User Guide Change Summary

The following is a summary of the primary changes to this user guide since the last version was released. This user guide, version 88-010985-01**C**, supersedes version 88-010985-01**B**.

When a user guide is updated, the new or changed text is differentiated with a change bar in the right margin (this paragraph is an example). If an entire chapter is changed, the change bar is located to the right of the chapter title.

Chapter 5, *Hardware Reference*, was removed and its information (system specs, I/O specs & connections, etc.) were placed in relevant areas in Chapters 2 and 3.

All references to the discontinued Photo-Trak absolute encoder have been removed.

Technical changes to each chapter are synopsized below.

Chapter 1. Introduction

- *Product Description* (pg. 1): Included discussion of the 500's compatibility with the Compumotor RP240 remote front panel

Chapter 2. Getting Started

- *Motor Driver Connections for non-Compumotor drives* (pg. 10): Changed description of **MOTOR DRIVER** connector pin #4 from In-position to Reserved (Table 2-3)

Chapter 3. Installation

- *End-of-Travel Limits* (pg. 18): Described option of changing active state of the CW and CCW limit inputs with the **OSA** command (**note that this may diminish the safety of the operating environment**)
- *Daisy-Chain Connections* (pg. 20): Edited Figure 3-8—reversed the address configuration of the daisy-chained units from 1, 2, 3 to 3, 2, 1

Chapter 4. Application Design

- *Registration* (pg. 37): Clarified capture of motor position and registration input, and when the registration profile will be executed
- *Go Home* (pg. 41): Included explanation of using the **OSJ** homing set-up command to (a) abort the homing operation of the Z channel pulse if not received within one rev of activating the home input (default), or (b) seek the Z channel pulse indefinitely
- *Interfacing to the 500* (pg. 71): Added the RP240 remote front panel as an operator interface option
- *Using the TM8 Module* (pg. 78): Edited Figures 4-13 and 4-14 to show that the user must provide an external +5VDC supply to power the module(s)
- *Remote Panel Operation* (pg. 88): New section added to introduce the RP240 remote panel as an operator interface option
- *Multi-Axis Control Daisy-chaining* (pg. 90): Edited Figure 4-20—reversed the address configuration of the daisy-chained units from 1, 2, 3 to 3, 2, 1

Chapter 5. Following

No Changes

Chapter 6. Maintenance & Troubleshooting

No Changes

Appendices

- *Quick-Reference Command List* (pg. 159): Updated to include the following commands added to the 500/500-FOL (see <u>**500/500-FOL Indexer Software Reference Guide**</u> for detailed descriptions):
  OSA — Set Active Level for End-of-Travel Limits
  OSF — Acknowledge Programmable Inputs on Power Up
  OSJ — Select Z Channel Homing to Single or Unlimited Revs
  TD — Set Inputs Debounce Time (applies to all inputs, including end-of-travel limits—use with caution)

Table of Contents

List Of Figures

List Of Figures (continued)

List Of Tables

How To Use This User Guide

You do not have to be a motion control expert to operate the Model 500 Indexer. This user guide will prepare you to use the system. For complex applications, you should consult a professional programmer to integrate the tasks that the computer must perform.

This user guide will help you install, develop, and maintain your system. Each chapter begins with specific objectives that should be met after you have read the chapter. This section will help you find and use the information in this user guide.

Assumptions

This user guide assumes that you have the skills or fundamental understanding of the following information:

❏ Basic electronics concepts (voltage, switches, current, etc.)
❏ Basic motion control concepts (torque, velocity, force, etc.)

Contents of This User Guide

This user guide contains the following information.

Chapter 1: ***Introduction***

This chapter provides a description of the product and a brief account of its specific features.

Chapter 2: ***Getting Started***

Getting Started contains a list of items you should have received with your Model 500 shipment. A basic *bench test* procedure is provided.

Chapter 3: ***Installation***

This chapter contains system mounting and wiring instructions. A system specifications table is provided at the beginning. Upon completion of this chapter, your system should be completely installed and ready to perform basic operations.

Chapter 4: ***Application Design***

Application Design provides additional information that will help you customize the system to meet your application's needs. Important application considerations are discussed. Sample applications are provided.

Chapter 5: ***Following***

This chapter helps you become familiar with the following functions of the Model 500-FOL indexer.

Chapter 6: ***Maintenance & Troubleshooting***

This chapter describes Compumotor's recommended system maintenance procedures. Included are methods for isolating and resolving hardware and software problems. Procedures for returning the Model 500 to affect repairs or upgrades are also provided .

Installation Process Overview

To ensure trouble-free operation, pay special attention to the following factors:

❏ The environment in which the Model 500 will operate
❏ The system layout and mounting
❏ The wiring and grounding practices used

Developing Your Application

Before you attempt to develop and implement your application, you should consider the following issues:

① Define the requirements of your application. Clarify system expectations.

② Assess your resources and limitations. This will help you find the most efficient and effective means of developing and implementing your application.

③ Follow the guidelines and instructions outlined in this user guide. ***Do not skip any steps or procedures.*** Proper installation and implementation can be ensured only if you complete all procedures in the proper sequence.

Installation Preparation

Before you attempt to install this product, you should complete the following steps. Successful completion of these steps should prevent subsequent performance problems and allow you to resolve any potential system difficulties before they affect your system's operation.

① Review this entire user guide. Become familiar with the user guide's contents so that you can quickly find the information you need. Also review the **Model 500 Indexer Software Reference Guide**.

② Develop a basic understanding of all system components, their functions, and interrelationships.

③ Complete the basic system configuration and wiring instructions **(in a simulated environment, not a permanent installation)** provided in Chapter 2, *Getting Started*.

④ Perform as many basic moves and functions as you can with the preliminary configuration. Try to simulate (with no load attached) the task(s) that you expect to perform when you permanently install your application.

⑤ After you have familiarized yourself with and/or tested the system's functions and features, carefully read *Chapter 3, Installation*.

⑥ Before you begin to customize your system, check all of the system functions and features to ensure that you have completed the installation process correctly.

Conventions

To help you understand and use this user guide effectively, the conventions used throughout this user guide are explained in this section.

Commands

The command examples in this user guide are presented vertically to help you read and understand them. When you actually type these commands at your computer keyboard, they will be displayed horizontally. All commands that you are instructed to enter are displayed as they should appear on your computer CRT. A one-line explanation of the command is provided next to each example. Be sure to add a delimiter (space or carriage return) after each command in a sequence. Refer to the example below.

Command	Description
`> A5`	Sets acceleration to 5 rps^2
`> V5`	Sets velocity to 5 rps
`> D1ØØØ`	Sets distance to 1,000 steps
`> G`	Executes the move (Go)

Responses are presented as the should appear on the CRT. An example is provided below.

Command	Response
`> 1XU1`	`*SSJ1_A2_V5_LD3_MN_SN1Ø`

In the interactive mode (default mode), the indexer responds with a prompt (**>**) when it receives a valid command and a question mark (**?**) when it does not receive a valid command. These interactive responses are preceded with a carriage return and a line feed. While defining loops or sequences, no responses are received until the loop or sequence is executed or until the **XT** command is executed. Refer to the **Model 500 Indexer Software Reference Guide** for detailed command descriptions.

Warnings & Cautions

Warning and caution notes alert you to possible dangers that may occur if you do not follow instructions correctly. Situations that may cause bodily injury are presented as warnings. Situations that may cause system damage are presented as cautions.

WARNING

Do not touch the motor immediately after it has been in use for an extended period of time. The unit will be hot.

CAUTION

System damage will occur if you power up the system improperly.

Voltage Levels

The Model 500 has inputs and outputs that you can turn on or off. The voltage levels and terminology associated with these inputs and outputs are discussed below.

Inputs

With the Model 500, you can configure the active input signal using the Input Level (**`INL`**) command. The Model 500 pulls the voltage level of these inputs up to +5VDC.

Command	Description
> **`INLØ`**	ON (1): 0V applied to the input (current flows) OFF (Ø): 5V - 30V applied to the input (no current flows)
> **`INL1`**	ON (1): 5V - 30V applied to the input (no current flows) OFF (Ø): 0V applied to the input (current flows)

Throughout this user guide, we will assume that the inputs are in **`INLØ`** mode (ON means that 0V are applied to the input and current is flowing). You can use 5VDC - 30VDC to control the input.

Programmable Output Bits

With the Model 500, you can configure the active output signal using the Output Level (**`OUTL`**) command.

Command	Description
> **`OUTLØ`**	ON (1): 0V Output (current flows) OFF (Ø): 5V - 30V Output (no current flows)
> **`OUTL1`**	ON (1): 5V - 30V Output (no current flows) OFF (Ø): 0V Output (current flows)

Throughout this user guide, we will assume that the outputs are in **`OUTLØ`** mode (ON means that the voltage level is 0V and the current is flowing). The outputs are open-collector outputs. You can use 5VDC to 30VDC to power outputs **OUT1 - OUT8** and sink up to 300mA of current.

Related Publications

The following publications may be helpful resources:

- ❑ Current Parker Compumotor Motion Control Catalog
- ❑ Seyer, Martin. *RS-232C Made Easy: Connecting Computers, Printers, Terminals and Modems*. Englewood Cliffs, NJ: Prentice Hall Inc., 1984
- ❑ Schram, Peter (editor). *The National Electric Code Handbook (Third Edition)*. Quincy, MA: National Fire Protection Association

Chapter 1. Introduction

Chapter Objectives

The information in this chapter will enable you to:

- ❑ Understand the product's basic functions & features
- ❑ Understand basic motion control concepts and apply them to your application

Product Description

The Model 500 is a stand-alone single-axis indexer that provides step and direction outputs directly compatible with all Compumotor stepper drives and step and direction input brushless servo drives. Compumotor's motion algorithm greatly enhances the Model 500's capability to control complex motion with Compumotor's stepper and servo drives.

The Model 500 uses Compumotor's latest X Series programming language enhancements. It has registration capability, as well as complex segmented move profiling capability.

With its 13 inputs and 8 outputs, the Model 500 offers PLC communication and thumbwheel input functionality. The Model 500 is compatible with Compumotor's TM8 Thumbwheel Module.

Compatibility with Compumotor's RP240 remote front panel gives Model 500 users an additonal operator interface option.

The Model 500 accepts both incremental and absolute encoder feedback for position maintenance. The Model 500 is compatible with Compumotor's -E Series incremental encoders and the AR-C and AL-C absolute encoders.

When ordered as the -FOL option, the Model 500 also provides position and velocity *following* capabilities.

Following Option (500-FOL)

The Model 500-FOL option can perform velocity following and distance following moves at a *following ratio*. The Model 500-FOL can follow the absolute or incremental encoder input. You can use the other encoder interface to perform position maintenance while in the *Following* mode.

Program the Model 500-FOL for following applications with its simple command language and report back/verification feature. You can enter following ratios via thumbwheels and change them on-the-fly.

Perform preset moves at a specified velocity ratio. You can perform registration moves while in the following mode. *Registration* moves can either follow at a ratio of the master velocity or be executed in the standard motion modes. The Model 500-FOL can jog the motor in the Following mode to help set up a system. You can also change the following ratio incrementally with the **UP** and **DOWN** pushbuttons on the front panel.

You can use the Model 500-FOL's special *Synchronization* mode to compensate for problems such as stretching in a web processing system.

Product Features

This section lists the primary features of the standard Model 500 and the Model 500-FOL option.

Standard 500

The Model 500 provides the following features:

❏ Four registration inputs (user-definable) are provided. These inputs will interrupt the Model 500 at the highest level of priority in the system.

❏ Calculation delay times for motion as low as 500 µs are possible.

❏ A complex motion profiling mode allows you to change velocity based on a distance without stopping, change distance or turning on outputs on-the-fly, and more.

❏ High level programming commands such as **IF/THEN/ELSE**, **WHILE**, **REPEAT/UNTIL**, **GOTO**, and **GOSUB** are available.

❏ Complex evaluations such as checking input levels, error conditions, boolean evaluations, and variable comparisons for basic program branching decisions can be made.

❏ An output can be configured to provide pulse and direction to a second axis to control a velocity and distance.

❏ Programmable logic controller (PLC) functionality and interfacing capability using the 13 programmable inputs and 8 programmable outputs.

❏ Two encoder interfaces allow you to perform position maintenance while in the Following mode.

❏ You can program separate acceleration and deceleration rates.

500-FOL Features

The Model 500-FOL provides the following additional features:

❏ Controls a speed based on a ratio of a primary axis speed

❏ Makes preset moves at a velocity ratio of a primary axis

❏ Synchronizes speed to a primary axis based on registration marks on material

❏ Changes following ratio and other functions based on the encoder position of a primary axis (*Cam Following*)

❏ Secondary axis accelerates to a ratio of the primary axis based on input from the encoder on the primary axis

Chapter 2. Getting Started

Chapter Objectives

The information in this chapter will enable you to:

- ❑ Verify that each component of your system has been delivered safely
- ❑ Properly configure and bench-test the system

What You Should Have

You should inspect your Model 500 shipment upon receipt for obvious damage to its shipping container. Report any such damage to the shipping company as soon as possible. Parker Compumotor cannot be held responsible for damage incurred in shipment. In addition to the Model 500, the items listed in Table 2-1 should be present and in good condition. *Be sure to check the serial plate to verify if you have a standard 500 or the 500-FOL.*

Part/Quantity	Part Number
Model 500 User Guide (1)	88-010985-01
Model 500 Software Reference Guide (1)	88-011150-01
Motor Driver Cable (1)	71-010432-10-A
Power Cable (1)	71-009039-01-Y1
25 Pin D-Connectors (2)	43-001989-01

Table 2-1. Ship Kit Table

Bench Test

This section leads you through step-by-step instructions to bench test your Model 500 system. ***This is a temporary configuration. The permanent installation will be performed in Chapter 3, Installation***. In this chapter, you will complete the following tasks:

① Connect Power Cable
② Establish Communications
③ Test RS-232C Interface
④ Connect Motor Driver
⑤ Test Procedure (successful completion indicates system components are operating properly)

Tools Required

To complete the procedures in this chapter, you will need a small flathead screwdriver.

Model 500 Front Panel

Figure 2-1 shows the front panel of the Model 500.

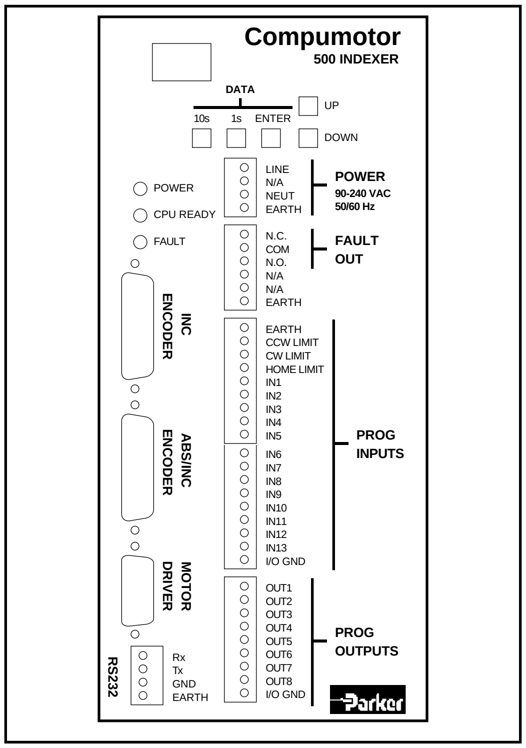


Figure 2-1. Model 500 Front Panel

① Connect Power Cable

The Model 500 is shipped with a power cable that is prewired and keyed. You can use this cable to apply power to the unit from an AC power source. If you want to wire your own power cord, use the wiring diagrams shown in Figure 2-2 for AC power (90VAC - 240VAC) or Figure 2-3 for DC power (127VDC-340VDC).

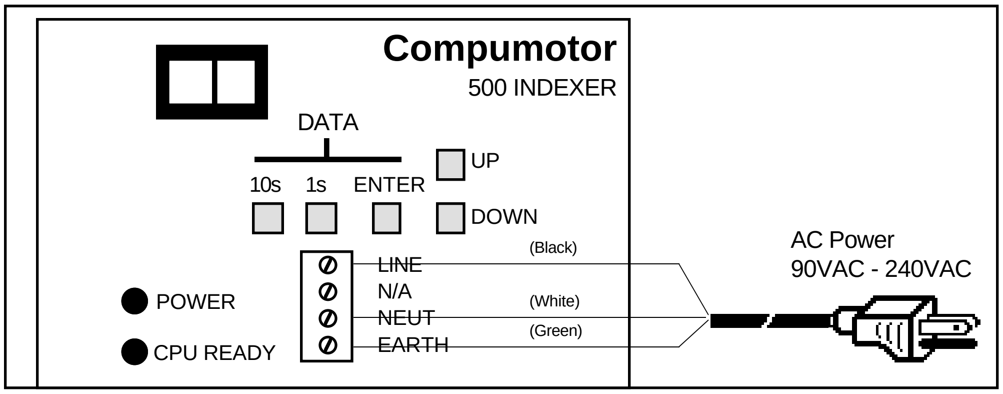


Figure 2-2. AC Power Wiring

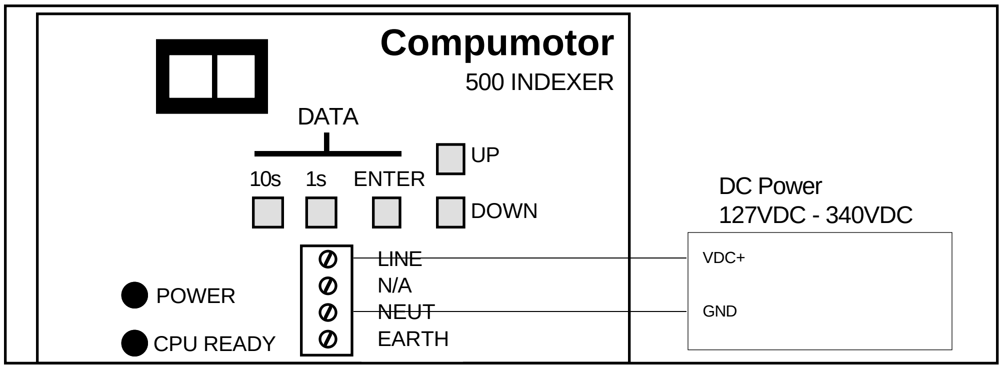


Figure 2-3. DC Power Wiring

Applying Power

After you have properly wired the cable, apply power to the Model 500 by plugging the power cord onto an AC power source or turning on a DC power source.

When you *power-up* the system, watch the LEDs on the Model 500 front panel. The LEDs are shown in Figures 2-2 and 2-3 — **POWER**, **CPU READY**, and **FAULT**. The **POWER** LED (green) and the **CPU READY** LED (green) should come on. This indicates that the Model 500 power supply and the microprocessor are functioning properly. If the **FAULT** LED stays on after power up, disconnect power and consult *Chapter 6, Maintenance & Troubleshooting*. ***After you apply power to the Model 500 successfully, disconnect power and proceed to Step 2. Establish Communications.***

② Establish Communications

To communicate with the Model 500, your computer or terminal must have an RS-232C serial port. If your computer or terminal does not have a serial port, you can purchase one from your local computer dealer.

The Model 500 uses a simple three-wire implementation of serial communication and accepts standard EIA RS-232C signals from +15 to -15VAC. The Receive Data (**Rx**), Transmit Data (**Tx**), and Ground (**GND**) signals are on the indexer's RS-232C connector (as shown in Figure 2-4). *The ground (**GND**) connection on the connector is signal ground or common as opposed to earth ground (**EARTH**).*

Disable Handshaking

The Model 500 does not support handshaking, so you should disable the computer or terminal's handshaking function. You can disable the handshaking function by connecting RTS to CTS (usually pins #4 and #5) and DSR to DTR (usually pins #6 and #20) on the computer or terminal's RS-232C port (25-pin D connector). Refer to your computer or terminal user guide for the exact instructions to disable handshaking. You can also disable handshaking by using a terminal emulation software package for the PC (see *Computer-to-Terminal Conversion* below).

Connect Terminal

Use Figure 2-4 to connect the computer or terminal to the Model 500.

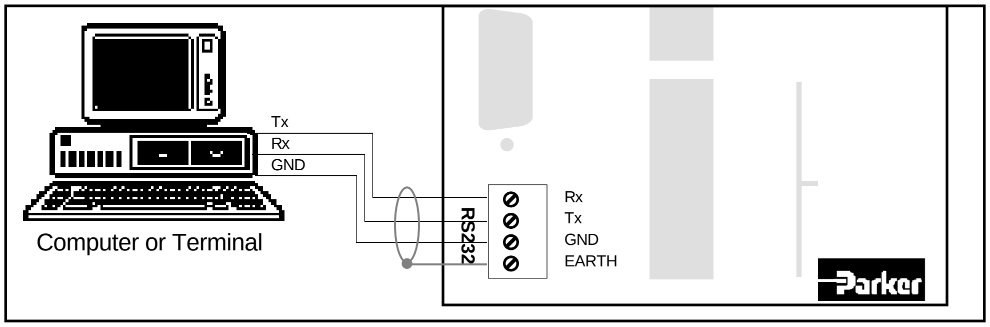


Figure 2-4. RS-232C Connections

Computer-to-Terminal Conversion

If you are using an IBM or IBM-compatible computer, you must convert the computer to a terminal to perform communications with the Model 500. Compumotor provides X-Ware, which performs this conversion for you. To order X-Ware, call Compumotor's Application Department at (800) 358-9070.

You may also use programs such as X-Talk™, PC-Talk™, and ProCom™ to convert your computer into a terminal. Sample programs are also provided in Chapter 4, *Application Design*, of this user guide. These programs can help you set-up your computer to interact with the Model 500.

You can change the baud rate parameter on the Model 500. The other communication settings are fixed. You must set the terminal's parameters to match the Model 500's settings. Refer to the terminal manufacturer's instructions on how to change the terminal's parameters. The variable baud rates and fixed parameters are listed below.

❑ Baud Rates: 9600 (default), 4800, 2400, 1200, 600, 300
❑ Data Bits: 8
❑ Parity: None
❑ Stop Bits: 1

Your terminal should be set to full-duplex mode. Consult your computer or terminal user guide for instructions on how to change the set-up to full-duplex.

Verify the Baud Rate

You can use the pushbuttons and the two-digit display to verify and change the baud rate. After you determine the baud rate that you want the system to operate at, follow the steps below to verify that the baud rate is set properly.

① Apply power to the Model 500. Both of the front panel green LEDs (**POWER** and **CPU READY**) should come on.

② To display the current baud rate setting (*the first two digits of the baud rate are shown*), push the **10s** and **1s** buttons at the same time as shown in Figure 2-5. The default baud rate for the Model 500 is 9600.

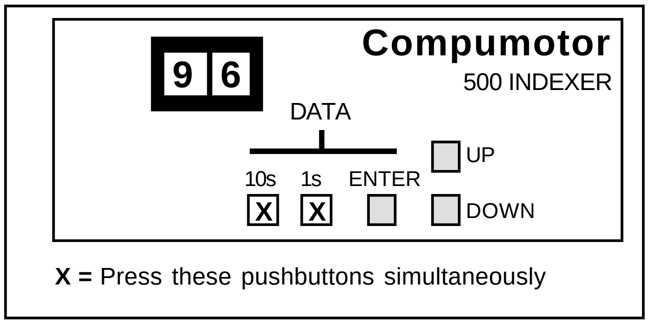


Figure 2-5. Displaying the Baud Rate

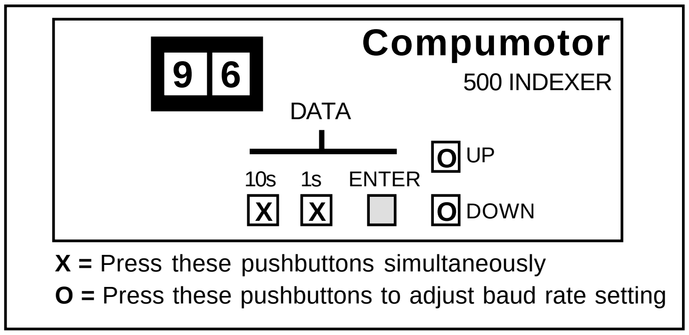
Changing the Baud Rate

Use the following steps to change the baud rate setting:

① Press and hold the **10s** and **1s** pushbuttons.

② While holding the **10s** and **1s** pushbuttons, press the **UP** or **DOWN** pushbutton until the desired baud rate is displayed.

The baud rate setting will be automatically saved in the Model 500's memory. The last baud rate setting that is displayed prior to the release of the **10s** and the **1s** buttons will be the baud rate value that will be saved and implemented until you change it again.



9 6
Compumotor
500 INDEXER
DATA
10s
1s
ENTER
X
X
O UP
O DOWN
X = Press these pushbuttons simultaneously
O = Press these pushbuttons to adjust baud rate setting

Figure 2-6. Changing the Baud Rate

Verify the Device Address

The default device address setting for the Model 500 is **01**. To confirm that the device address is set to 1, press the **10s** and the **ENTER** pushbuttons simultaneously. The digits **01** will be displayed (see Figure 2-6).

If the display shows a setting other than **01**, use the steps provided in *Changing the Device Address* (below) to change it to **01**.

The address should be left at setting 01 for the procedures discussed in this chapter.

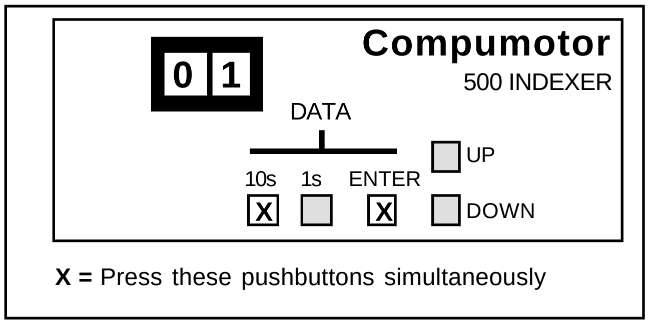


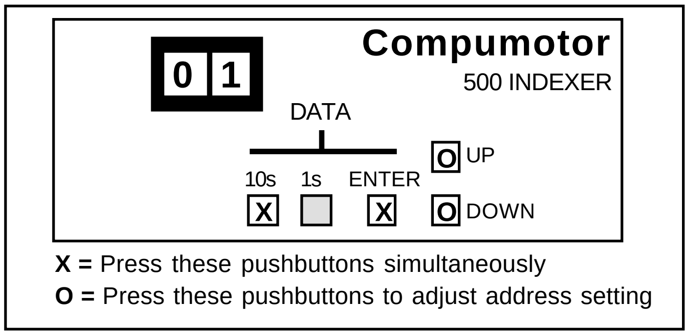
Figure 2-7. Displaying Device Address

Changing the Device Address

If you need to change the address, use the following steps:

① Press and hold the **10s** and **ENTER** pushbuttons.

② While holding the two pushbuttons, press the **DOWN** or **UP** pushbutton until the desired address is shown (see Figure 2-8).



0 1
Compumotor
500 INDEXER
DATA
10s
1s
ENTER
X
X
O UP
O DOWN
X = Press these pushbuttons simultaneously
O = Press these pushbuttons to adjust address setting

Figure 2-8. Changing the Device Address

Before you test the RS-232C interface, be sure that:

❑ The terminal or computer's handshaking function is disabled
❑ The RS-232C interface is connected properly
❑ If you are using a computer, it must be converted to operate as a terminal
❑ The communication parameters are properly set

③ Test RS-232C Interface

To test your RS-232C communication link, follow the steps below.

① Power-up your computer or terminal *and then* power-up the Model 500.

② Press the space bar several times. The cursor should move one space each time you press the space bar. If your terminal displays garbled characters, check the terminal's protocol set-up. The baud rate setting probably does not match the Model 500's setting.

③ If the terminal does not display garbled characters, press the space bar several times. If the cursor does not move, disconnect the **RS232** plug from the 500 and measure the cable's continuity . If you do not measure > -3V on Pin 1 (**Rx**) to Pin 3 (**GND**), your cable is wired incorrectly or your terminal is broken. If you measure > -3V on Pin 2 (**Tx**) to Pin 3 (**GND**), then switch the Model 500's transmit (**Tx**) and receive (**Rx**) wires and try this test again.

④ Once you are able to make the cursor move, enter some characters. These characters should appear on the computer or terminal display. If each character appears twice, your host is set to half-duplex. It should be set to full-duplex.

⑤ If your connections are correct, and the device address is set to **01**, reset the Model 500. Press the **UP** and **DOWN** pushbuttons on the front panel simultaneously as shown in Figure 2-9. This is the equivalent to removing power to the unit, then re-applying power.

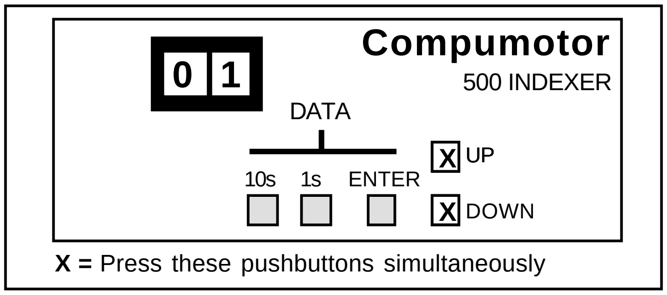


Figure 2-9. Resetting the Model 500

You should see the following message on your terminal or computer screen:

```
*READY
>
```

⑥ Issue the indexer Status Request (`1R`) command to check the general status of the indexer (do not forget to enter the carriage return after the command). The **1** preceding the command refers to Model 500 device address (one). The Model 500 should provide the following response: `*R`.

Command	Response
`> 1R`	`*R`

The `*R` response indicates that the indexer is ready to accept a command. This response confirms that you have proper two-way communication between the Model 500 and your computer. All Status Request (`R`) command responses are described in the **Model 500 Indexer Software Reference Guide**. If you do not get a response, check your connections and try to issue the command again. If the problems persists, refer to Chapter 6, *Maintenance & Troubleshooting*.

⑦ Issue the Position Report (`1PR`) command to determine the absolute motor position. The `1` preceding the command refers to Model 500 device address (one). The Model 500 should provide the following response: `*+ØØØØØØØØØØ`.

```
Command          Response
> 1PR            *+ØØØØØØØØØØ
```

The `*+ØØØØØØØØØØ` response indicates that the absolute motor position is Ø. If you do not get a response, check your connections and try to issue the command again. If the problems persists, refer to Chapter 6, *Maintenance & Troubleshooting*. After you have successfully completed this step, remove power from the Model 500 and proceed to Step ④, *Connect Motor Driver*.

④ Connect Motor Driver

Before you proceed with this step, set the drive functions and connect the motor to the drive according to the instructions that accompanied your motor/drive package.

CAUTION

Before connecting to your Motor/Drive system, be sure that power is not applied to the Model 500.

The Model 500 is compatible with all Compumotor step and direction drives. The **MOTOR DRIVER** connector on the front panel of the Model 500 interfaces with these drives. The Model 500 is shipped with Compumotor's standard 15-pin D-to-25-pin D motor/driver cable. As described below, connections are made either to the drive's 25-pin D connector or to the drive's screw-terminal connector, depending on the type of drive you are using.

Motor Driver Connections (Compumotor Drives)

The Model 500's **MOTOR DRIVER** connector (15-pin D) is compatible with the **INDEXER** connector (25-pin D) on the following Compumotor products. A 15-pin D-to-25-pin D cable is provided with the Model 500.

- ❑ **A** Microstepping Motor/Drive system
- ❑ **C** Microstepping Motor/Drive system
- ❑ **S** Microstepping Motor/Drive system
- ❑ **PK3** Full/Half stepping Motor/Drive system
- ❑ **LE** Microstepping Motor/Drive system
- ❑ **L** Microstepping linear Motor/Drive system
- ❑ **KH** Brushless servo Motor/Drive system
- ❑ **KS** Brushless servo Motor/Drive system

Figure 2-10 illustrates the connection for the motor/drive systems that support the standard 25-pin D indexer input interface.

With Compumotor's M microstepping motor/drive system, you can plug the Model 500's 25-pin D connector end of the motor/driver cable to the mating 25-pin D connector on the M Drive indexer cable. This cable is included in the M Drive's ship kit.

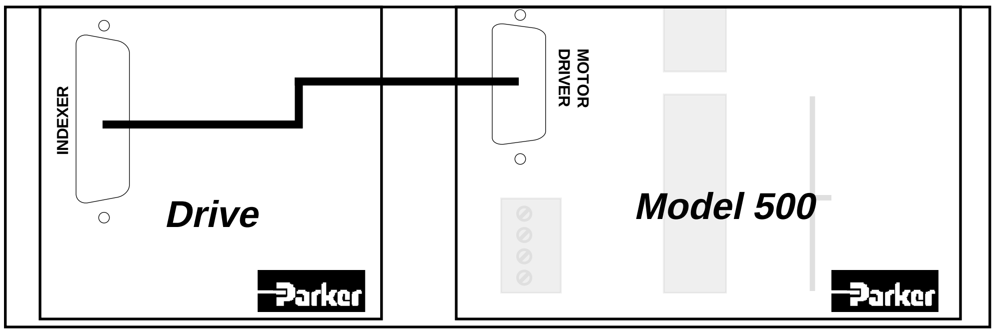


Figure 2-10. 25-Pin Motor/Driver Connections

Screw Terminal Connections

The following Compumotor products have screw-terminal indexer interface connectors:

❏ **Compumotor Plus** servo Motor/Drive system
❏ **PK2** Full/Half stepping Motor/Drive system
❏ **PK130** ministepping Motor/Drive system

To connect the Model 500 to these products, you must cut the 25-pin D connector off of the cable and wire the cable leads directly to the drive. Use Table 2-2 to wire the **MOTOR DRIVER** cable.

Model 500 Pin Number	Wire Color	Description
1	Red	Step+
2	Green	Direction+
4	Grey	In Position
5	Yellow	Drive Fault
8	Shield wire	Shield
9	Black	Step-
10	White	Direction-
11	Blue	Shutdown+
12	Purple	Shutdown-
13	Orange	Ground
14	Brown	Ground

Table 2-2. Screw Terminal **MOTOR DRIVER** Connections

Motor Driver Connections
(Non-Compumotor Drives)

If you are using a non-Compumotor drive, connect the drive to the Model 500's **MOTOR DRIVER** connector according to the pin outs in Table 2-3.

Pin #	IN/OUT	Description	Specification
1	OUT	Step +	Active high (0.75VDC low - 3.5VDC high) ±60 mA maximum
2	OUT	Direction +	Active high (0.75VDC low - 3.5VDC high) ±60 mA maximum
3	OUT	Ground	
4	-----	*Reserved*	-----
5	IN	Drive Fault	TTL-compatible w/internal 3.3KΩ pull-ups to +5VDC; 0 - 5VDC
6	OUT	CW Step	Open-collector; sink up to 250 mA; 24VDC maximum
7	OUT	CCW Step	Open-collector; sink up to 250 mA; 24VDC maximum
8	IN	Shield	
9	IN	Step -	Active high (0.75VDC low - 3.5VDC high) ±60 mA maximum
10	IN	Direction -	Active high (0.75VDC low - 3.5VDC high) ±60 mA maximum
11	OUT	Shutdown +	Active high (0.75VDC low - 3.5VDC high) ±60 mA maximum
12	IN	Shutdown -	Active high (0.75VDC low - 3.5VDC high) ±60 mA maximum
13 - 15	OUT	Ground	

Table 2-3. **MOTOR/DRIVER** Connector Pin Outs

⑤ *Test Procedure*

Use the following steps to test the Model 500 system. The Model 500-FOL is configured as an indexer (same as the standard Model 500) when you receive it from the factory. If you are using the Model 500-FOL leave it in the Indexer mode (**1FSIØ**) while performing the test procedures below. In Chapter 3, *Installation*, you will test the encoder interface. Chapter 4, *Application Design*, provides practical examples of the following mode operation.

① Check to make sure you properly connected the power cable, established RS-232C communications, and connected the motor driver to the Model 500. **Apply power to the system**.

② The following response should appear on your terminal:

```
*READY
>
```

③ If you do not have a Compumotor servo drive system, the two green LEDs (**POWER** and **CPU READY**) will be on and the display will be blank. *If you have a Compumotor servo drive (KS, KH or Compumotor Plus), error* **16** *will flash on the LED display. This indicates that your drive has a fault under normal operation*. Compumotor's servo products (KH Series, KS Series, and Compumotor Plus drives) use the *drive fault line* (pin 5 in Table 2-2) on the **MOTOR DRIVER** cable to indicate that the Model 500 drive system has faulted. For the Model 500 to recognize the signal, you must enable a software switch by issuing the **SSQ1** command.

Command	Description
> **SSQ1**	Enables drive fault indicator for Compumotor servos

To remove error **16** from the display, type the following:

Command	Description
> **ON**	Turns on the step output command

The Model 500 is on and no error codes should be flashing.

④ The Model 500 has two end-of-travel limit inputs: CW and CCW. These limit inputs prevent the load from crashing into a mechanical stop and damaging equipment or injuring personnel. Compumotor ships the Model 500 from the factory with the limits enabled. Therefor, to bench test your system you must disable the limit inputs with the Limit Disable (**LD3**) command (**make sure the load is not attached to the motor**). You will establish load-activated limit switches in your system in *Chapter 3, Installation*.

Command	Description
> **1LD3**	Disables hardware limits

To check the state of your hardware limits command, enter **1LD**.

Command	Response
> **1LD**	***3_No_Limits_Enabled**

⑤ Refer to the motor/drive user guide to determine what resolution is used. Use the **MR** command to set the Model 500's motor resolution to match the motor/drive system's resolution. ***The Model 500's default motor resolution is 25,000 steps/rev*** *(or* **MRnnnn***).*

Command	Description
> **MRnnnn**	Sets the resolution to **nnnn** steps/rev

⑥ Enter the following commands to move the motor 25,000 steps:

Command	Description
> **A1Ø**	Sets acceleration to 10 rps^2
> **AD1Ø**	Sets deceleration to 10 rps^2
> **V1**	Sets velocity to 1 rps
> **D25ØØØ**	Sets distance to 25,000 steps
> **G**	Initiates motion

The motor should move 25,000 steps. If your motor/drive resolution is set to 5,000 steps/rev and the Model 500 was set to 25,000 steps/rev, the motor should move 5 revs.

This completes the bench test. In Chapter 3, you will install and test the rest of your system's components.

Chapter 3. Installation

Chapter Objectives

The information in this chapter will enable you to:

- ❏ Mount all system components properly
- ❏ Connect all electrical system inputs and outputs properly
- ❏ Ensure that the complete system is installed properly
- ❏ Perform basic system operations

You must perform all the bench test procedures in Chapter 2, *Getting Started*, before you can proceed with the permanent installation process.

System Specifications

Refer to Table 3-1 for Model 500 system specifications and Table 3-2 for factory default settings.

Parameter	Value
Performance	
Stepping Accuracy	+0 steps from preset total
Velocity Accuracy	+0.02% of max rate above 1 rps
Velocity Repeatability	+0.02% of max rate
Motor Resolutions	200 steps/rev - 50,000 steps/rev
Position Range	+0 - 99,999,999 steps
Velocity Range	25,000 steps/rev: 0.00001 - 40 rps; 5000 steps/rev: 0.00001 - 100 rps
Acceleration Range	25,000 steps/rev: 0.01 - 999.99 rps^2; 5000 steps/rev: 0.01 - 999.99 rps^2
Power	
Voltage	90 - 240VAC, or 127 - 340VDC
Frequency	50 - 6Hz
Current	Less than 1.3A RMS max
Inputs	
Command Interface	
Type	RS-232C serial type, 3-wire (Rx, Tx & GND)
Parameters	Selectable baud rate (300, 600, 1200, 2400, 4800, 9600); 8 data bits, 1 stop bit, no parity
Configuration	Up to 16 indexers daisy-chained from a single host RS-232 port
Programmable Inputs (13 total)	Optically isolated 5 - 30VDC. May be used for BCD recall of motion programs and for interactive machine control.
CW & CCW Limits, Home Enable	TTL levels, optically isoltated, 0 - 30VDC
Outputs	
Step, Direction, & Shutdown	Differential drive +3.0VDC min., +60mA high, +1.0VDC max., -60mA signal low
Fault Output	Normal open or closed, 2A at 120VA
Programmable Outputs (8 total)	Open-collector, user must supply pull-up resistor with min. of 100Ω resistance
Encoder A, B and Z Channels	Differential or single-ended, active high, <0.5VDC = low, >3.0VDC = high, 4.5mA sink
Max Frequency (A & B channels)	80KHz (line rate)
Minimum Pulse (Z channel)	1MHz (40 rps) if using Step and Direction source at 25000 step/rev resolution (see `MR` command description for max. speed at other resolutions)
	500 nanoseconds
Motion Programming	
Memory Storage	8K battery-backed RAM
Number of Programs	99 sequences, dynamically allocated to 8K
RS-232C Execution	Program execution may be initiated from the RS-232C interface with Run (`XR`) command
BCD Input Execution	Sequence select BCD inputs using thumbwheels
Power-up Auto Run	Sequences may be automatically executed on power-up via power-up sequence (#100)
Front Panel Execution	Sequences may be executed using the front panel bushbuttons
Environmental	
Operating Temperature	32°F - 122°F (0°C - 50°C)
Storage Temperature	-22°F - 185°F (-30°C - 85°C)
Humidity	0 - 95%, non-condensing

Table 3-1. Model 500 System Specifications

Parameter	Value
Default Settings	
RS-232C Communication	9,600 Baud Rate; 8 Data Bits; 1 Stop Bit; No Parity; Full Duplex
Device Address	**01** (refer to Chapter 2 for procedures to display and change via front panel pushbuttons)
Hardwae Interfaces	Front panel pushbuttons enabled Sequence and Ratio Select disabled RS-232C enabled Inputs active low, configured as triggers, with dedicated CW, CCW and Home limits Outputs active low, configured as programmable outputs
Limits	Hardware limits enabled; Software limits disabled

Motion Parameters	Acceleration = 10 rps^2
	Deceleration = 10 rps^2
	Velocity = 1 rps
	Motor Resolution = 25,000 steps/rev
	Distance = 25,000
	Encoder Resolution = 4,000
	Use the **DR** command to display the present configuration of the Model 500

Table 3-2. Model 500 Factory Default Settings

Environmental Considerations

You must consider the environment in which your system will be operating. Proper mounting, wiring, and grounding will ensure trouble-free operation. The Model 500 is designed to operate in an industrial environment; however, severe atmospheric contamination, electrical noise, or temperature extremes can affect performance of the system. ***Operate the drive and motor within its designed specifications.***

Compumotor recommends that you operate and store the Model 500 in the conditions identified in Table 3-1 above.

Complete System Configuration

Safety is the primary concern when installing any motion control system. This chapter provides installation guidelines that are designed to preserve the safety of the operator and the equipment. **You should install all Compumotor hardware in conformity with local and national electrical and safety codes.**

This chapter provides detailed instructions on all aspects of the Model 500 's installation and configuration. Once the system has been properly installed and initial adjustments are made, there should be little or no adjustment required to maintain normal operation.

Indexer Mounting

The Model 500 should be installed in an enclosure that will protect it from atmospheric contaminants such as oil, metal flakes, moisture, and dirt. The National Electrical Manufacturers Association (NEMA) has established standards that define the degree of protection that electrical enclosures provide. Industrial application environments may contain airborne contaminants, so the enclosure used should conform to at least a *NEMA TYPE 12 standard*.

Refer to Figure 3-1 for the Model 500's dimensions.

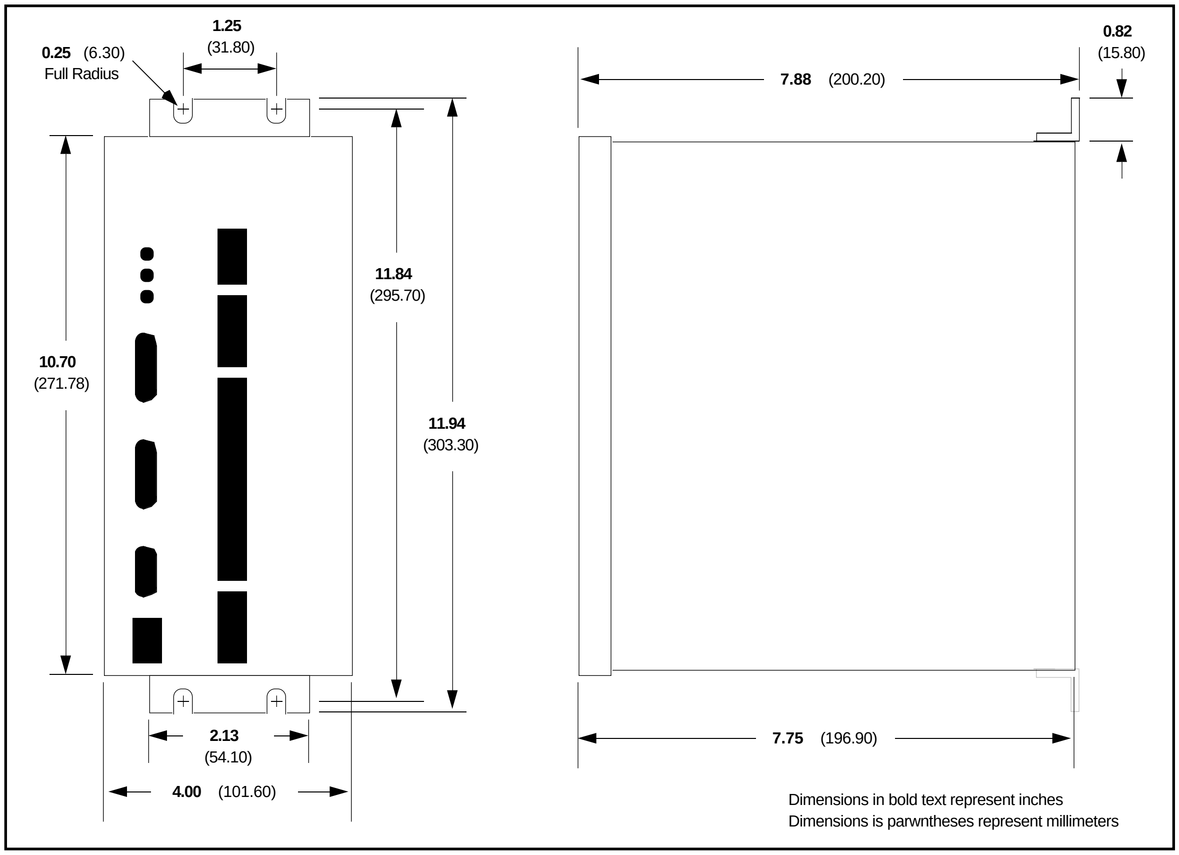


Figure 3-1. Model 500 Dimensions

Panel Layout

If you mount the Model 500 in an enclosure, observe the following guidelines:

❏ Make sure there is at least 4 inches of unrestricted air-flow space around the Model 500 to allow for efficient convection cooling (refer to Figure 3-2). Fan cooling may be necessary if adequate air flow is not provided.

❏ Do not mount large, heat-producing equipment (such as a drive) directly beneath the Model 500. The maximum allowable ambient temperature directly below the Model 500 is 40°C.

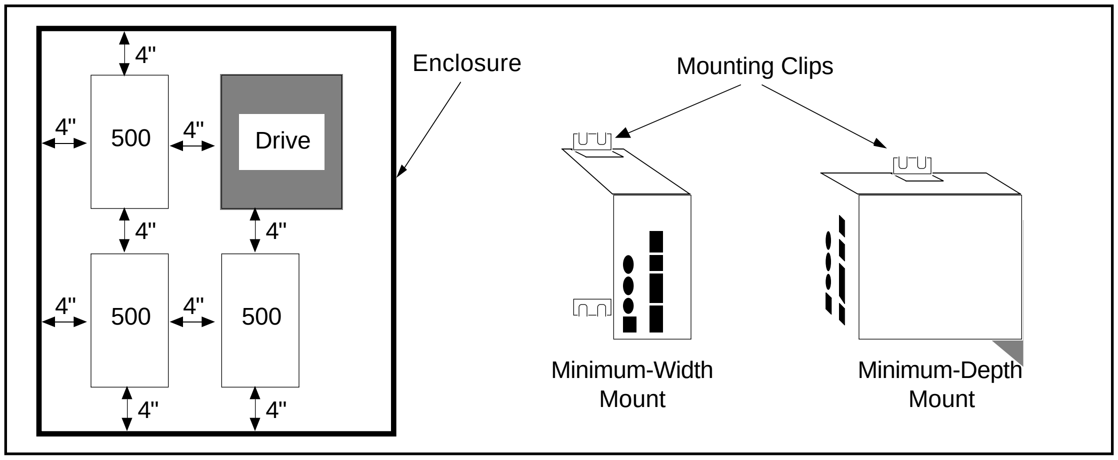


Figure 3-2. Panel Layout Guidelines and Mounting Options

Minimum Width or Depth

You can mount the Model 500 for minimum depth or width, depending on how you attach the mounting clips to the unit (see Figure 3-2 above).

The Model 500 is shipped with the two clips attached to the back of the indexer, opposite the power connectors. This allows the maximum amount of panel space possible.

For minimum depth, attach the two clips to the side of the indexer. This allows you to mount the drive in the shallowest possible enclosure.

System Wiring

This section provides information for system I/O and encoder connections and daisy-chaining. Refer to Chapter 2, *Getting Started*, for procedures to connect the following system components:

- ❏ Power
- ❏ RS-232C Communication
- ❏ Motor Driver

Test procedures are provided later in this chapter to verify that you have performed all the wiring correctly.

Wiring Guidelines

Proper grounding of electrical equipment is essential to ensure the safety of personnel. You can reduce the effects of electrical noise due to electromagnetic interference (EMI) by grounding. All Compumotor equipment should be properly grounded. A good source of information on grounding requirements is the National Electrical Code published by the National Fire Protection Association of Boston, Massachusetts.

In general, all components and enclosures must be connected to earth ground through a grounding electrode conductor to provide a low impedance path for ground fault or noise-induced currents. All earth ground connections must be continuous and permanent. Compumotor recommends a single-point grounding setup. Prepare components and mounting surfaces prior to installation so that good electrical contact is made between mounting surfaces of equipment and enclosure. Remove the paint from equipment surfaces where the ground contact will be bolted to a panel and use star washers to ensure solid bare metal contact.

For temporary installation, or when you cannot implement the grounding method described above, you must connect the **GND** terminal on the AC power connector to the earth ground.

WARNING

All AC power must be disconnected prior to installation wiring. Failure to observe safe working practices when installing or servicing this equipment can expose you to dangerous voltages.

PROG INPUTS Connections

The Model 500's **PROG INPUTS** connector has 16 inputs: 13 programmable inputs, 2 end-of-travel limit inputs, and 1 home limit input. These inputs have an internal 5V supply. **These inputs are <u>not</u> sinking inputs.** You can directly wire up to 30V to the inputs (no current-limiting resistor required). This allows you direct interface to a PLC. Figure 3-3 illustrates the Model 500's internal input circuit.

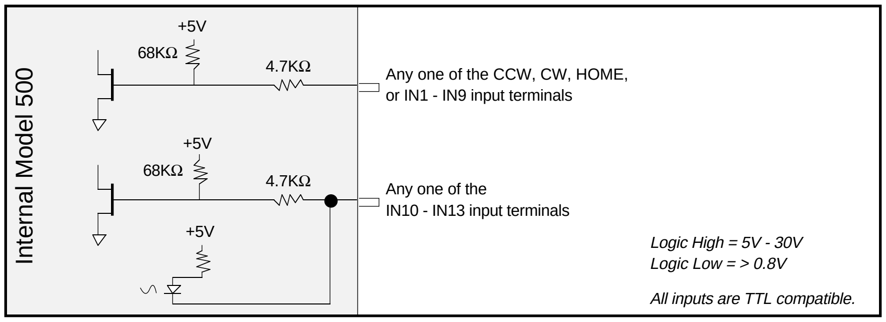

Internal Model 500
+5V
68KΩ
4.7KΩ
Any one of the CCW, CW, HOME, or IN1 - IN9 input terminals
+5V
68KΩ
4.7KΩ
Any one of the IN10 - IN13 input terminals
+5V
Logic High = 5V - 30V
Logic Low = > 0.8V
All inputs are TTL compatible.

Figure 3-3. Model 500 Internal Input Circuit

You can change the active level of the inputs with the Configure Input Level (**INL**) command. The default setting for these inputs is **INLØ** (input active low). Figure 3-4 illustrates a typical wiring configuration (**INLØ** mode) for the inputs on the **PROG INPUTS** Connector.

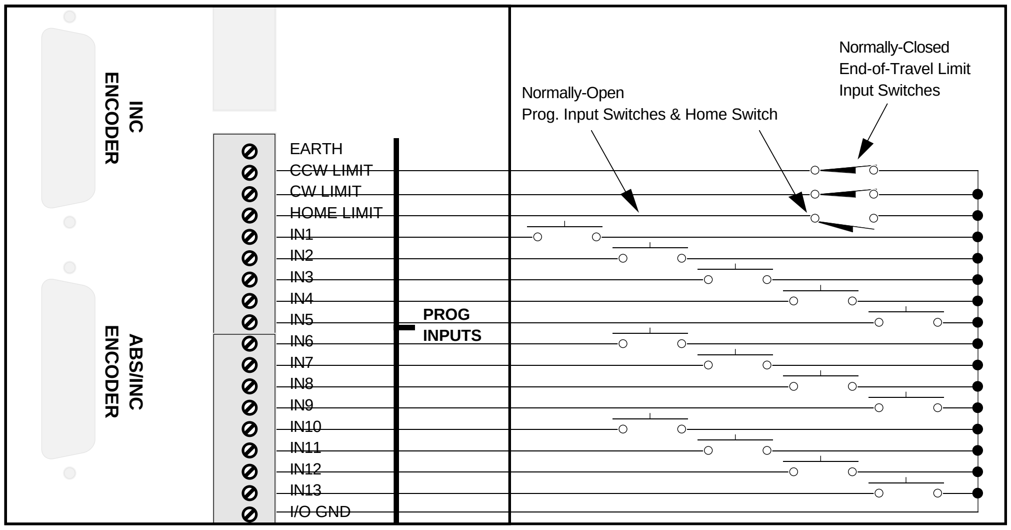


Figure 3-4. Typical Connections for **PROG INPUTS** Connector

End-of-Travel Limits

The Model 500 has two dedicated hardware end-of-travel limits (**CCW LIMIT** and **CW LIMIT** on the front panel). When you power up the Model 500 these limits are enabled. If you want to test the Model 500 without connecting the CW and CCW limit switches, you will have to disable the limit inputs with the Limit Disable (**LD3**) command. If you attempt to perform a move without disabling the inputs or using the **LD3** command, the **FAULT** LED will illuminate and error code **41** or **42** (depending on whether it was the CW or CCW limit) will flash on the display. These error codes indicate that the motor has reached the hardware limit.

The Model 500 also has software limit capabilities. The software limits are disabled when you power up the system. If you do not plan to use any hardware limit switches, you can provide added system safety by enabling software limits with the Software Limits (**SL**) command (refer to the **SL** command description in the **Model 500 Software Reference Guide**).

Figure 3-4 illustrates standard end-of-travel switch connections using normally-closed limit switches. This helps ensure a safer operating environment in which the motor will be stopped if the switch is opened or is faulty, or if the connection is broken somehow. You can, however, use normally-open switches by changing the active state of the end-of-travel limits to *active high* with the **OSA1** command.

Home Limit

The Model 500's dedicated **HOME LIMIT** input provides a reference position for your application's motion. The active state of the home limit input is determined by the **OSC** command. This input is useful if you want your machine to start its operation from a repeatable position. You can use this input in conjunction with the Go Home (**GH**) command or the Go Home input set-up using the Input (**IN**) command. When the Model 500 executes a **GH** command, it scans the Home Limit input until the switch activates the Home Limit input circuit. The Homing function is discussed in detail in Chapter 4, *Application Design*.

Figure 3-4 illustrates connections for a typical normally-open home input switch.

Programmable Inputs

The Model 500 has 13 programmable inputs. Each input can be programmed to serve any of up to 25 different functions with the **IN** command (refer to the **IN** command description in the **Model 500 Indexer Software Reference Guide**). The inputs can be used with PLCs and can be configured along with the outputs to interface with thumbwheel switches. As part of the set-up of your application, you may need to configure the inputs to have the functionality you require.

To activate the inputs when the input active level is low, connect the input pin to the **I/O GND**. If you are using a relay to activate the inputs, use solid state relays to prevent electrical noise from impairing the performance of your application.

Using the Input (**IN**) command, you can program the programmable inputs to perform various functions (i.e., stop input, sequence-select input, etc.).

Figure 3-4 illustrates typical normally-open switches connected to accommodate the default **INLØ** (input active-low) mode.

PROG OUTPUTS Connections

The Model 500's **PROG OUTPUTS** connector has 8 programmable open-collector outputs (**OUT1** - **OUT8**) that can be directly wired to 30VDC and can sink up to 300mA. Figure 3-5 is a diagram of the Model 500's output circuit.

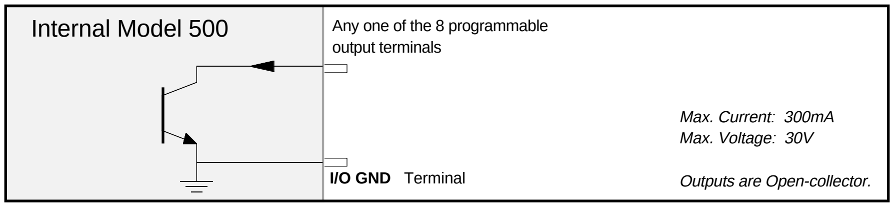

Internal Model 500
Any one of the 8 programmable output terminals
Max. Current: 300mA
Max. Voltage: 30V
I/O GND Terminal
Outputs are Open-collector.

Figure 3-5. Model 500 Internal Output Circuit

You can directly interface these outputs with a PLC, *if the PLC can directly interface with an open-collector output*. If not, you must use a pull-up resistor (minimum of 100Ω) and an external 5 - 30VDC power supply to use these outputs (refer to Figure 3-6).

If you are using a relay to activate the outputs, use solid state relays to prevent electrical noise from impairing the performance of your application.

These outputs can be configured for different functions (i.e., moving/not moving) with the Configure Output (**OUT**) command. You can also configure the active level of the outputs with the Configure Output Level (**OUTL**) command. The default setting for these outputs is **OUTLØ** (Output Active Low).

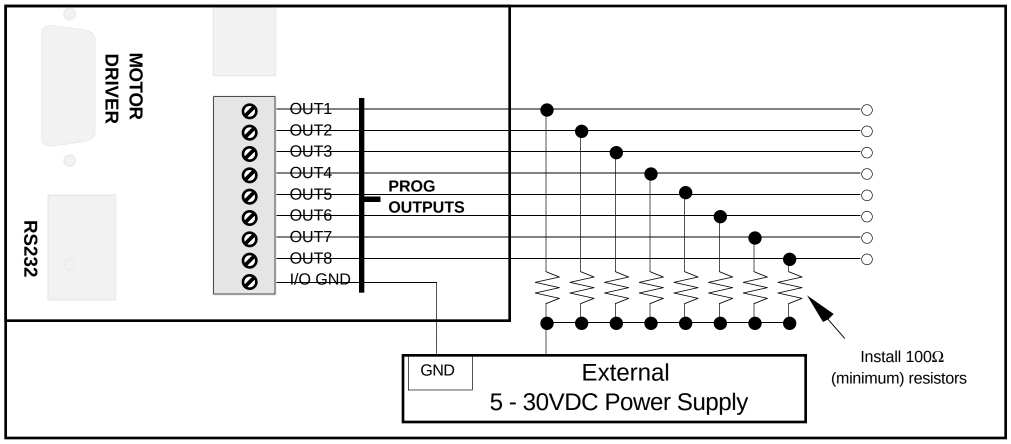


Figure 3-6. Programmable Output Wiring Diagram

Fault Output Connections

The Model 500 provides a programmable fault output that defaults to the same function as the other programmable outputs; referred to as **OUT9**, you can use the **OUT** command to program its function as you would the other eight outputs (refer to the **OUT** command description in the <u>**Model 500 Indexer Software Reference Guide**</u>.

It is also hard-wired so that if the microprocessor is reset, the **CPU READY** LED goes out, or if a fault occurs, the relay will change state.

The fault output is labeled **FAULT OUT** on the front panel. There are two types of outputs available: normally-open and normally-closed (see Figure 3-7 for connections). Normally-closed outputs will **open** if a fault occurs. Normally-open outputs will **close** if a fault occurs. The relay fault outputs are rated for up to 120VAC and 10A of current.

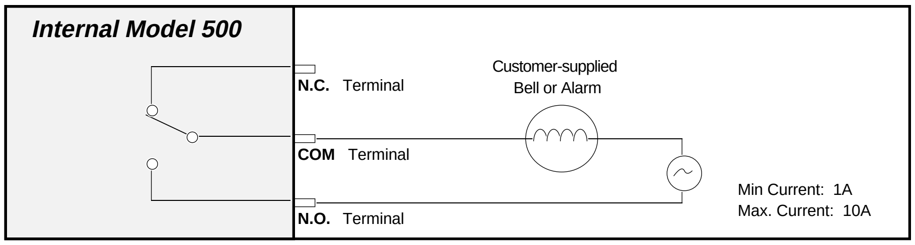


Figure 3-7. Fault Output Connections

Daisy-Chain Connections

If you are not daisy-chaining multiple Model 500 Indexers, you can skip this section.

You may daisy-chain up to 16 Model 500s to one serial port on a terminal. Figure 3-8 shows a three-indexer daisy-chain configuration from one controlling terminal or computer.

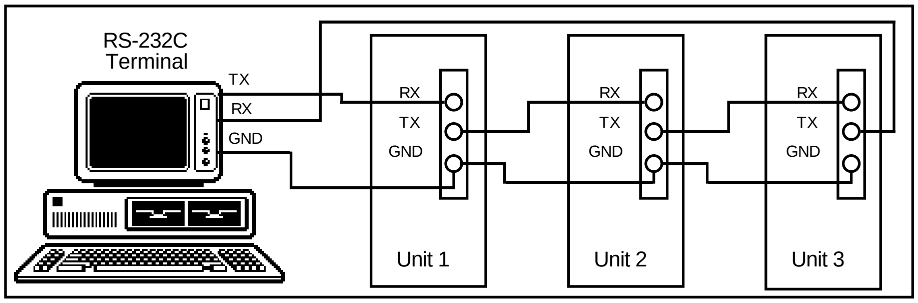


Figure 3-8. RS-232C Daisy-Chain Configuration

If you are daisy-chaining 500s, you should establish different addresses for each unit so you can distinguish them when you are programming. Refer to Chapter 2, *Getting Started*, for instructions on displaying and changing the 500's device address.

Commands prefixed with a device address affect only the unit specified. Commands without a device address affect all units on the daisy chain. For instance, entering the **1G** command instructs <u>only unit #1</u> to go (move), but entering the **G** command instructs <u>all units</u> on the daisy-chain to go.

Any command that causes the drive to transmit information from the RS-232C port (such as a status or report command), must be prefixed with a device address. This prevents daisy-chained units from all transmitting at the same time.

No 500 executes a device-specific command unless the unit number specified with the command matches the 500's unit number. Device-specific commands include both buffered and immediate commands.

You must use status request commands in an orderly fashion. Commands should only be issued when the host is ready to read the response. You should not send new commands until you receive a response from the previous status request command. In particular, you should not issue an immediate-status command until the host receives a buffered command status response. If this is not followed, the command responses will be intertwined, rendering the data useless.

If you enable the Interactive mode (**SSI1**), only the 500 that is set to address 1 will respond with a prompt (>). This prevents all the 500s from sending out > in a daisy-chain. Typically, you should disable the Interactive mode when you use a host computer with the 500.

Encoder Connections

If you purchased the Model 500 Indexer and will not be using encoder feedback, you can skip this section.

The Model 500 has two encoder interfaces. One is labeled **INC ENCODER** on the front panel and the other is labeled **ABS/INC ENCODER**. The **INC ENCODER** interface accepts only incremental encoders (*does not accept Z Channel*), while the **ABS/INC ENCODER** interface accepts either an incremental (*accepts Z Channel*) or an absolute encoder.

The Model 500-FOL (*following*) option can follow from either the **ABS/INC** or the **INC ENCODER** interface, depending on the setting of the **`FSJ`** command. When using one encoder interface for following, you can use the other encoder interface to perform position maintenance while in the following mode. The encoders used for position feedback that are compatible with the Model 500 and the Model 500-FOL are listed below:

- ❏ Compumotor -E Series encoders (or any other incremental encoder with quadrature signals and a Z channel)
- ❏ Compumotor's AR-C and AL-C absolute encoders

Encoder Configurations

Depending on your application, you can use the encoder interfaces in one of the configurations listed in Table 3-3.

Model 500

Application	Configuration
Open-loop Indexing	1. **INC ENCODER**—Not used **ABS/INC ENCODER**—Not used
Indexing, Position Maintenance	1. **INC ENCODER**—Incremental encoder used for position information **ABS/INC ENCODER**—Not used 2. **INC ENCODER**—Not used **ABS/INC ENCODER**—Incremental encoder used for position information 3. **INC ENCODER**—Not used **ABS/INC ENCODER**—Absolute encoder used for position information

Model 500-FOL

Application	Configuration
Following, No Position Maintenance	1. **INC ENCODER**—Incr. encoder provides following info. from primary axis **ABS/INC ENCODER**—Not used 2. **INC ENCODER**—Not used **ABS/INC ENCODER**—Incr. encoder provides following info. from primary axis 3. **INC ENCODER**—Not used **ABS/INC ENCODER**—Abs. encoder provides following info. from primary axis
Following, Position Maintenance	1. **INC ENCODER**—Incremental encoder used for position information. **ABS/INC ENCODER**—Incr. encoder provides following info. from primary axis 2. **INC ENCODER**—Incremental encoder used for position information. **ABS/INC ENCODER**—Abs. encoder provides following info. from primary axis 3. **INC ENCODER**—Incr. encoder provides following info. from primary axis **ABS/INC ENCODER**—Incremental encoder used for position information 4. **INC ENCODER**—Incr. encoder provides following info. from primary axis **ABS/INC ENCODER**—Absolute encoder used for position information

Table 3-3. Possible Encoder Configurations (Model 500 & Model 500-FOL)

Procedures to connect the encoder to the model 500 are provided in the sections below.

Position maintenance, *encoder step mode*, and *stall detect* functions are discussed in detail in Chapter 4, *Application Design*. *Following functions* are discussed in Chapter 5, *Following*.

Incremental Encoders

The Model 500 requires the standard phase A+, phase A-, phase B+, phase B-, phase Z+ and phase Z- inputs. You can use single-ended encoders by not connecting the encoder's A-, B-, and Z- outputs. The phase A and phase B inputs provide quadrature input signals. The Model 500 has an internal quadrature detection circuit. The phase Z inputs provide one pulse per revolution and are used to find the final home position when you execute a Go Home (`GH`) command (applies only if you use an incremental encoder connected to the **ABS/INC ENCODER** interface).

The resolution of the encoder is determined by the particular encoder that you choose. If the encoder that you choose provides 1,000 pulses per revolution (a 1,000-line encoder), it will have 4,000 distinct locations per revolution after going through the quadrature detection circuit in the Model 500. The 1,000 lines per revolution is known as the *pre-quadrature resolution*. This means that the phase A input provides 1,000 pulses in one revolution. The phase B input also provides 1,000 pulses per revolution. These pulses are offset from each other by 90° as shown in Figure 3-9.

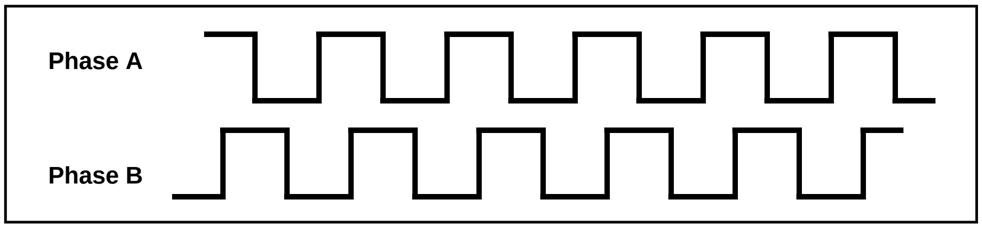


Figure 3-9. Quadrature Signals From an Incremental Encoder

The two phases go through a quadrature detection circuit. They provide four times the pre-quadrature resolution (1,000 pulses per revolution) or 4,000 counts per revolution. This is known as the *post-quadrature* resolution. The Z channel sends one pulse every time the encoder passes its zero point. Therefore, it provides one pulse per revolution.

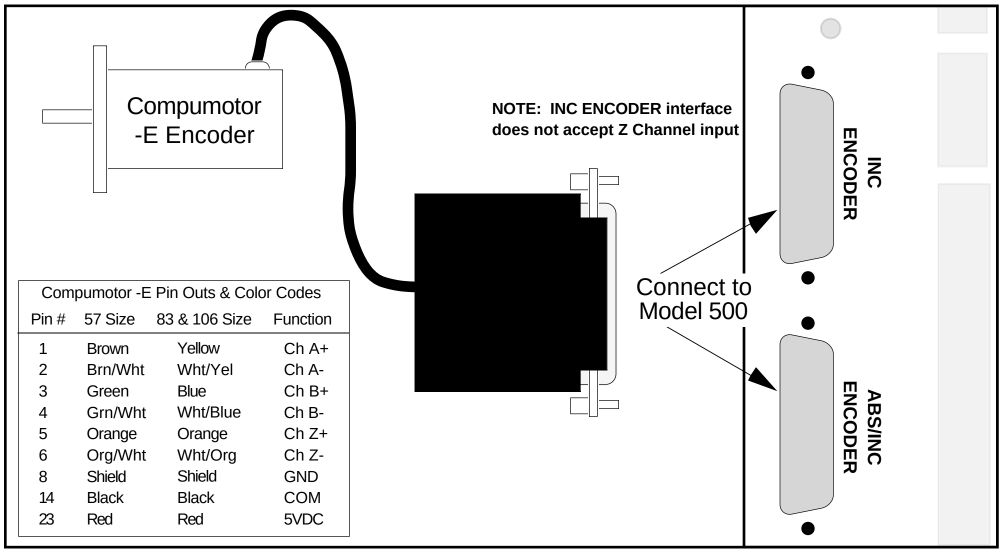
The maximum rate at which the phase A and phase B pulse can be detected determines the maximum speed that the motor can achieve. This is limited by the bandwidth of the encoder interface circuit. The Model 500 has a bandwidth of 80KHz pre-quadrature. With the incremental encoder in this example, 1,000 pulses per revolution are received. If the encoder is moving at 50 rps, the Model 500 will receive 1,000 • 50 pulses per revolution. This is less than the maximum bandwidth of the interface. A speed of 50 rps is adequate for most applications. If your application requires higher speeds, use a lower resolution encoder.

Incremental Encoder Connections

The **INC ENCODER** and **ABS/INC ENCODER** connectors accept two-phase quadrature incremental encoders with differential or single-ended outputs (+5V TTL-compatible). The Model 500 provides the 5VDC at 500mA supply for the encoder. The maximum frequency per channel is 80KHz.

The* INC ENCODER *does not accept Z channel inputs. If you need to home to a Z channel, use the* ABS/INC ENCODER *connector configured for incremental operation (refer to the Absolute Encoders section discussed later in this chapter).

If you use a Compumotor -E incremental encoder, it will be provided with a cable that has a 25-pin D connector compatible with the Model 500's 25-pin encoder input. Connect the encoder cable to the Model 500 as shown in Figure 3-10. Figure 3-10 also lists the encoder cable color codes and pin outs.



Compumotor -E Pin Outs & Color Codes

Pin #	57 Size	83 & 106 Size	Function
1	Brown	Yellow	Ch A+
2	Brn/Wht	Wht/Yel	Ch A-
3	Green	Blue	Ch B+
4	Grn/Wht	Wht/Blue	Ch B-
5	Orange	Orange	Ch Z+
6	Org/Wht	Wht/Org	Ch Z-
8	Shield	Shield	GND
14	Black	Black	COM
23	Red	Red	5VDC

Figure 3-10. Compumotor -E Encoder Connections (Differential)

If you do not use a Compumotor encoder, you must make your own cable. Two 25-pin D shells are included in the Model 500 ship kit for making a connector. You must solder the connections to make the cable.

The incremental encoder inputs can be used either differentially or single-ended. Refer to Figure 3-11 for differential and single-ended solder connections to the 25-pin D connector. Figure 3-12 shows the typical incremental encoder interface circuit.

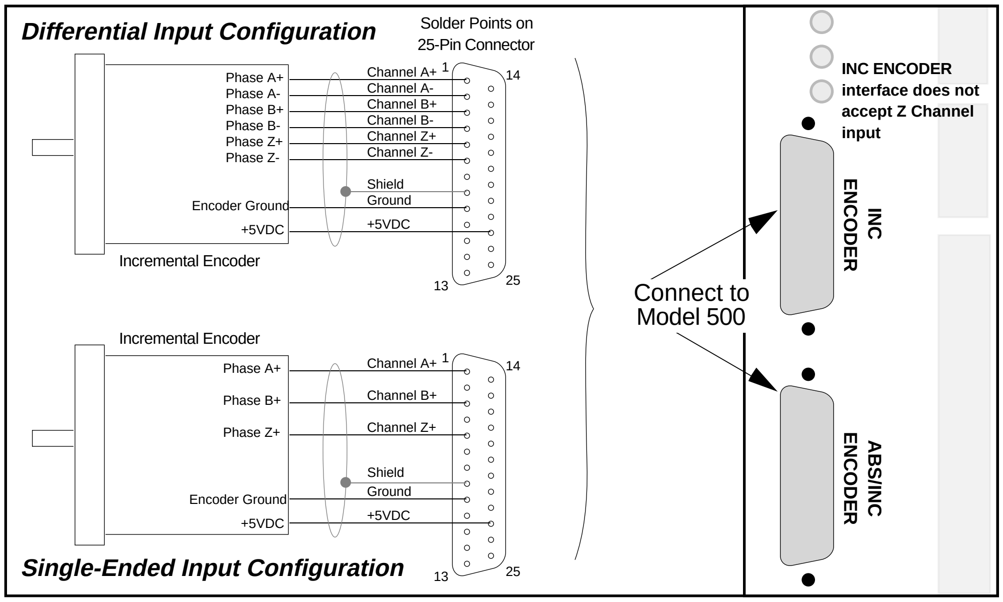


Figure 3-11. Differential and Single-Ended Incremental Encoder Connections

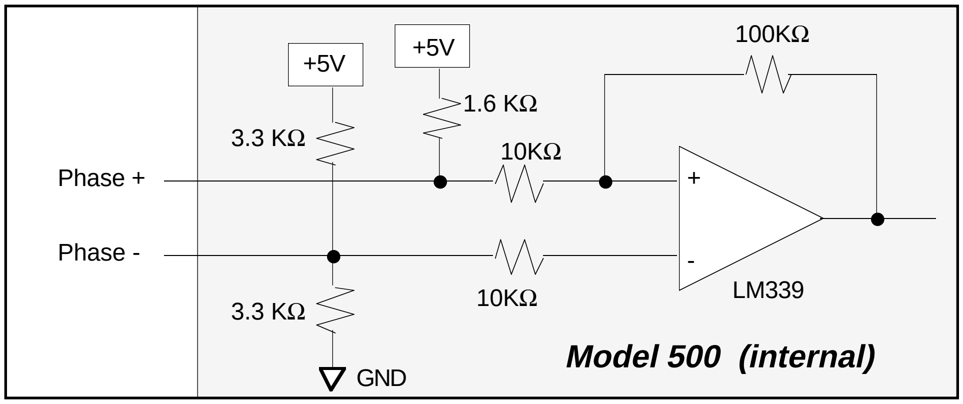


Figure 3-12. Typical Model 500 Encoder Input Circuit

For procedures to verify that your encoder interface is correctly wired, refer to the *Verifying Installation* section later in this chapter.

Absolute Encoders

The Model 500 allows you to use absolute encoders as position sensors. Compumotor offers three absolute encoders compatible with the Model 500 — one rotary (AR-C) and one linear (AL-C). Each absolute encoder uses a decoder box to translate the encoder input for the indexer. The decoder box has a cable with a 25-Pin D connector that plugs directly into the **ABS/INC ENCODER** interface.

The encoder inputs are optically isolated. With absolute encoders, the maximum speed achievable is 4,800 rpm. The absolute encoder differs from the incremental encoder in that each location in the travel of the absolute encoder is a physically distinct location giving the same position reading every time the encoder reaches that location. With an absolute encoder, a home limit is not needed if you specify a specific location as the *home reference position*.

Absolute Encoder Connections

If you are using the AR-C or AL-C, refer first to the encoder's user guide to configure the decoder box for operation with the Model 500 and familiarize yourself with its operation. After you have done this, you can connect the decoder box cable to the **ABS/INC ENCODER** interface as shown in Figure 3-13.

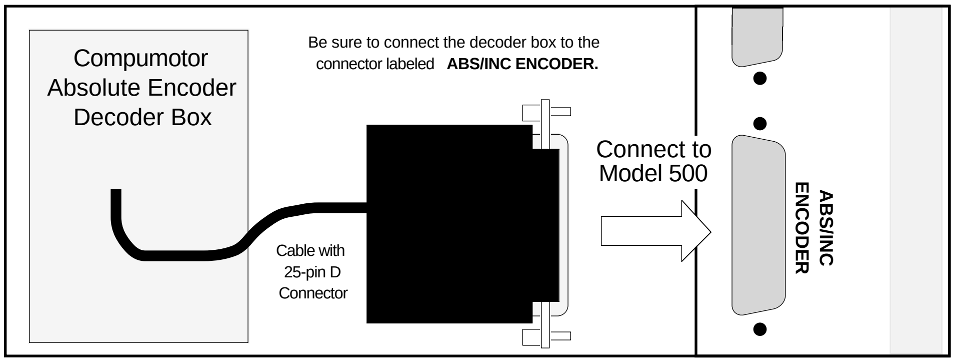

Compumotor Absolute Encoder Decoder Box
Be sure to connect the decoder box to the connector labeled ABS/INC ENCODER.
Connect to Model 500
Cable with 25-pin D Connector
ABS/INC ENCODER

Figure 3-13. Compumotor Absolute Encoder (Decoder Box) Connections

Table 3-4 lists the **ABS/INC ENCODER** interface pin functions. To create a custom cable for use with the absolute encoder, you can solder the wires to one of the two 25-pin D-connectors provided in the Model 500 ship kit.

Pin #	INput or OUTput	Description	Pin #	INput or OUTput	Description
1	IN	A+	13	IN	Data Valid
2	IN	A-	14	OUT	Ground
3	IN	B+	15	IN	DØ (LSB)
4	IN	B-	16	IN	D1
5	IN	Z+	17	IN	D2
6	IN	Z-	18	IN	D3
7	—	No Connection	19	IN	D4
8	IN	Shield	20	IN	D5
9	OUT	Ground	21	IN	D6
10	OUT	Strobe	22	IN	D7 (MSB)
11	OUT	Select Ø	23	OUT	+5VDC
12	OUT	Select 1	24 & 25	—	No Connection

Table 3-4. **ABS/INC ENCODER** Interface Pin Outs

Installation Verification

This section provides procedures to verify that you successfully connected the system components and I/O. Procedures for testing the Model 500-FOL's following functions and associated encoder(s) are provided in Chapter 5, *Following*.

CAUTION

For safety purposes, the procedures in this section are written under the assumption that you have <u>not</u> coupled the motor to the load or machine. Couple the load <u>after</u> you perform these procedures.

This section provides instructions to test proper installation of a single-axis Model 500 system. If you have a multi-axis (daisy-chained) system, you should test every axis. To test every axis, you must repeat every device-specific command for each axis. All universal commands will be performed by every Model 500 unit in the chain. Status reports for each axis should yield the same response. For example, the following is a series of universal commands which will be performed by every axis in a daisy-chain:

<u>Command</u>	<u>Description</u>
`> MN`	Sets mode to normal (all axes)
`> A1Ø`	Sets acceleration to 10 rps^2
`> V2`	Sets velocity to 2 rps
`> PZ`	Sets absolute position counter to zero
`> D25ØØØ`	Sets distance to 25000 steps
`> G`	Executes the move (Go)

Each motor on every axis will perform a 25,000-step move simultaneously.

To report the absolute position on all axes, you will have to issue a **PR** command (a device-specific command) with the appropriate device address as follows:

<u>Command</u>	<u>Description</u>
`> 1PR`	Reports absolute position for unit 1 (response is **`*ØØØØØ25ØØØ`**)
`> 2PR`	Reports absolute position for unit 2 (response is **`*ØØØØØ25ØØØ`**)
`> 3PR`	Reports absolute position for unit 3 (response is **`*ØØØØØ25ØØØ`**)

Verify CW & CCW Limit Switches

Before you test the limit switches, check the following:

- ❑ Ensure that the CW and CCW limit switches are wired properly.
- ❑ **Make sure that the load is not attached to the motor.**
- ❑ Make sure that you can manually open and close the limit switches.

To test the CW and CCW Limit switches with the Model 500 controller, complete the following test procedure:

① Close the CW and CCW limit switches.

② Type **`1IS`** [cr]. Assuming that you have not grounded any other inputs or issued an **`INL1`** command, you should receive the following response: **`*11ØØ_ØØØØ_ØØØØ_ØØØØ`**. This means that the CCW and CW limits (represented by the 1st and 2nd digits, respectively) are closed.

③ To test the CW limit switch, enter the following commands:

<u>Command</u>	<u>Description</u>
`> LDØ`	Enables the CW and CCW Limits
`> INLØ`	Normally-closed limit inputs
`> MC`	Sets controller to Continuous mode
`> A1ØØ`	Sets acceleration to 100 rps^2
`> AD1ØØ`	Sets deceleration to 100 rps^2
`> V3`	Sets velocity to 3 rps
`> H+`	Changes the direction of the motor (CW)
`> G`	Executes the move (Go)

The motor should move in the CW direction at a constant velocity.

④ While the motor is moving, open the CW limit switch. The motor should stop moving and error code **41** should be flashing in the two-digit LED. This verifies that the CW limit switch is working properly.

⑤ Close the CW limit switch. (Error code **41** will continue to flash until the motor is commanded to move in the opposite direction, or until you issue the **`ON`** command.)

⑥ To test the CCW limit, enter the following commands:

<u>Command</u>	<u>Description</u>
`> H-`	Changes the motor's direction (CCW)
`> G`	Executes the move (Go)

The motor should move in the CCW direction at a constant velocity.

⑦ While the motor is moving, open the CCW limit switch. The motor should stop moving and error code **42** should be flashing in the two-digit LED. This verifies that the CCW limit switch is working properly.

⑧ Close the CW limit switch. (Error code **42** will continue to flash until the motor is commanded to move in the opposite direction, or until you issue the **`ON`** command.)

If you are not able to stop the motor with the limit switches, issue the Stop (**`S`**) command, reverse the wiring of the CW and CCW limit inputs, and start over at step 1 above. If you are still unsuccessful, refer to Chapter 6, *Maintenance and Troubleshooting.*

Verify Homing Function

In this section you will test your home limit switch and home the motor. You can initiate the Go Home function by entering the Go Home (**GH**) command over the RS-232C interface or by enabling a Go Home input using the Configure Input (**IN**) command. You must also define the Go Home Velocity (**GHV**), Go Home Acceleration (**GHA**), Go Home Deceleration (**GHAD**) and Final Go Home Velocity (**GHF**) to properly initiate the Go Home function. You can define the initial direction of the homing function with the Go Home Velocity (**GHV**) command. The **OS** commands are used to set up the final go home approach and the edge of the switch to stop on.

When you command the motor to go home, it begins to move in the direction and at the velocity you specified. It performs this move at the acceleration rate specified with the **GHA** command, and looks for the Home limit input to open if in the **INLØ** mode (close if in the **INL1** mode). If the motor encounters an end-of-travel limit while it searches for home, it will reverse direction and look for the Home limit input to go active in the opposite direction. If the motor encounters the other limit before it detects the home signal, the Go Home move will be aborted and the motor will stop.

Use the following procedure to test the functionality of the Home limit switch (this is a *back up to home switch* homing example):

① Manually open the Home limit switch and type **1IS.** Assuming your end-of-limits are closed and all other inputs are open, the system should respond with ***11ØØ_ØØØØ_ØØØØ_ØØØØ**.

② Close the Home limit switch and type **1IS.** The system should respond with ***111Ø_ØØØØ_ØØØØ_ØØØØ**. This verifies that the home limit switch (represented by the 3rd digit) is functioning properly.

③ Open the Home limit switch.

④ To test the Model 500's homing function, enter the following string of commands:

Command	Description
> **1GHA1ØØ**	Sets go home acceleration to 100 rps^2
> **1GHAD1ØØ**	Sets go home deceleration to 100 rps^2
> **1GHF.5**	Sets final go home speed to 0.5 rps
> **GHV+1**	Sets the go home velocity to 1 rps in the clockwise (CW) direction
> **1OSB1**	Enables the back up to home switch function
> **1OSGØ**	The final approach direction is CW
> **1OSHØ**	The CW edge is the edge the move stops on
> **GH**	Executes the Go Home function

The motor moves in the CW direction at constant velocity of 1 rps.

⑤ Close the Home limit switch. The motor decelerates to a stop. It then reverses direction and moves off the switch at the final approach speed so that it can make its final approach to the home switch in the CW direction.

⑥ Open the switch to simulate moving back off the home switch. Once it is off the switch it begins its final approach in the CW direction.

⑦ Close the Home limit switch again. The Model 500 will continue moving because it has encountered only the CCW edge of the limit switch.

⑧ Open the Home limit switch again. This will be the CW edge and the motor will stop. The Model 500 automatically considers this position as the zero position.

Chapter 4, *Application Design*, explains the back-up to home limit function in detail. You will use the **OS** commands to set up the exact homing sequence that your application needs.

Verify Incremental Encoder Interface

Follow the steps below to test the incremental encoder interface.

① Using the 25-pin D connector that you wired, or the one that came with your Compumotor incremental encoder, connect the encoder to the **ABS/INC ENCODER** input on the front panel of the Model 500.

② Set the **ABS/INC ENCODER** interface to function as a incremental position feedback interface (as opposed to a *following* interface).

Command	Description
`> 1FSMØ`	Sets the **ABS/INC ENCODER** interface for incremental feedback
`> 1FSJØ`	Sets the **ABS/INC ENCODER** interface for position feedback

③ Turn the encoder shaft CW and display the encoder position using the `1DPA` command. The `1DPA` command transmits the actual encoder position to the screen every 10 ms. **If your terminal or computer can not display the position at this rate, use the `1DPA1` command; this transmits the response only once.**

Command	Description
`> 1DPA`	Periodically display encoder position

As you turn the encoder CW, the position count displayed by the command will increase. Turning the encoder CCW decreases the count. If it decreases while turning CW, it is possible that you have the phase A and phase B channels of the encoder reversed in their connection to the Model 500. Reverse the phase A and phase B channels and repeat steps 1-3. To discontinue the position display press the space bar or the carriage return.

④ Remove the 25-pin D connector from the **ABS/INC ENCODER** interface. Attach it to the **INC ENCODER** interface.

⑤ Set the **INC ENCODER** interface to function as a position feedback interface.

Command	Description
`> 1FSJ1`	Sets the **INC ENCODER** interface for position feedback

⑥ Repeat step number ③.

This test verifies that both the **INC ENCODER** and the **ABS/INC ENCODER** interfaces are working properly and that your 25-pin D incremental encoder connector is wired properly.

Verifying Absolute Encoder Interface

Perform the following steps to verify the functionality of your absolute encoder interface. It is assumed at this point that you have read the absolute encoder user guide and are familiar with it's operation.

① Remove power from the motor/drive, Model 500, and absolute encoder/decoder.

② Attach the encoder cable to the decoder box and to the Model 500. Ensure that the encoder is plugged into the decoder box as well. Apply power to the decoder box.

③ Type in the following commands to select the absolute encoder and the **ABS/INC ENCODER** interface.

Command	Description
`> 1FSM1`	Selects the absolute encoder
`> 1FSJØ`	Selects the **ABS/INC ENCODER** interface for positioning

④ Turn (move) the encoder and display the encoder position using the **1DPA** command. The **1DPA** command transmits the actual encoder position to the screen every 10 ms. **If your terminal or computer can not display the position at this rate, use the 1DPA1 command; this transmits the response only once.**

Command	Description
> **1DPA**	Periodically display encoder position

As you turn the encoder, the position count displayed by the command will increase CW and decrease CCW (with rotary encoders).

This test verifies that the **ABS/INC ENCODER** interface is working properly and that your 25-pin D encoder connector is wired properly.

Verify Input Wiring

Perform the following steps to verify proper wiring of your inputs.

① Place the inputs in the active low state and display their current state with the **IS** command.

Command	Description
> **INLØ**	Active level low
> **1IS**	Response is ***ØØØØ_ØØØØ_ØØØØ_ØØØØ**

② Activate input number 1 (**IN1**) by shorting the input to ground, and enter the status request (**IS**) command again.

Command	Description
> **1IS**	Response is ***ØØØ1_ØØØØ_ØØØØ_ØØØØ**

This indicates that the Model 500 has recognized that input number 1 has changed state. In this manner you can verify that each input is changing state correctly.

Verify Output Wiring

You can directly control each output by using the **O** command. Use the **O** command to turn on outputs 1 and 2 and turn off output 3. Use a voltmeter to verify that the output has changed state.

Command	Description
> **O11Ø**	Directly controls the outputs

Chapter 4. Application Design

Chapter Objectives

The information in this chapter will enable you to:

- ❏ Recognize and understand important considerations that must be addressed before you implement your application
- ❏ Understand the capabilities of the system
- ❏ Use examples to help you develop your application

Application Considerations

When designing and developing your system, you should consider and evaluate the following factors:

- ❏ Motion Profiles
- ❏ Move Times (calculated vs. actual)
- ❏ Backlash
- ❏ Preset vs. Continuous Moves
- ❏ Interactive vs. Non-interactive Modes

Motion Profiles

In any motion control application, an important requirement is precise position, whether it be with respect to time or velocity. A motion profile represents the velocity of the motor during a period of time in which the motor changes position. The type of motion profile that you need depends upon the motion control requirement that you specify. The basic types of motion profiles are described below. The 500 can perform all of the profiles discussed in this chapter.

Triangular and Trapezoidal Profiles

For constant acceleration control systems, velocity, acceleration, and distance parameters are defined before the system can execute a preset move. The value of these parameters determines the type of motion profile as either triangular or trapezoidal.

A triangular profile results when the velocity and acceleration are set so that the motor does not attain the defined velocity before the motor travels half of the distance that you specify. This results from either a relatively low acceleration, a relatively high velocity, or both. The motion profile for this move is shown in Figure 4-1.

A trapezoidal profile (see Figure 4-1) results when the motor reaches the defined velocity before the motor moves half of the specified distance. A trapezoidal profile may occur if you specify a low velocity with a high acceleration or a long distance.

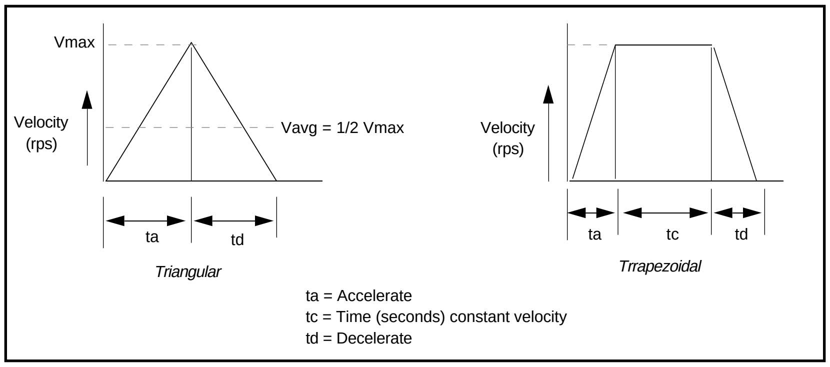


Figure 4-1. Triangular and Trapezoidal Profile

Separate Acceleration and Deceleration

The 500 has the capability of performing a move profile with different acceleration ramp and deceleration ramp values. This allows for more complex motion profiles. A motor can accelerate very quickly and decelerate very slowly as shown in Figure 4-2.

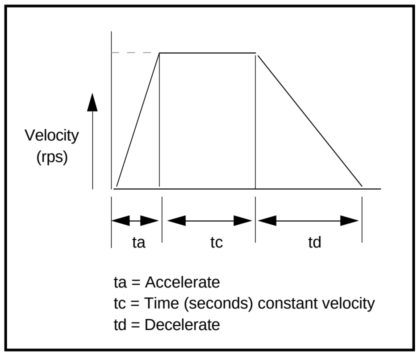


Figure 4-2. Move Profile With Separate Accel and Decel

The commands necessary for executing any of the motion profiles shown in Figures 4-1 or 4-2 are shown below.

Command	Description
> **An**	**n** specifies the acceleration in rps^2
> **ADn**	**n** specifies the deceleration in rps^2
> **Vn**	**n** specifies the velocity in rps for a move profile
> **Dn**	**n** specifies the distance to move in steps.
> **G**	The **G** command begins motion at the specified acceleration, deceleration,velocity and distance.

Complex Profiles

A complex profile is a profile in which the velocity is changed *on-the-fly* based on distance. You can use the Motion Profiling mode with the 500 and the 500-FOL (*following* version) to perform complex profiles. The Motion Profiling mode allows you to execute buffered commands immediately, while the motor is in motion. Buffered commands are explained in the **Model 500 Software Reference Guide**. These commands can also be executed at any distance point during a move. This allows you to change velocity *on-the-fly* (without stopping), turn on outputs, perform time delays, variable operations, and more, all based on a distance during the move profile. Figure 4-3 is a motion profile where velocity is changed on-the-fly using the Profiling mode.

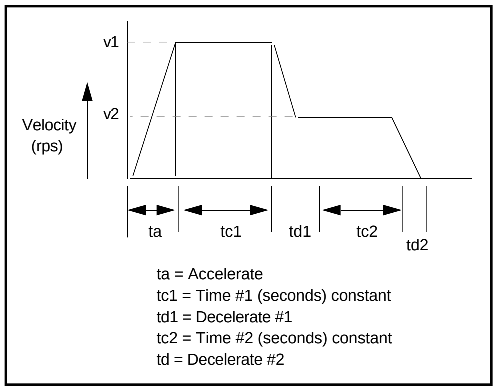


Figure 4-3. Motion Profiling Mode

To program in the motion profiling mode, the following commands are needed:

Command	Description
> **MPP**	This command enters the Profiling mode
> **NG**	This command ends the Profiling mode
> **DP**	This command marks a distance point in a profiling move at which some command should be processed (i.e., velocity changes)
> **FP**	*500-FOL only.* This marks a distance on the encoder input that the 500-FOL is following

Move Times—Calculated vs Actual

You can calculate the time it takes to complete a move by using the acceleration, velocity, and distance values that you define. However, you should not assume that this value is the actual move time. There is calculation delay and motor settling time that makes your move longer. You should also expect some time for the motor to settle into position. The 500 has minimal calculation-delay time associated with a Go (**G**) command. This delay can be as low as 500 µs. The 500 has an internal timer that allows you to monitor the elapsed time of your move. The response of the **TM** command shows you the previous move's execution time.

Predefined Goes

For the fastest possible calculation move times, predefined *goes* can be used. The **GDEF** command is used to execute predefined goes. The predefined goes are faster because any calculations are performed ahead of time and stored in memory. Refer to the **GDEF** command to execute a predefined go. Perform the following commands to make a predefined go. You can have up to 16 predefined moves.

Command	Description
> **GDEF1,A1ØØ,AD1ØØ,V4,D5ØØØØ**	Defines predefined move #1
> **GD1**	Execute predefined move #1

Positional Accuracy vs. Repeatability

Some applications require high absolute accuracy. Others require repeatability. You should clearly define and distinguish these two concepts when you address the issue of system performance.

If the positioning system is taken to a fixed place and the coordinates of that point are recorded. The only concern is how well the system repeats when you command it to go back to the same point. For many systems, what is meant by accuracy is really repeatability. Repeatability measures how accurately you can repeat moves to the same position.

Accuracy, on the other hand, is the error in finding a random position. For example, suppose the job is to measure the size of an object. The size of the object is determined by moving the positioning system to a point on the object and using the move distance required to get there as the measurement value. In this situation, basic system accuracy is important. The system accuracy must be better than the tolerance on the measurement that is desired. Consult the technical data section of the Compumotor Catalog for more information on accuracy and repeatability.

Backlash

Backlash is simply the *gear slop* that occurs in the gearing when the system changes direction of motion.

The 500 has the capability to compensate for backlash. You can specify different compensations depending on the direction the motor is moving. You will use the **BL** command for backlash compensation. Refer to the ***Model 500 Software Reference Guide*** for a detailed description of the backlash command. The syntax of the command is as follows:

Command	Description
`> BLn,m`	Variable **n** is the amount of CCW steps that should be compensated. Variable **m** is the number of CW steps that will be compensated.

If a move is made in the CW direction, an extra **m** steps will be made to account for the backlash. The concept is that the load will not begin to move until the motor moves enough to make contact with the gearing. Figure 4-4 illustrates this. The same is true for the opposite direction. The backlash compensation may be different in either direction so you can program them independently. The load will not move until point A contacts point B. This is the backlash associated with changing direction. Assuming this distance to be 1,000 motor steps the Backlash command will be typed as follows:

Command	Description
`> BL1ØØØ,2ØØØ`	**1000** is the amount of CCW steps that should be compensated. **2000** is the number of CW steps that will be compensated.

For a 25,000-step move in the CCW direction, the motor will actually move 26,000 steps to remove the backlash but the position counter will only change by 25,000 steps. This is done because the load is expected to move whenever the motor moves. The load should move a certain distance when the motor moves a certain distance. If there is backlash in the system, the load will not move the correct distance when the motor is commanded to move in the opposite direction from its previous move. In this example, the motor was moving in the CW direction. The gearing was flush and the teeth were touching. When the direction changes, the teeth must move 1,000 steps before they are again in contact with the load gear. Thus for the load to move 25,000 steps the motor will have to move 1,000 steps until the teeth are in contact then move 25,000 steps so that the load will actually move 25,000 steps. This mode allows you to compensate for the error in system between the motor and the actual load position. *The compensation only takes place when you change direction.*

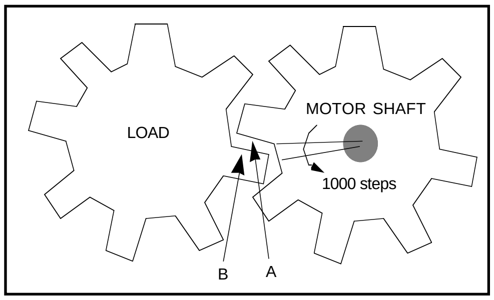


Figure 4-4. Backlash Compensation

Preset & Continuous Moves

This section describes preset or *normal* mode moves (incremental and absolute) and continuous mode moves.

Preset Mode Moves

A preset move is a point-to-point move of a specified distance. You can select preset moves by putting the 500 into normal mode with the Mode Normal (**MN**) command. Preset moves allow you to position the motor in relation to the motor's previous stopped position (incremental moves) or in relation to a defined zero reference position (absolute moves). You can select incremental moves by with the Mode Position Incremental (**MPI**) command. You can select absolute moves with the Mode Position Absolute (**MPA**) command. At any time, you can request the state in which the 500 is configured by issuing the **DR** command.

INCREMENTAL MOVES

When you are in Incremental mode (**MPI**), a preset move moves the motor the specified distance from its starting position. You can specify the direction with the optional sign (**D+8ØØØ** or **D-8ØØØ**), or you can define it separately with the Change Direction (**H+** or **H-**) command .

Command	Description
> **LD3**	Disables CW and CCW limits (***not needed if limits are installed***)
> **MPI**	Sets unit to Incremental Position Mode
> **MN**	Places the 500 in the preset mode
> **PZ**	Zeroes the position counter
> **A25**	Sets acceleration to 25 rps^2
> **AD25**	Sets deceleration to 25 rps^2
> **V5**	Sets velocity to 5 rps
> **D8ØØØ**	Sets distance to 8,000 steps
> **G**	Executes the move (Go)
> **D12ØØØ**	Sets distance to 12,000 steps
> **G**	Initiates motion
> **1PR**	Reports the setpoint (commanded) position Response: **2ØØØØ**

ABSOLUTE MOVES

A preset move in the Absolute mode (**MPA**) moves the motor to the distance in an absolute coordinate system that you specify relative to an absolute zero position. You can set the absolute position to zero with the Position Zero (**PZ**) command or by cycling the power to the indexer. The absolute zero position is initially the power-up position.

The direction of an absolute preset move depends upon the motor position at the beginning of the move and the position you command it to move to. If the motor is at absolute position +12,800, and you instruct the motor to move to position +5,000, the motor will move in the negative direction a distance of 7,800 steps to reach the absolute position of +5,000.

The 500 powers up in Incremental mode. When you issue the Mode Position Absolute (**MPA**) command, it sets the mode to absolute. When you issue the Mode Position incremental (**MPI**) command the unit switches to Incremental mode. The 500 retains the absolute position, even while the unit is in the Incremental mode. You can use the Position Report (**PR**) command to read the absolute position.

In the following example, the motor performs the same commands as the incremental position example. In this case, the **PR** command will report a different position because it is working in an absolute coordinate system.

Command	Description
`> LD3`	Disables the CW and CCW Limits (*not needed if limits are installed*)
`> MPA`	Sets unit to Incremental Position Mode
`> PZ`	Zeroes the position counter
`> A25`	Sets acceleration to 25 rps^2
`> AD25`	Sets deceleration to 25 rps^2
`> V5`	Sets velocity to 5 rps
`> D8ØØØ`	Sets distance to 8,000 steps
`> G`	Executes the move (Go)
`> D12ØØØ`	Sets distance to 12,000 steps
`> G`	Initiates motion
`> 1PR`	Reports the setpoint(commanded) position

The motor will move to absolute position 8,000. The second move is 4,000 more steps to the absolute position of 12,000 steps. The **PR** command reports a setpoint (commanded position) of 12,000 steps.

Continuous Mode Moves

The Continuous mode (**MC**) command accelerates the motor to the velocity that you last specified with the Velocity (**V**) command. The motor continues to move at the specified velocity until you issue the Stop (**S**) command or specify a velocity change. To change velocity while the motor is moving, use the instantaneous velocity command (**IV**). Another way of changing velocity while moving is to enter Motion Profiling mode (**MPP**).

In Motion Profiling mode, all buffered commands are executed immediately—therefore you only have to enter the **V** command to change the velocity. No **G** is needed following the **V**. The Continuous mode is useful for applications that require constant movement of the load, and the motion is not based on distance but is based on internal variables or external inputs, or when the motor must be synchronized to external events such as trigger input signals. In this example, velocity is changed based on external inputs.

Command	Description
`> PS`	Pauses motion until a **C** command is reached
`> MPP`	Places the 500 in the **MPP** mode
`> IN1A`	Sets up **IN1** (Input 1) as trigger bit 1
`> IN2A`	Sets up **IN2** (Input 2) as trigger bit 2
`> LD3`	Disables CW and CCW limits (***not necessary if limits are installed***)
`> MC`	Sets unit to the Continuous mode
`> A25`	Sets acceleration to 25 rps^2
`> AD25`	Sets acceleration to 25 rps^2
`> V1`	Sets velocity to 1 rps
`> G`	Executes the move (Go)
`> T1`	Waits 1 sec after the motor reaches constant velocity
`> V5`	Sets velocity to 5 rps
`> TR1Ø`	Waits for trigger bit 1 to go on and bit 2 to go off
`> STOP`	Stops the motor
`> NG`	Ends the profiling mode
`> C`	Continues execution of commands

These commands cause the 500 to run in Continuous mode. The motor reaches 1 rps , waits for 1 second, changes velocity to 5 rps, waits for you to turn **IN1** (Input 1) on and turn **IN2** (Input 2) off, and then stops. *The **VØ** and **STOP** commands stop the motor (the **S** command is not a buffered command and cannot be used in this situation, unless you wish to halt the operation in the middle of the program).* The **DIN** command (an immediate command) simulates the state you want the inputs to be in. In the example above, you could simulate the activation of the trigger state without physically toggling the inputs, by using the **DIN** command as follows:

Command	Description
`> DINEEE1ØEEEEE`	**E** means *do not affect the input*. A **1** makes the input one, a **Ø** makes the input zero. Each **E**,1, or 0 represents an input bit. There are 10 inputs. The **1** and **Ø** in this example correspond to **IN1** and **IN2** on the front panel. The first 3 **E**s correspond to the CCW, CW and Home limits.

Interactive vs. Non-Interactive Modes

You can operate the 500 in either the *interactive* or the *non-interactive* mode. The 500 is shipped factory-set to the interactive mode. The Disable Interactive Mode (**SSIØ**) command disables this mode, while the Enable Interactive Mode (**SSI1**) command enables it.

In the interactive mode (default mode), the indexer responds with a prompt (**>**) when it receives a valid command and a question mark (**?**) when it does not receive a valid command. These interactive responses are preceded with a carriage return and a line feed. While defining loops or sequences, no responses are received until the loop or sequence is executed or until the **XT** command is executed.

Registration

Registration with the Model 500 Indexer provides the ability to change the move profile which is being executed to an unrelated move profile defined as a registration move. This unrelated registration move is executed when an input to the Model 500 transitions from a high-level to a low-level or from a low-level to a high-level. The Model 500 has 4 inputs that you can define as registration inputs. The inputs will interrupt the Model 500 microprocessor at the highest priority level in the system. The registration input and the motor's current position will be captured within 50 µs. The registration profile will be executed at the next update period (refer to Figure 4-5). The registration move profile uses the actual position captured (within 50 µs) as its *zero* reference point.

The interrupt is *edge-sensitive* (if you have a bouncy switch for the registration input, you must use the debounce command to ensure that false registration interrupts do not occur). With the Model 500, a registration interrupt can interrupt another registration move. You can make a preset move, then issue a registration move and interrupt that registration move with the same or another registration move. In this manner, you can toggle between two velocities based on registration inputs. You can interrupt a registration move an unlimited number of times.

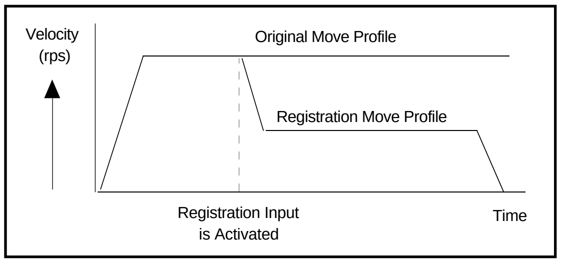
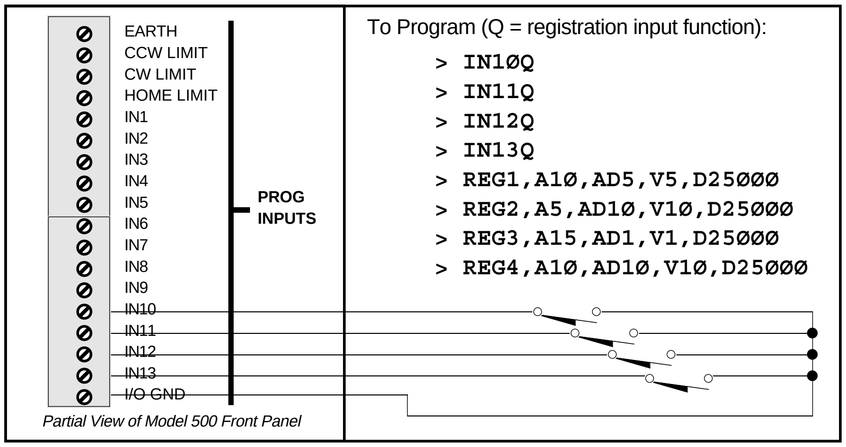


Figure 4-5. Registration Move

The Model 500 has 13 programmable inputs labeled **IN1** through **IN13**. Only the last four inputs (**IN10** - **IN13**) can be defined as registration inputs. Any combination of these four inputs may be defined as a registration input. *An input defined as a registration input* ***cannot*** *be toggled at a frequency higher than 1 kHz, or once every millisecond.*

REG1 defines the move when input **IN10** is configured as a registration input.
REG2 defines the move when input **IN11** is configured as a registration input.
REG3 defines the move when input **IN12** is configured as a registration input.
REG4 defines the move when input **IN13** is configured as a registration input.

Registration with the Model 500 is illustrated in Figure 4-6.



```
> IN1ØQ
> IN11Q
> IN12Q
> IN13Q
> REG1,A1Ø,AD5,V5,D25ØØØ
> REG2,A5,AD1Ø,V1Ø,D25ØØØ
> REG3,A15,AD1,V1,D25ØØØ
> REG4,A1Ø,AD1Ø,V1Ø,D25ØØØ
```

Figure 4-6. Registration With the Model 500

Figure 4-7 illustrates the multiple registration move capability of the Model 500.

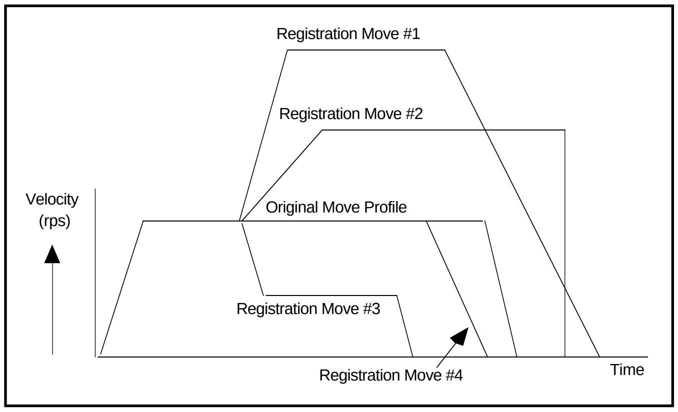


Figure 4-7. Model 500 Registration Profiles

The syntax for defining a registration move using input **IN10** on the front panel is: `> REG1,A1Ø,AD1Ø,V1Ø,D25ØØØ`.

The registration move REG1 will be performed when the REG1 input, #10 is activated. The acceleration (**A1Ø**) will be 10 rps^2, the deceleration will be 10 rps^2, the velocity will be 10 rps, and the distance traveled will be 25,000 steps.

Bouncy Registration Inputs

The switches may be bouncy or noisy and may take a few milliseconds to settle. Since the registration inputs to the Model 500 are interrupts, they are *edge sensitive*. Thus, with a bouncy switch, each edge appears like a registration interrupt and the registration move is made from the distance position at which the latest edge was detected. You can use a software command to debounce the inputs. This allows you to ignore any bouncing transitions from your switch after the initial registration interrupt occurred. The time in which the interrupts or false edges are ignored is determined by the number you enter for the **TDR** command. The **TDR** command is the debounce time (ms) that you specify so that the inputs cannot cause another registration interrupt until the switch is settled.

For example, an application needs two different registration moves performed depending on the input that is toggled. The motor has a resolution of 25,000 steps per revolution. One move must turn the motor 1 revolution at 10 rps, while the second is a 1/4 revolution move at 1 rps. If both occur at the same time, the higher speed move must take place. If an input does not occur, the move will be a 500,000-step move. The Model 500 is configured as follows:

Command	Description
`> 1IN1ØQ`	Defines IN10 as a registration input
`> 1IN13Q`	Defines IN13 as a registration input
`> REG1,A1Ø,AD1Ø,V1Ø,D25ØØØ`	Defines the IN10 registration move
`> REG4,A1Ø,AD1Ø,V1,D625Ø`	Defines the IN13 registration move
`> D5ØØØØØ`	Sets distance to 500,000 steps
`> V5`	Sets velocity to 5 rps
`> G`	The preset move is initiated

If **IN10** or **IN13** is toggled, the corresponding registration move is performed. The **DIN** command cannot activate a registration input because the registration inputs are hardware oriented.

Jogging the Motor

In some applications, you may want to move the motor manually. You can configure the 500 to allow you to move the motor manually with the Configure Input (**IN**) command. You must define the jogging velocity with the Jog Velocity High (**JVH**) and Jog Velocity Low (**JVL**) commands. You can define three different inputs for jogging: CW Jog input (**IN#J**), CCW Jog Input (**IN#K**), and Jog Speed Select High/Low (**IN#L**). You must also enable the jogging feature with the **OSE1** command. Once you set up these parameters, you can attach a switch to the jog inputs that you defined and perform jogging (**#** represents digits 1 - 13, which you enter). The following example shows how you can define power-up Sequence 100 to set up jogging.

STEP ① Define a power-up sequence.

Command	Description
`> XE1ØØ`	Erase sequence #100
`> XD1ØØ`	Define sequence #100
`> LD3`	Disables the limits (*not needed if you have limit switches installed*)
`> JA25`	Set jog acceleration to 25 rps^2
`> JAD25`	Set jog deceleration to 25 rps^2
`> OSE1`	Enables Jog function
`> JVL.5`	Sets low-speed jog velocity to 0.5 rps
`> JVH5`	Sets high-speed jog velocity to 5 rps
`> IN1J`	Sets **IN1** as a CW jog input
`> IN2K`	Sets **IN2** as a CCW jog input
`> IN3L`	Sets **IN3** as a speed-select input
`> XT`	Ends sequence definition

STEP ② Reset the 500.

Command	Description
`> Z`	Resets the 500

STEP ③ Turn on the **IN1** input to move the motor in the CW direction at 0.1 rps (until you turn off **IN1**).

STEP ④ Turn on the **IN2** input to move the motor in the CCW direction at 0.1 rps (until you turn off **IN2**).

STEP ⑤ Turn on the **IN3** input to switch to high-speed jogging.

STEP ⑥ Repeat steps 3 and 4 to perform high-speed jogging.

Defining a Home Location

The 500's go home function brings your motor to a home reference position. The homing function allows you to back up to a home switch and come to a stop on a specific edge of the switch. You can program the active level of the switch. The homing function also allows you to home to the Z channel of an incremental encoder (*only if using an incremental encoder on the* **ABS/INC ENCODER** *interface*).

Homing to a Switch

The 500's home position is located where the edge (selected with the **OSH** command) of the Home Limit input occurs (i.e., the 500 recognizes the home position as the position where the home limit signal makes a transition from on to off, or from off to on, depending on the selected edge and the initial direction of the go home move).

Once it recognizes the selected edge, the motor decelerates to a stop. After stopping, the 500 positions the servo motor away from the selected edge of home limit signal in the opposite direction of the final approach direction of the go home move. After coming to a stop a second time, the 500 creeps the motor towards the selected edge at Go Home Final Velocity (**GHF**) until the home limit input becomes active again or the home limit and the 500's resolver Z Channel pulse are active.

You must ensure that the final approach starts from the opposite side of the signal from the selected edge. If the final approach direction is positive , the final move must start from the negative side of the selected edge. If there is significant backlash and friction in the system, and the indexer is instructed to go home in the CW direction, the motor may end up on the wrong side of the signal and execute its final approach in the wrong direction.

This problem can also occur if the motor's go home speed is high, and the Home limit signal is delayed, (by a relay or PLC for example). In such a situation, you should initiate homing operations from the opposite side of the selected edge of home. When you conclude the homing operation, the indexer resets its internal position counter.

You can use the home limit input in conjunction with the resolver's internal Z Channel input to select a final home position. To enable Z Channel homing, you must activate the **OSD1** command. In this situation, a load-activated switch connected to the Home Limit input locates the general home position area, and the 500's internal resolver roll-over position is used for final Home positioning. ***The internal Z Channel pulse and the Home Enable input must both be active to mark the home position (refer to Figure 4-8).***

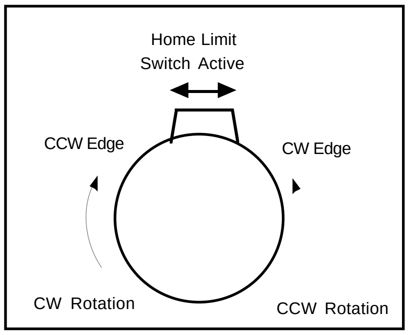


Figure 4-8. Go Home Description

Go Home

Under interface control, the Go Home (**GH**) command has the form **GH**<direction><velocity>. This indicates which direction to move, and at what velocity. For example, the command **GH-2** sends the motor in the negative direction at 2 rps in search of the home signal. The go home velocity can also be set by the **GHV** command. The last specified value will be used for the go home.

If you use an input to implement a go home move, the velocity provided by the **GHV** command is used to execute the homing function. The acceleration and deceleration parameters for this move are the home value for acceleration (**GHA**) and deceleration (**GHAD**).

If an end-of-travel limit is activated before home is found, the indexer reverses direction and attempts to find the home position again. If the other end-of-travel limit is activated before the indexer finds home, the indexer will stop trying to go Home. The homing operation will also be aborted (motor will stop) if the encoder's Z channel pulse is not received within 1 rev after the home switch is activated. Using the **OSJ1** command, you can make the indexer search for the Z channel pulse indefinitely.

The indexer can indicate whether or not the homing process was successful by responding to the Request Indexer Status (**R**) and Go Home Status (**RG**) commands.

Go Home Example

Command	Description
> **LD3**	Disables positive and negative limit inputs
> **OSB1**	Back up to home limit
> **OSG1**	The final approach direction is CCW
> **OSH1**	The reference edge is the CCW edge
> **GHA1**	Sets go home acceleration to 1 rps^2
> **GHAD1**	Sets go home deceleration to 1 rps^2
> **GH.5**	Sets go home in the positive direction at 0.5 rps

Figure 4-9 illustrates the homing procedure for these set-up commands. The motor starts to move toward the home position in the CW direction. Upon encountering the home switch, the motor comes to a stop. It will then continue to move CW at the final go home speed until it is off of the home switch. It will then be in position to reverse direction and make its final go home approach in the CCW direction. It will encounter the home switch on the CW edge first and will continue to move until the switch is inactive which will be the CCW edge.

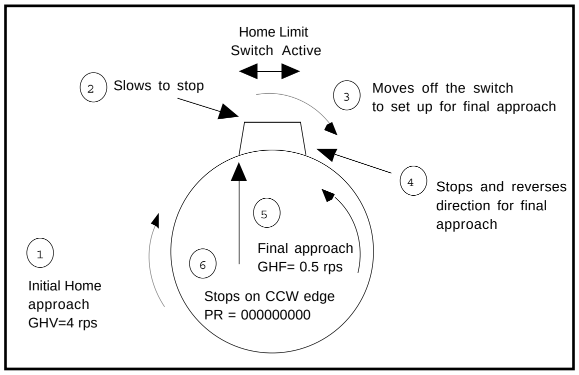

Home Limit
Switch Active
2 Slows to stop
3 Moves off the switch to set up for final approach
4 Stops and reverses direction for final approach
5 Final approach GHF= 0.5 rps
6 Stops on CCW edge PR = 000000000
1 Initial Home approach GHV=4 rps

Figure 4-9. Go Home Example

Closed-Loop Operation (Using an Encoder)

The Model 500 provides two encoder ports. Both of the encoder ports accept incremental encoder quadrature feedback. The encoder port labeled **ABS/INC ENCODER** accepts Compumotor's AR-C and AL-C absolute encoders. This section explains the different Model 500 features associated with using encoders. The features covered will be encoder step mode, position maintenance, and stall detection.

Motor Resolution and Encoder Resolution

The Model 500 provides step and direction motion data to a drive/motor system. For the Model 500 to command the proper distance it must have the drive/motor system resolution information. This information is provided to the Model 500 via the **MR** command. If you are using a Compumotor servo (e.g., the Compumotor Plus), you can set the resolution of the Compumotor Plus system with the Configure Motor Resolution(**CMR**) command. This tells the drive system how many pulses it must receive to move one revolution. This number of pulses per revolution is the number that must be provided to the Model 500 with the **MR** command. If the servo system is set to 5,000 steps/revolution, the Model 500 must be set to 5,000 steps/revolution using **MR**.

Command	Description
> **MR5ØØØ**	Sets the Model 500 to 5,000 motor steps/rev

If you use a stepper system, the number that you enter with the **MR** command is the motor resolution that defines the stepper motor/drive system. For example, the default with a Compumotor A Drive is 25,000 steps/revolution.

Encoder steps with the Model 500 are only needed when using the Encoder Step mode or Position Maintenance mode. In either case, the encoder resolution is set using the **ER** command. The value entered for **ER** is the number of encoder steps

Command	Description
> **ER4ØØØ**	Sets the Model 500 to 4,000 encoder steps/rev

Encoder Step Mode

In Encoder Step mode, all motion is in terms of the encoder resolution. The motor will not move if an encoder is not used. If the encoder has a resolution of 4,000 steps/rev and the encoder is mounted directly to the motor, a 4,000-step move would be required to make a one-revolution move. The motor will begin moving when a go (**G**) is issued and will continue to move until 4000 steps have come from the encoder. This mode is useful if the encoder is mounted to a load or through gearing and you want to position with respect to the encoder rather than the motor. The **FSB** command allows you to enter and exit Encoder Step mode. **FSB1** enables the mode. Once you enable Encoder Step mode, all moves and distances are made in the encoder's resolution rather than the motor's. The **PZ** command clears the commanded position counter and the actual position counter. The commanded position counter is reported by the **PR** command and the actual position counter is reported by the **PX** command. If you have an encoder attached, you can make moves in the motor step mode (**FSBØ**). The **PR** command will report the position counter in motor steps. The **PX** will report the actual encoder position in encoder counts.

Position Maintenance

The Model 500's Position Maintenance mode corrects for final positional accuracy after a move is completed. If a 25,000-step move is made, the motor will attempt to go 25,000 steps. After the move is complete, the system checks the encoder to ensure the encoder reads the correct number of encoder pulses. The 500 determines this from the **ER** command. If it is not correct, it will move the motor until the correct encoder count is read. The post move correction is determined from a set of information provided by the commands identified below. In addition, *report back* commands are provided to verify proper operation.

Command	Description
> **DPA**	**D**isplays **P**osition **A**ctual. This is the actual encoder indicating where the motor physically is. It is the encoder position report back and will be displayed in encoder steps.
> **DPS**	**D**isplays **P**osition **S**etpoint . This is the number of steps that the motor was commanded to go. It is indicated in motor steps if in the motor step mode and in encoder steps if in the encoder step mode.
> **DPE**	**D**isplays the **P**osition **E**rror between where the motor was commanded to go (setpoint) and where the encoder indicates the motor is(actual position). If you are in the Motor step mode then the error is displayed in motor steps. If you are in the encoder step mode then the error is displayed in encoder steps.
> **CPG**	**C**onfigure **P**osition **G**ain. This is the percentage gain that is applied to the position error to determine a correction velocity.
> **CPM**	This is the maximum gain value for which the **CPG** determines the % of to use for correction.
> **MV**	The maximum velocity that the Model 500 can command when correcting for position while in the position maintenance mode.
> **FSC**	Enters and exits Position Maintenance mode.
> **DW**	This is the Deadband Window which when the error is greater then this number of steps the Model 500 will perform position maintenance. This helps prevents dither.

In the Position Maintenance mode, the Model 500 performs a servo-like function. It corrects for any position error that is detected between the commanded position and the encoder position that is greater than the *deadband window*. The Model 500 has a 1-ms update rate. Every millisecond it will correct for any position error. It will continue to send pulses to the motor until the motor is in position according to the encoder.

The **DPE** report back command will display the error between the commanded motor position and the actual motor position. It reports in motor steps if you are in motor step (**FSBØ**) mode or in encoder steps if you are in encoder step mode (**FSB1**). With the Model 500 you can perform position maintenance in either the motor step mode or the encoder step mode. In the motor step mode the encoder counter is converted to motor steps using the **ER** command. In encoder step mode the commanded position setpoint is in terms of the encoder resolution. In a like manner, the deadband window (**DW** command) is entered in motor steps or encoder steps depending on the mode.

If **DPE**(position error) > **DW** (deadband window), position maintenance occurs. The correction velocity commanded to the motor is determined by the following equation:

Velocity = **CPG** (Proportional gain percentage) • **CPM** (Proportional gain maximum) • **DPE** (position error)

Since the 500 will typically be commanding a stepper system there is the chance that the correction velocity could be too much and the motor will stall. This is prevented by setting a maximum correction velocity with the **MV** command. If the velocity determined is less than the **MV** value (maximum correction velocity), then it will be the commanded correction velocity. If it is greater than the **MV** value, the **MV** value is the commanded velocity.

If **DPE** is < **DW**, no correction occurs.

Position Maintenance Example

Perform the following steps to make a move with position maintenance activated:

STEP ① Select the encoder interface you need for position maintenance.

Command	Description
> **FSJØ**	Selects **ABS/INC ENCODER** for position maintenance

STEP ② Type the following commands to enable the Position Maintenance mode:

Command	Description
> **1CPE2ØØØ**	Maximum position error is 2,000 motor steps
> **1CPG1Ø**	Sets position gain to 10% of maximum gain (CPM)
> **1CPM6Ø**	Maximum position gain is 60%
> **1MV1Ø**	Sets maximum correction velocity to 10 rps
> **1DW5**	Sets deadband window to 5 steps
> **1FSC1**	Invokes the position maintenance mode

Type the following commands to make a move:

Command	Description
> **A1ØØ**	Sets the acceleration to 100 rps^2
> **AD1ØØ**	Sets the deceleration to 100 rps^2
> **V1Ø**	Sets velocity to 10 rps
> **D1ØØØØØ**	The motor is set to move 100,000 steps
> **G**	Initiates motion

STEP ③ Use the following commands to observe the position error, setpoint, and actual position:

Command	Description
> **1DPA**	Display the actual position
> **1DPE**	Display the position error
> **1DPS**	Display the commanded position

The motor should move close to the commanded position of 100,000 steps.

This move was made in the motor step mode. You can type in **FSB1** and enable the encoder step mode. After doing this repeat the example.

Stall Detect

Another feature available on the Model 500 which requires an encoder is stall detect. The Stall Detect mode allows you to detect a motor stall or slippage. It is enabled with the **FSD1** command. With stall detect enabled the Model 500 will be comparing the position error between the commanded position and the actual position every sample period while in motion and while stopped. This can be done in either the encoder step mode or the motor step mode. If in the motor step mode the encoder counts are converted each sample period to the motor step resolution by the **ER** command and compared to the motor setpoint counter. A maximum allowable position error is set by the **CPE** command. If the error between the commanded position (**DPS**) and the actual position (**DPA**) is more than the value of the maximum allowable position error (**CPE**), the Model 500 will display fault error #**20** (maximum position error exceeded) on the LED display, and it will stop sending pulses to the motor. The unit must receive an **ON** or **STØ** command to continue.

> **CPE**	**C**onfigures the maximum **P**osition **E**rror that you will allow the actual motor position to be off from the setpoint or commanded position
> **FSD**	Enters and exits the **S**tall **D**etect mode.
> **1FSD1**	Invokes the stall detect mode

In the scenario shown in Figure 4-9, a motor with a resolution of 25,000 steps/rev and a directly coupled encoder of 4,000 steps/rev is commanded to move 25,000 steps. The **`ER`** command is set to 4000. The **`DW`** command is set to 100 steps. The maximum position error is 5,000 steps. The motor is commanded to move 25,000 motor steps. Three scenarios and the manner in which the Model 500 will operate are shown in Figure 4-10.

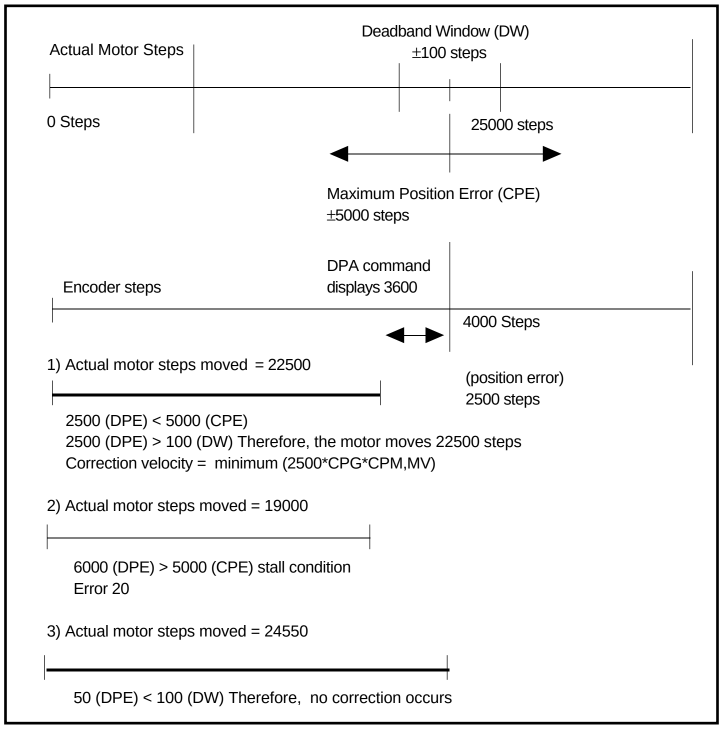

Actual Motor Steps
Deadband Window (DW)
±100 steps
0 Steps
25000 steps
Maximum Position Error (CPE)
±5000 steps
Encoder steps
DPA command
displays 3600
4000 Steps
1) Actual motor steps moved = 22500
(position error)
2500 steps
2500 (DPE) < 5000 (CPE)
2500 (DPE) > 100 (DW) Therefore, the motor moves 22500 steps
Correction velocity = minimum (2500*CPG*CPM,MV)
2) Actual motor steps moved = 19000
6000 (DPE) > 5000 (CPE) stall condition
Error 20
3) Actual motor steps moved = 24550
50 (DPE) < 100 (DW) Therefore, no correction occurs

Figure 4-10. Position Maintenance Example

To summarize, the report back commands available are:

> **PR**	Buffered report back of commanded setpoint
> **PX**	Buffered report back of actual encoder position
> **DPS**	Immediate report back of commanded setpoint
> **DPA**	Immediate report back of actual encoder position
> **DPE**	Immediate report back of position error

Set-up commands are:

> **FSB**	Enables/disable encoder step mode
> **FSC**	Enables/disable position maintenance mode
> **FSD**	Enables/disable stall detect mode

Position maintenance and stall detect parameters are:

> **CPG**	move correction gain percentage
> **CPM**	maximum move correction
> **CPE**	Maximum position error allowed
> **DW**	Deadband window
> **MV**	Maximum correction velocity allowed

With stall detect and position maintenance the Model 500 uses the following equation every sample period.

```
When in motion:
    position error > CPE ?  Yes, then a stall has occurred

When not in a move:
    Position error > CPE ? Yes, then a stall has occurred

    No then,
      Position error > DW ? No , then no action

    Yes then,
      Correction velocity = Minimum of (MV) or ( (CPG/100)*CPM * position error)
```

Using an Absolute Encoder

The model 500 is compatible with Compumotor's Photo-Trak, AR-C, and AL-C absolute encoders. You can still use the older version of Compumotor's absolute encoder (the AR23 -1), however it does not have the *rollover feature* or *variable resolution feature* that is available with the newer products mentioned above.

The AR-C and the AL-C both use the same interface protocol, and no special set-up commands are required to select one or the other. The Decoder box that comes with the AR-C and the AL-C has a ready-made cable in its ship kit that you can plug into the Model 500 with no additional wiring needed.

To configure the Model 500 to use absolute encoder feedback, you must enable the absolute encoder with the **FSM** command.

Command	Description
> **FSM1**	Enables the absolute encoder

You must use the encoder port labeled **ABS/INC**. You must also select this port with the **FSJ** command.

Command	Description
> **FSJØ**	Selects the **ABS/INC** encoder port

Position Maintenance and Stall Detect

The position maintenance and stall detect feature function the same for incremental encoders as well as absolute encoders. Follow the same procedures described in *Position Maintenance* and *Stall Detect* above.

Special Features

This section describes two special features when operating with an absolute encoder.

ERROR CONDITIONS

The absolute encoder option has two error conditions which will occur only with the absolute encoder. One error condition occurs when the absolute encoder does not respond to data requests by the Model 500. This can occur if power is not provided to the absolute encoder, or if the cable is disconnected between the absolute encoder and the Model 500. In either case, the step and direction signals to the drive system will be turned off and an error **71** will flash on the LED display. An **ON** or an **STØ** command is required to remove the latched error condition.

A second fault will occur if the 500 receives bad data reads from the absolute encoder. Each time the 500 reads the encoder, it checks to see if the change from the last encoder position is valid. If an invalid reading occurs two consecutive times an error condition will result. Invalid readings can be caused by noise in the wiring or in the absolute encoder itself. It can also occur if the cable between the absolute encoder head and the decoder box is disconnected. Refer to the *Troubleshooting* section of the encoder manual for prevention of noise problems. If this error occurs, the 500 will stop sending the step and direction signals to the drive system and an error **72** will flash on the LED display. An **ON** or an **STØ** command is required to remove the latched error condition.

ROLLOVER AND VARIABLE RESOLUTION

The absolute encoder can be set to have different resolutions. There is no command needed in the 500 to set it up for different resolutions; it will get this information automatically from the the absolute encoder's decoder box. You must still use the **ER** command to let the 500 know the number of encoder pulses to expect for one revolution of the motor. In a linear system, if you have the 500 set up with the desired motor resolution (**MR** setting), then the **ER** command is the number of encoder pulses that the 500 sees after sending out the number of pulses entered for the **MR** command value.

The absolute encoder has an absolute count range from 0 to 512 * encoder resolution. If the encoder resolution is set to 16384 positions/rev (ppr), then the count range is from 0 to 8388608 counts. If you move the motor and encoder outside of this range, then the position counter will *rollover*. If you rollover to a count of 9000000, then cycle power, you will then have a count of 611392. When rollover occurs a bit is set in the error flag (**ERXX111111**) This error flag is explained in the **Model 500 Software Reference Guide**. You can therefore check this error flag to detect if a rollover has occurred in your motion program.

Creating Motion Programs and Sequences

You must program the Model 500 to perform motion functions. A motion program consists of initialization (or 500 set-up), move profiles, and an I/O or RS-232C interface to execute motion instructions.

Sequence Commands

Sequences are the building blocks of motion programs for the 500. A sequence can be defined to be one command or up to 8 K bytes of commands. The sequences are stored in battery backed RAM. A sequence is a set of commands that is executed by issuing that sequence number. The 500 has programming capabilities that send program control from one sequence to another (i.e., a **GOTO** statement). The 500 also allows you to transfer program control to another sequence and return to the point of transfer (i.e., a subroutine call). The following commands define, erase, and run sequences as well as other specialized sequence functions. Refer to the **Model 500 Software Reference Guide** for detailed descriptions and syntax of the following commands.

Sequence Status Commands

Command	Description
`> XBS`	Reports the number of bytes available for sequence programming
`> XC`	Sequence checksum report
`> XDIR`	Reports the sequences that are defined and the number of bytes of memory they occupy

Sequence Programming Commands

Command	Description
`> XD`	Starts sequence definition
`> XT`	Ends sequence definition

Sequence Execution Commands

Command	Description
`> XQ`	Sets/resets Interrupted Run mode
`> XRP`	Runs a sequence with a pause
`> XR`	Runs a sequence
`> SSJ1`	Runs a sequence defined by binary weighted sequence inputs

Sequence Branching Commands

Command	Description
`> XG`	Exits current sequence and moves to execute another sequence
`> GOTO`	Exits current sequence and moves to execute another sequence
`> XR`	When used within a sequence it jumps to execute another sequence then returns to the sequence it was called from
`> GOSUB`	Jumps to execute another sequence then returns to the sequence it was called from

Sequence Debugging Commands

Command	Description
`> XTR`	Sequence Trace mode
`> XST`	Sequence Single Step mode
`> XS`	Sequence Execution status
`> #`	Step sequence command
`> DIN`	Simulate input state command
`> DOUT`	Simulate output state command

Special Sequence Commands

Command	Description
> **WHEN**	Special condition command
> **XWHEN**	Special condition sequence
> **XFK**	Fault sequence

A sequence is a series of commands. These commands are executed in the order in which they are programmed when the sequence is run. Immediate commands cannot be stored in a sequence, just as they cannot be stored in the command buffer. Only buffered commands may be used in a sequence.

The 500 has 8,000 bytes of non-volatile memory to store 100 sequences. You can use the **XBS** command to determine how many bytes are available in the sequence buffer and the **XDIR** command to determine what sequences have been programmed. The sequence buffers may have variable lengths, so you may have one long sequence or several short ones, as long as the total length does not exceed the 8,000 bytes of allocated space.

The commands that you enter to define a sequence are presented vertically in the examples below. This was done to help you read and understand the commands. When you are actually typing these commands into your terminal, they will be displayed horizontally.

To begin the definition of a sequence, enter the Define Sequence (**XD**) command immediately followed by a sequence identifier number (1 to 100) and a delimiter. The Terminate Sequence (**XT**) command ends the sequence definition. All commands that you enter after **XD** and before **XT** will be executed when the sequence is run. An example is provided below. Type the **DR** command to see query the state of the 500.

Command	Description
> **1DR**	Displays the present state of the 500.

Perform the following commands:

Command	Description
> **MPI**	Places the 500 in the incremental mode
> **MN**	Places the 500 in the preset mode
> **FSIØ**	Places the 500 in the indexer mode
> **LD3**	Disables the 500's limits

Command	Description
> **XE1**	Erases Sequence #1
> **XD1**	Begins definition of Sequence #1
A25	Sets acceleration to 25 rps^2
AD25	Sets deceleration to 25 rps^2
V1Ø	Sets velocity to 10 rps
D5ØØØ	Sets distance to 5,000 steps
G	Executes the move (Go)
H	Reverses direction
G	Executes the move (Go)
> **XT**	Ends definition of sequence
> **XR1**	Runs Sequence #1

You can run a sequence by entering the **XR** command immediately followed by a sequence identifier number (1 to 100) and a delimiter.

Once you define a sequence, it cannot be redefined until you delete it. You can delete a sequence with the **XE** command immediately followed by a sequence identifier (1 to 100) and a delimiter. You may then redefine that sequence.

Sequence #100 is a power-up sequence (if you have defined it). It is always run when you power up the system or when you reset the indexer with the Reset (**Z**) command. For convenience, you may find it advantageous to place all of your set-up commands in Sequence #100.

Sequences that you define are automatically saved into the 500's non-volatile memory. The only way to erase these sequences is by using the Erase Sequence (**XE**) command.

Creating and Executing Sequences

You can create sequences via RS-232C. Before you create sequences, you must understand the types of motion and the required user interfaces. To determine the proper user interface, you should be familiar with the methods of selecting sequences within your application.

Selecting Sequences

After you define the sequences from the RS-232C interface, you can execute the sequences by using one of the following modes of operation:

- ❏ Thumbwheel Switches: Use thumbwheel switches to select and run the sequence. Refer to the *Selecting Sequences with the Thumbwheel Module* section in this chapter for more on this feature.
- ❏ PLC (Programmable Logic Controller): Use the sequence select inputs to run a sequence. Refer to the *PLC Operation* section in this chapter for an example of this feature.
- ❏ Front Panel Pushbutton Execution: Refer to the *Front Panel Operation* section in this chapter for more on this feature.
- ❏ Computer Interface: Use the Execute Sequence (**XR**) command to run sequences. Refer to the *Host Computer Operation* section in this chapter for more on this feature.

Subroutines

When you use the Goto Sequence (**XG**) and Execute a Sequence (**XR**) commands, you can execute different sequences from within a sequence. These commands can be substituted for the **GOTO** and **GOSUB** commands respectively. If you use **XG** or **GOTO**, the program will move to the sequence that you specified in the **XG** or **GOTO** command. After executing the specified sequence, the system will not return to the original sequence. It will remain in the current sequence, unless it receives another execution command (**XG/GOTO** or **XR/GOSUB**). However, if you use the **XR** or **GOSUB** command, the program will return control to the original sequence that contained **XR** or **GOSUB**. Program control will return to the original sequence when a Terminate Sequence (**XT**) command is reached. This prompts the program to return to the sequence that initiated the move to another sequence.

You can nest up to 16 different levels of sequences within one program. For example, when you exit Sequence #1 to execute Sequence #2 with the **XR2** command, you can execute to Sequence #3 from Sequence #2. This nesting procedure can be repeated 16 times. The **XG** command has no limit since the program will not return control to the original sequence.

Command	Description
> **XE2**	Erases Sequence #2
> **XD2**	Defines Sequence #2
A1ØØ	Sets acceleration to 100 rps^2
AD1ØØ	Sets deceleration to 100 rps^2
V5	Sets velocity to 5 rps
D25ØØØ	Sets distance to 25000 steps
G	Executes the move (Go)
> **XT**	Ends Sequence 2 definition
> **XE3**	Erases Sequence #3
> **XD3**	Defines Sequence #3
A1Ø	Sets acceleration to 10 rps^2
V5	Sets velocity to 10 rps
D-25ØØØ	Sets distance to 25000 steps
G	Executes the move (Go)
> **XT**	Ends Sequence #3 definition
> **XE1**	Erases Sequence #1
> **XD1**	Defines Sequence #1
XR2	Executes Sequence #2
GOSUB3	Executes Sequence #3 (same as an XR command)
> **XT**	Ends Sequence #1 definition
1XTR1	Enables the trace mode
XR1	Executes Sequence #1

In the previous example, when you execute Sequence #1, the program moves to Sequence #2. After executing Sequence #2, the program returns to Sequence #1. The program then moves to execute Sequence #3. The trace mode was enabled to help you see how the sequence was executed.

Asynchronous Events-FAULT and WHEN

The 500 has special sequences that can be defined which run when a certain condition occurs. One such sequence that has already been explained is the power up sequence or sequence 100. This sequence is always run on power up and is often used to contain the necessary set-up commands for the application. The other two are the **FAULT** sequence and the **WHEN** sequence.

FAULT Sequence

The fault sequence is a sequence that is automatically executed when a fault condition or a Kill (**K**) occurs. Any condition that causes the 500 to fault and flash an error causes execution of the fault sequence if one is defined. If a kill command is issued, the sequence will also run. You can use the fault sequence to place the 500 in a safe state and turn off outputs that may be harmful to the rest of the system. You can use an **IF** command to determine what condition caused the fault so that the appropriate action can be performed. The **IF** statement is explained in the next section. The following steps illustrate the use of a fault sequence. The **IF** statement section explain more about the **ERnnnnn** flag that indicates what fault condition occurred.

STEP ① Define an input as a user fault input. You can use this input to indicate that a fault has occurred somewhere external to the system. This input will cause a fault condition.

Command	Description
> **IN1U**	Defines input #1 as a user fault input

STEP ② Designate Sequence #10 as the fault sequence.

Command	Description
> **XFK1Ø**	Designates sequence 10 as the fault sequence
> **XE1Ø**	Erases sequence #10
> **XD1Ø**	Defines sequence #10
1"External_Fault	Quote command
> **XT**	End definition of sequence #10

This defines the fault sequence. The quote command sends a message over the RS-232C link to the terminal to tell the operator that a fault has occurred. You can use the Quote command to write statements to the terminal. Sequence #10 will now be executed whenever a fault occurs. You may now program. In the following example, an alternating loop will be performed.

STEP ③ The normal *state* of this example application will be an alternating loop.

Command	Description
`> A5Ø`	Acceleration is 50 rps^2
`> AD4Ø`	Deceleration is 40 rps^2
`> D5ØØØØ`	Distance is 50000 steps
`> V7`	Velocity is 7 rps
`> L`	Loop command
`G`	Initiates motion
`H`	Change direction
`> N`	End the loop

The motor will alternate back and forth continuously.

STEP ④ You will now cause a system user fault error. Input states can be simulated using the **DIN** command. Type the following to simulate the activation of the user fault input.

Command	Description
`> DINEEE1`	Input #1 is activated

The user fault should have occurred and sequence #10 should have automatically been executed when the fault occurred.

Error #66 should be scrolled across the 500's display. To clear the fault from the display, enter the following command.

Command	Description
`> ON`	Clears fault message

An **IF** statement could have been used in the fault sequence to determine what fault condition occurred then branch to a sequence that handled that fault condition appropriately. For example the fault sequence will run for several faults. These faults are limits reached, general servo fault, and user faults. Each bit in the error flag (**ERnnnn**) will correspond to one of these fault conditions. Use the **IF** statement to determine which one occurred. Retype sequence #10 (the fault sequence) as follows:

Command	Description
`> 1XE1Ø`	Erases sequence 10
`> 1XD1Ø`	Defines sequence 10
`IF(ER1)`	IF the CCW limit is hit then
`1"CCW_LIMIT_WAS_HIT`	Display a message
`NIF`	End the IF (ER1) statement
`IF(ERX1)`	IF CW limit is hit then
`1"CW_LIMIT_WAS_HIT`	display a message
`NIF`	End the IF (ERX1) statement
`IF(ERXXXXXX1)`	IF the user fault occurred, display a message
`1"USER_FAULT`	
`NIF`	Ends the IF (ERXXXXXX1) statement
`> XT`	End fault sequence statement

Now depending on which error caused the program to branch to the fault sequence the appropriate message will be displayed. Before continuing with the manual disable the fault sequence with the **XFKØ** command.

WHEN Sequence

The **WHEN** sequence is a sequence that can be designated to run when a specific condition evaluates true. This could be a variable having a certain value, the inputs being in a specific state or the user flag having a set state. Whatever sequence or program that is currently being run will be interrupted when the condition evaluates true and the sequence designated by the **XWHEN** command will be executed. In the following example, when variable 3 is greater than 50, or input #1 is on, the 500 will execute the **XWHEN** sequence.

Command	Description
`> WHEN(VAR3>5Ø_OR_IN1)`	The when statement that must evaluate true in order for the **XWHEN** sequence to be run is defined.

STEP ① The **WHEN** sequence is now defined. Sequence #8 is designated as the **WHEN** sequence.

Command	Description
`> XWHEN8`	Sequence 8 is designated as a **WHEN** sequence
`> XE8`	Variable 3 is initialized to 0
`> XD8`	Defines sequence 8
`A5Ø`	Acceleration is 50 rps^2
`AD4Ø`	Deceleration is 40 rps^2
`D3ØØØØ`	Distance is 30000 steps
`V5`	Velocity is 5 rps
`G`	Loop command
`> XT`	Ends the sequence definition

STEP ② The normal program to be executed is defined and executed here.

Command	Description
`> VAR3=Ø`	Variable 3 is initialized to 0
`> A5Ø`	Acceleration is 50 rps^2
`> AD4Ø`	Deceleration is 40 rps^2
`> D5ØØØØ`	Distance is 50000 steps
`> V7`	Velocity is 7 rps
`> L`	Loop command
`G`	Initiates motion
`VAR3=VAR3+1`	Variable 3 is increased
`> N`	End the loop

STEP ③ You can cause the **WHEN** sequence to occur using either the **DIN** command to activate the input that will satisfy the **WHEN** condition or the current program will run 50 times, then the **WHEN** sequence will be executed.

STEP ④ Enter **XWHENØ** before you proceed with this user guide.

You can use the **WHEN** statement to change the mode of operation. This command can also preview the effect of multiple **WHEN** statements. The **WHEN** statement is similar to the **ONTRIG** command in Compumotor's Model 4000 indexer or Model 3000 indexer. Using an input to cause program execution to jump to the **XWHEN** sequence can change or interrupt the program. Within the **XWHEN** sequence, you can use an **IF** statement to check the state of one or more inputs. Based on the state of the inputs, different set ups of sequences can be executed. Disable the **XWHEN** sequence before continuing with this procedure by entering **XWHENØ**.

Power-Up Sequence Execution

You can program the 500 to execute a sequence of commands on power-up (sequences can be used as subroutines). Sequence #100 always runs on power up. To run another sequence on power up, put an **XR**<num> (or **XG**<num>) at the end of sequence #100. If sequence #100 is empty, nothing happens on power up. Refer to the **Model 500 Software Reference Guide** for detailed descriptions and syntax of the following commands.

Command	Description
> **XE1ØØ**	Erases sequence #100
> **XD1ØØ**	Begins definition of sequence #100
LD3	Disables limits if they are not connected
A2Ø	Sets acceleration to 20 rps^2
AD2Ø	Sets deceleration to 20 rps^2
V5	Sets velocity to 5 rps
D125ØØ	Sets distance to 12,500 steps
MN	Executes the move (Go)
MPI	Sets into the incremental mode
FSIØ	Sets unit to the indexer mode
XG1	Go to sequence #1
> **XT**	Ends sequence definition
> **XE1**	Erases sequence #1
> **XD1**	Defines sequence #1
G	Executes a go command
> **XT**	Ends definition of the sequence
> **Z**	Resets the indexer and runs sequence #100

A power-up sequence is typically used to store set-up or initialization parameters that your application requires. Having motion in your power-up sequence is not recommended. Examples of these set up commands are listed below.

Command	Description
> **SSJ1**	Continuous Sequence Scan Mode
> **SN**	Scan time
> **JA**	Jog acceleration
> **JVL**	Jog velocity low
> **JVH**	Jog velocity high

All **SS**, **FS**, or **OS** commands or setting up inputs with the **IN** commands, etc. You can put any buffered commands into sequence #100 (if you want to execute them during power up).

Sequence Debugging Tools

After create your sequences, you may need to debug the sequences to ensure that they are performing the functions properly. The 500 provides several debugging tools.

- ❏ In Trace mode, you can trace a sequence as it is executing.
- ❏ You can set the desired state of the 500 's inputs and outputs via software commands.
- ❏ You can enable error messages to explain why the 500 has stopped execution due to a programming error.

Trace Mode

You can use the Trace mode to debug a sequence or a program of sequences. The Trace mode allows you to track, command-by-command, the entire sequence as it runs. It displays to your terminal, over the RS-232C serial link, all of the commands as the they are executed. The following example demonstrates the Trace mode.

STEP ① Create the following sequence:

Command	Description
> XE1	Erases a sequence
> XD1	Defines sequence #1
A1Ø	Acceleration is 10 rps^2
AD1Ø	Deceleration is 10 rps^2
V5	Velocity is 5 rps
L5	Loop 5 times
GOSUB3	Jump to sequence #3
N	Ends the loop
> XT	Ends the definition of sequence #1

STEP ② Define sequence #3.

Command	Description
> XE3	Erases a sequence
> XD3	Defines sequence #3
D5ØØØØ	Sets the distance to 50000 steps
G	Initiates motion
> XT	Ends the definition of sequence #3

STEP ③ Enter the following command to enable the Trace mode.

Command	Description
> 1XTR1	Enables the Trace mode

STEP ④ You will now execute the sequence. The commands will be displayed on the terminal as each command in the sequence is executed. Enter the following command.

Command	Description
> XR1	Run sequence #1—response:

```
*SEQUENCE_ØØ1____COMMAND_A1Ø
*SEQUENCE_ØØ1____COMMAND_AD1Ø
*SEQUENCE_ØØ1____COMMAND_V5
*SEQUENCE_ØØ1____COMMAND_L5
*SEQUENCE_ØØ1____COMMAND_GOSUB3____LOOP_COUNT_1
*SEQUENCE_ØØ3____COMMAND_D5ØØØØ____LOOP_COUNT_1
*SEQUENCE_ØØ3____COMMAND_G____LOOP_COUNT_1
*SEQUENCE_ØØ3____COMMAND_XT____LOOP_COUNT_1
*SEQUENCE_ØØ1____COMMAND_N____LOOP_COUNT_1
*SEQUENCE_ØØ1____COMMAND_GOSUB3____LOOP_COUNT_2
*SEQUENCE_ØØ3____COMMAND_D5ØØØØ____LOOP_COUNT_2
*SEQUENCE_ØØ3____COMMAND_G____LOOP_COUNT_2
*SEQUENCE_ØØ3____COMMAND_XT____LOOP_COUNT_2
*SEQUENCE_ØØ1____COMMAND_N____LOOP_COUNT_2
*SEQUENCE_ØØ1____COMMAND_GOSUB3____LOOP_COUNT_3
*SEQUENCE_ØØ3____COMMAND_D5ØØØØ____LOOP_COUNT_3
.
.
.
*SEQUENCE_ØØ3____COMMAND_G____LOOP_COUNT_5
*SEQUENCE_ØØ3____COMMAND_XT____LOOP_COUNT_5
*SEQUENCE_ØØ1____COMMAND_N____LOOP_COUNT_5
*SEQUENCE_ØØ1____COMMAND_XT
```

The format for the Trace mode display is:

```
Sequence  Number_Command_Loop  Count
```

STEP ⑤ To exit the Trace mode, enter the following command:

Command	Description
> XSTØ	Exits Trace mode

Single-Step Mode

You can debug your program with another level of debugging with the Single-Step mode. Single-Step mode allows you to execute one command at a time when you want the command to be executed. Use the **XST** command to enable Single-Step mode. Once you are in the mode, you can execute a sequence one command at a time. To execute a command, you must use the **#** sign. By entering a **#** followed by a delimiter, you will execute the next command in the sequence. If you follow the **#** sign with a number (*n*) and a delimiter, you will execute the next *n* commands. To illustrate Single-Step mode, use the following steps:

STEP ① Enter Single-Step mode.

```
> XST1
```

STEP ② Begin execution of sequence #1

```
> XR1
```

STEP ③ You will not execute any commands until you use the # command. Type the following.

Command	Description
`> #`	Executes one command

The response will be:

```
*SEQUENCE_ØØ1____COMMAND_A1Ø
```

STEP ④ To execute more than one command at a time follow, the **#** sign with the number of commands you want executed.

Command	Description
`> #3`	Executes 3 commands, then pauses sequence execution

To complete the sequence, use the **#** sign until all the commands are completed. To exit Sequence-Step mode, type:

```
> XSTØ
```

Simulating I/O Activation

If your application has inputs and outputs that integrate the 500 with other components in your system, you can simulate the activation of these inputs and outputs so that you can run your sequences without activating the rest of your system. Thus, you can debug your program independently of the rest of your system. This is the same way in which a PLC program can be debugged by simulation of input and output states to run various portions of the program. The 500 uses two commands that allow you to simulate the input and output states desired. The **DIN** command controls the inputs and the **DOUT** command controls the outputs.

You will generally use the **DIN** command to cause a specific input pattern to occur so that a sequence can be run. Use the **DOUT** command to simulate the output patterns that are needed to prevent an external portion of your system from operating. You can set the outputs in a state that will be the inactive state of your external system. When you execute your program, a part in the program that will activate the outputs will not actually turn the outputs on to their active state because the **DOUT** command overrides this output and holds the external portion of the machine in an inactive state. When the program is running smoothly without problems you can activate the outputs and the 500 will affect the external system.

Outputs

The following steps describe the use and function of the **DOUT** command.

STEP ① Display the state of the outputs with the **OUT** command and the **O** command

Command	Description
`> 1OUT`	Displays the state of the outputs

The response will be:

```
*1_A_PROGRAMMABLE_OUTPUT______(STATUS_OFF)
*2_A_PROGRAMMABLE_OUTPUT______(STATUS_OFF)
*3_A_PROGRAMMABLE_OUTPUT______(STATUS_OFF)
*4_A_PROGRAMMABLE_OUTPUT______(STATUS_OFF)
```

Command	Response
`> 1O`	`*ØØØØ`
`> DOUT11EE`	

STEP ② Change the output state using the **O** command

Command	Description
`> O111Ø`	

STEP ③ Display the state of the outputs with the **OUT** command and the **O** command

Command	Description
`> 1OUT`	Displays the state of the outputs

The response will be:

```
*1_A_PROGRAMMABLE_OUTPUT______(DISABLED_ON)
*2_A_PROGRAMMABLE_OUTPUT______(DISABLED_ON)
*3_A_PROGRAMMABLE_OUTPUT______(STATUS_ON)
*4_A_PROGRAMMABLE_OUTPUT______(STATUS_OFF)
```

Command	Response
`> 1O`	`*111Ø`

STEP ④ You can now disable the outputs into the inactive state using the **DOUT** command. An **E** does not affect the output.

```
> DOUT00EE
```

STEP ⑤ Display the state of the outputs with the **OUT** command and the **O** command

Command	Description
`> 1OUT`	Displays the state of the outputs

The response will be:

```
*1_A_PROGRAMMABLE_OUTPUT______(DISABLED_OFF)
*2_A_PROGRAMMABLE_OUTPUT______(DISABLED_OFF)
*3_A_PROGRAMMABLE_OUTPUT______(STATUS_ON)
*4_A_PROGRAMMABLE_OUTPUT______(STATUS_OFF)
```

Command	Response
`> 1O`	`*ØØ1Ø`

Inputs

The following steps describe the use and function of the **DIN** command. You can use it to cause an input state to occur. The inputs will not actually be in this state but the 500 treats them as if they are in the given state and will use this state to execute its program.

STEP ① This sequence will wait for a trigger state to occur and will then begin moving in Continuous mode. An input that is configured as a stop (**S**) input will stop motion.

Command	Description
> **1IN1A**	Input #1 is a trigger input
> **1IN2A**	Input #2 is a trigger input
> **1IN3D**	Input #3 is a stop input
> **1INLØ**	The active input level is low
> **1XE1**	
> **1XD1**	Define sequence #1
TR11	Waits for the input trigger state to be 11
A1ØØ	Acceleration is set to 100 rps^2
AD1ØØ	Deceleration is 100 rps^2
V5	Velocity is 5 rps
MC	The continuous mode is activated
TR1Ø	Waits for a trigger input state of 10
G	Begins motion
> **XT**	Ends sequence definition

STEP ② Turn on the Trace mode so that you can view the sequence as it is executed.

Command	Description
> **1XTR1**	Turns on the trace mode

STEP ③ Execute the sequence.

Command	Description
> **XR1**	Runs sequence #1

STEP ④ The sequence will execute until the **TR11** command is encountered and will then pause waiting for the trigger condition to be satisfied. Simulate the input state using the **DIN** command. Inputs with an **E** value are not affected.

```
>  1DINEEE11EEEEE
```

STEP ⑤ The sequence will now execute until the **TR1Ø** trigger is encountered and will pause waiting for the new input pattern. Use the **DIN** command to simulate the desired input state.

```
>  1DINEEE1ØEEEEE
```

STEP ⑥ The motor will now move continuously until a the stop input is activated. Activate the stop input using the **DIN** command.

```
>  1DINEEE1Ø1EEEE
```

The motor will now stop. This illustrates the use of the **DIN** command to simulate the activation of input commands.

To deactivate the I/O simulation commands, type the following commands:

```
>  1DINEEEEEEEEEE
>  1DOUTEEEE
```

Error Messages

The 500 has an Error Message mode that can be activated to LED display and error message when an invalid command is attempted. This error message can be useful in debugging a sequence to determine what was wrong with the command that caused the sequence to not run properly. The **SSN** command enable the Error Message mode. Type the following commands to illustrate this mode.

Command	Description
`> 1SSN1`	Enters the error message mode
`> 1D1ØØØØØØØØØ`	

The 500 will respond with an error message:

```
*INVALID_DATA_FIELD
```

Command
`> 1DK1ØØØØ`

The 500 will respond with an error message;

```
*INVALID_COMMAND
```

The error message mode can be exited by typing:

Command	Description
`> 1SSNØ`	Exits the error message mode

Data

Inputs can be defined as data inputs to allow for external entry of motion data, loop counts, sequence select, time delays, and variable values. The following commands will read and enter data from the inputs:

❑ **VARD**	Variable Read
❑ **DRD**	Distance
❑ **VRD**	Velocity
❑ **LRD**	Loop Count
❑ **TRD**	Time Delay
❑ **XRD**	Sequence Number
❑ **FRD**	Following Ratio

The input lines when configured as data inputs are weighted differently than sequence select inputs. The weighting for data inputs is *binary coded decimal* or BCD. This weighting allows you to enter data via thumbwheels. To use the data inputs to enter data, the outputs must also be used. Outputs 1 - 3 must be configured as data strobe outputs. They are used to select or strobe the appropriate while reading data. Up to 16 digits of data and one sign bit may be entered. The recommended and most common method of controlling the input lines to read data is through thumbwheels. A PLC may also be used to enter data. An explanation of interfacing to thumbwheels or a PLC is introduced later in this chapter.

Delays

You can use the Time (**T**) command to halt the operation of the indexer function for a preset time. If you are in the Continuous mode, you may use the Time (**T**) command to run the motor at continuous velocity for a set time, then change to a different velocity. In Preset mode, the motor finishes the move before the indexer executes the time delay.

Command	Description
`> PS`	Waits for the indexer to receive a Continue (**C**) command before executing the next command
`> D25ØØØ`	Sets distance to 25000 steps
`> G`	Moves motor 25,000 steps
`> T5`	Waits 5 seconds after the move ends
`> H`	Changes motor direction
`> G`	Moves motor 25,000 steps in the opposite direction
`> C`	Continues execution

High-Level Programming Tools

The Model 500's X-language includes some commands that are common in most high-level programming languages (Pascal, Fortran or BASIC). In addition to these commands, 30 variables **VAR1-VAR3Ø** are provided for performing mathematical functions and boolean comparisons. You can access some system variables — **POS** (commanded position or setpoint), **ABS** (actual encoder position), and **FEP** (following encoder position). This section will introduce and explain how to use these structures with the 500.

Branching commands evaluate conditions statements to make branching decisions. If the condition is true, one set of commands is processed. If the condition is false, another set of commands will be executed. The structures that evaluate conditions are listed below.

IF (condition true)—*execute these commands*
ELSE—*execute these commands*
NIF

WHILE (condition true)—*execute these commands*
NWHILE

REPEAT—*Execute these commands*
UNTIL (condition true)

The condition statements that are evaluated can be very complex. The condition statements support all of the following decisions:

Variables

The 500 has up to 30 variables that can be used to perform multiplication, division, addition, and subtraction. You can assign these variables to various motion parameters. These parameters and the syntax of assigning a variable to them are listed below.

Command	Description
D(VAR1)	Loads the distance with the value of variable 1
V(VAR4)	Loads the velocity with the value of variable 4
A(VAR5)	Loads the acceleration with the value of variable 5
AD(VAR7)	Loads the deceleration with the value of variable 7
FP(VAR9)	Loads the following port with the value of variable 9
DP(VAR1)	Loads the distance point with the value of variable 1
L(VAR3Ø)	Loads the loop count with the value of variable 3Ø
XR(VAR21)	Executes the sequence number held in variable 21
T(VAR3)	Loads the time delay with the value of variable 3
FOL(VAR29)	Loads the following ratio with the value of variable 29

Variable assignments can be made in sequences or in the *immediate mode* (see the **Model 500 Software Reference Guide**. If a variable that has a fractional portion is assigned to a parameter that requires a whole number (such as distance), the variable is rounded to the nearest whole number and assigned to the parameter. To illustrate the use of the variables and the math functions they can perform, follow the steps below.

In addition, you can assign to the variables the value of the position counter (**POS**), encoder counter (**ABS**), or the following encoder counter. For instance, **VAR1=POS** is equivalent to leading **VAR1** with the value reported by the **1PR** status report. **ABS** is equivalent to the value of the **1PX** status report.

Assigning Variables to Constants

You can set parameters as variables in a sequence (**D**(**VAR**)). You can define these variables by assigning a constant value. When the sequence is run, this value is assigned to the corresponding parameter in the sequence.

STEP ① The sequence below executes a move and travels the distance provided in variable #1 and the velocity provided in variable #2.

Command	Description
`> LD3`	Disables limits (*Not needed if limits are installed*)
`> MPI`	Places the 500 in incremental mode
`> MN`	Enters the normal mode
`> FSIØ`	Places a 500F in the indexer mode. ***Not needed if you do not have the Following option***.
`> XE1`	Erases sequence #1
`> XD1`	Defines sequence #1
`> A1Ø`	Sets the Acceleration
`> AD1Ø`	Sets the Deceleration
`> V(VAR2)`	Assigns variable #2 to the velocity term
`> D(VAR1)`	Assigns variable #1 to the distance term
`> G`	Initiates motion
`> XT`	Ends the sequence definition

STEP ② You will now assign numbers to the variables **VAR1** and **VAR2**

Command	Description
`> VAR1=25ØØØ`	The distance parameter is assigned 25000
`> VAR2=5`	The velocity parameter is assigned 5

STEP ③ Execute sequence #1 with **XR1**. The motor will move 25000 steps at 5 rps. Verify that the distance (**D**) and the velocity(**V**) have been assigned these values.

Command	Response
`> 1V`	`*VØ5.ØØØØØ`
`> 1D`	`*D+ØØØ25ØØØ`

STEP ④ Now change the values of the variables as follows.

Command	Description
`> VAR1=75ØØØ`	The distance parameter is 75,000
`> VAR2=1Ø`	The velocity parameter is assigned 10

STEP ⑤ Repeat steps ③ and ④.

Entering Variables Via RS-232C

When using variables in sequences, you can use the **RSIN** command to interactively prompt the user to enter a variable number.

STEP ① Define the following sequence:

Command	Description
`> 1XE1`	
`> 1XD1`	Defines sequence #1
`1"ENTER_THE_NUMBER_OF_PARTS`	Prompts you for the # of parts to make
`1CR`	Inserts a carriage return
`1LF`	Inserts a line feed
`VAR5=RSIN`	The sequence stops here waiting for you to enter a #
`1CR`	Inserts a carriage return
`1LF`	Inserts a line feed
`L(VAR5)`	The loop count is loaded with the number of parts to make
`D25ØØØ`	Distance is 25000 steps
`G`	Initiates motion
`N`	Ends the loop
`1"FINISHED_WITH_PARTS_RUN`	Indicates that the run is finished
`1CR`	
`1LF`	
`> XT`	

The **RSIN** command prompts you to enter a variable via RS-232C that will be used in a sequence. When a variable is assigned to **RSIN**, the execution of the sequence is stopped until you enter the variable. To enter the variable, you must precede the number with an exclamation point (**!**).

STEP ② Turn on the Trace mode and run the sequence.

```
>  1XTR1
>  XR1
```

STEP ③ The program will stop at the **RSIN** command and wait for you to be enter a variable number. Enter the variable number.

```
>  !1Ø
```

The sequence executes a 25,000-step move 10 times making 10 parts.

Math Operations

In addition to assigning constants to variables, two system parameters can be assigned to a variable.

- **POS**—Commanded Position
- **FEP**—Following Encoder Position

These parameters are assigned as if they were constants. For example:

Command	Response
`>  VAR1=POS`	Assigns the current value of the command position to variable #1

Performing Math Operations with Variables

The 500 has the ability to perform simple math functions with its variables (add, subtract, multiply, and divide). The following sequence of steps illustrates the math capabilities of the 500.

STEP ① **Addition:**

```
Command                      Response
>  VAR1=5
>  VAR23=1ØØØ.565
>  VAR11=VAR1+VAR23
>  VAR23=VAR1Ø+VAR23
>  VAR1=VAR1+5
>  1VAR1                     *+ØØØØØØØØØØØØØ1Ø.ØØØØØ
>  1VAR11                    *+ØØØØØØØØØØØ1ØØ5.565ØØ
>  1VAR23                    *+ØØ1Ø1Ø.565ØØ
```

STEP ② **Subtraction:**

```
Command                        Response
>  VAR3=1Ø
>  VAR2Ø=15.5
>  VAR3=VAR3-VAR20
>  VAR19=VAR2Ø-VAR3
>  1VAR3                     *-ØØØØØØØØØØØØØ5.5ØØØØ
>  1VAR2Ø                    *+ØØØØØØØØØØØØØ15.5ØØØØ
>  1VAR19                    *ØØØØØØØØØØØØ21.ØØØØØ
```

STEP ③ **Multiplication:**

Command	Response
> VAR3=1Ø	
> VAR2Ø=15.5	
> VAR3=VAR3*VAR2Ø	
> VAR19=99	
> VAR19=VAR2Ø*VAR19	
> 1VAR3	*+ØØØØØØØØØØØØ155.ØØØØØ
> 1VAR2Ø	*+ØØØØØØØØØØØØØ15.5ØØØØ
> 1VAR19	*+ØØØØØØØØØØØ1534.5ØØØØ

STEP ④ **Division:**

Command	Response
> VAR3=1Ø	
> VAR2Ø=15.5	
> VAR3=VAR3/VAR2Ø	
> VAR3Ø=75	
> VAR19=VAR3Ø/VAR3	
> 1VAR3	*+ØØØØØØØØØØØØØØØ.64516
> 1VAR2Ø	*+ØØØØØØØØØØØØØ15.5ØØØØ
> 1VAR3Ø	*+ØØØØØØØØØØØØØ75.ØØØØØ
> 1VAR19	*+ØØØØØØØØØØØØ116.25Ø23

Complex Branching and Looping

The 500 supports the high-level language structures for branching and looping. Each conditional branch or loop evaluates a condition statement. Depending on whether this condition statement evaluates true or not determines where the 500 will branch to. The unconditional branching and looping statements have been introduced already. These are the **GOTO** (Branch), **GOSUB** (Branch & Return) and **L** (Loop) command. The **L** command is explained further here.

Unconditional Looping

The loop command is an unconditional looping command You may use the Loop (**L**) command to repeat a series of commands. You can nest Loop commands up to 16 levels deep.

Command	Description
> PS	Pauses command execution until the indexer receives a Continue (**C**) command
> MPI	Sets unit to Incremental mode
> A50	Sets acceleration to 50 rps^2
> V5	Sets velocity to 5 rps
> L5	Loops 5 times
> D2ØØØ	Sets distance to 2,000 steps
> G	Executes the move (Go)
> T2	Delays 2 seconds after the move
> N	Ends loop
> C	Initiates command execution to resume

The motor moves a total of 10,000 steps

The example below shows how you can nest a loop inside a loop. In this example, the motor makes two moves and returns a line feed. The unit repeats these procedures until you instruct the it to stop.

```
Description                      Command
Pauses command execution         > PS
Loops indefinitely               > L  ─────────────────┐
Sends a line feed                > 1LF                 │
Loops twice                      > L2 ──────┐          │
Executes 2,000-step move         > G        │  Nested  ├─ Loop
Waits Ø.5 seconds                > T.5      ├─ Loop    │
Ends loop                        > N  ──────┘          │
Ends loop                        > N                   │
Continues command                > C  ─────────────────┘
execution
```

This command execution continues until you issue the Stop (**S**) or Kill (**K**) commands.

Unconditional Branching

The unconditional branching commands **GOTO** and **GOSUB** were explained earlier in the sequence section.

Conditional Statements

You can use the following types of conditional statements with the 500.

- ❏ Error Flags
- ❏ User Flags
- ❏ Input State
- ❏ Boolean Comparisons

ERROR FLAGS

The error flag (**ERXXXXXXXX**) is useful if you want to trap different error conditions and create different sequences to respond to them. The ***Model 500 Software Reference Guide*** explains the errors that can be trapped in the evaluation command description. An example of the statement's use is provided below.

```
> IF(ER11ØXXXX)
> GOSUB2
> NIF
```

USER FLAGS

You can set the user flag (**FLØØØ111XØ**) and modify it within sequences to mark where the program has gone or to indicate any special state so that a conditional statement can be made. An example of the statement's use is provided below.

```
> WHILE(FLØØ11XXXX)
> D1ØØ
> G
> NWHILE
```

INPUT STATE

An example of this statement's use (**IN111100010**) is provided below.

```
> REPEAT
> GOSUB4
> UNTIL(INØØ1XX11)
```

Variable comparisons

- **VARn>VARm**
- **VARn<VARm**
- **VARn=VARm**

```
IF(VAR1>VAR2)
WHILE(VAR3=1Ø)
GOSUB2
NWHILE
NIF
```

BOOLEAN COMPARISONS

An example of the statement's use is provided below.

```
WHILE(VAR3>1Ø_AND_IN11ØØ11)
GOSUB8
NWHILE
```

Condition statements are explained in the ***Model 500 Software Reference Guide***.

Conditional Looping

The 500 supports two conditional looping structures—**REPEAT/UNTIL** and **WHILE**.

REPEAT/UNTIL

With the **REPEAT** command, all commands are repeated between **REPEAT** and **UNTIL**. The 500 stops executing the commands when the **UNTIL** condition is true.

STEP ①

Command	Description
`> VAR5=Ø`	Initializes variable 5 to 0
`> 1XE1Ø`	Erases sequence #10
`> 1XD1Ø`	Defines sequence #10
`REPEAT`	Begins the **REPEAT** loop
`A5Ø`	Acceleration is 50 rps^2
`AD5Ø`	Deceleration is 50 rps^2
`V5`	Sets velocity to 5 rps
`D25ØØØ`	Distance is 25,000 steps
`G`	Executes the move (Go)
`VAR5=VAR5+1`	Variable 5 counts up from 0
`UNTIL(INXXX111Ø_OR_VAR5>1Ø)`	When the 111Ø input condition occurs or VAR5 is greater than 10, the loop will stop.
`1"DONE_LOOPING`	Quote command indicates that the loop is finished
`1CR`	Inserts a carriage return
`1LF`	Inserts a line feed
`> XT`	

STEP ②

Use the Trace mode to display the commands as they are run.

```
> 1XTR1
> XR1Ø
```

The loop can be exited either by using the **DIN** command to satisfy the input state or letting the **VAR5** counter count to 10.

WHILE

With the **WHILE** command, all commands are repeated between **WHILE** and **NWHILE** until the **WHILE** condition is true.

STEP ①

Command	Description
`> VAR5=Ø`	Initializes variable 5 to 0
`> 1XE1Ø`	Erases sequence #10
`> 1XD1Ø`	Defines sequence #10
`WHILE(INXXX111Ø_OR_VAR5<1Ø)`	While the input pattern is equal to XXX111Ø or variable 5 is less than 10, repeat the loop
`A5Ø`	Acceleration is 50 rps^2
`AD5Ø`	Deceleration is 50 rps^2
`V5`	Velocity is 5 rps
`D25ØØØ`	Distance is 25000 steps
`G`	Executes the move (Go)
`VAR5=VAR5+1`	Variable 5 counts up from 0
`NWHILE`	
`1"DONE_LOOPING`	Indicates that the loop is done
`1CR`	Inserts a carriage return
`1LF`	Inserts a line feed
`> XT`	

STEP ② Use the Trace mode to display commands as they are run.

```
>  1XTR1
>  XR1Ø
```

You can exit the loop with the **DIN** command (cause the input state to not match the **IN** command). You can also exit the loop by setting **VAR5** counter count to 10. If the input pattern is not **XXX111Ø**, the loop will not be executed.

Conditional Branching

You can use the **IF** statement for conditional branching. All commands between **IF** and **ELSE** are executed if the condition is true. If the condition is false, the commands between **ELSE** and **NIF** are executed. If the **ELSE** is not needed, it may be omitted. The commands between **IF** and **NIF** are executed if the condition is true. Examples of these statements are provided.

❑ Error Flag
❑ User Flag
❑ Input State

Error Flags

The **IF** command checks for error conditions. If an error exists, a conditional command will be executed. This command is useful if you wish to trap different error conditions (Drive Disabled, User Fault Input Activated, Excessive Position Error, etc). Refer to the ***Model 500 Software Reference Guide***.

Command	Description
`> XE1Ø`	Erases sequence #10
`> XD1Ø`	Defines Sequence #10—when a fault occurs, sequence #10 will execute—the sequence is defined with the Set Fault or Kill Sequence (**XFK1Ø**) command
`IF(ER1)`	If hardware **CCW** limit switch is reached, perform the following commands:
`1"CCW_LIMIT_HIT`	Display the error message
`NIF`	Ends IF statement
`IF(ERX1)`	If hardware **CW** limit switch is reached, perform the following commands:
`1"CW_LIMIT_HIT`	Display the error message
`NIF`	Ends IF statement
`> XT`	Ends Sequence loop
`> XFK1Ø`	Sets Sequence #10 as the Fault sequence

User Flags

This command uses the pattern set by the User Flag (**SFL**) command to run conditional commands. This command is useful if you want to make a decision based on previous sequence executions that will set or clear the user flag bits. For example, if an application has several sequences, you can assign different bit patterns with the **SFL** command at the end of each sequence. If you select these sequences from the host computer, you may wish to make different moves depending on the sequence you ran. Refer to the ***Model 500 Software Reference Guide*** for a detailed description of the **SFL** command.

Command	Description
> **PS**	Waits for the indexer to receive a Continue (**C**) command before executing the next command
> **SFL1Ø1Ø**	Sets user flag bits 7 and 5 and clears bits 6 and 4, the remaining bits are not altered
> **IF(FL1Ø1Ø)**	If user flag bits 5 and 7 are set, and bits 6 and 4 are clear, perform the following commands
> **A1Ø**	Sets acceleration to 10 rps^2
> **V5**	Sets velocity to 5 rps
> **D25ØØØ**	Sets distance to 25,000 steps
> **G**	Executes the move (Go)
> **NIF**	Ends IF statement
> **C**	Continues execution

If the **FL** pattern matches the **SFL** setting, the motor moves 25,000 steps. You can change the **SFL** pattern at different points in sequences to map a path for sequence execution.

Input State

This command compares the input pattern (**CW**, **CCW**, and **HM** and **I1 - I7**) to execute the conditional commands. This command is useful for branching and performing conditional moves using the programmable inputs. For a detailed description of this command, refer to the ***Model 500 Software Reference Guide***.

Command	Description
> **1XE5**	Erases sequence #5
> **1XD5**	Defines sequence #5
IF(INXXX1Ø)	If **I1** is active and **I2** is not active, issue the following commands:
A1Ø	Sets acceleration to 10 rps^2
V5	Sets velocity to 5 rps
D25ØØØ	Sets distance to 25,000 steps
G	Executes the move (Go)
NIF	Ends IF statement
IF(INXXXØ1)	If **I1** is not active (open)and **I2** is active (closed), issue the following commands:
A1Ø	Sets acceleration to 10 rps^2
V5	Sets velocity to 5 rps
D-5ØØØ	Sets distance to 5,000 steps in the opposite direction
G	Executes the move (Go)
NIF	Ends IF statement
IF(INXXX1)	If **I1** is active, do the following command.
1"DONE	Ends message saying done
NIF	Ends IF statement
> **1XT**	Ends sequence definition

Use the **DIN** command or the inputs themselves to execute the different trigger input states. You can use the Trace mode to see what commands are executed.

Branching Using Variables and Boolean Logic

You can use the **IF** statement to branch based on variable values. Multiple comparisons can be made in one condition statement using the boolean **OR** and **AND** functions.

Command	Description
`> XE8`	Erases sequence 8
`> XD8`	Defines sequence 8
`VAR5=15`	Variable 5 = 10
`IF(VAR5>1Ø_AND_VAR4=2Ø)`	If Variable 5 is less than 10 and variable 4 = 20 then perform the commands until the ELSE
`A1ØØ`	Sets acceleration to 100 rps^2
`AD1ØØ`	Sets deceleration to 100 rps^2
`V5`	Sets velocity to 5 rps
`D25ØØØ`	Sets distance to 25,000 steps
`G`	Executes the move (Go)
`VAR5=VAR5-1`	Variable 5 decrements one
`ELSE`	Ends IF statement
`A1ØØ`	Sets acceleration to 100 rps^2
`AD1ØØ`	Sets deceleration to 100 rps^2
`V5`	Sets velocity to 5 rps
`D-5ØØØ`	Sets distance to 5,000 steps in the opposite direction
`G`	Executes the move (Go)
`NIF`	Ends the IF
`> XT`	Ends definition of sequence 8

Motion Profiling Mode—On-the-Fly Changes

The Motion Profiling mode allows you to execute buffered commands while a move is being made (on-the-fly). When you enter this mode, the 500 will enter all the commands that you enter immediately. You can enter and exit this mode from within a sequence. This mode allows you to change velocity on-the-fly based on distance, turn on outputs based on distance, perform math and other commands while in motion. The following commands are used with Motion Profiling mode.

❑ **MPP**—Enter Motion Profiling Mode
❑ **NG**—Exits Motion Profiling Mode
❑ **DP**—Sets Distance Points within Motion Profiling Mode

While the 500 is in Motion Profiling mode, you can execute any command while a move is being made. When the 500 encounters an **MPP** command in a sequence, all subsequent commands will be executed until the **NG** command is encountered. An example of the **MPP** command is provided below.

```
XD1  D5ØØØØ  V1  MPP  G  O1  TR1X1  V4  NG  XT
```

In this example, a 50,000-step move is made. The initial velocity is 1 rps. Motion begins with the **G** command. Output 1 is turned on with the **O1** command. The 500 then runs the trigger command (**TR**) until the condition is true, at which point it changes the velocity. When the 500 encounters the **NG** command, it will not receive any additional commands until the 50,000-step move is completed.

Changes Based on Distance

Changes are made based on distance using the distance point (**DP**) command. This command causes a delay in the processing of the commands until the motor reaches the specified distance point. Processing will continue once the distance point is reached. In this way, velocity changes and the activation of outputs can be done based on distance.

The distance point is interpreted differently for Absolute mode versus Incremental mode. To change velocity on-the-fly based on distance, the Motion Profiling Mode (**MPP**) command must be used with the Distance Point (**DP**) command. The following sequence example executes the profile shown in Figure 4-11.

```
1XD1  PZ  D1ØØØØØ  V2  MPP  G  DP25ØØØ  O1  DP5ØØØØ  OØ  NG  XT
```

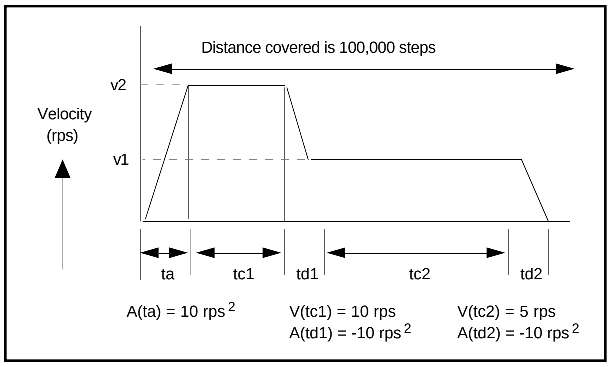


Figure 4-11. Motion Profiling Mode Example

In Incremental mode, commands are processed until the DP command is reached. The 500 pauses at the **DP** command until the motor moves 25,000 steps. The 500 then turns on output #1. The 25,000 steps are counted from the point at which the command is encountered. When **DP5ØØØØ** is reached, the 500 pauses until the motor moves 50,000 steps. The 500 then turns output #1 off. In this example, if the 500 in in Incremental mode, the output will be turned on at 25,000 steps and turned off after the motor has moved a total of 75,000 steps from the beginning of the first move.

In absolute mode, the value specified with **DP** is interpreted as an absolute position relative to the zero point location. In this example, the output would be turned on at 25,000 steps and turned off after the motor had passed the 50,000 step point.

Stopping Motion with a Stop Input

A common use of the Motion Profiling mode is to perform a continuous move and stop the move from the inputs. This can be done in two ways. You can define an input as a stop input. Motion can be stopped by activating the stop input.

The other method is to use the **STOP** command and place it within a sequence. The following step-by-step example illustrates these two methods.

STEP ① Define the input as a stop input.

Command	Description
> **IN1D**	Defines input #1 as a stop input

STEP ② Create the sequence with a continuous move.

Command	Description
> **XD1**	Begins definition of sequence #1
V5	Sets velocity to 5 rps
A1ØØ	Sets acceleration to 100 rps^2
MC	Sets the 500 to Continuous mode
MPP	Sets the 500 to Motion Mode Profiling
G	Executes the move (Go)
NG	Exits Motion Profiling mode
> **XT**	Ends sequence #1 definition

STEP ③ Execute sequence #1 with an **XR1** command. The motor will move at 5 rps and will not stop until you activate the stop input (input #1). Activate the stop input.

The motor will stop at a controlled deceleration. In this case, the buffer will be dumped and the sequences will not be finished. The **STOP** command caused program control to exit the sequence. The **G** command was the last command that was run.

STEP ④ Issue the Display Parameters (**DR**) command. The 500 is still in Motion Profiling mode. Enter the **NG** command to exit the mode.

Stopping Motion with the STOP Command

The following step-by-step example illustrates the method of stopping a continuous move with the **STOP** command.

STEP ① Configure input #1 as a trigger input.

Command	Description
`> IN1A`	Configures input #1 as a trigger input

STEP ② Create a sequence with a **STOP** command after the trigger.

Command	Description
`> XD1`	Begins the definition of sequence #1
`V5`	Sets velocity to 5 rps
`A1ØØ`	Sets acceleration to 100 rps^2
`MC`	Sets the 500 to Continuous mode
`MPP`	Sets the 500 to Motion Mode Profiling
`G`	Executes the move (Go)
`TR1`	Activates trigger #1
`STOP`	Stops motion when the trigger condition is met
`NG`	Exits Motion Profiling mode
`> XT`	Ends sequence #1 definition

STEP ③ Issue the Display Parameters (**DR**) command. The 500 is still in Motion Profiling mode.

In both of the examples, an **NG** command was required after motion was stopped. When a **STOP** command is issued, the command buffer is emptied. Therefore, the commands that have not been executed in the sequence at the time the stop occurs will not be executed. In both examples,the **NG** command is not executed because motion was stopped. In fact, **NG** can never be executed under these conditions. *The important point is that if the* ***STOP*** *command is used to stop continuous motion, the* ***NG*** *must be issued either at the beginning of the next sequence or directly via the RS-232C interface.*

To prevent the 500 from stopping without finishing the sequence that it is currently executing, a software switch has been provided that will cause the 500 to continue executing the sequence it was running when the **STOP** was issued. By entering the Clear/Save the Command Buffer on Stop (**SSH1**) command, the 500 will stop motion when it encounters a **STOP** and continue processing the commands in the sequence. *Enter* ***SSH1*** *and repeat the previous example. Notice how the Motion Profiling mode is exited.*

Sequence Scan Mode and the Stop Command

In applications that use the Sequence Scan mode and the **STOP** command, you must use the (**SSH**) command to prevent the Sequence Scan mode from being disabled. Under normal conditions, the Sequence Scan mode is aborted when the 500 encounters a **STOP** command. If the **STOP** command is issued from a stop input or from within a sequence, the Sequence Scan mode will be aborted. Usually, you will not want to abort the mode. The **SSH1** command will allow the 500 to complete the execution of the current sequence from the point at which the **STOP** was issued. You must re-enable the Sequence Scan mode by placing the **SSJ1** command in the sequence after the point at which the stop will occur. The following example illustrates such a sequence.

STEP ① Configure input #1 as a trigger input and sequence #2 as a sequence select input.

Command	Description
> **IN1A**	Configures input #1 as a trigger input
> **IN2B**	Configures input #2 as a sequence select input

STEP ② Enable the Sequence Scan mode with the **SSJ1** command.

STEP ③ Create a sequence with a **STOP** command after the trigger.

Command	Description
> **XD1**	Begins the definition of sequence #1
V5	Sets velocity to 5 rps
A1ØØ	Sets acceleration to 100 rps^2
MC	Sets the 500 to Continuous mode
MPP	Sets the 500 to Motion Mode Profiling
G	Executes the move (Go)
TR1	Activates trigger #1
STOP	Stops motion when the trigger condition is met
NG	Exits Motion Profiling mode
SSJ1	Re-enable Sequence Scan mode
> **XT**	Ends sequence #1 definition

STEP ④ Execute the sequence by activating input #21. Stop the move at any time by activating input #1.

STEP ⑤ The 500 is still in Sequence Scan mode and the move can be repeated by activating input #2 again (and stopped with input #1).

Other Uses of the Motion Profiling Mode

The Motion Profiling mode allows a great amount of flexibility in the complexity of the control of the 500 during motion. You can change gains on-the-fly based on distance or following error. You can turn on outputs, change velocity, and perform math functions. The primary application concern to consider during sequence execution is the amount of time required to perform the commands. In some cases, the execution of commands may depend on the motion. The following examples show additional uses of the Motion Profiling mode.

CHANGING GAINS ON-THE-FLY

Command	Description
`> 1XD1`	Begins the definition of sequence #1
`1D4ØØØØØ`	Sets distance to 400,000 steps
`1V5`	Sets velocity to 5 rps
`1MPP`	Sets 500 to Motion Profiling mode
`1G`	Executes the move (Go)
`1DP15ØØØØ`	Sets distance point to 150,000
`1V2`	Changes velocity to 2 rps
`1BCPP2Ø`	Changes gain to 20% of maximum
`1DP25ØØØØ`	Sets distance point to 250,000
`V5`	Changes velocity to 2 rps
`BCPP45`	Changes gain to 45% of maximum
`NG`	Exits 500 from Motion Profiling mode
`> XT`	Ends the definition of sequence #1

TURNING ON INPUTS, USING TIME DELAYS, AND MATH

Command	Description
`> 1XD1`	Begins the definition of sequence #1
`1PZ`	Sets axis #1 to Ø
`1MC`	Sets the 500 to Continuous mode
`1MPP`	Sets 500 to Motion Profiling mode
`1G`	Executes the move (Go)
`1T.5`	Sets a 0.5 second delay
`1O11Ø`	Turns outputs #1 and #2 on, #3 off
`1T.5`	Sets a 0.5 second delay
`1OØØ1`	Turns outputs #1 and #2 off, #3 on
`REPEAT`	Starts repeat loop
`VAR1=VAR1+1`	Increases variable #1 by 1
`T.1`	Sets a 0.1 second delay
`UNTIL(POS>4ØØØØØ)`	Continues looping until the 500's position is > 400,000
`STOP`	Halts command processing
`NG`	Exits 500 from Motion Profiling mode
`> XT`	Ends the definition of sequence #1

Triggers, input states (**`INXX111`**) time delays, and the distance points are the commands that will allow you to control when and where procedures occur during motion in your program. The Motion Profiling mode offers you the flexibility to accomplish a variety of different application needs.

Interfacing to the 500

This section discusses the various interfaces that you may use with the 500.

❏ Operation from Programmable Inputs and Outputs
  • Switches
  • Thumbwheels
❏ PLC Operation
❏ Front Panel Operation
❏ Remote Panel (RP240) Operation
❏ Host Computer Operation

Programmable Inputs and Outputs

The 500 has a very flexible input and output scheme for defining I/O in a way that is suitable for almost any application. There are 13 programmable inputs (**IN1 - IN13** on the front panel). The other three inputs are dedicated for limits. There are also 4 programmable outputs. Using the inputs in combination with the outputs you can use up to 16 digits of thumbwheels with the 500. This section explains some of the functions that the inputs and outputs can perform and explains how to use thumbwheels for an interface with the 500.

Output Functions

You can turn the programmable outputs (**OUT1 - OUT8**) on and off with the Output (**O**) and Immediate Output (**IO**) commands. Outputs **OUT1 - OUT8** are factory set as programmable outputs. However, you can configure all of the outputs to perform different functions (Moving/Not Moving, Amp Off, Strobe, etc.) with the Configure Output (**OUT**) command. Refer to the **OUT** command in the **Model 500 Software Reference Guide** for descriptions of the available functions. You can use these outputs to turn on and off other devices (i.e., lights, switches, etc.). The output functions have unique letter assignments.

A:	Programmable Output
B:	Moving/Not Moving
C:	Sequence in Progress
D:	At Soft of Hard Limits
E:	At Position Zero
F:	Fault Indicator
G:	Not Used
H:	Amp Off
J:	Strobe Out
K:	Invalid Command Error
L:	Position Error Fault
M:	Not Used
N:	CW Software Limit Reached
P:	CCW Software Limit Reached
R:	CW Hardware Limit Reached
S:	CCW Hardware Limit Reached
T:	Output Based on Position
U:	Programmable Pulse output

Command	Description
`> PS`	Pauses command execution until the indexer receives a Continue (**C**) command
`> MN`	Sets unit to Normal mode
`> A1Ø`	Sets acceleration to 10 rps^2
`> V5`	Sets velocity to 5 rps
`> D25ØØØ`	Sets distance to 25,000 steps
`> OUT1A`	Sets **OUT1** as a programmable output
`> OUT2A`	Sets **OUT2** as a programmable output
`> OUT3B`	Sets **OUT3** as a Moving/Not Moving output
`> O1Ø`	Turns **OUT1** on and **OUT2** off
`> G`	Executes the move
`> OØ1`	Turn **OUT1** off and **OUT1** on
`> C`	Initiates command execution to resume

This example defines **OUT1** and **OUT2** as programmable outputs and **OUT3** as a Moving/Not Moving output. Before the motor moves 25,000 steps, **OUT1** is turned on and **OUT2** is turned off. These outputs will remain in this state until the move is completed, then **OUT1** will turn off and **OUT2** will be turned on. While the motor is moving, **OUT3** remains on.

Pulse Output-Half Axis

A second half-axis can be programmed using the 500. This output can control the distance and a constant speed for another drive/motor system. A second output can be used to control the distance. The following commands will set up a second axis and make it move.

Command	Description
`> OUT1U`	Configures the output to be a pulse output
`> OUT2A`	The output will be used to control direction
`> PU1ØØØ,2`	The output will send 1000 pulses at a rate of 2 ms
`> PUL`	This commands begins the pulse train out of output 1

Input Functions

The inputs can individually be programmed to perform any of the following functions. Each function has an assigned letter:

A:	Trigger
B:	Sequence Select
C:	Kill
D:	Stop
E:	Command Enable
F:	Pause/Continue
G:	Go
H:	Direction
I:	Synchronization
J:	Jog+ (CW)
K:	Jog- (CCW)
L:	Jog Speed Select
M:	Not Used
N:	Data
O:	No Function assigned
P:	Memory Lock
Q:	Registration input (**IN10 - IN13** Only)
R:	Reset
S:	Go Home
T:	Position Zero
U:	User Fault
V:	Data Valid
W:	Data Sign
X:	Increase Following ratio
Y:	Decrease Following ratio
Z:	No Function assigned

To designate each input pin to a particular function, use the Set Input Functions (**IN**) command. To see what the inputs are currently defined as, type **1IN**. To see the inputs' states, use the **1IS** command. Enter the following commands.

```
> 1IN
```

Change input 1 to be a stop input by entering:

```
> IN1D
```

Check that it was assigned properly by again entering:

```
> 1IN1
```

With this method, you can assign all the inputs to any of the functions listed above. (*Except the registration input function. Only inputs **IN10 - IN13** can be assigned as registration inputs.*)

Switches

This section contains information on 500 triggers and sequence scanning with inputs.

Triggers

You can use the Trigger Pause (**TR**) command to pause a sequence of buffered commands until one or more inputs come to a preferred state. Inputs **IN10 - IN13** are set at the factory (default setting) to function as trigger inputs.

Command	Description
`> IN1D`	Sets **IN1** as Stop(S) input
`> IN2A`	Sets **IN2** as trigger input 1.
`> IN3A`	Sets **IN3** as trigger input 2
`> IN4A`	Sets **IN4** as trigger input 3
`> MC`	Sets the unit to Continuous mode
`> MPP`	Enters the Motion Profiling mode
`> A15`	Sets acceleration to 15 rps^2
`> AD15`	Sets acceleration to 15 rps^2
`> TR1`	Waits for trigger input 1 (**IN2**) to be on
`> G`	Executes a go (Go) command
`> L`	Loops infinitely
`V5`	Sets velocity to 5 rps
`TRXØ1`	Waits for trigger input 2 (**IN3**) to be off and trigger input 3 (**IN4**) to be on
`V1`	Sets velocity to 5 rps
`TRX1Ø`	Waits for trigger input 2 (**IN3**) to be on and trigger input 3 (**IN4**) to be off
`> N`	Ends the loop

This example program configures **IN1** as a stop input and **IN2** as a trigger (Start) input. **IN3** and **IN4** are programmable trigger inputs. If you activate the Start input (**IN2**), the motor will begin moving. Because the 500 is in Motion Profiling mode, it will execute the loop and subsequent commands. As it reaches each trigger statement, it waits for that state to become true and changes the velocity to the velocity following the command. The loop is infinite so it will continuously toggle between 1 and 5 rps as the trigger statements come true and will not stop until the stop input is activated. If you activate **IN1** during the operation of the 500, the indexer immediately stops the operation and clears the command buffer.

Sequence Select & Sequence Scanning

Inputs can be defined as sequence select inputs. This allows you to execute sequences defined via RS-232C by activating the sequence select inputs. Sequence select inputs are assigned BCD weights. The lowest input number assigned as a sequence select input will have the least significant value. Figure 4-12 shows the BCD weights of the 500's inputs when inputs **IN1—IN8** are configured as sequence select inputs.

The inputs are weighted with the least weight on the smallest numbered input, as shown in Figure 4-12. If we had selected **IN6**, **IN9**, **IN10** & **IN13** instead of **IN5**, **IN6**, **IN7** & **IN8**, then the weights would be as follows:

IN6	=	10
IN9	=	20
IN10	=	40
IN13	=	80

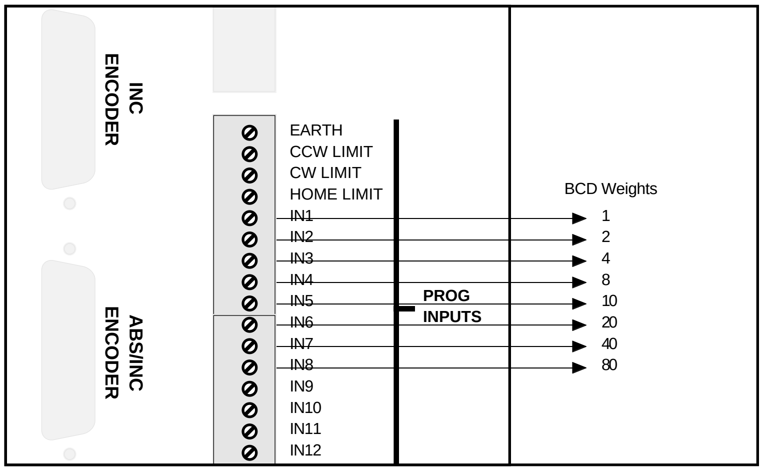


Figure 4-12. Dedicated BCD Weight of 500 Inputs

Table 4-1 shows one possible input configuration and BCD weighting.

Input	Function	Binary Weighting
IN1	Sequence Select	1
IN2	Sequence Select	2
IN3	Sequence Select	4
IN4	Sequence Select	8
IN6	Sequence Select	10
IN9	Sequence Select	20
IN10	Sequence Select	40
IN13	Sequence Select	80

Table 4-1. Input Configuration/BCD Weighting Example

If the inputs are configured as in Table 4-1, sequence #6 will be executed by activating **IN2** and **IN3**. Sequence #29 will be executed by activating inputs **IN1**, **IN4**, and **IN9**.

To execute sequences using the sequence select lines, place the 500 in Sequence Scan mode with the **SSJ1** command. In this mode, the 500 will continuously scan the input lines and execute the sequence selected by the active sequence select lines. To disable the sequence scan mode, enter **SSJØ**.

Once enabled (**SSJ1**), the 500 will run the sequence number that the active sequence select inputs and their respective BCD weights represent. After executing and completing the selected sequence, the 500 will scan the inputs again and run the selected sequences. If a sequence is selected that has not been defined via RS-232C, no sequence will be executed.

If it is not desirable for the 500 to immediately execute another sequence after running the currently selected sequences the Sequence Interrupted Run Mode (**XQ1**) can be enabled. In this mode, after executing a sequence, all sequence select lines must be placed in an inactive state before a new sequence can be selected. The active state of the inputs is determined by the **INL** command.

The Scan (**SN**) command determines how long the sequence select input must be maintained before the indexer executes the program. This delay is referred to as ***debounce time***. The following examples demonstrate how to use the Scan mode.

STEP ① Define a power-up sequence.

Command	Definition
> **XE1ØØ**	Erases sequence #100
> **XD1ØØ**	Defines sequence #100
SSJ1	Executes sequences via PLC input
SN5	Sets scan time to 5 msec
XQ1	Sets 500 to interrupted run mode
A1Ø	Sets acceleration to 10 rps^2
AD1Ø	Sets deceleration to 10 rps^2
V2	Sets velocity to 2 rps
IN1B	Sets Input 1 as a sequence-select input
IN2B	Sets Input 2 as a sequence-select input
IN3B	Sets Input 3 as a sequence-select input
IN4B	Sets Input 4 as a sequence-select input
IN5B	Sets Input 5 as a sequence-select input
IN6B	Sets Input 6 as a sequence-select input
IN7B	Sets Input 7 as a sequence-select input
IN8B	Sets Input 8 as a sequence-select input
OUT1C	Sets Output 1 as sequence-in-progress output
LD3	Disables the limits (if they are not connected)
> **XT**	Ends the sequence definition

Every time you power up the 500, it executes these commands and enables the 500 to read up to 99 sequences from the sequence-select inputs.

STEP ② Define any sequences that your application may require.

Command	Description
`> XE1`	Erases Sequence #1
`> XD1`	Defines Sequence #1
`D2ØØØ`	Sets distance to 2,000 steps
`G`	Executes the move (Go)
`> XT`	Ends Sequence #1 definition

Command	Description
`> XE2`	Erases Sequence #2
`> XD2`	Defines Sequence #2
`D4ØØØ`	Sets distance to 4,000 steps
`G`	Executes the move (Go)
`> XT`	Ends Sequence #2 definition

Command	Description
`> XE3`	Erases Sequence #3
`> XD3`	Defines Sequence #3
`D8ØØØ`	Sets distance to 8,000 steps
`G`	Executes the move (Go)
`> XT`	Ends Sequence #3 definition

Command	Description
`> XE99`	Erases Sequence #99
`> XD99`	Defines Sequence #99
`D-14ØØØ`	Sets distance to -14,000 steps
`G`	Executes the move (Go)
`> XT`	Ends Sequence #99 definition

STEP ③ Verify that your programs were stored properly by uploading each entered sequence (**XU**). If you receive responses that differ from what you programmed, re-enter those sequences.

STEP ④ Run each program from the RS-232C interface with the Run Sequence (**XR**) command. Make sure that the motor moves the distance that you specify.

STEP ⑤ Apply switches to the inputs that are normally-open (see Table 4-2).

Switch #	Input
Switch #1	IN1
Switch #2	IN2
Switch #4	IN3
Switch #8	IN4
Switch #10	IN5
Switch #20	IN6
Switch #40	IN7
Switch #80	IN8
GND	

Table 4-2. Applying Switches to Inputs

STEP ⑥ To execute sequences cycle power to the 500. The system will execute sequence #100.

STEP ⑦ You can now execute sequences by closing the corresponding switches.

❑ Close switch 1 to execute sequence #1
❑ Close switch 2 to execute sequence #2
❑ Close switches 1 & 2 to execute sequence #3
❑ Close switches 1, 8, 10, and 80 to execute sequence #99

Stopping Motion while in Seq. Scan Mode

The sequence scan mode can be activated by placing the **SSJ1** command into sequence #100 (power-up sequence). If you want to use a stop input or a **STOP** command in your sequences, then one of three methods of operation may be desired.

One method is to use the **STOP** command to stop a move and exit the sequence scan mode. In this method, an **SSJ1** command must be issued to re-enter the sequence scan mode. The current motion in the sequence being executed will be stopped and no more sequences will be executed, and the remaining commands in the sequence (after the motion) will not be executed.

A second method is to have the **STOP** command stop motion of the current move being executed and to complete execution of the remaining commands in that sequence. In this method, the sequence is finished although the move is stopped. This mode is enabled with the **SSH1** command. In this mode, sequence scanning will also stop, and must be re-enabled by issuing the **SSJ1**.

A third method is to stop motion, complete the current sequence, and to stay in the sequence scanning mode. In this method, the current move in the sequence is stopped, all subsequent commands in that sequence are executed, and when the sequence is complete the 500 again scans the inputs to execute the next valid sequence. This mode is enabled by issuing the **SSH1** and **OSI1** commands.

Thumbwheel Interface

The Model 500 allows two methods for thumbwheel use. One method allows you to wire your own thumbwheels. The other uses Compumotor's TM8 thumbwheel module.

With the 500, you can use up to 40 digits of thumbwheels. The TM8 requires a multiplexed BCD input scheme to read thumbwheel data. Therefore, a decode circuit must be used for thumbwheels. Compumotor recommends that you purchase Compumotor's TM8 module if you desire to use a thumbwheel interface. The TM8 contains the decode logic; therefore, only wiring is needed.

The 500 commands and format that allow for thumbwheel data entry are:

Command	Description
DRD	Read distance via thumbwheels
VRD	Read velocity via thumbwheels
LRD	Read loop count via thumbwheels
TRD	Read time delay via thumbwheels
VARDn	Read variables via thumbwheels
XRD	Read sequence count via thumbwheels
FRD	Read following ratio via thumbwheels

Using the TM8 Module

To use Compumotor's TM8 Module, follow the procedures below.

STEP ① To select the TM8 module, first type the **TW1** command to enable the *TM8 Mode*.

STEP ② Wire your TM8 module to the 500 as shown in Figure 4-13. If you are using more than one TM8 Module (the maximum allowed is six for each 500), wire the modules as in Figure 4-14.

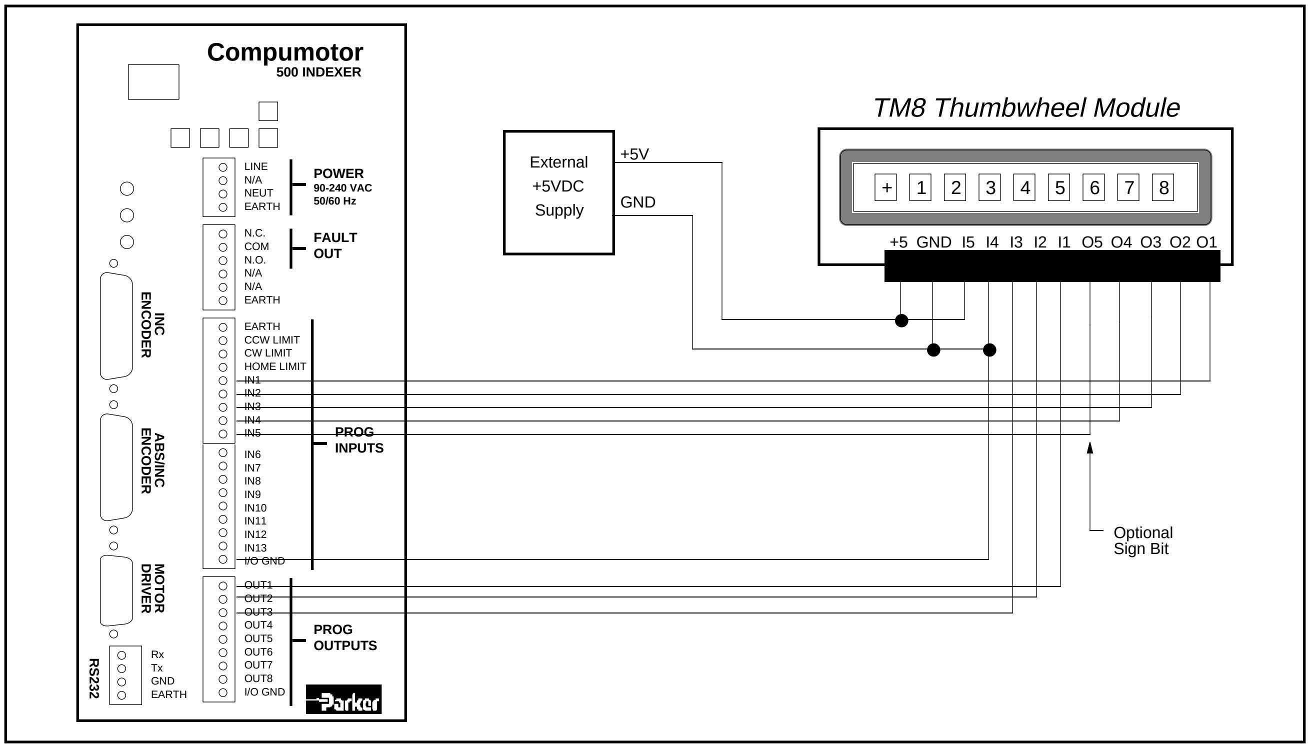


Figure 4-13. Wiring One TM8 Thumbwheel Module to the 500

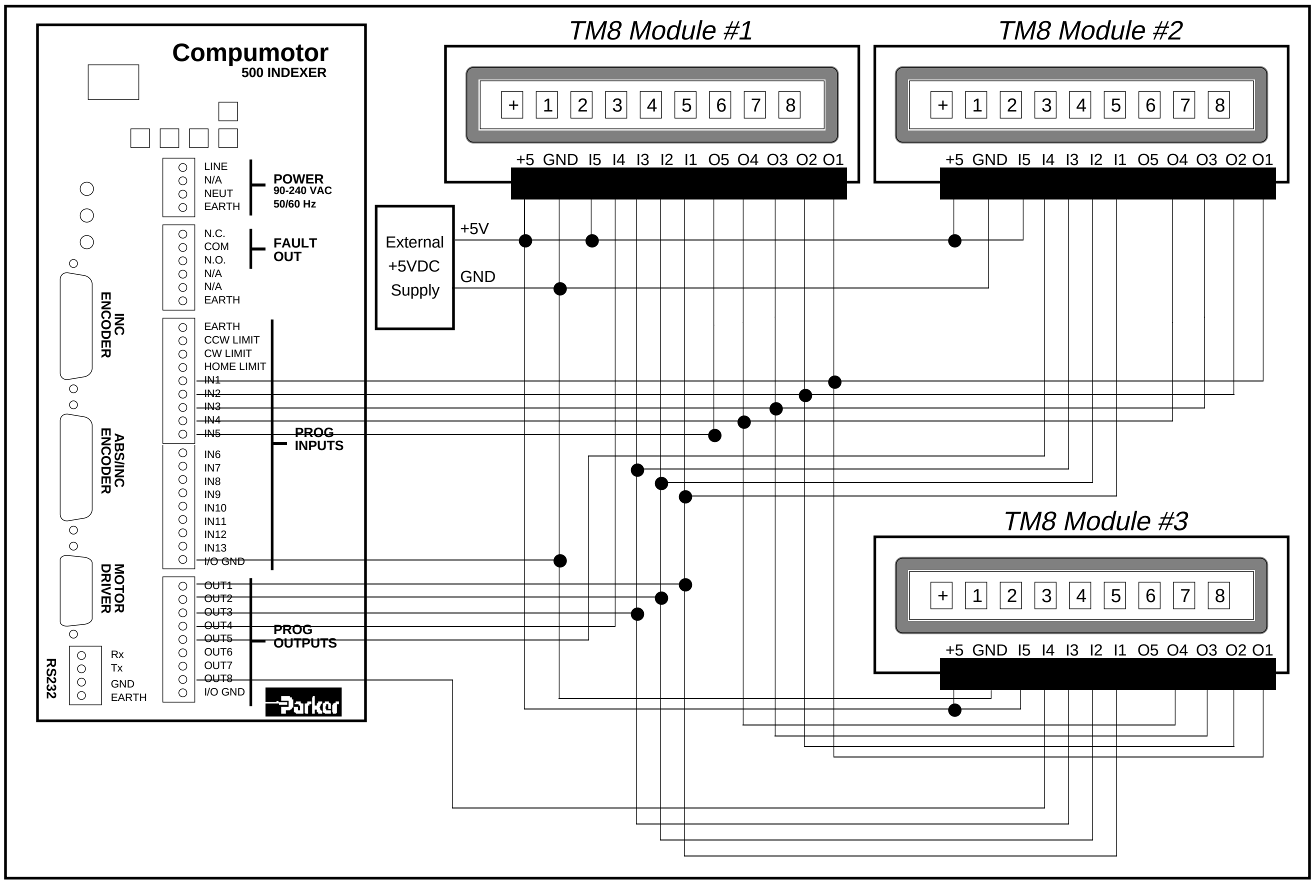


Figure 4-14. Wiring Multiple TM8 Thumbwheel Modules to the 500

STEP ③ Configure your 500 as follows:

Command	Description
`> OUT1J`	**OUT1** configured as a strobe
`> OUT2J`	**OUT2** configured as a strobe
`> OUT3J`	**OUT3** configured as a strobe
`> IN1N`	**IN1** configured as a data input
`> IN2N`	**IN2** configured as a data input
`> IN3N`	**IN3** configured as a data input
`> IN4N`	**IN4** configured as a data input
`> IN5W`	**IN5** configured as a sign input (optional)
`> INLØ`	Inputs configured active low
`> STR1Ø`	Data strobe time of 10 ms per digit read. If using one TM8 Module, you should now be ready to read in thumbwheel data. If using two TM8 Modules, enter the additional set-up commands. **The minimum strobe time recommended for the TM8 module is 10 ms**.
`> OUT4A`	**OUT4** configured as a programmable output
`> OUT5A`	**OUT5** configured as a programmable output
`> OUT8A`	**OUT8** configured as a programmable output
`> OUTL1`	Outputs set active high
`> OØ1XX1`	Set output 4 low—enables TM8 unit #1 if you are using multiple TM8s

STEP ④ Set the thumbwheel digits on your TM8 module to **+12345678** (*if using multiple TM8 modules, set the modules to some other value*). To verify that you have wired your TM8 module(s) correctly and configured your 500 I/O properly, enter the following commands:

Command	Description
`> DRD`	Request distance data from all 8 thumbwheel digits
`> 1D`	Displays the distance read—**`D+12345678.`** If you do not receive the response shown, return to step 1 and retry.

If you are using multiple TM8 modules, enter the following commands:

`> OØ1XX1`	Enables TM8 module #1 and disables modules #2 and #3
`> DRD`	Request distance data from all 8 thumbwheel digits
`> 1D`	Displays the distance read–**`D+12345678`** *The sign is positive*. This is because only one sign digit may be used when two TM8 modules are used. If you do not receive the response shown, return to step 1 and retry.
`> O11XX1`	Disables all TM8 modules if using multiple modules

Thumbwheel Example

The following thumbwheel example clarifies how to use the TM8 module. If using multiple modules, enter the following commands:

Command	Description
`> OØ1XX1`	Enables TM8 module #1 and disables modules #2 and #3
`> VARD1`	Reads the thumbwheels into Variable 1
`> O1ØXX1`	Enables TM8 module #2 and disables modules #1 and #3
`> VARD2`	Reads the thumbwheels into Variable 2
`> O11XXØ`	Enables TM8 module #3 and disables modules #1 and #2
`> VARD3`	Reads the thumbwheels into Variable 3

I4 on the TM8 module acts as an enable input that is active low. **I5** on the TM8 acts as an enable input that is active high. By using the first three Model 500 outputs (**OUT1** - **OUT3**) as *strobe* outputs for decoding the thumbwheel digits, the remaining outputs can be used for selecting multiple thumbwheels. In Figure 4-14, the active low input (**I4** on the TM8) functions as the *module select input*. One output line is dedicated to select a particular module. The three outputs that decode the digits are common to all modules used.

Using your own Thumbwheel Module

As an alternative to Compumotor's TM8 Module, you can use your own thumbwheels. To use you own thumbwheels, you must use 8 inputs for thumbwheel operation. Use the following steps to set up and read the thumbwheel interface. Refer to the **Model 500 Software Reference Guide** for detailed descriptions of the commands used below.

STEP ① Enable the PLC Mode with the `TWØ` command. The PLC mode allows you to use the 500 with a PLC or your own thumbwheels. The 500 provides certain commands that aide you in using a PLC. This will be covered later in the section entitled *PLC Operation*.

STEP ② Wire your thumbwheels according to the schematic in Figure 4-15.

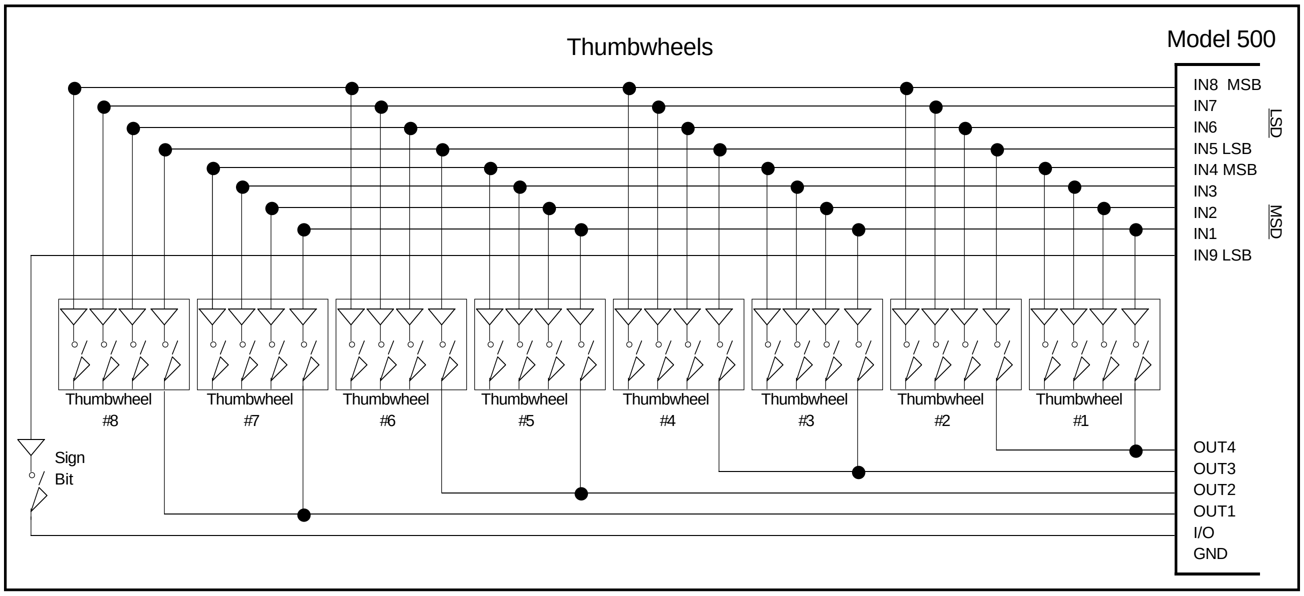


Figure 4-15. Thumbwheel Connections to the Model 500

STEP ③ Set up the inputs and outputs for operation with thumbwheels
The data valid input will be an input which the operator holds active to let the Model 500 read the thumbwheels. This input is not necessary; however, it is often used when interfacing with PLCs.

Command	Description
`> OUT1J`	Sets Output #1 (**OUT1**) as a programmable output
`> OUT2J`	Sets Output #2 (**OUT2**) as a programmable output
`> OUT3J`	Sets Output #3 (**OUT3**) as a programmable output
`> OUT4J`	Sets Output #4 (**OUT4**) as a programmable output
`> IN1N`	Sets Input #1 (**IN1**) as a data input
`> IN2N`	Sets Input #2 (**IN2**) as a data input
`> IN3N`	Sets Input #3 (**IN3**) as a data input
`> IN4N`	Sets Input #4 (**IN4**) as a data input
`> IN5N`	Sets Input #5 (**IN5**) as a data input
`> IN6N`	Sets Input #6 (**IN6**) as a data input
`> IN7N`	Sets Input #7 (**IN7**) as a data input
`> IN8N`	Sets Input #8 (**IN8**) as a data input
`> IN9W`	Sets Input #9 (**IN9**) as a data sign input
`> STR12`	Sets the strobe time to 12 milliseconds

STEP ④ Each output strobes in two digits at a time. This is how the thumbwheels are read. The sign bit is optional. Set the thumbwheels to `+12345678` and type in the following commands:

Command	Description
`> DRD`	Reads the thumbwheels into the distance parameter
`> 1D`	Reports the distance
`> VARD1`	Reads the thumbwheels into Variable 1
`> 1VAR1`	Reports Variable 1

Selecting Sequences with the Thumbwheel Module

The following example shows how the Model 500 can be used to execute sequences with remote thumbwheels. In the example, only 3 sequences are entered. As many as 100 sequences may be defined and up to 99 may be executed using remote thumbwheels. Sequence 100 is automatically executed during power up or reset. Refer to the Reset (`Z`) command in the **Model 500 Software Reference Guide**. Install your thumbwheels as shown in Figure 4-16.

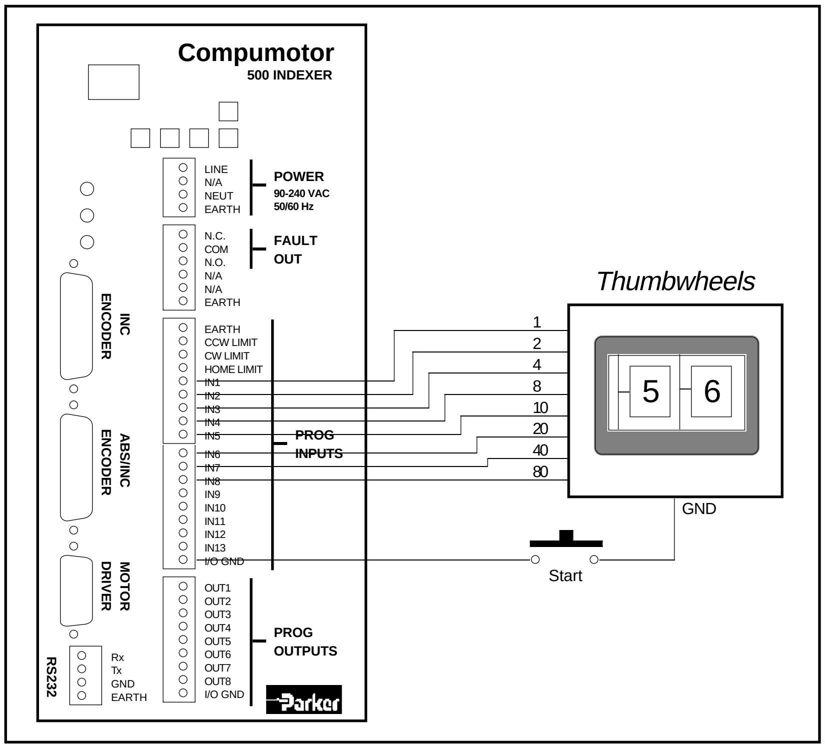


Figure 4-16. Model 500 Thumbwheel Installation

The inputs are defines as *sequence select* inputs rather than *data* inputs. The 500 is used in the sequence scan mode (set up with the `SSJ1` command). When the start button is pushed, the input lines are opened and closed according to the thumbwheel lines.

STEP ① Define a power-up sequence. Set up inputs **IN1** - **IN8** as sequence-select inputs using the Configure Input (**IN**) commands. The **IN1** command is the least significant bit of the 1s digit. The **IN8** command is the most significant digit of the 10s digit. The BCD values of inputs **IN1** - **IN8** are shown in Table 4-3.

Command	Description
> **XE1ØØ**	Erases sequence #100
> **XD1ØØ**	Defines sequence #100
IN1B	Sets up Input #1 as the sequence input
IN2B	Sets up Input #2 as the sequence input
IN3B	Sets up Input #3 as the sequence input
IN4B	Sets up Input #4 as the sequence input
IN5B	Sets up Input #5 as the sequence input
IN6B	Sets up Input #6 as the sequence input
IN7B	Sets up Input #7 as the sequence input
IN8B	Sets up Input 8 as the sequence input
INLØ	Sets active input level to active low (ON = ØV applied to the input)
SSJ1	Set Model 500 to run programs from remote interface
SN1Ø	Set scan time to 10 msec
XQ1	Enable sequence hold
> **XT**	Ends sequence definition

Inputs	**IN8**	**IN7**	**IN6**	**IN5**	**IN4**	**IN3**	**IN2**	**IN1**
BCD Value	80	40	20	10	8	4	2	1
	10's Digit				***1's Digit***			

Table 4-3. Input BCD Values

STEP ② Define any sequences that your application may need.

Command	Description
> **XE1**	Erases sequence #1
> **XD1**	Starts sequence #1 definition
MN	Sets mode to normal
A25	Sets acceleration to 25 rps
AD25	Sets deceleration to 25 rps
V5	Sets velocity to 5 rps
D25ØØØ	Sets distance to 25,000 steps
G	Executes the move (Go)
> **XT**	Ends sequence definition

Command	Description
> **XE5**	Erases sequence #5
> **XD5**	Starts sequence #5 definition
MN	Sets mode to normal
A25	Sets acceleration to 25 rps^2
AD25	Sets deceleration to 25 rps
V5	Sets velocity to 5 rps
D1ØØØØ	Sets distance to 10,000 steps
G	Executes the move (Go)
> **XT**	Ends sequence definition

Command	Description
> **XE99**	Erases sequence #99
> **XD99**	Starts sequence #99 definition
MN	Sets mode to normal
A25	Sets acceleration to 25 rps^2
AD25	Sets deceleration to 25 rps
V5	Sets velocity to 5 rps
D-35ØØØ	Sets distance to -35,000 steps
G	Executes the move (Go)
> **XT**	Ends sequence definition

STEP ③ Connect the thumbwheels to the **IN1** - **IN8** inputs as shown in Figure 4-16.

STEP ④ Reset the Model 500 Indexer. This will prepare the power up (Sequence #100) to be executed when you apply power to the indexer.

Command	Description
`> Z`	Resets the Model 500

STEP ⑤ Set your thumbwheel to 1 and push start to move the motor 25,000 steps in the positive direction.

STEP ⑥ Set your thumbwheel to 5 and push start to move the motor 10,000 steps in the positive direction.

STEP ⑦ Set your thumbwheel to 99 and push start to move the motor 35,000 steps in the negative direction.

If you select (with the thumbwheel) an invalid or unprogrammed sequence, nothing will happen.

PLC Operation

This section explains and provides examples of how to use a PLC with the 500 Indexer/Drive.

Interfacing with a PLC

In many applications it is desirable to interface to a PLC. The 500 performs the motion segment of a more involved process controlled by a PLC. In these applications, the PLC will execute sequences, load data, manipulate inputs and perform other specific input functions to control 500 and the motion segment of a process.

The PLC is selected with the **TWØ** command. It requires that eight of the inputs are used for data entry. The data is always provided to the 500 from the PLC in a BCD format. The PLC can be used to select sequences in conjunction with the sequence scan mode (**SSJ1**), or, as with thumbwheels, the PLC can be used to enter data using the data read commands (**XRD**, **DRD**, **VRD**, **LRD**, **TRD**, or **VARD**).

Data Read with a PLC

As in the thumbwheel case described earlier, the PLC can be used to enter data for sequence select (**XRD**), distance (**DRD**), velocity (**VRD**), loop count (**LRD**), time delay (**TRD**), and variable data (**VARD**). Detailed explanations of these commands are provided in the **Model 500 Software Reference Guide**.

When using data read commands, the data is read in the same manner as when using your own thumbwheels. Each strobe output selects two BCD digits of data. Multiple strobe outputs select multiple digit pairs.

The timing for entering data can be controlled in two ways: (1) setting the strobe time with the **STR** command, and (2), defining a Model 500 input as a *data valid* input.

CONTROL TIMING — *STROBE DURATION*

The first method of controlling timing for entering data is to use the **STR** command to set the length of time for which the strobe outputs stay active. Using this method, the strobe time would be set higher than the PLC's scan time. The PLC interprets the strobe and places data on the data lines. The 500 waits for the duration of the strobe time (**STR** setting), then reads the data and issues the next strobe. This is done internally in the 500 until it has issued all strobes that are defined. Of outputs #1 - #4 are defined as strobes, the 500 issues the strobes in the order of **OUT1**, **OUT2**, **OUT3**, and then **OUT4**.

To read data from the PLC, the 500's inputs #1 - #8 (**IN1 - IN8**) must be configured as data inputs with an active low level (**INLØ**).

If a sign digit is required, 500 input #9 should be configured as a sign input. Configure outputs #1 - #4 as data strobe outputs.

When the 500 executes a data read command, it will cycle its outputs and read BCD data as shown in Table 4-4.

Model 500 Outputs				**Model 500 Inputs**							
0UT1	**0UT2**	**0UT3**	**0UT4**	**IN1**	**IN2**	**IN3**	**IN4**	**IN5**	**IN6**	**IN7**	**IN8**
low	high	high	high	Digit 8 LSB	Digit 8	Digit 8	Digit 8 MSB	Digit 7 LSB	Digit 7	Digit 7	Digit 7 MSB
high	low	high	high	Digit 6 LSB	Digit 6	Digit 6	Digit 6 MSB	Digit 5 LSB	Digit 5	Digit 5	Digit 5 MSB
high	high	low	high	Digit 4 LSB	Digit 4	Digit 4	Digit 4 MSB	Digit 3 LSB	Digit 3	Digit 3	Digit 3 MSB
high	high	high	low	Digit 2 LSB	Digit 2	Digit 2	Digit 2 MSB	Digit 1 LSB	Digit 1	Digit 1	Digit 1 MSB

Table 4-4. Output Voltage Levels

Figure 4-17 shows a possible PLC-to-500 connection that would allow the 500 to input data from the PLC.

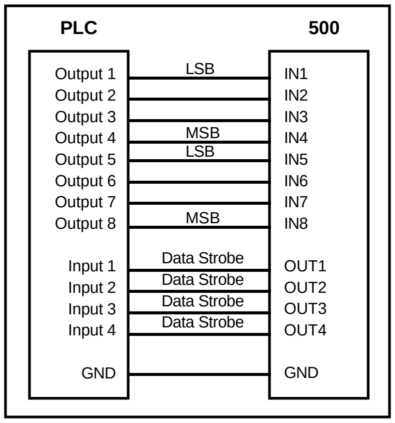


Figure 4-17. PLC-to-500 Connection

If the 500 executes a Read Distance Via Parallel I/O (**DRD**) command, the following events would have to occur to transfer distance data from the PLC to the 500.

① The 500 executes a **DRD** command and turns on **OUT1**.

② The PLC places and holds 2 BCD digits (digits 7 & 8) at the 500's **IN1** - **IN8**. (This value will be the most significant distance digit). The PLC places a sign value at the 500's **IN9**. (***This sign bit must be the same for each digit read***.)

③ After the strobe time has elapsed, the 500 will read the digits and sign values.

④ The 500 will place its **OUT1** at high voltage (5 - 24V) and and will turn on **OUT2**.

⑤ The PLC must place and hold its next set of BCD digits (digits 5 & 6) at the 500's **IN1** - **IN8**. The PLC should place the same sign value as that given for **IN9**.

⑥ After the strobe time has elapsed, the 500 will read this second set of BCD digits as the third and fourth significant distance digits.

This process continues (see Table 4-4) until the 500 reads the eighth digit (LSD). At this point, the 500 enters the eight digits read into its distance register and proceeds with the execution of subsequent commands.

CONTROL TIMING — *DATA VALID INPUT*

The second method of controlling timing is to define a 500 input as a *data valid* input using the **IN** command. In this case, the 500 will issue a strobe, then the PLC will place the data onto the data lines and toggle the data valid input to the 500. The 500 will not read the data until the data valid is issued. The 500 will then read the data and issue the next strobe.

For example, input #10 (**IN10**) could be defined as a data valid input (input function V) with the **IN1ØV** command. After each strobe from the 500, the PLC places two digits on the data lines, then issues data valid.

Sequences can be selected and executed using the **XRD** command. The next section explains an alternative to using the data entry format.

Sequence Select With a PLC

As described in the previous sections, you can execute sequences by reading the sequence to be executed from data inputs. As an alternative, you could also execute sequences using a PLC by defining the inputs as *sequence select* inputs.

Again, the inputs are BCD weighted. As explained earlier in the *Sequence Select & Sequence Scanning* section, you must use the sequence scan (**SSJ1**) mode. Different modes of sequence scanning are accomplished using the **XQ**, **SSH**, and **OSI** commands (refer to the **<u>Model 500 Software Reference Guide</u>**).

The 500 inputs are <u>not</u> *sinking inputs* (refer to Chapter 6, *Hardware Reference*).

The PLC activates the appropriate inputs to execute the desired sequence. At the end of each sequence, you can program the 500 to turn on an output (using the **O** command) to indicate to the PLC to select the next sequence.

The following are step-by-step procedures to run sequences from your PLC. First, you need to enter the programs into the Model 500. You will need a terminal or a computer with RS-232C communication capability. You need to define the sequences before you can execute them with your PLC's BCD outputs. The Model 500 indexer automatically saves these sequences (battery backed-up system).

STEP ① Define a power-up sequence.

Command	Definition
> **XE1ØØ**	Erases sequence #100
> **XD1ØØ**	Defines sequence #100
SSJ1	Executes sequences via PLC input
SN2Ø	Sets scan time to 20 msec
XQ1	Sets Model 500 to interrupted run mode
A1Ø	Sets acceleration to 10 rps^2
AD1Ø	Sets deceleration to 10 rps^2
V2	Sets velocity to 2 rps
IN1B	Sets Input 1 as a sequence-select input
IN2B	Sets Input 2 as a sequence-select input
IN3B	Sets Input 3 as a sequence-select input
IN4B	Sets Input 4 as a sequence-select input
IN5B	Sets Input 5 as a sequence-select input
IN6B	Sets Input 6 as a sequence-select input
IN7B	Sets Input 7 as a sequence-select input
IN8B	Sets Input 8 as a sequence-select input
OUT1C	Sets Output 1 as sequence-in-progress output
LD3	Disables the limits (if they are not connected)
> **XT**	Ends the sequence definition

Every time you power up the Model 500, it executes these commands and enables the Model 500 to read up to 99 sequences from the sequence-select inputs.

STEP ② Define any sequences that your application may require.

Command	Description
> **XE1**	Erases Sequence #1
> **XD1**	Defines Sequence #1
D2ØØØ	Sets distance to 2,000 steps
G	Executes the move (Go)
> **XT**	Ends Sequence #1 definition

Command	Description
> **XE2**	Erases Sequence #2
> **XD2**	Defines Sequence #2
D4ØØØ	Sets distance to 4,000 steps
G	Executes the move (Go)
> **XT**	Ends Sequence #2 definition

Command	Description
> **XE3**	Erases Sequence #3
> **XD3**	Defines Sequence #3
D8ØØØ	Sets distance to 8,000 steps
G	Executes the move (Go)
> **XT**	Ends Sequence #3 definition

Command	Description
> **XE99**	Erases Sequence #99
> **XD99**	Defines Sequence #99
D-14ØØØ	Sets distance to -14,000 steps
G	Executes the move (Go)
> **XT**	Ends Sequence #99 definition

STEP ③ Verify that your programs were stored properly by uploading each entered sequence (**XU**). If you receive responses that differ from what you programmed, re-enter those sequences.

STEP ④ Run each program from the RS-232C interface with the Run Sequence (**XR**) command. Make sure that the motor moves the distance that you specify.

STEP ⑤ Assuming your PLC accepts open-collector outputs connect the inputs and outputs as shown in Figure 4-18. If not, you will need to add pull-up resistors to the outputs.

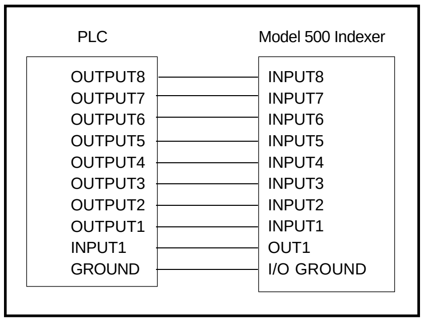


Figure 4-18. PLC Connection

STEP ⑥ Refer to the user guide that accompanied you PLC unit to turn on the proper combination of outputs to execute one of the four sequences programmed and stored in the Model 500 indexer. You should program the PLC to search for the sequence completion output from the 500 before the 500 begins searching for the next sequence to execute; this type of interaction is performed through a *handshake* between the 500 and the PLC — the signal transmitted indicates that another sequence is ready to be executed.

- ❑ Turn on Output 1 (only) to execute sequence #1.
- ❑ Turn on Output 2 (only) to execute sequence #2.
- ❑ Turn on Outputs 1 and 2 (only) to execute sequence #3.
- ❑ Turn on Outputs 1, 4, 5, and 8 (only) to execute sequence #99.

STEP ⑦ Cycle down power to the Model 500. The system will execute sequence 100.

STEP ⑧ Turn on the appropriate sequence-select input [set for the least Scan Time (**`SN`**)] to execute the proper sequences. Since the sequence hold feature (refer to the **`XQ1`** command) was enabled during the power-up sequence, your PLC program must turn off all of the sequence-select inputs before you can select another sequence.

Miscellaneous Control by a PLC

You can use a PLC to control the activation of the inputs for all of the input functions that the 500 supports (see the *Programmable Input* section). For example, you can use the PLC to stop, kill, go, go home, or reset the 500.

Front Panel Operation

The default function of the front panel pushbuttons is to display errors and to change baud rate, device address, and the gain for position maintenance.

Procedures for changing the baud rate and device address and checking the revision level are provided in Chapter 2, *Getting Started*. To change the proportional gain, use the following procedure:

STEP ① Display the proportional gain by pressing the **1s** and **ENTER** pushbuttons simultaneously (see Figure 4-19).

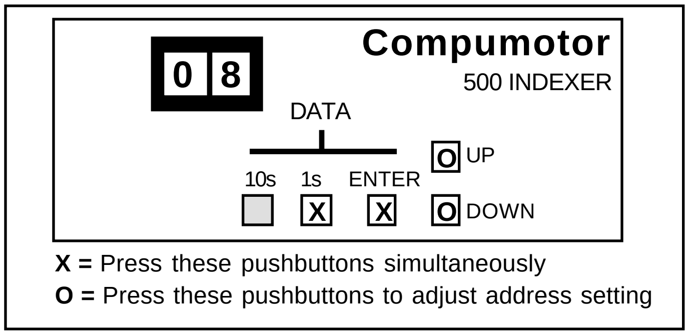

0 8
Compumotor
500 INDEXER
DATA
10s
1s
ENTER
UP
DOWN
X = Press these pushbuttons simultaneously
O = Press these pushbuttons to adjust address setting

Figure 4-19. Changing Proportional Gain Via Front Panel Pushbuttons

STEP ② While pressing the **1s** and **ENTER** pushbuttons simultaneously, you can use either the **UP** or the **DOWN** pushbuttons to increase or decrease the gain.

Executing Sequences via Pushbuttons

You can also use the pushbuttons to execute sequences. To use this feature you must issue the following commands:

<u>Command</u>	<u>Description</u>
`> SSO1`	Enables pushbuttons to function as sequence-select buttons
`> CPB1`	Enables mode for executing sequences via front panel pushbuttons

Use the following procedure to select a sequence number:

STEP ① Press and hold down the appropriate button (**10s** or **1s**) for the respective digit in the sequence number. In this mode of operation, the **10s** button represents the 10s digit of the sequence, and the **1s** button represents the 1s digit of the sequence.

STEP ② While pressing the **10s** or **1s** button, press the **UP** or **DOWN** button to select the proper digit.

STEP ③ Once you have the proper sequence number displayed, press the **ENTER** button to execute the sequence.

Summary of Pushbutton Features

Table 4-5 lists the functions of the pushbuttons. The **X** symbol means push the pushbutton; the **Ø** symbol means do not push the pushbutton.

10's	1's	ENTER	UP	DOWN	Function
Ø	Ø	Ø	Ø	Ø	Display Errors
X	Ø	Ø	Ø	Ø	Display Sequence* or Following ratio
X	Ø	Ø	X	Ø	Increase Sequence* or Following
X	Ø	Ø	Ø	X	Decrease Sequence* or Following
Ø	X	Ø	Ø	Ø	Display Sequence* or Following
Ø	X	Ø	X	Ø	Increase Sequence* or Following
X	X	Ø	Ø	X	Decrease Sequence* or Following
Ø	Ø	X	Ø	Ø	Execute Sequence* or Enter Following
Ø	Ø	X	X	Ø	Reserved
Ø	Ø	X	Ø	X	Reserved
X	X	Ø	Ø	Ø	Display RS-232 Baud Rate
X	X	Ø	X	Ø	Increase RS-232 Baud Rate
X	X	Ø	Ø	X	Decrease RS-232 Baud Rate
X	Ø	X	Ø	Ø	Display RS-232 Device Address
X	Ø	X	X	Ø	Increase RS-232 Device Address
X	Ø	X	Ø	X	Decrease RS-232 Device Address
Ø	X	X	Ø	Ø	Display Proportional Gain
Ø	X	X	X	Ø	Increase Proportional Gain
Ø	X	X	Ø	X	Decrease Proportional Gain
X	X	X	Ø	Ø	Display Software Rev.Level
X	X	X	X	Ø	Reserved
Ø	Ø	Ø	X	X	Reset
X	X	X	X	Ø	Reserved

* Function applicable only if `SSO1` and `CPB1` commands have been entered for executing sequences via front panel pushbuttons

Table 4-5. Model 500 Pushbutton Features

Remote Panel Operation

The 500 is compatible with the Compumotor RP240 remote front panel. The RP240 has a 2-line 40-character display, a numeric keypad, and programmable function keys. The RP240 communicates with the 500 over RS-232C and allows you to input data, display messages, and prompt the user for appropriate actions or data input. For more information on the RP240, refer to the ***Model RP240 User Guide***, or contact your local distributor or Compumotor.

Host Computer Operation

Another choice for a user interface is to use a host computer and execute a motion program using the RS-232C serial interface. Two types of operation can be performed when using a host computer. The host can run interactively from a Basic or C program. In this case, the high-level program controls the 500 and acts as an interface to the user. The following program is an example of this type of operation.

```
1 '                    500.BAS PROGRAM
2 '
3 ' ***************************************************************************************
4 ' *                                                                                      *
5 ' * This program controls the RS232 Communication line to execute 2 different moves      *
6 ' * using the 500                                                                        *
7 ' *                                                                                      *
*****************************************************************************************
15 OPEN "COM1:96ØØ,N,8,1,RS,CS,DS,CD" AS #1        ' Open Communication port
2Ø V$ = "": Q$ = "": ECHO$ = "": LF$ = "":         ' Initialize variables
9Ø CLS
1ØØ LOCATE 12,15
1Ø5 PRINT " PRESS ANY KEY TO START THE PROGRAM "
1Ø7 V$ = INKEY$: IF LEN(V$) = Ø THEN 1ØØ           ' Wait for input from user
12Ø Z$ = "K K LD3"                                 ' Reset and disable limits on the 500
122 PRINT #1,Z$;
124 Q$ = INPUT$(8,1)
9ØØ ' **************************************************************************************
9Ø1 ' *                                                                                     *
9Ø2 ' *  Lines 1ØØØ-1Ø6Ø sends a move down to the first 500.  Computer waits for the Line   *
9Ø3 ' *  Feed from the 500 indicating that the motor has finished its move.  Computer       *
9Ø4 ' *  will not command second 500 to move until the first move is completed.           *
9Ø5 ' *                                                                                     *
*****************************************************************************************
1ØØØ MOVE1$ = "MN A1 V2 D4ØØØØ G 1LF "             ' Define move 1
1ØØ5 CLS
1ØØ7 LOCATE 12,15: PRINT " DOING MOVE 1 "
1Ø1Ø PRINT #1,MOVE1$                               ' Perform move 1.
1Ø15 ECHO$ = INPUT$(23,1)                          ' Read echoes from 500.
1Ø2Ø LF$ = INPUT$(1,1)                             ' Wait for line feed from 500
1Ø4Ø IF LF$ <> CHR$(1Ø) GOTO 1Ø2Ø             ' indicating end of move.
1Ø45 CLS
1Ø47 LOCATE 12,15
1Ø5Ø PRINT "MOVE 1 DONE"                           ' Let user know that move 1 is done
1Ø6Ø LOCATE 15,15: PRINT " PRESS ANY KEY TO GO ON TO SECOND MOVE "
1Ø7Ø V$ = INKEY$: IF LEN(V$) = Ø THEN 1Ø6Ø
19ØØ '
*************************************************************************************
19Ø1 ' *                                                                                *
19Ø2 ' * After axis one is done, we request that you hit any key to go on to second move.   *
19Ø3 ' * In real application, we would expect you to go ahead with the process and work on  *
19Ø4 ' * on the part before going on to next move.  (i.e., Activate a punch)               *
19Ø5 ' * Now that first move is finished go on to move 2.  The 500 also prints a line feed  *
19Ø6 ' * after finishing the second move.                                                 *
19Ø7 ' * As soon as the computer receives the line feed from 500, the program will go back  *
19Ø8 ' * to the first move.                                                               *
19Ø9 ' *                                                                                *
*************************************************************************************
2ØØØ MOVE2$ = "H G 1LF "
2ØØ5 CLS
2ØØ7 LOCATE 12,15: PRINT " DOING MOVE 2 "
2Ø1Ø PRINT #1,MOVE2$
2Ø15 ECHO$ = INPUT$(8,1)
2Ø2Ø LF$ = INPUT$(1,1)
2Ø4Ø IF LF$ <> CHR$(1Ø) GOTO 2Ø2Ø
2Ø45 CLS
2Ø47 LOCATE 12,15
2Ø5Ø PRINT "MOVE 2 DONE "
2Ø6Ø FOR I = 1 TO 1ØØØ: NEXT I
2Ø7Ø GOTO 2Ø                                       ' Go back to beginning of program.
```

The second method is for the 500 to run a stored program and prompt the user interactively as a part of the program. To do this, you must use the RS-232C Input (**RSIN**) and Quote (**"**) commands. The Quote command sends messages over the RS-232C link to the host. The message may be status data or a request for information. If it is a request, **RSIN** allows you to enter data into any of the general-purpose variables. The motion program may use these variables. The following program is an example of a stored sequence that interactively requests the number of parts and the size.

Command	Description
`> 1XE1`	Erase sequence #1
`> 1XD1`	Define sequence #1
`1"Enter_the_number_of_parts_to_be_made`	Send string to host
`VAR1=RSIN`	Set variable #1 to the value input by the user
`1"Enter_the_size_of_the_parts`	
`VAR2=RSIN`	Set variable #2 to the value input by the user
`VAR2=VAR2*25000`	Scale variable #2 by 25000 for user units
`D(VAR2)`	Set distance (part size) to variable #2
`L(VAR1)`	Set loop count (part count) to variable #1
`G`	Execute move
`N`	Continue loop
`> XT`	Terminate sequence

The user inputs data by typing an exclamation mark (**!**) followed by the data.

Multi-axis Control (Daisy-chaining)

You may daisy-chain up to 16 Model 500s to one serial port on a terminal. Figure 4-20 shows a three-indexer daisy-chain configuration from one controlling terminal or computer.

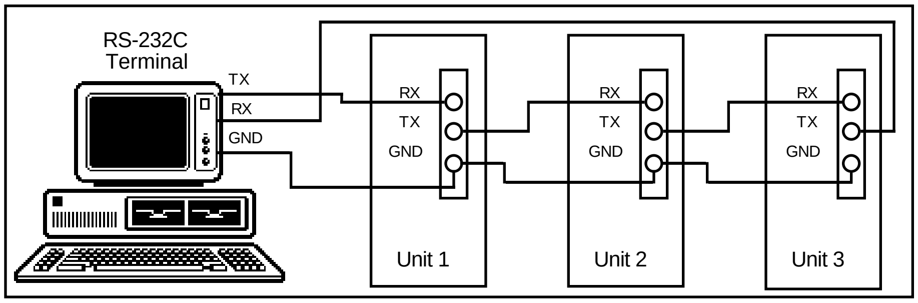


Figure 4-20. RS-232C Daisy-Chain Configuration

If you are daisy-chaining 500s, you should establish different addresses for each unit so you can distinguish them when you are programming. Refer to Chapter 2, *Getting Started*, for instructions on displaying and changing the 500's device address.

Commands prefixed with a device address affect only the unit specified. Commands without a device address affect all units on the daisy chain. For instance, entering the **1G** command instructs only unit #1 to go (move), but entering the **G** command instructs all units on the daisy-chain to go.

Any command that causes the drive to transmit information from the RS-232C port (such as a status or report command), must be prefixed with a device address. This prevents daisy-chained units from all transmitting at the same time.

No 500 executes a device-specific command unless the unit number specified with the command matches the 500's unit number. Device-specific commands include both buffered and immediate commands.

You must use status request commands in an orderly fashion. Commands should only be issued when the host is ready to read the response. You should not send new commands until you receive a response from the previous status request command. In particular, you should not issue an immediate-status command until the host receives a buffered command status response. If this is not followed, the command responses will be intertwined, rendering the data useless.

If you enable the Interactive mode (**SSI1**), only the 500 that is set to address 1 will respond with a prompt (>). This prevents all the 500s from sending out > in a daisy-chain. Typically, you should disable the Interactive mode when you use a host computer with the 500.

Sample Application and Commands

The following move example is for three indexers on an RS-232C daisy-chain.

Command	Description
> **MN**	Sets unit to Preset mode
> **A5**	Sets acceleration to 5 rps^2 for all three controllers
> **V1Ø**	Sets velocity to 5 rps for all three controllers
> **LD3**	Disables limits (if they are not connected)
> **1D25ØØØ**	Sets Axis 1 distance to 25,000 steps
> **2D5ØØØØ**	Sets Axis 2 distance to 50,000 steps
> **3D1ØØØØØ**	Sets Axis 3 distance to 100,000 steps
> **G**	Moves all axes.

Unit 1 moves 25,000 steps, unit 2 moves 50,000 steps, and unit 3 moves 100,000 steps. All three units use the same acceleration and velocity rates. All three units begin movement at about the same time.

Communicating with Other Compumotor Indexers

Because of their common programming language, Compumotor Model 500s, SXs, and ZXs can communicate with each other over a daisy-chain using the **TX** command. The **TX** command allows one unit to send a command to another unit with the value in a variable appended to the command. The **Model 500 Software Reference Guide** provides a detailed description of the **TX** command.

For example, if a 500 and an SX are daisy-chained, only one set of thumbwheels is needed to enter data into both units. One 500 can serve as the *master*, reading data in via the thumbwheels. The data is read into a variable. This data can then be sent the SX as a distance command. The commands for this example are as follows:

Command	Description
> **1VARD1**	Unit 1 (one of the 500s) reads the thumbwheels: **VAR1=125.ØØØØØ**
> **1VAR1=VAR1*25ØØØ**	Convert to motor pulses (25000 pulses/inch)
> **TX1,Ø,Ø,2D**	**2D3125ØØØ** is transmitted to unit 2 (the SX)

Multi-Axis Interface Program Example

The following program is very similar to **Model 500.BAS**, except this program controls two Model 500s on a daisy-chain. This program assumes the device address of two Model 500s to be #1 and #2 respectively. The program does the following:

① Executes the first move upon user input

② Waits for a line feed from the Model 500 indexer, which indicates the end of the move.

③ Executes the second move upon user input.

④ Waits for a line feed from the Model 500 indexer, which indicates the end of the move. It then begins the process again.

```
1 '                    500-2.BAS PROGRAM
2 '
3 ' ************************************************************************************
4 ' *                                                                                  *
5 ' * This program controls the RS232 Communication line to execute 2 different moves using *
6 ' * two Model 500 drives.                                                            *
7 ' *                                                                                  *
***************************************************************************************
15 OPEN "COM2:96ØØ,N,8,1,RS,CS,DS,CD" AS #1        ' Open Communication port
2Ø V$ = "": Q$ = "": ECHO$ = "": LF$ = "":         ' Initialize variables
9Ø CLS
1ØØ LOCATE 12,15
1Ø5 PRINT " PRESS ANY KEY TO START THE PROGRAM "
1Ø7 V$ = INKEY$: IF LEN(V$) = Ø THEN 1ØØ           ' Wait for input from user
12Ø Z$ = "K LD3"                                          ' Reset the Model 500 indexer
122 PRINT #1,Z$;
124 Q$ = INPUT$(8,1)
9ØØ ' *********************************************************************************
9Ø1 ' *                                                                               *
9Ø2 ' *  Line 1ØØØ-1Ø6Ø sends a move down to the first Model 500.  Computer             *
9Ø3 ' *  waits for the Line Feed from the Model 500 indicating that the motor         *
9Ø4 ' *  has finished its move.   Computer will not command second Model 500          *
9Ø5 ' *  until axis 1 has completed its move.                                          *
9Ø6 ' *                                                                               *
***************************************************************************************
1ØØØ MOVE1$ = "1A1 1V2 1D4ØØØØ 1G 1LF " ' Define move for Axis 1
1ØØ5 CLS
1ØØ7 LOCATE 12,15: PRINT " MOVING AXIS 1 "
1Ø1Ø PRINT #1,MOVE1$                              ' Move axis 1.
1Ø15 ECHO$ = INPUT$(23,1)                         ' Read echoes from Model 500
1Ø2Ø LF$ = INPUT$(1,1)                            ' Wait for line feed from Model 500
1Ø4Ø IF LF$ <> CHR$(1Ø) GOTO 1Ø2Ø          ' indicating end of move.
1Ø45 CLS
1Ø47 LOCATE 12,15
1Ø5Ø PRINT "AXIS 1 FINISHED ITS MOVE "             ' Let user know axis 1 done
1Ø6Ø LOCATE 15,15: PRINT " PRESS ANY KEY TO MOVE SECOND AXIS "
1Ø7Ø V$ = INKEY$: IF LEN(V$) = Ø THEN 1Ø6Ø
19ØØ '
***************************************************************************************
19Ø1 ' *                                                                              *
19Ø2 ' * After axis one is done, we request that you press any key to go on           *
19Ø3 ' * to second move.  In real application, we would expect you to                *
19Ø4 ' * go ahead with the process and work on the part before going on to           *
19Ø5 ' * next move.  (i.e.,  Activate a punch)                                         *
19Ø6 ' *                                                                              *
19Ø7 ' * Now that first axis finished its move, we go on to move axis 2.              *
19Ø8 ' * Second Model 500 also prints a line feed after finishing the move.           *
19Ø9 ' * As soon as computer receives the line feed from Model 500, program will      *
191Ø ' * go back to the first move.                                                   *
1911 ' *                                                                              *
***************************************************************************************
2ØØØ MOVE2$ = "2A1 2V2 2D1ØØØØ 2G 2LF "
2ØØ5 CLS
2ØØ7 LOCATE 12,15: PRINT " AXIS 2 MOVING "
2Ø1Ø PRINT #1,MOVE2$
2Ø15 ECHO$ = INPUT$(23,1)
2Ø2Ø LF$ = INPUT$(1,1)
2Ø4Ø IF LF$ <> CHR$(1Ø) GOTO 2Ø2Ø
2Ø45 CLS
2Ø47 LOCATE 12,15
2Ø5Ø PRINT "AXIS 2 FINISHED ITS MOVE "
2Ø6Ø FOR I = 1 TO 1ØØØ: NEXT I
2Ø7Ø GOTO 2Ø                                       ' Go back to beginning of program.
```

Model 500 Application Examples

This section provides application examples which illustrate the features and capabilities discussed in this chapter.

Application Development Approach

This section will help you to develop your application program. These steps provide a guideline for identifying and organizing your program's needs. Examples of actual programs demonstrate how the different programming concepts are used together. Use the following steps:

① Identify and define the types of motion required in your application. Examples are as follows:

- ❏ Moves to a home switch; go home moves
- ❏ Jogging the motor
- ❏ Preset moves of triangular or trapezoidal velocity profiles
- ❏ Registration moves
- ❏ Moves with changing velocities not based on distance
- ❏ Moves of changing velocities based on distance
- ❏ Motion based on math calculations
- ❏ Motion with outputs turning on at distance points

② Determine what will cause the motion to occur and when it will cause motion to occur. Choose the interface(s) for execution of the motion programs. Examples are as follows:

- ❏ Input states causing motion to occur; triggers
- ❏ Sequence execution
- ❏ Thumbwheel inputs
- ❏ Power up sequence
- ❏ PLC
- ❏ Variable conditions
- ❏ Time delays

③ Determine what configuration commands are necessary to set the 500 in the proper state for execution of the motion programs. Examples are as follows:

- ❏ Sequence scan mode—**SSJ1**, **XQ1**, **SSH1**
- ❏ Input/Output configuration—**IN1A**, **IN4D**, **OUT3J**
- ❏ Motion parameters—**A**, **AD**, **V**, **D**
- ❏ Profile commands—**MC**, **MN**, **MPI**, **MPA**, **MPP**
- ❏ Homing parameters—**GHA**, **GHV**, **GHF**, **GHAD**, **OSB**, **OSG**, **OSH**
- ❏ Other special functions of the **SS** and **FS** commands

Once you have determined what set-up commands are required, you will typically place them in the power-up sequence, sequence #100. However, this may not always be the case. In some applications, the 500 may be required to be in different modes for different parts of the application. For example, some moves may be Continuous mode moves and some may be Preset mode moves. In this case, you can place the appropriate set-up commands in the sequence in which that particular motion is required. After entering the power-up sequence, you can cycle power, issue a Reset (**Z**) command, or simply run sequence #100 with an **XR1ØØ** command to configure the 500 according to the set-up commands in the power-up sequence.

The manner in which the remaining sequences are programmed will be determined largely by the type of interface you choose. There are three typical types of applications that require different methods of executing motion through sequences.

Method ① The interface may be discrete inputs which select or control sequences. In this case a program of several sequences will have the program flow controlled by the inputs or there may be a small number of sequences selected by discrete inputs as sequence select lines.

Method ②

The interface is thumbwheels or a PLC and sequences are selected via the inputs. In this case the different motion requirements are usually in each sequence.

Method ③

The third typical application has data or move parameter information entered via the inputs. In this case, there are usually fewer sequences but they require some information from the data entered via the 500 inputs.

These types of applications are not all-inclusive, but they do offer ideas in how you might set up your interface to execute the required motion.

Application Example #1

In this application, parts are formed by a grinder. The part that is being ground moves at a high speed toward the grinder. Just before it hits the grinder, it slows to a lower speed and the grinding operation begins. Because there are different parts of different lengths, different distances are required for the grinding part of the operation. The distance from the point where the part is loaded to the point where the grinding of the part is finished is a fixed distance. What varies is the point at which the grinding begins. Only five different parts are made. The distance from the point of loading the part to the end of the grind is 24 inches. The motor is on a leadscrew with a pitch of 2. The operator interface must be very simple and include part-selection and run/stop switches.

Types of Motion Required

In this application, a preset move must be made. The first part of the distance must be moved at a high speed and the second part at a low speed. The point at which the move changes from high speed to low speed varies with each part type. A homing move positions the machine at the location where parts are loaded. From this location, the other grind moves can be made. Figure 4-21 is a motion profile of the grinding application.

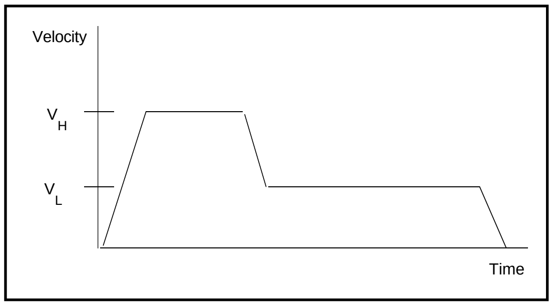


Figure 4-21. Grinding Application Motion Profile

What Will Cause Motion and When?

It is necessary to have the machine move to its part loading location when it starts so that it is ready to load parts and make grind moves. A home switch will be placed at the loading point to allow the 500 to position itself at the home location. A toggle switch allows the user to select either the Run mode or Stop mode. When the switch is in Run mode, it will continuously make parts of the type selected until the run/stop switch is placed in the Stop mode. The parts are selected by turning on the switch labeled for the particular part. The interface will be 6 switches. One switch will toggle between the run or stop mode. The user can select the part with the other 5 switches. The part-selection switches are on/off switches as opposed to momentary contact switches. Figure 4-22 shows the switch configuration.

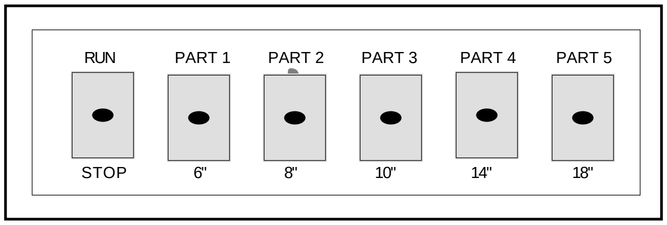


Figure 4-22. Front Panel of User Interface

What Configuration Commands Are Required?

❑	`SSJ1`	Sequence Scan mode
❑	`INnA`	Trigger input, n is the input number
❑	`INnB`	Sequence select input
❑	`OSB`	Back up to home switch
❑	`OSG`	Select final home approach direction
❑	`OSH`	Select edge of home switch which 500 stops on
❑	`GHV`	Select homing velocity
❑	`GHF`	Final homing approach velocity

Implementation

The part selection switches select a sequence that contains the necessary move for that particular part. The motion will not begin until the run/stop switch is in the run position. Parts will continue to be made until the run/stop switch is set to the stop position. The user can select Stop mode at any time during the grind move. The grind move that was executed at the time the Stop mode was selected will be completed. While in Stop mode, a new part type can be selected. The machine will operate this way as long as it is powered up. Upon power up, the machine will move to the home location to begin operation.

Required Sequences

The resolution set by the Configure Motor Resolution (**CMR**) command is 5000 steps/rev. With a pitch of 2, the total move distance is 240000 steps. The part sizes are 6, 8, 10, 14, and 18 inches. This makes grind distances of 60000, 80000, 100000, 140000, and 180000 steps.

SEQUENCE #100

Command	Description
`> XD1ØØ`	Begins definition of sequence #100
`MPI`	Incremental move mode
`MN`	Normal preset mode
`CMR5ØØØ`	Sets resolution to 5000 steps/rev
`GHV2`	Initial go home velocity if 2 rps
`GHF.3`	Final go home velocity of 0.3 rps
`OSB1`	Back up to home switch
`OSH1`	Stop on the CCW edge of the switch
`OSG0`	Final approach direction is CW
`GH`	Begin go home move
`IN1B`	Sets up input #1 as sequence select input
`IN2B`	Sets up input #2 as sequence select input
`IN3B`	Sets up input #3 as sequence select input
`IN4B`	Sets up input #4 as sequence select input
`IN5B`	Sets up input #5 as sequence select input
`IN6A`	Sets up input #6 as a trigger
`SSJ1`	Place the 500 in continuous scan mode
`A1ØØ`	Set the acceleration to 100 rps^2
`TR1`	Wait until run switch is enabled
`> XT`	Ends definition of sequence #100

SEQUENCE #1

Command	Description
`> XD1`	Begins definition of sequence #1
`V4`	Defines the fast velocity
`D24ØØØØ`	Total move distance
`REPEAT`	Repeat loop to make parts continuously
`MPP`	Enters the profiling mode so velocity can be changed on the fly
`G`	Initiates motion
`DP18ØØØØ`	After 180,000 steps decelerate to the grind velocity
`V.5`	Grind velocity
`NG`	Ends profiling mode (no commands executed until move is finished)
`V4`	Return for a new part at a high speed
`H`	Change direction
`G`	Return to the get a new part
`UNTIL(XXXXXXXXØ)`	Continue the repeat loop until the run/stop switch has been placed in the stop location
`TR1`	Wait until the run/stop switch is placed in the run location then finish this sequence and scan the inputs to select another part sequence
`> XT`	Ends definition of sequence #1

SEQUENCE #2

```
> XD2
  V4
  D24ØØØØ
  REPEAT
  MPP
  G
  DP16ØØØØ
  V.5
  NG
  V4
  H
  G
  UNTIL(XXXXXXXXØ)
  TR1
> XT
```

SEQUENCE #4

```
> XD4
  V4
  D24ØØØØ
  REPEAT
  MPP
  G
  DP14ØØØØ
  V.5
  NG
  V4
  H
  G
  UNTIL(XXXXXXXXØ)
  TR1
> XT
```

SEQUENCE #8

```
> XD8
  V4
  D24ØØØØ
  REPEAT
  MPP
  G
  DP1ØØØØØ
  V.5
  NG
  V4
  H
  G
  UNTIL(XXXXXXXXØ)
  TR1
> XT
```

SEQUENCE #16

```
> XD16
  V4
  D24ØØØØ
  REPEAT
  MPP
  G
  DP8ØØØØ
  V.5
  NG
  V4
  H
  G
  UNTIL(XXXXXXXXØ)
  TR1
> XT
```

Since the sequence select inputs are binary weighted, only one switch is needed to select each sequence above. That is why sequences 1, 2, 4, 8, and 16 were chosen. This method is useful only if there are a small number of sequences (7 or less).

Application Example #2

If we modify example #1 slightly, the same motion can be accomplished using a different user interface. In this example #2, we will have the same motion requirements, but 30 different parts will be made. The same run/stop criteria from example #1 are in effect; however, thumbwheels will be used to enter the grind distance data. The data will be entered as the length of the part. The distance of the grind move is the length of the part. After the distance is entered on the thumbwheels, the grind move will run until the stop switch is enabled. A homing move on power up is still required to place the machine in the parts loading position.

Required Motion

The required motion will be the same as example #1.

What Will Cause Motion and When?

Like example #1, a run/stop switch will be used to start and finish the grind moves. A homing move will be required to start the process. This will be done the same way as example 1. The point at which the velocity must slow to a lower velocity will be determined by thumbwheels and some math calculations. The user need only enter the length of the part in inches on the thumbwheels and then select the run mode. The 500 will then perform the required grind move.

What Configuration Commands Are Required?

❑ **VARDn**	Reads the input data into variable n
❑ **INnA**	Trigger input, n is the input number
❑ **INnN**	Data input
❑ **OUTnJ**	Sets the outputs as strobe outputs
❑ **STR**	Strobe time for reading the data inputs
❑ **OSB**	Back up to home switch
❑ **OSG**	Select final home approach direction
❑ **OSH**	Select edge of home switch which 500 stops on
❑ **GHV**	Select homing velocity
❑ **GHF**	Final homing approach velocity

Implementation

The process will run identically to the process above with the exception of selecting the grind move. In this case the grind move is determined by reading data in via the thumbwheels then beginning the move when the run mode is enabled. When the stop mode is enabled new grind move data can be entered on the thumbwheels. One bank of 8 thumbwheel switches is used so the enable inputs on the TM8 module are set to the enable state rather than to a 500 output.

Required Sequences

The resolution set by the Configure Motor Resolution (**CMR**) command is 5000 steps/rev.

SEQUENCE #100

Command	Description
> **XD1ØØ**	Begins definition of sequence #100
MPI	Incremental move mode
MN	Normal preset mode
CMR5ØØØ	Sets resolution to 5000 steps/rev
GHV2	Initial go home velocity if 2 rps
GHF.3	Final go home velocity of 0.3 rps
OSB1	Back up to home switch
OSH1	Stop on the CCW edge of the switch
OSGØ	Final approach direction is CW
GH	Begin go home move
IN1N	Sets up input #1 as data input
IN2N	Sets up input #2 as data input
IN3N	Sets up input #3 as data input
IN4N	Sets up input #4 as data input
IN5A	Sets up input 5 as a trigger input
OUT1J	Sets up Output #1 as strobe output
OUT2J	Sets up Output #2 as strobe output
OUT3J	Sets up Output #3 as strobe output
STR1Ø	Strobe time is 10 milliseconds
A1ØØ	Sets acceleration to 100 rps^2
XR1	Runs sequence #1
> **XT**	Ends definition of sequence #100

SEQUENCE #1

Command	Description
`> XD1`	Begins definition of sequence #1
`L`	Starts an infinite loop
`V4`	Defines the fast velocity
`D240000`	Total move distance
`VARD1`	Reads the thumbwheels for variable 1
`VAR1=VAR1*10000`	Convert part length inches to steps
`VAR1=240000-VAR1`	Determine distance point DP where velocity changes to a lower speed
`REPEAT`	REPEAT loop to repeatedly make parts
`MPP`	Enters the profiling mode so velocity can be changed on the fly
`G`	Initiates motion
`DP(VAR1)`	Loads the distance point determined in variable 1
`V.5`	Grind velocity
`NG`	Ends the profiling mode. No commands will execute until the move is finished
`V4`	Return for a new part at a high speed
`H`	Change direction
`G`	Return to the get a new part
`UNTIL(INXXXXXXXX0)`	Continue the repeat loop until the run/stop switch has been placed in the stop location
`TR1`	Wait until the run/stop switch is set to run then finish this sequence and scan the inputs to select another part sequence
`N`	Ends the infinite loop
`> XT`	Ends definition of sequence #1

This is the only sequence required. It is an infinite loop where parts are continuously run until the Stop mode is enabled. When this occurs, the program is waiting at the trigger command. It will remain there until the Run mode is selected. The thumbwheels will be read again so a new grind move can be run. This is more flexible and allows a many different grind moves to be executed.

Application Example #3

This application is a process of which the 500 is controlling one axis. A PLC controls the process. In this process the 500 is feeding material at two different speeds. It will either be feeding the material at a high speed or a low speed. The PLC will signal the 500 when to stop the material. The material must be able to switch between the high speed and low speed without stopping. When the PLC stops the material feed, the material is cut and the 500 must return to the point it started to begin the process again. The point at which the material feed begins is the home position for starting motion. On power up, the 500 should move to this position. The PLC controls motion. Three different types of material will require different high and low velocities.

Required Motion

This application will require continuous moves. It will also require a preset move to return to the starting location. On power up a homing move will be made. The PLC controls continuous moves. The moves will always start with the high velocity and will then be toggled between high and low by the PLC. The PLC will stop the operation when it is complete. Different move velocities will be required for the different material. The distance of the preset return move will be determined by the 500 based on the distance it traveled during the continuous move.

What Will Cause Motion and When?

The PLC will control the entire process. The 500 will home itself after the initial power up. The continuous move will then begin by the PLC selecting one of three materials. Each material will have its own high and low velocities. The continuous move will continue until the PLC issues a stop. Then the 500 will make a preset move back to the start and the PLC will select one of the sequences again.

What Configuration Commands Are Required?

❑	SSJ1	Sequence Scan mode
❑	XQ1	Interrupted Sequence Scan mode
❑	INnA	Trigger input, n is the input number
❑	INnB	Sequence select input
❑	OUTnC	Sequence in progress output
❑	OSB	Back up to home switch
❑	OSG	Select final home approach direction
❑	OSH	Select edge of home switch which 500 stops on
❑	GHV	Select homing velocity
❑	GHF	Final homing approach velocity
❑	MC	Sets the 500 in the Continuous mode
❑	MN	Sets the 500 in the Preset mode

Implementation The 500 will be in Sequence Scan mode. The PLC will select 1 of the 3 sequences. It will then control the high speed or low speed with one line. One input will also be configured as a stop input. It will set the velocity level input to a 1 for high speed and a Ø for low speed. When a stop is issued, the PLC will wait for the 500 to signal that it is back to the start position and then select the next sequence with the sequence select inputs. An output is set up to turn on when a sequence is in progress. When it is off, the PLC can select a new sequence. The new sequence will not be run until the 500 has made the preset move back to the start location. Returning to the start location ends the sequence and allows the next sequence to be scanned.

SEQUENCE #100

```
Command              Description
> XD1ØØ              Begins definition of sequence #100
  MPI                Incremental move mode
  MC                 Sets 500 to Continuous mode
  CMR5ØØØ            Sets resolution to 5000 steps/rev
  GHV2               Initial go home velocity if 2 rps
  GHF.3              Final go home velocity of 0.3 rps
  OSB1               Back up to home switch
  OSH1               Stop on the CCW edge of the switch
  OSGØ               Final approach direction is CW
  GH                 Begin go home move
  IN1B               Sets up inputs 1 and 2 as sequence select inputs
  IN2B
  IN3A               High/Low velocity input (trigger input)
  IN4A               Stop input
  OUT1C              Sets the output of a sequence-in-progress output
  XQ1                Enables the interrupted run mode
  SSJ1               Place the 500 in Continuous Scan mode
  A1ØØ               Set the acceleration to 100 rps^2
> XT                 Ends definition of sequence #100
```

SEQUENCE #1

```
Command                   Description
> XD1                     Begins definition of sequence #1
  PZ                      Zeroes (clears) the position counter
  MC                      Sets the 500 in continuous mode
  V18.5                   Sets the high velocity to 18.5 rps
  H+                      Sets the direction to CW
  MPP                     Enables MPP mode
  G                       Begins motion
  REPEAT                  Loops until stopped
   IF(INXXXXXX1)          Checks for high or low velocity
  V3                      Sets the velocity to the high velocity
  ELSE                    If not low velocity then
  V18.5                   Sets the velocity high
   UNTIL(INXXXXXX1)       Checks for the stop signal
  NG                      Ends the Profiling mode
  STOP                    Stops the move
  VAR1=POS                Sets variable one equal to the position counter
  D(VAR1)                 Loads the distance with the distance traveled in the Preset move
  H                       Sets the direction CCW
  MN                      Sets the 500 in the preset mode
  G                       Returns to the starting point
> XT                      Ends definition of sequence #1
```

SEQUENCE #2

```
> XD2
  PZ
  MC
  V12
  H+
  MPP
  G
   REPEAT  IF(INXXXXXX1)
  V1
   ELSE
  V12
   UNTIL(INXXXXXX1)
  NG
   STOP
   VAR1=POS
   D(VAR1)
  H
  MN
  G
> XT
```

SEQUENCE #3

```
> XD3
  PZ
  MC
  V9
  H+
  MPP
  G
   REPEAT
   IF(INXXXXXX1)
  V.5
   ELSE
  V9
   UNTIL(INXXXXXX1)
  NG
   STOP
   VAR1=POS
   D(VAR1)
  H
  MN
  G
> XT
```

Chapter 5. Model 500-FOL Follower

Chapter Objectives

The information in this chapter will enable you to:

- ❑ Understand basic following concepts
- ❑ Understand the basic types of following and their common applications
- ❑ Become familiar with the 500-FOL commands associated with following

What is Following?

The 500-FOL can perform velocity following and distance following moves. The 500-FOL can follow from an incremental or absolute encoder input. *Unless otherwise noted, **all the features that will be presented or have been discussed are also valid in Following mode. The only difference is that you replace the velocity command with a speed ratio and the acceleration with a following acceleration for distance following.** Instead of specifying the speed with the* **V** *command, you will specify the speed with respect to the primary using the* **FOL** *command.* Figure 5-1 shows the basic following system configuration.

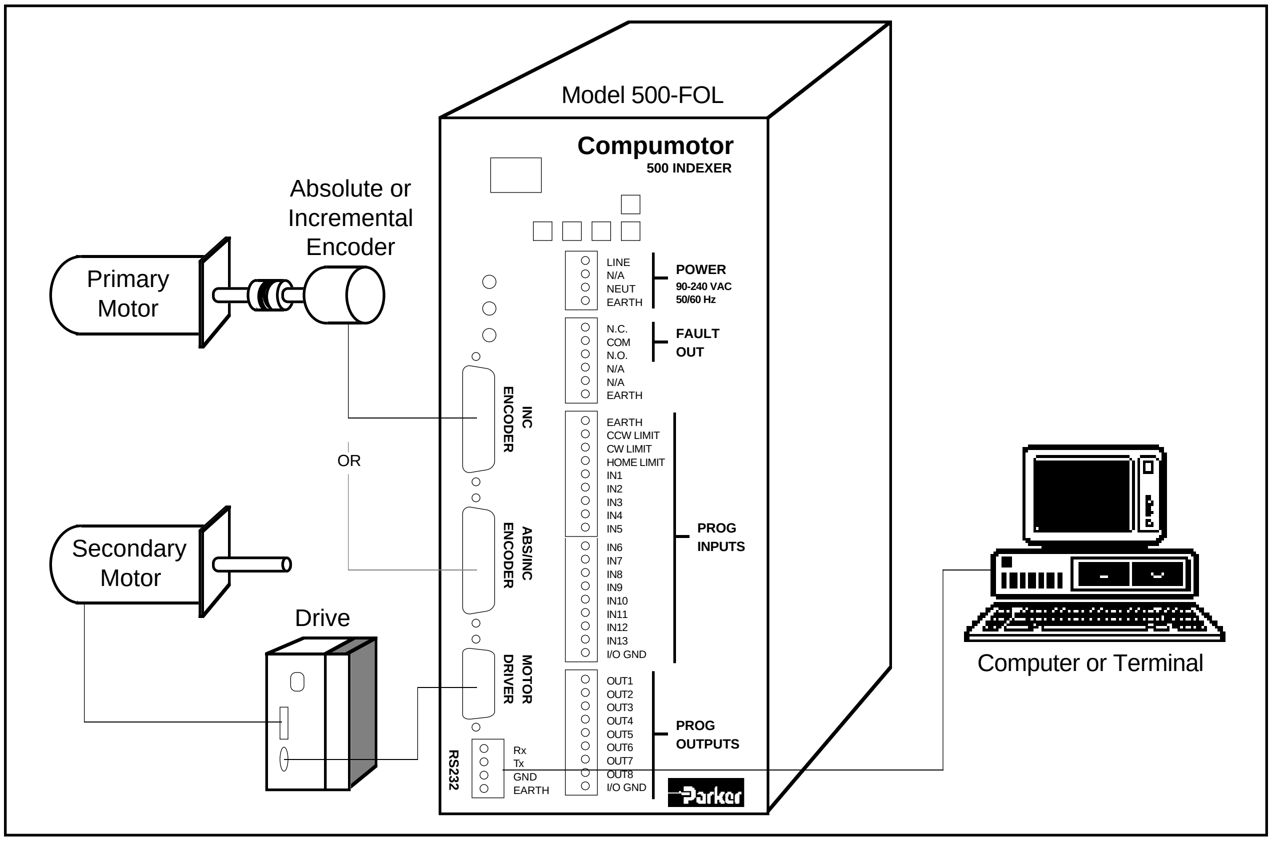


Figure 5-1. Typical Open-Loop Following System Configuration

In Figure 5-1, the primary axis is an AC or DC motor. The encoder is mounted on the shaft of the primary motor. The encoder provides the 500-FOL with data on the primary motor's position and velocity. The 500-FOL uses the position and velocity data to move the secondary axis. Therefore, the secondary motor is *following* the motion of the primary motor. The 500-FOL controls the motion of the secondary motor according to the motion of the primary. This concept of following can be used in different forms to satisfy many different applications.

Types of Following

There are four types of Following application motion.

- ❏ Velocity following
- ❏ Velocity and position following
- ❏ Recede and advance while following (includes electronic cam)
- ❏ Phase error correction (synchronization)

The order in which the categories are presented are roughly in order of increasing complexity. By identifying how the secondary axis motion relates to the primary axis, you will be able to determine your application's type and the applicable programming commands. The 500-FOL also provides several addition features that have been added to address more specific attributes of the various applications.

The 500-FOL has several additional features that enhance the functionality for an application.

- ❏ Registration while following
- ❏ Jogging in the following mode
- ❏ Following a pulse train and a direction
- ❏ Following encoder pulses and the encoder direction
- ❏ Entering following ratio via thumbwheels
- ❏ Entering following ratio via front panel pushbuttons
- ❏ Capability of loading the primary encoder position into a variable

The 500-FOL can perform all of the 500's programming functions. The only difference is that the motion profiles are now following moves and motion is based on a primary axis. The 500-FOL can also be used in its Indexer mode. You can easily switch back and forth from follower and indexer functionality and create motion programs in which the 500-FOL serves as both a follower and an indexer. The 500-FOL has the following programming capabilities:

- ❏ Variables, general-purpose and read-only (position, primary encoder position, etc.)
- ❏ Math
- ❏ Complex branching—**`IF ELSE`**, **`REPEAT UNTIL`**, **`WHILE`**
- ❏ On-the-fly changes—**`MPP`** mode
- ❏ Flexible I/O
- ❏ Closed-Loop mode while following

For an explanation of the features that are common to both the 500 and the 500-FOL, refer to Chapter 4, *Application Design*. You should understand how to set the 500-FOL for following. You can use one command (**`FSI`**) to enter and exit the Indexer and Following modes.

Command	Description
`> FSI1`	Enters the following mode
`> FSIØ`	Exits the following mode(indexer mode)

The rest of this chapter explains the four following types. Additional features of 500-FOL following will also be covered.

Velocity Following

In ***velocity following***, the secondary axis is concerned only with the primary axis' speed. The relationship of the primary axis position with respect to the secondary axis is irrelevant. In this type of application, the secondary axis accelerates at a specified acceleration up to a ratio of the primary axis' speed. Preset or continuous moves are performed in the same manner as in Indexer mode. Exact distances on the secondary axis can be moved in Preset mode. However, instead of moving at a velocity specified by the **V** command, the secondary axis moves at a ratio of the primary axis' velocity specified by the **FOR** and **FOL** commands. The acceleration is independent of the primary axis encoder and is specified by **A** and **AD** (as with an indexer).

Figure 5-2 shows velocity following. Once the secondary axis accelerates to the specified following ratio, it tracks the primary axis' speed and position at the specified ratio. During the acceleration, the primary encoder speed and position are not followed. The acceleration ramp is independent of the primary axis. Once the secondary axis has accelerated to the specified following ratio, it will be following the primary axis (if the primary axis slows down, the secondary axis slows down).

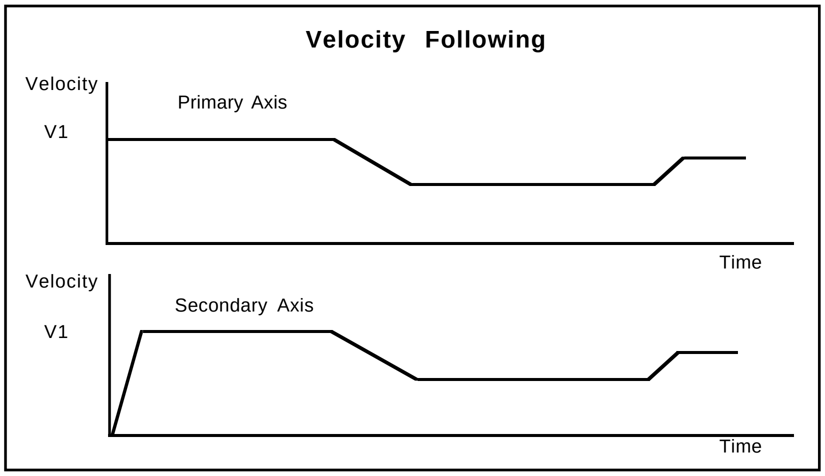


Figure 5-2. Velocity Following

Velocity following is used in dispensing and on-the-fly cutting applications.

In each of the following categories described, the secondary axis only moves in the direction specified by either the **D** command, the **V** command, or the **H** command. If the primary axis changes direction, the secondary axis will still move in the same direction. Only the number of pulses and the rate of the pulses determines the secondary axis' motion. With position and direction tracking enabled, the secondary axis will follow the primary axis' direction. ***To illustrate velocity following, detach the encoder from the primary axis and manually move the encoder***.

In Figure 5-2, if the primary velocity changed after the following ratio was achieved, position and velocity were tracked exactly. If the primary axis' velocity were half as fast as the example in Figure 5-3, a different phase or positional relationship would result during acceleration from rest. This is why it is velocity following. ***Again, once the secondary axis accelerates to the specified following percentage, it will follow velocity and position exactly***.

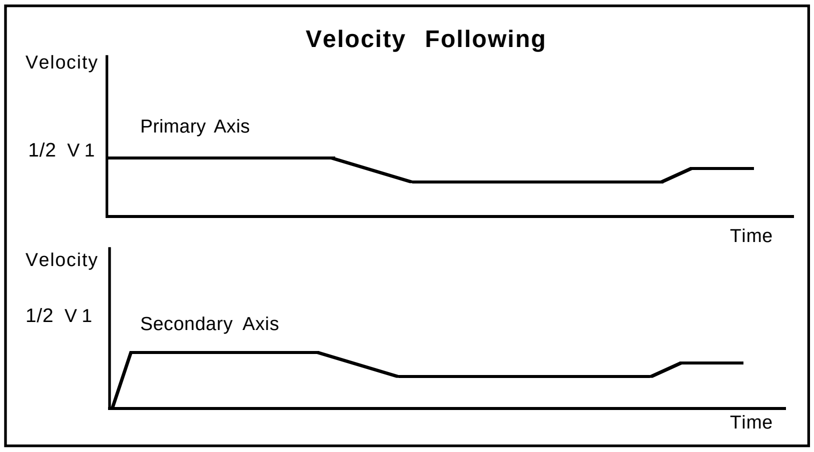


Figure 5-3. Primary Axis as Half Speed with Velocity Following

Setting Up Velocity Following

To perform velocity following, you must use two commands.

Command	Description
FOR	Relates the number of secondary motor steps per unit of travel to the corresponding primary encoder steps per unit of travel.
FOL	Relates the speed of the secondary axis to the speed of the primary axis as long as **FOR** is set correctly. The value is entered as a percentage of the primary axis speed.

The 500-FOL uses the equation to determine the number of motor steps.

$$\text{Motor Steps} = \textbf{FOR} * \frac{\textbf{FOL}}{100} * \text{Encoder Steps}$$

These two commands, (**FOR** and **FOL**) remove the complication of having a different resolution for measurement of distance on the primary and secondary axes. Once you relate the number of primary encoder pulses per unit of travel to the number of secondary motor pulses per that same unit of travel, you can relate their speeds. You can specify the secondary axis to move at 50% of the speed of the primary axis (1:2 ratio) or 100% of the speed (1:1 ratio). The encoder is usually mounted on the primary axis or through gearing and will measure a certain number of pulses per inch. The secondary axis would also have some certain number of pulses per inch. The ratio of the secondary axis steps to the primary axis' steps is entered in the **FOR** command. If the primary axis moves at 10 ips (inches per second) and the secondary axis is also set to move at 10 ips, a speed percentage of 100 is all that needs to be programmed (**FOL1ØØ**).

The remaining commands are standard 500 commands. Use the following commands to implement the velocity following feature:

Step ① Enter the number of secondary motor steps per unit of travel per number of primary encoder steps per the same unit of travel. In this example, the motor step resolution is 25,000 steps/rev and the encoder resolution is 4,000 steps/rev. The unit of travel is 1 revolution—the ratio is 25,000/4,000 or 6.25.

Command	Description
> **FOR6.25**	Sets the motor step to encoder step ratio of 6.25
> **OFF**	Turns the 500-FOL off
> **CMR25ØØØ**	Sets motor resolution to 25000 steps/rev
> **ON**	Turns the 500-FOL on

Step ② Enter the following mode with the **FSI** command.

Command	Description
`> FSI1`	Enters the following mode

Step ③ Set the speed ratio. If the secondary axis is to move at the same speed as the primary axis (100% of the primary axis' speed), enter **FOL1ØØ**.

Command	Description
`> FOL1ØØ`	The secondary axis will move at 100% of the primary axis' speed

Step ④ Set the 500-FOL to Continuous mode and begin motion with the **G** command.

Command	Description
`> MC`	Enters Continuous mode
`> A5ØØ`	Sets acceleration
`> AD5ØØ`	Sets deceleration
`> G`	Initiates motion

Turn the encoder. The 500-FOL should begin moving at the same speed that you are turning the encoder. Change the direction that you turn the encoder and note the motion of the secondary axis. Follow the steps below.

Step ⑤ Execute the following commands.

Command	Description
`> S`	Stops the continuous motion command issued above
`> 1FSP1`	Enables the Position and Direction Tacking mode
`> G`	Initiates motion

Step ⑥ Now turn the encoder or move the primary so that the encoder moves. The 500-FOL will begin moving. Change the direction that the encoder is moving. ***The 500-FOL will change direction***. To change the relative direction between the 500-FOL and the encoder, use the change direction command (**H±**)**.** Now turn the encoder and note that the 500-FOL's relative direction has changed. Tracking the direction as well as pulses can only be used in Continuous mode. Preset moves can also be performed in velocity following.

Preset Moves In Velocity Following

A preset move is preformed as in the standard Indexer mode. Issue the mode normal command (**MN**) and specify a distance in terms of secondary motor steps. In a preset move:

① The secondary axis accelerates (**A**) to the desired speed percentage.
② It decelerates at the rate set with the **AD** command.
③ It moves the specified distance (**D**) and stop.

If the primary axis' speed varies, the acceleration ramp will be the same, but the distance that the secondary axis travels to reach it following ratio will be different. The positional relationship or phase relationship is not maintained during acceleration. Once the secondary axis achieves the following percentage speed, it will track both velocity and position exactly.

Step ① Attach the encoder to the primary axis. Start the primary axis moving

Step ② Enter the following set of commands.

Command	Description
`> MN`	Enters Normal mode
`> FSPØ`	Exits the direction tracking mode
`> A5ØØ`	Sets acceleration
`> AD5ØØ`	Sets deceleration
`> FOL1ØØ`	Sets the following speed percentage to 100
`> D125ØØØ`	Sets the distance to 125000 steps
`> G`	Initiates motion

The secondary axis will move 125000 motor steps at the same speed as the primary axis. Repeat the example, but vary the speed of the primary axis. ***The secondary still moves 125000 steps, and its speed varies with the primary axis***.

Velocity following is the simplest form of following. All motion in the 500-FOL is programmed in the same manner as the 500. The only exception is that the velocity command is replaced by a following percentage.

Position and Velocity Following

Position and velocity following is the most common form of following. In this case, the secondary axis must maintain a specific positional relationship with the primary axis. The primary axis moves a specific number of primary encoder pulses while the secondary axis must move a specified number of secondary motor steps. The secondary axis moves at the same speed as the primary axis (a 1:1 speed ratio) in most of these applications. ***Coil winding applications, however, are an exception***. In coil winding applications, the relationship between the primary (usually the spindle axis) and secondary axes (usually the traverse axis) is based on a desired pitch. This pitch defines a positional and velocity relationship that will have an arbitrary speed relationship based on the particular application. The positional relationship is usually defined by a single traverse corresponding to a specific number of spindle axis turns, thus defining a specific number of primary encoder pulses during which a specific number of secondary pulses must be traveled.

Primary Axis at Rest

There are two variations of ***position and velocity following***. The difference is in the motion of the primary axis. The less common form starts the primary axis motion from rest. The secondary axis must follow the primary axis pulse for pulse. In this type of application, the secondary axis will be in a continuous move and will track the motion of the primary axis exactly. A *web positioning system* where two axes are guiding the web (one edge is the primary axis and the other edge is the secondary axis), is an example of such an application. The web can be positioned based on the motion relationship between the primary and secondary axes. Figure 5-4 shows the profiles of a secondary axis following a primary axis pulse for pulse.

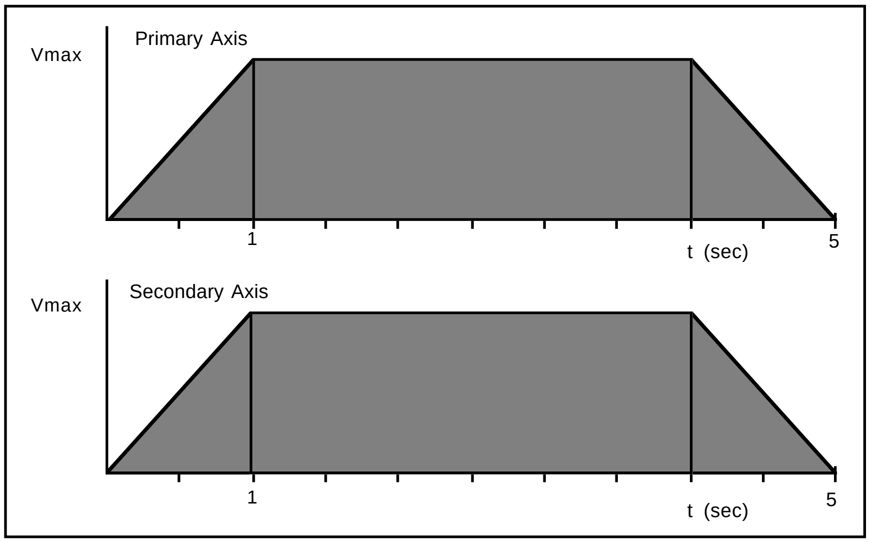


Figure 5-4. Pulse-for-Pulse Following

If the primary axis starts from rest, the secondary axis must track it pulse for pulse. **FSA** enables this capability. If position tracking is disabled, the 500-FOL will follow the pulse count only and will not change direction if the primary axis changes direction. This is acceptable in many applications. In some applications, however, the primary axis may overshoot when it comes to a rest and the encoder will change directions.

Since the 500-FOL is following the primary axis' pulse count only, it will actually move the secondary axis too many pulses. By enabling Position Tracking (**FSP1**), you can you can track the primary encoder's direction and pulse count and therefore not accumulate excess pulses caused by overshoot.

Command	Description
FSA1	Enables Pulse Tracking mode where secondary instantaneously accelerates between commanded velocities
FSP1	Secondary tracks both the direction & pulse count from the primary encoder

You must disable the Following Synchronized Acceleration mode (**FSPØ**) and activate the Continuous mode to use the Pulse Tracking mode (**FSA1**). **FSAØ** disables Pulse Tracking mode.

Primary Axis in Motion

The most common form of ***position and velocity following*** begins with the primary axis already in motion. The secondary axis must accelerate up to the specified speed ratio of the primary axis. In this type of application, the secondary axis must accelerate to a known positional or phase relationship with the primary axis. The primary axis is usually a conveyer or web and the secondary axis is performing an operation on the web or parts on the conveyer. The primary axis is always moving so the secondary axis must be moving at the same speed and with the correct orientation to perform some operation on the moving primary axis. To maintain positional and velocity relationships, the secondary axis must be able to accelerate over a known distance with respect to the primary axis (i.e., following acceleration is needed). The Set Following Synchronization Rate (**FAC**) and Set Following Synchronization Count (**FEN**) commands are used in conjunction with the velocity following commands.

The **A** and **AD** commands used in velocity following are replaced by a following acceleration. A following acceleration is accomplished by stepping through subsequent ratios from the present following ratio to the final following ratio (**FOL**). The increment between ratios is set by the **FAC** command. Incrementing from one ratio to the next is based on the primary encoder changing by a set number of primary encoder pulses. This number of pulses is set by the **FEN** command. For example, if the value for **FAC** is 1, the value for **FEN** is 10, and the final following speed percentage is 100 (**FOL**), the secondary axis must accelerate from zero speed to 100% of the primary speed. From rest, every 10 primary encoder steps (**FEN**) the following percentage will change by an increment of 1% (**FAC**) until it is equal to the final following percentage of 100% (**FOL**). Therefore, the secondary axis will accelerate over 1000 primary encoder steps. It does not matter if the speed of the primary axis varies, the secondary axis' acceleration is based on primary encoder steps (not time).

Figures 5-5 and 5-6 illustrate position and velocity following. The shaded area indicates the distance moved by both the primary and secondary axes while the secondary axis is accelerating. ***The secondary axis moves 1/2 the distance that the primary axis moves***. This will always be the case when the secondary axis accelerates from rest to the same speed as the primary axis, when the primary axis is already moving.

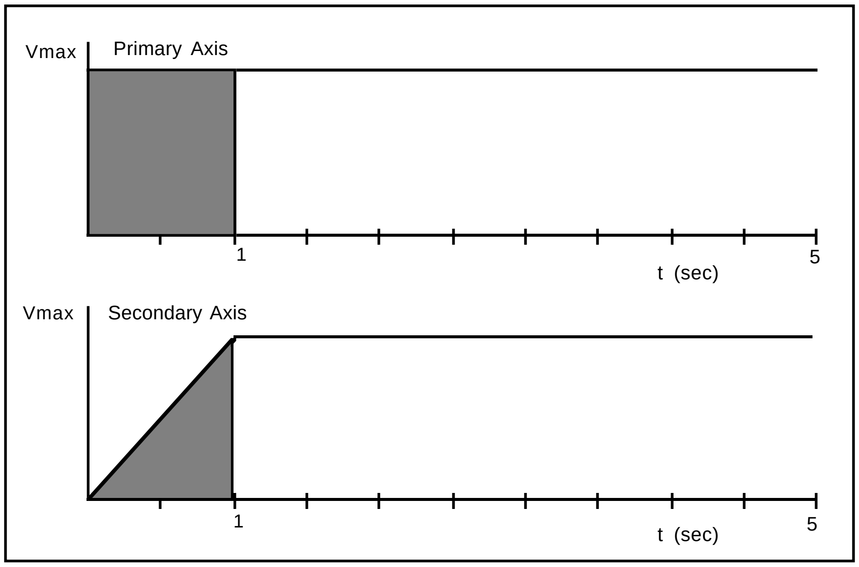


Figure 5-5. Following Acceleration

Figure 5-6 shows how the acceleration ramp varies if the primary axis' speed varies. If the primary axis' speed is 1/2 the maximum velocity, the acceleration ramp will be twice as long. The shaded portions in Figure 5-6 are equivalent to the shaded portions in Figure 5-5. ***The primary axis' velocity can change at any time, even during the secondary axis' acceleration ramp, without changing the positional relationship***.

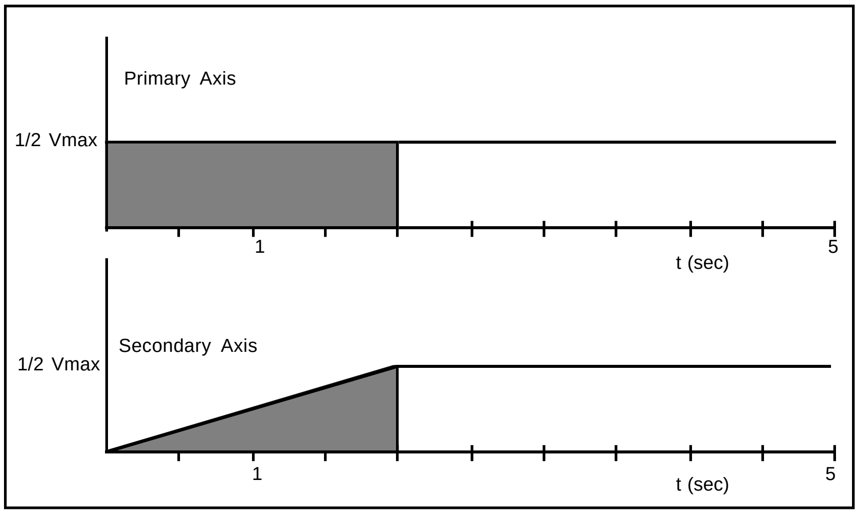


Figure 5-6. Following Acceleration With Primary Axis Velocity Change

The shaded area for the secondary axis is 1/2 of the shaded area under the primary axis curve. When the primary axis is moving and the secondary axis must start from rest and accelerate to the primary axis' velocity, the secondary axis will always move 1/2 the distance. Using following acceleration, the secondary axis adjusts its acceleration according to the primary axis' velocity so that it will always accelerate over the same distance while the primary axis moves a specified distance. To synchronize secondary and primary axes' positions, the secondary axis must start ahead of the primary axis (see Figure 5-7) to compensate for the fact that the primary axis is already moving (velocity = Vp).

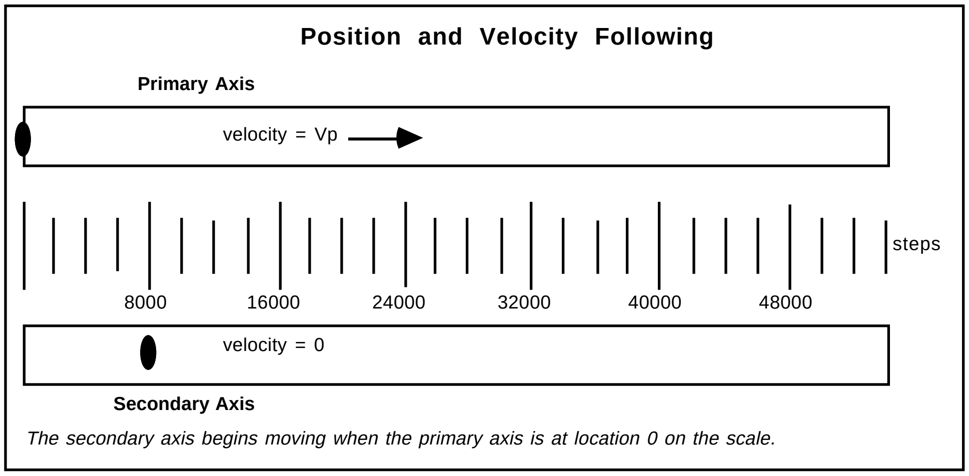

Position and Velocity Following
Primary Axis
velocity = Vp
steps
8000
16000
24000
32000
40000
48000
velocity = 0
Secondary Axis
The secondary axis begins moving when the primary axis is at location 0 on the scale.

Figure 5-7. Starting a Velocity & Position Following Move—1:1 Ratio

The key to position and velocity following is that the **Vp** value does not matter (Figure 5-7). Assuming the application in Figure 5-7 is programmed with following acceleration, the spots, while accelerating over 8000 primary encoder steps, will always be at a 1:1 speed ratio after the primary axis moves the 16000 steps (see Figure 5-8).

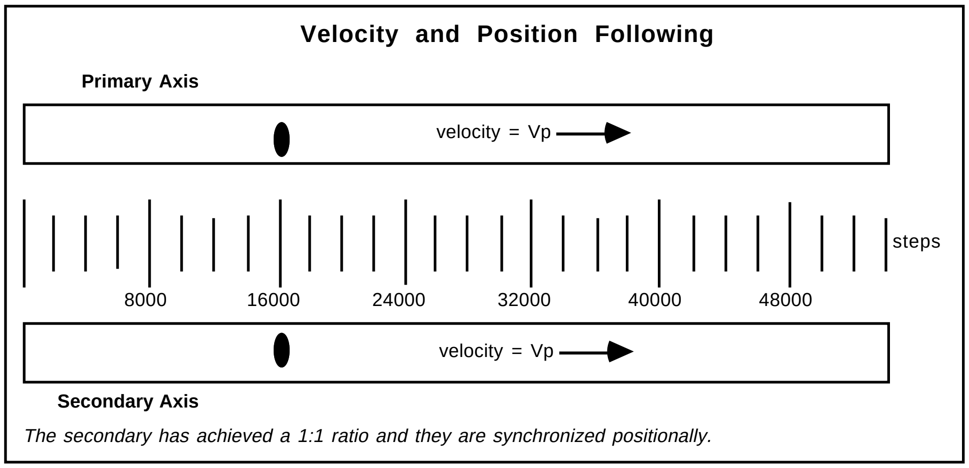

Velocity and Position Following
Primary Axis
velocity = Vp
steps
8000
16000
24000
32000
40000
48000
velocity = Vp
Secondary Axis
The secondary has achieved a 1:1 ratio and they are synchronized positionally.

Figure 5-8. Velocity & Position Following After Acceleration Is Done

Following Acceleration

Accelerating over a known distance with respect to a known primary axis distance allows you to synchronize the exact phase relationship you want between the primary and secondary axes. Following acceleration enables this synchronization.

Command	Description
FAC	The increment of following ratio by which the following ratio changes during acceleration
FEN	The number of encoder pulses that cause the following ratio to increment to the next value
FSF1	Enables Following Acceleration mode

Determining FAC and FEN with Primary Axis Data

To determine what **FAC** *and* **FEN** *are, you must know*:

- ❑ The maximum velocity that the primary axis can travel (the velocity must be in units of primary encoder steps per second)
- ❑ The distance in primary encoder steps that the primary axis will move during which time the secondary axis will accelerate ***or*** the maximum acceleration that the secondary axis can accelerate

These parameters will be denoted as follows:

Vp_{max} = Maximum primary velocity in encoder steps per second
Dp_{acc} = Distance primary axis travels while secondary axis accelerates in primary encoder steps

OR

As_{max} = Maximum acceleration of the secondary axis

Based on these equations, the values for **FAC** and **FEN** can be determined:

Equation 5-1. FEN

$$\mathbf{FEN} = Vp_{max} * \frac{\mathbf{TF}}{1000} \qquad Vp_{max} = \frac{\text{Primary Encoder Counts}}{\text{Second}}$$

TF = Primary Encoder Sample Period in ms

Equation 5-2. FAC

$$\mathbf{FAC} = \frac{\mathbf{FOL} * Vp_{max}}{Dp_{acc}} * \frac{\mathbf{TF}}{1000} \qquad Vp_{max} = \frac{\text{Primary Encoder Counts}}{\text{Second}}$$

TF = Primary Encoder Sample Period in ms
FOL = Following percentage in units of percent
Dp_{acc} = Distance primary moves during secondary axis accel in units of primary axis enc counts

Typically, the application will have the secondary axis start from rest and accelerate up to an **FOL** value of 100 (1:1 ratio). However, the **FOL** value can be any value that is within the limits of the motor/drive system. ***D_{acc} is in units of primary encoder steps***. Determining the **FAC** and **FEN** values sets the number of primary axis encoder steps over which the secondary axis will accelerate (independent of the primary axis' speed). The secondary axis always travels the same number of motor steps during acceleration while the primary encoder moves Dp_{acc}.

The **TF** command allows you to set the sample period of the primary axis' encoder. It is programmable from 1 - 32 ms. ***The default is 4 ms***. **TF** simply scales the **FAC** and **FEN** values. In Equations 5.1 and 5.2, **TF** is used in units of ms—the constant of 1000 converts it to seconds so that the units cancel properly. Typically you will want **TF** to be as fast as your system will allow (1 ms). If the primary encoder is moving slowly, you may need to increase the sample rate to more than 1 ms because the actual encoder count does not change by much in a sample period and thus you have coarser resolution on the changes in encoder counts. For example, if your sample period is 1 ms and the maximum speed is 1 rps, the encoder count only changes by 4 counts each sample period. If there is a slight variation in speed and you read a change of 3 counts, there is a 25% variation. This may cause choppier secondary axis motion. Changing the encoder sample period can have a smoothing effect.

Determining FAC and FEN with Secondary Axis Data

Instead of knowing the distance that you want the secondary axis to accelerate over, you may know the maximum acceleration that your secondary axis can accelerate at. You can determine Dp_{acc} using A_{max} and the following speed percentage that you are accelerating to. Use Equation 5-3 to determine Dp_{acc} from A_{max}.

Equation 5-3.
Dp_{acc}

$$Dp_{acc} = \frac{Vp_{max}^{2}}{As_{max}} * \textbf{FOR} * \frac{\textbf{FOL}}{100}$$

The acceleration is in units of secondary motor steps/sec^2. The maximum velocity of the primary axis is in primary motor steps/sec.

Using this value for Dp_{acc}, you can use Equations 5-1 and 5-2 to determine **FAC** and **FEN**. Remember to enable the Following Synchronized Acceleration mode (**FSF1**) to enable following acceleration.

How Following Acceleration Works

The concept of accelerating the secondary axis over a known distance with respect to a known primary axis distance (independent of the primary axis' speed) is developed on the analogies drawn between *following* and *time-based motion*. In ***time-based motion***, the velocity describes the rate of change of the position with respect to a change in time. In ***following-based motion***, the secondary axis moves at a ratio of the primary axis' velocity. This following ratio is of the same units as the velocity it is following, but is simply scaled. Therefore, the following ratio is analogous to a velocity. In the same manner, an acceleration in the time domain is defined as the rate of velocity change. The analogy in following would be to have a following acceleration that is a rate of change of the following ratio. ***Time-based motion is based on sampled time whereas following is based on the sampled primary axis encoder pulses (for digital systems)***. Examine the following example.

- ❑ Primary axis encoder → 4000 counts/revolution
- ❑ Secondary axis → 4000 steps/revolution.
- ❑ Primary axis speed = 1 rps
- ❑ Secondary axis following speed percentage = 100%
- ❑ Distance that secondary axis accelerates = 2000 primary axis encoder steps
- ❑ Primary axis encoder sample period = 1 ms

Therefore, the secondary axis must now accelerate over 2000 primary encoder steps to a following percentage of 100% or a speed of 1 rps. Specifying the number of primary encoder pulses and the final speed that the secondary axis must attain after acceleration defines the ***acceleration ramp***. If the secondary axis' acceleration is based on time, you can calculate an acceleration ramp that would accelerate the secondary axis in the desired fashion.

$$\text{Acceleration time} = \frac{2000 \text{ steps}}{4000 \, \frac{\text{steps}}{\text{sec}}} = 0.5 \text{ seconds}$$

Change in velocity = 4000 steps/sec — 0 steps/sec = 4000 steps/sec

$$\text{Acceleration} = \frac{4000 \, \frac{\text{steps}}{\text{sec}}}{0.5 \text{ sec}} = 8000 \, \frac{\text{steps}}{\text{sec}^2} = \frac{8 \, \frac{\text{steps}}{\text{sec}}}{\text{ms}}$$

If you changed the velocity by 8 steps/sec every sample period (**TF** - 1 ms), you would achieve the desired acceleration ramp. The problem is that application is dependent on time. If the primary axis' speed changes, the secondary axis would no longer be accelerating over 2000 primary encoder steps. Therefore, the application requires a following acceleration that is based on encoder pulses rather than time.

Change in Following Percentage = 100% - 0% = 100%

$$\text{Following Acceleration time} = \frac{2000 \text{ steps}}{4000 \, \frac{\text{steps}}{\text{sec}}} = 500 \text{ ms} = 500 \text{ sample periods}$$

$$\text{Following Acceleration} = \frac{100}{500} = 0.2 \text{ (FAC)}$$

The application is still time dependent. To remove the time dependency and make the acceleration dependent on the encoder pulses ***replace the time sample period by an encoder period***. In the above example, following percentage was based on changing the following percentage by 0.2 every sample period. It will take 500 sample periods to achieve a 100% following percentage. At 4000 steps/sec, the primary encoder is changing at a rate of 4 steps per ms or 4 steps/sample period.

1 sample period = 4 steps (encoder period—**FEN**)

Instead of changing the following percentage every sample period, change it every time the encoder count changes by 4 steps. If the primary encoder is moving at the maximum velocity, the acceleration ramp will be equal to the maximum acceleration provided. However, if the primary encoder velocity is less than the maximum velocity, the acceleration will also be reduced. The distance that the secondary axis accelerates and travels (with respect to the primary axis' moves during this acceleration) will remain unchanged. If the primary axis exceeds the maximum velocity, the acceleration ramp would also increase and exceed the maximum acceleration, which may cause an overcurrent or related servo error.

In summary, following acceleration uses the analogy that a change in velocity per change in time (normal acceleration) is the same concept as a following acceleration being a change in following percentage per change in encoder steps. In the 500-FOL, the following percentage changes by the **FAC** value for each change in encoder steps of **FEN** steps. Several different combinations of **FAC** and **FEN** can achieve the same acceleration ramp. However, only unique **FAC** and **FEN** values will satisfy a specific maximum velocity and maximum acceleration. **FAC** and **FEN** can easily be determined with the equations and examples discussed earlier.

Decelerating

The 500-FOL decelerates to zero speed using the **AD** value. Although you may expect that this will diminish the positional relationship, this is usually not a concern at the endpoint in a profile. The following move is usually started by a trigger, which indicates that the primary axis is at a particular location. The move could also be started based on the primary axis' encoder position. The move is typically of a preset distance. ***The important point is that the secondary axis is at a known position with respect to the primary axis when the profile begins***. This positional relationship is maintained during acceleration. After the secondary axis moves the appropriate distance, it will normally return the same distance it just traveled at a high speed to prepare for a repeat move. This is why the deceleration is not important. However, in cases where the 500-FOL must perform electronic cam profiles, a deceleration's positional relationship may be necessary. In this case, the 500-FOL can decelerate to a stop using the **FAC** and **FEN** values by setting the following ratio to 0 (**FOLØ**) while in Mode Position Profile mode. To terminate this move, a stop command must be issued after the secondary axis reaches zero speed.

Position and Velocity Following Example

In this example, you will perform a preset move using following acceleration (with the parameters and motion requirements listed below).

- ❑ Primary axis encoder resolution = 4000 counts/rev
- ❑ Secondary axis motor resolution = 25000 steps/rev
- ❑ Maximum primary encoder speed = 2 rps
- ❑ Distance in primary encoder steps that the secondary axis must accelerate over = 2000 steps
- ❑ Desired speed ratio—**FOL** = 100% (1:1 ratio)
- ❑ Preset secondary axis move distance = 3 motor revolutions
- ❑ Encoder sample period—**TF** = 4 ms

A preset move of 75000 steps will be made. The secondary axis will accelerate over 2000 primary encoder steps up to the same speed as the primary encoder. A trigger will initiate motion. Follow these steps to perform the move profile. The encoder sampling period is set to the default of 4 ms.

Step ① Determine the values for **FAC** and **FEN** from Equations 5-1 and 5-2.

$$\textbf{FEN} = Vp_{max} * \frac{\textbf{TF}}{1000} = 2\ \frac{revs}{sec} * 4000\ \frac{counts}{rev} * \frac{4}{1000}\ seconds = 32\ encoder\ counts$$

$$\textbf{FAC} = \frac{\textbf{FOL} * Vp_{max}}{Dp_{acc}} * \frac{\textbf{TF}}{1000} = \frac{100 * 8000\ \frac{counts}{sec}}{2000\ counts} * \frac{4}{1000}\ seconds = 1.6\ percent$$

Step ② Enter the values for **FEN** and **FAC**.

Command	Description
`> FEN32`	Number of encoder counts of change required to increment the following percentage by **FAC**
`> FAC1.6`	The amount the following percentage increments for each **FEN** change in encoder counts

If you want to change the sample period of the primary encoder to 1 ms, scale both **FEN** and **FAC** by the change in the **TF** value. For example, if you go from 4 ms to 1 ms, divide **FAC** and **FEN** by 4 to get the following values.

FEN = 8
FAC = 0.4

If we went from **TF4** to **TF8**, multiply **FAC** and **FEN** by 2. ***Remember to change* FAC *and* FEN *if you change* TF.**

Step ③ Enable the following acceleration mode.

Command	Description
`> FSF1`	Enables following acceleration

Step ④ Start the primary axis into motion, then enter the commands below to perform the following acceleration move. If the primary axis' speed exceeds 2 rps, the following acceleration will not work properly.

Command	Description
`> MN`	Sets 500-FOL to Normal mode
`> FSI1`	Enables following
`> D75000`	Sets the preset move distance to 75000 steps
`> FOL100`	Sets the following percentage to 100%
`> G`	Starts the following move

The secondary axis will accelerate over 2000 primary encoder steps. The secondary axis will move 1000 * **FOR** or 6250 motor steps over this acceleration ramp.

Following Acceleration Example: Bottle Filling

Typically, an application that requires position and velocity following will start the secondary axis from rest and accelerate it to 100% of the primary axis' speed (a 1:1 speed ratio). A trigger initiates motion on the secondary axis when a part or product is in a particular location on the primary axis. In this example (Figure 5-9), a conveyer belt moves bottles on a production line. The secondary axis is a filler that accelerates up to the conveyer line speed and fills the bottles. It fills six bottles at a time and then returns to the start point to fill six more.

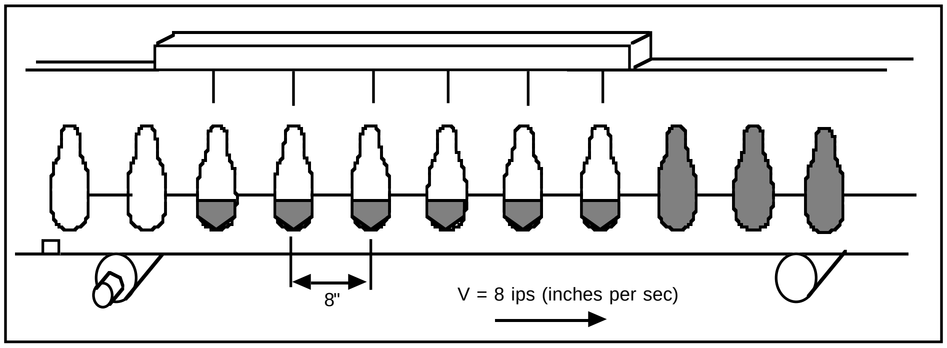


Figure 5-9. Position & Velocity Following—Bottle Filling Application

One cycle of operation consists of the following steps.

① The secondary axis accelerates to the conveyer line speed.

② The secondary axis enables an output that tells the dispenser to begin filling the bottles.

③ The secondary axis decelerates to a stop and returns to the starting point (at a high speed) to begin filling the next set of bottles.

In this application, the rate at which the bottles can be filled determines the maximum rate of the entire dispensing cycle. Note the following information about the application.

Maximum conveyer speed: 8 ips
Time to bill bottles: 2.5 seconds
Primary encoder resolution: 4000 counts/rev
Primary encoder linear resolution: 2 revs/inch = 8000 counts/inch
Distance between bottles: 8 inches
Bottle filler motor resolution: 25000 steps/revolution
Bottle filler linear resolution: 1 revolution/inch = 25000 steps/inch
Distance over which bottle filler accelerates: 2 inches

An output must be activated at the point that the bottle filler is moving at the same speed as the conveyer to initiate the dispensing of fluid into the bottles. A photoelectric sensor detects a bottle and begins the filling cycle. The 500-FOL waits for the sensor as a trigger. When a bottle is detected, the 500-FOL accelerates to the line speed, turns on an output, fills the bottle stops, and returns to the starting point to wait for the next trigger. Figure 5-10 depicts the conveyer and the filler axis at the start of a filling cycle.

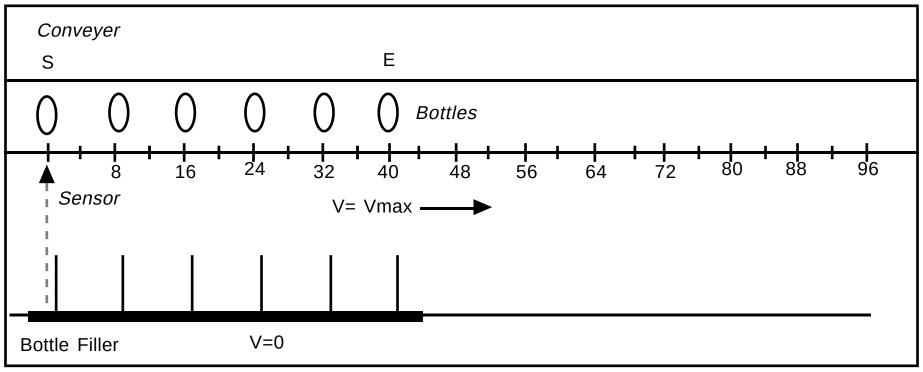


Figure 5-10. Start of A Filling Cycle

When the bottle marked **S** (see Figure 5-10) crosses the trigger sensor, the cycle begins.

① The secondary axis accelerates to the conveyer line speed.
② The secondary axis enables an output that tells the dispenser to begin filling the bottles.
③ The secondary axis decelerates to a stop and returns to the starting point (at a high speed) to begin filling the next set of bottles.

Figure 5-11 shows the two axes after the bottle filler reaches line speed and is ready to begin dispensing.

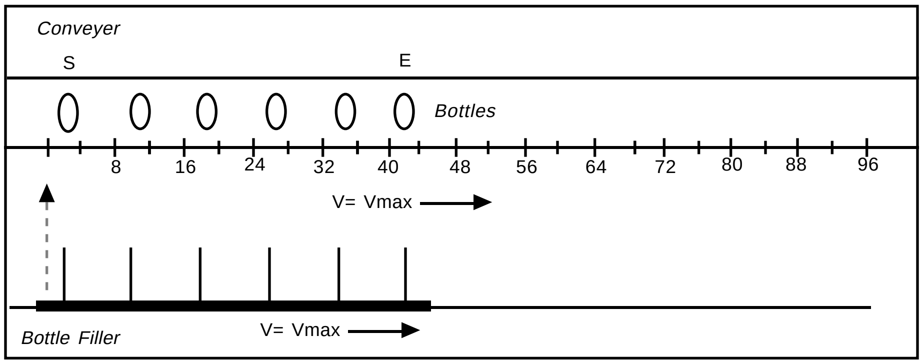


Figure 5-11. Bottle Filler Has Accelerated to Line Speed

A bottle can be filled in 2.5 seconds. At a maximum conveyer speed of 8 ips, the conveyer will move the bottles 20 inches. Figure 5-12 shows the bottle's location after the first six bottles have been filled. ***Note the location of the next set of bottles that will be filled.***

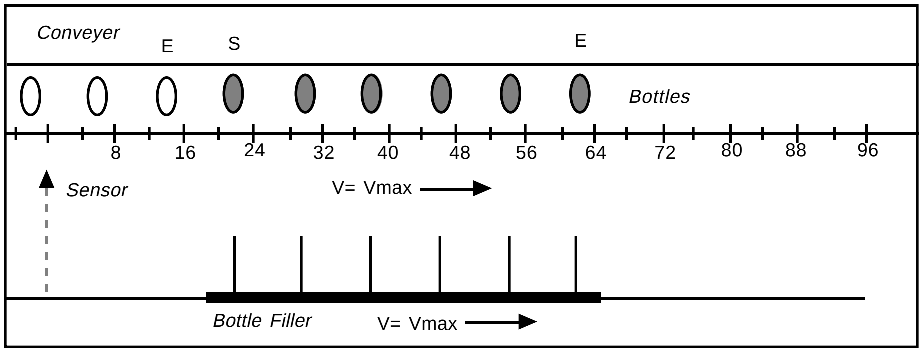


Figure 5-12. Dispensing is Completed.

The bottle filler must now stop and return to the start location before the bottle marked S crosses the trigger point. Figure 5-13 shows the location of the bottle filler after it has stopped. It must now return 22 inches to the start before the bottles have moved 20 inches. The bottle filler will be in place, ready for the next trigger from the next set of bottles. It will arrive in place 4 inches before the next bottle. At 8 ips, 500 ms will elapse before the next bottle.

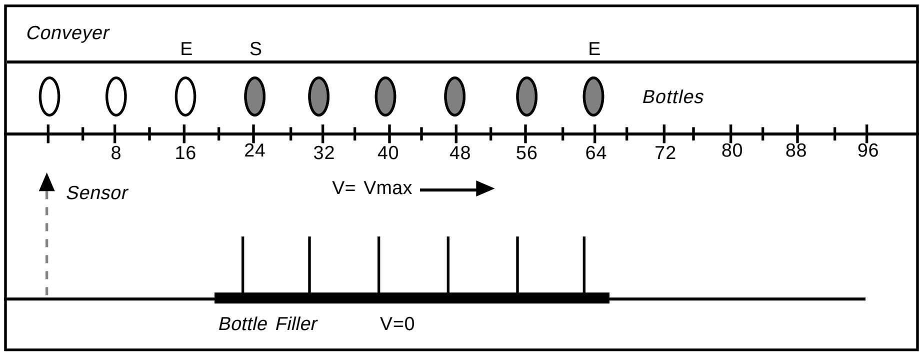


Figure 5-13. Bottle Filler Stops

To get back to the starting point before the conveyer has moved 20 inches, the bottle filler must return at a speed faster than the conveyer. It will accelerate at the same following acceleration set for the first part of the cycle. Figure 5-14 shows the bottle filler after it returns to the starting point. After the conveyer travels 4 more inches, the cycle will resume.

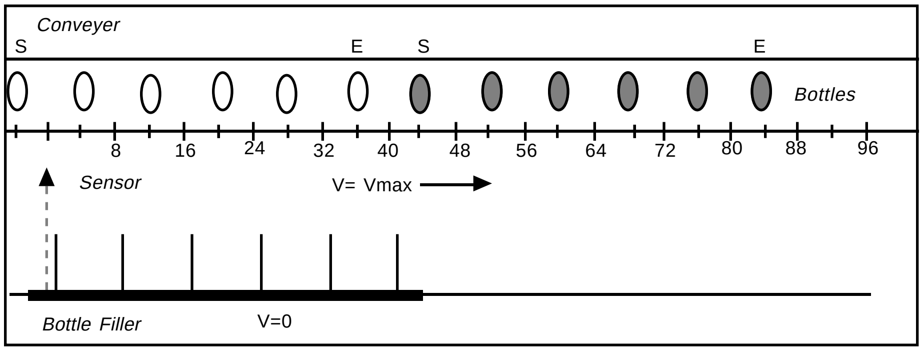


Figure 5-14. Bottle Filler Ready to Start a New Cycle

To program the application, use the following steps.

Step ① Determine the **FOR** value.

Primary conveyer axis steps per inch = 8000
Secondary bottle filler axis steps per inch = 25000

$$\textbf{FOR} = \frac{25000}{8000} = 3.125$$

Step ② Determine the **FAC** and **FEN** following acceleration parameters. The encoder sample period time (**TF**) is 1 ms.

$$\textbf{FEN} = 8000 \ \frac{\text{steps}}{\text{inch}} * 8 \ \frac{\text{inches}}{\text{second}} * \frac{1}{1000} = 64 \text{ steps}$$

$$\textbf{FAC} = \frac{100\% * 8000 \ \frac{\text{steps}}{\text{inch}} * 8 \ \frac{\text{inches}}{\text{second}}}{16000 \text{ steps}} * \frac{1}{1000} = 0.4$$

Step ③ Determine how far the primary and secondary axes will move during the dispensing part of the cycle. This includes the acceleration and deceleration parts of the move for the bottle filler when no fluid is dispensed.

$$Vp_{max} = 64000\frac{\text{steps}}{\text{sec}}$$

The dispensing takes 2.5 seconds. During the time that a constant following percentage occurs, the conveyer will have moved:

$$Dp_{con} = 64000\frac{\text{steps}}{\text{sec}} * 2.5 \text{ sec} = 160000$$

The secondary axis will move this distance * **FOR**:

$$Ds_{con} = 160000 \text{ primary encoder steps} * 3.125\frac{\text{secondary motor steps}}{\text{primary encoder steps}} = 500000 \text{ secondary motor steps}$$

The secondary axis or bottle filler axis will accelerate over 2 inches of the conveyer at maximum speed and decelerate over two inches. If the conveyer moves slower, the dispensing part of the process will become a smaller percentage of the total cycle. The distance the secondary axis travels during acceleration will be one half of the distance the conveyer or primary axis travels. The secondary axis starts from rest and accelerates to match the primary axis' speed. The secondary axis will move 1 inch during acceleration and 1 inch during deceleration. The total distance of the bottle filler move is:

$$D_{sec} = 500000 + 2 * (1 \text{ inch} * 25000 \frac{\text{steps}}{\text{inch}}) = 550000 \text{ steps (secondary motor steps)}$$

The value to be entered for the **AD** command for deceleration is:

$$\textbf{AD} = \frac{V^2}{2*D} = \frac{\left(8\frac{\text{inches}}{\text{sec}}\right)^2}{2*1 \text{ inch}} = 32 \frac{\text{inches}}{\text{sec}^2} * 1 \frac{\text{revolution}}{\text{inch}} = 32 \frac{\text{revs}}{\text{sec}^2}$$

If the primary axis' speed changes, the 500-FOL will still decelerate at this rate. The distance that the bottle filler moves will be identical, regardless of the primary axis' speed, so the bottle filler will still have to make a 22-inch move *back* while the primary axis moves 20 inches.

Step ④ You have determined the parameters necessary for the first part of the move. Next, you must determine the following percentage required to move the secondary axis back 22 inches while the primary axis moves 20 inches. Use Equation 5-4 to determine **FOL**.

Equation 5-4. FOL

$$\textbf{FOL} = \frac{D_{prim}}{200*K} - \sqrt{\left(\frac{D_{prim}}{200*K}\right)^2 - \frac{D_{sec}}{\textbf{FOR}*K}} \quad \text{Where } K = \frac{\textbf{FEN}}{100*\textbf{FAC}}$$

$$\textbf{FOL} = \frac{160000}{200*1.6} - \sqrt{\left(\frac{160000}{200*1.6}\right)^2 - \frac{550000}{3.125*1.6}} = 500 - 374.2 = 125.8\%$$

125.8% is the return following percentage.

Step ⑤ Enter the sequence below to implement the motion.

Command	Description
> **XE1**	Erases sequence #1
> **XD1**	Defines sequence #1
FOR3.125	Sets motor to encoder steps per unit travel ratio
FAC.4	Sets the following acceleration increment to .4 % per encoder period
FEN64	Sets the encoder period, which increases the percentage to 64 steps
D55ØØØØ	Sets the secondary axis move for the cycle to 550000 motor steps
FSI1	Enables following mode
AD32	Sets the deceleration to 32 rps2
IN1A	Defines input 1 as a trigger input
IN2D	Defines input 2 as a stop input
L	Starts a continuous loop
FOL1ØØ	Sets the initial following percentage to 100%
TR1	Waits on the input trigger
G	Starts motion
FOL125.8	Sets return move following percentage to 125.8%
H	Changes the direction
G	Starts the return following move
H	Changes direction again
N	Ends the loop—the following cycle will repeat
> **XT**	Ends the sequence

To decelerate using the **FAC** and **FEN** values, modify the program as follows:

Command	Description
> **XE1**	Erases sequence #1
> **XD1**	Defines sequence #1
SSH1	Sets save buffer on stop
FOR3.125	Sets motor to encoder steps per unit travel ratio
FAC.4	Sets the following acceleration increment to .4 % per encoder period
FEN64	Sets the encoder period, which increases the percentage to 64 steps
D55ØØØØ	Sets the secondary axis move for the cycle to 550000 motor steps
FSI1	Enables following mode
AD32	Sets the deceleration to 32 rps2
IN1A	Defines input 1 as a trigger input
IN2D	Defines input 2 as a stop input
L	Starts a continuous loop
FOL1ØØ	Sets the initial following percentage to 100%
TR1	Waits on the input trigger
MPP	Enters the profiling mode
G	Starts motion
FP176ØØØ	Waits until 176000 encoder pulses have passed
FOLØ	Stops the motion of the secondary
FP16ØØØ	Waits for the decel ramp distance
STOP	Stops the move itself
FOL125.8	Sets return move following percentage to 125.8%
H	Changes the direction
G	Starts motion
FP139782	Waits until 139782 encoder pulses have passed
FOLØ	Stops the motion of the secondary
FP2Ø218	Waits for the decel ramp distance
STOP	Stops the move itself
H	Changes direction again
N	Ends the loop—the following cycle will repeat
> **XT**	Ends the sequence

The value for **FP** during the return move is determined by calculating the distance the primary axis will move during acceleration and the constant following percentage portion and determining the distance it moves during the deceleration portion. This is determined from:

$$Dp_{acc} = \mathbf{FOL} * \frac{\mathbf{FEN}}{\mathbf{FAC}} = 125.8\% * \frac{64}{0.4} = 20218 \text{ encoder steps}$$

$$Dp_{dec} = Dp_{acc} = 20218 \text{ inches}$$

$$Dp_{con} = Dp_{prim} - Dp_{dec} - Dp_{acc} = \ 119564$$

$$Dp_{prim} = 20 \text{ inches } \ =160000$$

The *first* **FP** = Dp\S\DO2(con) + Dp\S\DO2(acc) = 119564 + 20218 = 139782. The *second* **FP** = Dp_{dec} = 20218. The second **FP** (20218 steps) measures the deceleration ramp. After deceleration, the move stops. The acceleration and deceleration ramps are based on the primary axis' speed.

Recede and Advance While Following

Receding and advancing while following requires position and velocity following. In this type of application, the secondary motor follows the primary encoder at a 1:1 ratio or at the same speed. The secondary motor has a specific positional or phase relationship with the primary encoder. This type of application is used when multiple operations (such as welds) must be performed on one moving part. The operations are performed at various places on the part, requiring the secondary axis to advance or recede.

In an ***advance application***, the secondary axis must accelerate and move a specific distance beyond the primary axis, then decelerate to a 1:1 ratio. The secondary axis moves a specific distance with respect to the primary axis while both axes are moving. In a ***recede application***, the secondary axis decelerates until it recedes a specific distance behind the primary axis and then resumes a 1:1 speed ratio with the primary axis.

The point at which the advance move or recede move occurs is based on a specific position on the primary axis or on an input trigger. This type of application requires that the following ratio be changed on-the-fly while based on either an input or the primary encoder's position. It also requires that the secondary axis be able to move a specific distance while the primary axis moves a corresponding specific distance. In this manner, the secondary can advance or recede a specific distance with respect to the primary axis.

To change the following ratio on-the-fly, you must use Motion Profiling (**MPP**) mode. You will need to measure the distance traveled by the primary encoder. Use the set of commands below.

Command	Description
> **FPn**	Delays command processing for n primary encoder steps
> **FPAn**	Delays command processing until the absolute count of the primary encoder has reached the value of n.
> **VAR1=FEP**	Allows you to read the value of the **F**ollowing **E**ncoder **P**osition into variable 1.

Advance Following Example

In this example, the primary axis has a 4000 count per revolution encoder. The secondary motor has a 4000 step per revolution motor. Therefore, the value for **FOR** is 1. The application requires that the secondary axis:

① Accelerate over 4000 primary encoder steps
② Move at a 1:1 ratio for 2 primary encoder revolutions
③ Advance (with respect to primary axis) 12000 primary encoder steps
④ Move 2 more primary encoder revolutions at 1:1 after advancing
⑤ Stop

The move profiles are shown in Figure 5-15.

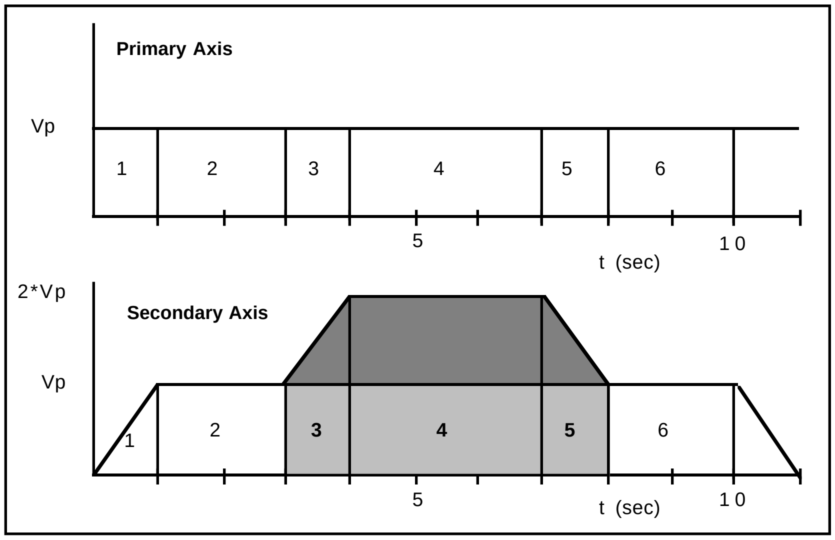


Figure 5-15. Advancing with Respect to the Primary While Following

The darkly shaded area in Figure 5-15 represents the distance that the secondary axis advances with respect to the primary axis. The lightly shaded area represents the distance that both the primary and the secondary axes move during the ***advance portion*** of the profile.

In these types of applications, the phase relationship or positional relationship is set with the **FAC** and **FEN** values (for as long as the following acceleration is performed). When the secondary axis accelerates from the 1:1 ratio to the 2:1 ratio, it will again accelerate at the following acceleration set. When it decelerates back to the 1:1 ratio, it decelerates in the same manner as it accelerates (using the **FAC** and **FEN** following acceleration rate). When the secondary axis decelerates back to *rest or zero speed*, it will decelerate at the **AD** rate. If the secondary axis must decelerate to zero speed at the **FAC** and **FEN** values, you must use **FOLØ** and set the following ratio to zero to make it stop.

Before programming this move profile, we will completely analyze the motion of the secondary and primary axes and then describe the sequence of commands necessary for performing the move.

In Figure 5-15, assume that the primary axis is moving at a maximum speed of 4000 steps/second. The area of each 1-second block represents 4000 primary encoder counts. Starting with Section #1, the primary encoder begins the section at a speed of $Vp_{max} = V_p$ = 4000 steps/sec. Therefore, the primary encoder moves 4000 counts during this section. From Section #1 of the secondary profile's plot, you can see that the secondary motor starts from rest and accelerates to a 1:1 speed ratio over 4000 counts of the primary encoder, Dp_{acc} = 4000 steps. You can calculate the desired **FAC** and **FEN** values to create such an acceleration ramp. This following acceleration will be used throughout the profile when changing from one following ratio to another. Set the encoder sample rate, **TF**, to 1 ms.

Vp_{max} = 4000 steps/sec

Dp_{acc} = 4000 steps

$$\mathbf{FEN} = Vp_{max} * \frac{\mathbf{TF}}{1000} = 4000\,\frac{\text{steps}}{\text{sec}} * \frac{1}{1000}\text{ sec} = 4 \text{ encoder counts}$$

$$\mathbf{FAC} = \frac{\text{FOL* Vpmax}}{Dp_{acc}} * \frac{\mathbf{TF}}{1000} = \frac{100\ \% * 4000\frac{\text{steps}}{\text{sec}}}{4000 \text{ steps}} * \frac{1}{1000}\text{ sec} = 0.1 \text{ percent}$$

By calculating the area under the secondary axis profile curve, you can determine that the secondary motor has moved 2000 motor steps. If the secondary axis is accelerating to the same speed as the primary axis, it will always travel half of the primary axis' distance. In this manner, the secondary axis will be physically aligned with this point when it reaches a 1:1 speed ratio. Figure 5-16 shows two conveyer belts—primary and secondary axes. In Figure 5-16, the locations of the primary and secondary axes spots at different times in the profile shown in Figure 5-15.

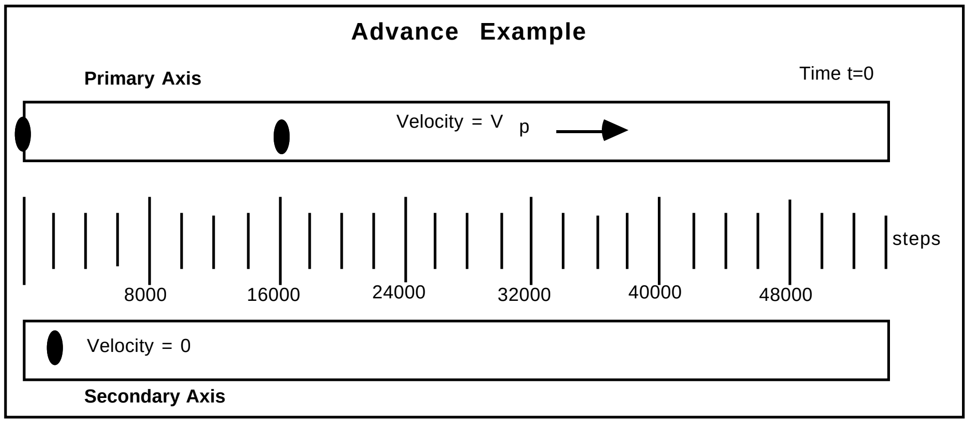


Figure 5-16. Advance Example at the Start of Section # 1

Usually, you will use an equation, not a graphic, to determine the distance traveled. Equation 5-5 calculates the distance the secondary axis travels for any acceleration, even if the secondary axis starts acceleration while it is already moving at a given ratio to the primary axis.

Equation 5-5. Ds_{acc}

$$Ds_{acc} = \mathbf{FOR} * \left(\frac{1}{2} * \Delta\mathbf{FOL}^2 * \frac{\mathbf{FEN}}{100\text{*FAC}} + \Delta\mathbf{FOL}\text{*}\mathbf{FOL}_I * \frac{\mathbf{FEN}}{100\text{*}\mathbf{FAC}}\right)$$

FOL= The change in following percentage

$\Delta\mathbf{FOL}_I$ = The initial following percentage

In Equation 5-5, the change in following percentage is the difference between the final following percentage that you are accelerating to and the following percentage you are starting from. If you are starting from rest, the initial following percentage is Ø and the change in following percentage is **FOLØ** or **FOL**. If you are at **FOL1ØØ** and want to accelerate to a following percentage of **FOL2ØØ**, the change in following percentage is 100 and the initial following percentage is 100. **FAC** and **FEN** are the values calculated from the equations for **FAC** and **FEN** using the Vmax and primary encoder acceleration distance.

The equation has two parts. The first part has the square of the following percentage change and the second part has a single following percentage change term. The first term determines the distance that the secondary axis travels due to the acceleration ramp portion of the curve. The second term gives the distance that the secondary axis travels due to its initial velocity. In this case, the secondary axis starts from rest (the second term contributes zero to the distance traveled). From the plot of the profile (Section # 1), the initial following percentage is Ø and the final percentage is 100, or the same speed as the primary axis. Using the general equation above, you should get 2000 steps, which is the area under the curve.

$$Ds_{acc} = 1^* \left(\frac{100^2}{2} * \frac{4}{100*0.1} + 100 * 0 * \frac{4}{100*0.1} \right) = \text{ 2000 secondary motor steps}$$

Therefore, in Section #1, the primary axis moved 4000 encoder counts and the secondary axis moved 2000 motor steps. We have determined the values for **FAC** and **FEN** based on the plots of the primary and secondary axes' profiles and the fact that V_{max} is 4000 steps/sec. At the end of Section #1, the spots will be in the locations shown in Figure 5-17 (t = 1 second).

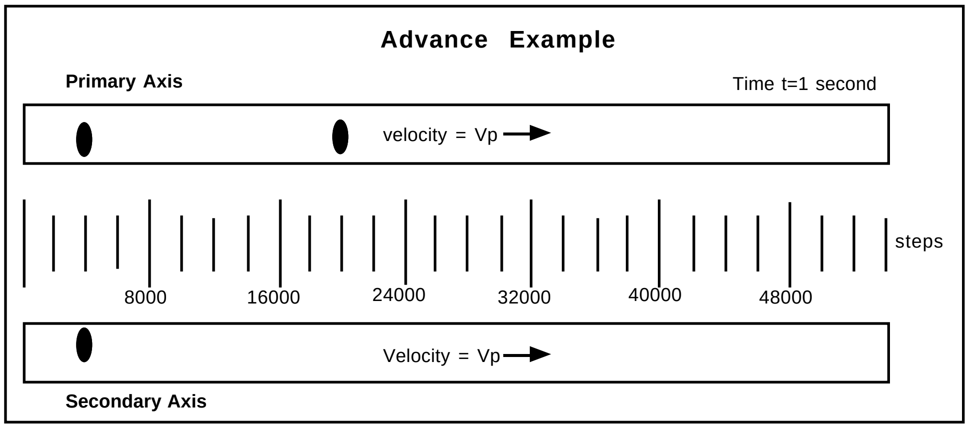


Figure 5-17. Advance Example—End of Section #1

In Section #2, the primary axis is moving at V_{max} and the secondary axis is moving at the same speed because the ratio is 1:1 (**FOL1ØØ**). Section #2 lasts for 2 seconds. The primary axis travels 8000 steps during this section. The secondary axis travels 8000 steps too. This can be determined from the profile plot by calculating the area beneath the curve for the section. The primary and secondary axes are lined up at the start of Section #2 and they travel at a 1:1 ratio for the duration of the section. At the end of Section #2, the primary axis has moved 12000 steps and the secondary axis has moved 10000 steps (thru Sections #1 and #2). Figure 5-18 shows the relative location of the spots at the end of Section #2 (t = 3 seconds).

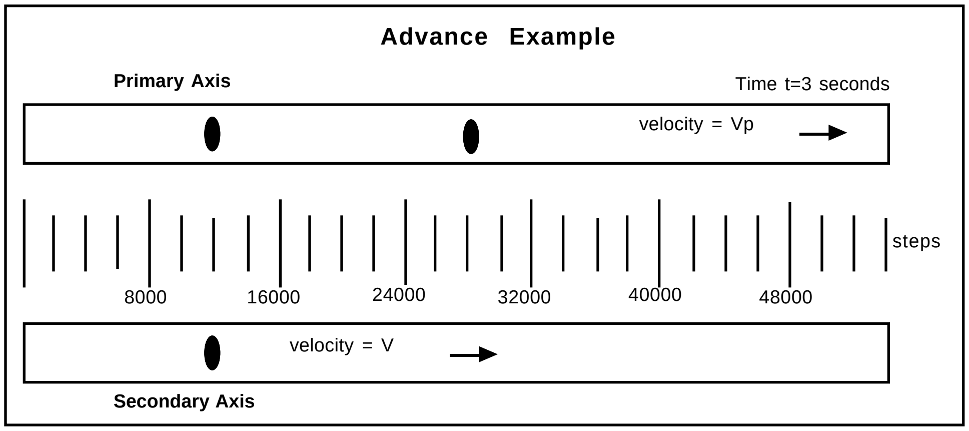


Figure 5-18. Advance Example—End of Section #2

In Section #3, the advance portion of the secondary profile begins. The secondary axis accelerates from a following percentage of 100% to 200% (a 2:1 ratio). Look at the plot of the profiles to graphically determine the distance that the primary and secondary axes have traveled. The primary axis moves 4000 steps and the secondary axis moves 6000 steps. The secondary axis' distance can also be determined from the equation above.

$$Ds_{acc} = 1 * \left(\frac{1}{2} * 100^2 * \frac{4}{100*0.1} + 100 * 100 * \frac{4}{100*0.1} \right) = 6000 \text{ steps}$$

At the end of Section # 3, the primary axis has moved 4000 steps since the beginning of the advance portion of the profile. The secondary axis has advanced 2000 steps with respect to the primary axis. Figure 5-19 shows the location of the spots at the end of Section #3.

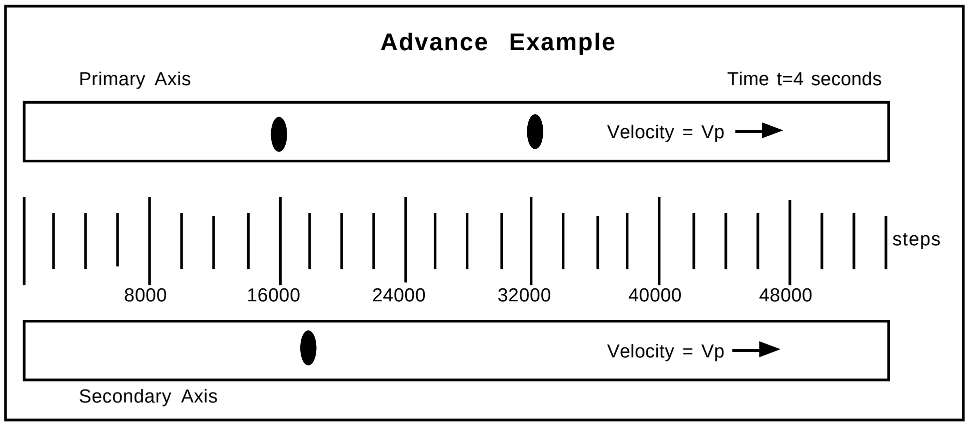


Figure 5-19. Advance Example—End of Section #3

At the start of Section #4, the secondary axis is at a following percentage of 200% and is moving at 2 * Vp (a 2:1 ratio). Section #4 lasts 3 seconds. The primary axis moves 12000 steps, while the secondary axis moves 24000 steps. The distance the secondary axis traveled can be determined from the equation below.

Equation 5-6. Ds_{con}

$$Ds_{con} = \mathbf{FOR} * \frac{\mathbf{FOL}}{100} * D_{prim} = 1 * \frac{200}{100} * 12000 = 24000$$

Since the start of the advance portion, the primary axis has moved a total of 16000 steps and the secondary axis has moved 30000 steps. The secondary axis has advanced 14000 steps with respect to the primary axis. Figure 5-20 shows the location of the spots at the end of Section #4.

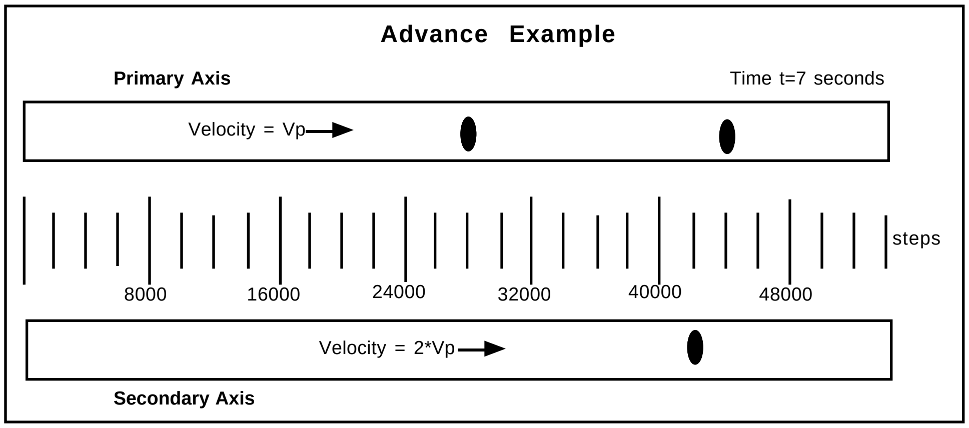


Figure 5-20. Advance Example—End of Section #4

During Section #5, the secondary axis decelerates to a following percentage of 100% (a 1:1 ratio). After it decelerates, it will have completed the advance portion of the profile. The primary axis travels 4000 steps in Section #5. The secondary axis travels 6000 steps. When the secondary axis accelerates from one following percentage to another, then decelerates to the original following percentage (as in this example), ***the secondary axis' acceleration distance will always equal the deceleration distance***. However, the deceleration distance can also be calculated from the following equation.

Equation 5-7. Ds_{dec}

$$Ds_{dec} = \mathbf{FOR} * \left(\frac{1}{2} * \Delta\mathbf{FOL}^2 * \frac{\mathbf{FEN}}{100 * \mathbf{FAC}} + \mathbf{FOL} \; * \mathbf{FOL}_I * \frac{\mathbf{FEN}}{100 * \mathbf{FAC}} \right)$$

When decelerating, the initial following percentage is the percentage you are at when you begin deceleration. In this example, it is 200%. The final following percentage is 100%. Therefore, the change in following percentage is a negative number (-100%).

$$Ds_{dec} = 1 * \left(\frac{1}{2} * (-100)^2 * \frac{4}{100*0.1} + -100 * 200 * \frac{4}{100*0.1} \right) = -2000 + 8000 = 6000 \text{ steps}$$

At the end of Section #5, the advance portion of the move profile is complete. The secondary axis is moving at a 1:1 speed ratio with the primary axis. The secondary axis has moved 36000 steps and the primary axis has moved 20000 steps during the advance portion. The secondary axis has advanced 16000 steps with respect to the primary axis. Figure 5-21 shows the locations of the axes at the end of Section #5.

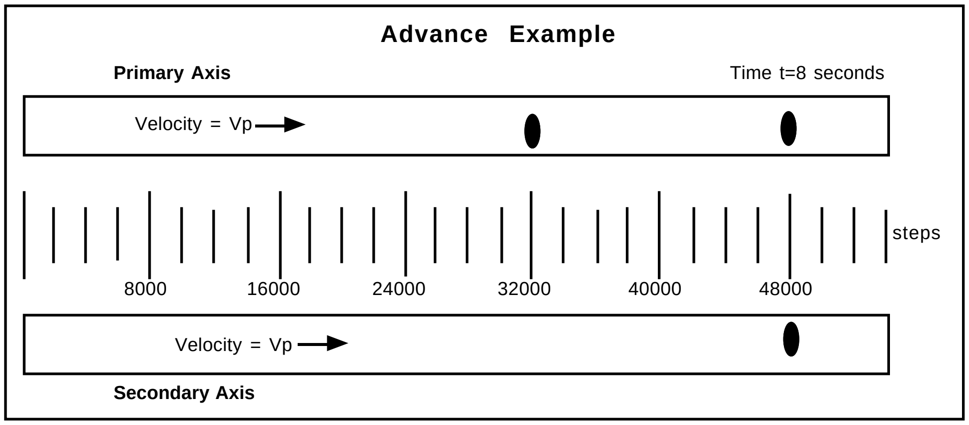


Figure 5-21. Advance Example—End of Section #5

During Section #5, the secondary axis travels at a 1:1 ratio until the deceleration ramp begins. If the move is a preset move, it will begin the deceleration ramp at the appropriate time based on the **AD** command. It will be at rest at the exact distance of the preset move. If the move is a continuous move, it will decelerate according to the **AD** value when a Stop (**S**) or Kill (**K**) command is encountered. From within a sequence, you can use the buffered Stop command. If you want the deceleration ramp to use the following acceleration value to decelerate, use the **FOLØ** command. After the secondary axis rests, a Stop command must be used to terminate the move. As depicted in the move profiles, the secondary axis moves at a 1:1 ratio for 8000 more steps, then decelerates to zero. During this time, the primary axis moves 8000 steps too.

Programming this profile requires the Following Encoder Distance Point (**FP**) or Following Encoder Absolute Point (**FPA**) command. **FP** is a delay-based on an incremental encoder distance. **FPA** is a delay-based on an absolute encoder distance.

FP Delay Example

To perform the advance move profile, breakpoints are needed to indicate when the secondary axis should accelerate to new following percentages. The **FP** and **FPA** commands define these breakpoints. These changes are performed on-the-fly and require the Motion Profiling mode. The breakpoints are the points at which acceleration or deceleration begin. This example will show the program using the **FP** command.

Step ①

Set up the 500-FOL with the appropriate encoder interface and enable the Following mode.

Command	Description
`> FSI1`	Enables 500-FOL's Following mode

Step ②

Set up the velocity following portion of the application. The number of secondary motor steps per unit of travel is 4000. The number of primary encoder steps per unit of travel is also 4000. Therefore, the **FOR** value is 1. The **FOL** command will be set up in the sequence for running the profile.

Command	Description
`> FOR1`	Relates the number of secondary motor steps for a distance to the number of primary encoder pulses for the same distance

Step ③

Set up the following acceleration value and enable the following acceleration mode. The values chosen are the same as were calculated in the explanation of the example above.

Command	Description
`> FAC.1`	Increases the following percentage to 0.1 for every change in encoder pulses by FEN
`> FEN4`	Sets the number of encoder pulses required before the following percentage is incremented by FAC.
`> FSF1`	Enables the use of following acceleration

Step ④ The breakpoints in the desired profile occur at the end of Section #2, the end of Section #4, and if **FAC** and **FEN** are used for deceleration, a breakpoint is set at the end of Section #6. If your **FOR** command differs from this example, the distance command is entered in terms of your secondary motor. The following sequence performs the desired profile. Each step of the sequence is explained. Enter the sequence.

Command	Description
> **XE1**	Erases Sequence #1
> **XD1**	Begins the definition of Sequence #1
D56ØØØ	The total distance the secondary moves is 56000 steps.
FOL1ØØ	The first following percentage to accelerate to is 100%.
MPP	Enters Motion Profiling mode so changes can be made on the fly.
G	The secondary motion begins.
FP12ØØØ	The first breakpoint occurs after the **primary axis** moves 12000 steps. **FP** causes command processing in the sequence to delay until 12000 primary encoder steps have been counted.
FOL2ØØ	After 12000 primary encoder steps the following percentage is changed to 200%. The **secondary axis** begins to accelerate at the following acceleration. Distance is known from previous analysis.
FP16ØØØ	Right after the command to begin acceleration to an **FOL** of 200%, the command processing is delayed until the **primary axis** has moved 16000 more pulses (from when **FP** is encountered and thus is an incremental encoder distance).
FOL1ØØ	After 16000 more primary encoder pulses, the following percentage changes to 100% and the **secondary axis** decelerates to a 1:1 ratio.
FP12ØØØ	Command processing is delayed 12000 more steps.
FOLØ	The secondary axis decelerates to an FOL of 0.
FP4ØØØ	It takes 4000 primary encoder steps to decelerate
STOP	A Stop command is needed because the 500-FOL would still think it was in a move and that the current following ratio was set to zero.
NG	Exits the Motion Profiling mode.
XT	Ends the definition of Sequence #1

Step ⑤ Typically, in an application that requires velocity and position following, it does not matter how you decelerate to a stop (you are moving from a synchronized state to rest). At the end of such a move, you will need to reverse direction and return to the starting location to repeat the profile. In this case, the optimal profile is as follows:

① Accelerate to a known positional relationship
② Perform the operation required at the synchronized speed
③ When it is complete, decelerate as fast as possible to repeat the cycle

You must decelerate at a rate unrelated to the primary encoder speed. The 500-FOL allows you to do this. This will also simplify the programming. The same profile is programmed below using the **AD** deceleration value.

Command	Description
> XE1	Erases Sequence #1
> XD1	Defines Sequence #1
D56ØØØ	The total distance the secondary moves is 56000 steps
FOL1ØØ	The first following percentage to accelerate to is 100%
MPP	Enters Motion Profiling mode so changes can be made on the fly
G	The secondary axis' motion begins
FP12ØØØ	Delays processing until primary axis moves 12000 encoder pulses
FOL2ØØ	Change to 200% following percentage
FPA16ØØØ	Delays processing until primary axis moves 16000 encoder pulses
FOL1ØØ	Change ratio back to 100%
NG	Exits the profiling mode
XT	Ends Sequence #1 definition

In this sequence, only two breakpoints are needed, the breakpoint to accelerate to 200% and then to decelerate back to 100%. Since the 500-FOL will decelerate at the value in **AD**, it will automatically decelerate to a distance of exactly 56000 secondary motor steps at the appropriate time.

Step ⑥

This step uses **FPA** instead of **FP**. **FPA** delays processing until the absolute value of the following encoder counter exceeds the **FPA** value.

Command	Description
> **XE1**	Erases Sequence #1
> **XD1**	Defines Sequence #1
D56ØØØ	The total distance the secondary moves is 56000 steps.
PFZ	Zero the following encoder counter.
FOL1ØØ	The first following percentage to accelerate to is 100%.
MPP	Enters Motion Profiling mode so changes can be made on the fly.
G	The secondary motion begins.
FPA12ØØØ	Delay command processing until primary encoder count exceeds 12000 encoder pulses.
FOL2ØØ	Change to 200% following percentage.
FP28ØØØ	Delay command processing until primary encoder counter exceeds 28000 encoder pulses or an incremental change of 16000 pulses.
FOL1ØØ	Change ratio back to 100%.
NG	Exits the profiling mode.
XT	Ends Sequence #1 definition

In Steps ⑤ and ⑥, the encoder is counting in the positive direction. If the encoder is counting in the negative direction, a negative sign is required for both the **FP** and **FPA** commands. Use either the **FP** or the **FPA** command, depending on your application.

*Use **FPA** when repetitive cycles of the same move profile are done without a trigger to start each cycle.* By making the delays dependent on an absolute encoder position, there is no accumulative error. In many cases, a trigger input from a sensor is used to start the move profile that is repeated.

*If you use a trigger to start the move each time, use **FP** and the trigger will remove any accumulative error.* The following sequence illustrates the uses of the **FPA** command and a variable to perform a repetitive move that does not use a trigger to start it. This case is more like a cam cycle and the position relationship must be maintained while the cycle repeats.

Command	Description
`> XE1`	Erases Sequence #1
`> XD1`	Defines Sequence #1
`VAR1=12000`	Sets variable 1 equal to the first breakpoint.
`VAR2=28000`	Sets variable 2 equal to the second breakpoint
`VAR3=0`	Sets variable 3 equal to the primary reference point
`D56000`	Sets distance of secondary axis move to 56000 steps
`PFZ`	Zeroes the following encoder counter
`L`	Begins the continuous loop of the profile cycle
`FOL100`	The first following percentage to accelerate to is 100%
`MPP`	Enters Motion Profiling mode so changes can be made on-the-fly
`FPA(VAR3)`	Variable 3 synchronizes the move start with the primary axis. For the first 40000 primary steps, the secondary axis is moving, then it moves back during the next 40000 steps of the primary then it repeats.
`G`	Begins secondary axis motion.
`VAR3=VAR3+80000`	Sets variable 3 equal to the start of the next cycle
`FPA(VAR1)`	Delays command processing until primary encoder count exceeds 12000 + 56000*n* encoder pulses, *n* = # of times through the loop.
`FOL200`	Changes following percentage to 200%.
`VAR1=VAR1+80000`	Sets variable 1 equal to the first breakpoint for the next cycle
`FPA(VAR2)`	Delays command processing until primary encoder count exceeds 28000 + 56000*n* encoder pulses, *n* = the # of times through the loop.
`FOL100`	Changes ratio back to 100%.
`VAR2=VAR2+80000`	Sets variable 2 equal to the second breakpoint for the next cycle
`NG`	Exits the profiling mode and complete the 56000 step move
`FOL200`	Sets the following ratio to a higher speed to move back to the starting point at a fast speed
`H`	Changes direction
`G`	Moves back to the starting point.
`H`	Changes direction
`N`	Repeats the cycle
`XT`	Ends Sequence #1 definition

In this example, the move profile is repeated. One cycle consists of the following events:

① Secondary axis moves 56000 steps while primary axis moves 40000 steps.

② The secondary axis retreats to the start and after another 40000 primary encoder steps the cycle is repeated.

③ No operation during the secondary axis' retreat.

④ The retreat is set to a high following ratio to get the secondary axis back to the start before the primary axis moves 40000 steps.

This cycle is very similar to a cam cycle (which will be described in the next section). You can load the **FP** command with a variable (like the **FPA** command). You can check the following encoder counter value at any time by loading it into a variable.

Command	Description
`> VAR1=FEP`	Loads variable 1 with the value of the following encoder counter

Calculating FOL, FP, or FPA For An Advance or Recede Application

The advance example explained how an advance move is made and how the different commands (**FOL**I, **FAC** and **FEN**) contribute to the move. This section provides some simple formulas that you can use to set up such a profile. To do position and velocity following, you must be able to accelerate the secondary axis to a known position with respect to the primary axis. This is what determines the **FAC** and **FEN** following acceleration values.

Once you determine these values, you will use them to calculate the acceleration ratio that you must use to make the advance move. The following information will help you understand the move profile in the example.

- ❏ **FAC**: Following speed percentage increment
- ❏ **FEN**: Change in primary encoder pulses to cause an increment of **FAC**

  - D_{prim}: The distance the primary axis will travel during the advance portion of the secondary move profile. This is 20000 primary encoder steps in the example above.
  - D_{sec}: The distance the secondary must advance with respect to the moving primary, measured in primary encoder steps. In the example above this is 16000 primary encoder steps.
  - $\mathbf{FOL_I}$: The initial following percentage that you will accelerate from to the new following percentage.

In applications that require an advance move, you will usually know the distance that you want to advance with respect to the primary axis and the distance the primary axis will move during the advance. The distance that you want the secondary axis to advance with respect to the primary axis is given in terms of primary encoder steps. The distance can be converted from secondary motor steps to primary motor steps (and vice versa) with the **FOR** command. After determining the parameters listed above, you can use the following formula to determine the following percentage you must accelerate to.

First, determine a following acceleration constant (**K**) to simplify the equations.

Equation 5-8. Following Constant

$$K = \frac{\mathbf{FEN}}{100*\mathbf{FAC}}$$

The constant **K** is used in Equation 5-9 to determine **FOL**.

Equation 5-9. FOL

$$\mathbf{FOL} = \mathbf{FOL}_I + \frac{D_{prim}}{200*K} - \sqrt{\left(\frac{D_{prim}}{200*K}\right)^2 - \left(\frac{D_{sec}}{K}\right)}$$

With an advance move, the value of $\mathbf{FOL}_I$ will always be 100. Apply the formula to the example above (the following percentage should be 200%). This is the following percentage that you must attain to advance 16000 steps with respect to the primary axis, while the primary axis moves 20000 steps.

D_{prim} = 20000 primary encoder steps
D_{sec} = 16000 primary encoder steps
FAC = 0.1
FEN = 4
$\mathbf{FOL}_I$ = 100

We will first determine the following acceleration constant K.

$$K = \frac{4}{100*0.1} = 0.4$$

We will now determine **FOL**.

FOL is the same as $\mathbf{FOL}_F$ in the equations used earlier for determining the distances traveled by the primary and secondary axes.

$$\mathbf{FOL} = 100 + \frac{20000}{200*0.4} - \sqrt{\left(\frac{20000}{200*0.4}\right)^2 - \left(\frac{16000}{0.4}\right)}$$

$$= 100 + 250 - \sqrt{(250)^2 - 40000} = 350 - \sqrt{22500} = 350 - 150 = 200 = \mathbf{FOL}_F$$

The following percentage that must be accelerated to is 200%. Now calculate what the breakpoint is for decelerating back to a 1:1 ratio or 100%. The **FP** value will be determined from the Equation 5-10:

Equation 5-10. Following Breakpoint

$$\mathbf{FP} = D_{prim} - \frac{\mathbf{FEN}}{\mathbf{FAC}} * \left(\mathbf{FOL}_F - \mathbf{FOL}_I\right) = 20000 - \frac{4}{0.1} * (200 - 100) = 16000 \text{ steps}$$

The value you would use for **FP** is 16000. The breakpoint at which you begin the advance portion of the move profile was not calculated. This value varies from application to application and you may want to use a trigger to begin the advance move rather than **FP**. An example of using a trigger to begin an advance move is described below.

Example

Configure an input as a trigger input with the **IN** command.

Command	Description
> **IN1A**	Configures input #1 as a trigger input
> **IN2D**	Configures input #2 as a stop input
> **XD1**	Defines Sequence #1
MC	We will make this a continuous move
FOL1ØØ	The first following percentage to accelerate to is 100%
MPP	Enters Profiling mode so changes can be made on-the-fly
G	Initiates motion
TR1	Command processing pauses until input #1 (trigger input) is activated. The secondary axis will move continuously at a speed percentage of 100% (with respect to the primary axis)
FOL2ØØ	The following percentage is changed to 200%. Acceleration begins.
FP16ØØØØ	Command processing will delay 16000 primary encoder steps.
FOL1ØØ	The following percentage is changed to 100%. Deceleration begins.
NG	Ends the Profiling mode.
XT	End Sequence #1 definition

In this sequence, the secondary axis will begin moving at a 100% speed percentage. When trigger input #1 is activated, the secondary axis will advance 16000 steps with respect to the primary axis. It will then decelerate to a 100% or 1:1 ratio and continue until the stop input (input #2), is activated (or a Stop [**S**] command is issued).

Recede vs Advance

Recede moves are similar to advance moves. In the illustration of the spots for the advance example, the secondary axis synchronized with the first primary axis, then receded while the primary axis moved. This motion can be analyzed in the same manner as the advance move with the exception that a different equation is used to determine the required value for **FOL**. Again, you will have to provide the distance that the primary axis will move during the recede move and the distance with respect to the primary axis that the secondary axis must recede (measured in terms of primary encoder steps). For example:

- ❑ A primary encoder has a resolution of 4000 steps/rev (1 rev = 1 inch)
- ❑ The secondary motor has 25000 steps/rev and also has 1 rev equal to 1 inch
- ❑ The **FOR** command is set to 6.25.

Typically you will know what distance you want the secondary axis to recede. If the secondary is to recede 1.5 inches with respect to the primary axis while the primary axis moves 3.5 inches, set D_{prim} and D_{sec} equal to:

$$D_{prim} = 3.5" = 3.5" * 4000\frac{\text{steps}}{\text{inch}} = 14000 \text{ primary steps}$$

$$D_{sec} = 1.5" = 1.5" * 25000\frac{\text{steps}}{\text{inch}} = 37500 \text{ secondary motor steps } = \frac{37500}{6.25} = 6000 \text{ primary encoder steps}$$

Both distances are provided in primary encoder steps. The two terms are used to determine the required **FOL** in Equation 5-11.

Equation 5-11. FOL

$$FOL = FOL_I - \frac{D_{prim}}{200*K} + \sqrt{\left(\frac{D_{prim}}{200*K}\right)^2 - \frac{D_{sec}}{K}}$$

Where K is the following acceleration constant (as in Equation 5-8).

$$K = \frac{FEN}{100 * FAC}$$

To calculate the **FOL** value, determine a value for **FAC** and **FEN**. This will depend on your application's maximum velocity and either the maximum acceleration for the secondary axis or the distance the primary axis travels while the secondary axis must accelerate. Use the same **FAC** and **FEN** values from the advance example:

FAC = 0.1
FEN= 4

In an advance or recede application, the initial following percentage FOL_I will always be 100.

The value for **FOL** produced by the equation above is:

$$\mathbf{K} = \frac{4}{100*0.1} = 0.4$$

$$\mathbf{FOL} = 100 - \frac{14000}{200*0.4} + \sqrt{\left(\frac{14000}{200*0.4}\right)^2 - \frac{6000}{0.4}} = 100 - 175 + \sqrt{(175)^2 - 15000} = 50\%$$

The value needed for **FP** can be determined from Equation 5-12.

Equation 5-12. FP

$$\mathbf{FP} = D_{prim} + \frac{\mathbf{FEN}}{\mathbf{FAC}} * \left(\mathbf{FOL}_F - \mathbf{FOL}_I\right) = 14000 + \frac{4}{0.1} * (50 - 100) = 12000 \text{ primary enc. steps}$$

The move profile is shown in Figure 5-22.

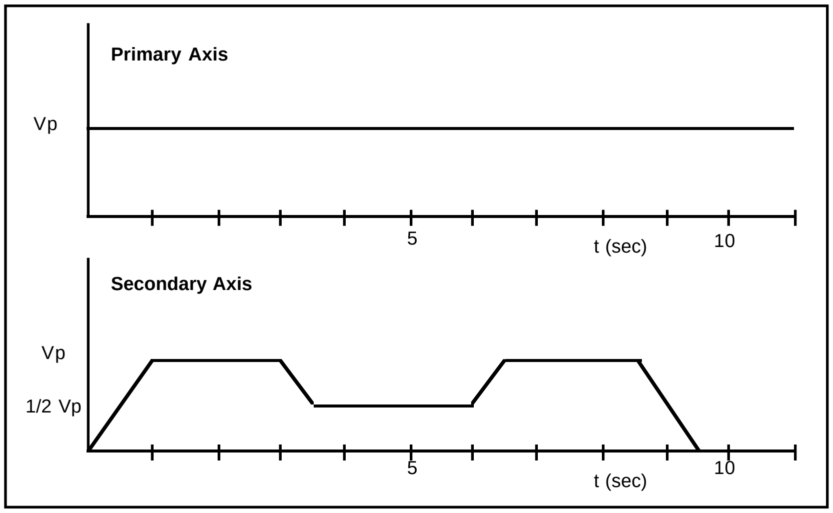


Figure 5-22. Recede While Following Profile

The sequence that will execute this profile is provided below:

Command	Description
`> FACØ.1`	Sets the percentage increment to 0.1
`> FEN4`	Sets the number of encoder counts for an increment to 4
`> FSF1`	Enables following acceleration
`> FOR6.25`	Sets the secondary motor steps per unit distance to primary encoder steps per unit distance ratio
`> XD1`	Defines Sequence #1
`MN`	Normal mode
`D175ØØØ`	Sets distance to 175000 steps
`FOL1ØØ`	The first following percentage to accelerate to is 100%
`MPP`	Enters Motion Profiling mode so changes can be made on-the-fly
`G`	Initiates motion
`FP12ØØØ`	The command processing will pause here until the primary encoder has moved 12000 steps. The secondary will then decelerate to 50%
`FOL5Ø`	Changes following percentage to 50%—the recede portion begins
`FP12ØØØ`	Command processing will delay 12000 primary encoder steps.
`FOL1ØØ`	The following percentage is changed to 100%—deceleration begins
`NG`	Ends the Motion Profiling mode
`XT`	End sequence definition

Cam Following

A common application that requires velocity and position following is the simulation of a cam or an electronic cam. To simulate the motion produced by a cam, you must satisfy the following requirements:

- ❏ Follow both the position and the velocity of a primary encoder.
- ❏ You must also be able to change following ratios during motion and still maintain a positional relationship.
- ❏ Change ratios based on primary encoder distance.
- ❏ Must be able to keep track of the primary encoder position even if the secondary axis is not moving.
- ❏ Must be able to continuously repeat a cam cycle without developing accumulative error.

You can simulate a cam profile electronically using the commands and equations developed earlier. Motion Profiling mode (**MPP**) is required for cam following. For more information on Motion Profiling mode, refer to *Chapter 4, Application Design*. Figure 5-23 shows a typical cam profile.

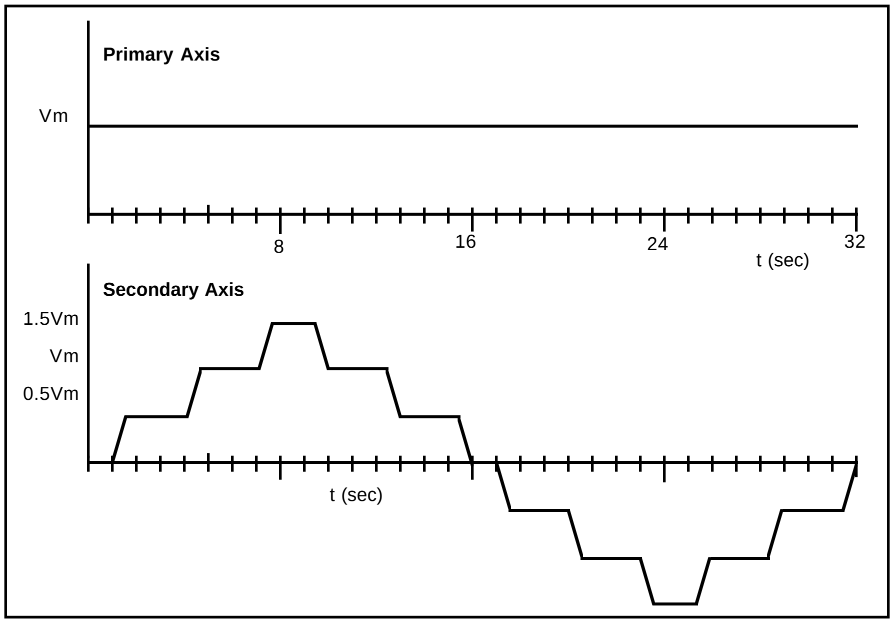


Figure 5-23. Cam Following

In this example, the encoder is a 4000 pulse per revolution encoder and it is mounted on the primary axis. The *secondary axis* will perform an electronic cam cycle, which consists of the following steps:

① Sitting at rest for one primary encoder revolution, then executing a step profile of 3 following percentages from 50% to 100%, then to 150%, and back down to 0%.

② It will then delay 1 primary encoder revolution and perform the same profile in the opposite direction.

③ This cycle is to be repeated until a stop is issued. The secondary axis will accelerate at its maximum rate when the primary axis is at its maximum velocity. Table 5-1 defines the cam cycle for this profile.

	Primary Position		Secondary Position		
Segment	Absolute	Incremental	Absolute	Incremental	Following %
	0	0	0	0	
①	4000	0	0	0	0%
②	16000	12000	5500	5500	50%
③	28000	12000	17000	11500	100%
④	38000	10000	31500	14500	150%
⑤	50000	12000	44000	12500	100%
⑥	62000	12000	50500	6500	50%
⑦	68000	6000	51000	500	0%

Table 5-1. Cam Cycle

The following percentage is given for each segment. Each of the distance points is a breakpoint where the following percentage changes. Use the following acceleration to change from one following percentage to the next. The data needed to program this type of profile is listed below.

Vp_{max}: Maximum velocity of the primary axis
As_{max}: Maximum acceleration of the secondary axis
FOR: Relative resolutions per unit of distance for the primary and secondary axes
FAC: Following acceleration value, percentage increments
FEN: Following acceleration value, encoder counts for an increment

For each ratio segment, you must know the distance the primary axis will travel and the corresponding distance the secondary axis will travel. The breakpoints can be determined from the graph. Usually, you will not be able to graphically describe the motion relationship, and will simply know that you want the primary axis to move ***x*** steps and the secondary axis to move ***y*** steps in the same time frame.

After you determine **FAC** and **FEN** from your maximum acceleration and maximum velocity, or from the distance you want the secondary axis to accelerate over and the maximum velocity, you can use **FAC** and **FEN** and the primary and secondary axes' travel distances to determine the following percentages and the values for the breakpoints to change to new following percentages. When accelerating to a higher following percentage, you can use Equation 5-13 to determine the required **FOL** percentage that you must accelerate to using the **FAC** and **FEN** values you have determined.

Equation 5-13.
FOL ACCEL

$$\textbf{FOL} = \textbf{FOL}_I + \frac{D_{prim}}{100*K} - \sqrt{\left(\frac{D_{prim}}{100*K}\right)^2 + \frac{2*\textbf{FOL}_I*D_{prim}}{100*K} - \frac{2*D_{sec}}{\textbf{FOR}*K}}$$

Where K is the following acceleration constant determined by **FAC** and **FEN** in Equation 5-8.

$$K = \frac{\textbf{FEN}}{100*\textbf{FAC}}$$

If you are decelerating to a lower following percentage, use Equation 5-14 to determine the following percentage you must use to move the secondary axis the specified number of steps for the corresponding motor steps.

Equation 5-14.
FOL DECEL

$$\textbf{FOL} = \textbf{FOL}_I + \frac{D_{prim}}{100*K} + \sqrt{\left(\frac{D_{prim}}{100*K}\right)^2 - \frac{2*\textbf{FOL}_I*D_{prim}}{100*K} + \frac{2*D_{sec}}{\textbf{FOR}*K}}$$

In both the accelerating and decelerating equations, the terms listed below for the primary encoder steps and the secondary motor steps are required.

D_{prim} =	The number of primary encoder steps that the motor will move in the segment.
D_{sec} =	The number of corresponding secondary motor steps that the secondary will move during which time the primary encoder will move D_{prim}.

To illustrate the programming of the profile above, we will assign values to the application's requirements. The maximum velocity, acceleration and the **FOR** value are to be determined by the application. The **FAC** and **FEN** values are calculated. The maximum velocity and acceleration are given below. The example has the following parameters:

Primary encoder resolution	4000 steps/revolution
Secondary motor resolution	25000 steps/revolution

1 encoder rev = 1 motor rev
FOR 6.25
Vp_{max} = 4000 steps/second
As_{max} = 1 rev/second ^{2}
Encoder sample period TF= 1 ms

From the equations in the Velocity and Position following section **FAC** and **FEN** are determined:

$$\textbf{FEN} = V_{max} * \frac{\textbf{TF}}{1000}$$

$$\textbf{FAC} = \frac{\textbf{FOL}*Vp_{max}}{Dp_{acc}} * \frac{\textbf{TF}}{1000}$$

Since we are starting with Vp_{max} and As_{max}, we must determine D_{acc} for the equations above.

$$Dp_{acc} = \frac{Vp_{max}^2 * \textbf{FOR} * \textbf{FOL}}{As_{max} * 100} = \frac{\left(4000\frac{steps}{sec}\right)^2 * 6.25 * 100}{25000\frac{steps}{sec^2} * 100} = 4000 \text{ primary encoder steps}$$

Vp_{max} is in primary enc. steps/sec
As_{max} is in secondary motor steps/sec^2

The **FOR** term converts the acceleration units to primary encoder steps units. **FAC** and **FEN** can now be calculated.

$$\textbf{FEN} = 4000\,\frac{\text{steps}}{\text{sec}} * \frac{1\text{ ms}}{1000} = 4\text{ steps}$$

$$\textbf{FAC} = \frac{100 * 4000\frac{\text{steps}}{\text{sec}}}{4000\text{ steps}} * \frac{1\text{ ms}}{1000} = 0.1$$

You can now use these equations to determine the **FOL** value for each segment of primary encoder distance and secondary motor distance. The **FOL** values are already given in Table 5-1, but it will be illustrative to determine the **FOL** required for some of the segments. Evaluate **FOL** for Segments #4 and #6. Typically, you will know the distance you want the primary axis to move and the corresponding distance that you want the secondary axis to move. With this data, you can create a table like Table 5-1. You will have to enter the **FOL** values for the table from the equations given in this chapter.

Segment ④

D_{prim} = 10000 primary encoder steps
D_{sec} = 90625 secondary motor steps

Using the equation for accelerating, we can evaluate for **FOL** in Segment #4.

$$\text{The acceleration constant K} = \frac{4}{100*0.1} = 0.4$$

$$\textbf{FOL} = 100 + \frac{10000}{100 * 0.4} - \sqrt{\frac{2*100*10000}{100*0.4} + \left(\frac{10000}{100 * 0.4}\right)^2 - \frac{2*90625}{6.25*0.4}}$$

$$= 100 + 250 - \sqrt{50000 + (250)^2 - 72500} = 150\%$$

The breakpoint is given automatically by the table and is 10000 for **FP** and is 38000n for **FPA** where n is the number of cycles completed thus far.

Segment ⑥

D_{prim} = 12000 primary encoder steps
D_{sec} = 40625 secondary motor steps

Using the equation for decelerating to a lower percentage we can calculate the value for **FOL** for Segment 6.

$$\textbf{FOL} = 100 - \frac{12000}{100*0.4} + \sqrt{\left(\frac{12000}{100*0.4}\right)^2 - \frac{2*12000*100}{100*0.4} + \frac{2*40625}{6.25*0.4}}$$

$$= 100 - 300 + \sqrt{(300)^2 - 60000 + 32500} = 50\%$$

The breakpoints for **FP** and **FPA** are 12000 and 62000***n*** steps respectively. In many cases involving a cam cycle, a trigger is not used to start each cycle and the repetition of the cycle is based on the primary encoder. In these situations, use the **FPA** command (it is based on the following encoder's absolute count). The absolute count comes from a hardware counter that can be accessed by assigning it to a variable:

`> VAR1=FEP`	**FEP** is the value in the hardware counter, it is a read only value

The following sequence will perform the cam profile. The secondary axis will be put in a continuous move. Two parts occur in a cycle. The first part moves the stepped profile in one direction. The second part reverses direction and returns to the start to repeat the cycle.

Command	Description
`> FOR6.25`	Sets the secondary motor steps to primary encoder steps ratio
`> FACØ.1`	Sets the change in following percentage for following acceleration
`> FEN4`	Sets the number of encoder pulses required to change by **`FAC`**
`> FSF1`	Enables following acceleration
`> FSI1`	Enables following
`> SSH1`	Saves buffer on stop
`> VAR1=64ØØØ`	Variable for the incrementing the cycle
`> VAR2=4ØØØ`	Breakpoint 1
`> VAR3=16ØØØ`	Breakpoint 2
`> VAR4=28ØØØ`	Breakpoint 3
`> VAR5=38ØØØ`	Breakpoint 4
`> VAR6=5ØØØØ`	Breakpoint 5
`> VAR7=62ØØØ`	Breakpoint 6
`> VAR8=64ØØØ`	Breakpoint 7
`> 1XE1`	Erases Sequence #1
`> 1XD1`	Defines Sequence #1
`MC`	Enables continuous mode
`FOLØ`	Sets the current following percentage to 0%
`L`	Begins the loop cycle
`MPP`	Enters the Motion Profiling Mode
`G`	Initiates motion
`FPA(VAR2)`	Pauses execution until absolute value of the primary encoder counter exceeds breakpoint 1
`FOL5Ø`	Following % is changed to 50% or 1/2 as fast as the primary motor
`VAR2=VAR2+VAR1`	Set variable 2 to the breakpoint value for the next cycle
`FPA(VAR3)`	Pauses execution until absolute value of the primary encoder counter exceeds breakpoint 2
`FOL1ØØ`	Speed ratio is changed to 1:1
`VAR3=VAR3+VAR1`	Set variable 3 to the breakpoint value for the next cycle
`FPA(VAR4)`	Pauses execution until absolute value of the primary encoder counter exceeds breakpoint 3
`FOL15Ø`	Speed ratio is changed to 1.5:1
`VAR4=VAR4+VAR1`	Set variable 4 to the breakpoint value for the next cycle
`FPA(VAR5)`	Pauses execution until absolute value of the primary encoder counter exceeds breakpoint 4
`FOL1ØØ`	Speed ratio is changed to 1:1
`VAR5=VAR5+VAR1`	Set variable 5 to the breakpoint value for the next cycle
`FPA(VAR6)`	Pauses execution until absolute value of the primary encoder counter exceeds breakpoint 5
`FOL5Ø`	Speed ratio is changed to 5:1
`VAR6=VAR6+VAR1`	Set variable 6 to the breakpoint value for the next cycle
`FPA(VAR7)`	Pauses execution until absolute value of the primary encoder counter exceeds breakpoint 6
`FOLØ`	Speed ratio is changed to 0
`VAR7=VAR7+VAR1`	Set variable 7 to the breakpoint value for the next cycle
`FPA(VAR8)`	Pauses execution the absolute value of the primary encoder counter exceeds breakpoint 7
`STOP`	Ends the move (this is required)
`NG`	Ends Motion Profiling mode
`VAR8=VAR8+VAR1`	Set variable 8 to the breakpoint value for the next cycle
`H`	change the direction
`N`	Ends the loop cycle
`XT`	Ends Sequence #1 definition

This sequence is an example of a complex following profile. Position and velocity are synchronized and the positional relationship is maintained.

Position Maintenance While in the Following Mode

In addition to using one encoder for following, you can also use an additional encoder for position maintenance. As depicted in Figure 5-24, the *following encoder* must be coupled to the primary motor (whose motion you want to follow) and the *position encoder* must be coupled to the secondary motor to perform position maintenance when a move is completed.

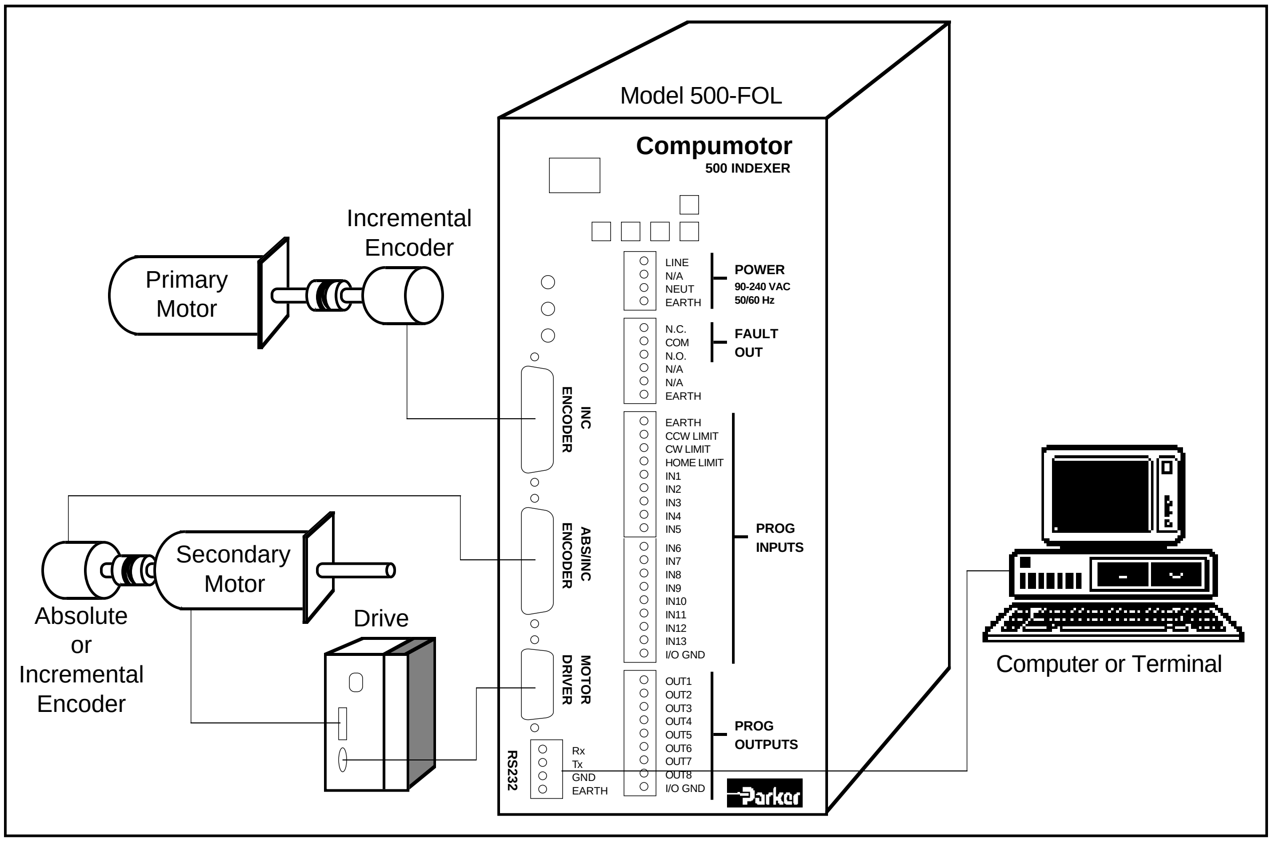


Figure 5-24. Configuration for Position Maintenance While Following

Sample Application and Commands

The following example performs a following move with position maintenance at the end of the move. The configuration for this example is shown in Figure 5-24 above.

Step ① Enter the appropriate **FSJ** and **FSM** commands for your encoder interface selection.

Command	Description
> FSJØ	Encoder connected to **INC ENCODER** now set for following; Encoder connected to **ABS/INC ENCODER** now set for positioning
> FSM1	Sets the **ABS/INC ENCODER** interface for absolute encoder

Step ② Set up position maintenance with the following commands:

Command	Description
> 1CPE2ØØØ	Maximum allowable position error is 2000 motor steps
> 1CPG5Ø	Sets the position gain to 50 % of the Maximum gain (**CPM**)
> 1CPM6Ø	Maximum position gain is 60
> 1MV1Ø	Sets the maximum allowable correction velocity to 10 rps
> 1DW5	Sets the Deadband window to 5 steps
> 1FSD1	Invokes the stall detect mode
> 1FSC1	Invokes the position maintenance mode

The Model 500 is now set up to perform position maintenance after a move is completed.

Step ③ Type the following commands to set up and perform a following preset move:

Command	Description
> **1FSI1**	Invokes the following mode
> **A1Ø**	Sets the acceleration to 10 rps^2
> **AD1Ø**	Sets the Deceleration to 10 rps^2
> **FOL1ØØ**	Sets the % of the primary speed that the secondary will move at — in this case, a 1:1 ratio
> **TF1**	Sets the sample period time for the encoder sampling to 1 millisecond
> **D5ØØØØ**	Sets the distance that the Model 500 will move to 50,000 steps
> **G**	Initiates motion

The Model 500 will now move 50,000 steps at a speed that is a 1:1 ratio of the primary's speed. At the end of the move, position maintenance will be performed on the secondary motor.

Synchronization

The 500-FOL can synchronize its speed and phase with respect to a primary axis. In many applications, it is necessary to have the position and speed of a secondary axis synchronized with the speed and position of the primary axis with registration marks on the secondary axis parts or material. These marks must be evenly spaced so that at a constant speed (with respect to the primary axis) the number of primary axis encoder steps recorded between registration marks is an expected constant number. If these marks should come further apart (e.g., the material stretches) the 500-FOL will adjust the speed ratio to correct for the error between the registration marks. Figures 5-25 through 5-27 illustrate this process.

In Figure 5-25, a secondary axis has parts that are to be synchronized to the primary axis' parts. The registration sensor detects the location of the parts with respect to the primary axis. This sensor goes to the 500-FOL. It indicates the start of the part. The 500-FOL then counts the encoder pulses from the primary axis that occur between registration marks.

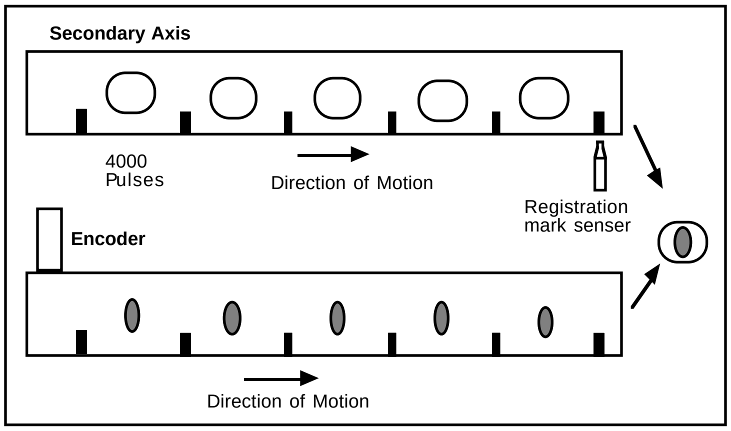


Figure 5-25. Secondary Parts Synchronized With the Primary Axis

If the material on the secondary axis stretches, as indicated in Figure 5-26, all parts after the stretched material will no longer be synchronized with the primary axis' parts. Figure 5-26 depicts the result of not using the Synchronization mode.

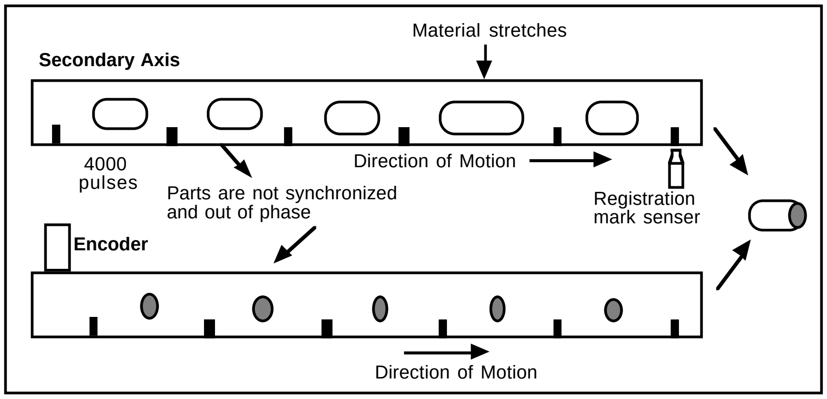


Figure 5-26. Material Stretches—Parts Out of Sync

With the 500-FOL's Synchronization mode, the secondary axis accelerates to re-synchronize with the primary axis. It removes the phase shift between the two axes. The 500-FOL detects that the number of pulses between the registration marks has increased (due to stretching). The speed ratio is increased, so the secondary axis speeds up. The material after the stretched portion is good material, so the pulses between the next two registration marks will be slightly less than 4000 because the speed ratio is now higher. The speed will now be reduced. The secondary and primary axes will be synchronized again. Every time a registration mark is encountered, a new actual count is latched and the speed ratio is adjusted to synchronize the axis. Corrections will continue until the secondary axis again has the expected number of pulses between registration marks (see Figure 5-27).

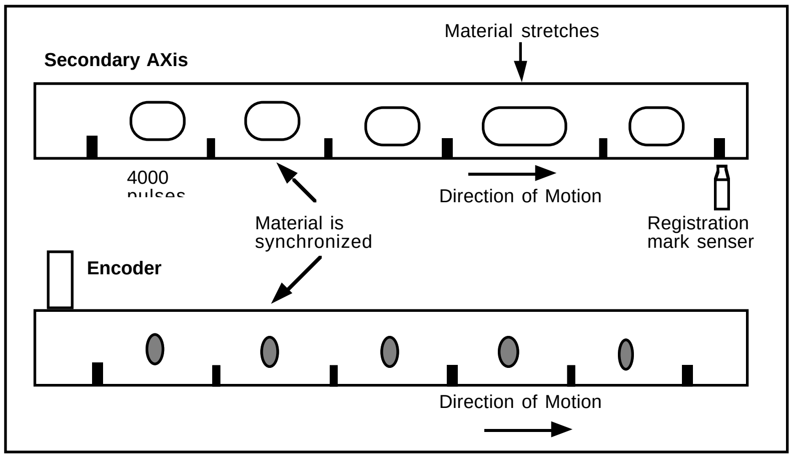


Figure 5-27. 500-FOL Synchronizes Parts (After Stretching)

You can determine and program the amount of correction that is applied to the speed ratio between each registration mark to fit your application. Use the following commands to program a synchronization application.

Command	Description
> **FC**	Expected encoder count between Registration marks
> **FBS**	Normalizing count for determining new speed ratio to move at
> **FIN**	Increment used to determine the new following speed
> **FSL**	Enables the Synchronization mode
> **FSK**	Enables the expected encoder count teach mode
> **INnI**	Defines an input as the Registration mark synchronization input.
> **TF**	Sets the sample rate of the encoder input
> **FOR**	Motor to encoder count ratio
> **FOL**	Primary to Secondary Speed ratio
> **FSI**	Enables following mode

The **FOL** and **FOR** commands determine the number of secondary motor steps that will be commanded for the encoder pulses that are received. The **FBS** and **FIN** commands determine the amount of correction that will be applied to the **FOL** and **FOR** motor to encoder ratio. **FC** is the number of encoder pulses that are expected between the registration marks if the speeds are synchronized properly. This number is compared to the actual number of pulses that are counted between each registration mark. The difference between these two values represents the error. The **FIN** and **FBS** numbers are applied to this error to determine the new speed ratio.

The **INnI** command configures an input to the 500-FOL for accepting the signal from the sensor that detects the registration marks on the parts or material. The **TF** command is the rate at which the encoder interface is sampled. The **FSL** command enables the Synchronization mode. When Synchronization mode is enabled the first time, the Synchronization input is toggled, the encoder counter is started. The next time it is toggled, the count is latched and the counter is reset to zero. The latched count is the actual number of encoder pulses counted between registration inputs. This is compared to the expected value (**FC**). The difference is multiplied by the correction factor (**FIN** and **FBS**) to determine the speed ratio to run at until the synchronization input is toggled again. This process is continuous as long as the synchronization mode (**FSL**) is enabled.

If you do not know the expected encoder count between registration marks, use the Self Learn mode (**FSK**) to determine the expected count (**FC**). To determine the expected count (**FC**) using the Self Learn mode (**FSK**) you must start the process at the speed ratio that you want to run at, turn on the Self Learn mode. The 500-FOL will count the pulses between the registration marks. When the secondary axis stops, the last recorded number will be placed in the expected count (**FC**) number. This will be used when you are in Synchronization mode.

As an example, the process in Figures 5-25 through 5-27 uses an encoder that has 4000 pulses per revolution and the secondary motor has 25000 steps per revolution. The **FOR** command is set to 6.25. The motor is mounted on the secondary conveyer belt so that one revolution is 4 inches. The encoder is mounted on the primary axis so that one encoder revolution is 4 inches. The **FOL** command must be set to 100% (**FOL1ØØ**) for the secondary axis to move at the same speed as the primary axis. If the primary axis moves at 1 rps, the secondary axis must move at 1 rps. In Figure 5-25, the two axes start moving at the same speed. The registration marks are 4 inches apart on the secondary axis. Use the Self Learn mode to determine how many encoder pulses are between the registration marks on the material on the secondary axis. The following steps show how to program the application.

Step ① Set up the Self-Learn mode.

Command	Description
> **FOR6.25**	Set the motor to encoder ratio to 6.25
> **FOL1ØØ**	Set the motor to encoder ratio speed percentage to 100%
> **FSI1**	Enable the Following mode
> **FSK1**	Enable the Self Learn mode
> **TF1**	Set the encoder sample rate to 1 ms
> **A5ØØ**	Set the acceleration of 500 rps^2
> **AD5ØØ**	Set the deceleration of 500 rps^2
> **MC**	Place the 500-FOL in the continuous mode
> **IN1I**	Defines input number 1 as a synchronization input

Step ②

The registration sensor should be wired to the synchronization input. The primary and secondary axes should now be started. After the 500-FOL has passed more than 3 registration marks, it can be stopped. The number of encoder pulses between registration marks can be checked with the **FC** command. In this example, the number that is determined is 4000 counts.

Command	Response
`> 1FC`	`*4ØØØ`

If you know the number of encoder pulses you expect to record between registration marks, this number can be entered directly for the **FC** command and will override the number determined in the self learn mode.

Command	Description
`> 1FC4ØØØ`	Manually entering the expected count

Step ③

The 500-FOL now has the number of counts expected between registration marks. The next step is to determine the correction gain desired. The correction will be applied to the difference between the expected count that was just determined and the actual counts that will be counted during actual operation. The equation for the determination of the correction is;

Equation 5-15. Correction

$$\text{Correction} = \frac{(\text{Actual encoder count} - \text{Expected encoder count}) * \mathbf{FIN}}{\mathbf{FBS}}$$

This motor-to-encoder step ratio is determined by the following equation.

Equation 5-16. Speed Ratio

$$\text{New Speed Ratio} = \mathbf{FOR} * \mathbf{FOL} + \text{Correction}$$

To determine the number of motor steps that will be commanded for the number of encoder pulses received, use the following equation.

Equation 5-17. Motor Step Correction

$$\text{Motor Steps} = \text{Encoder Steps} * (\mathbf{FOR} * \mathbf{FOL} + \text{Correction})$$

You must determine the amount of correction you want to have for a given amount of error. Once this has been determined, you can enter the **FIN** and **FOR** commands.

Command	Description
`> 1FIN3.12`	The following increment is 3.12
`> 1FBS1ØØ`	The following base number is 100

Step ④

Disable Self Learn mode and enable Synchronization mode.

Command	Description
`> 1FSKØ`	Disable Self Learn mode
`> 1FSL1`	Enable Synchronization mode

Step ⑤ Orient the primary and secondary axes to attain the desired phase relationship. Start both axes at the same time, or start the 500-FOL first and the primary axis second.

The 500-FOL will now correct any errors in the phase relationship between the two axes and maintain a synchronized speed. The new speed ratio that is determined will be applied for the entire period between registration marks. The time between the registration marks is effectively the sample period. A correction is made for each sample period.

Another method for synchronization is to use the inputs to the 500-FOL for increasing and decreasing the following speed ratio. The 500-FOL inputs can be defined to increase or decrease the following ratio. By setting one input for increasing the following ratio and one input for decreasing the following ratio, synchronization can be achieved. In this case, use an external circuit to determine whether the secondary axis should accelerate (increase ratio) or decelerate (decrease ratio) the secondary axis.

Define the input with the **IN** command. **INnX** defines the input for increasing following ratio, **INnY** defines the input for decreasing the following ratio. The following ratio will be increased or decreased while the input is active. During the 500-FOL's sample periods, the ratio will increase or decrease while the inputs are active. The inputs have a 2 ms debounce time. If the input remains active for 4 ms, the following ratio will be increased or decreased twice. The amount that the following ratio is increased or decreased is determined by the **FIN** command. If **FIN** is 1, **FOL** is increased or decreased by 1 during each sample period.

Other Following Features

This section discusses following features that the 500-FOL provides for special following requirements.

Registration in Following Mode

With the 500-FOL, registration can be performed in Following mode. It is like programming registration in the indexer version, *but the velocity term is replaced by* **FOL** *for the desired speed*. Figure 5-28 illustrates registration in the Following mode

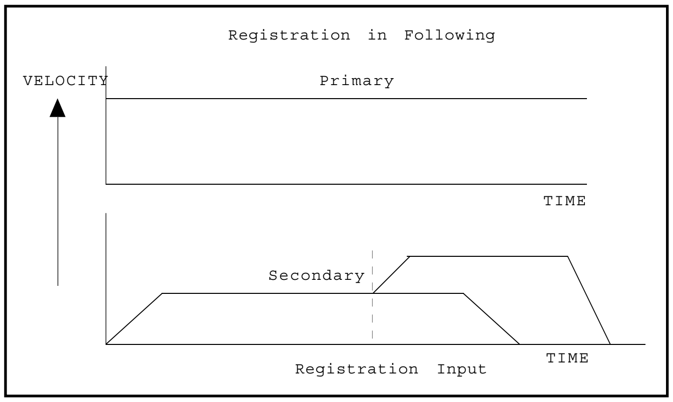


Figure 5-28. Registration in the Following Mode

Registration can only be performed if the Following Synchronized Acceleration mode is disabled (**FSFØ**). For this example, re-attach the encoder to the primary axis.

Step ① Repeat steps ① through ⑤ of the *Velocity Following Example*.

Step ② Set up input #7 as a registration input

Command	Description
`> IN7Q`	Defines input #7 as a registration input

Step ③ Define registration move #1 as follows:

```
> REG1,A1Ø,AD1Ø,FOL2ØØ,D25ØØØ
```

Step ④ Begin motion on the primary axis, then begin motion with the 500-FOL.

Command	Description
`> MC`	Change to continuous mode
`> G`	Initiate motion

Step ⑤ Toggle input #7. The motor will begin following at a speed ratio of 2:1 for 25,000 motor steps.

Jogging in Following Mode

In some applications, you may want to move the motor manually while in Following mode. This allows you to follow the primary axis at a following ratio by toggling a switch. You can configure the 500-FOL to allow you to follow the primary motor manually with the Configure Input (**IN**) command. Jogging in Following mode does not require the **JHV** or **JVL** commands. In this case, you will jog at whatever speed ratio the **FOL** command is set to. To use the inputs, you can either configure the input as a CW or a CCW jog as with the preset indexer jog. However, you cannot use the high-speed/low-speed jog input because you can only jog at the speed set by the **FOL** speed ratio.

Therefore, you use the two jog input functions: CW Jog input (**IN#J**) and CCW Jog Input (**IN#K**). You must also enable the jogging feature with the **OSE1** command. Once you set these parameters, you can attach a switch to the jog inputs (predefined) and jog the motor(s). The **#** character represents digits 1 - 13, which you enter. You must have the 500-FOL in Following mode to jog at a speed ratio of the primary. The following example shows you how to define power-up sequence #100 to jog.

Step ① Define a power-up sequence. ***Position Tracking mode must be disabled (FSPØ) to enable direction jogging.***

Command	Description
`> XE1ØØ`	Erases sequence #100
`> XD1ØØ`	Defines sequence #100
`> LD3`	Disables the limits(*not needed if you have limit switches installed*)
`> JA25`	Sets Jog Acceleration to 25 rps^2
`> JAD25`	Sets Jog Deceleration to 25 rps^2
`> OSE1`	Enables Jog function
`> JVL.5`	Sets low-speed jog velocity to 0.5 rps
`> JVH5`	Sets high-speed jog velocity to 5 rps
`> IN1J`	Sets **IN 1** as a CW jog input
`> IN2K`	Sets **IN 2** as a CCW jog input
`> FOR6.25`	Sets the motor to encoder steps
`> FOL75`	Secondary moves at 75% of the primary speed
`> FSI1`	Enables the following mode
`> XT`	Ends Sequence Definition

Step ② Reset the 500-FOL. Move the primary or primary axis.

Command	Description
`> Z`	Resets the 500-FOL

Step ③ Turn on **IN 1** to move the motor CW at 75% of the primary axis' speed (until you turn **IN 1** off).

Step ④ Turn on **IN 2** to move the motor CCW at 75% of the primary axis' speed (until you turn **IN 2** off).

Changing Following Ratio Via Front Panel Pushbuttons

The following ratio or *speed percentage* that is entered with the **FOL** command can be changed by using the front panel pushbuttons. This in conjunction with the jog mode can be very helpful in determining by experimentation what following ratio is best for your application. You can jog the motor at the following ratio and if it is not quite the ratio needed you can adjust it using the front panel. In this manner a stand alone method exists for the determination of the following ratio. Perform the following steps to adjust the following ratio via pushbuttons.

Step ① Repeat steps ①and ②in the *Jogging in Following Mode* section above.

Step ② The Model 500 is now ready for jogging in the following mode. You will now set it up for changing the front panel pushbuttons via the front panel.

Command	Description
`> SSP1`	Enables front panel following ratio entry
`> CPB1`	Enables front panel pushbuttons

The following speed percentage has the range of 0.1% to 4096.9%. The speed percentage can have the last 4 digits modified via the front panel. If we were to change the **FOL** value from 201.4 to 199.5 it would be modified as follows:

Step ③ Press and hold down the **1s** button to display the 2 least significant digits (1.4).

Step ④ Press the **UP** or the **DOWN** button while simultaneously holding down the **1s** button to change the least significant digits to 9.5.

Step ⑤ Press and hold down the **10s** button to display the next highest significant pair of digits (20).

Step ⑥ Press the **UP** or the **DOWN** button while simultaneously holding down the **10s** button to change these digits of the following percentage to 9.5.

Step ⑦ After you have changed the digits to the desired following percentage, press the **ENTER** button to change the following ratio.

Example The following example changes the following ratio from 80.0 to 121.5. This provides a method to manually determine the following ratio in your application.

Step ①

Command	Description
`> FOL8Ø`	Sets the following percentage to 80

Step ② Start the master moving and move the secondary motor at 80% of the master speed by enabling input number 1.

Step ③ Disable the input number 1. Press the **1s** button on the front panel. The digits **00** will be displayed. While pressing the **1s** button, press the **UP** button until the display reads **15**.

Step ④ Now press the **10s** button. The digits **08** will be displayed. While pressing the **10s** button, press the **UP** button until the display shows **12**.

Step ⑤ Press the **ENTER** button and repeat step ② to jog the motor at 121.5% of the master speed.

Following a Step and/or Direction Signal

The 500-FOL can follow a step (or pulse) and a direction signal rather than quadrature encoder pulses. The same incremental encoder interfaces used for quadrature following are used for the step and direction following. The **`Phase A+`** and **`Phase A-`** inputs are now the **`Step+`** and **`Step-`** inputs. The **`Phase B+`** and **`Phase B-`** inputs are now the **`Direction+`** and **`Direction-`** inputs. The only other requirement is to put the 500-FOL in the pulse and direction mode. This is accomplished with the **`FSN`** command. By typing **`FSN1`** the pulse and direction capability is enabled and the **`Phase A`** and **`Phase B`** inputs are now step and direction inputs. Once the pulse and direction capability is added, any following applications are performed exactly as if the input were quadrature signals. If your application requires pulse and direction, enable the Pulse and Direction mode and repeat the procedures in this following section.

Following Equation and Command Summary

This section provides a reference for the following equations and the 500-FOL software commands that are associated with following.

Following Command Summary

Table 5-2 lists the Following commands. For a complete explanation of these following commands, refer to the ***Model 500 Software Reference Guide***. *Set-up commands are required for any following application.*

FSA	Followed by a 1—enables instant acceleration between commanded velocities for each resulting velocity change after sampling the encoder and determining the ratio.
FSF	Followed by a 1—enables the use of following acceleration as determined by **`FAC`** and **`FEN`**.
FSI	Followed by a 1–enables following mode versus indexer mode
FSJ	Followed by a 1—enables enables the encoder port labeled **ABS/INC ENCODER** for following
FSK	Followed by a 1—enables the calculation of **`FC`** for Synchronization mode.
FSL	Followed by a 1—enables Synchronization mode
FSN	Followed by a 1—enables a step and direction signal to be followed or just a pulse train if direction is not used in conjunction with **`FSP`**.
FSP	Followed by a 1—enables position tracking.
FOR	The number of secondary motor pulses per unit of travel divided by the primary encoder pulses per the same unit of travel.
FOL	The % of the primary encoder speed that the secondary axis moves at.
FAC	The change in following % for each change in encoder pulse count of **`FEN`** during following acceleration.
FEN	The number of encoder pulses that the encoder count must change by to increment the following percentage by **`FAC.`**
FP	In Motion Profiling (**`MPP`**) mode, the execution of commands is paused for the number of following encoder steps entered in the **`FP`** command.
FPA	In Motion Profiling (**`MPP`**) mode, the execution of commands pauses until the value in the encoder counter exceeds the **`FPA`** value.
FIN	The amount by which the following % changes when changed by inputs or by the pushbuttons.
FBS	In synchronization, used with **`FIN`** to determine the amount of following percentage correction.
FC	In synchronization, the expected number of encoder counts between registration marks.
TF	The following encoder sample period
VARn=FEP	**`FEP`** is a read only variable of the actual encoder count. Set it equal to the variable to get the current value of the encoder counter.
PF	Gives you a report back of the encoder count.
PFZ	Clears the encoder counter.
SSP	Followed by a 1—enables you to modify the following percentage by the front panel pushbuttons.
INnX	An input function that lets you increase the following percentage by **`FIN`**..
INnY	An input function that lets you to decrease the following percentage by **`FIN`**.
INnI	An input function used for the registration mark sensor of the synchronization mode.

Table 5-2. Summary of Following Commands

The Following commands are categorized according to the applications they support.

Velocity Following		***Velocity and Position Following***	
FOR		FOR	FSF
FOL		FOL	FEN
FSI		FSI	FAC
FSJ		FSJ	FSA
TF		TF	FSP

Advance and Recede	
FOR	FSF
FOL	FAC
FSI	FEN
FSJ	FP
TF	FPA

Synchronization		
FOR	FSF	FBS
FOL	FAC	FSK
FSI	FEN	FSL
FSJ	FC	INnI
TF	FIN	INnX
INnY		

Special Function

FSN
SSP
FSM
PF
PFZ

Following Equation Summary

The following equations were discussed throughout this chapter. They are provided again here for reference and convenience

Velocity Following

$$\text{Secondary Motor steps} = \text{Primary encoder count} * \textbf{FOR} * \frac{\textbf{FOL}}{100}$$

Velocity and Position Following

Given V_{max} and D_{acc}:

$$\textbf{FEN} = V_{max} * \frac{\textbf{TF}}{1000} \qquad V_{max} = \frac{\text{Primary encoder counts}}{\text{second}}$$

TF = Primary Encoder Sample Period in ms

$$\textbf{FAC} = \frac{\textbf{FOL} * V_{max}}{D_{acc}} * \frac{\textbf{TF}}{1000} \qquad V_{max} = \frac{\text{Primary encoder counts}}{\text{second}}$$

TF = Primary Encoder Sample Period in units of ms
FOL = Following percentage in units of percent
D_{acc} = Distance primary axis moves during secondary axis acceleration in units of primary encoder counts

Given V_{max} and A_{max} Determine D_{acc} from $D_{acc} = \frac{V_{max}^2}{A_{max}} * \textbf{FOR} * \frac{\textbf{FOL}}{100}$

The acceleration is in units of secondary motor steps/sec^2. The maximum velocity of the primary is in primary motor steps/sec.

Figure 5-29 illustrates the motion profiles of a secondary and the primary. The different parameters are shown on the profiles and the equations to determine the parameters are given below. **AD** is used for the deceleration.

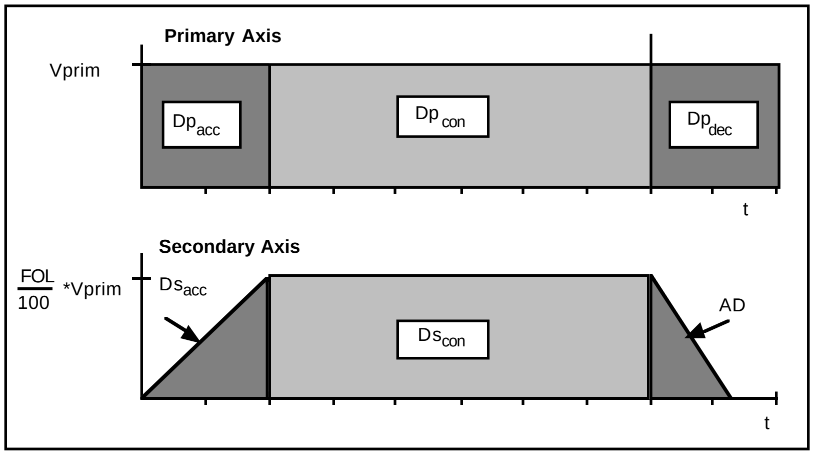


Figure 5-29. Velocity and Position Following With AD Decel

$$Ds_{acc} = \mathbf{FOR} * \left(\frac{1}{2} * \Delta\mathbf{FOL}^2 * \frac{\mathbf{FEN}}{100*\mathbf{FAC}} + \Delta\mathbf{FOL}*\mathbf{FOL}_I * \frac{\mathbf{FEN}}{100*\mathbf{FAC}} \right)$$

Δ**FOL**= The change in following percentage
$\mathbf{FOL}_I$ = The initial following percentage

The equations above can be simplified by defining a following acceleration constant determined by **FAC** and **FEN**.

$$K = \frac{\mathbf{FEN}}{100*\mathbf{FAC}}$$

The equations can now be written as:

$$D_{sec} = \mathbf{FOR} * K * \left(\frac{1}{2} * \Delta\mathbf{FOL}^2 + \Delta\mathbf{FOL}*\mathrm{FOL}_I \right)$$

$$D_{sec\ (Deceleration)} = - \mathbf{FOR} * K * \left(\frac{1}{2} * \Delta\mathbf{FOL}^2 + \Delta\mathbf{FOL}*\mathrm{FOL}_I \right)$$

Use the equations and parameters in Figure 5-30 to make a trapezoidal move (deceleration is done according to the **FAC** and **FEN** commands).

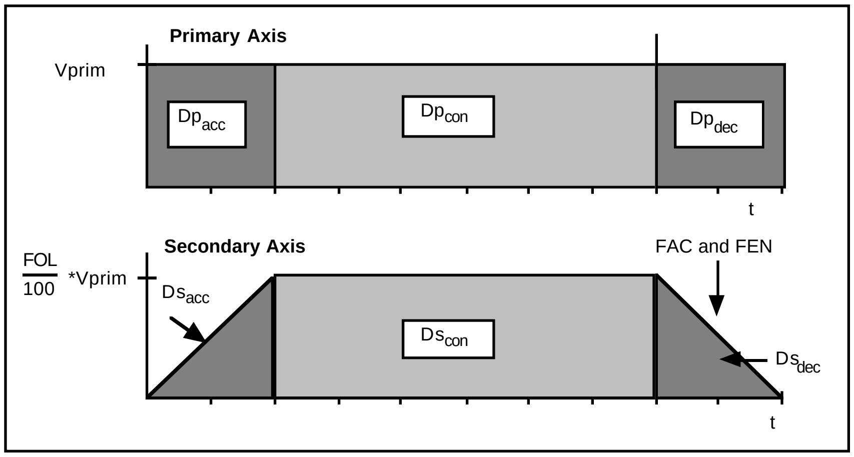


Figure 5-30. Velocity and Position Following With Following Decel

$$D_{prim} = Dp_{acc} + Dp_{con} + Dp_{dec}$$
$$D_{sec} = Ds_{acc} + Ds_{con} + Ds_{dec}$$
$$Ds_{acc} = \textbf{FOR} * K * \left(\frac{1}{2} * \Delta\textbf{FOL}^2 + \Delta\textbf{FOL}*\textbf{FOL}_I\right)$$
$$Ds_{dec} = -\textbf{FOR} * K * \left(\frac{1}{2} * \Delta\textbf{FOL}^2 + \Delta\textbf{FOL}*\textbf{FOL}_I\right)$$
$$Dp_{acc} = \Delta\textbf{FOL} * \frac{\textbf{FEN}}{\textbf{FAC}}$$
$$Ds_{con} = Dp_{con} * \textbf{FOR} * \frac{\textbf{FOL}}{100}$$

$$\textbf{FOL} = \frac{D_{prim}}{200*K} - \sqrt{\left(\frac{D_{prim}}{200*K}\right)^2 - \left(\frac{D_{sec}}{K}\right)}$$

K is the following acceleration constant, D_{prim} and D_{sec} are the distances that the primary axis and secondary axes will move, respectively. $\textbf{FOL}_I$ is the initial following percentage. In the case of a trapezoidal move, it will always be 0. The required sequence is:

Command	Description
`> XE1`	Erases Sequence #1
`> XD1`	Defines Sequence #1
`FSI1`	Enables Following mode
`FORn`	Sets motor to encoder steps per unit travel ratio
`FACn`	Sets the following acceleration increment
`FENn`	Sets the encoder period
`Dn`	Sets the secondary axis distance to n motor steps
`FOLn`	Sets the initial following percentage to n
`TR1`	Waits on the input trigger
`MPP`	Enters the Motion Profiling mode
`G`	Starts motion
`FPa`	Waits until a encoder pulses have passed
`FOLØ`	Stops the motion of the secondary
`FPb`	Waits for the decel ramp distance
`STOP`	Stops the move itself
`> XT`	Ends the Sequence #1 definition

The value for a in the first **FP** = $Dp_{acc} + Dp_{con}$
The value for b in the second **FP** = Dp_{dec}

Advance and Recede

Figure 5-31 shows an advance move profile.

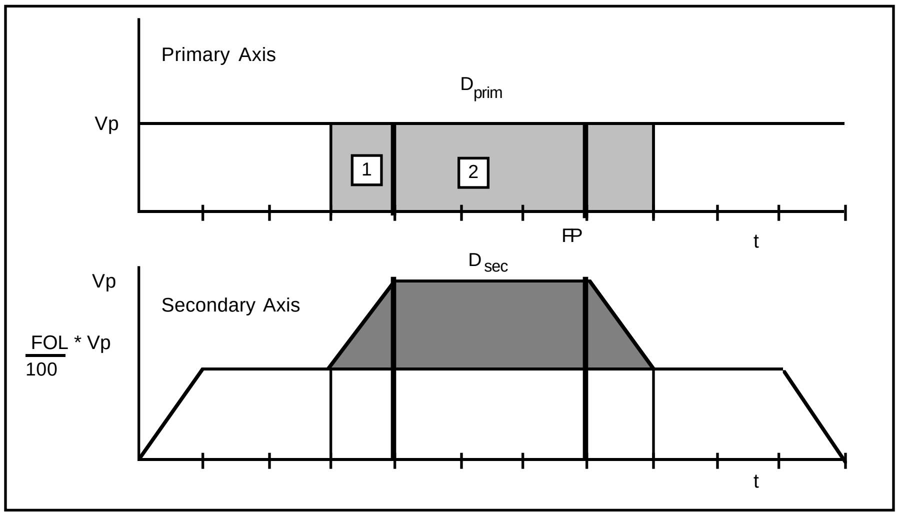


Figure 5-31. Advance Profile

In Figure 5-31, D_{prim} is the distance that the primary axis moves in encoder steps during the advance portion of the profile. It is the lightly shaded region. D_{sec} is the distance in secondary motor steps that the secondary axis advances with respect to the moving primary axis. This distance is the darkly shaded region.

$$\mathbf{FOL} = \mathbf{FOL}_I + \frac{D_{prim}}{200*K} - \sqrt{\left(\frac{D_{prim}}{200*K}\right)^2 - \left(\frac{D_{sec}}{\mathbf{FOR}*K}\right)}$$

The breakpoint to decelerate to **FOL1ØØ** is entered for the **FP** value in the advance sequence. **FP** is a distance equal to the sum of areas 1 and 2 in the primary profile.

$$\mathbf{FP} = D_{prim} - \frac{\mathbf{FEN}}{\mathbf{FAC}} * (\mathbf{FOL}_F - \mathbf{FOL}_I)$$

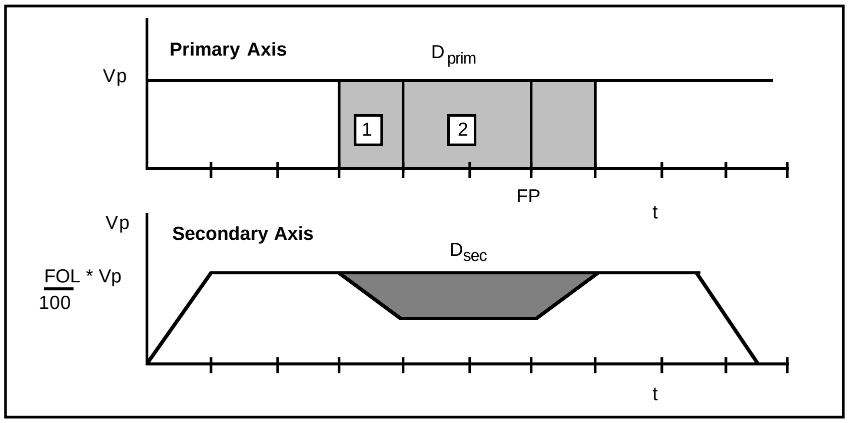


Figure 5-32. Recede Profile

The dark shaded area in Figure 5-32 is the distance that the secondary will recede with respect to the moving primary. The lightly shaded are is the distance the primary will move while the secondary recedes.

$$\mathrm{FOL} = \mathrm{FOL}_I - \frac{D_{prim}}{200*K} + \sqrt{\left(\frac{D_{prim}}{200*K}\right)^2 - \frac{D_{sec}}{\mathrm{FOR}*K}}$$

K is the following acceleration constant and is equal to: $\frac{\mathbf{FEN}}{100*\mathbf{FAC}}$

FP is equal to: $D_{prim} + \frac{\mathbf{FEN}}{\mathbf{FAC}} * (\mathbf{FOL}_F - \mathbf{FOL}_I)$

Cam Following

Figure 5-33 is a cam profile.

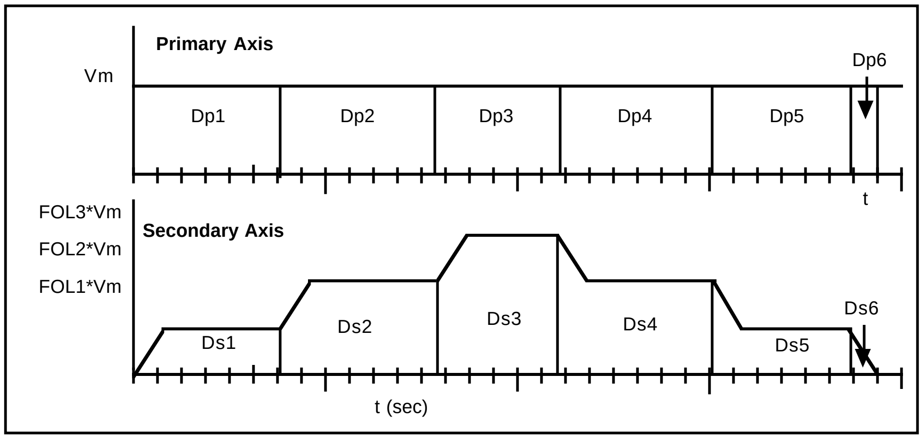


Figure 5-33. Cam Profile

Each segment of the secondary and primary move profile is marked by a primary distance Dpn and a secondary distance Dsn. The following percentage required for each segment is determined from the equations below. D_{prim} and D_{sec} are equal to Dpn and Dsn for each segment. When accelerating to a higher percentage use.

$$\mathbf{FOL} = \mathbf{FOL}_I + \frac{D_{prim}}{100*K} - \sqrt{\left(\frac{D_{prim}}{100*K}\right)^2 + \frac{2*FOL_I*D_{prim}}{100*K} - \frac{2*D_{sec}}{\mathbf{FOR}*K}}$$

When decelerating to a lower percentage, use:

$$\mathbf{FOL} = \mathbf{FOL}_I - \frac{D_{prim}}{100*K} + \sqrt{\left(\frac{D_{prim}}{100*K}\right)^2 - \frac{2*\mathbf{FOL}_I*D_{prim}}{100*K} + \frac{2*D_{sec}}{\mathbf{FOL}*K}}$$

FP or **FPA** are equal to Dpn or the accumulative Dpn respectively.

Each segment of the cam can be broken down as shown in Figure 5-34 and described the the equations below.

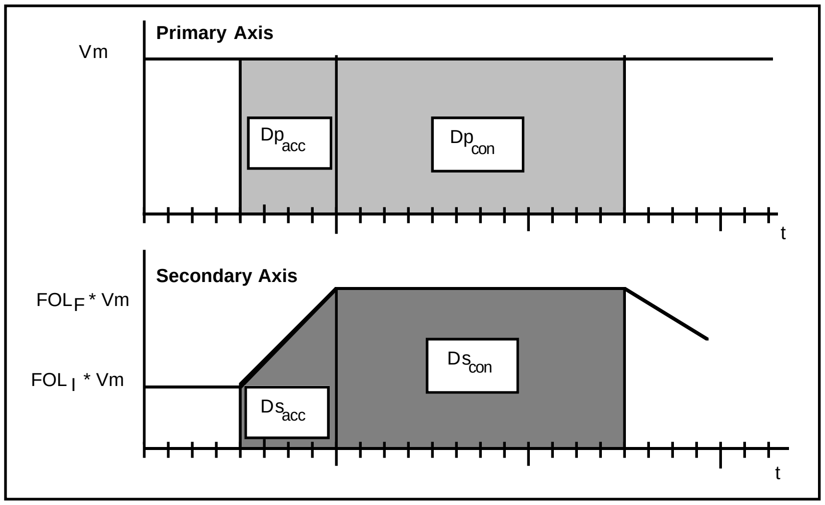


Figure 5-34. Cam Profile Segment

$$D_{prim} = Dp_{acc} + Dp_{con}$$
$$D_{sec} = Ds_{acc} + Ds_{con}$$

$$Ds_{acc} = FOR * K * \left(\frac{1}{2} * (FOL_F - FOL_I)^2 + (FOL_F - FOL_I) * FOL_I\right)$$

$$Dp_{acc} = (FOL_F - FOL_I) * \frac{FEN}{FAC}$$

$$Ds_{con} = Dp_{con} * FOR * \frac{FOL}{100}$$

The distance for **FP** is $Dp_{acc} + Dp_{con}$, for **FPA** is the absolute value of the primary distance since the start of the cam cycle.

Chapter 6. Maintenance & Troubleshooting

Chapter Objectives

The information in this chapter will enable you to:

- ❑ Maintain the system's components to ensure smooth, efficient operation
- ❑ Isolate and resolve system hardware and software problems
- ❑ Use this chapter as a quick-reference tool for a description of system error codes

Maintenance

This section describes Compumotor's recommended maintenance on the Model 500 system.

Spare Parts Table

Table 6-1 is a list of recommended spare parts for the Model 500 system.

Description	Function	Part Number
4-Pin Connector	Power Connection	43-005560-01
9-Pin Connector	Programmable Output	43-008755-01
Battery	Maintains RAM memory	47-007709-01

Table 6-1. Spare Parts Table

Battery

The non-volatile memory of the Model 500 is a Battery-Backed RAM (Random Access Memory). The battery is used to keep the RAM powered at all times. The life of this battery is approximately 10 years. When the battery runs low, the Model 500 will be unable to hold a program in nonvolatile memory. If the battery back up is not functioning properly, error **30** will be displayed when you cycle power. To verify that the battery is OK, type **1XC**, then cycle power to the Model 500. If the battery is OK, Error **30** (Check Sum error) should be gone. If the message is still flashing after you issue **1XC** and cycle power, replace the battery. Call your local distributor to order the battery.

Battery Replacement

Use the following procedure to access and replace the battery *(be sure to have the replacement battery on hand before you begin this procedure)*:

CAUTION

Use proper electro-static discharge (ESD) precautions.

① Remove the two screws in the back of the Model 500 enclosure and the four screws around the perimeter of the front panel.

② Slide the enclosure away from the front panel, exposing the PCB.

③ The battery is located on the left side of the circuit board (facing the front panel). It is located on the corner of the PCB nearest the LED Display. It is labeled **BT1** and is the size of a U.S. quarter.

④ Remove the battery from the battery holder and replace it immediately with the new battery.

⑤ Replace the enclosure and retaining screws.

Troubleshooting

This section provides methods to identify and resolve possible indexer-related hardware and software problems. You should also refer to the drive's user guide for troubleshooting procedures specific to the drive you are using.

Problem Isolation

When your system does not function properly (or as you expect it to operate), the first thing that you must do is identify and isolate the problem. When you accomplish this, you can effectively begin to eradicate and resolve the problem.

The first step is to isolate each system component and ensure that each component functions properly when it is run independently. You may have to dismantle your system and put it back together piece by piece to detect the problem. If you have additional units available, you may want to use them to replace existing components in your system to help identify the source of the problem.

Try to determine if the problem is mechanical, electrical, or software-related. *Can you repeat or recreate the problem?* Do not attempt to make quick rationalizations about problems. Random events may appear to be related, but they are not necessarily contributing factors to your problem. You must carefully investigate and decipher the events that occur before the subsequent system problem.

You may be experiencing more than one problem. You must solve one problem at a time. Log (document) all testing and problem isolation procedures. You may need to review and consult these notes later. This will also prevent you from duplicating your testing efforts.

Once you have isolated the problem, take the necessary steps to resolve it. Refer to the problem solutions contained in this chapter. If your system's problem persists, contact Parker Compumotor's Applications Department at (800) 358-9070.

WARNING

Be sure to remove power before disconnecting Model 500 system components or changing wiring.

CW, CCW & Home Switches

If you are having problems using the Trigger (`TR`), Home (`GH`), CW, CCW and Sequence Select inputs, you must first check your wiring for proper installation. Use an Ohm meter for proper connection of the switches and inputs. If the hardware connection seems correct, you must use the Input Status (`IS`) command to report the hardware status of each input and see if the Model 500 recognizes the input change. You may do this by changing the input state manually and issue the `IS` command. If the status does not change, check the hardware settings.

Remote Sequencing (BCD Inputs)

If you are trying to run sequences from BCD interfaces, the first thing you must verify is the hardware interface. Use the Ohm meter to verify proper wiring. Then use the `IS` command to read the status of the inputs. Change the input setting and check the Input Status (`IS`) again to make sure that the Model 500 recognized the change in the sequence select input. Make sure that your BCD input is calling the proper sequences. Check Chapter 4, *Application Design*, for the Sequence Select Table. If you have a problem running a sequence from the remote input, try running the sequence using the `XR` command before attempting to run it using BCD input.

Front Panel LEDs

This section describes the functionality of the front panel LEDs.

POWER

The **POWER** LED will be green if you power up the Model 500 and the power supply is functioning properly.

CPU READY

The **CPU READY** LED will be green if the microprocessor and its peripherals are functioning properly. If this LED is off, the microprocessor has faulted and the Model 500 watchdog timer has latched the unit in a reset condition. The error will cause all communications to be lost and the front panel will not be operable.

To clear the CPU ready fault you must cycle the power to the unit. If the green **CPU READY** LED again goes out, then you probably have a noise-related problem. *It is important to ensure that you have properly grounded your Model 500 to earth ground.* The noise can often be transmitted through the inputs or outputs from poorly wired or grounded I/O wiring.

To verify where the noise is coming from, disconnect one interface at a time from the Model 500 until the **CPU READY** LED no longer goes out. If you have only the power and the Motor/Driver cable attached and the problem persists, then consult the factory.

FAULT

The **FAULT** LED will be off under normal operations. If a hardware or software error exists or if the shutdown output is active, this LED will turn red. Refer to the *Diagnostic Codes* section in this chapter for a listing of related codes that will flash on the two-digit display.

Diagnostic Codes Table

The two-digit LED flashes an error code if any type of error occurs. You can also light the LEDs with the `LED` command.

The following table lists the error codes and the explanation of each error code. The symptoms and solutions table shows how you can try to correct these error conditions.

CONDITION	LED DISPLAY CODE
BLANK	NO ERRORS
16	AMPLIFIER OFF
20	EXCESSIVE POSITION ERROR
30	BBRAM CHECKSUM ERROR
41	CW LIMIT SWITCH ENGAGED
42	CCW LIMIT SWITCH ENGAGED
43	CW SOFTWARE LIMIT ENGAGED
44	CCW SOFTWARE LIMIT ENGAGED
60	COMMANDED SHUTDOWN
66	USER FAULT
71	ABSOLUTE ENCODER IS DISCONNECTED
72	BAD ABSOLUTE ENCODER READS

Table 6-2. Diagnostic Codes Table

Error Code 30

Error code 30 is caused by having the checksum that is calculated on a reset or from cycling power not match the previous checksum that was calculated by the 500. The Checksum error 30 will occur if you are changing PROMS, if you are in the middle of defining a sequence but have not issued an `XT` command yet, and if power is cycled or a reset is issued. If memory is corrupted at any time during operation or during the time the unit is powered down an error 30 will occur. If the battery fails then the memory cannot be saved and an error 30 will occur every time the power is cycled or a reset is issued.

In most cases you can simply issue the `XC` (create Checksum) command and the error 30 will be cleared. Then issue the `ON` command to clear the error 16. If the error persists and occurs each time you power up your unit, then you should consult the factory.

Error Code 16

Error 16 is caused by the drive fault line coming back to the Model 500. It indicates that the step and direction output have been disabled by the software. To clear the error simply type an `ON` command. If the error persists then toggle the drive fault enable software switch. This is done by typing `SSQ1` or `SSQØ` depending on the present state of the `SSQ` bit.

RS-232C Problems

If you are having problems communicating with the Model 500, try the following procedure to troubleshoot the communications interface.

① Power-up your computer or terminal *and then* power-up the Model 500.

② The serial port of your computer/terminal may require handshaking. If so, you must disable handshaking with your terminal emulator software package. You can also disable handshaking by connecting the computer's/terminal's RTS to CTS (usually pins 4 and 5) and DSR to DTR (usually pins 6 to 20).

③ Verify that the computer/terminal and 500 are configured to the same baud rate, number of data bits, number of stop bits, and parity.

④ Check to make sure you are using DC common or signal ground as your reference, *not* earth ground.

⑤ Cable lengths should not exceed 50 ft. unless you are using some form of line driver, optical coupler, or shield. As with any control signal, be sure to shield the cable to earth ground at one end only.

⑥ Press the space bar several times. The cursor should move one space each time you press the space bar. If your terminal displays garbled characters, check the terminal's protocol set-up. The baud rate setting probably does not match the Model 500's setting.

⑦ If the terminal does not display garbled characters, press the space bar several times. If the cursor does not move, disconnect the RS232 plug from the 500 and measure the cable's continuity . If you do not measure > -3V on Pin 1 (Rx) to Pin 3 (GND), your cable is wired incorrectly or your terminal is broken. If you measure > -3V on Pin 2 (Tx) to Pin 3 (GND), then switch the Model 500's transmit (Tx) and receive (Rx) wires and try this test again.

⑧ Once you are able to make the cursor move, enter some characters. These characters should appear on the computer or terminal display. If each character appears twice, your host is set to half-duplex. It should be set to full-duplex.

Reducing Electrical Noise

Try to eliminate sources of possible noise interference. Potential noise sources include inductive devices such as solenoids, relays, motors, and motor starters operated by a hard contact.

For more information on identifying and suppressing electrical noise, refer to the Technical Data section of the *Compumotor Programmable Motion Control Catalog.*

Software Debugging Tips

This section offers some helpful tips for debugging your programs. The Model 500 has several software tools (listed below) that to aide you in identifying a problem in the system design.

Software Tool	Command to Use
Trace Mode	XTR
I/O Simulation	DIN, DOUT
Single Step	XST
Sequence Execution status	XS
Display indexer status	DFX
Display state of the Indexer	DR
Display interface option status	FS
Display indexer status options	SS
Display Homing/Jog status	OS

Trace Mode

The trace mode is used to display what is occurring as you execute your sequence. **XTR1** enables the trace mode, **XTRØ** disables it. When enabled you can execute sequences using the **XR** command. As the sequence is running, the commands are displayed on the screen. If the program stops running, you can see what command was last executed.

The interactive mode (**SSI**) is also helpful as a way to to find commands that the indexer may not recognize.

Refer to the *Sequence Debugging Tools* section in Chapter 4 for a more thorough explanation of the trace mode.

I/O Simulation

Using the **DIN** and **DOUT** commands, you can perform I/O simulation without actually physically toggling the inputs or outputs. Use these commands to simulate the input or output state you desire so that you can test portions of your sequences and program. Refer to the *Sequence Debugging Tools* section in Chapter 4 for a more thorough explanation.

Displaying Model 500 Status

The Model 500 provides several commands which should be checked when you are experiencing difficulties while programming the Model 500. You can report back the value or setting of almost all of the Model 500 commands by typing the device address followed by the command, then a carriage return or a space bar. In this manner you can find out what values you have entered in different commands.

In addition you can use the **DR** command to report the current state of the Model 500. Use this to verify that the unit is configured the way you want it.

Four other report back commands can be used to tell you the state of the Model 500. These commands report back a binary number. Each bit of the response corresponds to different functions or modes that the Model 500 could be in. A quick-reference section is provided here to help you determine what state your Model 500 is in.

DFX Command Report Back

The **DFX** command reports the 500's current states and conditions

```
32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9   8  7  6  5   4  3  2  1  0
*0  0  0  0_0  0  0  0  0_0  0  0  0_0  0  0  0_0  0  0  0_0  0  0   0_0  0  0   0_0  0  0  0
```

Bit representations 25-32,13 are reserved

<u>**Bit**</u>	
24	Mode Profile: no = Ø; yes = 1
23	Mode alternate: no = Ø;, yes = 1
21	Hit a software CCW limit: no = Ø; yes = 1
20	Hit a software CW limit: no = Ø; yes= 1
19	Home limit: not found = Ø; found = 1
18	Jogging: no = Ø; yes = 1
17	Queued for **RM** mode: no = Ø; yes = 1
16	Run sequence on power up: no=Ø; yes=1
15	**U** command: Ø=not waiting; 1=waiting for continue
14	Waiting for a trigger: no = Ø; yes = 1
12	Back up to home limit: Ø = no; 1 = yes
11	High-speed portion of home move: no = Ø; in process = 1
10	Execute a sequence: no = Ø; yes = 1
9	Wait on a timer: no = Ø; yes = 1
8	Hit a CCW limit: no = Ø; yes = 1
7	Hit a CW limit: no = Ø; yes = 1
6	**PS** command: Ø = not waiting; 1 = waiting for continue
5	Absolute move direction: Ø = CW; 2 = CCW
4	Incremental/absolute: Ø = **MPI**; 1 = **MPA**
3	Mode: preset = Ø; Continuous = 1
2	Commanded move direction: Ø = CW; 1 = CCW
1	Preset move in progress: Ø = not moving; 1 = moving
Ø	Continuous move: Ø = not moving; 1 = moving

SS, FS, & OS Report Back Commands

The FS command reports back a binary word which has the various interface options associated with each bit. The OS command reports back the Homing options and the Jog enable option. The SS command reports back various indexer software options. A reference is provided below along with the function of each bit location. If a 1 is in the bit location the feature or mode is enabled.

```
      A B C D _ E F G H _ I J K L _ M N O P _ Q R S T
SS   *0 0 0 0 _ 0 0 0 0 _ 0 0 0 0 _ 0 0 0 0 _ 0 0 0 0
FS   *0 0 0 0 _ 0 0 0 0 _ 0 0 0 0 _ 0 0 0 0
OS   *0 0 0 0 _ 0 0 0 0
```

Command	Function
SSA	RS232 Echo: 0 = Echo on 1 = Echo off
SSD	Alternate Mode Stop: 0 = end of cycle 1 = immediately
SSG	Clear/Save buffer on limit: 0 = clear 1 = Save
SSH	Clear/Save buffer on stop: 0 = clear 1 = Save
SSI	Enable/Disable Interactive Mode
SSJ	Enable/Disable Continuous scan mode
SSL	Resume execution enable
SSN	Enable error message mode
SSO	Enable Sequence Select via front panel
SSP	Enable ratio select via front panel
SSQ	Enable Drive Fault indicator

Command	Function
FSB	Enable Indexer to Motor/Encoder step Mode 1=Encoder
FSC	Enable/Disable Position Maintenance
FSD	Enable/Disable Stop on Stall
FSI	Enable/Disable Following
FSJ	Select Encoder input Function: Ø = **ABS/INC** for position **INC** for following; 1 = **ABS/INC** for following **INC** for position
FSK	Enable following learn mode
FSL	Enable following self correction mode
FSM	Select Absolute or Incremental encoder 1=Absolute
FSN	Enable Pulse and Direction following
FSO	Enable "lights out" or system recovery from power failure
FSP	Enable position tracking

Command	Function
OSB	Enable back-up to home switch
OSC	Define Active edge of Home switch 1=Active high signal
OSD	Enable Encoder Z channel Input for Homing
OSE	Enable Jogging
OSG	Define Final Home approach direction 1=CCW
OSH	Define Active edge of home switch to stop on 1=CCW

Common Programming Mistakes

This section describes common mistakes made while using the X language.

An indexer move is commanded and no motion occurs: The following mode may be enabled when you are trying to make a move as an Indexer, a limit may be enabled and active, or you may be in the absolute mode and are already at the position you are commanding the motor to move to.

A following move is commanded and no motion occurs: The unit may not be in the following mode (FSI), the wrong encoder interface has been selected (FSJ), limits are enabled, or you may be in the absolute mode and are already at the position you are commanding the motor to move to.

If motion is jittery in the following mode you can smooth it out by decreasing the sampling of the encoder interface (increase **TF**).

If a sequence that was entered, the backspace (**^H**) command may cause misinterpretation of a Quote command.

The unit may appear to not be responding to commands. If you were defining a sequence and never issued an **XT** command, then the Model 500 still thinks you are defining a sequence.

Returning the System

If you must return your Model 500 system to affect repairs or upgrades, use the following steps:

① Get the serial number and the model number of the defective unit, and a purchase order number to cover repair costs in the event the unit is determined by the manufacturers to be out of warranty.

② Before you return the unit, have someone from your organization with a technical understanding of the Model 500 system and its application include answers to the following questions:

- What is the extent of the failure/reason for return?
- How long did it operate?
- Did any other items fail at the same time?
- What was happening when the unit failed (i.e., installing the unit, cycling power, starting other equipment, etc)?
- How was the product configured (in detail)?
- What, if any, cables were modified and how?
- With what equipment is the unit interfaced?
- What was the application?
- What was the system environment (temperature, enclosure, spacing, unit orientation, contaminants, etc.)?
- What upgrades, if any, are required (hardware, software, user guide)?

③ In the USA, call Parker Compumotor for a Return Material Authorization (RMA) number. Returned products cannot be accepted without an RMA number. The phone number for Parker Compumotor Applications Department is (800) 358-9070.

Ship the unit to: Parker Hannifin Corporation
Compumotor Division
5500 Business Park Drive, Suite D
Rohnert Park, CA 94928
Attn: RMA # xxxxxxx

④ In the UK, call Parker Digiplan for a GRA (Goods Returned Authorization) number. Returned products cannot be accepted without a GRA number. The phone number for Parker Digiplan Repair Department is 0202-690911. The phone number for Parker Digiplan Service/Applications Department is 0202-699000.

Ship the unit to: Parker Digiplan Ltd.,
21, Balena Close,
Poole,
Dorset,
England.
BH17 7DX

⑤ Elsewhere: Contact the distributor who supplied the equipment.