** A$ is the designated string variable or string literal. The string variable format is: “0XHH” or “HH”. Where H is an ASCII 0-9 or a-f.

The converted value is returned as a decimal value.

**EXAMPLES:**

```
x=HVAL(“0XFF”)          ‘ x is set to 255.

A$=“1F”
x=HVAL(A$)              ‘ x is set to 31
```

# IF...THEN...ELSE IF... ELSE...END IF — Program Flow Command

**ACTION:** Allows conditional execution based on the evaluation of a Boolean condition.

**PROGRAM SYNTAX 1:** IF condition THEN thenpart [ELSE elsepart]

**PROGRAM SYNTAX 2:**

```
IF condition THEN
    statement block 1
[ELSE IF condition THEN]        ELSE IF and statement block 2 are optional
    statement block 2
        [ELSE]                  ELSE and statement block 3 are optional
    statement block 3
END IF
```

**REMARKS:** The argument condition is an expression that is evaluated as true (nonzero) or false (zero).

The argument statement block includes any number of statements on one or more lines.

The argument thenpart includes the statement or branches performed when the condition is true.

The argument elsepart includes the statement or branch performed when the condition is false. If the ELSE IF or ELSE clause is not present, control passes to the next statement in the program following the END IF.

**EXAMPLES:**

```
Start:
IF BCD(101)=0 THEN
    GOTO program0
ELSE IF BCD(101)=1 THEN
    GOTO program1
ELSE IF BCD(101)=2 THEN
    GOTO program2
ELSE
    GOTO Start
END IF
program0:
        statements
END
program1:
        statements
END
program2:
```

statements
END

# IN — I/O Function

**ACTION:** Returns the state's of the specified digital I/O inputs.

**PROGRAM SYNTAX:**
IN(bnn) - used in an expression
IN(bnn,len) - used in an expression

**REMARKS:** bnn specifies the I/O point terminal.

| board 1 | board 2 | board 3 | board 4 |
|---|---|---|---|
| 101-124 | 201-224 | 302-324 | 401-424 |

len is the number of I/O points to return (1-24).

A true (1) is returned if the state of the input is active. Otherwise, a false (0) is returned.

IN(bnn) - used in an expression
Returns a single input state.

IN(bnn,len) - used in an expression
Returns a number corresponding to the states of multiple inputs, binary weighting of inputs bnn to (bnn+len-1). IN(bnn,len) is equivalent to: IN(bnn) + (2*(bnn+1) + (4*(bnn+2) + ... +($2^{len-1}$*(bnn+len-1)

**EXAMPLES:**
x=IN(207)
The state of board 2 input 7 is returned to variable x.

x=IN(207,3)
The sum of the input states from board 2 inputs 7-9 is returned to x. The value returned will be: IN(207) + (2*IN(208) + (4*IN(209).

# INCHAR — I/O Function

**ACTION:** Return the ASCII code of a character from the designated serial port.

**PROGRAM SYNTAX:** INCHAR(n) - used in an expression

**REMARKS:** The n specifies the serial port (1 or 2). Port 1 is the Host port and Port 2 is the Auxiliary port.

If no character has been received by the designated serial port a 0 is returned. Otherwise, the ASCII code value equivalent is returned.

**EXAMPLES:**

```
DO
      x=INCHAR(2)
LOOP UNTIL x > 0          ' wait for Auxiliary port character.
A$=A$+chr$(x)             ' add character to A$
```

# #INCLUDE — Miscellaneous Command

**ACTION:** Includes a file name with define statements in a user task.

**PROGRAM SYNTAX:** #INCLUDE drive:\subdir\...\subdir\filename.inc

**REMARKS:** Drive is the root directory of the drive.

Subdir is the path required to find the file.

Filename is the include filename with extension .inc.

The include file must be a series of #DEFINE statements only and can be used in any project task file.

The iws.inc file is included in the MCPI software. This file can be used to control a IWS-127-SE, IWS-30-SE or IWS-120-SE interface panel.

**EXAMPLES:**

```
#INCLUDE  c:\mx2000\iws.inc        ' include file iws.inc

#INCLUDE  c:\mx2000\iws30.inc      ' include file iws30.inc
```

# INPUT — I/O Command

**ACTION:** Reads a Line of data from the designated serial port into a string variable.

**PROGRAM SYNTAX:**

```
INPUT#1,N$
INPUT#1,N$,var1$[,var2$][, ...] [,var_n$]
INPUT#2,N$
INPUT#2,N$,var1$[,var2$][, ...] [,var_n$]
```

**REMARKS:** This command accepts input characters until a carriage return or line-feed is received by the designated port.

Multiple arguments strings can be entered on one input line and are separated by a comma.

INPUT#1 designates the Host port and INPUT#2 designates the Auxiliary port as the serial receiver port.

**EXAMPLES:**

```
PRINT#2, "enter accel value, decel value, speed value"
INPUT#2,acc$,dcc$,spd$          ' input variable values
FOR x=1 TO 3                    ' axis numbers 1-3
    ACCEL(x)=VAL(acc$)          ' load ACCEL value
    DECEL(x)=VAL(dcc$)          ' load DECEL value
    SPEED(x)=VAL(spd$)          ' load SPEED value
NEXT x
```

# INSTR — String Manipulation

**ACTION:** Returns the character position of the first occurrence of a specified string in another string.

**PROGRAM SYNTAX:** INSTR(string1$,string2$) - used in an expression

**REMARKS:** Returns the starting position that string2$ matches in string1$. The comparison is case sensitive and returns a 0 if no match is found.

**EXAMPLES:**

```
a$="WE part#215629"
x=INSTR(a$,"part#")      ' x is set to 4 which is the starting position
                           of part#.
```

# INTLIM — Servo Parameter

**ACTION:** Sets the Integral limit for the controller. This is the limit of the contribution to the servo output from the integral of the position error.

**PROGRAM SYNTAX:**

```
INTLIM (axis)=expression
INTLIM=expression1, ... , expression8
INTLIM(axis, ... ,axis)=expression, ... ,expression
INTLIM (axis)  - used in an expression
```

**REMARKS:** This command is defined in more detail in Section 9 Servo Drive.

# IXT — Servo Parameter

**ACTION:** Sets or returns the Excessive Duty Cycle Shutdown time in seconds.

**PROGRAM SYNTAX:**

```
IXT(axis) = expression
IXT = expression, ... , expression
IXT(axis, ... ,axis)=expression, ... ,expression
IXT(axis)  - used in an expression
```

**REMARKS:** This command is defined in more detail in Section 9 Servo Drive.

# JOG Motion Parameter

**ACTION:** Runs the motor continuously in a specified direction.

**PROGRAM SYNTAX:**

JOG(axis)=expression
JOG=expression1, ... ,expression8
JOG(axis, ... ,axis)=expression, ... , expression

**note: JOGSTART can be substituted for JOG.**

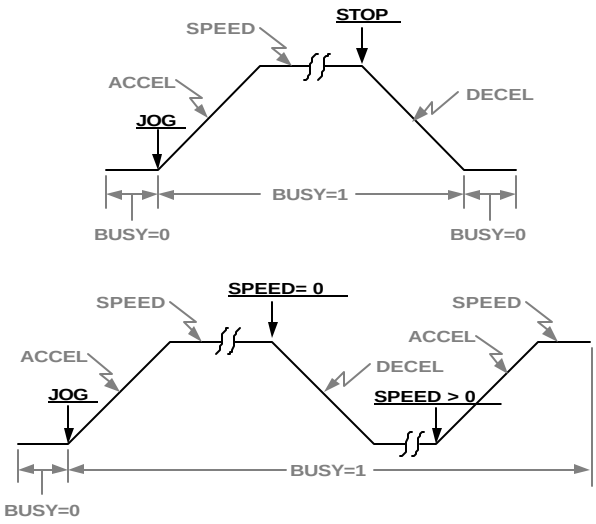
**REMARKS:** The axis specifies the number of the axis (1-8).

The expressions sign determines the motion direction . If the expression is positive or 0, jogging will be in the positive direction. If the expression is negative, jogging is in the negative direction.

Use the STOP command for stopping the motor.

**Note: A JOG cycle will be stopped if an ARC, LINE, MOVE, MOVEHOME or MOVEREG motion command is issued for the same axis during a JOG command.**

**JOG CYCLE**




**Note: The underlined text is the command required to generate the velocity profile. The remaining text are related commands.**

**EXAMPLES:**

```
JOG(2)=-1        'jog axis 2 in the negative direction.
DO : LOOP UNTIL EXIN(101)=1 'Execute Loop until the Expansion
                            I/O input 1 is active.
STOP(2)
WAITDONE(2)      'Allow axis 2 to stabilize at zero speed prior to
                 executing the next command
JOG=1,,-1        'jog axis 1 in +dir. and jog axis 3 in -dir.
DO : LOOP UNTIL EXIN(101)=1
STOP(1,3)
WAITDONE(1,3)
JOG(1,3)=1,-1    'jog axis 1 in +dir. and jog axis 3 in -dia.
DO : LOOP UNTIL EXIN(101)=1
```

```
STOP(1,3)
WAITDONE(1,3)
```

# JOYSTICK — Motion Parameter

**ACTION:** Enables Joystick motion.

**PROGRAM SYNTAX:**
JOYSTICK=ax1, ... ,ax8
JOYSTICK(ax1, ... , ax8)

**REMARKS:**

The JOYSTICK command sets up to eight axes, ax1 to ax8, to move in response to the voltage applied to their respective analog inputs. Each axis will run at a speed proportional to the input voltage and in the direction determined by the polarity of the input voltage. There is a ±0./25 dead band.

The axis will run in the negative direction when the input voltage range is -10.0 to -0.25 volts. The speed it will attain is: ((Vin+0.25)/10)*SPEED. The axis will run in the positive direction when the input voltage range is +0.25 to +10.0 volts. The speed it will attain is: ((Vin-0.25)/10)*SPEED.

The JOYSTICK mode is terminated by a STOP command.

**EXAMPLES:**

```
SPEED(1,2)=10,10          ' set speed for axes
JOYSTICK=1,2              ' enable joystick mode axis 1 and 2
DO: LOOP UNTIL EXIN(100)=1 ' stay in joystick mode until
                                input=1
STOP(1,2)

JOYSTICK(1,2)             ' enable joystick mode axis 1 and 2
DO: LOOP UNTIL EXIN(100)=1 ' stay in joystick mode until
                                input=1
STOP(1,2)
```

# KAFF Servo Parameter

**ACTION:** Sets or returns the acceleration feed forward gain for a servo axis.

**PROGRAM SYNTAX:**
KAFF(axis)=expression
KAFF=expression1, ... , expression8
KAFF(axis, ... ,axis)=expression, ... ,expression
KAFF(axis) - used in an expression

**REMARKS:** This command is defined in more detail in Section 9 Servo Drive.

# KD Servo Parameter

**ACTION:** Sets or returns the derivative gain for the servo axis.

**PROGRAM SYNTAX:**
KD(axis)=expression
KD=expression1, ... , expression8
KD(axis, ... ,axis)=expression, ... ,expression
KD(axis) - used in an expression

**REMARKS:** This command is defined in more detail in Section 9 Servo Drive.

# KI Servo Parameter

**ACTION:** Sets or returns the integral gain of a servo axis.

**PROGRAM SYNTAX:**
KI(axis)=expression
KI=expression1, ... , expression8
KI(axis, ... ,axis)=expression, ... ,expression
KI(axis) - used in an expression

**REMARKS:** This command is defined in more detail in Section 9 Servo Drive.

# KP Servo Parameter

**ACTION:** Sets or returns the proportional gain of the servo axis.

**PROGRAM SYNTAX:**
KP(axis)=expression
KP=expression1, ... , expression8
KP(axis, ... ,axis)=expression, ... ,expression
KP(axis) - used in an expression

**REMARKS:** This command is defined in more detail in Section 9 Servo Drive.

# KVFF — Servo Parameter

**ACTION:** Sets or returns the velocity feed forward gain for the servo axis.

**PROGRAM SYNTAX:**
KVFF(axis)=expression
KVFF=expression1, ... , expression8
KVFF(axis, ... ,axis)=expression, ... ,expression
KVFF(axis) - used in an expression

**REMARKS:** This command is defined in more detail in Section 9 Servo Drive.

# LCASE$ — String Manipulation

**ACTION:** Converts and returns a string with lower case letters.

**PROGRAM SYNTAX:** string1$=LCASE$(string2$)

**REMARKS:** String2$ is copied and all upper case letters are converted to lower case letters and the resulting string is returned to string1$.

**EXAMPLES:**
```
a$="HELLO"
b$=LCASE$(a$)          ' sets b$="hello"
```

# LEFT$ — String Manipulation

**ACTION:** Returns the leftmost characters of a string.

**PROGRAM SYNTAX:** string2$=LEFT$(string1$,n)

**REMARKS:** The n is the number of leftmost characters to return. If n is greater than the length of string1$ then the entire string is returned to string2$.

**EXAMPLES:**
```
b$="Hello World"
a$=LEFT$(b$,7)          ' sets a$="Hello W"
```

# LEN — String Manipulation

**ACTION:** Returns the number of characters in the designated string.

**PROGRAM SYNTAX:** LEN(string$) - used in an expression

**REMARKS:** If the input string is a null string a 0 is returned.

**EXAMPLES:**
```
A=LEN("ABCD")          ' sets A=4
```

# LINE Motion Parameter

**ACTION:** Initiates a coordinated linear move involving up to 8 axes.

**PROGRAM SYNTAX:**
LINE=expression1, ... , expression8
LINE(axis, ... , axis)=expression, ... ,expression
LINE=expression1,expression2 (syntax for PATH command)

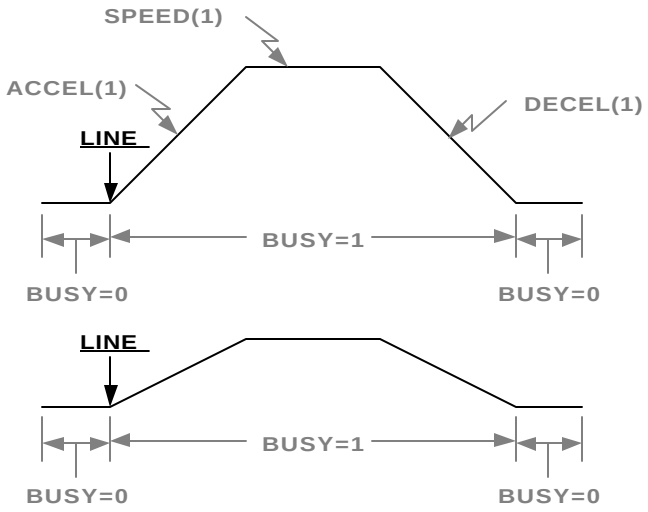
**REMARKS:** The axis specifies the number of the axis (1-8).

The expression represents the move distance. All defined axes will start and end at the same time.

The lower numbered axis is considered the master and its parameters: SPEED, ACCEL,.DECEL, and PROFILE are used.

**LINE COMMAND CYCLE**

**LINE = 4,2**




**Note: The underlined text is the command required to generate the velocity profile. The remaining text are related commands.**

The individual axis velocity, acceleration and deceleration calculations are based on the move distance in units. The velocity, acceleration and deceleration vales for each axis will be a ratio of the master distance (axis 1) to the individual axis distances, (ratio=axis distance / master distance)

**EXAMPLES:**

```
LINE=1.0,,-2.0          'Linear interpolated axis 1 and 3. Axis 1 moves
                         +1.0 units and axis 3 moves -2.0 units.
WAITDONE(1,3)

LINE(1,3)=1.0,-2.0      'Linear interpolated axis 1 and 3. Axis 1 moves
                         +1.0 units and axis 3 moves -2.0 units.
WAITDONE(1,3)

PATH=1,2
   LINE=1,-2.0          ' linear interpolates axis 1 and axis 2
```

statements
PATH END

# LOF — Miscellaneous Command

**ACTION:** Returns the number of character in the designated RS232 port.

**PROGRAM SYNTAX:** LOF(port) - used in an expression.

**REMARKS:** Port is the designated serial port (1 or 2). Port 1 is the Host port and port 2 is the Auxiliary port.

**EXAMPLES:**

```
DO : LOOP UNTIL LOF(2)>=10      ›wait for 10 characters in auxiliary port
     A$=@A       'clear A$
     cnt=0
DO
     A$=A$+CHR$(INCHAR(2))  › load characters
LOOP UNTIL LOF(2)=0
```

# LOG — Mathematics Function

**ACTION:** Returns the natural logarithm of a numeric expression.

**PROGRAM SYNTAX:** LOG(expression) - used in an expression

**REMARKS:** The argument expression must be greater than zero. The natural logarithm is the logarithm to the base **e.** The constant **e** is approximately equal to 2.718282.

You can calculate base 10 logarithm as follows: LOG 10(x)= LOG(x)*.4342945

**EXAMPLES:**

```
x=LOG(2.718282)      ‘ sets x= 1
```

# LOWSPD — Trajectory Parameter

**ACTION:** Sets or returns the Low Speed (starting speed) value of a stepping motor axis.

**PROGRAM SYNTAX:**

```
LOWSPD(axis)=expression
LOWSPD=expression1, ... ,expression 8
LOWSPD(axis, ... ,axis)=expression, ... ,expression
LOWSPD(axis)  - used in an expression
```

**REMARKS:** This command is defined in more detail in Section 10 Stepper Drive.

# MAXSPD — Trajectory Parameter

**ACTION:** Sets or returns the maximum allowed speed of the specified axis.

**PROGRAM SYNTAX:**
MAXSPD(axis)=expression
MAXSPD=expression1, ... , expression 8
MAXSPD(axis, ... ,axis)=expression, ... , expression
MAXSPD(axis) - used in an expression

**REMARKS:** The axis specifies the number of the axis (1-8).

The expression specifies the maximum speed allowed for an axis.

Motion will not be performed at speeds higher than this value, even if an axis is programmed or commanded to do so.

**EXAMPLES:**
MAXSPD(3)=50
Sets the maximum speed for axis 3 to 50 units/second.

MAXSPD=50,,60
Sets the maximum speed for axis 1 to 50 units/second and axis 3 to 60 units/second.

MAXSPD(1,3)=50,60
Sets the maximum speed for axis 1 to 50 units/second and axis 3 to 60 units/second.

# MID$ — String Manipulation

**ACTION:** Returns the designated middle number of characters of a string.

**PROGRAM SYNTAX:** string1$=MID$(string2$,start,number)

**REMARKS:** The start specifies the starting position of the input string string2$.

The number specifies the number of characters to return. If the number is greater than the (length of the string - start position) the string returned is from starting position to the end of the string.

If the string is null then a “”(no characters) will be returned.

**EXAMPLES:**
```
a$="P/N 123AC"
b$=MID$(a$,5,3)     ' sets b$="123"
c$=MID$(a$,5,9)     ' sets c$="123AC"
```

# MOD — Mathematics Function

**ACTION:** Returns the remainder of a number divided by the base.

**PROGRAM SYNTAX:** y=x MOD base

**REMARKS:** The y is the returned remainder.

The x is the number that is divided by the base.

The base is the divisor.

**EXAMPLES:** y=31 MOD 16 ‘ y is set to 15 which is the remainder.

# MOTIONSTATE — Trajectory Parameter

**ACTION:** Returns the follower motion state for an axis.

**PROGRAM SYNTAX:** MOTIONSTATE(axis) - used in an expression.

**REMARKS:** This command is defined in more detail in Section 8 Following.

# MOVE — Motion Parameter

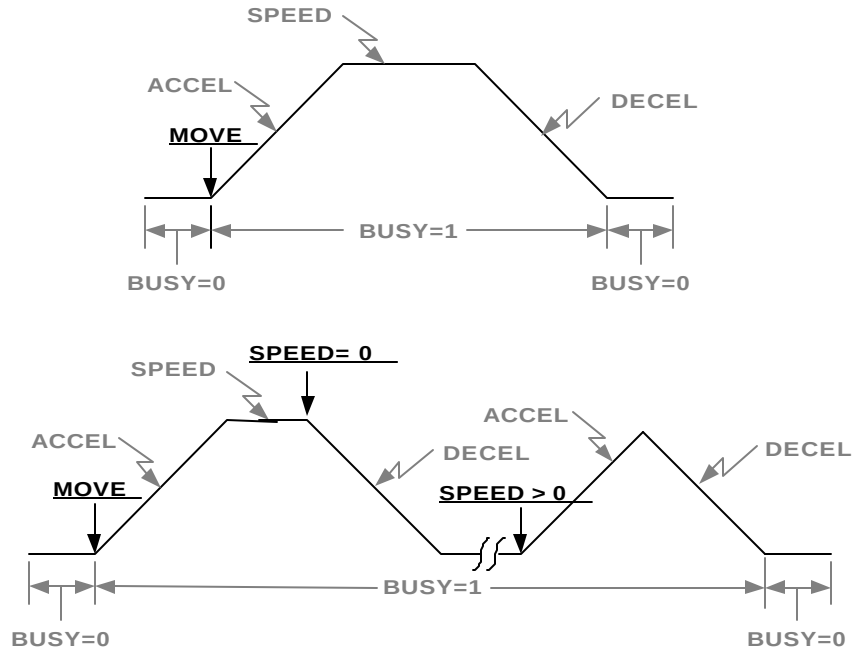
**ACTION:** Initiates a non-coordinated move.

**PROGRAM SYNTAX:**
MOVE(axis)=expression
MOVE=expression1, ... , expression 8
MOVE(axis, ... ,axis)=expression, ... , expression

**REMARKS:** The axis specifies the number of the axis (1-8).

The expression represents the incremental distance or absolute position to be moved to. The POSMODE command setting of an axis determines whether an incremental distance or absolute position is commanded. If the incremental distance, POSMODE(axis)=0, is used the sign of the expression determines the direction (positive or negative) of motion for the move. Incremental position mode is the default.

**MOVE CYCLE**




**Note: The underlined text is the command required to generate the velocity profile. The remaining text are related commands.**

**EXAMPLES:**

```
POSMODE(1,3)=0,0     'incremental position mode for axis 1 & 2
MOVE(3)=-2           'axis 3 moves -2 units
WAITDONE(3)

MOVE=1,,3            'axis 1 moves +1 units and axis 3 moves +3 units.
WAITDONE(1,3)

MOVE(1,3)=1,3        'axis 1 moves +1 units and axis 3 moves +3 units.
```

WAITDONE(1,3)

# MOVEHOME Motion Parameter

**ACTION:**

Runs the motor until the home input is activated, captures and records the position of the switch activation as home (electrical zero), then decelerates the motor to a stop.

**PROGRAM SYNTAX:**

MOVEHOME(axis)=expression
MOVEHOME=expression1, ... , expression 8
MOVEHOME(axis, ... ,axis)=expression, ... , expression

**REMARKS:**

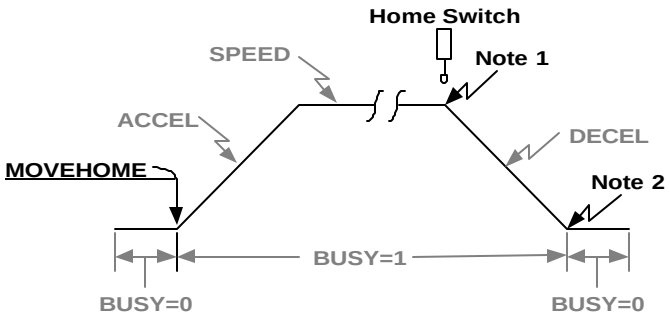
The axis specifies the number of the axis (1-8).

The sign of the expression determines the direction (positive or negative) of the motion for the home cycle. The non-zero value of the number is not significant. The commanded speed of the axis is determined by the last SPEED command that was executed.

The MOVEHOME trigger can be EVENT1 input, EVENT2 input and/or Encoder marker state. This trigger is defined by the user program *Configuration and Setup*, and also by the EVENT1 or EVENT2 commands if they have been executed prior to the MOVEHOME.

Prior to starting a MOVEHOME motion, the appropriate trigger input (EVENT1 or EVENT2) is checked to see if it has already been triggered. If the trigger is already enabled the ABSPOS and ENCPOS are set to zero and no motion occurs. Otherwise, the motor accelerates at the ACCEL rate to the commanded SPEED and continues at this speed until the home trigger condition is met. The capture position is recorded when the home trigger occurs. The motor decelerates to a stop at the DECEL rate. Once at a stop, the distance traveled from the trigger becomes the new ABSPOS and ENCPOS value. The exact position that the motor was at when the trigger occurred becomes the zero position, or home. The captured absolute position can be monitored by the CAPPOS command.

**MOVEHOME CYCLE**




**Note 1: The Home switch activates and the current position is captured.**

**Note 2: Motion is completed and the Absolute and Encoder positions are set to the difference between the captured position and the ending position.**

**Note: The underlined text is the command required to generate the velocity profile. The remaining text are related commands.**

**EXAMPLES:**

```
MOVEHOME(3)=1       'Axis 3 executes a home cycle in the positive
                     direction.
WAITDONE(3)         'Wait for motion to stop.
POSMODE(3)=1        'Activates Absolute Mode for axis 3.
MOVE(3)=0           ' move axis 3 to the captured home position
WAITDONE(3)

MOVEHOME=-2,,3      'Axis 1 executes a home cycle in the negative
                     direction and axis 3 executes a home cycle in
                     the positive direction.
WAITDONE(1,3)
POSMODE(1,3)=1,1    'activates Absolute position mode for axis 1 & 3
MOVE(1,3)=0,0       ' move axis 1 & 3 to the captured home position
WAITDONE(1,3)

MOVEHOME(1,3)=-1,1 'Axis 1 executes a home cycle in the negative
                     direction and axis 3 executes a home cycle in
                     the positive direction.
WAITDONE(1,3)
POSMODE(1,3)=1,1
MOVE(1,3)=0,0       ' move axis 1 & 3 to the captured home position
WAITDONE(1,3)
```

# MOVEREG Motion Parameter

**ACTION:** Runs the motor until the mark registration input is activated; then moves the motor the desired registration distance.

**PROGRAM SYNTAX:**
MOVEREG(axis)=expression
MOVEREG=expression1, ... , expression 8
MOVEREG(axis, ... ,axis)=expression, ... , expression

**REMARKS:** The axis specifies the number of the axis (1-8).

The expression represents the incremental distance to move after a registration trigger has occurred. The sign of the expression determines the direction (positive or negative) of motion for the registration cycle.

The registration trigger can be the EVENT1 input, EVENT2 input and/or Encoder marker state. This trigger is defined in the user program *Configuration and Setup,* and also by the EVENT1 or EVENT2 command if they have been executed prior to the MOVEREG.

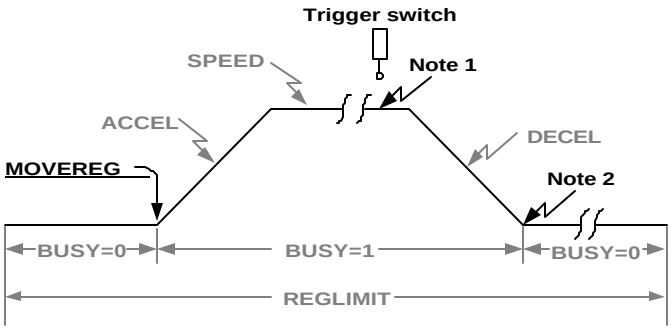
The Registration Travel Limit, which is set by command **REGLIMIT**, limits the distance that the motor will rotate if no trigger occurs. A **REGLIMIT** setting of 0, sets no limit for motor rotation while awaiting a trigger. **THIS IS THE CONDITION AFTER POWER UP OR RESET**. The motor speed during a **MOVEREG** move is set by the **SPEED** command. When the registration trigger occurs, the registration distance is checked to determine if the motion can be stopped in the given distance. If it can not, then the motion will be stopped using the project's *Configuration and Setup* setting for Max. ACCEL, and an error code 7 is set. This error can be eliminated by increasing the registration distance, decreasing the speed or increasing the deceleration.

The captured absolute position can be monitored by the CAPPOS command.

Prior to starting a MOVEREG motion the appropriate trigger input (EVENT1 or EVENT2) is checked to see if it has already been triggered. If the trigger has already occurred, an incremental move of the distance specified by the expression to the right of the MOVEREG will occur.

A **MOVEREG** can be started with its trigger disabled (except for the two encoder index marker selections). The registration trigger may then be enabled later by an EVENT1 or EVENT2 command.

**MOVEREG CYCLE**




**Note 1: The registration input triggers. The distance specified by the command begins to be counted down.**
**Note 2: Motion is completed. The distance traveled from the registration trigger is the command distance.**

**Note: The underlined text is the command required to generate the velocity profile. The remaining text are related commands.**

**EXAMPLE:**

A label is to be put down on top of the material passing by. A sensor connected to Event1 on axis 3 detects when the material leading edge occurs. The label is put down on the material as soon as axis 3 starts motion. When the trailing edge of the label is detected the registration distance is traveled, 5 units. The cycle repeats until input 1 on Digital I/O board 1 is activated.

```
REGLIMIT(3)=10          'set registration limit of axis 3 to 10 units
ACCEL(3)=500            'accel rate of axis 3 set to 500 units/sec^2
DECEL(3)=500            'accel rate of axis 3 set to 500 units/sec^2
start:
DO
    DO
        SPEED(3)=ENCSPD(1) 'set speed of axis 3 equal to the speed of material
    LOOP UNTIL EVENT1(3)=1 ' wait for leading edge of material
    MOVEREG(3)=2        'start laying down label
    WAITDONE(3)         'wait for motion to stop on label roll
LOOP UNTIL IN(101)=1    'repeat cycle if Input 1 on board 1 is inactive
END

ERROR_HANDLER:
        ERR=0,0         ' if an error occurs restart cycle
GOTO start
```

# NOT — Boolean Operator

**ACTION:** The logical NOT operator is used in Boolean expressions.

**PROGRAM SYNTAX:** NOT expression

**REMARKS:** The NOT operator uses the truth table: The result is TRUE if the expression is FALSE.

| expression | condition result |
|---|---|
| TRUE | FALSE |
| FALSE | TRUE |

**EXAMPLES:**

```
DO
    statements
WHILE NOT(DONE(axis))
```

The controller will continue to execute the loop until the axis is done with the motion.

# NVR — Miscellaneous Command

**ACTION:** The NVR array is used for non-volatile variable storage.

**PROGRAM SYNTAX:**

```
NVR(number)
NVR(number)=expression
```

**REMARKS:** The number is the NVR element number being addressed (1-2048). In the MX2000-2-32, (1-32768) NVR elements can be stored.

The expression is the value that will be stored at the specified NVR element.

The NVR array has 2048 elements (1-2048) and is accessible by all program tasks.

To set the NVR element to a default setting use the Host Command SNVR.

**EXAMPLES:**

```
A=NVR(2)
```

Returns the NVR element 2 value to variable A.

```
NVR(2048)=10.5
```

Sets the NVR element 2048 to a value 0f 10.5.

```
NVR(3)=A
```

Sets the NVR element 3 to the value of variable A.

# NVRBIT Miscellaneous Command

**ACTION:** Store or return the bit value in NVR memory.

**PROGRAM SYNTAX:**
NVRBIT(bit)= expression
NVRBIT(bit) - used in an expression

**REMARKS:** The bit value range is 1 - 65536. The expression must be a value of 0 or 1.

When using this command care must be taken not to alter elements used by the NVR and NVRBYTE commands.

The NVR array is used for non-volatile storage. The array consist of 2048 elements, 8192 Bytes or 65536 Bits. Thus, there are 32 bits in each word.

The bit assignments for each 32 bit word is as follows:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | (Word Most Significant Byte) |
| 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | |
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | |
| 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | (Word Least Significant Byte) |

The array element (word) and bit number being addressed is calculated as follows:
element number = ((int) (bit number + 31) / 32)
bit number = mod (bit number / 32)

**EXAMPLES:**
```
NVRBIT(65505)=1    ' sets Bit 1 of element 2048 = 1
NVRBIT(65536)=0    ' sets Bit 32 of element 2048 = 0
```

# NVRBYTE — Miscellaneous Command

**ACTION:** Stores or returns the byte value in NVR memory.

**PROGRAM SYNTAX:**
NVRBYTE(byte)= expression
NVRBYTE(byte) - used in an expression

**REMARKS:** The byte value range is 1 - 8192. The expression must be a value between 0 and 255.

When using this command care must be taken not to alter elements used by the NVR and NVRBIT commands.

The NVR array is used for non-volatile storage. The array consist of 2048 elements, 8192 Bytes or 65536 Bits. Thus, there are 4 bytes in each word.

The array element (word) being addressed is calculated as follows: element= ((int) (number + 3) / 4)
examples: Byte 1 addresses (element 1 Byte 1) MSB
Byte 2 addresses (element 1 Byte 2)
Byte 3 addresses (element 1 Byte 3)
Byte 4 addresses (element 1 Byte 4) LSB
Byte 5 addresses (element 2 Byte 1) MSB

**EXAMPLES:**
NVRBYTE(8192)=255
sets MSB byte = 255 in element 2048

NVRBYTE(8189)=0
sets LSB byte= 0 in element 2048

# OPTION DECLARE — Miscellaneous Command

**ACTION:** This option requires that all local variable be declared as REAL or STRING variables.

**PROGRAM SYNTAX:** OPTION DECLARE

Arrays are not required to be declared since the DIM statement declare them as REALS or STRINGS.

If this option is not used the non-arrayed local variables are not required to be declared but simply used in the program.

**EXAMPLES:**
```
OPTION DECLARE
REAL  a,b,c,d,e,f              ‘ variables are declared
STRING  a$,b$,c$,d$,e$,f$  ‘ variable strings are declared
```

# OR — Boolean Operator

**ACTION:** The logical OR operator is used in Boolean expressions.

**PROGRAM SYNTAX:** expression1 OR expression2

**REMARKS:** The OR operator uses this truth Table: The result is TRUE, if either expression is TRUE.

| Expression1 | Expression2 | Condition Result |
|---|---|---|
| FALSE | FALSE | FALSE |
| FALSE | TRUE | TRUE |
| TRUE | FALSE | TRUE |
| TRUE | TRUE | TRUE |

**EXAMPLES:**

```
DO
        statements
LOOP UNTIL (A>5 OR X=0)
‘ The controller continues to do the loop Until variable A>5 or variable X=0
```

# OUT I/O Function

**ACTION:** Sets or returns the condition of a specified digital output.

**PROGRAM SYNTAX:**
OUT(bnn)=expression
OUT(bnn,len)=expression
OUT(bnn) - used in an expression
OUT(bnn,len) - used in an expression

**REMARKS:** bnn is the board and Output number.

| **board 1** | **board 2** | **board 3** | **board 4** |
|---|---|---|---|
| 101-124 | 201-224 | 301-324 | 401-424 |

The expression specifies the output state's.

The len specifies the number of Output points (1-24) involved in the instruction.

Outputs b01-b16 are physical outputs and b17-b24 are internal flags which can be set or cleared and can be read just like the physical outputs.

OUT(bnn)=expression
Specifies one output state. If the expression is non-zero the output is on. If the expression is zero the output state is off.

OUT(bnn,len)=expression
Specifies multiple output states (len) and the starting output point (bnn). The expression is evaluated and converted to an integer value. The least significant len bits of the binary representation are then used to set outputs bnn to (bnn+len-1) respectively.

OUT(bnn) - used in an expression
Return the state of the specified output point (bnn).

OUT(bnn,len) - used in an expression
Return the state of the designated outputs (bnn to (bnn+len-1)). Evaluates to a number corresponding to the last output commanded (0 or 1) for these output pins. The returned number is the binary weighted sum of the commanded outputs (bnn to(bnn+len-1)) respectively.

**EXAMPLES:**
OUT(107)=1
Digital I/O board 1 output 7 is set to a 1.

OUT(101,6)=48
digital I/O board 1 outputs 1-4 are set to a 0 and outputs 5 and 6 are set to a 1.

x=OUT(107)
A 1 is returned to variable x since output 7 is set to a 1.

x=OUT(101,7)

A 112 is returned to variable x since outputs 1-4 are 0 and output 5-7 are 1.

# OUTLIMIT Servo Parameter

**ACTION:** Sets or returns the servo command voltage limit.

**PROGRAM SYNTAX:** OUTLIMIT(axis)=expression
OUTLIMIT=expression1, ... , expression8
OUTLIMIT(axis, ... , axis)=expression, ... , expression
OUTLIMIT(axis) - used in an expression

**REMARKS:** This command is defined in more detail in Section 9 Servo Drive.

# PATH ...PATH CLOSE ... PATH END Motion Parameter

**ACTION:** Specifies a continuous motion path.

**PROGRAM SYNTAX:**

```
PATH=axis1,axis2
    EXOUT(bnn)=expression
    EXOUT(bnn,len)=expression
    OUT(bnn)=expression
    OUT(bnn,len)=expression
    LINE=axis distance,axis2 distance
    FEEDRATE=expression
    ARC=axis1 center, axis2 center, ±angle
    POINT=axis1 distance, axis2 distance
    RADIUS=expression
    PATH CLOSE
PATH END
```

**REMARKS:** Axis1 and axis2 are the axes used in the path.

The commands listed above are the only commands allowed in a motion path. Path motion (LINE, ARC and POINT) proceeds from one segment to another without stopping. The path speed can be changed with the FEEDRATE command. Outputs can be set in various segments with standard output commands (EXOUT and OUT). When two consecutive segments are lines, then a radius is inserted if the last RADIUS command specified is a non-zero radius.

When Path statements are used in each task, a maximum of 100 points are allowed per PATH ... PATHEND block. Multiple, consecutive PATH 's are allowed within a task. However, motion stops between PATH 's. Up to 700 points may be used to specify a single Path if the only task using the PATH ... PATHEND is task 1 and no other task contains an ARC command.

The PATH CLOSE specifies that the starting points Coordinates are the ending point Coordinates during Path motion.

**EXAMPLES:**

```
PATH=1,2
    LINE=1.5,3
    EXOUT(101)=1
    ARC=3,0,+360
    EXOUT(101)=0
PATH END
```

The above example will move from the present position to position (1.5,3) using the LINE motion. EXOUT(101) is set, a 360 degree ARC is executed and then EXOUT(101) is cleared.

# POINT — Motion Parameter

**ACTION:** Specifies coordinates, which the motors will move through in a path.

**PROGRAM SYNTAX:** POINT=expression1, expression2

**REMARKS:** This command is only valid between a PATH and a PATHEND statement. Expression1 is the first axis coordinate, expression2 is the second axis coordinate. The path connecting points is smooth.

**EXAMPLES:**

```
POSMODE=1,1
PATH=1,2
    POINT=1.5,3
    POINT=4,5
    POINT=6,7
PATH END
```

The above example will move the axes from the present position, through points(1.5,3) and (4,5) to position (6,7) smoothly. The points can be incremental or absolute as set by the POSMODE command.

# POSERR — Trajectory Parameter

**ACTION:** Returns the positional error of the designated axis.

**PROGRAM SYNTAX:** POSERR(axis) - used in an expression

**Note: ENCERR can be substituted for POSERR.**

**REMARK:** The axis specifies the number of the axis (1-8).

Position error is the difference between the absolute position and the encoder position (ABSPOS - ENCPOS).

**EXAMPLE:**

```
X=POSERR(1)
IF  X > 10 THEN
        PRINT#1,"Large Error"
END IF
```

# POSMODE Motion Parameter

**ACTION:** Sets or returns the positioning mode for the specified axis.

**PROGRAM SYNTAX:**

POSMODE(axis)=expression
POSMODE=expression1, ... , expression8
POSMODE(axis, ... , axis)=expression, ... , expression
POSMODE(axis) - used in an expression

**REMARKS:** The axis specifies the number of the axis (1-8).

If the expression is TRUE (non-zero) then the absolute positioning mode is enabled. If the expression is FALSE (zero) then the incremental mode is enabled. Incremental positioning mode is the default mode.

**EXAMPLES:**

POSMODE(2)=1
Sets the positioning mode for axis 2 to absolute.

POSMODE=1,,0
Sets the positioning mode for axis 1 to absolute and axis 3 is set to incremental positioning mode.

POSMODE(1,3)=1,0
Sets the positioning mode for axis 1 to absolute and axis 3 is set to incremental positioning mode.

# PRINT

## String Manipulating

**ACTION:** Transmits designated data via the designated serial port.

**PROGRAM SYNTAX:**
PRINT#1,[expression][, or ;][expression][, or ;]
PRINT#2,[expression][, or ;][expression][, or ;]

**REMARKS:** Port 1 is the Host port and Port 2 is the Auxiliary Port.

**expression** can be an variable, parameter, string variable or Literal string. Literal strings must be enclosed in quotation marks.

If a comma "**,**" is used between expressions five spaces will separate expressions.

If a semicolon "**;**" is used between expressions there will be no space between expressions.

Up to 20 expressions can be used with one PRINT command.

If a semicolon "**;**" is used at the end of the PRINT command, no carriage-return/line-feed sequence will be generated.

**EXAMPLES:**

```
ACCEL(2)=10.5
DECEL(2)=2.1
PRINT#1,"accel(2)= ";ACCEL(2),"decel(2)= ";DECEL(2)
' Host port out  "accel(2)= 10.5     decel(2)= 2.1" <cr> <lf>
```

```
 ACCEL(2)=10.5
DECEL(2)=2.1
PRINT#2,"accel(2)= ";ACCEL(2),"decel(2)= ";DECEL(2)
'Auxiliary port  out  "accel(2)= 10.5     decel(2)= 2.1" <cr> <lf>
```

```
ACCEL(2)=10.5
DECEL(2)=2.1
PRINT#2,"accel(2)= ";ACCEL(2),"decel(2)= ";DECEL(2);
' Auxiliary port  out  "accel(2)= 10.5     decel(2)= 2.1"
```

# PRINT USING — String Manipulation

**ACTION:** Prints strings character or formatted numbers.

**PROGRAM SYNTAX:**

```
PRINT USING #1,"literal string",[exp][, or;][exp][;]
PRINT USING #1,Format$,[exp][, or;][exp][;]
PRINT USING #2,"literal string",[exp][, or;][exp][;]
PRINT USING #2,Format$,[exp][, or;][exp][;]
```

**REMARKS:** Port 1 is the Host Port and Port 2 is the Auxiliary Port.

The numeric values are formatted only using the literal string or a designated Format$ variable string. This string can contain non-format characters that will be printed prior to the formatted number. The following characters in the string will not be printed from the string:
"+" "#" "0" " ." "\" and ",". However, these character can be printable characters by preceding the character with a "\".

**Example:**

requirement to send the following ASCII string with the current state of OUT(101) (Output #1 on board 1 is <state> which is the coolant control)

```
a$="Output \#1 is #
PRINT USING #1,a$,OUT(101); " which is the coolant control"
```

The resulting serial output:
Output #1 is n which is the coolant control
where: n is the state of output (101)

The comma (,) which is the delimiter for expressions, will not print spaces like the PRINT # command. If spaces are required, between expressions, they must be added to the literal string or format$.

**Example:**

```
ACCEL(1)=100
DECEL(1)=200
a$="Acc=0000   Dcc=0000"
PRINT USING#1,a$,ACCEL(1),DECEL(1)
```

The resulting serial output:
Acc= 0100 Dcc= 0200

If the numeric data is larger than the specified format than an ***** will be substituted for the 0's and #'s in the output.

**Example:**

```
ABSPOS(1)=1000.54
a$="Position= +0##.##"
PRINT USING #1,a$,ABSPOS(1)
```

The resulting serial output:
Position= +***.**

The following special characters are used to format the numeric field:

- **+** The sign of the number will always be printed.
- **-** Only the negative sign will be printed. If the data is positive a space will be printed in place of the sign.
- **#** represents each digit position. If no data exist at the digit position substitute a space. The Digit field will always be filled.
- **.** A decimal point may be inserted at any position in the field.
- **0** represents a digit position. If no data exist at the digit position substitute a 0. The Digit field will always be filled.

**Any other character will be printed as encountered.**

**Note: if no sign is used the - sign is assumed.**

The valid formats are:

| **Left side format** | **Comments** |
|---|---|
| +0000 | The sign with leading zero's will be printed. |
| +0000. | The sign with leading zero's and decimal point will be printed. The right side format is optional |
| +#### | The leading spaces with a sign and digits will be printed. |
| +####. | The leading spaces with a sign, digits and decimal point will be printed. The right side format is optional. |
| 0000 | The - sign or a space with leading zero's will be printed. |
| 0000. | The - sign or a space with leading zero's and decimal point will be printed. The right side format is optional. |
| #### | The leading spaces with a -sign or a space and digits will be printed. |
| ####. | The leading spaces with a -sign or a space, digits and decimal point will be printed. The right side format is optional. |
| +. | The sign and decimal point will be printed. This requires the right side format also. |
| . | The sign and decimal point will be printed. This requires the right side format also. |

| **Right side format** | **Comments** |
|---|---|
| 0000 | Prints digits with trailing zer's. |
| #### | Prints digits with trailing spaces |
| 00## | Print two digits with trailing spaces. |

If the expressions are literal strings or variable strings they will be printed as is.

If a semicolon is used at the end of the Print Using command, no carriage-return / line-feed sequence will be generated.

When numeric data is to be printed, the format string is searched from the beginning for a format character (+0#.). The string data up to this position is sent via the serial port. The format characters (+0#.) are now processed and the formatted value is sent via the serial port. When the next numeric data is to be printed, this process continues from the current position in the string. When the end of the format string is encountered and numeric data is to be printed, a default format (PRINT # format) is used. If the format string end is not encountered and the command is complete the remaining characters in the format string will be printed.

**The following example illustrates how the format string is processed.**

**Example:**
PRINT USING#1,"Numbers are +###.## ### 0##",100.54,"mv", 999,"cnts" ,54," is limit"

The **"Numbers are "** is extracted from the string and sent via serial port. The **"+###.##"** is extracted from the string as the data format, which results in "+100.54" being sent via serial port. The string **"mv"** is sent via serial port. The **" "** is extracted from the string and sent via serial port. The **"###"** is extracted from the string as the data format, which results in "999" being sent via serial port. The string **"cnts"** is sent via serial port. The **" "** is extracted from the string and sent via serial port. The **"0##"** is extracted from the string as the data format, which results in "054" being sent via serial port. The string **" is limit"** is sent via serial port. A crlf is appended and sent via serial port.

**Resulting string:**
Numbers are +100.54mv 999cnts 054 is limit<cr><lf>

**EXAMPLES:**

```
        accel(1)=10000
A$=@accel(1)= 000000"
PRINT USING #1, Aaccel(1)= 000000", accel(1)
        accel(1)= 010000 crlf              printed
PRINT USING #1, A$, accel(1)
        accel(1)= 010000 crlf              printed
End

PRINT USING #1, A   +####@, 1234.6, 234
        +1235   + 234 cr lf                printed
PRINT USING #1, A   +0000@, 1234.6, 234
        +1235   +0234 cr lf                printed

PRINT USING #1, A   +####.###@, 1234.6, 234
        +1234.6     + 234.   cr lf         printed
PRINT USING #1, A   +0000.000@, 1234.6, 234
        +1234.600   +0234.000 cr lf        printed
PRINT USING #1, A   ###+.000", 23.45, 22.3515
        +23.450   +22.352 cr lf            printed
```

# PROFILE — Trajectory Parameter

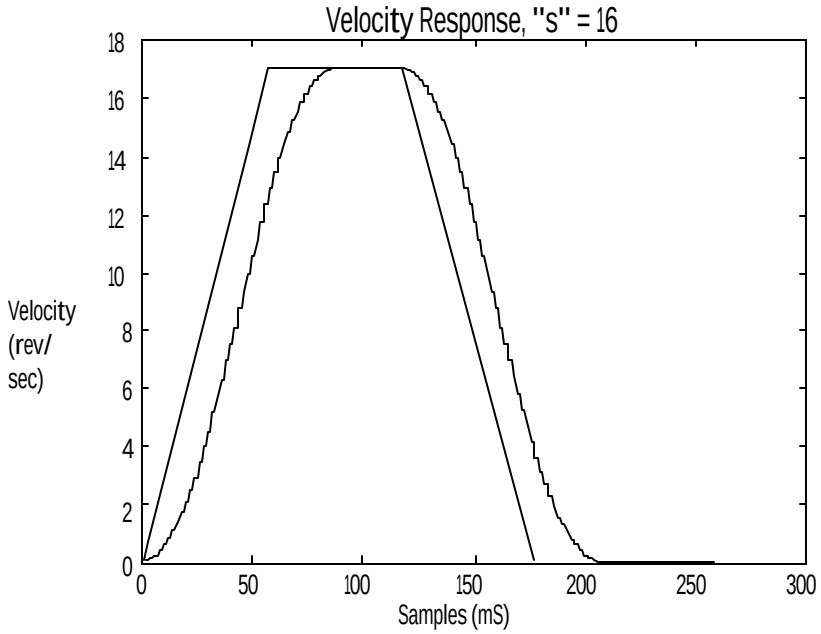
**ACTION:** Determines how the motor's speed changes.

**PROGRAM SYNTAX:**
PROFILE(axis)=expression
PROFILE=expression1, ... , expression8
PROFILE(axis, ... , axis)=expression, ... ,expression
PROFILE(axis) - used in an expression

**REMARKS:** The axis specifies the axis number (1-8).

The expression specifies the profile setting (1-32).

Speed changes require a period of accel/decel to increase/decrease the motor's speed. The Profile value determines how the accel/decel is applied. The MX controller has 32 choices. A profile setting of 1 results in a "Trapezoidal" profile. This yields the minimum move time. Settings 2-32 yields "S-curve" profiles with varying degrees of "S". The higher the profile setting, the more "S" like the profile. Move times with profile settings 1-32 are from 1 to 31 ms longer respectively than those with a setting of 1. The "S-curve" profiles usually results in smoother motion at the expense of longer move times.




**EXAMPLES:** PROFILE(1,3)=16,32
axis 1 profile is set to a value of 16 and axis 3 profile is set to 32.

PROFILE(2)=10
axis 2 profile is set to a value of 10.

PROFILE=16,,32
axis 1 profile is set to a value of 16 and axis 3 profile is set to 32.

# RADIUS — Motion Parameter

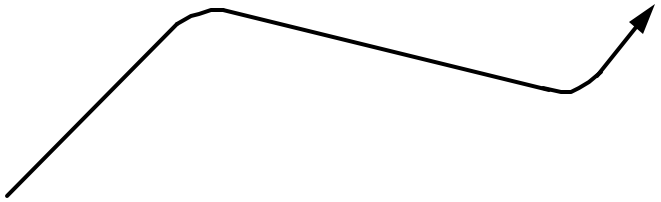
**ACTION:** Sets or returns the ARC radius for Path blending.

**PROGRAM SYNTAX:**
```
RADIUS = expression
RADIUS  -  used in an expression
```

**REMARKS:** Blending only occurs between lines in a path.

The first syntax type sets the ARC radius for Path blending equal to the expression. The second syntax type (expression = RADIUS) returns the current value of Radius.

**EXAMPLES:**
```
X=RADIUS     'sets X equal to the current RADIUS value

RADIUS=.25  'sets the RADIUS for Path blending to .25 units

PATH=1,2
    RADIUS=.25
    LINE=1, 1
    LINE=2, -.5     'bending occurs
    LINE=.5, .5     'bending occurs
PATH END
```



# READ — Miscellaneous Command

**ACTION:** Reads numbers from data statements and assigns them to the variables in the list.

**PROGRAM SYNTAX:**
```
READ variable, variable, etc
```

**REMARKS:** All numbers in the data statements are floating point numbers.

The data statements are contained in the BASIC program. Refer to the DATA statement description for more detail.

The DATA statement must always appear ahead of the READ statement.

**EXAMPLES:**
```
DATA 1,2,3,4
READ a, b, c, d      'reads next four values from the data statement into
                      variables a, b, c and d
RESTORE
```

# REDUCE — Motion Parameter

**ACTION:** Enables, disables the Reduce current or returns the enable status.

**PROGRAM SYNTAX:**
REDUCE(axis)=expression
REDUCE=expression1, ... , expression8
REDUCE(axis, ... , axis)=expression, ... , expression
REDUCE(axis) - used in an expression

**REMARKS:** This command is defined in more detail in the Stepper Section of the Manual.

# REGLIMIT — Over Travel Limit

**ACTION:** Sets or returns the distance to be moved during a MOVEREG cycle, while awaiting a trigger.

**PROGRAM SYNTAX:**
REGLIMIT(axis)=expression
REGLIMIT=expression1, ... , expression8
REGLIMIT(axis, ... , axis)=expression, ... , expression
REGLIMIT(axis) - used in an expression

**REMARKS:** The axis specifies the axis number (1-8)

The expression set the registration travel distance limit for the specified axis. A value of 0 disables the registration travel distance limit.

If no trigger occurs, a MOVEREG cycle behaves like an incremental MOVE cycle, with the distance specified by REGLIMIT. **REGLIMIT must be set prior to a MOVEREG cycle.**

**EXAMPLES:**
REGLIMIT(2)= 10
set the MOVEREG travel distance limit on axis 2 to 10 units

REGLIMIT=0,,10
disables the REGLIMIT for axis 1 and axis 3 has MOVEREG travel distance limit of 10 units.

REGLIMIT(1,3)=0,10
disables the REGLIMIT for axis 1 and axis 3 has MOVEREG limit of 10 units.

## REM or ‘ Miscellaneous Command

**ACTION:** Allows source code comments to be inserted in the program.

**PROGRAM SYNTAX:** REM or '

**REMARKS:** All text to the right of REM or ' to the end of the line is not considered part of the command during execution.

**EXAMPLES:**
```
ACCEL(1)=10.2     REM axis 1 acceleration=10.2 units
DECEL(1)=5.4      ' axis 1 deceleration=5.4 units
```

## RESET Miscellaneous Command

**ACTION:** Resets the controller system.

**PROGRAM SYNTAX:** RESET

**REMARKS:** This command causes the system to halt, and then restart as though power had been recycled. This command can be used to start a different project , as selected by the SEL1 SEL2 and SEL4 inputs on the DSP Controller card.

A hardware input reset can also be configured in the I/O folder of the *Configuration and setup*.

**EXAMPLES:** RESET

## RESTORE Miscellaneous Command

**ACTION:** Allows DATA statements to be read again.

**PROGRAM SYNTAX:**
```
RESTORE
RESTORE(number)
```

**REMARKS:** Sets the pointer for DATA statements to the start (0) or to a designated position (number).

**EXAMPLES:**
```
RESTORE(10)
```
Sets the pointer for DATA statements to position 10, the first variable in the next READ statement will be loaded with element 10

```
DATA 1,2,3,4
READ a, b, c, d    'reads next four values from the data statement into
                    variables a, b, c and d
RESTORE
```

# RIGHT$ String Manipulation

**ACTION:** Returns the rightmost characters of a string.

**PROGRAM SYNTAX:** string1$=RIGHT$(string2$,n)

**REMARKS:** The n is the number of rightmost characters to return. If n is greater than the length of the string2$ then the entire string is returned to string1$.

**EXAMPLES:**
b$="Hello World"
a$=RIGHT$(b$,4) 'sets a$="orld".

# SETCOM Miscellaneous Command

**ACTION:** Sets the baud rate and data format for Auxiliary serial port.

**PROGRAM SYNTAX:** SETCOM#n, baud, parity, data, stop

**REMARKS:** The variable "n" signifies the port number. Presently, only the second serial port (the Auxiliary Port) is supported, therefore only a value of 2 is valid for "n".

The baud rate can be any value up to 38,400.
parity setting:
0 no parity
1 odd parity
2 even parity
data
7 7 bit data length
8 8 bit data length
stop
1 1 stop bit
2 2 stop bits

If the inputs are outside the above setting the command will be ignored and an error warning will be issued.

**EXAMPLES:**
SETCOM#2,9600,0,8,1
Sets Auxiliary port to 9600 baud, no parity, 8 bit data and 1 stop bit.

# SHIFT — Miscellaneous Command

**ACTION:** Shifts the elements of a single-dimension numeric array up or down.

**PROGRAM SYNTAX:** SHIFT (array, n)

**REMARKS:** **n** is the number of shifts to perform on the array. If n is a positive number, the array is shifted up and the top elements are discarded. If n is a negative number the array is shifted down and the bottom elements are discarded. Zeroes are shifted into the array.

**EXAMPLES:** This example illustrates the effect of shift commands on a 4-element array "x".

| x(0) | x(1) | x(2) | x(3) | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | x before shift command |
| 0 | 1 | 2 | 3 | x after SHIFT(x,1) |
| 2 | 3 | 4 | 0 | x after SHIFT(x, -1) |

# SIGN — Mathematics Function

**ACTION:** Returns the sign of an expression.

**PROGRAM SYNTAX:** SIGN(expression) - used in an expression

**REMARKS:** If the expression is positive, the SIGN function returns 1.

If the expression is zero, the SIGN function returns 0.

If the expression is negative, the SIGN function returns -1.

**EXAMPLES:**
SIGN(-10.0) ‘ evaluates to -1
SIGN(10) 'evaluates to 1
SIGN(0) ‘ evaluates to a 0

# SIN — Mathematics Function

**ACTION:** Returns the sine of the angle x, where x is in radians.

**PROGRAM SYNTAX:** SIN(x) - used in an expression

**REMARKS:** To convert values from degrees to radians, multiply the angle (in degrees) times Pi/180 (or .017453) where Pi= 3.141593.

To convert a radian value to degrees, multiply it by 180/Pi (or 57.295779).

**EXAMPLES:**
conv = 3.141593 / 180'converts degrees to radians
A = SIN (conv * 45)’ A = sin (45 degrees ) or .7071

# SOFTLIMIT Over Travel Limit

**ACTION:** Enables/disables or returns the SOFTLIMIT enable state for the selected axis.

**PROGRAM SYNTAX:**
SOFTLIMIT(axis)=expression
SOFTLIMIT=expression1, ... , expression8
SOFTLIMIT(axis, ... , axis)=expression, ... ,expression
SOFTLIMIT(axis) - used in an expression

**REMARKS:** axis selects the designated axis (1-8).

The expression sets the SOFTLIMIT state of the designated axes. A "0" disables the SOFTLIMPOS and SOFTLIMNEG soft limits of the designated axis. Any other value will enable the SOFTLIMPOS and SOFTLIMNEG soft limits of the designated axis.

**EXAMPLES:**
SOFTLIMIT(2)=0
Disables the SOFTLIMPOS and SOFTLIMNEG soft limits of axis 2.

SOFTLIMIT=1,,0
Enables the SOFTLIMPOS and SOFTLIMNEG soft limits of axis 1 and disables the axis 3 soft limits.

SOFTLIMIT(1,3)=1,0
Enables the SOFTLIMPOS and SOFTLIMNEG soft limits of axis 1 and disables the axis 3 soft limits.

# SOFTLIMNEG Over Travel Limit

**ACTION:**

Programmable **software limit switch** for motion in the negative direction. Sets or returns the absolute negative travel position value for the specified axis.

**PROGRAM SYNTAX:**

SOFTLIMNEG(axis)=expression
SOFTLIMNEG=expression1, ... ,expression8
SOFTLIMNEG(axis, ... , axis)=expression, ... , expression
SOFTLIMNEG(axis) - used in an expression

**REMARKS:**

The "axis" specifies the number of the axis (1-8).

The expression sets the absolute value for the negative direction soft limit in units.

If during motion the absolute position becomes less than its software limit value, the motion is aborted.

Software travel limits are used to stop the motor when the commanded position exceeds the programmed software travel limit. There are two software travel limits, one for + and one for - motor rotation. The + software travel limit is tested when the motor is rotating in the + direction. The - software travel limit is tested when the motor is rotating in the - direction.

The software travel limits are checked if they are enabled and a motion other than MOVEHOME is occurring.

The software travel limits power up disabled.

When the travel limit is exceeded, the motor is decelerated to a stop using the Max. ACCEL value, and an error code is set.

**EXAMPLES:**

SOFTLIMNEG(2) =-4
Sets the negative direction soft limit of axis 2 at -4 units.

SOFTLIMNEG=-5,,-6
Sets the negative direction soft limit of axis 1 at -5 units and axis 3 is set to -6 units.

SOFTLIMNEG(1,3)=-5,-6
Sets the negative direction soft limit of axis 1 at -5 units and axis 3 is set to -6 units.

# SOFTLIMPOS Over Travel Limit

**ACTION:**

Programmable **software limit switch** for motion in the positive direction. Sets or returns the absolute positive travel position value for the specified axis.

**PROGRAM SYNTAX:**

SOFTLIMPOS(axis)=expression
SOFTLIMPOS=expression1, ... ,expression8
SOFTLIMPOS(axis, ... , axis)=expression, ... , expression
SOFTLIMPOS(axis) - used in an expression

**REMARKS:**

The "axis" specifies the number of the axis (1-8).

The expression sets the value for the positive direction soft limit in units.

If during motion the absolute position becomes greater than its limit, the motion is aborted.

Software travel limits are used to stop the motor when the commanded position exceeds the programmed software travel limit. There are two software travel limits, one for + and one for - motor rotation. The + software travel limit is tested when the motor is rotating in the + direction. The - software travel limit is tested when the motor is rotating in the - direction.

The software travel limits are checked if they are enabled and a motion other than MOVEHOME is occurring.

The software travel limits power up disabled.

When the travel limit is exceeded, the motor is decelerated to a stop using the Max. ACCEL value, and an error code is set.

**EXAMPLES:**

SOFTLIMPOS(2) =4
Sets the positive direction soft limit of axis 2 at +4 units.

SOFTLIMPOS=5,,6
Sets the positive direction soft limit of axis 1 at +5 units and axis 3 is set to +6 units.

SOFTLIMPOS(1,3)=-5,-6
Sets the positive direction soft limit of axis 1 at +5 units and axis 3 is set to +6 units.

# SPEED Trajectory Parameter

**ACTION:** Sets and returns the target velocity of the motor.

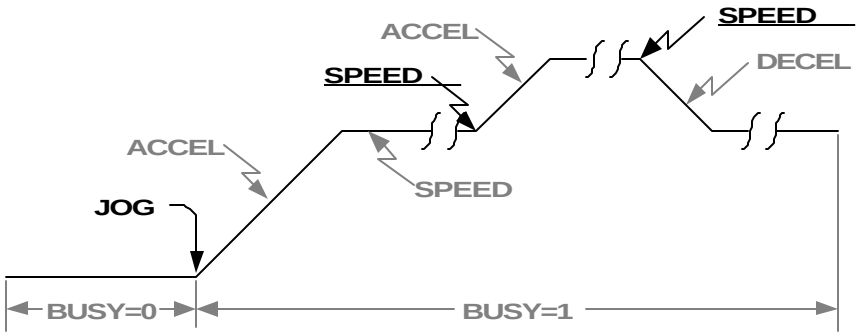
**PROGRAM SYNTAX:**
SPEED(AXIS)=expression
SPEED=expression1, ... , expression8
SPEED(axis, ... ,axis)=expression, ... , expression
SPEED(axis) - used in an expression

**REMARKS:** The axis specifies the number of the axis (1-8).

The expression sets the speed of the designated axis in units/second and must be a positive value.

The velocity of an axis can be changed during motion by issuing a new value for the SPEED command. The velocity change will use the ACCEL or DECEL rate change value. A SPEED of 0 will stop the motor but the cycle will remain busy. To resume the cycle simply change the SPEED value from 0 and the cycle will resume motion.

**SPEED Change During Motion**




**Note: The underlined text is the command required to generate the velocity profile. The remaining text are related commands.**

**EXAMPLES:**
SPEED(2)=10
Sets the speed of axis 2 to 10 units/second.

SPEED=0,,5
Sets the speed of axis 1 to 0 units/second and axis 3 to 5 units/second.

SPEED(1,3)=0,5
Sets the speed of axis 1 to 0 units/second and axis 3 to 5 units/second.

# SQRT — Mathematics Function

**ACTION:** Returns the square root of the expression.

**PROGRAM SYNTAX:** SQRT(expression) - used in an expression

**REMARKS:** The expression must greater than or equal to zero, or an warning will occur.

**EXAMPLES:** x = SQRT(16)
Sets variable x equal to a value of 4.

# STOP — Motion Parameter

**ACTION:** Stops any motion with a control stop, uses the DECEL value for decelerating to a stop.

**PROGRAM SYNTAX:** STOP(axis)
STOP=expression1 , ... , expression8
STOP(axis, ... ,axis)

**note: JOGSTOP can be substituted for STOP.**

**REMARKS:** The axis specifies the number of the axis (1-8).

Any value for the expression will stop the designated axis.

This command will stop any motion using the DECEL value for normal motion and FOLDCCDIST for following motion.

**EXAMPLES:** STOP(2)
requests following axis 2 to stop.

STOP=1,,1
requests following axis 1 and axis 3 to stop.

STOP(1,3)
requests following axis 1 and axis 3 to stop.

# STOPERR — Motion Parameter

**ACTION:** Sets or returns the maximum position error allowed when motion is stopped, referred to herein as "position error band."

**COMMAND SYNTAX:** STOPERR(axis) =expression
STOPERR=expression1, … , expression8
STOPERR(axis, … , axis)=expression, … , expression
STOPERR(axis) - Used in an expression

**Note: ENCBAND can be substituted for STOPERR.**

**REMARKS:** STOPERR is a stepper drive and servo drive parameter.

STOPERR is defined in detail in both the Servo Drive and Stepper Drive Sections of this manual.

# STR$ String Manipulation

**ACTION:** Returns a string representation of a numeric expression.

**PROGRAM SYNTAX:** String1$=STR$(numeric expression)

**REMARKS:** The STR$ command is the complement of a VAL command.

**EXAMPLES:**

```
ACCEL(2)=100
x=50
y=2.1
a$=STR$(ACCEL(2))      'sets a$="100"
b$=STR$(x)             'sets b$="50"
c$=STR$(y)             'sets c$="2.1"
```

# STRING$ String Manipulation

**ACTION:** Returns a string of characters.

**PROGRAM SYNTAX:** String1$=STRING$(number, code)

**REMARKS:** The number indicates the length of the string to return.

The code is the ASCII code of the character to use to build the string.

**EXAMPLES:**

```
a$ = STRING$(10,63)    'sets a$="??????????"
```

# TAN Mathematics Function

**ACTION:** Returns the tangent of the angle x, where x is in radians.

**PROGRAM SYNTAX:** TAN(x) - used in an expression

**REMARKS:** To convert values from degrees to radians, multiply the angle (in degrees) times Pi/180 (or .017453) where Pi= 3.141593.

To convert a radian value to degrees, multiply it by 180/Pi (or 57.295779).

**EXAMPLES:**

```
PI = 3.141593
'assign the constant "PI"
x = TAN (PI/4)
'calculate tangent of 45 degrees, sets x equal to the value 1.0,  which is the tangent of 45 degrees
```

# TIMER — Time Function

**ACTION:** Sets or read the Timer value in seconds.

**PROGRAM SYNTAX:**

```
TIMER - used in an expression
TIMER=expression
TIMER1 - used in an expression
TIMER1=expression
```

**REMARKS:** The Timer should be set to a value at the beginning of the task in which it is used.

The Timer is incremented every millisecond.

**EXAMPLES:**

```
TIMER=0                 'set the timer to zero
DO
   statements
WHILE TIMER < 1.0     ' do the loop for 1 second
```

# TIMER2 — Time Function

**ACTION:** Sets or read the Timer2 value in seconds.

**PROGRAM SYNTAX:**

```
TIMER2 - used in an expression
TIMER2=expression
```

**REMARKS:** The Timer should be set to a value at the beginning of the task in which it is used.

The Timer is incremented every millisecond.

**EXAMPLES:**

```
TIMER2=0                 'set the timer to zero
DO
   statements
WHILE TIMER2 < 1.0     ' do the loop for 1 second
```

# TOLERANCE — Miscellaneous Command

**ACTION:** Sets a tolerance on a numeric comparison.

**SYNTAX:**
TOLERANCE = expression
TOLERANCE - used in an expression

**REMARKS:** Sets a numeric tolerance for all comparison operators (<, <=, = >=, >, <>). If the comparison is within the bounds of a ± tolerance the comparison is true.

**EXAMPLES:**

```
TOLERANCE=.001
IF x <= 2 THEN            ' if x <= +1.999 then comparison is true
    statements
ELSE IF x >= 4 THEN   ' if  x >= +3.999 then comparison is true
    statements
ELSE IF x = 3 THEN      ' if x is 2.999 to 3.001 the comparison is
                          true
    statements
ELSE IF x <>3 THEN      ' if x  < 2.999 or >3.001 the comparison is
                          true
    statements
END IF
```

# UCASE$ — String Manipulation

**ACTION:** Returns a string with all letters converted to upper case.

**PROGRAM SYNTAX:** string1$=ucase$(string2$)

**REMARKS:** String2$ is copied and all lower-case letters are converted to upper case.

This command is useful for making the INSTR command case insensitive.

**EXAMPLES:**

```
a$="hello"
b$=UCASE(a$)     ' b$="HELLO"
```

# VAL — String Manipulation

**ACTION:** Returns the floating point value of the designated string variable.

**PROGRAM SYNTAX:** VAL(n$) - used in an expression

**REMARKS:** n$ is the designated string variable.

The string variable format for conversion is: [sign]digits[.digits[e or E[sign]integer]

The sign and scientific notions are optional.

Only numeric values are returned. The first character that cannot be part of the number terminates the string. If no digits have been processed, a value of zero is returned.

**EXAMPLES:**

```
a$="134 Main St"
b$="10.55 dollars"
x=VAL(a$)          ' sets x=134
y=VAL(b$)    ' sets y=10.55
```

# VELOCITY — Trajectory Parameter

**ACTION:** Sets or returns the path speed to be used for coordinated motion.

**PROGRAM SYNTAX:** VELOCITY = expression
VELOCITY - used in an expression

**REMARKS:** This velocity is used in the LINE, ARC, and PATH commands.

**EXAMPLES:** VELOCITY=10.1
Sets the coordinated velocity for a LINE or ARC command to 10.1 units/sec

k1 = VELOCITY
Sets the variable k1 equal to the value of VELOCITY used in the program

# WAIT — Time Function

**ACTION:** Waits for the period of time (expressed in seconds) to expire before continuing.

**PROGRAM SYNTAX:** WAIT = expression

**REMARKS:** The expression defines the wait delay in seconds.

Program execution is suspended until the desired time has elapsed.

**EXAMPLES:** WAIT = 1.1

Wait 1.1 seconds and then continue

# WAITDONE Motion Parameter

**ACTION:** Waits for motion to be done for the specified axes. "Done" means motion is complete.

**PROGRAM SYNTAX:**
```
WAITDONE(axis)
WAITDONE=expession1, ... , expression8
WAITDONE(axis, .... ,axis)
```

**REMARKS:** The "axis" specifies the number of the axis (1-8).

The expression specifies the axis to wait for motion complete.

An alternate way to accomplish the WAITDONE function is as follows:

```
DO:  LOOP WHILE BUSY(1)
        ' Waits until axis 1 motion is completed.
```

**EXAMPLES:**
```
WAITDONE(3)
' Waits for axis 3 motion to be complete before continuing program execution

WAITDONE (1,2,4,5,6,7,8)
' Waits for axis 1, 2, 4, 5, 6, 7, and 8 motion to be complete before continuing program execution.
```

# WARNING — Miscellaneous Command

**ACTION:** Returns the warning number of the task.

**PROGRAM SYNTAX:** WARNING - used in expression

**REMARKS:** A non-zero indicates no warnings have been encountered in the task.

The predefined Warning codes are

| | |
|---|---|
| 11 | Command axis is not in task group. |
| 12 | Analog I/O selected is out of range. |
| 13 | BCD selected is out of range. |
| 14 | Expansion Input selected is out of range. |
| 15 | Expansion Output selected is out of range. |
| 16 | Digital Input selected is out of range. |
| 17 | Digital Output selected is out of range. |
| 18 | Log command argument is zero or negative |
| 19 | SQRT command argument is negative. |
| 20 | NVR element is out of range. |
| 21 | READ command is out of data arguments. |
| 22 | MAXSPD command is out of range. |
| 23 | Motion occurring at program end. |
| 24 | RS232 Configuration Error. |
| 25 | Servo Parameter is out of range. |

**EXAMPLES:**

```
IF WARNING > 0 then        ' warning occurred?
   Warn = Warning          ' set Warn to WARNING number
END IF
```

# WNDGS — Motion Parameter

**ACTION:** Enables or disable a motor drive.

**PROGRAM SYNTAX:**

```
WNDGS(axis)=expression
WNDGS=expression1, ... ,expression8
WNDGS(axis, ... , axis)=expression, ... , expression
WNDGS(axis) - used in an expression
```

**REMARKS:** This command is defined in detail in both Section 9 Servo Drive and Section 10 Stepper Drive.

## 7.5 Host Commands Grouped By Functions

| **Motion Parameter continued** | | **Page** |
|---|---|---|
| ENCFOL | Sets or returns the maximum position error allowed during motion. | 181 |
| ENCMODE | Sets or returns the operating mode of a closed loop stepper axis. | 181 |
| ENCRES | Returns the encoder line count of an axis. | 182 |
| EVENT1 | Returns the state of the trigger input labeled EVNT1 or sets the trigger polarity and enable , which are used in a MOVEHOME, MOVEREG or FOLMOVREG cycle. | 186 |
| EVENT2 | Returns the state of the trigger input labeled EVNT2 or sets the trigger polarity and enable , which are used in a MOVEHOME, MOVEREG or FOLMOVREG cycle. | 187 |
| FOLERR | Sets or returns the maximum position error allowed during motion. | 189 |
| JOG | Runs the motor continuously in the specified direction. | 192 |
| JOGSTART | Runs the motor continuously in the specified direction. | 192 |
| LINE | Initiates a coordinated linear move involving up to 8 axes. | 195 |
| MOVE | Initiates a non-coordinated move. | 196 |
| MOVEHOME | Runs the motor until the home input is activated, captures and records the position of the switch activation as home. | 197 |
| MOVEREG | Runs the motor until the mark registration input is activated; then moves the motor the desired registration distance. | 197 |
| POSMODE | Sets or returns the position mode of an axis. | 200 |
| STOP | Stops any motion with a control stop. | 204 |
| STOPERR | Sets or returns the maximum position error allowed at standstill. | 205 |
| WNDGS | Enables or disables a motor drive. | 207 |
| **Over Travel Limit** | | |
| HARDLIMNEG | Returns the - Limit hardware state of an axis. | 190 |
| HARDLIMPOS | Returns the + Limit hardware state of an axis. | 190 |
| REGLIMIT | Sets or returns the distance to be moved during a MOVEREG cycle, while awaiting a trigger. 201 | |
| SOFTLIMNEG | Sets or return the - direction software travel limit. | 203 |
| SOFTLIMPOS | Sets or return the + direction software travel limit. | 203 |
| **Servo Parameter** | | |
| INTLIM | Sets the integral limit for a servo axis. | 191 |
| KAFF | Sets or returns the acceleration feed forward gain of a servo axis. | 192 |
| KD | Sets or returns the derivative gain of a servo axis. | 193 |
| KI | Sets or returns the integral gain of a servo axis. | 193 |
| KP | Sets or returns the proportional gain of a servo axis. | 194 |
| KVFF | Sets or returns the velocity feed forward gain of a servo axis. | 194 |
| OUTLIMIT | Sets or returns the servo axis command limit voltage. | 199 |
| **Trajectory Parameter** | | |
| ABSPOS | Sets or returns the commanded absolute position of an axis. | 173 |
| ACCEL | Sets or returns the acceleration value of the motor. | 173 |
| DECEL | Sets or returns the deceleration value of an axis. | 178 |
| ENCERR | Returns the positional error of the designated axis. | 180 |
| ENCPOS | Returns the encoder position of an axis. | 182 |
| ENCSPD | Returns the current encoder speed in Units/second. | 182 |
| LOWSPD | Sets or returns the Low Speed (starting speed) value of a stepping motor axis. | 195 |
| MAXSPD | Sets or returns the maximum allowed speed of an axis. | 196 |
| POSERR | Returns the positional error of the designated axis. | 199 |
| PROFILE | Determines how the motor speed changes. | 200 |

| | | |
|---|---|---|
| SPEED | Sets or returns the target velocity of an axis. | 204 |
| VELOCITY | Sets or returns the path speed to be used for coordinated motion. | 206 |

## 7.6 Host Command Summary (alphabetical list)

FOLMOVREG cycle. 186

## 7.7 Host Commands - Alphabetical Listing

### " <n "

### Miscellaneous Command

**ACTION:** This command activates/deactivates a controller from accepting commands from a host computer.

**COMMAND SYNTAX:** <n or <n cr
<n?
<0 or <0 cr

**REMARKS:** In order to daisy chain multiple controllers to communicate with a single host, each controller must be given a unique identification number. The Unit ID # selector switch defines the identification number of the control. This switch is interrogated on power turn on only. The factory setting is device 1.

**Each Controller must be given a unique identification (1-9) before the system is wired.**

In order to accept commands from a host device, a Control must be set to the active mode. To do this, the host must send the device attention command (<) followed by the device identification followed by a carriage return, line feed or non-numeric character. If n matches the controller id number, that unit becomes the active controller.

If the host requires an acknowledgement of the active controller the <n? command is transmitted by the host and if the device exists it will respond with its id number.

If all controllers are to be placed in the listen mode the host issues a <0cr command. No data can be transferred from the Control to the host in this mode. However, all other commands will be honored by the controllers.

### " ? "

### Miscellaneous Command

**ACTION:** The ? key (or ? character code, ASCII 63, sent via a serial port) requests the space remaining in the Host Receiver buffer.

**COMMAND SYNTAX:** ?

**REMARKS:** The controller receiver buffer is 255 characters long.

# ABSPOS — Trajectory Parameter

**ACTION:** Sets or returns the commanded absolute position of an axis.

**COMMAND SYNTAX:**
ABSPOS(axis)=number cr
ABSPOS=number1, . . . , number8 cr
ABSPOS cr
ABSPOS(axis) cr

**REMARKS:** See Programming Command **ABSPOS.**

**EXAMPLES:**
ABSPOS(3)=2
Sets the absolute position of axis 3 to 2 units.

ABSPOS=1,,3
sets the absolute position of axis 1 to 1 unit, axis 2 no change and axis 3 to 3 units.

ABSPOS(3)
Returns the current absolute position of axis 3.

# ACCEL — Trajectory Parameter

**ACTION:** Sets or returns the acceleration value of an axis.

**COMMAND SYNTAX:**
ACCEL(axis)=number cr
ACCEL=number1, . . . , number8 cr
ACCEL(axis) cr
ACCEL cr

**REMARKS:** See Programming Command **ACCEL.**

**EXAMPLES:**
ACCEL(3)=200
sets the acceleration of axis 3 to 200 units/sec$^2$.

ACCEL=100,,200
sets the acceleration rate of axis 1 to 100 units/sec$^2$, axis 2 no change and axis 3 to 200 units/sec$^2$.

ACCEL(3)
Returns the current acceleration rate for axis 3

# ANALOG — I/O Function

**ACTION:** Sets or returns a numeric value representing the voltage on the analog port.

**PROGRAM SYNTAX:**
ANALOG(b0n) cr
ANALOG(b0n)=number cr

**REMARKS:** See Programming Command **ANALOG**

**EXAMPLES:**
ANALOG(102)=2.5
Sets the voltage on board 1 output 2 to 2.5 volts

ANALOG(102)
Return the current voltage on board 1 input 2.

# ARC — Motion Parameter

**ACTION:** Initiates a coordinated motion to move in an arc.

**PROGRAM SYNTAX:** ARC = x, y, xcenter, ycenter, ±angle

**REMARKS:** See Programming Command **ARC**

**EXAMPLES:**
ARC=1,2,3,0,+180
‘ Initiates a 180° clockwise arc rotation, using axis 1 and 2, with a 3 unit radius.

# AXISBRD — Miscellaneous Command

**ACTION:** Sets or returns the number of axis cards in the system.

**COMMAND SYNTAX:**
AXISBRD cr
AXISBRD = number cr

**REMARKS:**
The AXISBRD command returns the current value for the number of axis cards.

The AXISBRD=number command is only honored if the project directory is empty, DIR command return no project names. The number (1-4) sets the number of axis cards in the system.

This value is altered when a project is loaded into active ram and reflects the number of axis defined in a project.

The Power-on default with no projects is 1.

The Current value determines the maximum number of axes to be returned during a Host command.

**EXAMPLES:**
AXISBRD
Returns the current value of axis cards.

AXISBRD=4

Sets the current value of axis cards to 4 (8 axis).

# AXSTAT — Miscellaneous Command

**ACTION:** Returns the Axis Drive type, Units/Rev, Drive resolution and Task assigned to an axis.

**COMMAND SYNTAX:** AXSTAT(axis) cr

**REMARKS:** The axis specifies the number of an axis (1-8)

The returned line for the Drive Type is one of the following:
CL STEPPER
STEPPER
SERVO

The returned line for the Units/Rev is:
UNITS/REV = value

The returned line for Drive resolution (Stepper axis) is:
PULSES/REV = value

The returned line for Drive Resolution (Servo axis) is:
ENC LINES = value

The returned line for the Task assigned to an axis is
TASK n
Where n is the task number.

**EXAMPLES:** AXSTAT(1)
Returns the axis status for axis 1.

AXSTAT(2)
Returns the axis status for axis 2.

# "BACKSPACE" — Miscellaneous Command

**ACTION:** The Backspace key or ASCII code 08 can be used to delete one character from the host receiver buffer.

**COMMAND SYNTAX:** BACKSPACE (ASCII 08)

# BCD — I/O Function

**ACTION:** Returns the number set on a BCD switch bank connected to an expansion I/O board.

**COMMAND SYNTAX:** BCD(b0n) cr

**REMARKS:** See Programming Command **BCD**

**EXAMPLES:**

BCD(101)
Returns the setting of BCD switch bank 1 connected to expansion board 1.

# BUSY — Motion Parameter

**ACTION:** Returns the motion status of the selected axis. An axis is "busy" if motion is occurring.

**COMMAND SYNTAX:**
BUSY(axis) cr
BUSY cr

**REMARKS:** See Programming Command **BUSY**

**EXAMPLES:**
BUSY(1)
Returns the motion status of axis 1.

BUSY
Return the motion status for all assigned axes.

# CAPPOS — Miscellaneous Command

**ACTION:** Returns the last captured position of an axis from a MOVEHOME, MOVEREG or CAPTURE cycle.

**COMMAND SYNTAX:**
CAPPOS(axis) cr
CAPPOS cr

**REMARKS:** See Programming Command **CAPPOS.**

CAPPOS(axis) returns the last captured position for the specified axis.

CAPPOS returns the last captured position for all axes.

**EXAMPLES:**
CAPPOS(1) cr ‘Returns the last captured position for axis 1.

CAPPOS cr ‘Returns the last captured position for all axes.

# CAPTURE — Miscellaneous Command

**ACTION:** Sets the position capture trigger condition or returns the position capture status.

**COMMAND SYNTAX:**
CAPTURE(axis)= number cr
CAPTURE=number1, … , number8 cr
CAPTURE(axis) cr
CAPTURE cr

**REMARKS:** See Programming Command **CAPTURE.**

**EXAMPLES:**
CAPTURE(1) = 0 cr
Arms the trigger to capture the position of axis 1 when EVNT 1 becomes active.

CAPTURE = 0,,1 cr
Arms the trigger to capture the position of axis 1 when EVNT 1 becomes active and arms the trigger to capture the position of axis 3 when EVNT 1 becomes inactive.

CAPTURE(2) cr
Returns a 0 or a 1 to indicate whether or not a capture has occurred on axis 2.

CAPTURE cr
Returns a 0 or a 1 to indicate whether or not a capture has occurred on all axes.

# "CTRL A" — Miscellaneous Command

**ACTION:** Stops all motion and all tasks.

**COMMAND SYNTAX:** Simultaneously press the keyboard keys marked "A" and the control key "CTRL".

**REMARKS:** "CTRL A" will stop execution of all tasks presently running on the controller; all motion ceases immediately.

If the axis is a servo axis "CTRL A" does not turn off the servo output voltage.

# "CTRL C" — Miscellaneous Command

**ACTION:** Stops all motion and all tasks.

**COMMAND SYNTAX:** Simultaneously press the keyboard keys marked "C" and the control key "CTRL".

**REMARKS:** "CTRL C" will stop execution of all tasks presently running on the controller; all motion ceases immediately.

If the axis is a servo axis "CTRL C" turns off the servo output voltage. To turn the servo output back on use the "WNDGS(axis) = 1 " command.

# DECEL — Trajectory Parameter

**ACTION:** Sets or returns the deceleration value of the selected axis.

**COMMAND SYNTAX:**
DECEL(axis)=number cr
DECEL=number1, number2, . . . , number8 cr
DECEL(axis) cr
DECEL cr

**REMARKS:** See Programming Command **DECEL.**

**EXAMPLE:**
DECEL(2)=50
Sets the deceleration rate for axis 2 to 50 units/sec$^2$.

DECEL=50,,75
Sets the deceleration rate for axis 1 to 50 units/sec$^2$ , axis 2 is unchanged and axis 3 to 75 units/sec$^2$.

DECEL(2)
Returns the deceleration rate for axis 2.

DECEL
Returns the deceleration rate for all assigned axes.

# DELTACAPPOS — Miscellaneous Command

**ACTION:** Returns the difference between the current captured position and the previously captured position.

**COMMAND SYNTAX:**
DELTACAPPOS(axis) cr
DELTACAPPOS cr

**REMARKS:** See Programming Command **DELTACAPPOS.**

**EXAMPLES:**
DELTACAPPOS(3) cr
Returns the difference between the current captured position and the previously captured position for axis 3.

DELTACAPPOS cr
Returns the difference between the current captured position and the previously captured position for all axes.

# DIR — Miscellaneous Command

**ACTION:** List the names of projects and tasks stored in non-volatile memory.

**COMMAND SYNTAX:** DIR cr

**REMARKS:**

The transfer format is:

```
n*s  Project name  checksum
     Task name
     Task name
     etc
n*s  Project name  checksum
     Task name
     Task name
     etc
free space = nnnn
```

**n** is the project number (0 to 6).

***** indicates that this project is loaded into DSP card memory

**s** indicates that the project source code is loaded

**Project Name** is the name of the project

**checksum** is a unique value for that project.

**Task name** is the name of the Task in a project.

**Note: If the CLR input is open circuited at power-on no projects will be loaded into the DSP memory.**

**EXAMPLES*:***

DIR

Transfers the names of the user projects and tasks stored in memory.

# DRVREADY — Motion Parameter

**ACTION:** Enables or disables the checking of the drive (READY) signal on the axis card.

**PROGRAM SYNTAX:**

DRVREADY(axis)=number cr
DRVREADY=number1, **…** , number8
DRVREADY(axis) cr

**REMARK:** See Programming Command **DRVREADY**.

**EXAMPLE:**

DRVREADY(3)=1
Bypasses the drive READY signal checking for axis 3.

DRVREADY= 1,,1
Bypasses the drive READY signal checking for axis 1 and axis 3.

DRVREADY(3)
Return the Drive Ready status for axis 3.

DRVREADY
Return the Drive Ready status for all axes.

# ENCBAND — Motion Parameter

**ACTION:** Sets or returns the maximum position error allowed when motion is stopped.

**PROGRAM SYNTAX:**
ENCBAND(axis)=number cr
ENCBAND=number1, …, number 8 cr
ENCBAND(axis) cr
ENCBAND cr

**REMARK:** See Programming Command **ENCBAND**.

**EXAMPLE:**
ENCBAND(3)=.1
Sets the maximum position error of axis 3 to .1 unit.

ENCBAND=.1 ,,1.5
Sets the maximum position error of axis 1 to .1 unit, and axis 3 to .15 unit.

ENCBAND(3)
Returns the maximum position error of axis 3.

ENCBAND
Returns the maximum position error of all axes.

# ENCERR — Trajectory Parameter

**ACTION:** Returns the position error of the designated axis.

**PROGRAM SYNTAX:**
ENCERR(axis) cr
ENCERR cr

**REMARK:** See Programming Command **ENCERR**.

**Note: POSERR can be used in place of ENCERR.**

**EXAMPLE:**
ENCERR(1)
Returns the present position error of axis 1.

ENCERR
Returns the present position error of all axes.

# ENCFOL — Motion Parameter

**ACTION:** Sets or returns the maximum position error allowed during motion.

**PROGRAM SYNTAX:**
ENCFOL(axis)= number cr
ENCFOL=number1, … , number8 cr
ENCFOL(axis) cr
ENCFOL cr

**REMARK:** See Programming Command **ENCFOL**.

**Note: FOLERR can be used in place of ENCFOL.**

**EXAMPLE:**
ENCFOL(2)=.4
Sets the following error of axis 2 to .4 units.

ENCFOL=.4 ,, .3
Sets the following error of axis 1 to .4 units, and axis 3 is set to .3 units.

ENCFOL(2)
Returns the current following error set for axis 2.

ENCFOL
Returns the current following error set for all axes.

# ENCMODE — Motion Parameter

**ACTION:** Sets or returns the operating mode of a closed loop stepper axis.

**PROGRAM SYNTAX:**
ENCMODE(axis)=number cr
ENCMODE=number1, ... , number8 cr
ENCMODE(axis) cr
ENCMODE cr

**REMARK:** See Programming Command **ENCMODE**

**EXAMPLE:**
ENCMODE(2)=0
Sets axis 2 to open loop operation.

ENCMODE=1,,2
Sets axis 1 to halt execution on excessive error, axis 2 no change and axis 3 to correct position on excessive following error.

ENCMODE(2)
Returns the operating mode of a closed loop for axis 2.

ENCMODE
Returns the operating mode of a closed loop for all assigned axes.

# ENCPOS — Trajectory Parameter

**ACTION:** Returns the encoder position of an axis.

**PROGRAM SYNTAX:**
ENCPOS(axis) cr
ENCPOS cr

**REMARK:** See Programming Command **ENCPOS.**

**EXAMPLE:**
ENCPOS(1)
Returns the encoder value of axis 1.

ENCPOS
Returns the encoder value for all assigned axes.

# ENCRES — Motion Parameter

**ACTION:** Returns the encoder line count of the selected axis.

**COMMAND SYNTAX:**
ENCRES(axis) cr
ENCRES cr

**REMARKS:** The axis specifies the number of the axis (1-8).

ENCRES(axis)
Returns the current line count of the specified axis.

ENCRES
Returns the current line count of all axes.

**EXAMPLES:**
ENCRES(2)
Returns the current line count of axis 2.

# ENCSPD — Trajectory Parameter

**ACTION:** Returns the current encoder speed in units/second.

**PROGRAM SYNTAX:**
ENCSPD(axis) cr
ENCSPD cr

**REMARK:** See Programming Command **ENCSPD.**

**EXAMPLE:**
ENCSPD(2)
Returns the current encoder speed of axis 2.

ENCSPD
Returns the current encoder speed of all assigned axes.

# ERASE — Miscellaneous Command

**ACTION:** Erases a specific project or all projects stored in non-volatile memory.

**COMMAND SYNTAX:**
ERASE DIR cr
ERASE project name cr

**REMARKS:** The project erased is not recoverable.

ERASE DIR
Erases all projects stored in non-volatile memory.

ERASE project name
Erases the defined project name in non-volatile memory.

**EXAMPLES:**
ERASE DIR
Erases all projects stored in non-volatile memory.

ERASE CONVEYER
Erases project CONVEYER if it exists.

# ERR — Miscellaneous Command

**ACTION:** Returns the controller error/warning number for this task.

**PROGRAM SYNTAX:** ERR cr

**REMARKS:** This command returns the error/warning status for all task and clears the errors and axis which created the error. See the **ERR** basic Command for the error/warning code listing.

The transfer format for task 0-7 is:
nn nn nn nn nn nn nn nn <cr> <lf>
where: nn is 0-99 for task 0-7

The axis which created the error/warning can be interrogated using the ERRAXIS command. Send the ERRAXIS command prior to the ERR command to interrogate the error and the axis which created the error.

**EXAMPLES:**
ERRAXIS : ERR <cr>
This sequence returns the axis which created the error/warning and the error/warning number on two separate lines.

# ERRAXIS Miscellaneous Command

**ACTION:** Returns the controller axis number which created the error/warning for the task.

**PROGRAM SYNTAX:** ERRAXIS cr

**REMARKS:** If a zero is returned for the task then the error/warning was not axis related or there is no error/warning.

To determine the error/warning use the **ERR** or **ERRM** command.

The transfer format for task 0-7 is:
n n n n n n n n <cr> <lf>
where: n is 0-8 for task 0-7

**EXAMPLE:** ERRAXIS : ERR <cr>
This sequence returns the axis which created the error/warning and the error/warning number on two separate lines.

# ERRM Miscellaneous Command

**ACTION:** Returns the error/warning message's for the task's (0-7).

**PROGRAM SYNTAX:** ERRM cr

**REMARKS:** This command returns the error/warning message's for all task and clears the errors and axis which created the error.

The error messages returned are:

```
 0  no Errors
 1  + Limit activated        ' motion in +dir activated +Limit
 2  - Limit activated        ' motion in -dir activated -Limit
 3  Soft Limit in +dir       ' +dir soft limit exceeded
 4  Soft Limit in -dir       ' -dir soft limit exceeded
 5  CL attempts                    ' CL stepper attempts elapsed
 6  Follow Error             ' Following Error exceeded
 7  MoveReg Dist Small       ' MOVEREG distance to small for
                             DECEL rate.
 8  DRVREADY fault                 ' Drive not ready
 9  Drive Not Enabled        ' Servo drive not enabled
10  Program Out of Memory    ' Program Ram all used up
26  IXT Servo Error          ' Excessive Duty Cycle Shutdown
27-99 User define ERR nn     ' User Program defined Error
```

The warning messages returned are:

```
11  Warn Axis not in task    ' axis is not defined in this task.
12  Warn ANALOG I/O range    ' ANALOG point does not exist.
13  Warn BCD range           ' BCD bank does not exist.
14  Warn EXIN range          ' EXIN point does not exist.
15  Warn EXOUT range         ' EXOUT point does not exist.
16  Warn IN range            ' IN point does not exist.
17  Warn OUT range           ' OUT point does not exist.
18  Warn LOG value <=0       ' LOG value out of range.
19  Warn SQRT arg negative   ' SQRT value is negative.
20  Warn NVR range                 ' NVR element does not exist.
21  Warn READ out of arg           ' READ command out of data.
22  Warn MAXSPD range              ' MAXSPD value out of range
23  motion at program end    ' Motion occurring when program ended
24  SETCOM error                   ' Aux. serial port parameter er-
                                   ror
25  Warn Servo Gain range    ' Servo axis Gain out of range
```

**EXAMPLES:**

```
ERRAXIS : ERRM <cr>
```

This sequence returns the axis which created the error/warning and the error/warning message's on two separate lines.

# "ESC" Miscellaneous Commands

**ACTION:** The ESC key (or ESC character code sent via a serial port) is used during program execution to force execution of a command in the host buffer.

**COMMAND SYNTAX:** ESC (ASCII 27) command

**REMARKS:** When the controller is executing a BASIC program, any host commands received are queued for execution after the BASIC program finishes. The execution of a host command can be forced to happen immediately by preceding it with the ESC character (ASCII 27). The command will consist of all characters from the ESC to the cr (carriage return). Multiple commands can be placed on one line, but they must be separated by colons (:).

**EXAMPLE:** <ESC>ABSPOS(1)
Returns the absolute position of axis 1 during program execution.

# EVENT1 Motion Parameter

**ACTION:** Returns the state of the trigger input labeled EVNT1 or sets the trigger polarity and enable for a Movehome and Movereg cycle.

**PROGRAM SYNTAX:** EVENT1(axis)=number cr
EVENT1=number1, ... , number8 cr
EVENT1(axis) cr
EVENT1 cr

**REMARKS:** See Programming Command **EVENT1.**

**EXAMPLES:** EVENT1(2)=0
disables Event1 as a MOVEREG trigger on axis 2

EVENT1(2)=1
enables Event1 to trigger when activated on axis 2

EVENT1(2)=-1
enables Event1 to trigger when open circuited on axis 2.

EVENT1(2)
Returns the current input state for the EVNT1 input on axis 2.

EVENT1
Returns the current input states for all EVNT1 inputs on all assigned axes.

# EVENT2 — Motion Parameter

**ACTION:** Returns the state of the trigger input labeled EVNT2 or sets the trigger polarity and enable for a Movehome and Movereg cycle.

**PROGRAM SYNTAX:**
EVENT2(axis)=number cr
EVENT2=number1, ... , number8 cr
EVENT2(axis) cr
EVENT2 cr

**REMARKS:** See Programming Command **EVENT2**.

**EXAMPLES:**
EVENT2(2)=0
disables Event2 as a MOVEREG trigger on axis 2.

EVENT2(2)=1
enables Event2 to trigger when activated on axis 2.

EVENT2(2)=-1
enables Event2 to trigger when open circuited on axis 2.

EVENT2(2)
Returns the current input state for the EVNT2 input on axis 2.

EVENT2
Returns the current input states for all EVNT2 inputs on all assigned axes.

# EXIN — I/O Function

**ACTION:** Returns the state of the specified expansion I/O inputs.

**PROGRAM SYNTAX:**
EXIN(nnn) cr
EXIN(nnn,len) cr

**REMARKS:** See Programming Command **EXIN**.

**EXAMPLE:**
EXIN(207)
returns the state of board 2 input 7

EXIN(207,3)
Returns a number 0-7 depending on the states of inputs 207-209.
EXIN(207)+2*EXIN(208)+4*EXIN(209)

# EXOUT — I/O Function

**ACTION:** Sets or returns the state of the specified expansion I/O outputs.

**PROGRAM SYNTAX:**
EXOUT(nnn) cr
EXOUT(nnn,len) cr
EXOUT(nnn)=number cr
EXOUT(nnn,len)=number cr

**REMARKS:** See Programming Command **EXOUT**.

**EXAMPLES:**
EXOUT(207)=-3
turns output 7 on board 2 on

EXOUT(207)=0
turns output 7 on board 2 off

EXOUT(207)
Returns the last commanded output for 207.

EXOUT(207,3)=6.2
outputs 209=on, output 208=on and output 207=off

EXOUT(207,3)=4
output 209=on, output 208=off and output 207=off.

EXOUT(208,2)
Returns the last commanded state for output 208 and 209

# FILTER — Miscellaneous Command

**ACTION:** Sets the filter value for the for the defined analog input

**PROGRAM SYNTAX:**
FILTER(b0n)=number cr
FILTER(b0n) cr

**REMARKS:**
The "b" specifies the board (1-4).

The "n" specifies the analog input (1-4).

The number sets the filter value (.01 - 1). Where 1.0 is no filtering.

FILTER(b0n)=number
Sets the filter value for the designated board and input.

FILTER(b0n)
Returns the filter value for the designated board and input.

**EXAMPLES:**
FILTER(101)=.1
Sets the filter value for board 1 input to a value of .1.

FILTER(302)=.1
Sets the filter value for board 1 input to a value of .1.

# FOLERR — Motion Parameter

**ACTION:** Sets or returns the maximum position error allowed during motion, herein referred to as "following error."

**COMMAND SYNTAX:**
FOLERR(axis)=number cr
FOLERR=number1, number2, . . . , number8 cr
FOLERR(axis) cr
FOLERR cr

**REMARKS:** See Programming Command **FOLERR**.

**Note: ENCFOL can be substituted for FOLERR.**

**EXAMPLES:**
FOLERR(2)=.4
Sets the following error of axis 2 to .4 units.

FOLERR=.4,, .3
Sets the following error of axis 1 to .4 units, axis 2 is unchanged and axis 3 is set to .3 units.

FOLERR(2)
Returns the current following error set for axis 2.

FOLERR
Returns the current following error set for all assigned axes.

# FREE — Miscellaneous Command

**ACTION:** Transfers the free space, in sectors, available in non-volatile memory.

**COMMAND SYNTAX:** FREE cr

**REMARKS:** A sector consists of 128 bytes of non-volatile memory. With no program(s) loaded, the free space value is 2044 sectors.

The transfer format is:
free space = nnnncr

# FREEMEM — Miscellaneous Command

**ACTION:** Returns the amount of free memory for program execution allocation.

**COMMAND SYNTAX:** FREEMEM cr

**REMARKS:** The value returned is the number of 32 bit word free for allocation.

The DIM command uses free memory for allocating an area for arrays.

A new variable string uses free memory for storing the string characters.

The maximum free memory size is 45055 words.

If an "Out of Memory" error occurs during program execution the FREEMEM command can be used to determine whether the error was created by a string command or that the memory allocated for program storage was exceeded. If the FREEMEM command returns a negative value the memory allocated for program storage was exceeded.

**EXAMPLE:** FREEMEM

# HARDLIMNEG — Over Travel Limit

**ACTION:** Returns the -LIMIT state for the selected axis.

**PROGRAM SYNTAX:** HARDLIMNEG cr
HARDLIMNEG(axis) cr

**REMARKS:** See Programming Command **HARDLIMNEG**.

**EXAMPLES:** HARDLIMNEG(2)
Returns the -LIMIT state of axis 2.

HARDLIMNEG
Returns the -LIMIT state of all assigned axes.

# HARDLIMPOS — Over Travel Limit

**ACTION:** Returns the +LIMIT state of the selected axis.

**PROGRAM SYNTAX:** HARDLIMPOS cr
HARDLIMPOS(n) cr

**REMARKS:** See Programming Command **HARDLIMPOS**.

**EXAMPLES:** HARDLIMPOS(2)
Returns the +LIMIT state of axis 2.

HARDLIMPOS
Returns the +LIMIT state of all assigned axes.

# IN — I/O Function

**ACTION:** Returns the state of the specified digital I/O inputs.

**PROGRAM SYNTAX:**
IN(bnn) cr
IN(bnn,len) cr

**REMARKS:** See Programming command **IN**.

**EXAMPLES:**
IN(207)
Returns the state of board 2 input 7.

IN(207,3)
Returns the sum of input states 7-9 on board 2. The value returned will be: IN(207) + (2*IN(208) + (4*IN(209).

# INTLIM — Servo Parameter

**ACTION:** Sets the Integral limit for the controller. This is the limit of the contribution to the servo output from the integral of the position error.

**PROGRAM SYNTAX:**
INTLIM(axis)=number cr
INTLIM=number1, ... , number8 cr
INTLIM(axis) cr
INTLIM cr

**REMARKS:** See Programming Command **INTLIM**.

**EXAMPLES:**
INTLIM(2)=50
sets the integral limit for axis 2 to 50 volts.

INTLIM(2)
returns the integral limit of axis 2.

INTLIM=50,,100
sets the integral limit for axis 1 to 50 volts, axis 2 is unchanged and axis 3 is set to 100 volts.

INTLIM
Returns the integral limits on all assigned axes.

# JOG — Motion Parameter

**ACTION:** Runs the motor continuously in a specified direction.

**PROGRAM SYNTAX:**
JOGSTART(axis)=number cr
JOGSTART=number1, ... ,number8 cr

**Note: JOGSTART can be substituted for JOG**

**REMARKS:** See Programming Command **JOG**.

**EXAMPLES:**
JOGSTART(2)=1
Runs axis 2 continuously in the +direction.

JOGSTART=1,,-1
Runs axis 1 continuously in the +direction, axis 2 is unchanged and axis 3 runs continuously in the -direction.

# KAFF — Servo Parameter

**ACTION:** Sets or returns the acceleration feed forward gain for a servo axis.

**PROGRAM SYNTAX:**
KAFF(axis)=number cr
KAFF=number1, ... , number8 cr
KAFF(axis) cr
KAFF cr

**REMARKS:** See Programming Command **KAFF**.

**EXAMPLES:**
KAFF(2)=.5
Sets the acceleration feed forward gain of axis 2 to .5 volts/encoder count/msec$^2$.

KAFF=.2,,0
Sets the acceleration feed forward gain of axis 1 to .2 volts/encoder count/msec$^2$, axis 2 is unchanged and axis 3 is set to 0 volts/encoder count/msec$^2$.

KAFF(2)
Returns the acceleration feed forward gain of axis 2.

KAFF
Returns the acceleration feed forward gain of all assigned axes.

# KD Servo Parameter

**ACTION:** Sets or returns the derivative gain for the servo axis.

**PROGRAM SYNTAX:**
KD(axis)=number cr
KD=number1, ... , number8 cr
KD(axis) cr
KD cr

**REMARKS:** See Programming Command **KD**.

**EXAMPLES:**
KD(2)=4
Sets the derivative gain of axis 2 to 4 milliseconds.

KD=10,,8
Sets the derivative gain of axis 1 to 10 milliseconds, axis 2 is unchanged and axis 3 is set to 8 milliseconds.

KD(2)
Returns the derivative gain of axis 2.

KD
Returns the derivative gain of all assigned axes.

# KI Servo Parameter

**ACTION:** Sets or returns the integral gain of a servo axis.

**PROGRAM SYNTAX:**
KI(axis)=number cr
KI=number1, ... , number8 cr
KI(axis) cr
KI cr

**REMARKS:** See Programming Command **KI**.

**EXAMPLES:**
KI(2)=4
Sets the Integral gain of axis 2 to 4 milliseconds.

KI=1,,4
Sets the Integral gain of axis 1 to 1 milliseconds, axis 2 is unchanged and axis 3 is set to 4 milliseconds.

KI(2)
Returns the Integral gain of axis 2.

KI
Returns the Integral gain of all assigned axes.

# KP — Servo Parameter

**ACTION:** Sets or returns the proportional gain of the servo axis.

**PROGRAM SYNTAX:**
KP(axis)=number cr
KP=number1, ... , number8 cr
KP(axis) cr
KP cr

**REMARKS:** See Programming Command **KP**.

**EXAMPLES:**

KP(2)=20
Sets the Proportional gain of axis 2 to 20 millivolts/encoder count.

KP=18,,20
Sets the Proportional gain of axis 1 to 18 millivolts/encoder count, axis 2 is unchanged and axis 3 is set to 20 millivolts/encoder count.

KP(2)
Returns the Proportional gain of axis 2.

KP
Returns the Proportional gain of all assigned axes.

# KVFF — Servo Parameter

**ACTION:** Sets or returns the velocity feed forward gain for the servo axis.

**PROGRAM SYNTAX:**
KVFF(axis)=number cr
KVFF=number1, ... , number8 cr
KVFF(axis) cr
KVFF cr

**REMARKS:** See Programming Command **KVFF**.

**EXAMPLES:**

KVFF(2)=95
Sets the Velocity feed forward gain of axis 2 to 95%.

KVFF=98,,95
Sets the Velocity feed forward gain of axis 1 to 98% , axis 2 is unchanged and axis 3 is set to 95%.

KVFF(2)
Returns the Velocity feed forward gain of axis 2.

KVFF
Returns the Velocity feed forward gain of all assigned axes.

# LINE — Motion Parameter

**ACTION:** Initiates a coordinated linear move involving up to 8 axes.

**PROGRAM SYNTAX:** LINE=number1, ... , number8 cr

**REMARKS:** See Programming Command **LINE**.

**EXAMPLES:** LINE=1.0,,-2.0
Linear interpolated axis 1 and 3. Axis 1 moves +1.0 units , and axis 3 moves -2.0 units.

# LOAD — Miscellaneous Command

**ACTION:** Loads the designated project from non-volatile memory into operating memory.

**COMMAND SYNTAX:** LOAD project name cr

**REMARKS:** The name is limited to eight characters.

**EXAMPLES:** LOAD CONVEYER
Load project CONVEYER into operating memory.

# LOWSPD — Trajectory Parameter

**ACTION:** Sets or returns the Low Speed (starting speed) value of a stepping motor axis.

**PROGRAM SYNTAX:** LOWSPD(axis)=number cr
LOWSPD=number1, ... ,number8 cr
LOWSPD(axis) cr
LOWSPD cr

**REMARKS:** See Programming Command **LOWSPD**.

**EXAMPLES:** LOWSPD(2)=1.5
sets axis 2 to 1.5 units/second.

LOWSPD=1.3,, 1.5
sets axis 1 to 1.3 units/second, axis 2 is unchanged, and axis 3 to 1.5 units/second.

LOWSPD(2)
Returns the low speed value for axis 2.

LOWSPD
Returns the low speed value for all assigned axes.

# MAXSPD — Trajectory Parameter

**ACTION:** Sets or returns the maximum allowed speed of the specified axis.

**PROGRAM SYNTAX:**
MAXSPD(axis)=number cr
MAXSPD=number1, ... , number 8 cr
MAXSPD(axis) cr
MAXSPD cr

**REMARKS:** See Programming Command **MAXSPD**.

**EXAMPLES:**
MAXSPD(3)=50
Sets the maximum speed for axis 3 to 50 units/second.

MAXSPD=50,,60
Sets the maximum speed for axis 1 to 50 units/second, axis 2 is unchanged and axis 3 to 60 units/second.

MAXSPD(2)
Returns the maximum speed for axis 2.

MAXSPD
Returns the maximum speed for all assigned axes.

# MOVE — Motion Parameter

**ACTION:** Initiates a non-coordinated move.

**PROGRAM SYNTAX:**
MOVE(axis)=number cr
MOVE=number1, ... , number8

**REMARKS:** See Programming Command **MOVE**.

**EXAMPLES:**
POSMODE(1,3)=0,0
MOVE(3)=-2
axis 3 moves -2 units.

MOVE=1,,3
axis 1 moves +1 units, and axis 3 moves +3 units.

# MOVEHOME — Motion Parameter

**ACTION:** Runs the motor until the home input is activated, captures and records the position of the switch activation as home (electrical zero), then decelerates the motor to a stop.

**PROGRAM SYNTAX:**
MOVEHOME(axis)=number
MOVEHOME=number1, ... , number8

**REMARKS:** See Programming Command **MOVEHOME**.

**EXAMPLES:**
MOVEHOME(3)=1
Axis 3 executes a home cycle in the positive direction.

MOVEHOME=-2,,3
Axis 1 executes a home cycle in the negative direction, axis 2 is unchanged and axis 3 executes a home cycle in the positive direction.

# MOVEREG — Motion Parameter

**ACTION:** Runs the motor until the mark registration input is activated; then moves the motor the desired registration distance.

**PROGRAM SYNTAX:**
MOVEREG(axis)=number
MOVEREG=number1, ... , number8

**REMARKS:** See Programming Command **MOVEREG**.

**EXAMPLES:**
MOVEREG(3)=2
Initiates a positive registration cycle of 2 units for axis 3.

MOVEREG=1,,-2
Initiates a positive registration cycle of 1 unit for axis 1, axis 2 is unchanged and initiates a negative registration cycle of 2 units for axis 3.

# NVR — Miscellaneous Command

**ACTION:** The NVR array is used for non-volatile variable storage.

**PROGRAM SYNTAX:**
NVR(number) cr
NVR(number)=value cr

**REMARKS:** See Programming Command **NVR**.

**EXAMPLES:**
NVR(2)
Returns the NVR element 2 value.

NVR(2048)=10.5
Sets the NVR element 2048 to a value of 10.5.

# NVRBIT — Miscellaneous Command

**ACTION:** Store or return the bit value in NVR memory.

**PROGRAM SYNTAX:**
NVRBIT(bit)= number cr
NVRBIT(bit) cr

**REMARKS:** See Programming Command **NVRBIT**.

**EXAMPLES:**
NVRBIT(65505)=1
sets Bit 1 of element 2048 = 1

NVRBIT(65536)=0
sets Bit 32 of element 2048 = 0

# NVRBYTE — Miscellaneous Command

**ACTION:** Stores or returns the byte value in NVR memory.

**PROGRAM SYNTAX:**
NVRBYTE(byte)=number cr
NVRBYTE(byte) cr

**REMARKS:** See Programming Command **NVRBYTE**.

**EXAMPLES:**
NVRBYTE(8192)=255
sets MSB byte = 255 in element 2048

NVRBYTE(8189)=0
sets LSB byte = 0 in element 2048

# OUT — I/O Function

**ACTION:** Sets or returns the condition of a specified digital output.

**PROGRAM SYNTAX:**
OUT(bnn)=number cr
OUT(bnn,len)=number cr
OUT(bnn) cr
OUT(bnn,len) cr

**REMARKS:** See Programming Command **OUT**.

**EXAMPLES:**
OUT(107)=1
Digital I/O board 1 output 7 is set to a 1.

OUT(101,6)=48
digital I/O board 1 outputs 1-4 are set to a 0 and outputs 5 and 6 are set to a 1.

OUT(107)
Returns output 7 on digital I/O board 1.

OUT(101,7)
Returns outputs 1-7 on digital I/O board 1.

# OUTLIMIT — Servo Parameter

**ACTION:** Sets or returns the servo command voltage limit.

**PROGRAM SYNTAX:**
OUTLIMIT(axis)=number cr
OUTLIMIT=number1, ... , number8 cr
OUTLIMIT(axis) cr
OUTLIMIT cr

**REMARKS:** See Programming Command **OUTLIMIT**.

**EXAMPLES:**
OUTLIMIT(2)=5
Limits the magnitude of the servo output voltage for axis 2 to ± 5 volts.

OUTLIMIT=5,,10
Limits the magnitude of the servo output for axis 1 to ± 5 volts, axis 2 is unchanged and axis 3 to ± 10 volts.

OUTLIMIT(2)
Returns the magnitude of the servo output for axis 2.

OUTLIMIT
Returns the magnitude of the servo output for all assigned axes.

# POSERR — Trajectory Parameter

**ACTION:** Returns the position error (absolute position - encoder position) of the selected axis.

**COMMAND SYNTAX:**
POSERR(axis) cr
POSERR cr

**Note: ENCERR can be substituted for POSERR**

**REMARKS:** See Programming Command **POSERR.**

**EXAMPLES:**
POSERR(1)
Returns the present position error of the specified axis.

POSERR
Returns the present position error of all assigned axes.

# POSMODE — Motion Parameter

**ACTION:** Sets or returns the positioning mode for the specified axis.

**PROGRAM SYNTAX:**
POSMODE(axis)=number cr
POSMODE=number1, ... , number8 cr
POSMODE(axis) cr
POSMODE cr

**REMARKS:** See Programming Command **POSMODE**.

**EXAMPLES:**
POSMODE(2)=1
Sets the positioning mode for axis 2 to absolute.

POSMODE=1,,0
Sets the positioning mode for axis 1 to absolute, axis 2 is unchanged and axis 3 is set to incremental positioning mode.

POSMODE(2)
Returns the positioning mode for axis 2.

POSMODE
Returns the positioning mode for all assigned axes.

# PROFILE — Trajectory Parameter

**ACTION:** Determines how the motor speed changes.

**PROGRAM SYNTAX:**
PROFILE(axis)=number cr
PROFILE=number1, ... , number8 cr
PROFILE(axis) cr
PROFILE cr

**REMARKS:** See Programming Command **PROFILE**.

**EXAMPLES:**
PROFILE(2)=10
axis 2 profile is set to a value of 10.

PROFILE=16,,32
axis 1 profile is set to a value of 16 and axis 3 profile is set to 32.

PROFILE(2)
Returns the profile value for axis 2.

PROFILE
Returns the profile value for all assigned axes.

# REGLIMIT — Over Travel Limit

**ACTION:** Sets or returns the distance to be moved during a MOVEREG cycle, while awaiting a trigger.

**PROGRAM SYNTAX:**
REGLIMIT(axis)=number cr
REGLIMIT=number1, ... , number8 cr
REGLIMIT(axis) cr
REGLIMIT cr

**REMARKS:** See Programming Command **REGLIMIT**.

**EXAMPLES:**
REGLIMIT(2)= 10
set the MOVEREG travel distance limit on axis 2 to 10 units

REGLIMIT=0,,10
disables the REGLIMIT for axis 1, axis 2 is unchanged and axis 3 has MOVEREG travel distance limit of 10 units.

REGLIMIT(2)
Returns the Registration travel limit for axis 2.

REGLIMIT
Returns the Registration travel limit for all assigned axes.

# RESET — Miscellaneous Command

**ACTION:** Resets the MX2000 controller.

**COMMAND SYNTAX:** RESET cr

**REMARKS:** This command causes the system to halt, and then restart as though power had been recycled.

# REVISION — Miscellaneous Command

**ACTION:** Returns the current revision level of the controller's operating system software.

**COMMAND SYNTAX:** REVISION cr

**REMARKS:** The return format for this command is:
MX2000 REV n , date
where "n" is the current revision number and "date" is the release date.

# RUN — Miscellaneous Command

**ACTION:** Runs the loaded project or specified task number.

**COMMAND SYNTAX:** RUN cr

**REMARKS:** RUN starts execution of all loaded tasks from their respective beginnings.

# SNVR — Miscellaneous Command

**ACTION:** Sets the default value for the designated NVR elements.

**COMMAND SYNTAX:** SNVR(start, end)=value cr lf

**REMARKS:** **start** is the starting element number in NVR. The range is 1-2048.

**end** is the ending element in NVR. The range start-2048.

The value is stored from the starting element to the ending element.

A hardware option is available that allows up to 32720 variables to be saved.

**EXAMPLES**:

SNVR(1,2000)=0
Sets NVR(1-2000) to 0

SNVR(10,100)=-1
Sets NVR(10-100) to -1

SNVR(1000,1200)=0xff
Sets NVR(100,1200) to 255.

# SOFTLIMNEG Over Travel Limit

**ACTION:** Programmable "software limit switch" for motion in the negative direction. Sets or returns the absolute negative travel position value for the specified axis.

**PROGRAM SYNTAX:**
SOFTLIMNEG(axis)=number cr
SOFTLIMNEG=number1, ... ,number8 cr
SOFTLIMNEG(axis) cr
SOFTLIMNEG cr

**REMARKS:** See Programming Command **SOFTLIMNEG**.

**EXAMPLES:**
SOFTLIMNEG(2) =-4
Sets the negative direction soft limit of axis 2 at -4 units.

SOFTLIMNEG=-5,,-6
Sets the negative direction soft limit of axis 1 at -5 units, axis 2 is unchanged and axis 3 is set to -6 units.

SOFTLIMNEG(2)
Returns the negative direction soft limit value for axis 2.

SOFTLIMNEG
Returns the negative direction soft limit value for all assigned axes.

# SOFTLIMPOS Over Travel Limit

**ACTION:** Programmable "software limit switch" for motion in the positive direction. Sets or returns the absolute positive travel position value for the specified axis.

**PROGRAM SYNTAX:**
SOFTLIMPOS(axis)=number cr
SOFTLIMPOS=number1, ... ,number8 cr
SOFTLIMPOS(axis) cr
SOFTLIMPOS cr

**REMARKS:** See Programming Command **SOFTLIMPOS**.

**EXAMPLES:**
SOFTLIMPOS(2) =4
Sets the positive direction soft limit of axis 2 at +4 units.

SOFTLIMPOS=5,,6
Sets the positive direction soft limit of axis 1 at +5 units, axis 2 is unchanged and axis 3 is set to +6 units.

SOFTLIMPOS(2)
Returns the positive direction soft limit value for axis 2.

SOFTLIMPOS
Returns the positive direction soft limit value for all assigned axes.

# SPEED — Trajectory Parameter

**ACTION:** Sets and returns the target velocity of the motor.

**PROGRAM SYNTAX:**
SPEED(axis)=number cr
SPEED=number1, ... , number8 cr
SPEED(axis) cr
SPEED cr

**REMARKS:** See Programming Command **SPEED**.

**EXAMPLES:**
SPEED(2)=10
Sets the speed of axis 2 to 10 units/second.

SPEED=0,,5
Sets the speed of axis 1 to 0 units/second, axis 2 is unchanged and axis 3 to 5 units/second.

SPEED(2)
Returns the speed value for axis 2.

SPEED
Returns the speed value for all assigned axes.

# STOP — Motion Parameter

**ACTION:** Stops any motion with a control stop.

**PROGRAM SYNTAX:**
STOP(axis)=number cr
STOP=number1 , ... , number8

**REMARKS:** See Programming Command **STOP**.

**note: JOGSTOP can be substituted for STOP.**

**EXAMPLES:**
STOP(2)=1
requests axis 2 to stop.

STOP=1,,1
requests axis 1 and axis 3 to stop.

# STOPERR — Motion Parameter

**ACTION:** Sets or returns the maximum position error allowed when motion is stopped, referred to herein as "position error band."

**COMMAND SYNTAX:**
STOPERR(axis)=number cr
STOPERR=number1, … , number8 cr
STOPERR(axis) cr
STOPERR cr

**REMARKS:** See Programming Command **STOPERR**.

**Note: ENCBAND can be substituted for STOPERR.**

**EXAMPLES:**
STOPERR(3)=.1
Sets the maximum position error for axis 3 to .1 units.

STOPERR=.1,,,.15
Sets the maximum position error for axis 1 to .1 units and axis 4 to .15 units.

STOPERR(3)
Returns the maximum position error value of axis 3.

# UNIT — Miscellaneous Command

**ACTION:** Returns the pulses/ unit value. Used for programming in "user units," such as inches or revolutions or meters, etc.

**COMMAND SYNTAX:**
UNIT(axis) cr
UNIT cr

**REMARKS:** The axis specifies the number of the axis (1-8).

Unit is a signed value, and represents the number of pulses or counts per unit. A positive value defines CW direction as the positive direction. A negative value defines CCW direction as the positive direction.

**EXAMPLES:**
UNIT(2)
Returns the unit value for axis 2.

UNIT
Returns the unit value for all assigned axes.

# VELOCITY — Trajectory Parameter

**ACTION:** Sets or returns the path speed to be used for coordinated motion.

**COMMAND SYNTAX:**
VELOCITY = number cr
VELOCITY cr

**REMARKS:** This velocity is only used as the path speed for the Host commands LINE and ARC.

**EXAMPLES:**
VELOCITY=1.0
Sets the coordinated for linear motion to 1 unit/second.

VELOCITY
Returns the current velocity for host mode.

# WARNING — Miscellaneous Command

**ACTION:** Returns the warning number of each task.

**PROGRAM SYNTAX:** WARNING cr

**REMARKS:** Returns the warning number for task 1-7 and clears the task warnings and axis which created the warning.

The Warning return format is:
nn nn nn nn nn nn nn <cr><lf>
where: nn is a 0 or 11-25

The Predefined Warning codes are listed in the Programming Command **WARNING**.

**EXAMPLES:**
ERRAXIS : WARNING
Returns the axis which created the warning and the warning number for task 1-7.

# WNDGS — Motion Parameter

**ACTION:** Enables or disable a motor drive.

**PROGRAM SYNTAX:**
WNDGS(axis)=number cr
WNDGS=number1, ... ,number8 cr
WNDGS(axis) cr
WNDGS cr

**REMARKS:** See Programming Command **WNDGS**.

**EXAMPLES:**
WNDGS(2)=1
Sets the WNDGS state to 1 on axis 2.

WNDGS=0,,1
Sets the WNDGS state on axis 1 to a 0, axis 2 is unchanged and axis 3 WNDGS state is 1.

WNDGS(2)
Returns the winding state for axis 2.

WNDGS
Returns the winding state for all assigned axes.

# "XON XOFF" — Miscellaneous Command

**ACTION:** The XON/XOFF command is a serial communication protocol, executed in software, that allows communications between two devices without the need for additional hardware control. The protocol is used for controlling the flow of data between the Control and another device.

**COMMAND SYNTAX:**
Xon (ASCII 17)
Xoff (ASCII 19)

**REMARKS:** The Xoff character is used to stop the transmission of RS232 or RS485 characters. When one device sends an Xoff to the other device, it is telling the other to stop transmitting characters. The transmitting device should comply with the request.

The Xon character is used to resume transmission of the RS232 or RS485 characters. When one device sends an Xon character to the other, it is signifying that it is ready to receive more characters.

The MX2000 controller sends an Xoff character when the host buffer gets within 80 characters from being full and a minimum of one CR or LF has been received. The receiver will continue to receive characters until the buffer is full. The controller will than issues an Xon character when the host buffer has only 25 characters left in the buffer.

This page left intentionally blank

# Section 8
# Following

## 8.1 - Following Description

The controller has the ability to position follow numerous axes from a single master device. The following features are listed below.

- Flexible Follower definition.
- Programmable follower ratio.
- Three types of following motions can be performed, (JOG, MOVE and MOVEREG).
- Programmable Follower motion trigger.
- Programmable Delay Distance before Follower motion.
- Programmable Follower Acceleration distance.
- Programmable Follower Deceleration distance.
- Positional advance/recede cycles can be performed during a FOLJOG cycle.

## 8.1.1 - Follower Definition

The initialization of the follower requires the follower axes as well as the follower source to be defined. This is accomplished using the FOLINPUT command. When this command is encountered during program execution it enables following.

**Note: If a new follower definition command is encountered during program execution it will become the follower definition.**

**Command Syntax:**

FOLINPUT(Axis,…,Axis) = ACTSPD(Axis)
FOLINPUT(Axis,…,Axis) = ENCSPD(Axis)
FOLINPUT(Axis,…,Axis)=ANALOG(b0n)*exp
FOLINPUT(Axis,…,Axis)=variable

**Axis** defines the follower axes. These axes must be numeric values and be assigned to the task this command is being used in.

**Exp** may be an **equation**, **variable**, **command** and/or a **constant**.

**equation** operators are limited to multiply, add and subtract.

**variable** can be a LOCAL or COMMON variable.

**command** listing:
ACTSPD(axis) commanded velocity of an axis
ENCSPD(axis) encoder velocity of an axis
ANALOG(b0n) analog input voltage
- 1=A side for analog input
- 2=B side for analog input
- Board # of dual axis board

## 8.1.1.1 - Analog Following

An analog input with a center frequency and a deviation frequency for a 10 volt input can be defined as the master source for following.

**Command Syntax**:

**FOLINPUT(axis, ... ,axis) =**
**(ANALOG (b0n) * .1 * DevFreq) + CenterFreq**

**ANALOG(b0n)** defines the analog source.
- 1=A side for analog input
- 2=B side for analog input
- Board # of dual axes board

The **.1 * DevFreq** defines the velocity change per analog input volt in Units/second. **DevFreq** can be a variable or a constant.

The **CenterFreq** variable defines the 0 volt input velocity in Units/sec. The **ACTSPD** or **ENCSPD c**ommands can be substituted for the **CenterFreq** variable.

## 8.1.1.2 - Encoder Following

An Encoder input can be defined as the master source. If the master axis is a stepper axis it must configured as a **closed loop stepper** with the **error action** set to **disabled.**

**Command Syntax:**

**FOLINPUT (axis, ... ,axis)=ENCSPD (axis)**

The **ENCSPD (axis)** defines the master encoder axis (1-8) and can be assigned to any task. A mathematical operators and/or Constant can be used in conjunction with **ENCSPD** if desired.

## 8.1.1.3 – Command & Variable Following

The Master source can be defined by specific basic command or variable. The commands are: **ACTSPD** and **ENCSPD**. The variable can be a COMMON or LOCAL variable.

**Command Syntax:**

**FOLINPUT(axis, ... ,axis)=ACTSPD(MASTER)**
**FOLINPUT(axis, ... ,axis)=ENCSPD(MASTER)**
**FOLINPUT(axis, ... ,axis)=SpeedControl**

**axis** specifies the number of the following axis.

**MASTER** is defined for a particular axis using the #DEFINE statement.

**ACTSPD** & **ENCSPD** may have a mathematical operator and/or Constants used in conjunction with these commands.
**SpeedControl** can be a local variable or a shared variable. Mathematical operators and/ or Constants

can be used in conjunction with this variable as well. This expression specifies the velocity of the master source in units/seconds.

## 8.1.2 - Following Ratio

The ratio of the follower axis to the master is specified by the **FOLRATIO** command. A value of 1.0 represents 100% of master. Any REAL value can be used and a negative value will be converted to its absolute value. The ratio of the follower can be changed during motion but will only become effective when the follower is in synchronization with the master device. The rate at which the ratio will change is controlled by the **FOLRATIOINC** command. This command specifies the following ratio change per second. The maximum recommended ratio is 10.0.

**Command Syntax:**

FOLRATIO(axis) = exp
FOLRATIO = exp, … , exp
FOLRATIO(axis, … , axis) = exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the following axis ratio to the master velocity.

**Command Syntax:**

FOLRATIOINC = exp, … , exp
FOLRATIOINC(axis, … , axis) = exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the ratio acceleration rate in ratio increment per second.

Example:

```
FOLRATIO(2)=.5        ' 50% of master velocity
FOLRATIOINC(2)= 5 ' 500%  change per second
                        is the rate to achieve a
                        new folratio during
                        motion.
```

**Note: If folratio(2)=1 is commanded during motion it would take .1 seconds to achieve 100% ((100-50)/500).**

## 8.1.3 - Follower Motions

There are three following motion commands **FOLJOG**, **FOLMOVE** and **FOLMOVEREG**.

**FOLJOG** commands continuous motion and allows a positional advance/recede cycle to be performed. The **STOP** command is used to stop this cycle.

**Command Syntax:**

FOLJOG(axis) = exp
FOLJOG = exp, … , exp
FOLJOG(axis, … , axis) = exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the direction of travel for the follower.

**FOLMOVE** moves the follower a specific distance.

**Command Syntax:**

FOLMOVE(axis) = exp
FOLMOVE = exp, … , exp
FOLMOVE(axis, … , axis) = exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the follower distance to travel.

**FOLMOVEREG** performs a mark registration move once a registration trigger occurs.

**Command Syntax:**

FOLMOVEREG(axis) = exp
FOLMOVEREG = exp, … , exp
FOLMOVEREG(axis, … , axis) = exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the registration distance for the follower axis.

**STOP** command can be used to stop any of the above motion.

**Command Syntax:**

STOP(axis)
STOP=exp, … , exp
STOP(axis, … , axis)

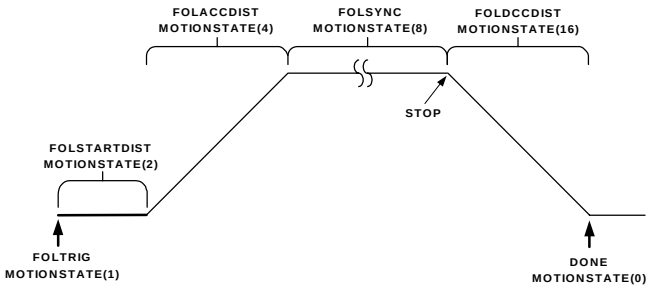
**axis** specifies the number of the following axis.

**exp** stops the designated axes.

## 8.1.4 - Basic Following States

The basic following states for all motion commands consist of a **wait for trigger** state, **wait for distance** state, **Acceleration** state, **Constant** state, **Deceleration** state and **Done** state. These states are depicted in the figure below.

**Basic Following States**




## 8.1.4.1 - Following Trigger

A starting trigger for follower motion can be programmed with the FOLTRIG command. If the FOLTRIG value of a follower axis is non-zero, the follower motion will not begin until the specified trigger condition is met.

The trigger choices are:

0 no trigger
1 Event1 input on closure
2 Event2 input on closure
3 Event1 input on opening
4 Event2 input on opening

The Event inputs are located on the follower axis.

**Command Syntax:**

FOLTRIG(axis)=exp
FOLTRIG=exp, … , exp
FOLTRIG(axis, … , axis)=exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the starting trigger value.

## 8.1.4.2 - Follower Start Delay Distance

A start distance delay can be introduced after the follower trigger condition is met using the **FOLSTARTDIST** command. The master must travel the programmed distance before the follower axis motion begins.

**Command Syntax:**

FOLSTARTDIST(axis)=exp
FOLSTARTDIST=exp, … , exp
FOLSTARTDIST(axis, … , axis)=exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the master travel distance in units.

## 8.1.4.3 - Follower Acceleration

The follower acceleration rate to initially synchronize with the master device at motion start is controlled by the **FOLACCDIST** command. This command defines the distance, in units, that the master device must travel for the follower to synchronize with it.

The follower velocity starts at 0 and ramps linearly to the master speed. The average speed for the follower is 50% of the master during this time thus, the follower distance traveled during acceleration is (.5 * FOLACCDIST * FOLRATIO).

**Command Syntax:**

FOLACCDIST(axis) = exp
FOLACCDIST = exp, … , exp
FOLACCDIST(axis, … , axis) = exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the master travel distance in units.

## 8.1.4.4 - Follower Synchronization

The follower is considered in Synchronization when the follower velocity matches the master velocity times the following ratio of the follower axis. This synchronization state can be monitored using the FOLSYNC or MOTIONSTATE command.

**Command Syntax:**

FOLSYNC(axis) - used in an expression
MOTIONSTATE(axis) - used in an expression

**axis** specifies the number of the following axis.

Each follower axis contains a register that indicates the current state of the follower. The individual follower states are defined as a series of unique numbers 0, 1, 2, 4, 8, 16, 32, 64, 128, 256 and 512. The MOTIONSTATE command is used to return the current follower state number.

## 8.1.4.5 - Follower Deceleration

The follower Deceleration rate is controlled by the **FOLDCCDIST** command. This command defines the distance, in units, that the master device must travel for the follower to stop and terminate motion.

**Command Syntax:**

FOLDCCDIST(axis) = exp
FOLDCCDIST = exp, … , exp
FOLDCCDIST(axis, … , axis) = exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the master travel distance in units.

The follower velocity starts at master velocity and ramps linearly to 0. The average speed for the follower is 50% of the master during this time thus, the follower distance traveled during deceleration is (.5 * FOLDCCDIST * FOLRATIO).

**Note: Issuing a STOP command can stop Follower motion. The master will travel the FOLDCCDIST before the follower terminates motion. The exception would be if the master velocity reaches zero before the FOLDCCDIST has been traveled. It is recommended that the WAITDONE command be used to allow a complete stop prior to executing the next line of code.**

## 8.1.5 - Advance/Recede cycle

The follower position can only be advanced or receded during a FOLJOG cycle. The issuing of a FOLOFFSET commands a positional offset to be performed when the follower and master velocities are in synchronization. The FOLOFFSET cycle consists of two parts a synchronization portion and an offset portion. If material is to be cut it is done during the synchronization portion of this cycle.

**Command Syntax:**

FOLOFFSET(axis)=exp
FOLOFFSET=exp1, … , exp8
FOLOFFSET (axis, … , axis)=exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the follower travel distance in units. If the travel distance is positive a positional advance cycle will be performed. If the travel distance is negative a positional recede cycle will be performed.

## 8.1.5.1 - Offset Wait Distance

An Offset wait distance can be programmed at the start of an advance/recede cycle via the FOLSYNCDIST command. This wait distance can be used as the cutting distance in a flying shear application, material and rotary knife synchronization distance or any other operation requiring synchronization at the beginning of an advance/recede cycle.

**Command Syntax:**

FOLSYNCDIST(axis) = exp
FOLSYNCDIST = exp, … , exp
FOLSYNCDIST (axis, … , axis) = exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the master device travel distance in units.

## 8.1.5.2 - Offset Velocity Limits

A velocity limit can be imposed on an advance/recede cycle. The limit is specified as a ratio of the master device velocity. The advance cycle velocity limit is specified by the FOLMAXRATIO command and must be a positive number and a value greater than the FOLRATIO value.

**Command Syntax:**

FOLMAXRATIO(axis) = exp
FOLMAXRATIO = exp, … , exp
FOLMAXRATIO (axis, … , axis)=exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the maximum velocity.

The recede cycle velocity limit is specified by the FOLMINRATIO command and may be a negative value if the follower direction is allowed to reverse. If the value is positive it must be less than the FOLRATIO value.

**Command Syntax:**

FOLMINRATIO(axis) = exp
FOLMINRATIO = exp1, … , exp8
FOLMINRATIO(axis, … , axis) = exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the minimum velocity allowed.

## 8.1.5.3 - Offset Distances

The distance the follower will advance or recede from the master during an offset cycle is specified by the FOLOFFSET command. The distance traveled by the master during the advance/recede part of an offset cycle is specified by the FOLOFFSETDIST command. If the FOLOFFSET distance sign is positive the follower will advance by this distance and if the sign is negative the follower will recede by this distance.

**Command Syntax:**

FOLOFFSETDIST(axis) = exp
FOLOFFSETDIST = exp, … , exp
FOLOFFSETDIST(axis,…, axis)=exp, … , exp

**axis** specifies the number of the following axis.

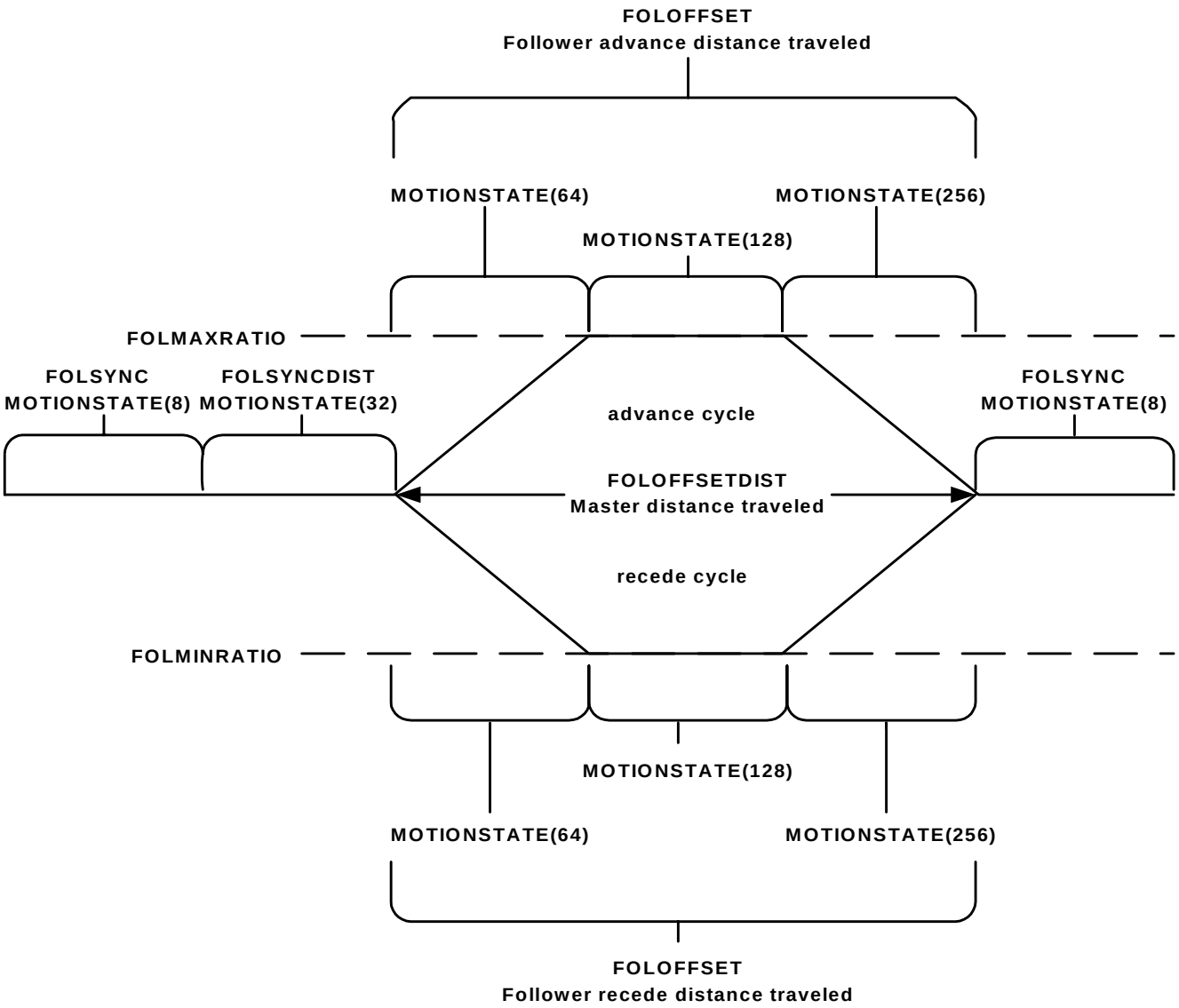
**exp** specifies the master device travel distance in units during an offset cycle.

**Command Syntax:**

FOLOFFSET(axis) = exp
FOLOFFSET = exp1, … , exp8
FOLOFFSET(axis, … , axis) = exp, … , exp

**axis** specifies the number of the following axis.

**exp** specifies the follower travel distance in units during an offset cycle.




# Foloffset Cycle Velocity Profile

## 8.1.6 - Following program Template

The following template can be used as a guideline for writing a follower program.

```
'****** Follower Parameter initialization *********
#DEFINE   FOLLOWER   1
#DEFINE   MASTER      2

FOLTRIG(FOLLOWER)=0
FOLSTARTDIST(FOLLOWER)=0
FOLACCDIST(FOLLOWER)=expression
FOLDCCDIST(FOLLOWER)=expression
FOLRATIO(FOLLOWER)=1
FOLRATIOINC(FOLLOWER)=10
FOLMAXRATIO(FOLLOWER)=2
FOLMINRATIO(FOLLOWER)=0
' *** Follower and Master definitions (chose one) ***
FOLINPUT(FOLLOWER)=variable
FOLINPUT(FOLLOWER)=ACTSPD(MASTER)
FOLINPUT(FOLLOWER)=ENCSPD(MASTER)
FOLINPUT(FOLLOWER)=ANALOG(b0n)*expression
' ********** advance/recede motion **********
FOLJOG(FOLLOWER)==1

DO
LOOP UNTIL FOLSYNC(FOLLOWER)=1

DO
  FOLSYNCDIST(FOLLOWER)=expression
  FOLOFFSETDIST(FOLLOWER)=expression
  FOLOFFSET(FOLLOWER)=expression

  DO
  LOOP UNTIL MOTIONSTATE(FOLLOWER)=32

     'cut material statements (in synchronization)

  DO
  LOOP UNTIL MOTIONSTATE(FOLLOWER)<>32
LOOP UNTIL EXIN(100)=1

DO : LOOP UNTIL FOLSYNC(FOLLOWER)=1
STOP(FOLLOWER)
WAITDONE(FOLLOWER)
' ********** Follower move cycle **********
FOLMOVE(FOLLOWER)=expression
WAITDONE(FOLLOWER)
' ******* Follower mark registration cycle ********
 REGLIMIT(FOLLOWER)=expression
FOLMOVEREG(FOLLOWER)=expression
WAITDONE(FOLLOWER)
```

## 8.1.7 – Distance Measurements

The distance between items can be measured by using the combination of the CAPTURE, CAPPOS and DELTACAPPOS commands. These commands can be helpful when uniform spacing between items is required.

The CAPTURE command arms a position capture cycle or returns the current capture status. The captured position can be read via the CAPPOS command. The distance between capture positions can be read via the DELTACAPPOS command.

**Command Syntax:**

CAPTURE(axis)=exp
CAPTURE=exp1, … , exp8
CAPTURE(axis, … , axis)=exp, … , exp
CAPTURE(axis) – used in an expression

**axis** specifies the number of the axis.

**exp** specifies the trigger condition.

**Command Syntax:**

CAPPOS(axis) – used in an expression

**axis** specifies the number of the axis.

**Command Syntax:**

DELTACAPPOS(axis) – used in an expression

**axis** specifies the number of the axis.

## 8.1.8 - Cut to length Example

The cutting cycle requires that the material and cutter be in synchronization when the material is being cut and that the cutter be returned to the next cutting position.

**Example:** The material is to be cut in 11 units lengths. The cutting portion of the cycle will take 1 second and the material is moving at 1 unit/second. The FOLRATIO is assumed to be 1.0.

This cutting cycle is accomplished by using the FOLOFFSET command. The FOLOFFSET cycle consists of a synchronization section (FOLSYNCDIST) and offset travel section (FOLOFFSET and FOLOFFSETDIST). The FOLSYNCDIST command is used to define the material cutting distance and the FOLOFFSET, FOLOFFSETDIST for defining the next cutting position.

The cut length is the summation of the FOLSYNCDIST and the FOLOFFSETDIST distances. This is the incremental distance traveled by the master during the cycle.

The FOLOFFSET distance is the negation of the cut length. This is the recede distance traveled by the follower during the offset cycle.

# Cut to length Cycle

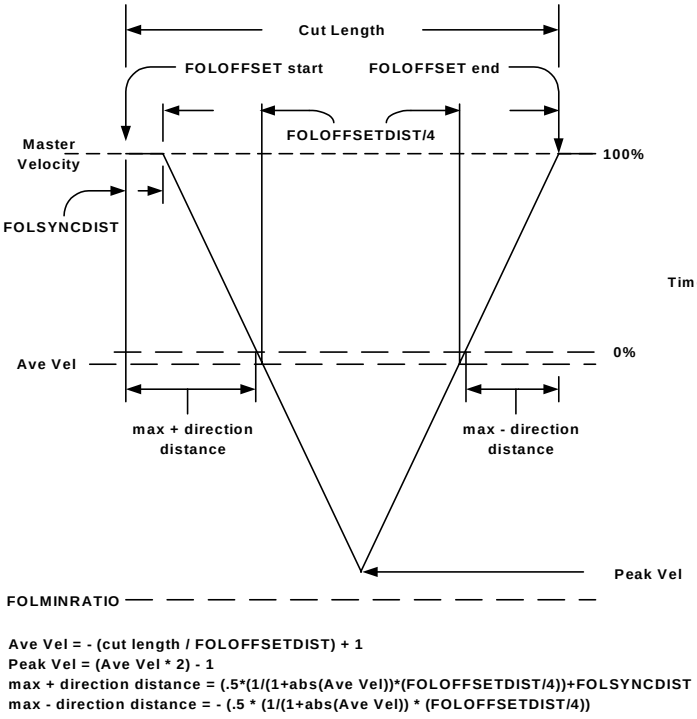



**Fig 1. Shows the Velocity Profile for this application**

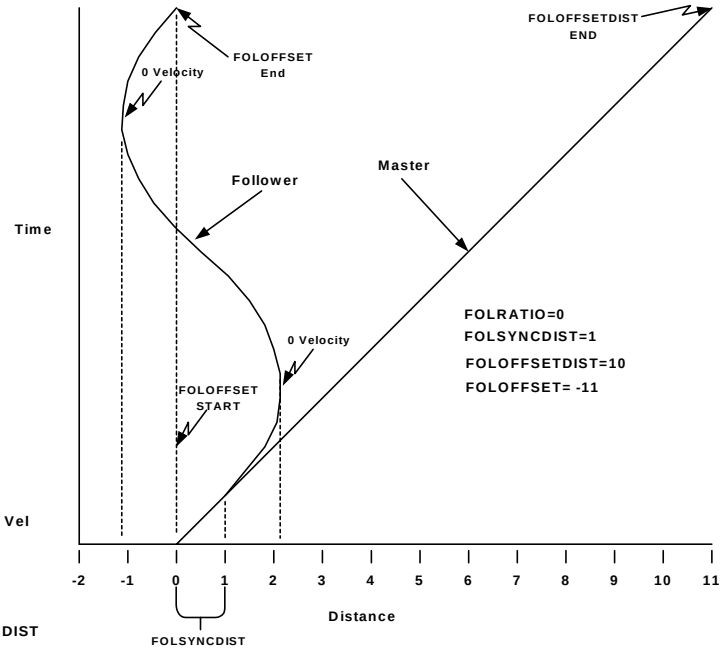



**Fig 2. Shows the Positional Profile for this application.**

Ave Vel = -(11/10) + 1 = -.1 (-10%)
Peak Vel = (-.1 * 2) – 1 = -1.2 (-120%)
Max + direction distance traveled =
(.5 * (1/(1 + .1)) * (10/4)) + 1 = +2.136 units
Max – direction distance traveled =
- (.5 * (1/(1 + .1)) * (10/4)) = -1.136 units

**Notes:**
**1) The cutting position is considered the 0 position on the Positional Profile.**
**2) The longest cut length creates the maximum distance excursion around the cutting position.**
**3) The shortest cut length creates the highest Peak Velocity.**

## 8.1.8.1 - Cut to Length Program Example

```
#DEFINE     MASTER              1     ‘master axis number
#DEFINE     FOLLOWER            2     ‘follower axis number
#DEFINE     SYNC_DIST_FLAG      32    ‘wait for Sync distance
#DEFINE     CUT_LENGTH          11    ‘material cutting length
#DEFINE     IN_SYNC_DIST        1     ‘distance master travels in sync with follower

‘********* initialize follower parameters ************************
ABSPOS(MASTER,FOLLOWER)=0,0          ‘ set starting position to 0
PROFILE(FOLLOWER)=16                 ‘ profile set to S Curve with setting of 16
FOLRATIO(FOLLOWER)=1.0               ‘ follow at 100% of master velocity
FOLTRIG(FOLLOWER)=0                  ‘ no trigger required
FOLSTARTDIST(FOLLOWER)=0             ‘ no delay distance
FOLACCDIST(FOLLOWER)=1               ‘ master travels 1 unit before follower matches master velocity
FOLDCCDIST(FOLLOWER)=1               ‘ master travels 1 unit before follower stops
FOLSYNCDIST(FOLLOWER)=IN_SYNC_DIST  ‘distance master travels in sync with follower
FOLMINRATIO(FOLLOWER)=-1.5           ‘ offset velocity allowed to reverse up to 150% of master
SPEED(MASTER)=1                      ‘ master velocity 1 units/sec

‘******** define and activate follower axis ***********************
JOG(MASTER)=1                        ‘ start master axis
FOLINPUT(FOLLOWER)=ACTSPD(MASTER)
FOLJOG(FOLLOWER)=1                   ‘ follow in the same direction as master source

DO : LOOP UNTIL FOLSYNC(FOLLOWER)=1 ‘wait for initial velocity synchronization

‘******** perform cut to length cycle ****************************
DO
    FOLOFFSETDIST(FOLLOWER) = CUT_LENGTH – IN_SYNC_DIST ‘ setup offset cycle
    FOLOFFSET(FOLLOWER) = -CUT_LENGTH                   ‘ command offset cycle

    ‘***** wait for offset cycle in synchronization portion to begin *****
    DO : LOOP UNTIL MOTIONSTATE(FOLLOWER) = SYNC_DIST_FLAG
        ‘ Material cutting statements
    ‘***** wait for offset portion of cycle to begin *********
    DO : LOOP UNTIL MOTIONSTATE(FOLLOWER) <> SYNC_DIST_FLAG
LOOP UNTIL EXIN(101)=1               ‘ exit on stop request

‘****** cut to length cycle termination requested *******
‘****** wait for velocity synchronization *****
DO : LOOP UNTIL FOLSYNC(FOLLOWER) = 1

‘******* stop follower and wait for motion stopped **********************
‘******* follower moves FOLSYNCDIST before deceleration occurs ********
STOP(FOLLOWER)
WAITDONE(FOLLOWER)

‘******* stop master and wait for motion stopped **********************
STOP(MASTER)
WAITDONE(MASTER)

‘******** move to starting position of follower ***********************
POSMODE(FOLLOWER)=1
MOVE(FOLLOWER)=0
WAITDONE(FOLLOWER)
END
```

## 8.1.9 - Rotating Knife Examples

A knife located on the follower axis is synchronized with the material controlled by the master axis. The knife is located at 12 o'clock initially and its cutting area is located 36° on each side of 6 o'clock. Thus the Knife must be in synchronization with the master in the material cutting area. This system is set up such that one revolution of the master and follower axes is equivalent to 1 unit. The rotating knife axis is not allowed to reverse for safety purposes.

The FOLACCDIST command value is used to synchronize the Knife with material at startup. This is accomplished by setting the FOLACCDIST to twice the distance required to move the follower the initial 144° ((144/360)*2= .8 units) to the cutting area.

The FOLSYNCDIST command is used to control the cutting area of the knife. Since 72° of cutting area is required the FOLSYNCDIST is set to (72/360) .2 units.

The FOLOFFSET command controls the moving of the knife to the new cutting position. Thus the FOLOFFSET = (1 – cut length)) and varies with different cut lengths. The FOLOFFSET command can create an advance or recede cycle depending on the cut length.

The cut length is controlled by the distance traveled by FOLSYNCDIST + FOLOFFSETDIST. Thus the FOLOFFSETDIST = cut length – FOLSYNCDIST and varies with different cut lengths.

The FOLMINRATIO command controls the minimum speed allowed by the follower during an Offset cycle. Since the follower axis is not allowed to reverse direction the FOLMINRATIO must be set to 0. This command comes into play whenever cut length is greater than 1.

The FOLMAXRATIO command controls the maximum speed allowed by the follower during an Offset cycle. This command comes into play whenever the cut length is less than 1. The maximum attainable speed for the follower axis limits the minimum cutting distance in this application. Although this speed can be limited by the FOLMAXRATIO the slope of the acceleration/deceleration becomes steeper as the cut distance becomes shorter.

The FOLMAXRATIO value must be set in between the instantaneous rate and the triangular rate that is calculated as follows:

Instantaneous rate = FOLRATIO *
**((FOLOFFSET / FOLOFFSETDIST) + 1)**

Triangular rate = FOLRATIO *
**(((FOLOFFSET / FOLOFFSETDIST) * 2) + 1)**

where :
FOLOFFSET = 1 – cut length
FOLOFFSETDIST = cut length – FOLSYNCDIST

### 8.1.9.1 Rotating Knife Cycle

When a FOLJOG is commanded the follower axis ramps up to match the master velocity. The distance traveled by the follower is .4 units (144°). It is now in position 1 of the Rotary Knife Cycle. This is the starting position for cutting the material.

An offset cycle is commanded and the follower and master move .2 units in synchronization, 72° of motion on the follower axis. The material is cut during this portion of the cycle. It is now in position 2 of the Rotary Knife Cycle.

The offset portion of the cycle is now executed. The master moves the FOLOFFSETDIST distance and the follower end up at the starting position for cutting the material (144°). It is now in position 3 of the Rotary Knife Cycle. The material has now moved the cut length.

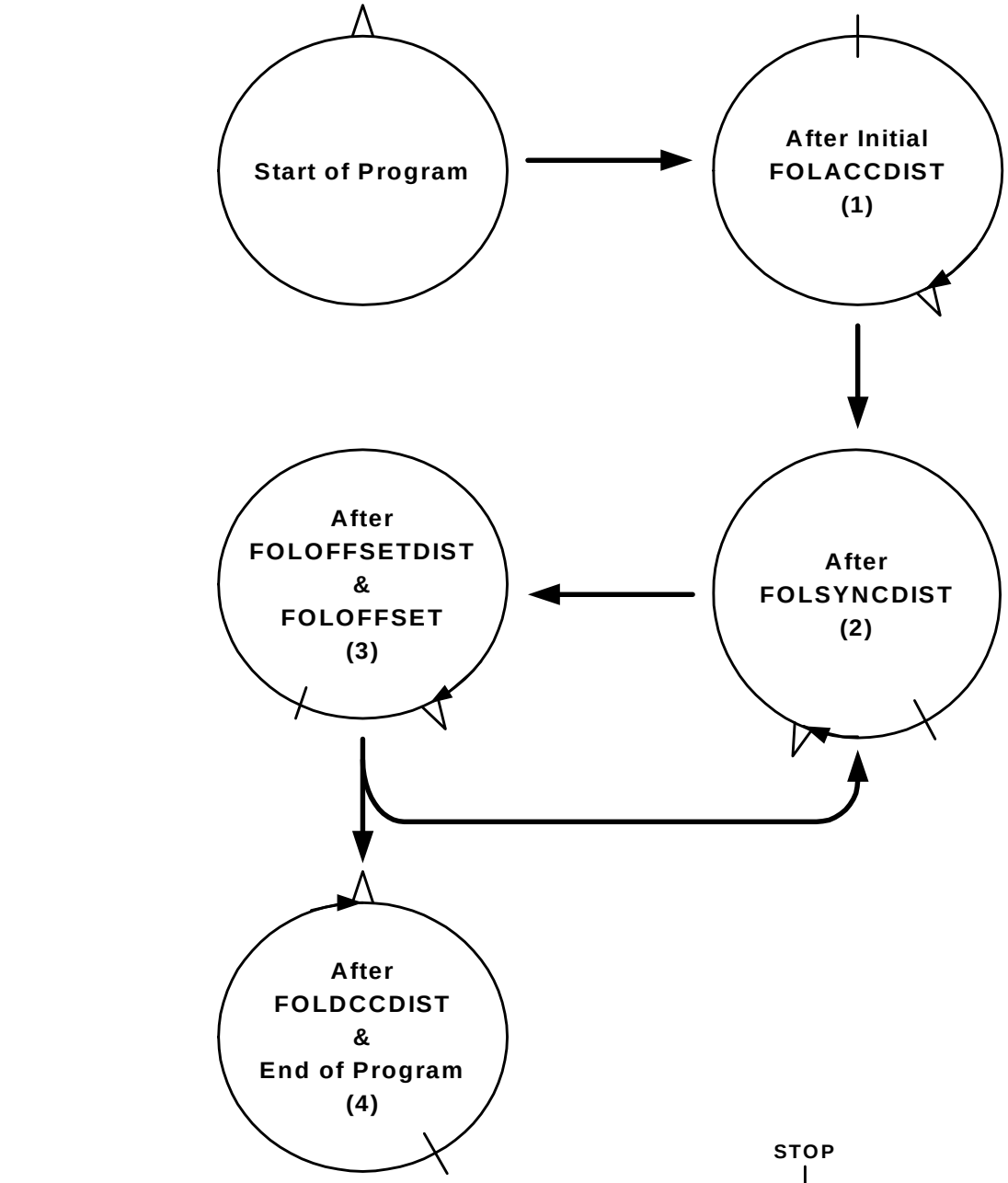
This cutting cycle continues until EXIN(101)=1. Then the program now waits for the last offset cycle to complete. It is now in position 3 of the Rotary Knife Cycle. A follower axis stop is now commanded.

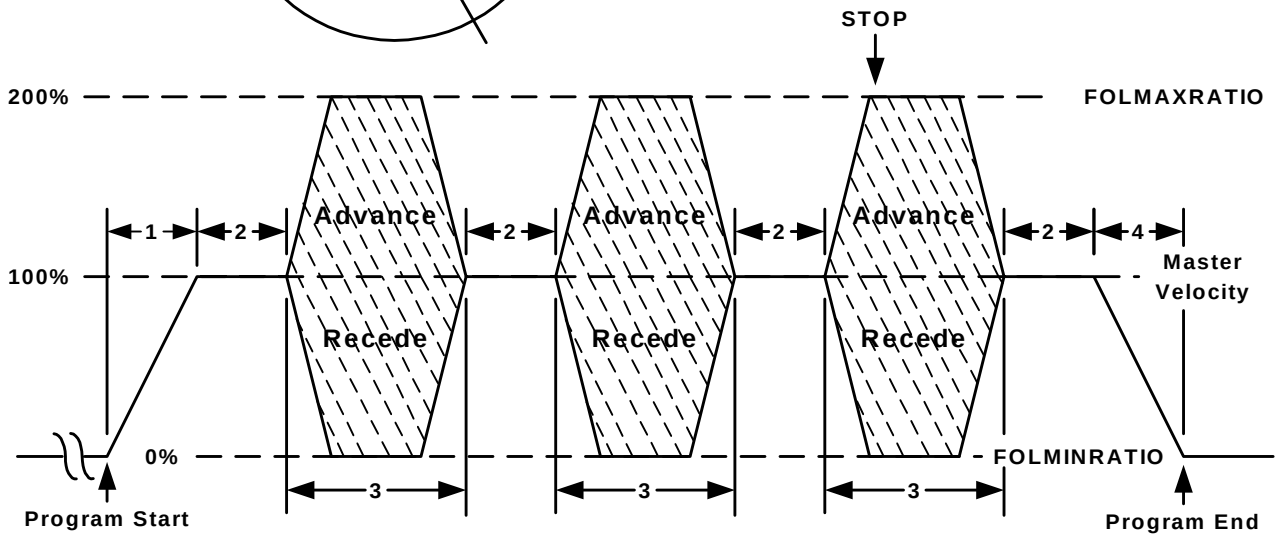
Both axes now travel the FOLSYNCDIST. It is now in position 2 of the Rotary Knife Cycle.

The follower now decelerates to a stop. The distance traveled by the follower is .4 units (144°). This puts the knife back at 12 o'clock, which is the starting position.

The master axis is now commanded to stop ending the cutting cycle.

## Rotary Knife Cycle

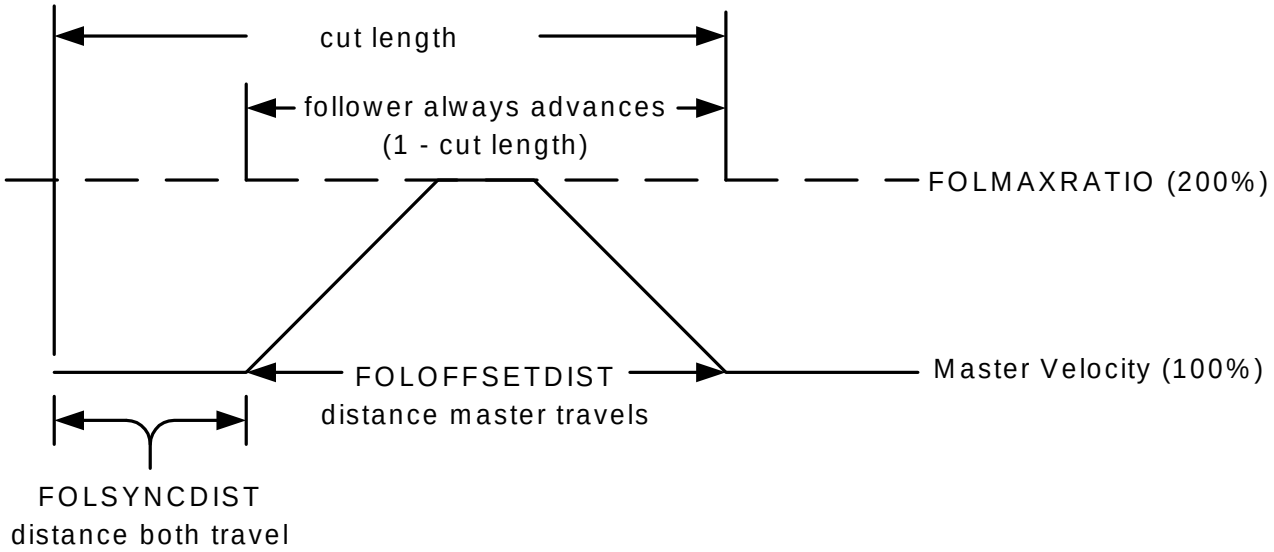


Example 1: Material is cut in .7 units lengths and the knife cutting area is 72° (.2 units).

| | | |
|---|---|---|
| FOLRATIO | 1 | |
| FOLSYNCDIST | (72/360) = .2 | |
| FOLOFFSET | .3 | (1 – cut length) |
| FOLOFFSETDIST | .5 | (cut length – FOLSYNCDIST) |
| Instantaneous rate = ((.3 / .5) + 1) * 1 = 1.6 | | (160%) |
| Triangular rate = (((.3 / .5) * 2)) + 1 = 2.2 | | (220%) |
| FOLMAXRATIO | 2 | (1.6 to 2.2) |

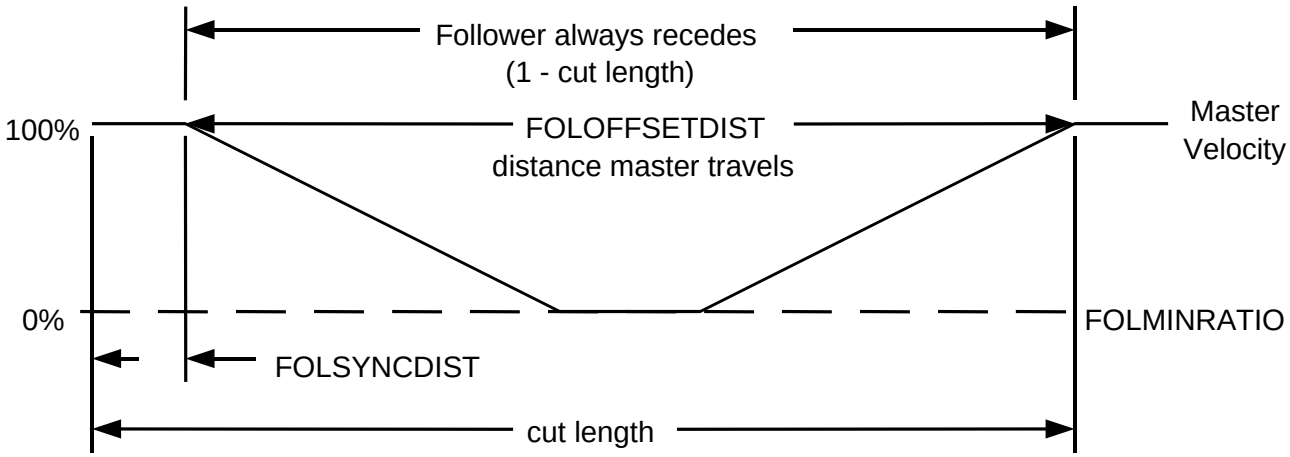
Distance traveled by follower during offset is always (1 – cut length)




Example 2: Material is cut in 2.2 unit lengths and the knife cutting area is 72° (.2 units).

| | | |
|---|---|---|
| FOLRATIO | 1 | |
| FOLSYNCDIST | .2 | |
| FOLOFFSET | -1.2 | (1 – cut length) |
| FOLOFFSETDIST | 2.0 | (cut length – FOLSYNCDIST) |
| FOLMINRATIO | 0 | |

Distance traveled by follower during offset is always (1 – cut length)



## 8.1.9.2 - Rotating Knife Program Example 1 (advance cycle)

```
#DEFINE     MASTER              1       ‘master axis number
#DEFINE     FOLLOWER            2       ‘follower axis number
#DEFINE     IN_SYNC             8       ‘wait for in sync state
#DEFINE     SYNC_DIST_FLAG      32      ‘wait for Sync distance state
#DEFINE     CUT_LENGTH          .7      ‘material cutting length

‘********* initialize follower parameters *************************
FOLSYNCDIST(FOLLOWER)=72/360          ‘ distance the master must travel for material to be cut
ABSPOS(MASTER,FOLLOWER)=0,0           ‘set position to 0
PROFILE(FOLLOWER)=16                  ‘ profile set to S Curve with a setting of 16
FOLRATIO(FOLLOWER)=1.0                ‘ follow at 100% of master velocity
FOLTRIG(FOLLOWER)=0                   ‘ no trigger required
FOLSTARTDIST(FOLLOWER)=0              ‘ no delay distance
FOLACCDIST(FOLLOWER)=(144/360) * 2    ‘align knife with cutting surface
FOLDCCDIST(FOLLOWER)=(144/360) * 2    ‘ align knife at 12 o’clock
FOLMAXRATIO(FOLLOWER)=2.0             ‘ maximum offset velocity is 200% of master
SPEED(MASTER)=5                       ‘ master velocity set to 5 units/sec

‘********* define and activate follower axis ************************
JOG(MASTER)=1                         ‘ start master axis
FOLINPUT(FOLLOWER)=ACTSPD(MASTER)
FOLJOG(FOLLOWER)=1                    ‘ follow master source in same direction

DO : LOOP UNTIL FOLSYNC(FOLLOWER)=1 ‘wait for initial velocity synchronization

DO
   FOLOFFSETDIST(FOLLOWER)= CUT_LENGTH - FOLSYNCDIST(FOLLOWER) ‘setup offset cycle
   FOLOFFSET(FOLLOWER)= 1 - CUT_LENGTH                          ‘command offset cycle

   ‘******** wait for offset cycle in synchronization portion to begin ********
   DO : LOOP UNTIL MOTIONSTATE(FOLLOWER)=SYNC_DIST_FLAG
   ‘******** wait for offset portion of cycle to begin ************************
   DO:LOOP UNTIL MOTIONSTATE(FOLLOWER) <> SYNC_DIST_FLAG

LOOP UNTIL EXIN(101)=1                ‘ wait for stop input

‘********* rotary cutter cycle termination request **********************
‘********* wait for velocity synchronization ***************************
DO:LOOP UNTIL MOTIONSTATE(FOLLOWER)=IN_SYNC

‘******** stop follower and wait for motion to stop
‘******* follower moves FOLSYNCDIST before deceleration occurs ********
STOP(FOLLOWER)                        ‘ motion stops after master travels the FOLDCCDIST.
WAITDONE(FOLLOWER)                    ‘ wait for follower axis to stop

‘******** stop master and wait for motion to stop
STOP(MASTER)
WAITDONE(MASTER)
END
```

## 8.1.9.3 - Rotating Knife Program Example 2 (recede cycle)

```
#DEFINE     MASTER              1       ‘master axis number
#DEFINE     FOLLOWER            2       ‘follower axis number
#DEFINE     IN_SYNC             8
#DEFINE     SYNC_DIST_FLAG      32      ‘wait for Sync distance state
#DEFINE     CUT_LENGTH          2.2     ‘material cutting length in units

‘******* initialize follower parameters ************************
FOLSYNCDIST(FOLLOWER)=72/360            ‘ distance the master must travel for material to be cut
ABSPOS(MASTER,FOLLOWER)=0,0             ‘ set position to 0
PROFILE(FOLLOWER)=16                    ‘ profile set to S Curve with a setting of 16
FOLRATIO(FOLLOWER)=1                    ‘ follow at 100% of master velocity
FOLTRIG(FOLLOWER)=0                     ‘ no trigger required
FOLSTARTDIST(FOLLOWER)=0                ‘ no delay distance
FOLACCDIST(FOLLOWER)=(144/360) * 2      ‘ align knife with cutting surface 144° from start
FOLDCCDIST(FOLLOWER)=(144/360) * 2      ‘ align knife at 12 o’clock
FOLMINRATIO(FOLLOWER)=0                 ‘ offset cycle not allowed to reverse
SPEED(MASTER)=5                         ‘ master velocity set to 5 units/sec

‘********* define and activate follower axis ***********************
JOG(MASTER)=1                           ‘ start master axis
FOLINPUT(FOLLOWER)=ACTSPD(MASTER)
FOLJOG(FOLLOWER)=1                      ‘ follower in the master direction

‘********* wait for initial velocity synchronization ******************
DO : LOOP UNTIL FOLSYNC(FOLLOWER)=1

‘********* perform rotary cutter cycle *****************************
DO
  FOLOFFSETDIST(FOLLOWER)= CUT_LENGTH - FOLSYNCDIST(FOLLOWER) ‘setup offset cycle
  FOLOFFSET(FOLLOWER)= 1 - CUT_LENGTH                         ‘command offset cycle

  ‘******** wait for offset cycle in synchronization portion to begin ********
  DO:LOOP UNTIL MOTIONSTATE(FOLLOWER)=SYNC_DIST_FLAG
   ‘******** wait for offset portion of cycle to begin ************************
  DO:LOOP UNTIL MOTIONSTATE(FOLLOWER) <> SYNC_DIST_FLAG

LOOP UNTIL EXIN(101)=1                  ‘ wait for stop input

‘********* rotary cutter cycle termination request *********************
‘********* wait for velocity synchronization **************************
DO:LOOP UNTIL MOTIONSTATE(FOLLOWER)=IN_SYNC

‘******** stop follower and wait for motion to stop
‘******* follower moves FOLSYNCDIST before deceleration occurs ********
STOP(FOLLOWER)
WAITDONE(FOLLOWER)

‘******** stop master and wait for motion to stop
STOP(MASTER)
WAITDONE(MASTER)
END
```

## 8.1.10 - Gear Box Following Example

This type of application only requires a ratio between 2 axes that must be synchronized.

## Program Example

This example simulates a gearbox with a 5:1 reduction.

```
#DEFINE MASTER          1
#DEFINE FOLLOWER        2

‘******* initialize master axis and follower parameters **********
ACCEL(MASTER)=50                        ‘ master axis acceleration  = 50 units/sec2
DECEL(MASTER)=50                        ‘ master axis deceleration = 50 units/sec2
SPEED(MASTER)=5                         ‘ master axis speed = 5 units/sec
PROFILE(MASTER,FOLLOWER)=16,16          ‘ profile set to S Curve with a setting of 16
ABSPOS(MASTER,FOLLOWER)=0,0             ‘ set position to 0
FOLRATIO(FOLLOWER)=0.2                  ‘ follows at 20% of master velocity
FOLSTARTDIST(FOLLOWER)=0                ‘ no delay distance
FOLACCDIST(FOLLOWER)=0                  ‘ no acceleration distance
FOLDCCDIST(FOLLOWER)=0                  ‘ no deceleration distance
FOLTRIG(FOLLOWER)=0                     ‘ no following trigger required

‘********* define and activate follower axis ***********************
FOLINPUT(FOLLOWER)=ACTSPD(MASTER) ‘follower cycle definition
FOLJOG(FOLLOWER)=1                      ‘ start follower axis

‘******** Execute main program *******************************
DO
        ‘ Program statements
LOOP UNTIL EXIN(100)=1                  ‘ wait for program end

STOP(FOLLOWER)                          ‘ motion stops
WAITDONE(FOLLOWER)                      ‘ wait for FOLLOWER axis to stop
END
```

### 8.1.11 – Following Command Listing

# ACTSPD — Trajectory Parameter

**ACTION:** Returns the current commanded velocity of an axis in Units/second.

**PROGRAM SYNTAX:** ACTSPD(axis) - used in an expression

**REMARK:** This command can be used in conjunction with a FOLINPUT command to specify the master source. It can also be used to monitor the current commanded velocity of an axis.

**EXAMPLES:**

```
FOLINPUT(1,3)=ACTSPD(2)
'Sets the current commanded velocity of axis 2 as the master
velocity. Axis 1 and axis 3 are follower axes.

axspd=ACTSPD(2)
'Sets variable axspd to the current commanded velocity of axis 2.
```

# ENCSPD — Trajectory Parameter

**ACTION:** Returns the current encoder speed in units/second.

**PROGRAM SYNTAX:** ENCSPD(axis) - used in an expression

**REMARK:** The encoder speed is monitored at the sample rate selected for the axis. This results in an encoder count/sample time value that is converted to units/second. Since this value is digital and not filtered a velocity, deviations will result.

**EXAMPLE:**

```
X=ENCSPD(2)
Sets variable X to the current encoder speed of axis 2.

outputspd=0                          ' initial value
FOR x=1 TO 10                        ' number of samples
   outputspd=outputspd+ENCSPD(1)     ' sample update
   wait=.001                         ' sample time
NEXT x
outputspd=outputspd/10               ' filtered value

FOLINPUT(1,3)=ENCSPD(2)
' Sets the current encoder velocity of axis 2 as the master source for following. Axis 1 and axis 3 are
follower axes.
```

# FOLINPUT Following Parameter

**ACTION:** This command specifies the follower axes and the master velocity source.

**PROGRAM SYNTAX:** FOLINPUT(axis, ... ,axis)= expression

The axis specifies the follower axes (1-8). These axes must be assigned to the task the FOLINPUT command is used.

The expression specifies the master velocity source for the follower. The expression may be an equation, variable, command or a constant. The mathematical operators that are allowed in the expression are limited to multiply, add and subtract. If a variable is used it can be a LOCAL or COMMON variable. The commands allowed in the expression are limited to: VELOCITY which specifies the velocity of a task, SPEED(axis) which specifies the target velocity of the specified axis, ACTSPD(axis) which specifies the current commanded velocity of the specified axis, ENCSPD(axis) which specifies the encoder (mechanical) velocity of the specified axis, ANALOG(b0n) which specifies the analog input port to use in the expression and ABSPOS(axis) the current absolute position of an axis.

**EXAMPLES:** FOLINPUT(1, 3)=(analog(101) * .1 * DevFreq) +VELOCITY
Analog follow axis board 1 "A input". The follower axes are axis 1 and axis 3. The VELOCITY command controls the center frequency and variable DevFreq controls the 10 volt deviation frequency from the center frequency. The sign of the Analog input voltage control how the deviation frequency is applied to the center frequency (add or subtract). The commands ACTSPD, SPEED or ABSPOS can be substituted for the VELOCITY in the expression if desired.

FOLINPUT(1,3)=ENCSPD(2)
encoder follow axis 1 and axis 3 using the encoder input of axis 2 as the master velocity.

FOLINPUT(2,3)=ACTSPD(1)
Axis 2 and axis 3 follows the actual commanded speed of axis 1.
Note: Commands VELOCITY(1), SPEED(1) or ABSPOS(1) can be substituted for ACTSPD(1).

FOLINPUT(2,3)= MasterSpd
Axis 2 and axis 3 follows the numeric value of variable MasterSpd. A numeric value can also be substituted for variable MasterSpd if desired.

# FOLTRIG Following Parameter

**ACTION:** Defines the follower starting trigger for motion.

**PROGRAM SYNTAX:**

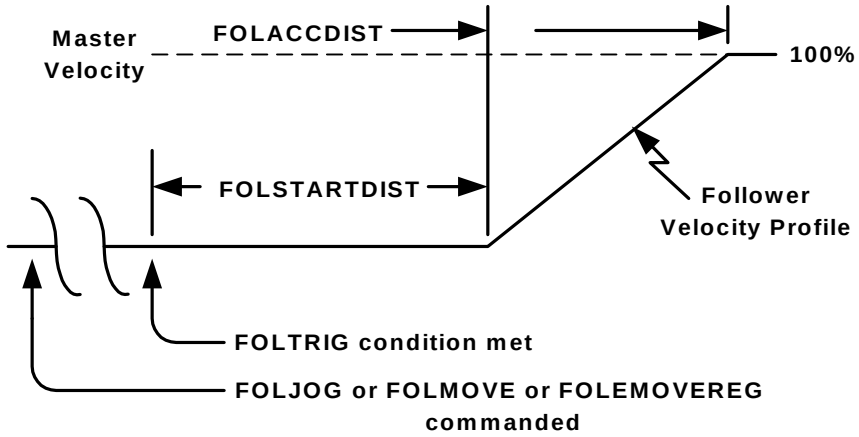
FOLTRIG(axis)=expression
FOLTRIG=expression1, ... , expression8
FOLTRIG(axis, ... ,axis)=expression, ... , expression
FOLTRIG(axis) - used in an expression

**REMARKS:** The axis specifies the number of the following axis (1-8).

The expression specifies the starting trigger.

| Value | Specified Trigger |
|---|---|
| 0 | No trigger |
| 1 | Event 1 on closure |
| 2 | Event 2 on closure |
| 3 | Event 1 on opening |
| 4 | Event 2 on opening |

This command is related to the FOLSTARTDIST command as follows. The follower will not start motion until the FOLSTARTDIST has been traveled by the master source once the trigger condition of the FOLTRIG command has been met.




**EXAMPLES:**

FOLTRIG(1)=2
sets event 2 on closure as the starting trigger for axis 1.

FOLTRIG=0,,3
sets no trigger for axis 1, sets event 1 on opening as starting trigger for axis 3.

FOLTRIG(1,3)=0,3
sets no trigger for axis 1, sets event 1 on opening as starting trigger for axis 3.

# FOLSTARTDIST Following Parameter

**ACTION:**

Specifies a master distance that is used as a delay distance for starting motion. The distance delay starts when the specified starting trigger of FOLTRIG command occurs.

**PROGRAM SYNTAX:**

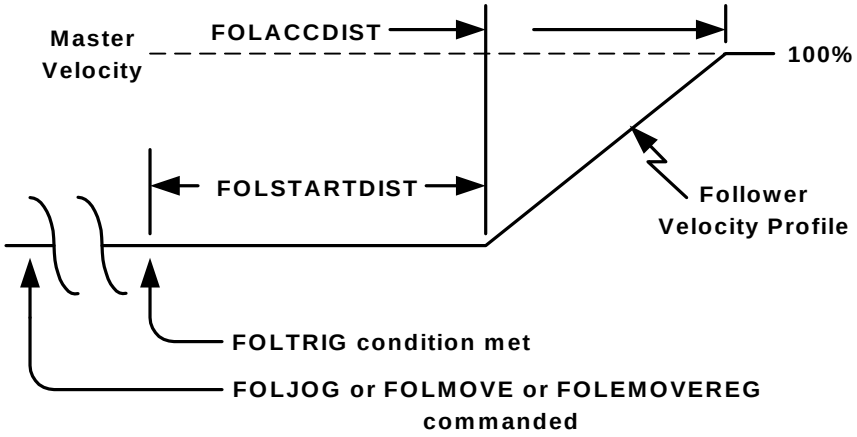
FOLSTARTDIST(axis)=expression
FOLSTARTDIST=expression1, ... , expression8
FOLSTARTDIST(axis) - used in an expression
FOLSTARTDIST(axis, ... .axis)=expression, ... , expression

**REMARKS:**

The axis specifies the number of the following axis (1-8).

The expression specifies the master distance traveled in units.

This command is related to the FOLTRIG command as follows. The follower will not start motion until the FOLSTARTDIST has been traveled by the master source once the trigger condition of the FOLTRIG command has been met.




**EXAMPLE:**

FOLSTARTDIST(1)=1.5
axis 1 master distance delay is 1.5 units before starting motion.

FOLSTARTDIST=1,,3
axis 1 master distance delay is 1 unit and axis 3 master distance delay is 3 units before starting motion.

FOLSTARTDIST(1,4)=1,3
axis 1 master distance delay is 1 unit and axis 4 master distance delay is 3 units before starting motion.

# FOLACCDIST Following Parameter

**ACTION:**

Specifies the master distance traveled for the follower to catch the master velocity after follower motion begins.

**PROGRAM SYNTAX:**

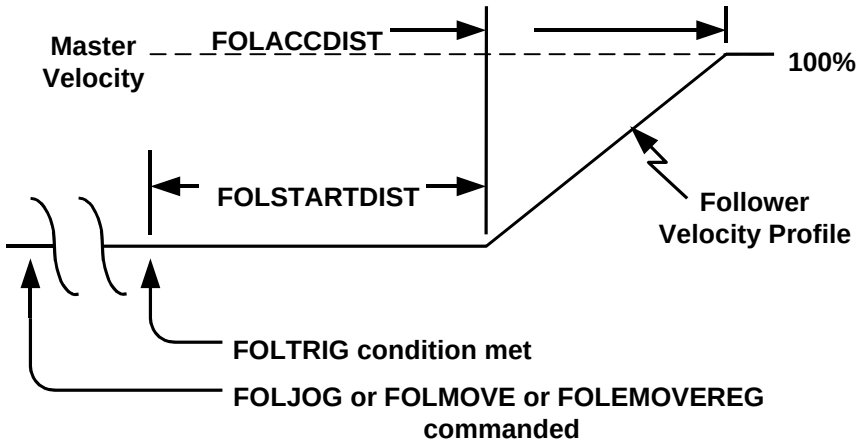
FOLACCDIST(axis)=expression
FOLACCDIST=expression1, ... , expression8
FOLACCDIST(axis, ... ,axis)=expression, ... , expression
FOLACCDIST(axis) - used in an expression

**REMARKS:**

The axis specifies the number of the following axis (1-8).

The expression specifies the master distance to travel in Units.

The follower axis will start motion once the trigger condition is met and the master distance specified by the FOLSTARTDIST command is achieved. Once motion begins the follower will match the master velocity in the specified master distance. The distance traveled by the follower is 50% of the FOLACCDIST distance times the FOLRATIO value.




**EXAMPLES:**

FOLACCDIST(1)=1.5
axis 1 match the master velocity in 1.5 units after starting motion.

FOLACCDIST=1,,3
axis 1 match the master velocity in 1 unit and axis 3 match the master velocity in 3 units after starting motion.

FOLACCDIST(1,4)=1,3
axis 1 match the master velocity in 1 unit, axis 4 match the master velocity in 3 units after starting motion.

# FOLDCCDIST Following Parameter

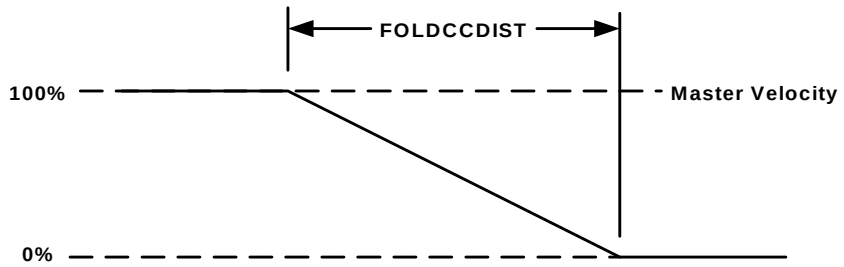
**ACTION:** Specifies the master distance traveled for the follower to attain a velocity of zero from the current velocity.

**PROGRAM SYNTAX:**
FOLDCCDIST(axis)=expression
FOLDCCDIST=expression1, ... , expression8
FOLACCDIST(axis, ... ,axis)=expression, ... , expression
FOLDCCDIST(axis) - used in an expression

**REMARKS:** The axis specifies the number of the following axis (1-8).

The expression specifies the master distance of travel in Units.

The follower axis will decelerate to a velocity of zero in the specified master distance. The distance traveled by the follower is 50% of the FOLDCCDIST distance times the FOLRATIO value.




**EXAMPLES:** FOLDCCDIST(1)=1.5
axis 1 must stop from current velocity in 1.5 units.

FOLDCCDIST=1,,3
axis 1 must stop from the current velocity in 1 unit, axis 3 must stop from the current velocity in 3 units

FOLDCCDIST(1,4)=1,3
axis 1 must stop from the current velocity in 1 unit, axis 4 must stop from the current velocity in 3 units.

# FOLRATIO Following Parameter

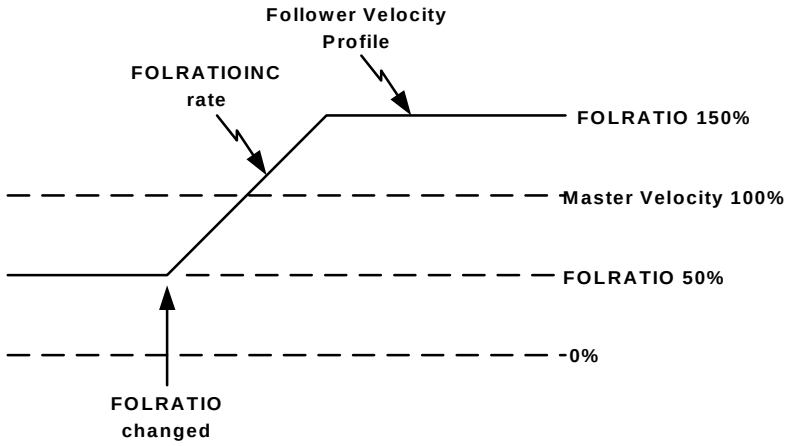
**ACTION:** Sets the ratio of the following axis to the master. A value of 1 represents 100% of the master.

**PROGRAM SYNTAX:**
FOLRATIO(axis)=expression
FOLRATIO=expression1, ... , expression8
FOLRATIO(axis, ... ,axis)=expression, ... , expression
FOLRATIO(axis) - used in an expression

**REMARKS:** The axis specifies the number of the following axis (1-8).

The expression specifies the following axis ratio to the master.

If the FOLRATIO is changed during a follower motion the follower will accelerate/decelerate to the new FOLRATIO at the FOLRATIOINC rate.




**FOLRATIO acceleration time = (FOLRATIO(new) - FOLRATIO(old)) / FOLRATIOINC**

**EXAMPLES:** FOLRATIO(2)=1.5
Sets axis 2 folratio to 150% of the master velocity.

FOLRATIO=1,,1.5
sets axis 1 folratio to 100% and axis 3 to 150% of the master velocity.

FOLRATIO(1,3)=1,1.5
sets axis 1 folratio to 100% and axis 3 to 150% of the master velocity.

# FOLRATIOINC Following Parameter

**ACTION:** Specifies the acceleration rate for a folratio change during motion in ratio increment per second.

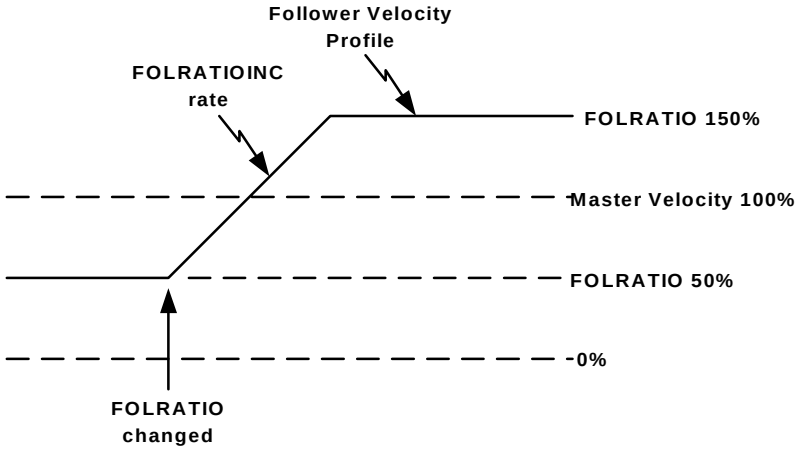
**PROGRAM SYNTAX:**

FOLRATIOINC(axis)=expression
FOLRATIOINC=expression1, ... , expression8
FOLRATIOINC(axis, ... , axis)=expression, ... , expression
FOLRATIOINC(axis) - used in an expression

**REMARKS:** The axis specifies the number of the following axis (1-8).

The expression specifies the ratio acceleration rate in ratio increment per second.

Used in conjunction with the FOLRATIO command to specify the acceleration rate for a FOLRATIO. If the FOLRATIO is changed during a follower motion the follower will accelerate/decelerate to the new FOLRATIO at the FOLRATIOINC rate.




**FOLRATIO acceleration time = (FOLRATIO(new) - FOLRATIO(old)) / FOLRATIOINC**

**EXAMPLES:**

FOLRATIOINC(1,3)=2,4
axis 1 FOLRATIO changes at a 200% rate every second and axis 3 FOLRATIO changes at a 400% rate every second.

FOLRATIOINC(2)=1
axis 2 FOLRATIO changes at a 100% rate every second.

FOLRATIOINC=2,,4
axis 1 FOLRATIO changes at a 200% rate every second and axis 3 FOLRATIO changes at a 400% rate every second.

**ACTION:** Requests a Following axis jog cycle.

**PROGRAM SYNTAX:**
FOLJOG(axis)=expression
FOLJOG=expression1 , ... , expression8
FOLJOG(axis, ... , axis)= expression, ... , expression

**REMARKS:** The axis specifies the number of the following axis (1-8).

The expression specifies the following axis to jog. If the expression is negative the motion will take place in the opposite direction of the master. The value of the expression is irrelevant.

**EXAMPLES:**
FOLJOG(2)=1
Requests following axis 2 to start a Jog cycle in the same direction of the master.

FOLJOG= 1,,-1
Requests following axis 1 to start a Jog cycle in the same direction as the master and following axis 3 to start a Jog cycle in the opposite direction of the master.

FOLJOG(1,3)=1,-1
requests following axis 1 to start a Jog cycle in the same direction as the master and following axis 3 to start a Jog cycle in the opposite direction of the master.

**RELATED COMMANDS:**
FOLACCDIST
FOLDCCDIST
FOLINPUT
FOLTRIG
FOLRATIO
FOLSYNC
FOLSTARTDIST
MOTIONSTATE
FOLOFFSET
FOLOFFSETDIST
FOLSYNCDIST
STOP
FOLMAXRATIO
FOLMINRATIO

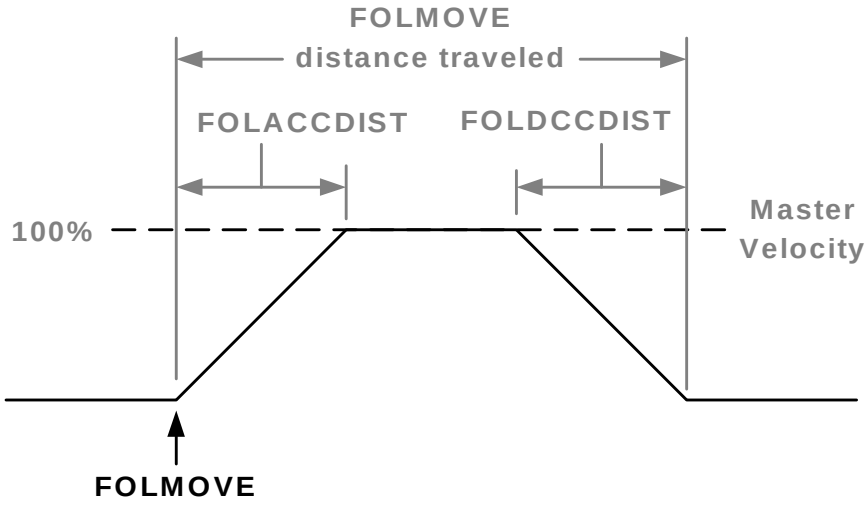
# FOLMOVE Following Motion

**ACTION:** Request a Following axis move.

**PROGRAM SYNTAX:**
FOLMOVE(axis)=expression
FOLMOVE=expression1 , ... , expression8
FOLMOVE(axis, ... , axis)=expression, ... ,expression

**REMARKS:** The axis specifies the number of the following axis (1-8).

The expression specifies the incremental move distance in units. If the expression is negative the motion will take place in the opposite direction of the master.




**EXAMPLES:**
FOLMOVE(2)=10
requests following axis 2 to move 10 units and follow the master direction.

FOLMOVE=-5,,10
request following axis 1 to move 5 unit in the opposite direction of the master. Following axis 3 to move 10 units and follow the master direction.

FOLMOVE(1,3)=-5,10
request following axis 1 to move 5 unit in the opposite direction of the master. Following axis 3 to move 10 units and follow the master direction.

**RELATED COMMANDS:**
FOLACCDIST
FOLDCCDIST
FOLINPUT
FOLTRIG
FOLRATIO
FOLSYNC
FOLSTARTDIST
MOTIONSTATE

# FOLMOVEREG Following Motion

**ACTION:**

Request a Following axis move registration cycle.

**PROGRAM SYNTAX:**

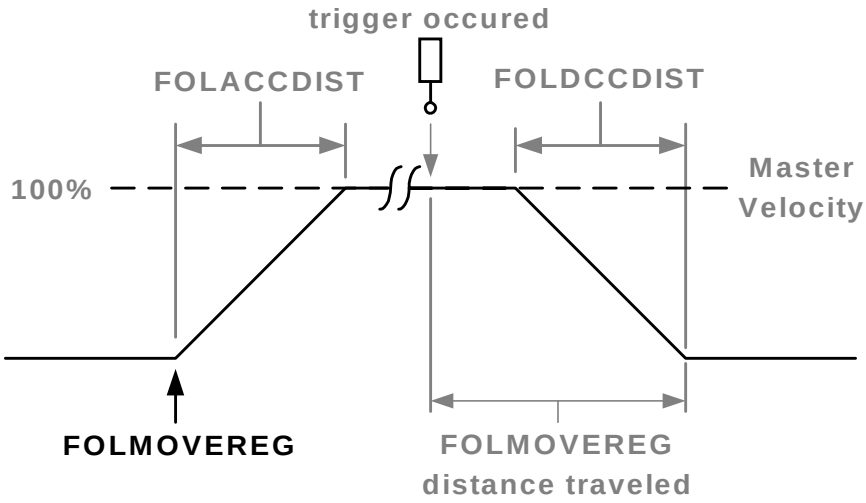
FOLMOVEREG(axis)=expression
FOLMOVEREG=expression1 , ... , expression8
FOLMOVEREG(axis, ... , axis)=expression, ... , expression

**REMARKS:**

The expression specifies the follower distance to move after the registration trigger occurs.

If the expression is negative the motion will take place in the opposite direction of the master.

The axis specifies the number of the following axis (1-8).




**EXAMPLES:**

FOLMOVEREG(2)=10
following axis 2 movereg distance is 10 units.

FOLMOVEREG=5,,10
following axis 1 movereg distance is 5 units and following axis 3 movereg distance is 10 units.

FOLMOVEREG(1,3)=5,10
following axis 1 movereg distance is 5 units and following axis 3 movereg distance is 10 units.

**RELATED COMMANDS:**

FOLACCDIST
FOLDCCDIST
FOLINPUT
FOLTRIG
FOLRATIO
FOLSYNC
FOLSTARTDIST
MOTIONSTATE

# STOP — Motion Parameter

**ACTION:** Stops any motion with a control stop.

**PROGRAM SYNTAX:**
STOP(axis)
STOP=expression1 , ... , expression8
STOP(axis, ... ,axis)

**note: JOGSTOP can be substituted for STOP.**

**REMARKS:** The axis specifies the number of the following axis (1-8).

This command will stop any motion using the DECEL value for normal motion and FOLDCCDIST for following motion.

Any value for the expression will stop the designated axis.

The WAITDONE, DONE or BUSY commands are related to the STOP command. One of these related commands should follow the STOP command to assure that motion has stopped in the designated axes before proceeding with program execution.

**EXAMPLES:**
STOP(2)
requests following axis 2 to stop.
DO : LOOP UNTIL DONE(2)

STOP=1,,1
requests following axis 1 and axis 3 to stop.
WAITDONE(1,3)

STOP(1,3)
requests following axis 1 and axis 3 to stop.
DO : LOOP WHILE BUSY(1,3)

# FOLSYNC — Following Motion

**ACTION:** Returns the following sync status of the specified axis.

**PROGRAM SYNTAX:** FOLSYNC(axis) - used in an expression

**REMARKS:** The axis specifies the number of the following axis (1-8).

The value returned is either a 0 (out of sync) or 1 (in sync).

**EXAMPLE:**
DO : LOOP UNTIL FOLSYNC(2)=1
wait for axis 2 to synchronize with master velocity.

DO : LOOP WHILE FOLSYNC(2)=0
wait for axis 2 to synchronize with master velocity.

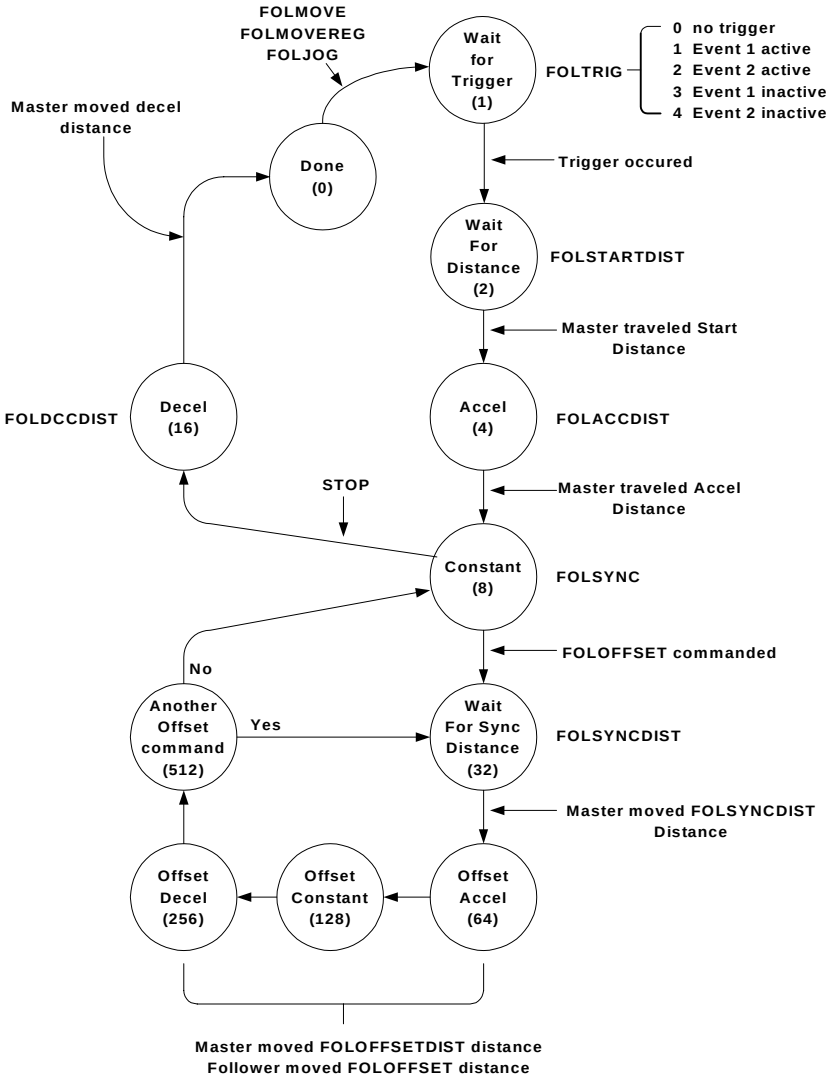
# MOTIONSTATE Trajectory Parameter

**ACTION:** Returns the motion state for an axis.

**PROGRAM SYNTAX:** MOTIONSTATE(axis) - used in an expression.

**REMARKS:** The motion states for a following cycle are:

| | |
|---|---|
| 0 | Following cycle Done. |
| 1 | Waiting for Following Trigger. |
| 2 | Waiting for master to move FOLSTARTDIST distance. |
| 4 | Acceleration to Master Velocity in FOLACCDIST distance. |
| 8 | In Synchronization with master Velocity. |
| 16 | Decelerating to Stop in FOLDCCDIST master distance. |
| 32 | Offset command issued and Waiting for master to move FOLSYNCDIST distance before starting the Offset Acceleration. |
| 64 | Offset Acceleration occurring. |
| 128 | At FOLMAXRATIO or FOLMINRATIO limit. |
| 256 | Offset Deceleration occurring. |
| 512 | Checking for pending Offset Cycle. |



Motion state 0 (Done)
No following motion is taking place or being commanded.

Motion state 1 (Wait for Trigger)
A following motion has been commanded and is waiting for the specified trigger to occur. The trigger is specified by the FOLTRIG command.

Motion state 2 (Waiting for Distance)
Waiting for the master delay distance to be completed. This master distance traveled is specified by the FOLSTARTDIST command.

Motion state 4 (ACCEL)
The follower motion has started and is accelerating to the master velocity. The master distance traveled during acceleration is specified by the FOLACCDIST command.

Motion state 8 (Constant)
The follower is in synchronization with the master velocity and no offset cycle has been commanded. This state sets the return state of the FOLSYNC command to a one.

Motion state 16 (DECEL)
The follower is decelerating to a stop.

Motion state 32 (Wait For Sync Distance)
The follower and master velocities are in synchronization. This is the first portion of the offset cycle. The master travels the FOLSYNCDIST distance during this portion of the offset cycle.

Motion state 64 (Offset Accel)
The follower is accelerating to the FOLMAXRATIO velocity during an advance-offset cycle or decelerating to the FOLMINRATIO velocity during a recede-offset cycle. The master is executing the FOLOFFSETDIST distance.

Motion state 128 (Offset Constant)
The follower is running at the FOLMAXRATIO or FOLMINRATIO velocity. The master is still executing the FOLOFFSETDIST distance.

Motion state 256 (Offset Decel)
The follower is decelerating from the FOLMAXRATIO value during an advance-offset cycle or accelerating from the FOLMINRATIO value during a recede-offset cycle. The offset cycle will be completed when the master velocity times the FOLRATIO value is reached. The master is still executing the FOLOFFSETDIST distance and is completed when the offset cycle is completed.

Motion state 512 (Another Offset command)
The follower has just completed an offset cycle and is checking to see if a pending offset cycle is requested. If a pending offset cycle is requested will proceed to motion state 32 otherwise, will go to motion state 8.

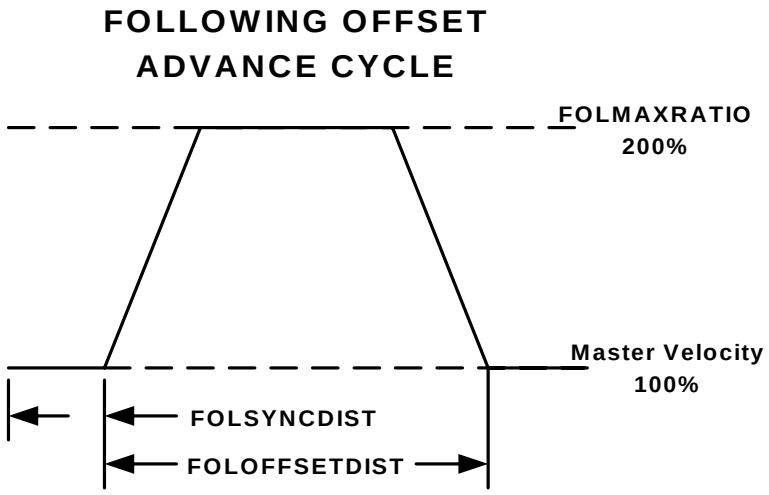
# FOLMAXRATIO Following Parameter

**ACTION:** Sets or returns the maximum allowable following axis speed during an offset advance cycle.

**PROGRAM SYNTAX:**
FOLMAXRATIO(axis)=expression
FOLMAXRATIO=expression1 , ... , expression8
FOLMAXRATIO(axis, ... ,axis)=expression, ... , expression
FOLMAXRATIO(axis) - used in an expression

**REMARKS:** The expression sets the maximum speed ratio to the master. This value must be larger than the FOLRATIO of the axis. The value must be a positive number.

The axis specifies the number of the following axis (1-8).




**EXAMPLES:** FOLMAXRATIO(2) = 3
sets the folmaxspeed of axis 2 to 300% of the master.

FOLMAXRATIO=.5,,1
sets the folmaxratio of axis 1 to 50% of master and folmaxratio of axis 3 to 100% of the master.

FOLMAXRATIO(1,3)=.5,1
sets the folmaxratio of axis 1 to 50% of master and folmaxratio of axis 3 to 100% of the master.

# FOLMINRATIO Following Parameter

**ACTION:** Sets or returns the minimum allowable following axis speed during a recede offset cycle.

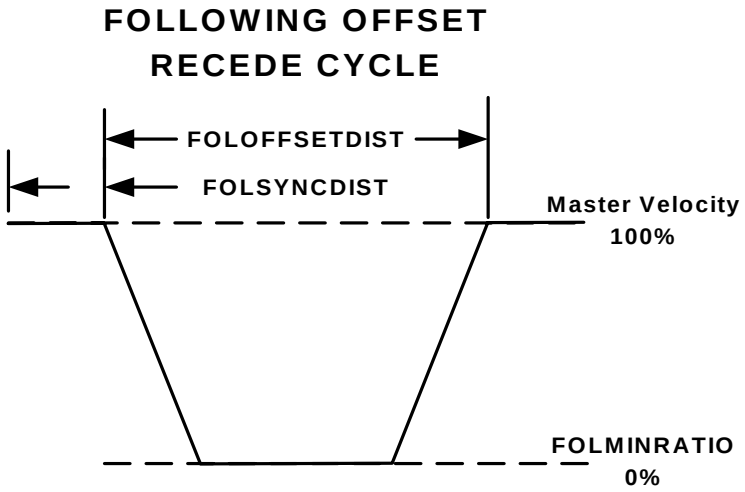
**PROGRAM SYNTAX:**

FOLMINRATIO(axis)=expression
FOLMINRATIO=expression1 , ... , expression8
FOLMINRATIO(axis, ... ,axis)=expression, ... , expression
FOLMINRATIO(axis) - used in an expression

**REMARKS:** The expression sets the minimum speed ratio to the master. This value must be less than the FOLRATIO of the axis and can be a negative value.

If the value is a negative number the following axis will be allowed to reverse the direction during a recede offset cycle.

The axis specifies the number of the following axis (1-8).




**EXAMPLES:**

x=FOLMINRATIO(axis)
Sets the expression to the current FOLMINRATIO of the specified axis.

FOLMINRATIO(2)= -2.0
sets the following minimum speed for axis 2 to -200% which allows an offset recede cycle to reverse directions.

FOLMINRATIO=.1,,0
sets the folminratio of axis 1 to 10% of master and folminratio of axis 3 to 0% of the master.

FOLMINRATIO(1,3)=.1,0
sets the folminratio of axis 1 to 10% of master and folminratio of axis 3 to 0% of the master.

# FOLOFFSET Following Parameter

**ACTION:** Defines a following incremental offset distance from the current position.

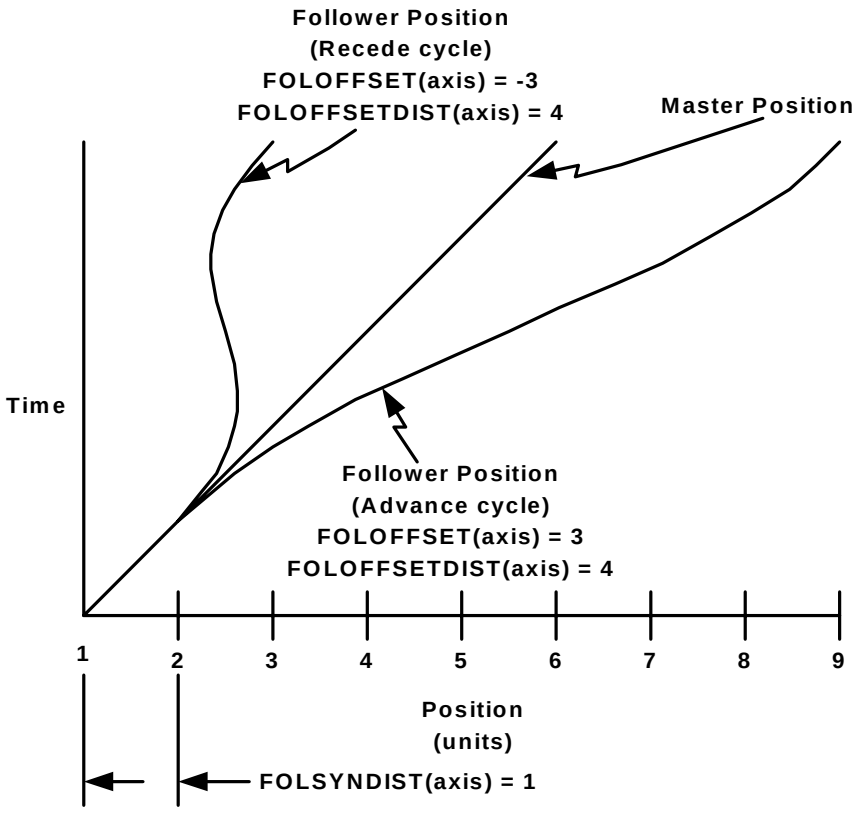
**PROGRAM SYNTAX:** FOLOFFSET(axis)=expression
FOLOFFSET=expression1, ... , expression8

**REMARKS:** The expression specifies the following axis offset in units.

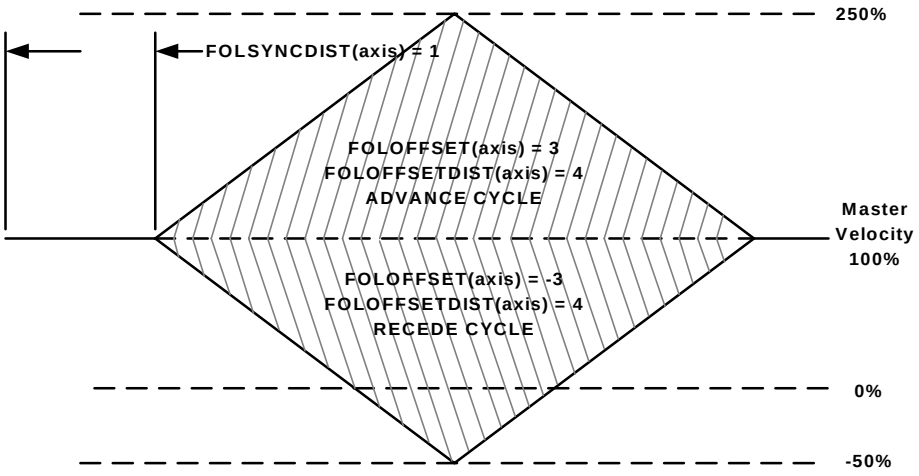
The axis specifies the number of the following axis (1-8).

Used in conjunction with FOLSYNCDIST, FOLOFFSETDIST, FOLMAXRATIO and FOLMINRATIO to advance or recede the follower axis. The FOLSYNCDIST defines the master delay distance travel in synchronization after an FOLOFFSET command is issued. The FOLOFFSETDIST defines the master distance traveled while the FOLOFFSET is being performed. The FOLMAXRATIO defines the upper velocity limit for an advance cycle. The FOLMINRATIO defines the lower limit velocity for a recede cycle.

The FOLOFFSET command only works during a FOLJOG cycle .



**FOLOFFSET continued**




**EXAMPLES:**

FOLOFFSET(1,3)=1,-1
advance axis 1 one unit and recede axis 3 one unit.

FOLOFFSET(2)=1
advance axis 2 one unit.

FOLOFFSET=1,,-1
advance axis 1 one unit and recede axis 3 one unit.

# FOLOFFSETDIST Following Parameter

**ACTION:** Sets or returns the master distance traveled for a FOLOFFSET command.

**PROGRAM SYNTAX:**
FOLOFFSETDIST(axis)=expression
FOLOFFSETDIST=expression1, ... , expression8
FOLOFFSETDIST(axis, ... , axis)=expression, ... , expression
FOLOFFSETDIST(axis) - used in an expression

**REMARKS:** The axis specifies the number of the following axis (1-8).

The expression specifies the master distance traveled in Units.

Used in conjunction with the FOLOFFSET command to specify the master distance traveled during a FOLOFFSET command.

**EXAMPLES:** FOLOFFSETDIST(1,3)=1,1
axis 1 master distance is one unit and axis 3 master distance is one unit.

FOLOFFSETDIST(2)=1
axis 2 master distance is one unit.

FOLOFFSETDIST=1,,1
axis 1 master distance is one unit and axis 3 master distance is one unit.

# FOLSYNCDIST Following Parameter

**ACTION:**

Specifies the master distance to travel when a FOLOFFSET command is issued. This distance will be traveled before the FOLOFFSET command is executed.

**PROGRAM SYNTAX:**

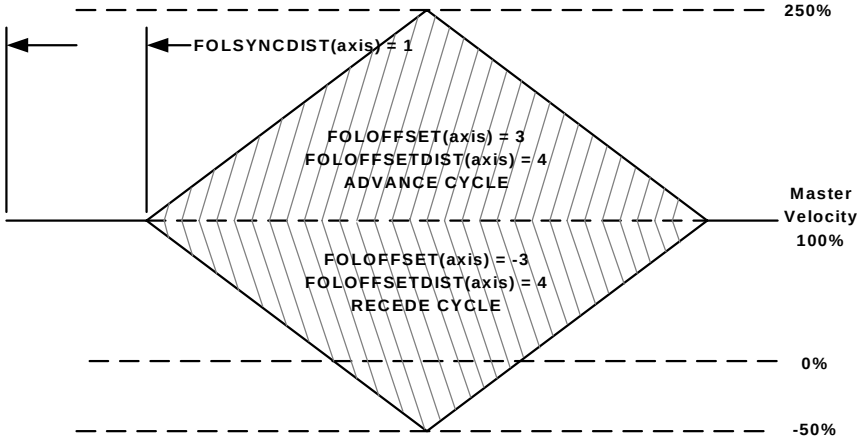
FOLSYNCDIST(axis)=expression
FOLSYNCDIST(axis)=expression1, ... , expression8
FOLSYNCDIST(axis) - used in an expression
FOLSYNCDIST(axis, ... ,axis)=expression, ... ,expression

**REMARKS:**

The axis specifies the number of the following axis (1-8).

The expression specifies the master distance traveled.

Used in conjunction with the FOLOFFSET command to specify the in synchronization master distance traveled during a FOLOFFSET cycle.




**EXAMPLES:**

FOLSYNCDIST(2)=.5
sets the master distance traveled for axis 2 to .5 units.

FOLSYNCDIST(1,3)=.5,.6
sets the master distance traveled for axis 1 to .5 units and the master distance traveled for axis 3 to .6 units

FOLSYNCDIST=.5,,.6
sets the master distance traveled for axis 1 to .5 units and the master distance traveled for axis 3 to .6 units

## 8.1.11 - Follower Exercise

1) Fill in the Motion states and Following Commands on the following sheet. Put the Following command letter inside the Parenthesis and the motion state number in between brackets. All of the Following commands and Motion states are used at least once.
2) Write a user program that will generate the velocity profile on the following sheet. The master axis should be stable before the follower axis is allowed to move.
3) Set up data logging to prove that the profile will be generated.

| | |
|---|---|
| '********** Follower will follow the Master axis speed at 100% ********** | |
| '********** Follower triggers when EVENT 1 is active ********** | |
| #DEFINE MASTER 2 | |
| #DEFINE FOLLOWER 1 | |
| SPEED(MASTER)=5 | ' Master velocity is 5 units/sec |
| ACCEL(MASTER)=50 | ' Master acceleration is 50 units/sec$^2$ |
| DECEL(MASTER)=50 | ' Master deceleration is 50 units/sec$^2$ |
| DRVREADY(FOLLOWER,MASTER)=1,1 | ' drives not required to run program |
| JOG(MASTER)=1 | 'START MASTER AXIS |
| DO : LOOP UNTIL ACTSPD(MASTER) >= 4.9 | 'WAIT FOR MASTER TO ACHIEVE SPEED |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| STOP(FOLLOWER) | ' follower axis stop |
| WAITDONE(FOLLOWER) | ' wait for follower axis done |
| STOP (MASTER) | ' master axis stop |
| WAITDONE(MASTER) | ' wait for master axis done |
| END | |

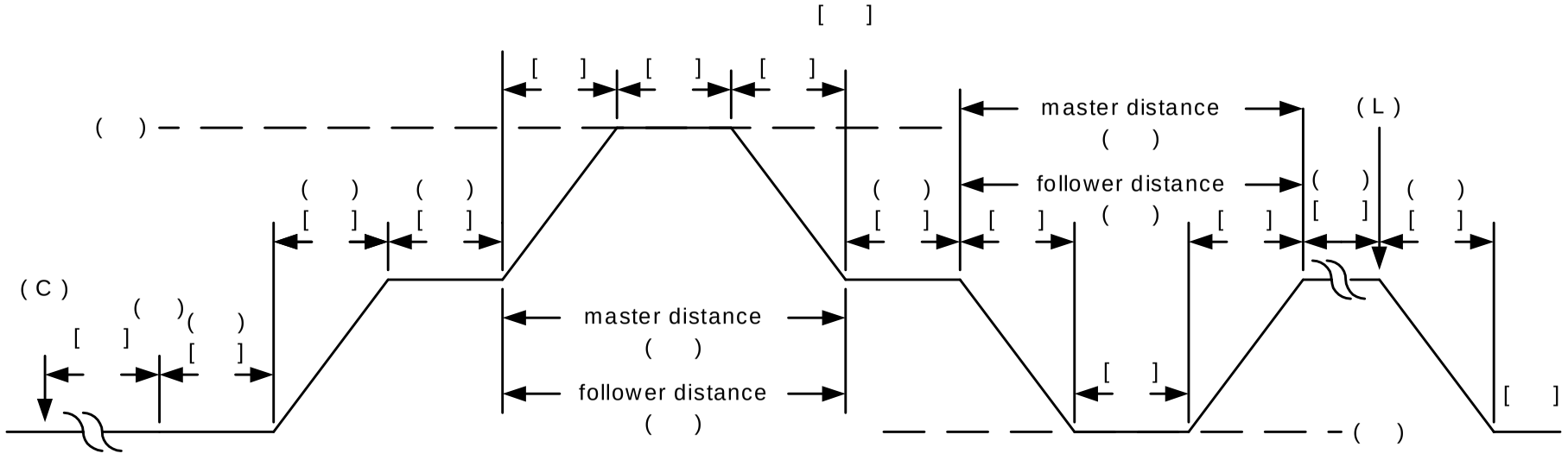
MotionStates

| | |
|---|---|
| 0 | Following cycle done |
| 1 | Waiting for trigger |
| 2 | Waiting for start distance to elapse |
| 4 | Accellerating to master velocity |
| 8 | In Velocity Synchonization |
| 16 | Decelerating to stop |
| 32 | Waiting for offset sync distance to elapse |
| 64 | Offset cycle acceleration taking place |
| 128 | Offset cycle at constant speed |
| 256 | Offset cycle deceleration taking place |
| 512 | Checking for pending offset cycle |

Following Commands

a) FOLACCDIST
b) FOLDCCDIST
c) FOLJOG
d) FOLMAXRATIO
e) FOLMINRATIO
f) FOLOFFSET
g) FOLOFFSETDIST
h) FOLSTARTDIST
i) FOLSYNC
j) FOLSYNCDIST
k) FOLTRIG
l) STOP

( ) enter following command letter
[ ] enter Motion State Number



Program answer:

```
'********************** Follower will follow the Master axis speed  at 100% ***************************
'********************** Follower triggers when EVENT 1 is active  *************************************
#DEFINE      MASTER          2
#DEFINE      FOLLOWER     1
SPEED(MASTER)=5                                    ' Master velocity is 5 units/sec
ACCEL(MASTER)=50                                   ' Master acceleration is 50 units/sec^2
DECEL(MASTER)=50                                   ' Master deceleration is 50 units/sec^2
DRVREADY(FOLLOWER,MASTER)=1,1                      ' drives not required to run program
JOG(MASTER)=1                                      'START MASTER AXIS
DO : LOOP UNTIL ACTSPD(MASTER) >= 4.9              'WAIT FOR MASTER TO ACHIEVE SPEED
'******* initialize the follower parameters ********************************************************
FOLRATIO(FOLLOWER)=1.0                             ' following ratio is 100%
FOLTRIG(FOLLOWER) = 1                              ' follower triggers on event1 going active
FOLACCDIST(FOLLOWER) = 1                           ' follower catches master in 1 unit of travel of the master
FOLDCCDIST(FOLLOWER) = 1                           ' follower stop in 1 unit of travel of the master
FOLMAXRATIO(FOLLOWER) = 2.0                        ' offset maximum velocity limit is 200% of master
FOLMINRATIO(FOLLOWER) = 0                          ' offset minimum velocity limit is 0% of master
FOLSTARTDIST(FOLLOWER) = 1                         ' delay 1 unit of travel of the master before motion
'******* define follower & master axes *************************************************************
FOLINPUT(FOLLOWER) = ACTSPD(MASTER)
'******* request for motion start of follower axis
FOLJOG(FOLLOWER) = 1
FOLSYNCDIST (FOLLOWER) = 1                         ' distance master travels during in sync portion of offset
FOLOFFSETDIST(FOLLOWER) = 1                        ' distance master travels after in sync portion of offset
FOLOFFSET(FOLLOWER) = 1                            ' offset cycle request to advance follower 1 unit
'****** wait for in sync portion of offset cycle to begin ******************************************
DO : LOOP UNTIL MOTIONSTATE(FOLLOWER) = 32

'****** wait for offset portion of cycle to begin
DO : LOOP UNTIL MOTIONSTATE(FOLLOWER) <> 32

FOLSYNCDIST (FOLLOWER) = 1                         ' distance master travels during in sync portion of offset
FOLOFFSETDIST(FOLLOWER) = 1                        ' distance master travels after in sync portion of offset
FOLOFFSET(FOLLOWER) = -1                           ' offset cycle request to recede follower 1 unit
'****** wait for offset cycle to finish ************************************************************
DO : LOOP UNTIL FOLSYNC(FOLLOWER) =1
WAIT=1                                             ' optional wait before stopping
STOP(FOLLOWER)                                     ' follower axis stop
WAITDONE(FOLLOWER)                                 ' wait for follower axis done
STOP (MASTER)                                      ' master axis stop
WAITDONE(MASTER)                                   ' wait for master axis done
END
```

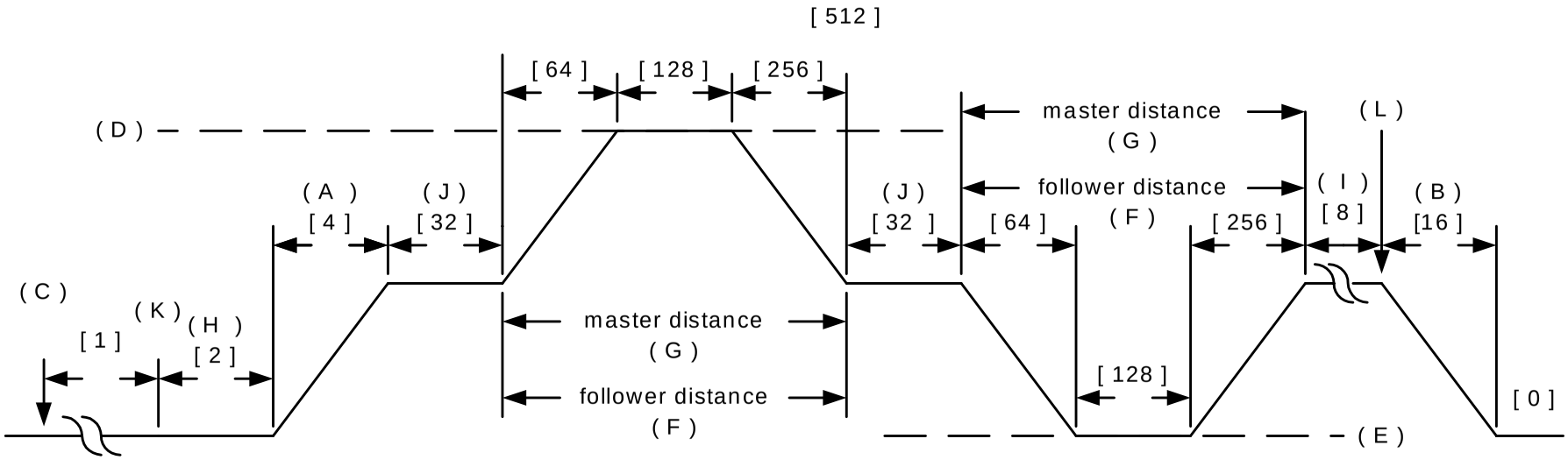
MotionStates

| | |
|---|---|
| 0 | Following cycle done |
| 1 | Waiting for trigger |
| 2 | Waiting for start distance to elapse |
| 4 | Accellerating to master velocity |
| 8 | In Velocity Synchonization |
| 16 | Decelerating to stop |
| 32 | Waiting for offset sync distance to elapse |
| 64 | Offset cycle acceleration taking place |
| 128 | Offset cycle at constant speed |
| 256 | Offset cycle deceleration taking place |
| 512 | Checking for pending offset cycle |

Following Commands

A) FOLACCDIST
B) FOLDCCDIST
C) FOLJOG
D) FOLMAXRATIO
E) FOLMINRATIO
F) FOLOFFSET
G) FOLOFFSETDIST
H) FOLSTARTDIST
I) FOLSYNC
J) FOLSYNCDIST
K) FOLTRIG
L) STOP

( ) enter following command letter
[ ] enter Motion State Number



This page left intentionally blank

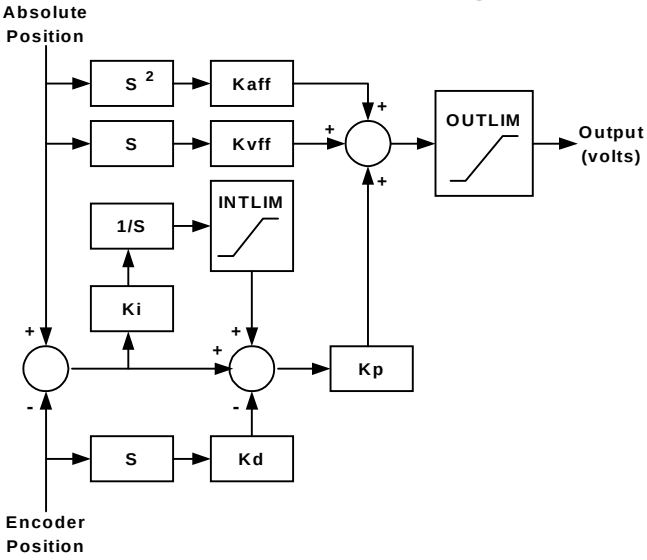
# Section 9
# Servo Drive

## 9.1 - Servo Control

A servo is a closed loop system. The loop is closed by taking a measurement of the actual output (usually a position or velocity) and comparing it to the desired command or reference input. Subtracting the output signal from the reference generates an error signal. The error signal tells the controller how far away the output is from the desired position. Then, a control law (algorithm) modifies this error signal to provide an output to drive a servo amplifier.

The controller uses a modified form of the classic PID (Proportional, Integral, Derivative) control law with velocity feed forward. The commanded position is compared to the Encoder position and a position error is generated. A control algorithm modifies this error to provide an output torque command to drive the servo amplifier. The PID control loop uses or derives the following parameters or commands:

### Servo Block Diagram




The user project configuration allows access to the majority of the servo parameters and in some case this is the only access.

The servo parameters that can be modified in the program configuration only are **Integration during motion** (Servo Drive Folder), **Sample time** (Servo Drive Folder**), Encoder line count** (Encoder Folder), **Encoder direction** (Encoder Folder). The **Integration during motion** parameter control whether the integral term has any effect on the output voltage to the servo drive during motion. The **Sample time** controls how often the voltage output is updated. The **Encoder line count** indicates the resolution of the encoder. This value must match the encoder line count of the servo motor encoder. The pulse count per revolution is four times the Encoder Line count. One pulse count is the finest resolution that can be attained. The **Encoder direction** parameter allows a convenient method of changing the encoder direction if incorrect.

If the servo drive has a limitation on the commanded input voltage, other than +10v to –10v, a limit could be imposed using the OUTLIM command. However, this parameter is limited to a user program command only. **This command must be placed at the beginning of the user program to protect a servo drive that has a voltage input limitation.**

The remaining servo parameters can be modified in the program configuration as well as the users program. These are **Kp** (Proportional gain), **Ki** (Integral gain), **Kd** (Derivative gain), **Kaff** (Acceleration feed forward), **Kvff** (Velocity feed forward), **INTLIM** (integral limit voltage), **FOLERR** (maximum error allowed). These parameters can be modified in the configuration using the Servo Drive Folder**. Note: The Kp, Ki, Kd and Kaff parameters are modified during auto-tuning**. **Some controls do not have Kaff as a parameter.**

It is highly recommended to tune a servo drive using the AUTO-TUNE environment. Under certain conditions, mostly compliant loads, this may not be possible. Thus, if the servo drive must be manually tuned a means for this is also available in the servo-tuning environment.

## 9.1.1 - Servo Tuning

Tuning is a process of determining the PID and feed forward gains to get the desired system response. Typical performance indicators like: overshoot, response time, stiffness, settling time, bandwidth and damping can all be used to measure how well the system is tuned. Tuning a gain to improve its performance characteristic may cause another characteristic to get worst.

Before attempting servo tuning the following must be done: Modify the project configuration for the System folder, Encoder folder, Servo Drive folder and then Compile and download the user project to the controller.

## 9.1.1.1 - System folder

The System folder allows the Drive type, Task assignment for the drive, motor direction for a + motion and Units per motor revolution to be configured.

System

| | Task assignment | Drive Type | Motor Direction | Units per motor resolution |
|---|---|---|---|---|
| Axis 1 | c:\mcpi\name.tsk | open loop stepper | + = cw motor direction | 1.0 |
| Axis 2 | c:\mcpi\name.tsk | open loop stepper | + = cw motor direction | 1.0 |

1) Assign the servo drive to an axis by selecting the servo drive item from the **Drive Type** drop list.
2) The servo drive must be assigned to a specific task. Choose the task from the **Task assignment** drop list.
3) If the motor direction requires a reversal for a + direction motion make the necessary choice under the **Motor Direction** drop list.
4) Define the unit value of the axis. Enter the desired value in the **Units per motor revolution** text box. Example: 1=1 Unit/motor rev.

## 9.1.1.2 - Encoder Folder

This folder defines the Servo Encoder direction and Encoder resolution.

Encoder

| | Encoder type | Encoder direction | Line count (lines / rev) | pulse count (pulses/rev) |
|---|---|---|---|---|
| Axis 1 | quadrature | normal direction | 500 | 2000 |
| Axis 2 | quadrature | normal direction | 500 | 2000 |

**Encoder type** must be set to quadrature.

**Encoder direction** determines how the encoder rotation direction is interpreted. The choices are normal direction or reverse direction. Use the default setting to start.

**Encoder line count** defines the encoder resolution in lines for a quadrature encoder. An Encoder with 1000 lines will provide 4000 counts/revolution, or quadrature counts. Set this value to the encoder line count of the servomotor.

**Pulse count** defines the pulse count per motor revolution. This value is always 4 times the Encoder line count. If the encoder input is pulse and direction, the pulses/rev value should be entered here.

## 9.1.1.3 - Servo Drive Folder

This folder allows the user project servo drive parameters to be modified. The PID loop gains, acceleration feed forward gain, velocity feed forward gain, integral limit, following error, sample time, and enable/disable integration during motion. The default settings for this folder are suggested before tuning the servo drive.

This folder is modified during auto or manual tuning of a servo drive and requires compilation and downloading of the project to save the tuning settings.

Servo drive

| | Proportional gain (millivolts/count) | Integral gain (msec) | Derivative gain (msec) | Accel feed forward (volts/count/msec$^2$) | Velocity feed forward (%) |
|---|---|---|---|---|---|
| Axis 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Axis 2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

Servo drive

| | Integral limit (volts) | Following error (units) | Sample time (milliseconds) | Integration during motion |
|---|---|---|---|---|
| Axis 1 | 100.0 | 0.05 | 1.024 msec | enabled |
| Axis 2 | 100.0 | 0.05 | 1.024 msec | enabled |

**Proportional gain**

This gain is multiplied by the position error and thus contributes **proportionally** to the output torque. Generally, the higher the Kp, the lower the error at any time during the move. However, if Kp is too high, the system can overshoot severely or "buzz" loudly. This type of buzzing instability may be seen as "grass" on the error response curve in the move response screen. In this case, Kp should be lowered. Kd may also be lowered, but to a lesser extent.

Generally the range for Kp is 10 to 150. Kp less than 10 will usually produce a soft or sluggish system. Kp over 175 produces a stiff system, but one that may be approaching instability. Note these are general ranges, not absolute requirements.

**Integral gain**

The reciprocal (1/Ki) of this term is multiplied by the **sum** of the position error over time. The effect of Ki is thus time related, and affects the steady state error. The higher Ki, the longer it will take for the controller to "integrate out" any steady state error. The effect of Ki is seen mostly at constant speed (including standstill). Ki is NOT required for stability, and generally has a de-stabilizing effect on the system, especially if it is too low. If Ki is TOO LOW the system may oscillate slowly and wildly back and forth like a washing machine. Ki is required, if the system must achieve a very low steady state error (within a few counts).

The general range for Ki is 10 to 70. Ki less than 10 may lead to wild, low frequency oscillations. If steady state error is not a consideration, Ki may be set to zero. Ki is often disabled during motion to reduce overshoot at the end of the move.

**Derivative gain**

This term is multiplied by the encoder velocity at any point in time. Generally, raising Kd will reduce overshoot in the move response, however, Kd is the term most susceptible to "digital instability". This is where the quantification effects of the digital encoder feedback in conjunction with too high a Kd cause the system to "buzz".

The general range for Kd is 5 to 20. Kd less than 5 usually leads to an unstable system, Kd >20 usually leads to "buzzing".

**Accel feed forward**

Some controllers have a Kaff term. This term is multiplied by the commanded acceleration to contribute to the output torque command. This term only takes effect to reduce the error during acceleration and deceleration. Generally Kaff is less than 4. Most applications will run fine with Kaff set at zero.

**Velocity feed forward**

This term is multiplied by the commanded velocity to contribute to the output torque command. It has no effect on general stability, and may be set to as high as 100% to reduce position error during motion. Too high a Kvff causes undue motor heating. Generally, Kvff should be set between 50 and 100.

**Integral limit**

Limits the contribution of the integral term to the servo loop's output. This limit is imposed on the internal calculation within the controller, and is used to prevent excessive buildup of the integrator output which can occur if a constant error is allowed to exist for extended periods of time. Too low an integral limit may reduce the effectiveness of the integrator by limiting its contribution to the output torque command. This would cause a constant steady state error. Too high an integral limit may allow the integrator to build up a large error stored in the controller memory. This error would then be "unwound" at the end of a move causing excessive overshoot and a long settling time. The limit can be set between 0 and 319 volts. A setting of 100 is a good midrange starting point, and this parameter rarely needs adjustment.

**Following error**

Defines the maximum error allowed during motion in units. If this limit is exceeded the servo drive shuts down. The default setting is a good starting point.

**Sample time**

Determines how often the servo loop output is updated. The possible settings are .256 milliseconds to 2.048 milliseconds in .256 millisecond increments. A setting of 1.024 is a good starting point.

**Integration during motion**

This feature allows you to select whether the integration gain is used during the profile motion. Enabling the integrator during motion will reduce your error at speed, but may cause some unacceptable overshoot in the response. Some controllers allow you to set this parameter in the servo tuning screen, while others require that you change it in the Servo folder in the program configuration (be sure to compile and download the project each time you change the configuration or the change will not take effect).

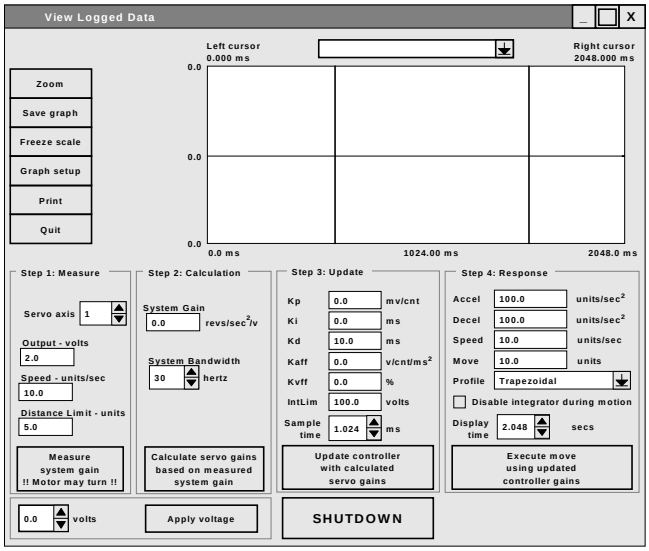
**Note: A program does not have to be written in the task in order to tune the servomotor.**

## 9.1.1.4 – Servo Tuning Environment

The servo-tuning Environment allows a servo drive to be manually tuned, auto-tuned and testing the results of the tuning.

Clicking on the **Servo Tuning** command button can access the servo-tuning Environment. The project in the controller must match the project in the PC. If necessary compile and download the project at this time.

The servo gains, Integration Limit, Sample Time, Integration during motion parameters can be modified for an individual axis on the opening screen. The four steps for tuning a servo can be executed from this screen as well.




**Servo Tuning environment**

**Command Buttons**

**Zoom** toggles between displaying the graph between cursors and the full screen view. The two vertical lines in the display window are the cursors.

**Save Graph** saves the currently displayed graph and appears as an item on the drop list.

**Freeze scale** freezes the current logged scale value.

**UnFreeze** allows the next commanded motion graph to be auto scaled.

**Graph setup** allows for the selection of color and style for each logged item.

**Print** prints the current graph.

**Quit** exits the Servo Tuning environment.

**Shutdown** disables the servo drive and outputs a torque command of 0 volts. The **Update controller gains** command button will re-enable the servo output voltage.

**Step 1: Measure**

**Servo axis** selects the servo axis.

**Output – volts** selects the stimulus voltage for measuring system gain. The default is 2 volts.

**Speed – units/sec** selects the target speed for measuring system gain. The default is 10 units/sec.

**Distance Limit – units** limits the bump travel distance allowed when measuring system gain. The default is 5 units.

**Measure system gain** commands a system gain measurement when clicked. The **System Gain** will be updated when the cycle is completed.

**Step 2: Calculation**

**System Gain** displays the result of a measure system gain cycle or a manually entered value.

**System Bandwidth** selects the system bandwidth for the Gain Calculation. The default is 30 hertz.

**Calculate servo gains** commands a gain calculation cycle. The **Kp**, **Ki**, **Kd** and **Kaff** values will be updated at the completion of the cycle.

**Step 3: Update**

**Kp** displays the current value of the proportional gain. This can be manually changed if desired.

**Ki** displays the current value of the integral gain. This can be manually changed if desired.

**Kd** displays the current value of the derivative gain. This can be manually changed if desired.

**Kaff** displays the current value of the acceleration feed forward gain. This can be manually changed if desired.

**Kvff** displays the current value of the velocity feed forward gain. This is used to reduce the positional error during acceleration. This can be manually changed if desired.

**IntLim** displays the current value of the integral limit. This can be manually changed if desired.

**Sample Time** selects the servo sample time of the servo axis.

**Update controller gains** transfers the current values of Kp, Ki, Kd, Kaff, Kvff, IntLim and Sample time to the controller. The servo drive is now enabled.

**Step 4: Response**

**Accel** selects the acceleration rate for a move response. Default is 100 units/sec$^2$.

**Decel** selects the deceleration rate for a move response. Default is 100 units/sec$^2$.

**Speed** selects the target speed for a move response. Default is 10 units/sec.

**Move** select the incremental distance traveled during a move response. Default is 10 units.

**Profile** selects the motion profile for a move response. Default is trapezoidal.

**Disable integrator during Motion** enables or disables the integrator during motion. When checked the integrator is disabled during motion.

**Display time** selects the logging period for a move response cycle. Up to 10 seconds can be logged.

**Execute move** commands a move response. The logging results are transferred when the cycle is completed. The individual logged items can be selected by clicking on the arrow in the Display Drop List.

**Display**

**Display Drop list** selects the logged item to be displayed.

**View Port** displays the results of a move response cycle.

**Torque Control**

**Volts** select the stimulus torque voltage that will be applied to the servo drive when the **Apply voltage** button is clicked.

**Apply voltage** transfers the selected stimulus voltage selected by the **volts** spin controller as the torque command for the servo drive.

## 9.1.1.5 - Auto Tuning

Before a servo can run properly, the servo gains Kp, Ki, Kd, and Kvff must be set up to yield the appropriate move response. The controller has the ability to automatically set the servo gains using an automatic tuning procedure.

Auto-tuning can be broken into four separate steps measure gain (step 1), calculate gains (step 2), update gains (step 3) and move Response (step 4).

**Step 1: Measure**

The system gain is a measure of the overall responsiveness of the system. Higher inertia and/or lower torque yields lower system gain. Lower inertia and/or higher torque yields higher system gain. The system gain number is used when the software calculates the servo gains. A lower system gain requires higher calculated controller gains in order for the motor to track a given profile response.

Clicking **Measure System Gain** instructs the controller to provide a "bump" of torque to the motor. Three parameters, **Output, Speed**, and **Distance Limit** are used to measure system gain.

The **Output** text box is used to select the amount of voltage that the controller will use to bump the motor. The range of the Output is 0 to 10 volts where 10 volts represents peak torque. Typically the default parameter of 2 volts is adequate, although some large inertia systems may require the Output be set to 3 or 4 volts.

The **Speed** text box is used to select the target velocity for the gain measurement. During gain measurement, the output torque will be applied to the motor until the speed set here is reached. The default speed is usually sufficient if revs/sec is used for the unit of measure.

The value in the **Distance Limit** text box limits the distance that the motor will turn during the gain measurement. If the **distance limit** is reached before the motor reaches the **speed** indicated, or if the speed cannot be reached with the voltage entered, an error message will appear. If an error appears, try increasing the distance limit or raising the voltage output slightly.

Generally, the default parameters for these three parameters should be used during the gain measurement, provided that the unit per motor rev was left at the default of 1.

## Caution! *When the* Measure System Gain *button is clicked, the motor will move quickly and abruptly for a short distance.*

If the gain measurement is unsuccessful, verify that the motor moves properly with a constant torque command applied. Clicking the Apply Voltage button on the tuning screen will do this. Clicking the **Apply Voltage** button will output a constant torque to the motor proportional to the command voltage. Start with zero and click on the up or down arrows to apply positive or negative torque respectively. Unless the system has high friction, the motor should begin to move with less than one volt applied. Check that the motor torque is smooth and continuous in both directions by applying small amounts of positive and negative voltage.

**Step 2: Calculation**

The system bandwidth is essentially the maximum frequency of excitation to which the system will respond. Generally, higher bandwidth systems are "stiffer" or "tighter". Lower bandwidth systems are "soft" or "sluggish". Generally bandwidths range between 10 to 60 Hz (cycles per second). The auto tuning procedure uses the bandwidth setting along with the measured system gain to calculate the appropriate servo gains for the system. The default bandwidth of 30 Hz is usually a good starting point, although sometimes the bandwidth must be lowered to achieve a stable system, or raised to achieve a fast enough response.

**Calculate Servo Gains**

Clicking the **Calculate Servo Gains** button will use the bandwidth and measured system gain to calculate the Kp, Ki, Kd, Intlim (and Kaff if applicable) parameters. These fields will be updated after the calculation is complete.

**Step 3: Update**

Clicking Update Gains will update the gains to the controller immediately. **Caution! Updating the gains may change the dynamics of the system such that it becomes unstable and oscillatory.** If a loud buzzing or vibration occurs after updating gains, the **Shutdown** button should be clicked. It is also possible that a fault will occur if the oscillation overtaxes the servo drive. In this case you will have to enter the terminal screen and clear the error by typing ERR or ERRM. If necessary, go to **Step 2** and **lower the bandwidth** and re-calculate the servo gains. Now update the gains again. Repeat this process until the system is stable and will smoothly resist loading in both directions.

**Step 4: Response**

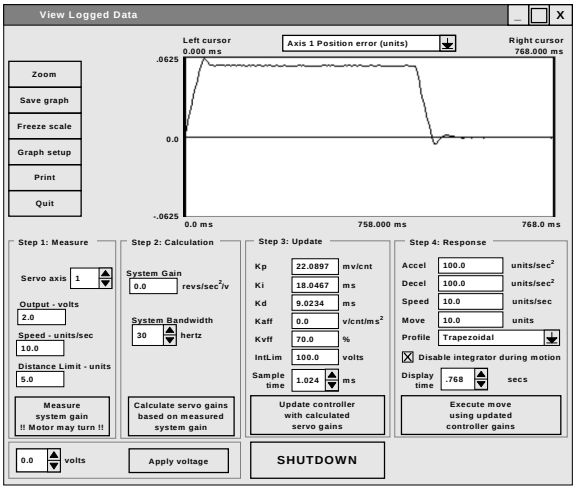
Step 4 allows test motion profile parameters to be entered so that the proper motion response may be verified. **Accel, Decel, Speed** and **Move Distance** parameters describe the move that the motor will try to follow during the test. The **display time** is adjustable so that shorter or longer moves may be fully displayed. The unit for each parameter is configured in the System Folder.

Once the profile parameters setup is complete, the system is ready to attempt to execute the move. Clicking the **Execute move** command button will command the motor to execute the move profile. The controller will log the response of the motor and display the results on the screen graphically. The position error, torque command, encoder velocity, etc. may be viewed by clicking on the drop down list at the top of the window. The displayed graph of the position error is the error based on quadrature signal feedback from the encoder (for example there are 4000 counts or pulses per revolution on a 1000 line encoder).The response may be observed to verify proper performance for the programmed profile. If the response is acceptable, Quit the servo screen and Save the configuration. You will now have to Compile and Download the project for the new servo information to permanently take effect.

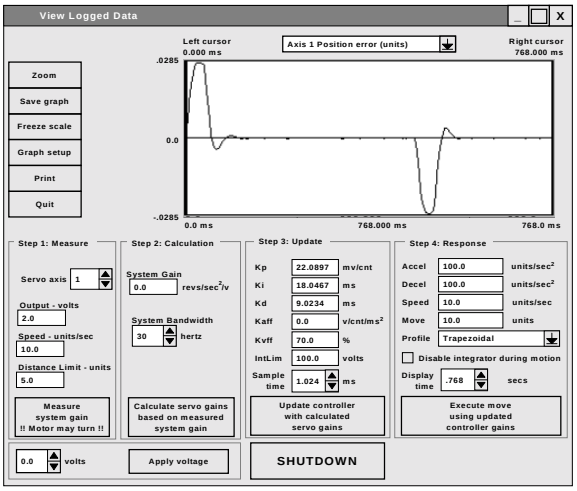
## Step 4A: Response Fine Tuning

### Integrate During Motion

This feature allows you to select whether the integration gain is used during the profile motion. Enabling the integrator during motion will reduce your position error at speed, but may cause some unacceptable overshoot in the response. Some controllers allows this parameter to be set in the servo tuning screen, while others require that the change be completed in the Servo folder in the program configuration. Be sure to compile and download the project each time a change is made to the configuration or the change will not take effect. Stable responses with and without integration during motion enabled are shown below.



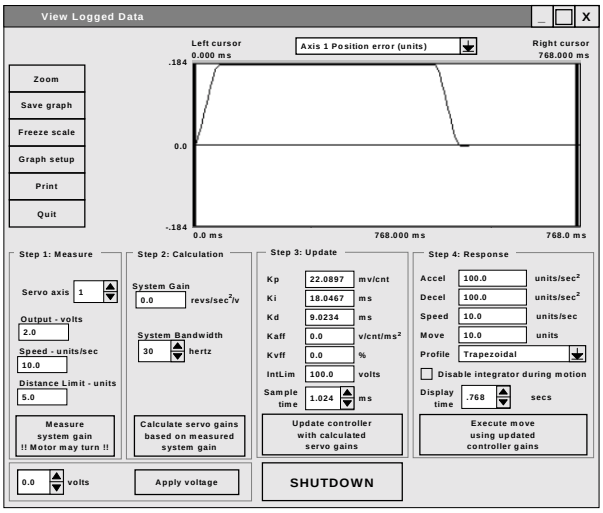

**Stable response with integration during motion disabled**



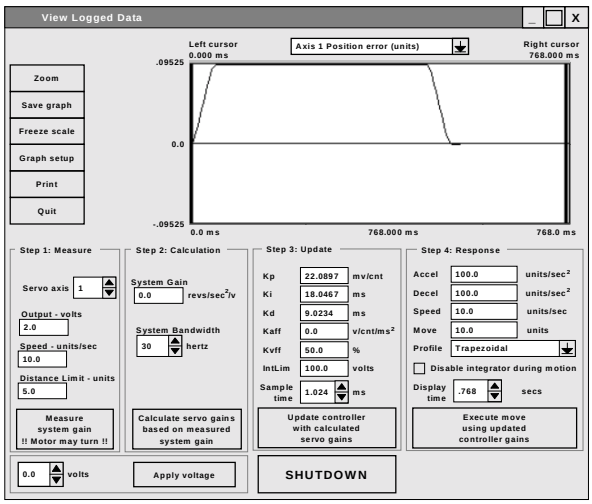

**Stable response with integration during motion enabled**

### Velocity Feed Forward

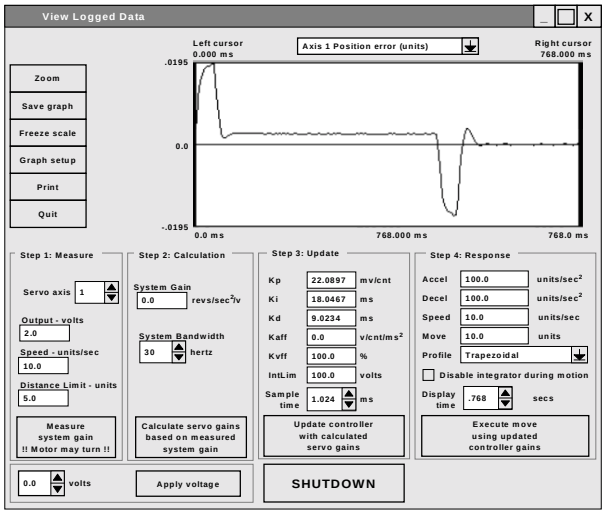
This term reduces the error during motion. It should typically be set between 50% and 100%. The figures below show a response with Kvff set to 0%, 50% and 100%. In all three cases the integration during motion was disabled, although integration can be enabled if it yields the response required.




**Response with Kvff = 0%**




**Same profile as above with Kvff = 50%**
**Note reduction in error**




**Same profile as above with Kvff = 100%**
**Note reduction in error**

# 9.1.1.6 - Manual Tuning Adjustment

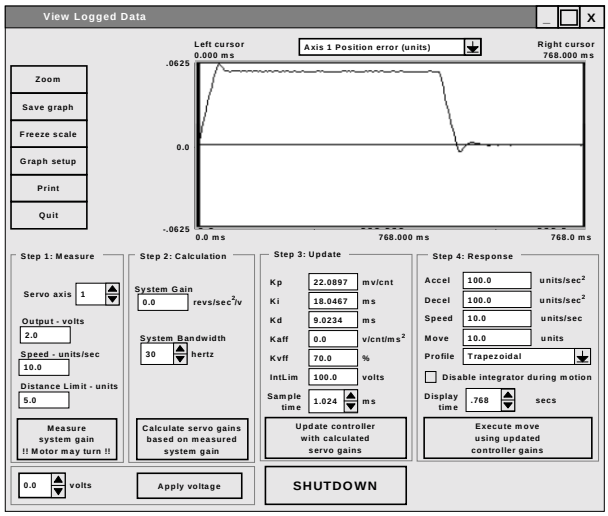
Most applications work acceptably using the results of the auto tuning procedure. However, if the results of the auto tuning sequence do not yield a satisfactory move response, the servo gains may be adjusted manually to achieve the required performance. Manual tuning of the servo can be quite involved. Be sure to read this section a few times through before deciding to begin manual adjustments.

The single most important rule to remember when adjusting the servo manually is to **gradually change one gain at a time**. There can be interactions between the parameters that will affect the response, and changing more than one gain at a time will certainly lead to confusion.

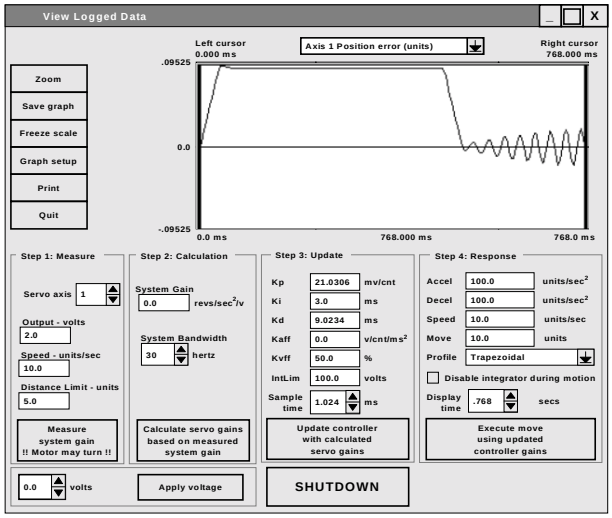
First let's begin with some definitions along with a description of each parameter and its function. The control loop uses a modified **PID** algorithm to compensate the system response. The servo parameters adjust the controller's output torque command based on **position error**, i.e. the difference between commanded position and encoder position at any given point in time. The **encoder velocity** and **commanded velocity** are also used in some cases. Each parameter contributes to the output torque command in a different way.

**Stability or instability:**

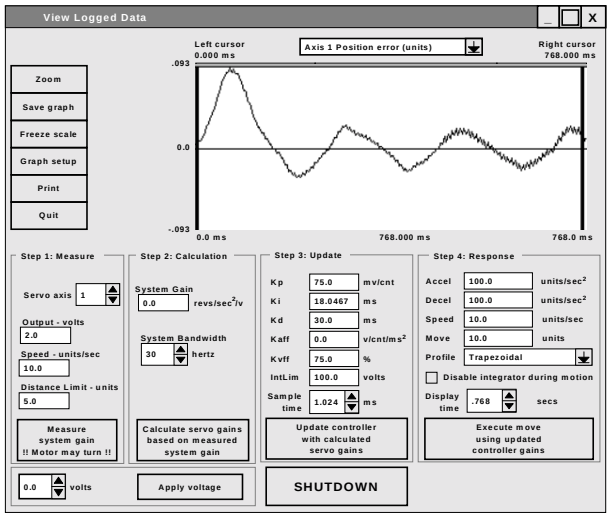
If the servo system behaves smoothly and without loud buzzing, vibration or oscillation it is said to be **stable.** Conversely, if the system buzzes, vibrates, or oscillates it is said to be **unstable**. The first goal of servo tuning is to achieve a stable system. Once stable the system may be adjusted or "tweaked" to optimize performance. Adjustments should only be made if the response is outright unacceptable. The figures below show a stable and unstable system response.




**Shows stable response**




**Shows unstable response (due to Ki to low)**




**Shows unstable response (due to Kp and/or Kd too high) Note "fuzz" from motor "buzzing"**

**Kp:**

**Proportional gain**. This gain is multiplied by the position error and thus contributes **proportionally** to the output torque. Generally, the higher the Kp, the lower the error at any time during the move. However, if Kp is too high, the system can overshoot severely or "buzz" loudly. This type of buzzing instability may be seen as "grass" on the error response curve in the move response screen. In this case, Kp should be lowered. Kd may also be lowered, but to a lesser extent.

Generally the range for Kp is 10 to 150. Kp less than 10 will usually produce a soft or sluggish system. Kp over 175 produces a stiff system, but one that may be approaching instability. Note these are general ranges, not absolute requirements.

**Ki:**

**Integral gain.** The reciprocal (1/Ki) of this term is multiplied by the **sum** of the position error over time. The effect of Ki is thus time related, and affects the steady state error. The higher Ki, the longer it will take for the controller to "integrate out" any steady state error. The effect of Ki is seen mostly at constant speed (including standstill). Ki is NOT required for stability, and generally has a de-stabilizing effect on the system, especially if it is too low. If Ki is TOO LOW the system may oscillate slowly and wildly back and forth like a washing machine. Ki is required, though, if the system must achieve a very low steady state error (within a few counts).

The general range for Ki is 10 to 70. Ki less than 10 may lead to wild, low frequency oscillations. If steady state error is not a consideration, Ki may be set to zero. Ki is often disabled during motion to reduce overshoot at the end of the move.

**Kd:**

**Derivative gain**. This term is multiplied by the encoder velocity at any point in time. Generally, raising Kd will reduce overshoot in the move response, however, Kd is the term most susceptible to "digital instability". This is where the quantification effects of the digital encoder feedback in conjunction with too high a Kd cause the system to "buzz".

The general range for Kd is 5 to 20. Kd less than 5 usually leads to an unstable system, Kd >20 usually leads to "buzzing".

**Kvff:**

**Feed forward velocity gain**. This term is multiplied by the commanded velocity to contribute to the output torque command. It has no effect on general stability, and may be set to as high as 100% to reduce position error during motion. Too high a Kvff causes undue motor heating.

Generally, Kvff should be set between 50 and 100.

**Kaff:**

Some controllers have a Kaff term. This term is multiplied by the commanded acceleration to contribute to the output torque command. This term only takes effect to reduce the error during acceleration and deceleration. Generally Kaff is less than 4. Most applications will run fine with Kaff set at zero.

## 9.1.1.6.1 - Adjustment based on auto tune calculation

It is usually desirable to use the auto tuning gains as a starting point for further adjustment. If the system is unstable at given bandwidth, the bandwidth may be lowered, and the auto tuning run again. If the move response at this lower bandwidth is unacceptable, the following procedure may be attempted.

Set bandwidth to 25 Hz and calculate gains. Then:

1) Update gains and energize system .
2) If the system "buzzes", cut Kp in half, and lower Kd by 25%.
3) If the system no longer buzzes, check your move response.
4) If the move response over shoots too much, or the system buzzes sometimes, then lower Kp until the buzz goes away and the overshoot is acceptable.
5) Check your move response, and set Kvff to between 50-100%. This should reduce the error during the move, and may also improve the overshoot.
6) If the response is well behaved, but sluggish, raise Kp in increments of 2 until acceptable response is achieved. If ever the system "buzzes" Kp must be lowered again.
7) Verify proper response.
8) The system should now be stable and well behaved.

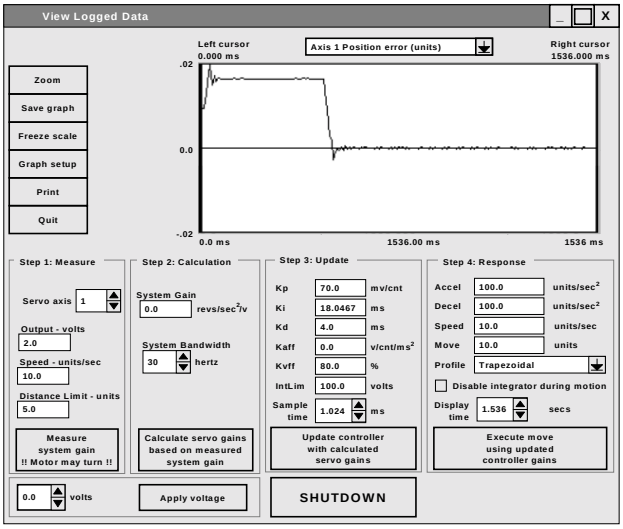
## 9.1.1.6.2 - Full Manual Adjustment

Although it is much more involved, the servo can be tuned "from scratch". The trick here is to be very **patient and methodical.** Make sure to record each change and its resultant effect on the response. In step 1 the **measure system gain** button is used to determine proper encoder direction. Step 2 is not used at all. Step 3 is used to enter and update the servo gains. Step 4 is used to enter the move profile parameters and execute a move response. Make sure to Update Gains after each adjustment so they take effect. You can use the example response screens at the end of this procedure as a guide. **CAUTION! Motor instability can cause severe vibration or sudden movements. Insure that appropriate safety measures such as mechanical limits are employed to prevent dangerous movements of the motor and load.**

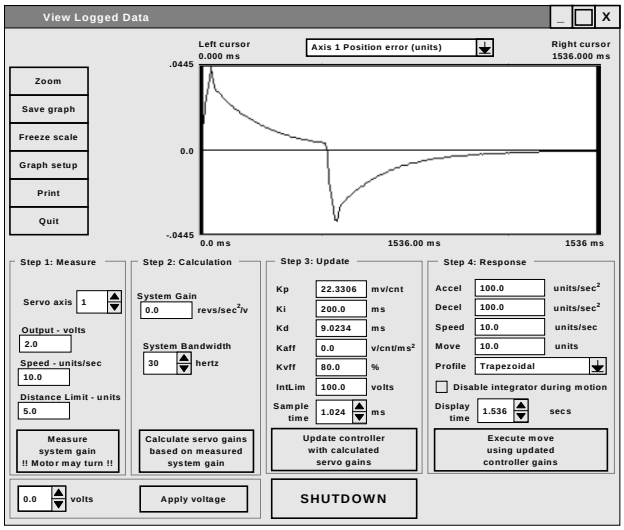
1) Click **measure system gain**. **Caution! the motor will move suddenly during this process.** This will verify that the encoder direction is correct for the servo to run properly.
2) If the encoder direction is found to be reversed, then quit the auto tune screen immediately and enter the Configuration. Select the encoder folder and change the encoder direction to the opposite of the present setting. Save the configuration information, compile and download the modified project.
3) Re-enter the servo tuning screen and set Ki, Kvff, and Kaff to zero.
4) Together, set Kp to a low number, say 5, and Kd to a mid-range number, say 10.
5) Update the gains and see if the motor is stable by moving the load slightly by hand (if this is safe). Be ready to **shutdown** if the motor oscillates.
6) If the motor is stable and does not vibrate, raise Kp by 2.
7) If not, lower Kp by 1. Repeat until the motor is stable.
8) Once Kp is as high as it will go and still be stable, reduce Kp by **50%** to provide some stability margin.
9) Now try your move response.
10) If the move is stable but overshoots severely, lower Kp slightly. Slight overshoot is o.k. at this point.

11) Continue lowering Kp until the overshoot is close to acceptable.
12) Now we can try to reduce the error during the motion.
13) Set Kvff to 50 and check the response.
14) If the error is not acceptable increase Kvff by 10 and check the response, repeat until the response is acceptable.
15) Now let's try to use Ki to reduce the error at rest.
16) Set Ki to a high number, say 75 and check the move response.
17) If the response smooth out takes a long time to settle at the end, then decrease Ki by 10. If the motor goes unstable, raise Ki back up again.
18) Verify the proper response to your profiles.
19) If the response still exhibits oscillation or overshoot, you may need to dampen the system response by raising Kd and repeating the process from step number 5. See the effect of lower Kp and higher Kd in the response graphs below.
20) If the motor will not respond as required, check the torque command response in the pull down window to verify that the controller is not saturating at 10 volts during accel/decel. This would indicate too high an acceleration for this motor and load. Lower the accel or decrease the load inertia.
21) **THAT'S IT!**

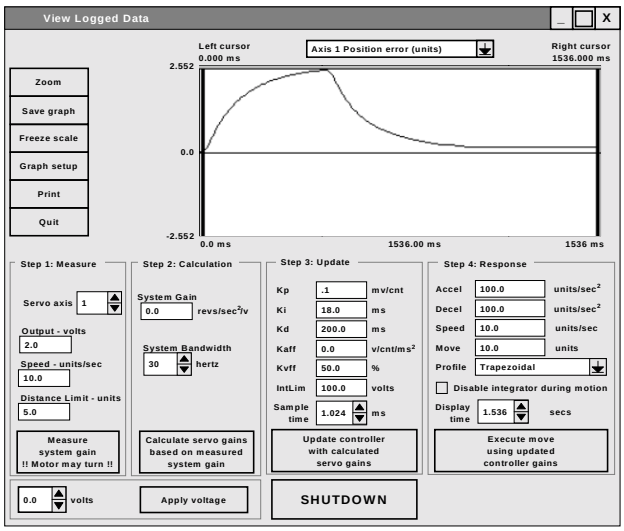
The following screens show examples of tuning responses. Each has a description of what caused the response shown.



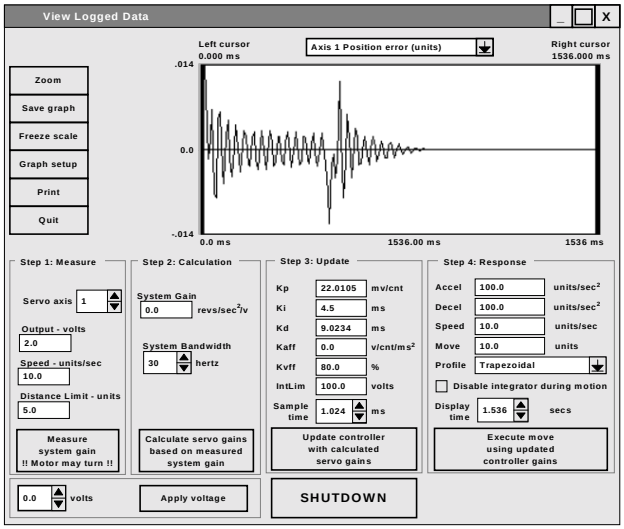

**Stiff response**
**(with high Kp and low Kd)**



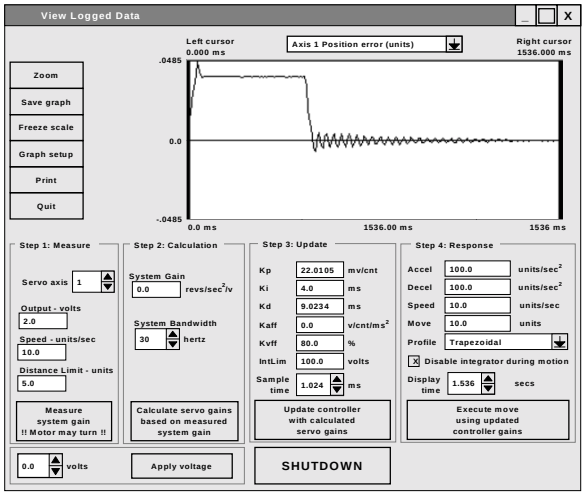

**Response with high Ki and**
**integration enabled during motion**
**Note very long settling time**



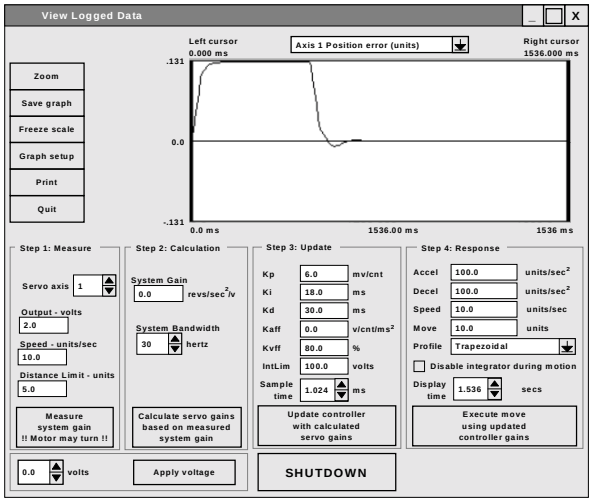

**Sluggish response**
**(due to low Kp and high Kd)**



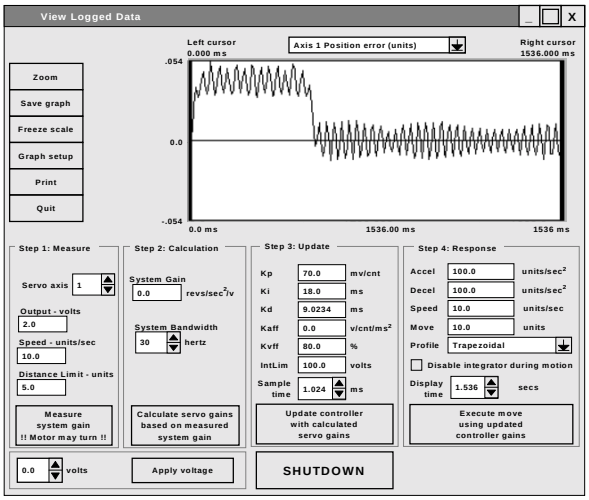

**Response with low Ki and**
**integration enabled during motion**
**Note excessive ringing**



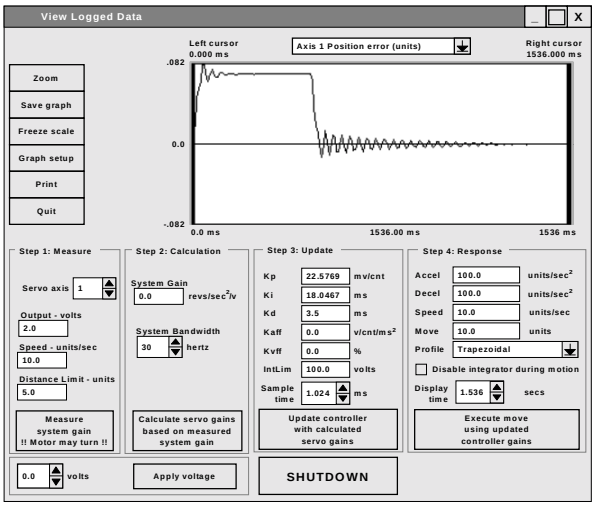

**Response with low Ki and integrator disabled during motion**
**Note excessive ringingout at the end of the move only. The integrator is engaged when the profile stops**



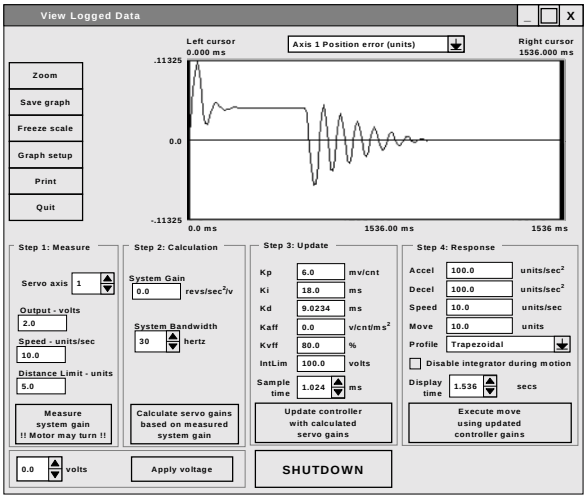

**Previous profile with Kd raised to dampen out oscillation.**



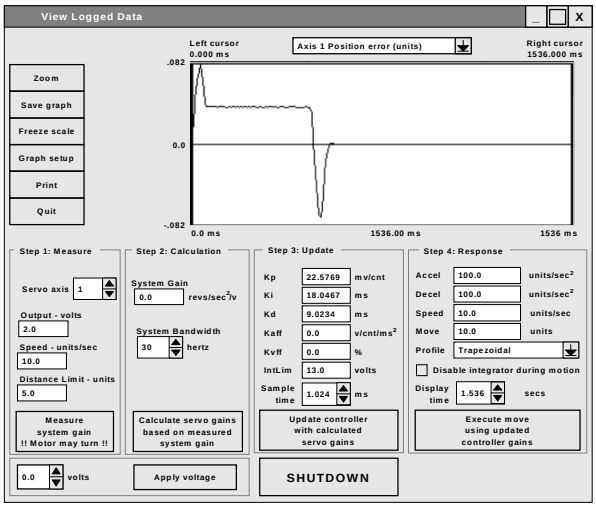

**Response with high Kp**
**Note instability can be seen as vibration.**
**Kp should be lowered to eliminate instability.**



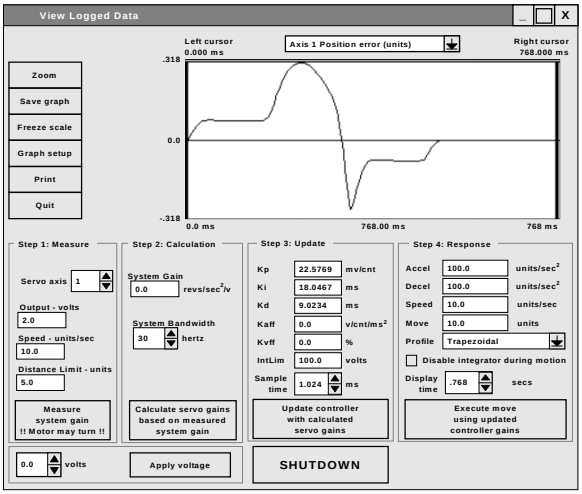

**Response with low Kd.**
**Note oscillation at end of profile.**



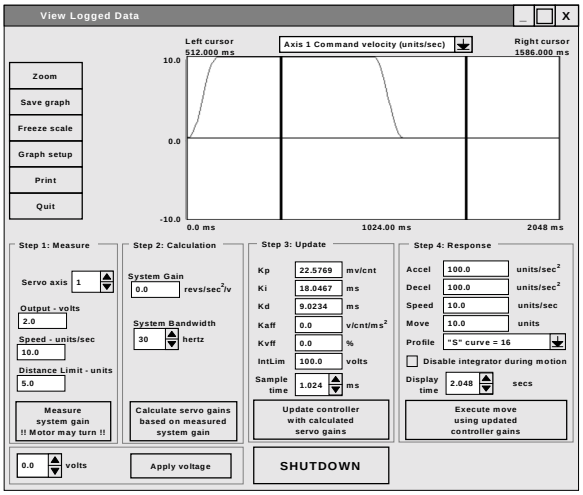

**Response with low Kp. Note oscillation.**
**If Kp can not be raised, Kd may be raised to reduce the ringing as shown below.**



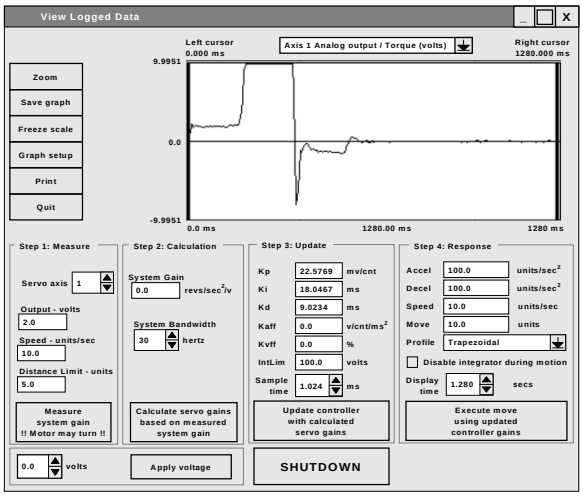

**Response if IntLim is too low and integrator enabled during motion.**
**Note that the integrator cannot bring the error to zero during the flat top part of the profile**



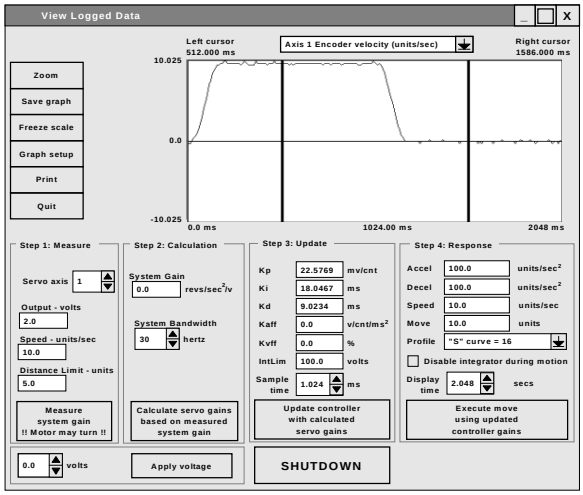

**Response if the programmed speed is too high for the motor. This also can be caused by the drive running at too low a bus voltage.**




**Command velocity profile**




**Response of the torque command for the previous profile with speed set too high. Note that the torque command saturates at 10 volts. Any time the command goes + or -10V, the motor is not producing the required torque to bring the error down.**




**Encoder velocity profile**

## 9.1.2 Excessive Duty Cycle Shutdown

As the servo system responds to shaft displacement due to move commands or reaction torque's, the servo amplifier produces current to drive the motor. A feature has been added that prevents the unit from generating too much current and/or motor heating due to an excessive duty cycle situation. Here, duty cycle refers to the percentage of time that the system is required to generate a current (and therefore resultant torque) above its continuous rating.

The peak current is assumed to be twice the continuous current rating of the servo amplifier. Also, the servo amplifier produces peak current when a 10-volt signal is applied to the amplifier. For example a 4 ampere continuous current rating (5-volt amplifier signal) would produce a peak available current of 8 amperes (10-volt amplifier signal). The continuous current can be maintained indefinitely. However, currents above the continuous current rating (up to the peak current) can only be generated for a limited length of time before damage to the servo amplifier and/or motor will result. If the amplifier and motor are allowed to cool (i.e. the motor rests for a short period) as a result of the current dropping below the continuous current rating, then repetitive occurrences of currents above the continuous current rating may be acceptable.

If an excessive duty cycle situation occurs, the user task will error trap and all motors in the task will be stopped. The servo axis with the excessive duty cycle will be disabled resulting **in the motor shaft spinning freely** unless it is held by an external brake. Error code 26, **IXT Servo Error** will be generated for this axis. The axis that created the error can be interrogated using the **ERRAXIS** command. The error that created the error trap can be interrogated using the **ERR** command. This error can be cleared by commanding an **ERR=0,0** statement in the error handler.

The excessive duty cycle is defined as a time that the amplifier is saturated (peak current). The default time is 3 seconds and can be changed using basic command **IXT** in a user task. The excessive duty cycle check can be disabled if the time value for the IXT command is zero. A **Peak Current vs. Timeout chart** is depicted below. This chart can be used to calculate the excessive duty cycle time for currents above the continuous current rating of the amplifier.

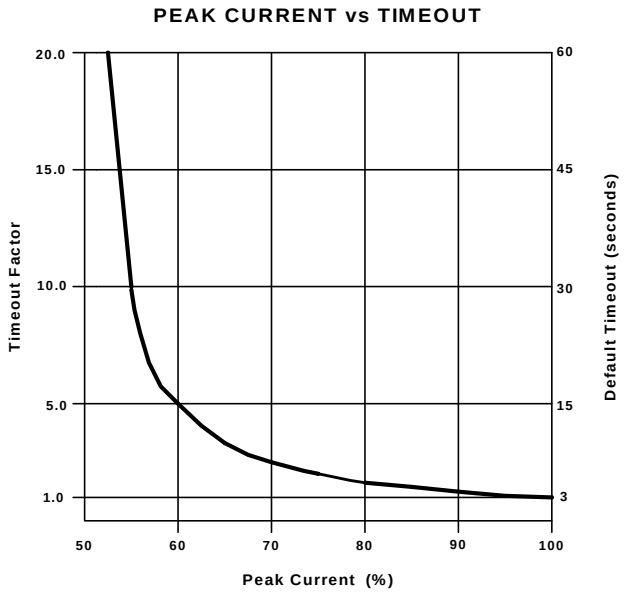
Example:

Amplifier continuous rating is 4 amperes.
Amplifier peak current is 8 amperes.
IXT time is set to 3 seconds (default)

| Continuous Current (Amperes) | Current (%) | Timeout (seconds) |
|---|---|---|
| 4.2 | 52.5 | 60 |
| 4.4 | 55 | 30 |
| 4.8 | 60 | 15 |
| 5.2 | 65 | 10 |
| 5.6 | 70 | 7.5 |
| 6.0 | 75 | 6 |
| 6.4 | 80 | 5 |
| 6.8 | 85 | 4.286 |
| 7.2 | 90 | 3.75 |
| 7.6 | 95 | 3.333 |
| 8.0 | 100 | 3 |

IXT trip point = (rated peak current * .5) * IXT time
IXT trip point = (8 * .5) * 3 = 12 amp seconds

Timeout (secs)=IXT trip point/(current – cont. rated)
Timeout (secs)= 12/(6-4) = 6 seconds for 6 amperes



# FOLERR Motion Parameter

**ACTION:**

Sets or returns the maximum position error allowed during motion, herein referred to as "following error."

**COMMAND SYNTAX:**

FOLERR(axis)=expression
FOLERR=expression1, number2, . . . , number8
FOLERR(axis, … , axis)=expression, … , expression
FOLERR (axis) - Used in an expression

**Note: ENCFOL can be substituted for FOLERR.**

**REMARKS:**

The axis specifies the number of the axis (1-8).

The expression specifies the maximum position error allowed during motion in units.

Position error = absolute position - encoder position.

**EXAMPLES:**

FOLERR(2)=.4
Sets the following error of axis 2 to .4 units.

FOLERR=.4,, .3
Sets the following error of axis 1 to .4 units and axis 3 is set to .3 units.

FOLERR(1,3)=.4,.3
Sets the following error of axis 1 to .4 units and axis 3 is set to .3 units.

# INTLIM Servo Parameter

**ACTION:**

Sets the Integral limit for the servo output. This is the limit of the contribution to the servo output from the integral of the position error.

**PROGRAM SYNTAX:**

INTLIM (axis)=expression
INTLIM=expression1, ... , expression8
INTLIM (axis,...,axis)=expression, ... ,expression
INTLIM (axis) - used in an expression

**REMARKS:**

The axis specifies the number of the axis (1-8).

The setting limits the contribution of the integral term to the servo loop's output. This limit is imposed on the internal calculation within the controller, and is used to prevent excessive buildup of the integrator output which can occur if a constant error is allowed to exist for extended periods of time. Too low an integral limit may reduce the effectiveness of the integrator by limiting its contribution to the output torque command. This would cause a constant steady state error. Too high an integral limit may allow the integrator to build up a large error stored in the controller memory. This error would then be “unwound” at the end of a move causing excessive overshoot and a long settling time. The limit can be set between 0 and 319 volts. A setting of 100 is a good midrange starting point, and this parameter rarely needs adjustment.

If the input value is out of range, the previous setting is retained. Reading INTLIM returns the present setting in volts.

**EXAMPLES:**

```
INTLIM(2) = 5          ‘ sets the integral limit for axis 2 to 5 volts.

X = INTLIM(2)          ‘ sets x to the integral limit of axis 2.
```

# IXT Servo Parameter

**ACTION:** Sets or returns the Excessive Duty Cycle Shutdown time in seconds.

**PROGRAM SYNTAX:**

IXT(axis) = expression
IXT(axis, … , axis) =expression, … , expression
IXT = expression, … , expression
IXT(axis) - used in an expression

**REMARKS:** The axis specifies the number of the axis (1-8).

The expression specifies the time the servo peak current can be maintained. The time value is in seconds and the default value for each axis is 3 seconds. Setting the expression equal to 0 will disable the Excessive Duty Cycle Shutdown check.

**Caution: Disabling the Excess Duty Cycle or setting the time too large may result in damage to the servo drive and/or motor if the duty cycle of the servo amplifier is exceeded.**

The **IXT(axis)=expression** program command should precede the **WNDGS(axis)=1** command.

The default value for IXT is set each time a project is loaded or executed. Thus, adding an IXT basic command to a task is the only way to change the default value.

If an Excessive Duty Cycle Shutdown occurs the user task will error trap and all motors in the task will be stopped. The servo axis with the excessive duty cycle will be disabled resulting in the motor shaft spinning freely unless it is held by and external brake. Error code 26, **IXT Servo Error** will be generated for this axis.

**EXAMPLES:**

```
IXT(1) = 5
‘ sets the Peak Current time for axis 1 to 5 secs.
WNDGS(1)=1
‘ enable the servo drive on axis 1.

IXT(1,3) = 5,6
‘ sets the Peak Current time for axis 1 to 5 secs and axis 3 to 6 secs.
WNDGS(1,3)=1,1
‘ enable the servo drive on axis 1 and axis 3.

IXT = 5, ,6
‘ sets the Peak Current time for axis 1 to 5 secs and axis 3 to 6 secs.
WNDGS(1,3)=1,1
‘ enable the servo drive on axis 1 and axis 3.

time = IXT(1)
‘ return the Peak Current time setting of axis 1
```

# KAFF — Servo Parameter

**ACTION:** Sets or returns the acceleration feed forward gain for a servo axis.

**PROGRAM SYNTAX:**

**KAFF(axis)=expression**
**KAFF=expression1,..., expression8**
**KAFF(axis,…,axis)=expression,...,expression**
**KAFF(axis) - used in an expression**

**REMARKS:** The axis specifies the number of the axis (1-8).

The expression is the acceleration feed forward gain of the servo axis. The expression value must be positive.

The KAFF units are in volts/encoder count/msec$^2$.

**EXAMPLES:**

**KAFF(2)=.5**
Sets the acceleration feed forward gain of axis 2 to .5 volts/encoder count/msec$^2$.

**KAFF=.2,,0**
Sets the acceleration feed forward gain of axis 1 to .2 volts/encoder count/msec$^2$ and axis 3 is set to 0 volts/encoder count/msec$^2$.

**KAFF(1,3)=.2,0**
Sets the acceleration feed forward gain of axis 1 to .2 volts/encoder count/msec$^2$ and axis 3 is set to 0 volts/encoder count/msec$^2$.

# KD — Servo Parameter

**ACTION:** Sets or returns the derivative gain for the servo axis.

**PROGRAM SYNTAX:**

KD(axis)=expression
KD=expression1, ... , expression8
KD(axis, ... ,axis)=expression, ... ,expression
KD(axis) - used in an expression

**REMARKS:** The axis specifies the number of the axis (1-8).

The expression is the derivative gain value of the servo axis. The expression value must be positive.

The KD units are milliseconds.

KD must be non-zero for system stability. Setting KD affects the gains for the velocity and feed forward terms. Reading KD returns the present setting.

**EXAMPLES:**

KD(2)=4
Sets the derivative gain of axis 2 to 4 milliseconds.

KD=10,,8
Sets the derivative gain of axis 1 to 10 milliseconds and axis 3 is set to 8 milliseconds.

KD(1,3)=10,8
Sets the derivative gain of axis 1 to 10 milliseconds and axis 3 is set to 8 milliseconds.

# KI Servo Parameter

**ACTION:** Sets or returns the integral gain of a servo axis.

**PROGRAM SYNTAX:**

KI(axis)=expression
KI=expression1, ... , expression8
KI(axis, ... ,axis)=expression, ... ,expression
KI(axis) - used in an expression

**REMARKS:**

The axis specifies the number of the axis (1-8).

The expression is the Integral gain value of the servo axis. The expression value must be positive.

The KI units are milliseconds.

KI determines how fast the integral term grows with a non-zero position error. The growth rate is inversely related to the value of KI. For example the integral term grows 5 times faster with KI=10 than with KI=50. A special case is Ki=0, which disables the integral action and set the integral term to zero. When the drive is disabled, the integral term is set to zero. Setting KI only affects the gain for the integral term. Reading KI returns the present setting.

**EXAMPLES:**

KI(2)=4
Sets the Integral gain of axis 2 to 4 milliseconds.

KI=1,,4
Sets the Integral gain of axis 1 to 1 milliseconds and axis 3 is set to 4 milliseconds.

KI(1,3)=1,4
Sets the derivative gain of axis 1 to 1 milliseconds and axis 3 is set to 4 milliseconds.

# KP Servo Parameter

**ACTION:** Sets or returns the proportional gain of the servo axis.

**PROGRAM SYNTAX:**

KP(axis)=expression
KP=expression1, ... , expression8
KP(axis, ... ,axis)=expression, ... ,expression
KP(axis) - used in an expression

**REMARKS:**

The axis specifies the number of the axis (1-8).

The expression is the proportional gain value of the servo axis. The expression value must be positive.

The KP units are millivolts/encoder count.

KP determines the size of the proportional term for a given position error. **Setting KP affect the gains for the proportional, integral, velocity and feed forward terms.** Reading KP returns the present setting.

**EXAMPLES:**

KP(2)=20
Sets the Proportional gain of axis 2 to 20 millivolts/encoder count.

KP=18,,20
Sets the Proportional gain of axis 1 to 18 millivolts/encoder count and axis 3 is set to 20 millivolts/encoder count.

KP(1,3)=18,20
Sets the Proportional gain of axis 1 to 18 millivolts/encoder count and axis 3 is set to 20 millivolts/encoder count.

# KVFF — Servo Parameter

**ACTION:** Sets or returns the velocity feed forward gain for the servo axis.

**PROGRAM SYNTAX:**

**KVFF(axis)=expression**
**KVFF=expression1, ... , expression8**
**KVFF(axis, ... ,axis)=expression, ... ,expression**
**KVFF(axis) - used in an expression**

**REMARKS:** The axis specifies the number of the axis (1-8).

The expression is the velocity feed forward gain value of the servo axis. The expression value must be positive.

The KVFF units are percent.

KVFF can be used to reduce the position error during motion. It does not affect system stability. The minimum error occurs with KVFF near 100%. Setting KVFF only affects the gain for the velocity feed forward term. Reading KVFF returns the present setting.

**EXAMPLES:**

**KVFF(2)=95**
**Sets the Velocity feed forward gain of axis 2 to 95%.**

**KVFF=98,,95**
**Sets the Velocity feed forward gain of axis 1 to 98% and axis 3 is set to 95%.**

**KVFF(1,3)=98,95**
**Sets the Velocity feed forward gain of axis 1 to 98% and axis 3 is set to 95%.**

# OUTLIMIT — Servo Parameter

**ACTION:** Sets or returns the servo command voltage limit.

**PROGRAM SYNTAX:**

**OUTLIMIT(axis)=expression**
**OUTLIMIT=expression1, ... , expression8**
**OUTLIMIT(axis, ... , axis)=expression, ... , expression**
**OUTLIMIT(axis) - used in an expression**

**REMARKS:** The axis specifies the number of the axis (1-8).

The expression is the OUTLIMIT value set for the designated axis.

Limits the magnitude of the servo loop's output voltage. OUTLIMIT is set to 10 volts at power up. OUTLIMIT can be set between 0 and 10 volts inclusive. Setting it to a value outside this range will cause it to be set to the nearest valid value.

**EXAMPLES:**

**OUTLIMIT(2)=5**
**Limits the magnitude of the servo output voltage for axis 2 to ± 5 volts.**

**OUTLIMIT=5,,10**
**Limits the magnitude of the servo output for axis 1 to ± 5 volts and axis 3 to ± 10 volts.**

**OUTLIMIT(1,3)=5,10**
**Limits the magnitude of the servo output for axis 1 to ± 5 volts (50% torque output) and axis 3 to ± 10 volts (100% torque output).**

# STOPERR Motion Parameter

**ACTION:**

Sets or returns the maximum position error allowed when motion is stopped, referred to herein as "position error band."

**COMMAND SYNTAX:**

STOPERR(axis) =expression
STOPERR=expression1, … , expression8
STOPERR(axis, … , axis)=expression, … , expression
STOPERR(axis) - Used in an expression

**REMARKS:**

The axis specifies the number of the axis (1-8).

The expression specifies the maximum position error allowed.

The STOPERR specifies the maximum position error allowed when motion is stopped for a servo motor without causing an error.

**EXAMPLES:**

STOPERR(3)=.1
Sets the maximum position error for axis 3 to .1 units

STOPERR=.1,,,.15
Sets the maximum position error for axis 1 to .1 units and axis 4 to .15 units.

STOPERR(1,4)=.1,.15
Sets the maximum position error for axis 1 to .1 units and axis 4 to .15 units.

# WNDGS Motion Parameter

**ACTION:** Enables or disable a servo drive.

**PROGRAM SYNTAX:**

WNDGS(axis)=expression
WNDGS=expression1, ... ,expression8
WNDGS(axis, ... , axis)=expression, ... , expression
WNDGS(axis) - used in an expression

**REMARKS:** The axis specifies the number of the axis (1-8).

The expression enables or disables the specified servo drive, a zero disables the servo drive and a non-zero enables the servo drive.

The WNDGS command is set to zero on power up. This ensures a safe condition for a servo drive.

Although the WNDGS command can be executed at any time, it becomes effective when no motion is taking place on an axis.

When a servo drive axis is disabled, the servo loop's integral term is zeroed and the servo loop output voltage is 0 volts. When the servo drive axis is enabled, the commanded position (ABSPOS) is set equal to the encoder position (ENCPOS). This forces the position error to zero so that the servo loop output does not cause unexpected motion.

**EXAMPLES:**

WNDGS(2)=1
Disables the servo drive on axis 2.

WNDGS=0,,1
Disables the servo drive on axis 1 and enables the servo drive on axis 3.

WNDGS(1,3)=0,1
Disables the servo drive on axis 1 and enables the servo drive on axis 3.

# Section 10
# Stepper Drive

## 10.1 – Stepper Features

The MX2000 provides some additional stepper drive controls features.

- Allows a starting speed for the stepper to be programmed.
- Reducing motor heating at standstill.
- Ability to increase motor current during motion.
- Configurable as an open loop or closed loop stepper drive.
- Position Verification and Correction capability on a closed loop stepper.

The starting speed of the stepper can be controlled using the LOWSPD command. A good starting point is 1.5 revolution/second. This is important if low speed mechanical resonance is encountered during acceleration or deceleration of the load.

A stepping motor can get hot when no motion is taking place, the selected drive current is flowing in the windings at standstill causing heating. This heating can be reduced by enabling the REDUCE current feature of the stepper, this reduces the drive current to 50% when the motor is standing still. Another method of reducing heating of the motor is turning off the current to the windings at standstill. Some caution should be taken under certain conditions when doing this since the motor has no holding torque. The WNDGS command is used to control this.

A stepping motor may require some additional torque during acceleration or deceleration of the load. A 50% increase in current can be realized during motion with the use of the BOOST command. This should be used with caution since it produces additional motor heating during motion. A consideration to duty cycle should be taken into account when using this feature.

A stepper motor without an encoder must be configured as an open loop stepper in the System Configuration. The default configuration settings for the open loop stepper can be selected in the **Open Loop Stepper** folder in the user program configuration. Some of these setting can be modified during program execution. The Low speed setting can be modified by the LOWSPD command. Motor standstill current can be modified by the REDUCE or WNDGS command. The Motor Boost current setting can be altered by the BOOST command. The Steps per motor revolution and Motor current delay are only selectable in the Open Loop Stepper folder.

A stepper motor with an encoder can be used for position verification and or position correction. This feature can be selected in the user program configuration System folder by assigning this motor axis as a closed loop stepper. The default configuration for a closed loop stepper can be selected in the **Closed Loop Stepper** folder in the user program configuration. Some of these setting can be modified during program execution. The Low speed setting can be modified by the LOWSPD command. Motor standstill current can be modified by the REDUCE or WNDGS command. The Motor Boost current setting can be altered by the BOOST command. Error Action can be changed using the ENCMODE command. Following error can be modified using the FOLERR command and the Position Error can be modified using the STOPERR command. The Steps per motor revolution, Motor current delay, Correction attempts and Time between attempts are only selectable in the Closed Loop Stepper folder. The Encoder folder is used to configure the stepping motor encoder, the Encoder direction and Line count items are used to configure the stepper motor encoder.

## 10.2 - Open Loop Stepper Folder

This folder sets the **steps per motor revolution**, **Low speed**, **Motor standstill current**, **Motor boost current** and **Motor current delay** for an open loop stepper drive.

Open Loop Stepper

| | Steps per motor revolution | Low speed (units/sec) | Motor standstill current | Motor Boost current | Motor current delay (sec) |
|---|---|---|---|---|---|
| Axis 1 | 2000 | 1.5 | normal 100% | normal 100% | 0.05 |
| Axis 2 | 2000 | 1.5 | normal 100% | normal 100% | 0.05 |

**Steps per motor revolution** specifies the stepping motor drive setting for each axis.

**Low speed** specifies the starting speed of each axis in units/second.

**Motor standstill current** specifies the state of the motor current at standstill for each axis. The choices are normal (100%), reduced (50%) and off (0%).

**Motor boost current** enables or disables the boost current feature of the stepper drive during motion. The choices are normal (100%) and boost (150%).

**Motor current delay** specifies the time delay between current modes in seconds. This allows time for the drive to respond to the change in current level as a result of the BOOST or REDUCE command (see Program Command section).

## 10.3 – Closed Loop Stepper Folder

This folder sets the **Steps per motor revolution, Starting speed, Motor standstill current**, **Motor boost current, Motor current delay, Error Action, Following error**, **Position error**, **Correction attempt**s and **Time between attempts** for a closed loop stepper drive.

Closed Loop Stepper

| | Steps per motor revolution | Low speed (units/sec) | motor standstill current | Motor Boost current | Motor current delay (sec) |
|---|---|---|---|---|---|
| Axis 1 | 2000 | 1.5 | normal 100% | normal 100% | 0.05 |
| Axis 2 | 2000 | 1.5 | normal 100% | normal 100% | 0.05 |

Closed Loop Stepper

| | Error action | Following error (units) | Position error (units) | Correction attempts | Time between attempts (sec) |
|---|---|---|---|---|---|
| Axis 1 | disabled | 0.05 | 0.005 | 10 | 0.1 |
| Axis 2 | disabled | 0.05 | 0.005 | 10 | 0.1 |

**Steps per motor revolution** See open loop Stepper Folder for description.

**Low speed** See open Low Speed Folder for description.

**Motor standstill current** See Open Loop Stepper Folder for description.

**Motor boost current** See Open Loop Stepper Folder for description.

**Motor current delay** See Open Loop Stepper Folder for description.

**Error action** selects what action, if any, is taken by the controller when the commanded motor position does not match the encoder position within the range set by the **FOLERR** command (see programming commands). This is also referred to as a stall condition. Once the **FOLERR** range is exceeded, one of four things can happen according to the **Error Action** selected.

If **Error action** is **disabled**, the controller takes no action.

If **Error action** is **stop on error**, the motor will stop and a controller error will result (see **ERR** command). The fault light will illuminate.

If **Error action** is **correct on error**, separate correction attempts (moves) will be commanded to try and re-align the motor. The user may specify **how many correction** attempts will occur, and the **Time between attempts**. If after the specified maximum number of correction attempts the motor still is not aligned, motion stops and a controller error will result.

If **Error Action** is **Restart on error**, the entire move is restarted. The motor returns to the starting position of the move in progress, and attempts to repeat the move. If during this repeat cycle the motor stalls, the motor will again return to the start position and retry the move. Each stall and restart counts as a correction attempt. This continues until the motor reaches the desired position, or the maximum number of **correction attempts** is reached. In the case of the latter a controller error results and the fault light illuminates.

**Correction attempts** specifies the maximum number of consecutive attempts allowed when **error action** is set to **correct on error** or **restart on error** mode and the motor stalls.

**Time between attempts.** Specifies the time between correction attempts when **error action** is set to **correct on error** or **restart on error** mode and the motor stalls.

## 10.4 - Encoder Folder

This folder sets the Encoder direction and Encoder resolution for a closed loop stepper.

Encoder

| | Encoder type | Encoder direction | Line count (lines / rev) | pulse count (pulses/rev) |
|---|---|---|---|---|
| Axis 1 | quadrature | normal direction | 500 | 2000 |
| Axis 2 | quadrature | normal direction | 500 | 2000 |

**Encoder direction** determines how the encoder rotation direction is interpreted. The choices are normal direction or reverse direction.

**Encoder line count** defines the encoder resolution in lines. An Encoder with 1000 lines will provide 4000 counts/revolution, or quadrature counts. Set this value to the encoder line count of the motor.

## 10.5 - Special Programming Notes for Closed-Loop Stepper Operation

The parameters for closed loop are set in the project configuration of the user's program. These parameters are:

**Encoder resolution**
Number of lines the encoder has. The line count times four is the equivalent of encoder pulses/ revolution. The direction for this parameter controls the quadrature detection direction value.

**Encoder position error (units)**
Allowable error at standstill before a correction is required.

**Encoder following error (units)**
Allowable error during motion before an error is reported. **This value should be a minimum of 1/20 of a motor revolution**.

**Number of correction attempts allowed**
How many consecutive corrections cycles are allowed.

**Time between correction attempts (seconds)**
Time between correction attempts. Allows motor to settle out before correcting.

**Error action**
This setting selects what action, if any, is taken by the controller when the commanded motor position does not match the encoder position within the range set by the **FOLERR** command (see programming commands). This is also referred to as a stall condition. Once the **FOLERR** range is exceeded, one of four things can happen according to the **Error Action** selected. This can be changed during program execution using the **ENCMODE** command.

If **Error action** is **disabled (ENCMODE=0)**, the controller takes no action.

If **Error action** is **stop on error (ENCMODE=1)**, the motor will stop and a controller error will result (see **ERR** command). The fault light will illuminate.

If **Error action** is **correct on error (ENCMODE=2)**, separate correction attempts (moves) will be commanded to try and re-align the motor. The user may specify **how many correction** attempts will occur, and the **Time between attempts**. If after the specified maximum number of correction attempts the motor still is not aligned, motion stops and a controller error will result.

If **Error Action** is **Restart on error (ENCMODE =3)**, the entire move is restarted. The motor returns to the starting position of the move in progress, and attempts to repeat the move. If during this repeat cycle the motor stalls, the motor will again return to the start position and retry the move. Each stall and restart counts as a correction attempt. This continues until the motor reaches the desired position, or the maximum number of **correction attempts** is reached. In the case of the latter a controller error results and the fault light illuminates.

**Testing closed loop operation**
1) Send the following Host commands:

```
ABSPOS(axis)=0
ENCMODE(axis)=0
MOVE(axis)= 1          ' 1 rev of motor
```

2) After the motion is completed send:

```
ABSPOS(axis) : ENCPOS(axis)
```

3) If the absolute position and encoder position values and signs are alike the closed loop stepper is set up properly.

4) If the values are the same and the directions are reversed toggle the Encoder direction setting in the program Configuration. Recompile the program and download project and repeat steps 1-3.

5) If the values are different the encoder line count is not correct or the encoder is miss wired.

# BOOST — Stepper Parameter

**ACTION:**

Enables or disables the Boost Current feature or returns the boost enable status for the specified stepper axis. When enabled the **stepper drive** BOOST output turns on during motion. This causes the **stepper drive** to boost the motor current by 50% during motion.

**PROGRAM SYNTAX:**

BOOST(axis)=expression
BOOST=expression1, ... , expression8
BOOST(axis, ... , axis)=expression, ... , expression
BOOST(axis) - used in an expression

**REMARK:**

The axis specifies the number of the axis (1-8).

If the expression is true (non-zero) then the BOOST feature is enabled for the specified axis. If the expression is false (zero) then the BOOST feature is disabled for the specified axis.

**EXAMPLE:**

BOOST(7)=1
Enables the Boost feature for axis 7.

BOOST=1,,0
Enables the BOOST feature for axis 1 and disables the BOOST feature for axis 3.

BOOST(1,3)=1,0
Enables the BOOST feature for axis 1 and disables the BOOST feature for axis 3.

# ENCMODE Closed Loop Stepper Parameter

**ACTION:** Sets or returns the operating mode of a closed loop stepper axis.

**PROGRAM SYNTAX:**

ENCMODE(axis)=expression
ENCMODE=expression1, ... , expression8
ENCMODE(axis, ... ,axis)=expression, expression
ENCMODE(axis) - used in an expression

**REMARK:** The axis specifies the number of the axis (1-8).

The operating mode are:

0 closed loop disabled - operates open loop.
1 halt execution on excessive following error.
2 correct position on excessive following error.
3 restart move on excessive following error.

**Note: This command is only used for a Closed Loop Stepper.**

**EXAMPLE:**

ENCMODE(1)=0
Sets axis 1 to open loop operation.

ENCMODE=1,,2
Sets axis 1 to halt execution on excessive error and axis 3 to correct position on excessive following error.

ENCMODE(1,3)=1,2
Sets axis 1 to halt execution on excessive error and axis 3 to correct position on excessive following error.

# FOLERR Closed Loop Stepper Parameter

**ACTION:** Sets or returns the maximum position error allowed during motion, herein referred to as "following error."

**COMMAND SYNTAX:**
FOLERR(axis)=expression
FOLERR=expression1, number2, . . . , number8
FOLERR(axis, … , axis)=expression, … , expression
FOLERR (axis) - Used in an expression

**Note: ENCFOL can be substituted for FOLERR.**

**REMARKS:** The axis specifies the number of the axis (1-8).

The expression specifies the maximum position error allowed during motion in units.

Position error = absolute position - encoder position.

**EXAMPLES:** FOLERR(2)=.4
Sets the following error of axis 2 to .4 units.

FOLERR=.4,, .3
Sets the following error of axis 1 to .4 units and axis 3 is set to .3 units.

FOLERR(1,3)=.4,.3
Sets the following error of axis 1 to .4 units and axis 3 is set to .3 units.

# LOWSPD Stepper Parameter

**ACTION:** Sets or returns the Low Speed (starting speed) value of a stepping motor axis.

**PROGRAM SYNTAX:**
LOWSPD(axis)=expression
LOWSPD=expression1, ... ,expression 8
LOWSPD(axis, ... ,axis)=expression, ... ,expression
LOWSPD(axis) - used in an expression

**REMARKS:** The axis specifies the number of the axis (1-8).

The expression set the LOWSPD value of the specified axis in units/second.

This command is only used by a stepper axis and is zeroed if the axis is a servo.

**EXAMPLES:**
LOWSPD(2)=1.5 ‘ set axis 2 to 1.5 units/second.

LOWSPD=1.3,, 1.5 ‘ sets axis 1 to 1.3 units/second and axis 3 to 1.5 units/second.

LOWSPD(1,3)=1.3, 1.5 ‘ sets axis 1 to 1.3 units/second and axis 3 to 1.5 units/second.

# REDUCE Stepper Parameter

**ACTION:** Enables, disables the Reduce current or returns the enable status.

**PROGRAM SYNTAX:**

REDUCE(axis)=expression
REDUCE=expression1, ... , expression8
REDUCE(axis, ... , axis)=expression, ... , expression
REDUCE(axis) - used in an expression

**REMARKS:** The "axis" specifies the number of the axis (1-8).

When enabled, the stepper drive REDUCE output turns on when there is no motion. This causes the drive to reduce the motor current to 50%. This feature requires a compatible stepper motor drive.

If the expression is true (non-zero) then the REDUCE feature is enabled for the specified axis. If the expression is false (zero) then the REDUCE feature is disabled for the specified axis.

**EXAMPLES:**

REDUCE(7)=1
enables the REDUCE feature for axis 7

REDUCE=1,1,,0,0,0,1,0
enables the REDUCE feature for axis 1,2,7, and disables the feature for axis 4,5,6,8.

# STOPERR Closed Loop Stepper Parameter

**ACTION:**

Sets or returns the maximum position error allowed when motion is stopped, referred to herein as "position error band."

**COMMAND SYNTAX:**

STOPERR(axis) =expression
STOPERR=expression1, … , expression8
STOPERR(axis, … , axis)=expression, … , expression
STOPERR(axis) - Used in an expression

**Note: ENCBAND can be substituted for STOPERR.**

**REMARKS:**

The axis specifies the number of the axis (1-8).

The expression specifies the maximum position error allowed.

STOPERR specifies the position dead band allowed for a Closed Loop Stepper Drive. If this value is exceeded at standstill it creates a correction motion cycle, and moves to the zero error position.

STOPERR specifies the maximum position error allowed when motion is stopped for a Stepper Drive.

**EXAMPLES:**

STOPERR(3)=.1
Sets the maximum position error for axis 3 to .1 units

STOPERR=.1,,,.15
Sets the maximum position error for axis 1 to .1 units and axis 4 to .15 units.

STOPERR(1,4)=.1,.15
Sets the maximum position error for axis 1 to .1 units and axis 4 to .15 units.

# WNDGS Stepper Parameter

**ACTION:** Enables or disables a stepper motor drive, winding current controlled.

**PROGRAM SYNTAX:**

WNDGS(axis)=expression
WNDGS=expression1, ... ,expression8
WNDGS(axis, ... , axis)=expression, ... , expression
WNDGS(axis) - used in an expression

**REMARKS:** The axis specifies the number of the axis (1-8).

The expression specifies the state of the windings for the specified axis.

A zero indicated normal current or reduced current and a non-zero turns the stepper motor winding current off.

The WNDGS command is set to zero on power up. This insures a safe condition for a stepper drive on powered up.

Although the WNDGS command can be executed at any time, it becomes affective when no motion is taking place on an axis.

**EXAMPLES:**

WNDGS(2)=1
Sets the WNDGS state to 1 (Windings Off) on axis 2.

WNDGS=0,,1
Sets the WNDGS state on axis 1 to a 0 (Windings On) and axis 3 WNDGS state is 1 (Winding Off).

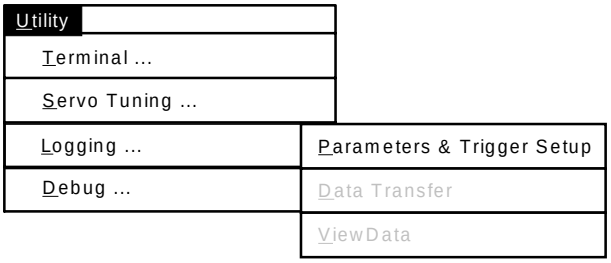
WNDGS(1,3)=0,1
Sets the WNDGS state on axis 1 to a 0 (Windings On) and axis 3 WNDGS state is 1(Windings Off).

# Section 11
# Data Logging

## 11.1 - Data Logging

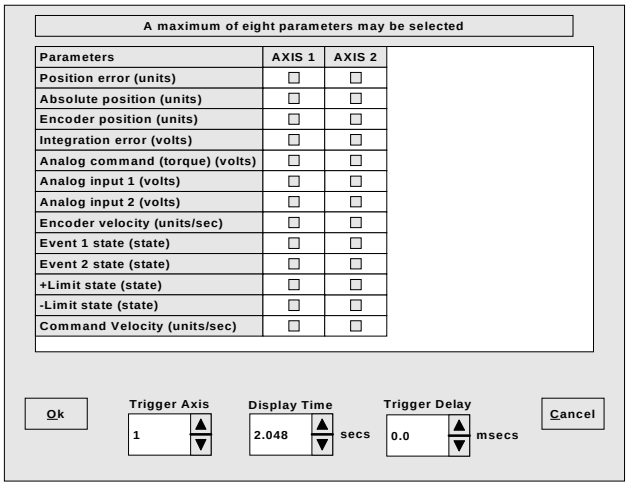
The controller has the capability to perform data logging of eight items when the selected trigger occurs. Each logged item has 200 points. The Data Logging is accessible from the **Utility** menu.

An MX2000 project in the PC needs to be open to perform data logging. Data logging can be triggered by either Host or program execution commanded motion.




## 11.1.1 - Parameter & Trigger Setup

The parameter & trigger setup is accessed by clicking on the **Utility** menu, **Logging** item and then **Parameter & Trigger Setup** item. A maximum of eight parameters, independent of the number of axes, may be selected for data logging. To select or deselect a data logging item click on the desired axis of the parameter check box.




The parameter selection list is: Position error, Absolute position, Encoder position, Integration error, Analog command, Accel feed forward, Analog input 1, Analog input 2, Encoder velocity, Event 1 state, Event 2 state, +Limit state, –Limit state and Command Velocity.

**Trigger Axis** selects the axis that will trigger the logging. The trigger occurs when the selected axis motion starts.

**Display Time** select the logging period for data logging in seconds.

**Trigger Delay** selects the delay, in millisecs, after the trigger occurs and data logging begins.

**Cancel** exits the logging Parameter & Trigger setup without saving the values.

**Ok** sends the parameter listing, trigger axis, display time, Trigger delay to the controller and arms the motion trigger for data logging. The terminal mode window opens at this time allowing motion to be commanded.

## 11.1.1.1 Parameter List Descriptions

**Position error** is the position difference between the commanded position and the encoder position. This waveform is in units.

**Absolute position** is the commanded position of an axis. This waveform is in units.

**Encoder position** is the encoder position of an axis. This is the actual position of a closed loop stepper or servo axis. This waveform is in units.

**Integration error** is the integration error contribution to the analog output voltage for a servo drive. This waveform is in volts.

**Analog command (torque)** is the commanded torque voltage for a servo drive axis or the commanded analog output voltage of an axis. This waveform is in volts.

**Analog input 1** is the IN+ input voltage referenced to AGND if single ended mode is selected or the differencial input voltage of an axis. This waveform is in volts.

**Analog input 2** is the IN- input voltage referenced to AGND if single ended mode is selected. This waveform is in volts.

**Encoder velocity** is the measured velocity of the encoder input. This waveform is in units/sec.

**Event 1 state** is the state of the event 1 input of the axis.

**Event 2 state** is the state of the event 2 input of the axis.

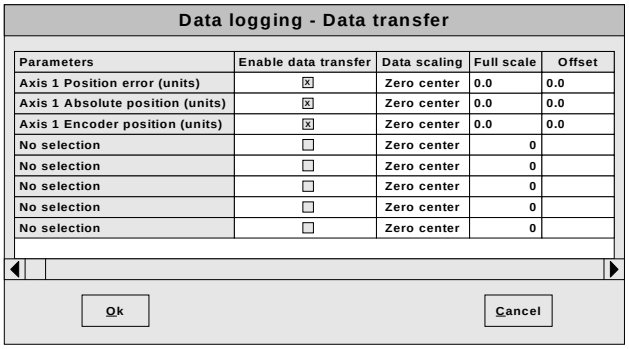
**+Limit state** is the state of the +limit input of the axis.

**-Limit state** is the state of the -limit input of the axis.

**Command Velocity** is the commanded velocity of the axis. This waveform is in units/sec.

## 11.1.2 - Data Transfer

After the selected motion trigger has occurred the individual logged parameter data transfer can be enabled, disabled and scaled. To select Data Transfer click on the **Utility** menu, **Logging** item and then **Data Transfer** item.




**Enable data transfer** allows the individual logged parameters transfer to be enabled or disabled. Clicking on the check box will toggle the transfer setting. Data can be selected or deselected for later viewing.

**Data scaling** scales the individual logged parameter for zero-centered, min-max or manual.

**Full scale** is only allowed if manual scaling is selected. This sets the peak value for the logged data.

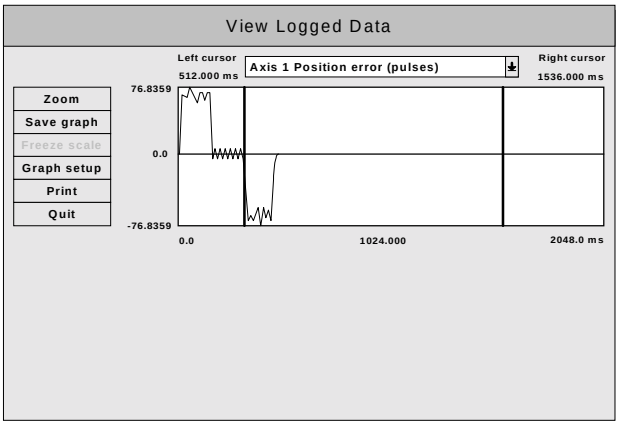
**Offset** is only allowed if manual scaling is selected. This sets the offsetvalue which represents the vertical center of the displayed graph.

**Cancel** exits the data transfer without transferring logged data.

**Ok** transfers the selected logged data from the controller. The View Data window opens allowing the transfered data to be viewed.

## 11.1.3 - View Data

The individual logged parameter can be viewed by clicking on the **Utility** menu, **Logging** item and then the **View Data** item.




**Zoom** toggles displaying the graph between the two cursors and the full screen on the view port.

**Save Graph** saves the currently displayed graph.

**Graph setup** allows for the selection of color and style for each logged item.

**Print** prints the currently displayed graph.

**Quit** exits the View logged Data environment.

**Display Drop list** selects the logged item to be displayed.

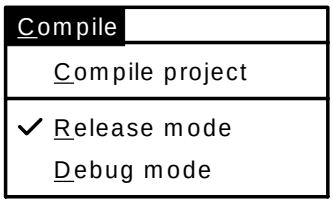
The cursors can be dragged to any horizontal position on the waveform. The elapsed time from the start of the waveform for the current cursor position is displayed above the logged waveform.

This page left intentionally blank

# Section 12
# DEBUG Environment

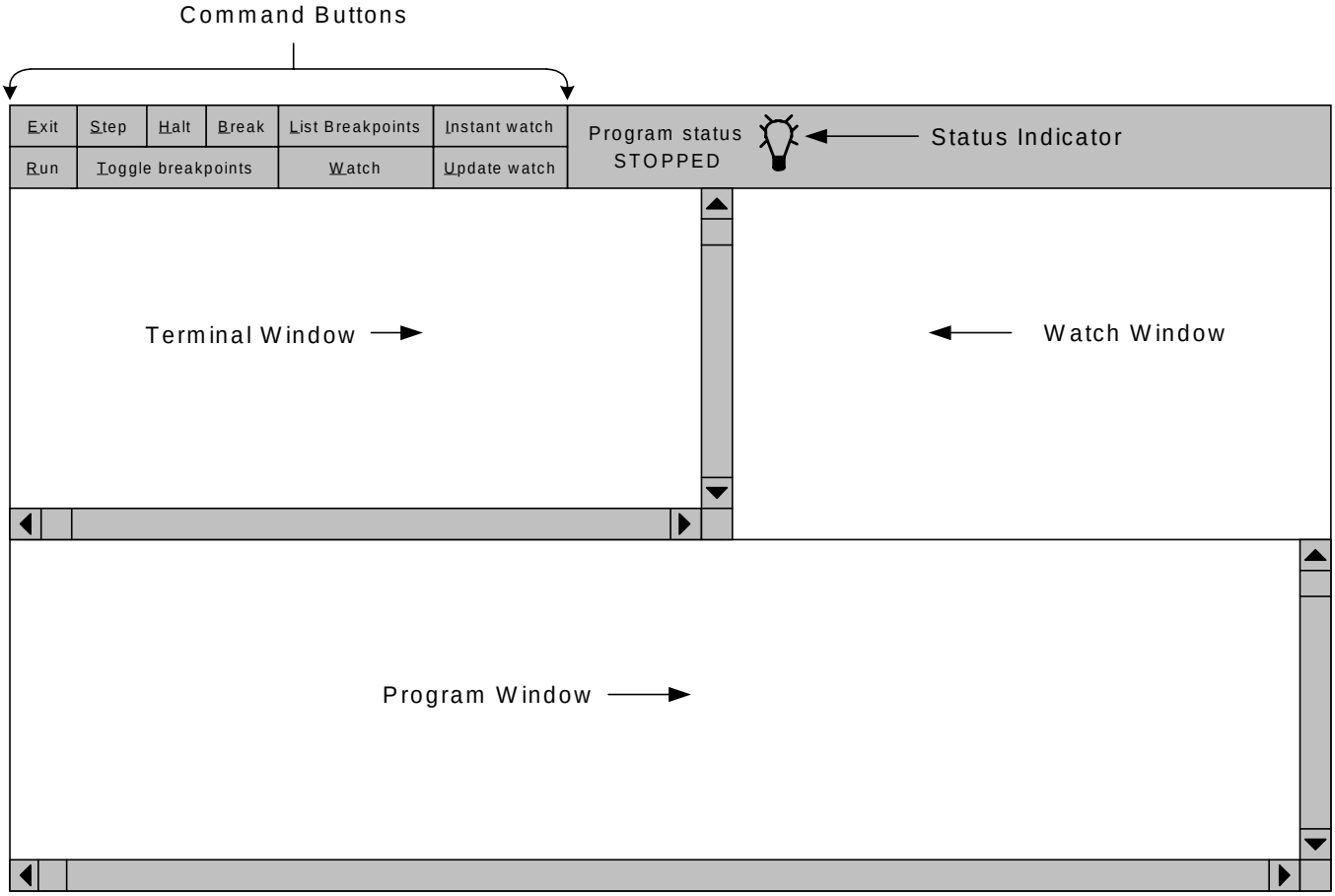
## 12.1 - Setting Project Debugging

To set the debug mode click on the **Compile** menu and then on the **Debug mode** item. The project must be compiled and downloaded before task debugging can begin. To cancel the debugging mode selection click on the **Compile** menu and then the **Release mode** item. To complete this cancellation the project must now be compiled and downloaded.




## 12.2 - Task Debugging

A project that is loaded into the controller can be debugged if the project has been compiled in Debug mode and downloaded. The project to be debugged must be open. To enter the debug environment click on the **Debug** command button. This environment consist of an **Exit** command button, **Step** command button, **Halt** command button, **Break** command button, **List Breakpoints** command button, **Instant Watch** command button, **Run** command button, **Toggle breakpoints** command button, **Watch** command button, **Update Watch** command button, program status indicator, **Terminal** window, **Watch** window and **Program** window.



## 12.2.1 - Debug program execution

A program can be executed in different ways from the Debug Environment. Single line execution of the current line can be initiated by clicking on the **Step** command button. The >>>>>> symbol preceding the line number indicates the line to be executed. The program can be executed to the next breakpoint encountered or end of program by clicking on the **Run** command button. Clicking on the **Halt** command button will stop a Running program. A program that is running can also be placed in the Single line execution mode by clicking on the **Step** or **Break** command button.

**Note: The program status indicator shows the status of program execution. The only time this status will indicate Stopped is when the program is halted or has executed an end statement in the program. The indicator is green for running and red for stopped.**

## 12.2.2 - Breakpoint Setting/Clearing

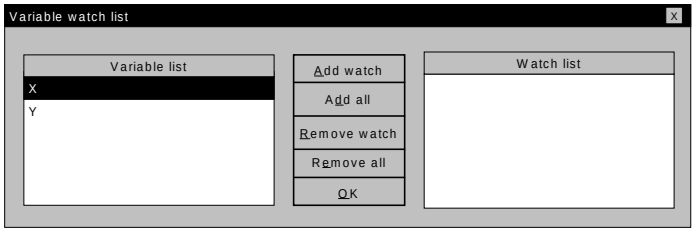
Up to five breakpoints can be set in debug mode. To change the breakpoint setting of a line, click on the desired line and then click on the **Toggle breakpoints** command button. When a line is set as a breakpoint, a **(BRK)** will precede the line. The breakpoint line numbers can be listed or cleared by clicking on the **List breakpoints** command button and than the appropriate command button.

## 12.2.3 - Terminal Window

The terminal window allows host command execution without leaving the Debug Environment. The Terminal Window is selected by clicking inside the Terminal window. A blinking cursor indicates that the Terminal window is selected for host commands.

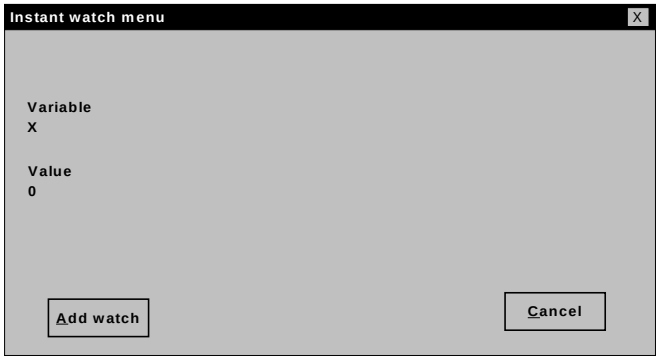
## 12.2.4 - Watch variables

The watch variable allows the programmer to view the values of selected variables. To add or remove a watch variable from the watch window click on the **Watch** command button.




To add a specific variable to the watch list, select the variable from the Variable **list** and then click on the **Add watch** command button. To remove a specific variable from the watch list, select the variable from the **Watch list** and then click on the **Remove watch** command button. To add all the variables to the watch list click on the **Add all** command button. To remove all variables from the watch list click on the **Remove all** command button. To return to the Debug Environment screen click on the **Ok** command button. The variable in the watch list will appear in the **Watch Window** and its current value will be displayed.

Another method of watching a variable is to highlight the variable and then click on the **Instant Watch** command button. The variable name and value will be displayed. This variable can be added to the watch window by clicking on the **Add watch** command button.




## 12.2.5 - Exit Debug Environment

The debug environment can be exited by clicking on the **Exit** command button.

This page left intensionally blank

# Section 13
# Application Examples

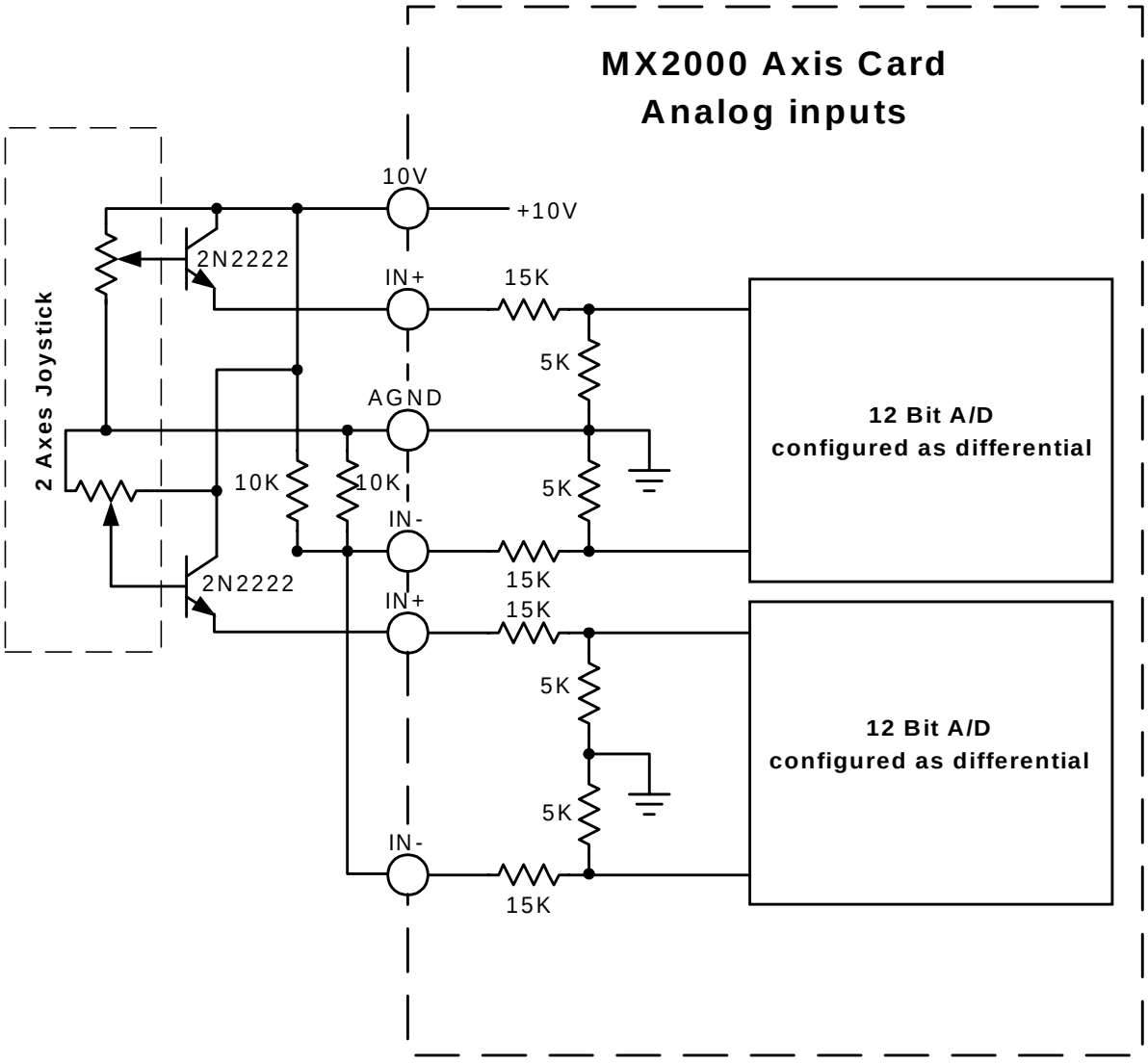
# 13.1 – Using a Joystick to teach an Arbitrary shape program

## 13.1.1 - MX2000 Joystick connection

A joystick is easily interfaced to the MX2000 analog inputs to control two motors. This allows positioning of a device for setup, or capturing positions for an arbitrary shape prior to machining, etc. The following diagram shows the method of connecting a two axis joystick to the MX2000 controller.

The analog inputs of the joystick axes must be configured as differential inputs in the user projects. The JOYSTICK basic command is used to enable the joystick mode of operation in the user program. The joystick mode can be canceled in the user program by execution of a STOP basic command on the joystick axes.

Each axis will run at a speed proportional to the input voltage and in the direction determined by the polarity of the input voltage. There is a ±0.25 volt dead band at the center of the input range, from +4.75 volts to +5.25 volts, and represents a speed of 0. The axis will run in the negative direction when the input voltage range is 0 volts to +4.75 volts. The speed it will attain is :
((4.75 – VIN) / 10) * SPEED(axis). The axis will run in the positive direction when the input voltage range is +5.25 volts to +10 volts. The speed it will attain is:
((VIN – 5.25) / 10 ) * SPEED(axis).



## 13.1.2 - Example Description

The example program allows an arbitrary shape to be taught, printed or executed. Four inputs on the axis card are used to accomplish this. Axis 1 (A side) Event 1 input is assigned as the **Teach** input. Axis 1 (A side) Event 2 input is assigned as the **Print** input. Axis 2 (B side) Event 1 is assigned as the **Register** input. Axis 2 (B side) Event 2 input is assigned as the **Execute** input. The **Teach** input switch must be a toggle switch and the remaining inputs can be momentary switches.

A two axis Joystick is connected to the designated axes analog input. This joystick is used to teach the arbitrary path to the controller. The circuit above should be used if possible to accomplish this.

The Program is broken into four distinct sections. The sections are main, execute taught program, print taught program, and teach program. These sections are described in detail below.

A sample program is included on the next page of this manual.

## 13.1.3 - Main Section

Moves the axes to the mechanical home positions and scans the Teach, Print and Execute inputs. When an input becomes active start executing the selected section.

## 13.1.4 - Teach section

This subroutine allows the user to trace an arbitrary shape by positioning, under joystick control, points on the shape's periphery. With one task running a maximum of 700 points are allows for a PATH command.

First the joystick is used to position the motors to the starting position for the desired shape. This position is recorded in NVR(1) and NVR(2) and becomes the starting position for the shape when the Register button is pressed.

Thereafter, pressing the Register button and then releasing it records the different points on the arbitrary shape. The coordinates of each point are automatically recorded into the MX2000 non-volatile memory. The X coordinates are captured in the even NVR elements and the Y coordinates are captured in the odd elements starting at element 4. NVR (3) contains the ending element of the coordinates captured.

The recording session is ended when the Teach input switch is open circuited.

## 13.1.5 - Print program section

Transmits ASCII text on the Auxiliary serial port that can be used as the program text to execute the arbitrary shape profile. Thus, this program will free up the non-volatile ram for another shape.

## 13.1.6 - Execute program section

This section allows the arbitrary shape program to be tested. The program ends after the arbitrary shape program is executed. If the shape needs a correction, print out the program and adjust the data in the appropriate NVR locations. Then restart the program and execute the arbitrary shape again.

```
************************* EXAMPLE PROGRAM *******************************
‘***** This example program allows a two axis arbitrary path Pattern to be taught, executed or printed.
‘***** Event 1 input, toggle switch, on axis 1 selects the Teach mode.
‘***** Event 2 input, momentary switch, on axis 1 prints the resulting program.
‘***** Event 1 input, momentary switch, on axis 2 registers the pattern points.
‘***** Event 2 input, toggle switch, executes the taught pattern.
‘***** NVR(1-2) is the x-y coordinates for the starting position of the pattern
‘***** NVR(3) is the ending element of the point array
‘***** NVR(4-5) is the first coordinate points of the Pattern
‘***** each additional set of points are in pairs
#DEFINE     AX1          1              ‘x axis defined
#DEFINE     AX2          2              ‘y axis defined
#DEFINE     DCNT         10             ‘input debounce count (msec)
POSMODE(AX1,AX2)=1,1                    ‘set absolute position mode
MOVEHOME(AX1,AX2)= -1, -1               ‘GOTO Mechanical Home in -direction
WAITDONE(AX1,AX2)                       ‘wait until mechanical cycle complete
E1_1=0                                  ‘initialize variable state
E1_2=0                                  ‘initialize variable state
E2_1=0                                  ‘initialize variable state
E2_2=0                                  ‘initialize variable state
SPEED=10,10                             ‘joystick speed for 10 volt differential voltage
DO
        state=0
        PRINT#1,”Select Teach Program, Print Program or Execute Program”
        DO
                GOSUB debounce_E1_1     ‘test Teach input
                GOSUB debounce_E2_1     ‘test Print input
                GOSUB debounce_E2_2     ‘test Execute input
                IF E1_1=1 THEN
                        GOSUB teach     ‘Teach input true
                        state=1
                ELSE IF E2_1=1 THEN
                        GOSUB prt_program ‘Print input true
                        state=1
                ELSE IF E2_2=1 THEN
                        state=1         ‘Execute input true
                END IF
        LOOP UNTIL state=1              ‘wait for an input being true
LOOP UNTIL E2_2=1                       ‘wait for Execute input being true
IF NVR(3) > 700 then                    ‘prevents operating system crash
        END
END IF
‘***** Execute Program and End
MOVE(AX1,AX2)=NVR(1),NVR(2)             ‘goto starting position of pattern
WAITDONE(AX1,AX2)                       ‘wait until at starting position
element=4                               ‘starting element for points
PATH=AX1,AX2                            ‘define path axes
        DO WHILE element  <  NVR(3)
                POINT=NVR(element), NVR(element + 1)
                element = element + 2
        LOOP
PATH END
END
```

```
prt_program:
        PRINT#2,"#DEFINE    AX1    1"
        PRINT#2,"#DEFINE    AX2    2"
        PRINT#2,
        PRINT#2,"POSMODE(AX1,AX2) = 1,1"
        PRINT#2,"MOVEHOME(AX1,AX2) = -1,-1"
        PRINT#2,"WAITDONE(AX1,AX2)"
        PRINT#2,"MOVE(AX1,AX2)=";NVR(1);",";NVR(2)
        PRINT#2,"WAITDONE(AX1,AX2)"
        PRINT#2,"PATH=AX1,AX2"
        FOR X=4 TO NVR(3) STEP 2
                PRINT#2,"    POINT=";NVR(X);",";NVR(X+1)
        NEXT X
        PRINT#2,"PATH END"
        PRINT#2,"END"
        state=1
        DO
                GOSUB debounce_E2_1          'test Print switch
        LOOP UNTIL E2_1=0                    'wait for Print switch to open
RETURN




'****** debounce Teach Input
debounce_E1_1:
        cnt = DCNT                           'debounce delay in msec
        DO
                IF EVENT1(AX1) = state THEN
                        RETURN               'return if same state
                ELSE
                        cnt = cnt – 1
                        wait=.001            'wait 1 msec
                END IF
        LOOP UNTIL cnt < 1                   'wait for debounce switch state change
        IF state=1 THEN
                E1_1 = 0                     'change state
        ELSE
                E1_1 = 1                     'change state
        END IF
RETURN                                       'return with different state
```

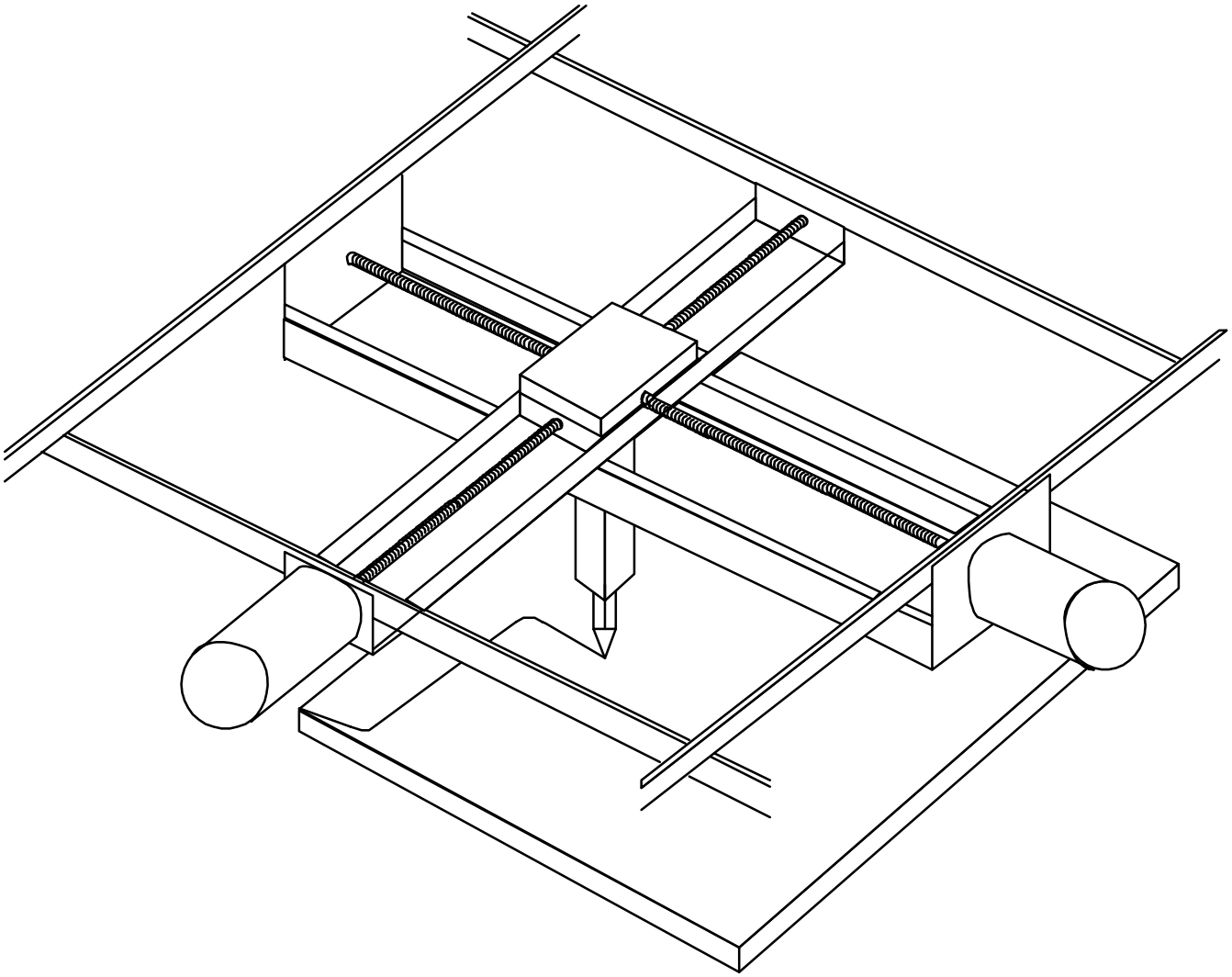
```
‘****** debounce Register Input
debounce_E1_2:
        cnt = DCNT                      ‘debounce delay in msec
        DO
                IF EVENT1(AX2) = state THEN
                        RETURN          ‘return if same state
                ELSE
                        cnt = cnt – 1
                        wait=.001       ‘wait 1 msec
                END IF
        LOOP UNTIL cnt < 1              ‘wait for debounce switch state change
        IF state=1 THEN
                E1_2 = 0                ‘change state
        ELSE
                E1_2 = 1                ‘change state
        END IF
RETURN                                  ‘return with different state

‘****** debounce Print Input
debounce_E2_1:
        cnt = DCNT                      ‘debounce delay in msec
        DO
                IF EVENT2(AX1) = state THEN
                        RETURN          ‘return if same state
                ELSE
                        cnt = cnt – 1
                        wait=.001       ‘wait 1 msec
                END IF
        LOOP UNTIL cnt < 1              ‘wait for debounce switch state change
        IF state=1 THEN
                E2_1 = 0                ‘change state
        ELSE
                E2_1 = 1                ‘change state
        END IF
RETURN                                  ‘return with different state

‘****** debounce Execute Input
debounce_E2_2:
        cnt = DCNT                      ‘debounce delay in msec
        DO
                IF EVENT2(AX2) = state THEN
                        RETURN          ‘return if same state
                ELSE
                        cnt = cnt – 1
                        wait=.001       ‘wait 1 msec
                END IF
        LOOP UNTIL cnt < 1              ‘wait for debounce switch state change
        IF state=1 THEN
                E2_2 = 0                ‘change state
        ELSE
                E2_2 = 1                ‘change state
        END IF
RETURN                                  ‘return with different state
```

```
teach:
        JOYSTICK(AX1,AX2)                       ‘ enable 2 axis joystick
        NVR(1)=0                                ‘ default starting position
        NVR(2)=0                                ‘ default starting position
        NVR(3)=4                                ‘ default element

        PRINT#1,”Move to pattern starting position”
        PRINT#1,”     and press Register button”
        PRINT#1,”               or”
        PRINT#1,”   open Teach Switch to exit”
        DO
                DO
                        state=0
                        GOSUB debounce_E1_2     ‘test Register button
                        state=1
                        GOSUB debounce_E1_1     ‘test Teach input
                        IF E1_1=0 THEN
                                STOP(AX1,AX2)   ‘disable 2 axis joystick
                                RETURN          ‘teach complete
                        END IF
                LOOP UNTIL E1_2=1               ‘wait for register switch closing

                NVR(1)=ABSPOS(AX1)              ‘register starting position
                NVR(2)=ABSPOS(AX2)              ‘register starting position
                PRINT#1,”Start Position”,NVR(1),NVR(2)
                DO                              ‘state=1 from above
                        GOSUB debounce_E1_2     ‘test Register button
                LOOP UNTIL E1_2=0               ‘wait for Register switch opening

                X=4                             ‘starting element
                Y=1                             ‘point number
                DO
                        state=0
                        GOSUB debounce_E1_2     ‘test Register button
                        state=1
                        GOSUB debounce_E1_1     ‘test Teach input
                        IF E1_1=0 THEN
                                STOP(AX1,AX2)   ‘disable 2 axis joystick
                                NVR(3)= X – 1   ‘save  last element number
                                RETURN          ‘teach complete
                        END IF
                LOOP UNTIL E1_2=1               ‘wait for Register switch closing

                NVR(X) = ABSPOS(AX1)            ‘register position
                NVR(X+1) = ABSPOS(AX2)          ‘register position
                PRINT#1,”Point ”;Y,NVR(X),NVR(X+1)
                X = X + 2                       ‘next element
                Y = Y + 1                       ‘next point
                DO                              ‘state=1 from above
                        GOSUB debounce_E1_2     ‘test Register button
                LOOP UNTIL E1_2=0               ‘wait for Register switch opening
        LOOP UNTIL 1=2                          ‘loop indefinitely
```

## 13.2 - Arbitrary Continuous Motion

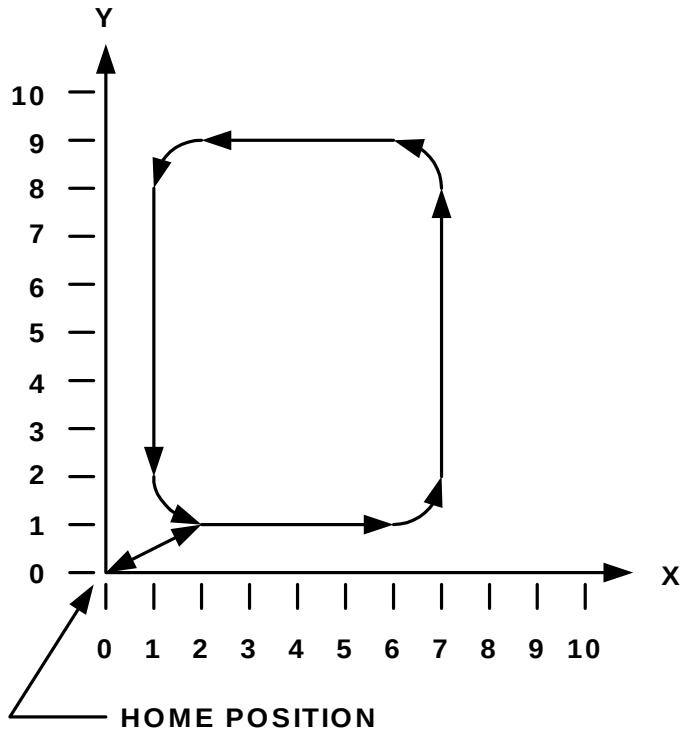
This program illustrates the simplicity of using an arbitrary continuous motion path in any application.

In this application, the operator places a sheet of sponge material on a sponge cutting machine and then activates a cycle start switch (IN101). The cutting blade moves to a starting position, lowers, and then cuts a predetermined shape sponge. After a sponge is cut, the blade is raised and returned to a home position.



## 13.2.1 – EXAMPLE PROGRAM

```
POSMODE =1,1                    'enable absolute mode
DO : LOOP UNTIL IN(101)=1       'loop until input 101 is high
MOVE=2,1                        'move to starting position
OUT(111)=1                      'turn output 111 on (high)
PATH=1,2                        'begin continuous motion path
        LINE=6,1                'first coordinate of the path
        POINT=7,2               'second coordinate of the path
        LINE=7,8
        POINT=6,9
        LINE=2,9
        POINT=1,8
        LINE=1,2
        POINT=2,1
PATH END                        'end of continuous motion path
OUT(111)=0                      'turn output 111 off (low)
MOVEHOME=1,1                    'return to home position
END
```

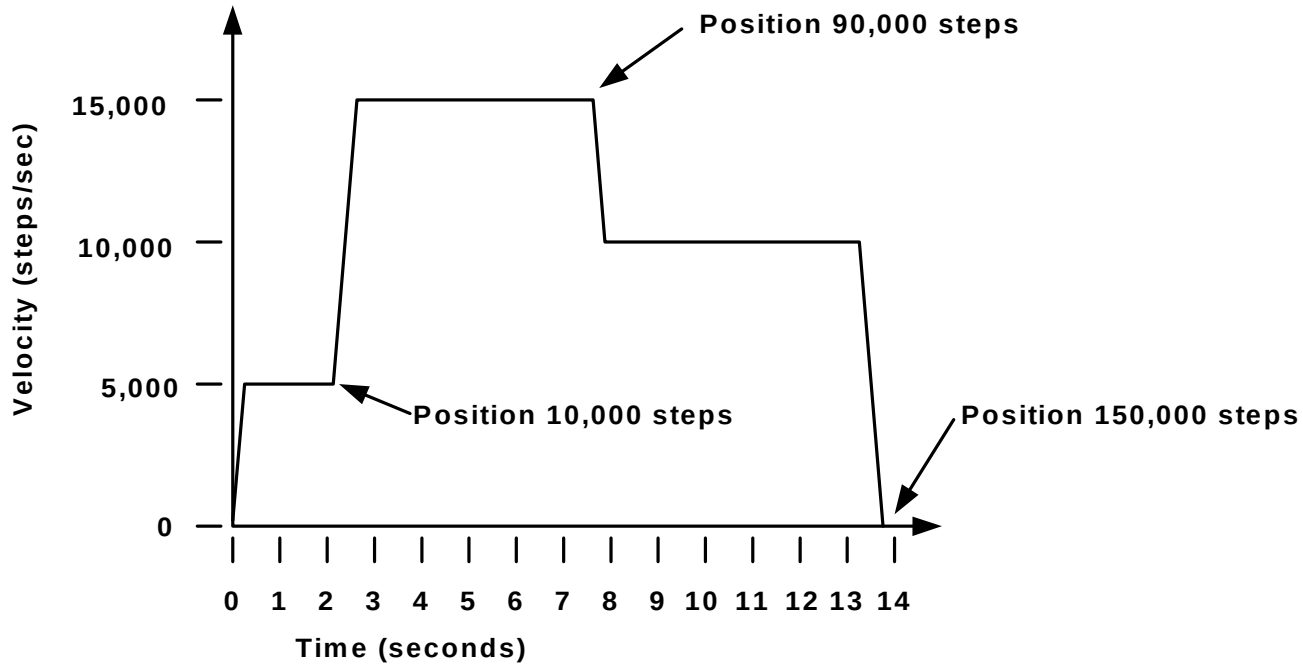


## 13.3 - Changing Velocity during motion

This program illustrates changing velocity of an axis during a path motion and the velocity change is based on position.

## 13.3.1 – Example Program

```
POSMODE(1,2)=1,1              ‘sets absolute position mode
SPEED(1)=10000                'set velocity to 10000 steps/sec
ACCEL(1,2)=20000,20000        'set acceleration to 20000 steps/sec2
MOVEHOME(1,2)=1,1             'go to home position
WAITDONE (1,2)                'wait until axes 1 and 2 are at home position
PATH=1,2
      FEEDRATE=0.5            'set velocity to 50% its value (0.5 x 10000 steps/s = 5000 steps/sec)
      LINE=10000,0            'move to 10,000 steps
      FEEDRATE=1.5            'change velocity to 1.5 x 10000 steps/s = 15000 steps/sec
      LINE=90000,0            'move to 90,000 steps
      FEEDRATE=1              'set velocity back to 100% (10000 steps/sec)
      LINE=150000,0           'move to 150,000 steps
PATH END                      'stop motion
END
```

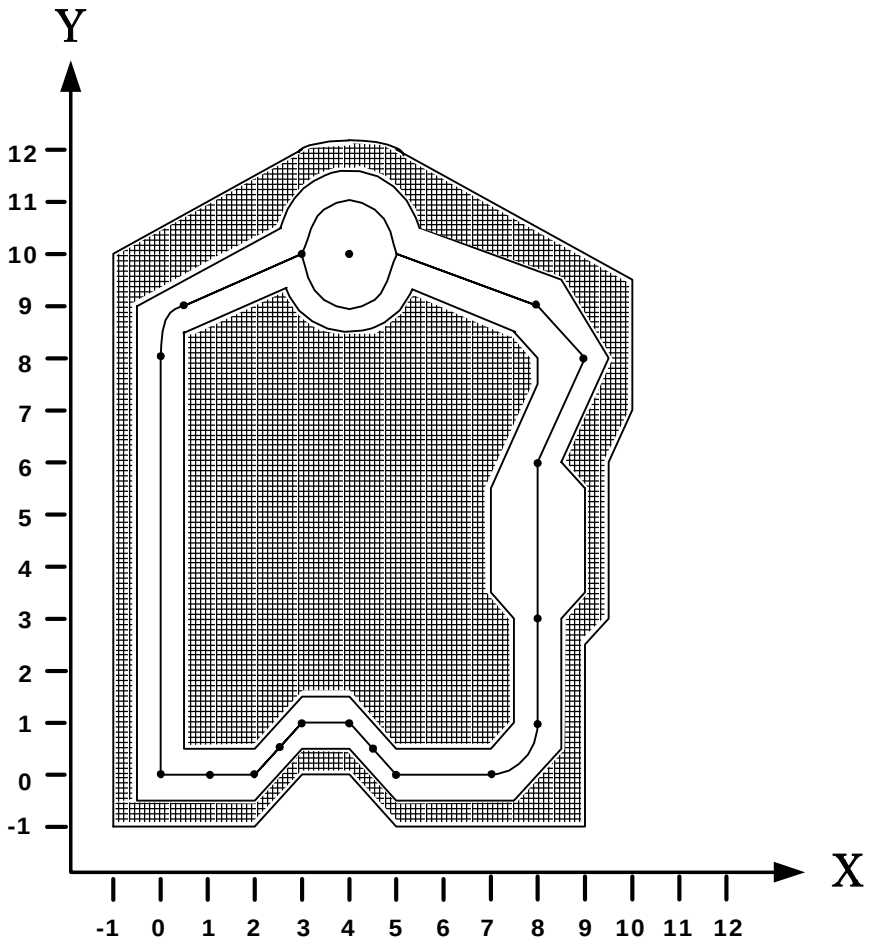


## 13.4 -Glue application on a Gasket

This program generates a complex continuous motion path for applying glue on a gasket.

An operator activates a cycle start switch (EXIN111), the glue head returns to a home position, and an absolute position is set to zero. The glue head is moved 5" from home to a starting position, and absolute position is set to zero again (all path coordinates are relative to this point). Next the glue head is lowered (EXOUT102) and waits for .25 seconds. Then glue is applied along the pattern, which is described by the x-y coordinates of the lines, arcs, and paths in the Gluing Subroutine section of the program. Finally, the glue is turned off and the glue head is raised (EXOUT101 and 102).

**Gasket Pattern:**




## 13.4.1 –Example Program

```
‘******** Parameter Setup ************************************
RADIUS=0                    'radius for path blending
VELOCITY=5                  'path speed = 5in/sec
ACCEL=10,10                 'acceleration rate = 10in/sec2
DECEL=10,10                 ‘deceleration rate = 10in/sec2
SOFTLIMIT=0,0               'disable software limits
HARDLIMIT=1,1               'enable hard limits
POSMODE=1,1                 ‘enable absolute mode
```

```
‘********* Main Program ****************************************
BEGIN:
DO
        DO : LOOP WHILE EXIN(111)=0      'wait for cycle start input
        GOSUB HOME                       'go to home position
        LINE= 5,5                        'offset each axis to starting position (5" from home)
        WAITDONE=1,1                     'wait until axes 1 and 2 are in position
        ABSPOS = 0,0                     'reset absolute position to zero
        GOSUB GLUE_PATH                  'go to subroutine GLUE_PATH
LOOP UNTIL 1=2                           ‘loop indefinitely


‘******** Gluing Routine ****************************************
GLUE_PATH:
        EXOUT(101)=1                     'head down
        EXOUT(102)=1                     'glue on
        WAIT=.25                         'wait for glue to start flowing
        PATH = 1,2                       ‘beginning of path
                LINE = 0,8
                POINT = 0.5,9
                LINE =3,10
                ARC = 4,10,+540
                LINE = 8,9
                POINT = 9,8
                POINT = 8,6
                LINE =8,6
                FEEDRATE = 0.5           'decrease velocity to 50% = 2.5in/sec
                LINE = 8,3
                FEEDRATE = 1             'increase velocity back to 100% = 5in/sec
                LINE =8,1
                POINT = 7,0
                LINE =5,0
                POINT = 4.5,.5
                POINT = 4,1
                LINE = 3,1
                POINT =2.5,.5
                POINT = 2,0
                LINE = 0,0
        PATH END                         'end of path
        EXOUT(101,2)=0                   'turn glue off and raise glue head
RETURN                                   'end of subroutine

'****** Home routine ************************************************
HOME:
        EXOUT(123,2)=0                   'glue off and head up
        SPEED =2,2                       'home speed = 2in/sec
        MOVEHOME =-1,-1                  'move to home switch x & y "-" dir
        WAITDONE(1,2)
        ABSPOS = 0,0                     'set absolute position to 0
RETURN                                   'end of subroutine
```

## 13.5 - Spring Winding Machine

In this application two motors must be moved simultaneously to wind a spring. An expansion I/O board is used to provide the required inputs to the controller.

The sequence of events for this application is as follows:

1) A cam will actuate a switch (EXIN(101)) to start the machine cycle

2) The wire will be fed (EXOUT(112))

3) Delay 0.1 seconds to feed enough wire out before a clamp (EXOUT(111)), used to hold the wire in place, is turned on.

4) Next a center form clamp (EXIN(102)), activated by a cam, is moved into position, the winding pin (EXOUT(113)) slides in and the wire is cut (EXOUT(114)).

5) The wire is stopped from being fed (EXOUT(112)) then the wire clamp and the cutter is lifted up (EXOUT(111) and EXOUT(114)).

6) The cam actuated U-bender (EXIN(103)) bends the wire into a U shape and the spring is wound.

7) Once the spring has been wound, wire sensing probes move in (EXOUT(115) & EXOUT(116)) and check if it has been wound enough (EXIN(105) and EXIN(106)). If not, the spring is wound one step and checked again. This procedure is continued for a predefined number of steps.

8) Recoil to release the spring from the arbors, retract the wire sensing probes (EXOUT(115) & EXOUT(116)), and slide the winding pin out (EXOUT(113)) to drop the spring in a bin.

9) Move back to absolute zero.

10) Check whether the auxiliary feed has been depleted, if so end the cycle, otherwise go back to the beginning of program and make another spring.

## 13.5.1 – Example Program

```
‘********************* PARAMETER SETUP ****************************
WIND=145                            'number of steps to wind wire
AUX=20                              '# of steps for auxiliary wind
RECOIL=50                           '# of steps to recoil
BOOST=1,1                           'enable boost current function
ABSPOS=0,0                          'set absolute position to zero

‘******************** START OF MAIN PROGRAM **********************
BEGIN:
    DO : LOOP UNTIL EXIN(101)=1     'wait for switch to be activated by cam
    WAIT=.1                         'wait .1 sec
    EXOUT(111)= 1                   'turn clamp on(expansion output 111) to hold wire
    WAIT=.1                         'wait .1 sec
    DO : LOOP UNTIL EXIN(102)=1     'wait until center form clamp is in position
    EXOUT(113,2)=3                  'winding pin in & cutter down
    EXOUT(111,2)=0                  'turn output 112 (feed) and output 111 (clamp) off
    EXOUT(114)=0                    'turn output 114(cutter) off
    DO : LOOP UNTIL EXIN(103)=1     'wait until U-bender bends spring
    MOVE=WIND,WIND                  'wind spring "wind" # of steps
    EXOUT(115,2)=3                  'turn on probe x (out 115) & probe y (out 116)

    FOR X=1 TO AUX                  'go through loop A number of times
        IF EXIN(105)=0 THEN 'if input 105 is off
                MOVE=1              'move X-axis 1 step
                A=X                 'A = number of auxiliary feed steps
        END IF
        IF EXIN(106)=0 THEN 'if input 106 is off
                MOVE=,1             'move y-axis 1 step
                A=X                 'A = number of auxiliary feed steps
        END IF
        WAIT=.1                     'wait .1 sec
    NEXT X

    MOVE= -RECOIL,-RECOIL           'move "recoil" # of steps
    EXOUT(115,2)=0                  'turn output 115 (probe x) & output 116 (probe y) off
    EXOUT(113)=0                    'turn out 113( winding pin) off
    POSMODE=1,1                     'enable absolute mode
    MOVE=0,0                        'move to absolute zero
    WAITDONE=1,1                    'wait until motion stops on axes 1 and 2
    POSMODE=0,0                     'switch back to incremental mode

    IF A=AUX THEN                   'if auxiliary feed equals aux then part is bad
        END                         'end program
    ELSE
        GOTO BEGIN                  'if go back to beginning and wind another spring
    END IF
END
```

# SECTION 14
# TROUBLESHOOTING GUIDE

***High voltages are present inside the unit. Always disconnect the power before performing any work on the unit. An electrical shock hazard exists that may cause serious injury or death if this unit is operated without its protective covers in place.***

## 14.1 – Status Indicator Lights

The status indicator lights (red LED's) on the front panel of the Controller provide an invaluable troubleshooting aid.

### 14.1.1 - Power Led

The POWER indicator light is located on the Power Supply Card on the Controller. When lit, it signifies the unit's power supply is energized. Should this light fail to come on, follow this procedure:

1) Check if the AC input power is applied; if not, apply power to the unit.
2) Check if the AC input power is within the operational range. Refer to page 12-6 for power supply specifications.
3) Check for an open fuse. Refer to the Power Supply specifications on page 12-6 for fuse ratings. If the fuse(s) are O.K., or if they fail after being replaced, an internal failure has occurred → contact Superior Electric. Do not apply power again.

### 14.1.2 - Fault Led

The "FAULT" indicator light is located on the DSP controller card. When lit, it signifies a programming error, a processor error, or a motion error has occurred.

### 14.1.3 - Busy Led

The "BUSY" indicator lights are located on the dual axis card. When lit, they signify the control has received or executed a motion command.

## 14.2 - Serial Communications

If you are unable to establish serial communications between a host computer and the Controller:

1) Make sure that all hardware connections have been made properly, cable lengths do not exceed specified limits, and that power cables are isolated from the communications cables.
2) Make sure that a user program is not being executed while trying to establish communications. If a program is running, pressing Ctrl-A from the terminal mode can stop it.
3) Make sure that the controller's baud rate matches that of the host computer and the correct communications protocol (RS232 or RS485) is selected. Also, ensure that the correct com port is chosen on the host computer.

The Controller baud rate and communications protocol is selected with the **BAUD** switch located on the front panel of the DSP card. If this switch setting has to be changed you must cycle power to the controller, since these switches are read only at power up.

The host computer's baud rate can be selected using the System menu items Terminal Setting- Com Port. The serial communications format is 8 data bits, no parity, and 1 stop bit ("8-N-1").

Verify correct Com Port selection and pin out. Place a jumper between pins 2 & 3. Press a key on the keyboard in terminal mode. The letter pressed is the letter that will appear on the terminal screen.

## 14.3 - If You Can Not Access Axis I/O

Make sure that the polarity jumper on the Axis card is in the appropriate setting, sink or source, depending on your particular application. (Refer to section 5.11)

**If a problem persists, contact**
**Motion Control Applications Engineering Department at 1-800-SUPELEC (1-800-787-3532), between the hours of 8:00 am and 5:00 pm EST.**

## 14.4 – No Motion Occurring

If motion is commanded, the busy LED will illuminate for the specified axis during motion. No motion occurring indicates that the CLR to COM jumper is not in place, drive is not ready, windings are not enabled or a servo drive has not been tuned.

# SECTION 15
# GLOSSARY

**ABSOLUTE MODE** - Motion mode in which all motor movements are specified in reference to an electrical home position.

**ABSOLUTE POSITION** - A data register in the Controller which keeps track of the commanded motor position. When the value in this register is zero, the position is designated "Electrical Home".

**ACCELERATION** - The rate at which the motor speed is increased from its present speed to a higher speed (specified in units/second/second).

**ACCURACY** (of step motor) - The non-cumulative incremental error which represents step to step error in one full motor revolution.

**ALL WINDINGS OFF** - Applying an average zero motor current at standstill to alleviate motor heating or eliminate holding torque.

**AMBIENT TEMPERATURE** - The temperature of the air surrounding the motor or drive.

**ASCII** - (American Standard Code for Information Interchange). A format to represent alphanumeric and control characters as seven-or eight-bit codes for data communications.

**ATTENTION CHARACTER** - <nn, where "nn" is a unique integer from 1-99 (set by use of the unit ID# select switches) that is assigned to a Motion Controller arrayed in a multi-Controller system. The Attention Character directs the program command to the specified Motion Controller.

**BASE SPEED** - Starting speed for the motor (also known as low speed).

**BAUD RATE** - The rate of serial data communications expressed in binary bits per second.

**BCD** - (Binary Coded Decimal), a format to represent the digits 0 through 9 as four digital signals. Systems using thumb wheel switches may program commands using BCD digits. A BCD digit uses a standard format to represent the digits 0 through 9 as four digital signals.

The following table lists the BCD and complementary BCD representation for those digits. The Motion Controller uses the complementary BCD codes because the signals are active low.

| Digit | BCD Code | Complementary BCD Code |
|---|---|---|
| 0 | 0000 | 1111 |
| 1 | 0001 | 1110 |
| 2 | 0010 | 1101 |
| 3 | 0011 | 1100 |
| 4 | 0100 | 1011 |
| 5 | 0101 | 1010 |
| 6 | 0110 | 1001 |
| 7 | 0111 | 1000 |
| 8 | 1000 | 0111 |
| 9 | 1001 | 0110 |

To represent numbers greater than 9, cascade the BCD states for each digit. For example, the decimal number 79 is BCD 0111:1001.

**BOOST CURRENT** - Increase of motor current during acceleration and deceleration to provide higher torque, which permits faster acceleration/deceleration times.

**CLEAR** - Input or Command to immediately halt all motor motion and program execution.

**COLLECTORS (OPEN)** - A transistor output that takes the signal to a low voltage level with no pull-up device; resistive pull-ups are added to provide the high voltage level.

**CYCLE START** - Command to initiate program execution.

**CYCLE STOP** - Command to stop program execution.

**DAISY-CHAIN**- A method to interface multiple Motion Controllers via RS485 to a single host using only one serial port.

**DAMPING** - A method of applying additional friction or load to the motor in order to alleviate resonance and ring out. Stepper motor shaft dampers are commercially available from several sources, including Superior Electric.

**DECELERATION** - The rate in which the motor speed is decreased from its present speed to a lower speed (specified in units/second/second).

**DEVICE ADDRESS** - A unique number used to assign which Motion Controller in a multi-drive stepper system is to respond to commands sent by a host computer or terminal. Device addresses from 1 - 9 are set by means of the ID # select switch. "0" is reserved to address all Motion Controllers in a system. Factory default is 1.

**DWELL** - See "WAIT".

**ELECTRICAL HOME** - The motor commanded position is zero (the Absolute Position register is zero).

**FEEDRATE** - The speed or velocity (in units per second) at which a move will occur.

**FRICTION** - Force that is opposite to the direction of motion as one body moves over another.

**FULL-STEP** - Position resolution in which 200 pulses corresponds to one motor revolution in a 200 step per revolution (1.8 degree) motor.

**HALF-STEP** - Position resolution in which 400 pulses corresponds to one motor revolution for a 200 step per revolution (1.8 degree) motor.

**HANDSHAKE** - A computer communications technique in which one computer's program links up with another's. The Motion Controller uses a software "Xon, Xoff" handshake method. See "XON" below.

**HOST** - The computer or terminal that is connected to the HOST serial port on the motion controller, and is responsible for primary programming and operation of the controller.

**INCREMENTAL MODE** - Motion mode in which all motor movements are specified in reference to the present motor position.

**INDEXER** - A Microprocessor-based programmable motion controller that controls move distance and speeds; possesses intelligent interfacing and input/output capabilities.

**INDEX FROM RUN** – See Mark Registration.

**INERTIA** - Measurement of a property of matter that a body resists a change in speed (must be overcome during acceleration).

**INERTIAL LOAD** - A "flywheel" type load affixed to the shaft of a step motor. All rotary loads (such as gears or pulleys) have inertia. Sometimes used as a damper to eliminate resonance.

**INSTABILITY** - Also frequently called, "mid-range instability" or "mid-range resonance," this term refers to a resonance that occurs in the 500 - 1,500 steps/sec range. Mid-range instability is important because it refers to a loss of torque or a stalled motor condition at higher stepping rates. Since step motors do not start instantaneously above the mid-range resonance frequency, an acceleration scheme will have to be used to pass through the troublesome region.

**JOG MOVE** - moves the motor continuously in a specified direction.

**LOAD** - This term is used several ways in this and other manuals.

**LOAD (ELECTRICAL)**: The current in Amperes passing through a motor's windings.

**LOAD (MECHANICAL)**: The mass to which motor torque is being applied (the load being moved by the system).

**LOAD (PROGRAMMING)**: Transmits a program from one commuter to another. "DOWNLOAD" refers to transmitting a program from a host computer (where a program has been written) to the Motion Controller where it will be used. "UPLOAD" refers to transmitting a program from a Motion Controller back to the host computer.

**MARK REGISTRATION** - A motion process (usually used in web handling applications) whereby a mark placed on the material is sensed (e.g., through the use of an optical sensor) and, following detection of this mark, the material is moved (indexed) a fixed length.

**MECHANICAL HOME** - The position where a switch input is used as a reference to establish electrical home.

**MICROSTEPPING** - A sophisticated form of motor control that allows for finer resolution than full step (200 Pulses Per Revolution PPR) or half step (400 PPR) by adjusting the amount of current being applied to the motor windings. Microstepping up to 250 pulses per full step (50,000 PPR on a 200 step/rev or 1.8 degree motor) is supported. For 200 step per revolution motors, typical microstepping levels are 1/10-step and 1/125 step (2000 PPR and 25,000 PPR, respectively). Note: this is a DRIVE function.

**MOVE TO MECHANICAL HOME** - Function which allows the Motion Controller to move the motor and seek a switch to establish electrical home and set Absolute Position = zero.

**NESTING** - The ability of an active subroutine to call another subroutine. The Motion Controller can nest up to 16 levels.

**NONVOLATILE MEMORY** - Data storage device that retains its contents even if power is removed. Examples are EEPROM, flash memory, and battery-backed RAM.

**OPTO-ISOLATION** - The electrical separation of the logic section from the input/output section to achieve signal separation and to limit electrical noise. The two systems are coupled together via a transmission of light energy from a sender (LED) to a receiver (phototransistor).

**PARITY** -- An error checking scheme used in serial communications (via the RS-232 or RS-485 port) to ensure that the data is received by a Motion Controller is the same as the data sent by a host computer or terminal.

**REDUCE CURRENT** - Reduction of motor current during standstill to alleviate motor heating.

**RESOLUTION** - The minimum position command that can be executed. Specified in steps per revolution or some equivalent.

**RINGOUT** - The transient oscillatory response (prior to settling down) of a step motor about its final position. Note: a small wait or dwell time between moves can alleviate ringout problems.

**RS232-C** - EIA (Electronic Industries Association) communication standard to interface devices employing serial data interchanges. Single-wire connections for transmit and receive, etc.

**RS-485** - EIA (Electronic Industries Association) communication standard to interface devices employing serial data interchanges. Two-wire connections (differential circuits) for transmit and receive, etc. Better than RS-232 for long wire runs and multi-drop circuits with many devices.

**SINKING** - An input that responds to, or output that produces, a "low" level (signal common or low side of the input/output power supply) when active.

**SOURCING** - An input that responds to, or output that produces, a "high" level (the voltage used for the input/output power supply) when active.

**SUBROUTINE** - A sequence of lines that may be accessed from anywhere in a program to preclude having to program those lines repetitively. This allows shorter, more powerful, and more efficient programs. See also NESTING.

**TORQUE** - Product of the magnitude of a force and its force arm (radius) to produce rotational movement. Units of measure are pound-inches, ounce-inches, newton-meters, etc.

**TRANSLATOR** - A motion control device (also called "translator drive") that converts input pulses to motor phase currents to produce motion.

**WAIT** - A programmed delay or dwell in program execution (specified in seconds).

**XON / XOFF** - A computer software "handshaking" scheme used by a Motion Controller. The Motion Controller sends an XOFF character (ASCII Code 19) when it receives a command string with a Carriage Return and has less than 82 characters remaining in its host serial port buffer. The Controller sends an Xon when available buffer space reaches 100 characters or in response to an ID attention with adequate buffer space remaining. Since it is impossible for the host device to immediately cease transmissions, the next three characters (subject to the total serial buffer capacity of forty characters) received subsequent to the Motion Controller sending the XOFF character will be stored in the Motion Controller's serial buffer (a memory dedicated to store characters that are in the process of transmission).

Similarly, the Motion Controller will not transmit data if the host device has sent an XOFF character to the Controller; Motion Controller transmissions will resume when the Controller receives an XON character.

# ASCII Table

| ASCII Char | Dec Code | ASCII Char | Dec Code | ASCII Char | Dec Code | ASCII Char | Dec Code |
|---|---|---|---|---|---|---|---|
| Null | 0 | Space | 32 | @ | 64 | ` | 96 |
| SOH | 1 | ! | 33 | A | 65 | a | 97 |
| STX | 2 | A | 34 | B | 66 | b | 98 |
| ETX | 3 | # | 35 | C | 67 | c | 99 |
| EOT | 4 | $ | 36 | D | 68 | d | 100 |
| ENQ | 5 | % | 37 | E | 69 | e | 101 |
| ACK | 6 | & | 38 | F | 70 | f | 102 |
| BELL | 7 | › | 39 | G | 71 | g | 103 |
| BS | 8 | ( | 40 | H | 72 | h | 104 |
| HT | 9 | ) | 41 | I | 73 | i | 105 |
| LF | 10 | * | 42 | J | 74 | j | 106 |
| VT | 11 | + | 43 | K | 75 | k | 107 |
| FF | 12 | , | 44 | L | 76 | l | 108 |
| CR | 13 | - | 45 | M | 77 | m | 109 |
| SO | 14 | . | 46 | N | 78 | n | 110 |
| SI | 15 | / | 47 | O | 79 | o | 111 |
| DLE | 16 | 0 | 48 | P | 80 | p | 112 |
| DC1 | 17 | 1 | 49 | Q | 81 | q | 113 |
| DC2 | 18 | 2 | 50 | R | 82 | r | 114 |
| DC3 | 19 | 3 | 51 | S | 83 | s | 115 |
| DC4 | 20 | 4 | 52 | T | 84 | t | 116 |
| NAK | 21 | 5 | 53 | U | 85 | u | 117 |
| SYNC | 22 | 6 | 54 | V | 86 | v | 118 |
| ETB | 23 | 7 | 55 | W | 87 | w | 119 |
| CAN | 24 | 8 | 56 | X | 88 | x | 120 |
| EM | 25 | 9 | 57 | Y | 89 | y | 121 |
| SUB | 26 | : | 58 | Z | 90 | z | 122 |
| ESC | 27 | ; | 59 | [ | 91 | { | 123 |
| FS | 28 | < | 60 | \ | 92 | \| | 124 |
| GS | 29 | = | 61 | ] | 93 | } | 125 |
| RS | 30 | > | 62 | ^ | 94 | ~ | 126 |
| DEL | 31 | ? | 63 | _ | 95 | DEL | 127 |

This page left intentionally blank

## WARRANTY AND LIMITATION OF LIABILITY

Superior Electric (the "Company"), Bristol, Connecticut, warrants to the first end user purchaser (the "purchaser") of equipment manufactured by the Company that such equipment, if new, unused and in original unopened cartons at the time of purchase, will be free from defects in material and workmanship under normal use and service for a period of one year from date of shipment from the Company's factory or a warehouse of the Company in the event that the equipment is purchased from the Company or for a period of one year from the date of shipment from the business establishment of an authorized distributor of the Company in the event that the equipment is purchased from an authorized distributor.

**THE COMPANY'S OBLIGATION UNDER THIS WARRANTY SHALL BE STRICTLY AND EXCLUSIVELY LIMITED TO REPAIRING OR REPLACING, AT THE FACTORY OR A SERVICE CENTER OF THE COMPANY, ANY SUCH EQUIPMENT OF PARTS THEREOF WHICH AN AUTHORIZED REPRESENTATIVE OF THE COMPANY FINDS TO BE DEFECTIVE IN MATERIAL OR WORKMANSHIP UNDER NORMAL USE AND SERVICE WITHIN SUCH PERIOD OF ONE YEAR. THE COMPANY RESERVES THE RIGHT TO SATISFY SUCH OBLIGATION IN FULL BE REFUNDING THE FULL PURCHASE PRICE OF ANY SUCH DEFECTIVE EQUIPMENT.** This warranty does not apply to any equipment which has been tampered with or altered in any way, which has been improperly installed or which has been subject to misuse, neglect or accident.

**THE FOREGOING WARRANTY IS IN LIEU OF ANY OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE,** and of any other obligations or liabilities on the part of the Company; and no person is authorized to assume for the Company any other liability with respect to equipment manufactured by the Company. The Company shall have no liability with respect to equipment not of its manufacture. **THE COMPANY SHALL HAVE NO LIABILITY WHATSOEVER IN ANY EVENT FOR PAYMENT OF ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR INJURY TO ANY PERSON OR PROPERTY.**

Written authorization to return any equipment or parts thereof must be obtained from the Company. The Company shall not be responsible for any transportation charges.

**IF FOR ANY REASON ANY OF THE FOREGOING PROVISIONS SHALL BE INEFFECTIVE, THE COMPANY'S LIABILITY FOR DAMAGES ARISING OUT OF ITS MANUFACTURE OR SALE OF EQUIPMENT, OR USE THEREOF, WHETHER SUCH LIABILITY IS BASED ON WARRANTY, CONTRACT, NEGLIGENCE, STRICT LIABILITY IN TORT OR OTHERWISE, SHALL NOT IN ANY EVENT EXCEED THE FULL PURCHASE PRICE OF SUCH EQUIPMENT.**

Any action against the Company based upon any liability or obligation arising hereunder or under any law applicable to the sale of equipment, or the use thereof, must be commenced within one year after the cause of such action arises.

---

## Distribution Coast-To-Coast and International

Superior SLO-SYN products are available worldwide through an extensive authorized distributor network. These distributors offer literature, technical assistance and a wide range of models off the shelf for fastest possible delivery and service.
In addition, Superior Electric sales engineers are conveniently located to provide prompt attention to customers' needs. Call the nearest office listed for ordering and application information or for the address of the closest authorized distributor.

**In U.S.A. and Canada**
383 Middle Street
Bristol, CT 06010
Tel: 860-585-4500
Fax: 860-589-2136
Customer Service: 1-800-787-3532
Product Application: 1-800-787-3532
Product Literature Request: 1-800-787-3532
Fax: 1-800-766-6366
Web Site: www.superiorelectric.com

Superior Electric 383 MIDDLE STREET BRISTOL, CT 06010 (860) 585-4500 FAX (860) 589-2136

ISO 9001 SGS