ESP7000

UNIVERSAL MOTION CONTROLLER/DRIVER

User's Manual

Warranty

Newport Corporation warrants this product to be free from defects in material and workmanship for a period of one year from the date of shipment. If found to be defective during the warranty period, the product will either be repaired or replaced at Newport's option.

To exercise this warranty, write or call your local Newport office or representative, or contact Newport headquarters in Irvine, California. You will be given prompt assistance and return instructions. Send the instrument, transportation prepaid, to the indicated service facility. Repairs will be made and the instrument returned transportation prepaid. Repaired products are warranted for the balance of the original warranty period or at least 90 days.

Limitation of Warranty

This warranty does not apply to defects resulting from modification or misuse of any product or part. This warranty also does not apply to fuses, batteries, or damage from battery leakage.

This warranty is in lieu of all warranties, expressed or implied, including any implied warranty of merchantability or fitness for a particular use. Newport Corporation shall not be liable for any indirect, special, or consequential damages.

First Printing February 2000

This manual has been provided for information only and product specifications are subject to change without notice. Any change will be reflected in future printings.

1791 Deere Ave.
Irvine, CA 92606
(949) 863-3144

P/N 28314-01, Rev. E, IN-04993 (04/02)

Newport®

EU DECLARATION OF CONFORMITY

We declare that the accompanying product, identified with the CE mark, complies with requirements of the Electromagnetic Compatibility Directive, 89/336/EEC and Low Voltage Directive 73/23/EEC.

Model Number: ESP7000

Year CE mark affixed: 2000

Type of Equipment:
Electrical equipment for measurement, control and laboratory use

Standards Applied:
Compliance was demonstrated to the following standards to the extent applicable:

BS EN61326: 1998 "Electrical equipment for measurement, control and laboratory use – EMC requirements"

This equipment meets the CISPR 11 Class A radiated and conducted emission limits.

BS EN 61000-3-2, Harmonic current emissions, Class A

BS EN 61000-3-3, Voltage fluctuations and flicker

BS EN 61010-1 "Safety requirements for electrical equipment for measurement, control and laboratory use"

Alain Danielo
VP European Operations
Zone Industrielle
45340 Beaune-la-Rolande, France

Bob LaTouche
VP of IMS
1791 Deere Avenue
Irvine, CA USA

BF-05991 12/99

Table of Contents

Please see Command Index List on page v

Section 4 – Advanced Capabilities 4-1

List of Figures

List of Tables

Command Index (Section 3)

Section 1 - Introduction

1.1 Scope

This manual provides descriptions and operating procedures for the Enhanced System Performance (ESP), ESP7000 Stand Alone Motion Controller/Driver.

Section 1, Introduction contains:

- Safety Considerations
- Conventions and Definitions
- System Overview
- Procedures for hardware and software requirements
- Descriptions of controls and indicators
- Setup procedures
- Instructions for configuring, powering up the ESP7000 and stage motors for home and jog motions
- System shutdown procedures.

Section 2, Modes of Operation contains:

- Overview of operating modes, Local and Remote
- Operating options in LOCAL Mode.

Section 3, Remote Mode contains:

- Features and operation of the Windows software utilities
- Newport-provided commands, language-specific information and error-handling procedures.

Section 4, Advanced Capabilities contains:

- Grouping
- Contouring
- Data Acquisition
- Master-Slaving

Section 5, Motion Control Tutorial contains:

- Overview of motion parameters and equipment.

Section 6, Servo Tuning contains tuning principles and procedures:

Section 7, Optional Equipment contains:

- Hand-held keypad description and activation
- Rack Mount Brackets description
- Motor Driver Card description.

The following information is provided in the Appendices:

- Error messages
- Trouble-shooting and maintenance
- Connector pin assignments
- Decimal/ASCII/binary conversion table
- System upgrades for software and firmware
- ESP Configuration Logic
- Programming Non-ESP Compatible Stages
- Factory service.

1.2 Safety Considerations

The following general safety precautions must be observed during all phases of operation of this equipment. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the equipment.

Disconnect or do not plug in the AC power cord in the following circumstances:

- If the AC power cord or any other attached cables are frayed or damaged
- If the power plug or receptacle is damaged
- If the unit is exposed to rain or excessive moisture, or liquids are spilled on it
- If the unit has been dropped or the case is damaged
- If the user suspects service or repair is required
- When the user cleans the case.

To protect the equipment from damage and avoid hazardous situations, follow these recommendations:

- Do not open the ESP7000 Stand Alone Motion Controller. There are no user-serviceable parts inside the ESP7000.
- Do not make modifications or parts substitutions.
- Return equipment to Newport Corporation for service and repair.
- Do not touch, directly or with other objects, live circuits inside the unit.
- Keep air vents free of dirt and dust.
- Do not block air vents.

- Keep liquids away from unit.
- Do not expose equipment to excessive moisture (>85% humidity).

WARNING

All power receptacles to which this unit is connected, are to be of the grounding type and properly polarized. Contact an electrician to check faulty or questionable receptacles.

WARNING

This product is equipped with a 3-wire grounding type plug. Any interruption of the grounding connection can create an electric shock hazard. If the user is unable to insert the plug into their wall plug receptacle, contact an electrician to perform the necessary alterations to assure a proper ground connection.

WARNING

This product operates with voltages that can be lethal. Pushing objects of any kind into cabinet slots or holes, or spilling any liquid on the product, may touch hazardous voltage points or short out parts.

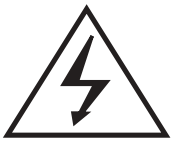
1.3 Conventions and Symbols

This section provides a list of symbols and their definitions and commonly used terms found in this manual.

1.3.1 Safety and General Symbols/Definitions

The following are definitions of safety and general symbols used on equipment or in this manual.

WARNING



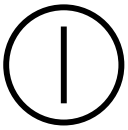
Calls attention to a procedure, practice or condition which, if not correctly performed or adhered to, could result in injury or death. Caution, risk of electric shock.

CAUTION

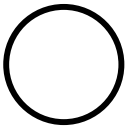
Calls attention to a procedure, practice, or condition which, if not correctly performed or adhered to, could result in damage to equipment. Caution (Refer to accompanying documents)

NOTE

Note. Calls attention to a procedure, practice, or condition that is considered important to remember in the context.

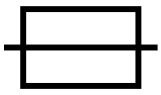


This symbol indicates the principal On/Off push-push switch is in the **ON** position.

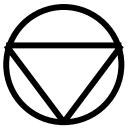


This symbol indicates the principal On/Off switch is in the **OFF** position.

Protective ground terminal



Fuse



Stop (of action or operation)

1.3.2 Terminology

The following is a brief description of the terms specific to motion control and the ESP7000 Universal Motion Controller/Driver.

Axis – A logical name for a stage/positioner/motion device

Encoder – A displacement measuring device, term usually used for both linear and rotary models

DSP – Digital Signal Processor

ESP – Enhanced System Performance motion system. ESP is synonymous with a plug-and-play motion system.

ESP – compatible – Refers to Newport Corporation stages that contain the necessary parameters to automatically configure an ESP type motion controller. Newport stages or other stages without this feature must be individually configured by the user.

Home (position) – The unique point in space that can be accurately found by an axis, also called origin.

Home search – A specific motion routine used to determine the home position.

Jog – A motion of undetermined-length, initiated manually.

Local Mode – The user has access to a sub-set of motion control functions through the use of Front Panel buttons and display menus of the ESP7000.

Motion device – Electro-mechanical motion device. Used interchangeably with stage and positioner.

Move – A motion to a destination, initiated manually or remotely.

Origin – Used interchangeably with home.

PID – A closed loop algorithm using proportional, integral, and derivative gain factors for position control.

Positioner – Used interchangeably with stage and motion device equipment.

Remote Mode – The ESP7000 motion controller receives motion commands through one of its interfaces (IEEE-488, RS-232C, or USB) using a computer or terminal.

Stage – Used interchangeably with motion device and positioner.

1.4 System Overview

The ESP7000 is an integrated controller and driver in one chassis that simplifies system hook-up and provides improved reliability.

The ESP7000 can drive and control up to six axes of motion using any combination of DC and/or stepper motors. Each driver module will drive 2 or 4 phase stepper and brush DC servo motors at 3A (max.) per axis. This capability will allow the user to drive any of Newport's wide selection of standard stages. The controller uses a 32-bit, floating point, 60 MHz DSP processor for high precision synchronized control, and supports sophisticated trajectories like: synchronized circular/linear interpolation and continuous path contouring for complex motion profiling. The controller can also execute trapezoidal and S-curve velocity profile for smooth, jerk free positioning.

The previously defined ESP architecture consists of the DSP based controller, motor driver, and ESP-compatible stages.

The system is designed to operate with Newport Corporation's ESP-compatible stages, but can be configured to function with other stages.

1.4.1 Features

- 24 user programmable digital I/O Opto22™ compatible – for internal/external event synchronization
- 8-channel, 16-bit A-to-D input – for high resolution analog data acquisition
- Differential or single-ended encoder inputs – for feedback signal integrity
- "Watchdog" timer and remote interlock – for added system security/safety
- 64kB Flash non-volatile user program memory – for storing up to 100 user defined programs
- 512kB Flash non-volatile firmware memory – for easy field upgrades
- Field Programmable Gate Array (FPGA) – for flexible hardware reconfiguration and upgrades
- RS232 communications link – for easy user interfacing
- IEEE-488 (GPIB) interface – for high-speed parallel communication
- USB interface – for high-speed serial communication.

1.4.2 Specifications

Universal Motion Controller/Driver:

- Integrated controller and driver in one chassis – for simplified system hook-up and improved reliability
- 32-bit, floating point, 60MHz DSP processor – for high precision synchronized control
- Universal 6-axis stepper/DC servo control in any combination – for single unit "XYZ θX θY θZ" solution.

Supported Trajectories:

- Trapezoidal and S-curve velocity profile
- Non-synchronized, point-to-point
- Synchronized Circular/Linear Interpolation
- Continuous path contouring – for complex motion profiling.

Motor Control:

- Digital PID-FF (feed-forward) servo loop – for precise velocity profile tracking and accurate positioning
- 18-bit DAC command output – improves stability for high precision applications
- 16 MHz (max.) encoder input frequency – for high-speed, high-resolution applications.

Motion Drive Capability:

- ESP Stage compatible – for 'plug and play' capability
- Easily configurable to drive non-ESP stages
- 2 or 4-phase stepper/Brush DC servo at 3A (max.) per axis – drive large selection of stages and actuators
- 250x (max.) programmable micro-step resolution – for ultra-smooth stepper positioning.

Options:

- Front Panel with large graphical LCD display, pushbuttons, and 3.5" floppy drive – for easy jog control and axis configuration
- Hand-held keypad – for local, non-computer programming
- 19" Rack-mount brackets
- Blank Front Panel

Physical Specifications:

- Six(6), universal motor driver cards – for any combination of stepper and DC servo motors
- 500W power supply
- Fuse Type: T10A/250V

- 48V Pulse Width Modulated (PWM) output
- Line voltage: 115 – 230 VAC, 50/60 Hz Auto-Range
- Operating Temperature: 5°C to 40°C
- Humidity: 20% to 85% Relative Humidity
- Weight: 26 lbs. (10.8 Kg) max.
- Dimensions (4U): 16.9" wide x 17" deep x (6.5" tall + 0.5" bottom clearance).

Figure 1.1: ESP7000 Motion Controller/Driver

1.4.3 Description of Front Panel Versions

The ESP7000 is available with either a blank front panel or a front panel with an LCD display and pushbuttons. When operated via the front panel with display, a menu system allows the user to set up the controller and perform simple motion sequences without a computer interface.

BLANK FRONT PANEL

This version does not provide local operation via a display (See **Figure 1.2**). Use of the Hand-held Keypad is the only local operation.

Power Section

This section is equipped with:

- Power Button (BLACK)
- Stop All Button (RED)

The black push button type switch on the left corner is used to turn power **ON** or **OFF.** The on state is indicated with a ***green*** LED above the push-button.

The Blank Front Panel also has:

- Keypad Connector Receptacle
- An LED for each of the six axes to tell the user axis status.

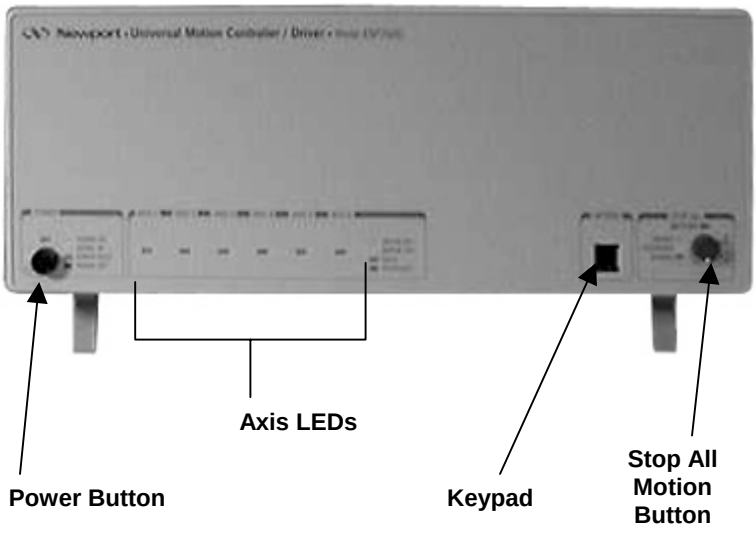


Figure 1.2: Blank Front Panel

FRONT PANEL WITH DISPLAY

A general view of the front panel is shown in **Figure 1.3.** The areas on the panel include:

- 3.5" Floppy Disk Drive
- Display LCD with menu screen capability
- Axis Selection Buttons
- Control Knob
- Stop All Button (Emergency Stop)
- Keyboard Connector
- Direction Function Buttons
- Numeric Keypad
- Menu Button
- Menu Selection Buttons
- Power Button

Front Panel Detail Descriptions

3.5" Floppy Disk Drive

The 3.5" Floppy Disk Drive is used for installation and to store motion programs, firmware upgrades.

Display LCD

The front panel display is a backlit LCD used for a variety of controller functions. The title of the current menu screen is displayed along the top portion of the LCD. Depending on the menu, submenu choices or controller functions are displayed along the bottom, and will correlate with a Menu Selection Button.

Axis dependent selections will typically be displayed along the right hand side of the LCD. These will correlate with the Axis Selection Buttons. The rest of the LCD is used for display of data.

Located underneath the display, the user will find the five Menu Selection Buttons. These buttons are used to navigate through the menu system and, at times, select certain function criteria (See **Figure 1.3**).

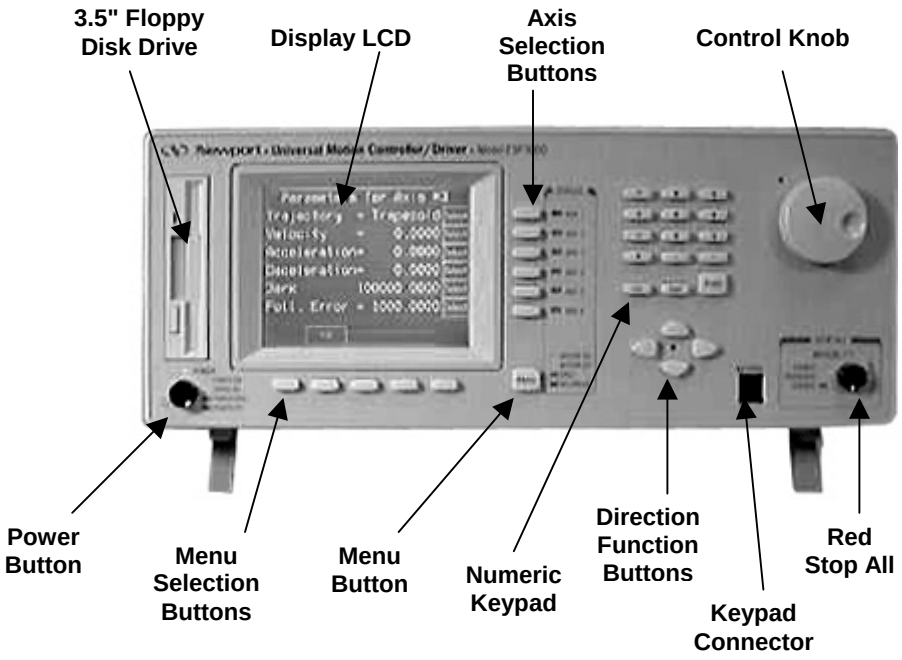


Figure 1.3: ESP7000 Front Panel with displays

Figure 1.4: Menu Item Display

Axis Selection Buttons

The ***six*** Axis Selection push buttons allow the user to select an axis in any chosen menu. Each button has a Status LED, which will illuminate ***yellow,*** indicating a configured axis that is currently disabled.

Control Knob

This knob allows the user to control a specific function in the current menu display (e.g., Jog, Home, etc.).

Red Stop All Button

For safety reasons, motor power can be controlled separately. This is done from the front panel with the STOP ALL button. For easier identification, the STOP ALL is the only one with a RED push-button. This RED button allows the user to Disable the Motor.

A ***green*** LED is displayed above the button, in the ENABLE mode. When the user presses this RED button, a ***red*** LED will display above the button, indicating that the motor is DISABLED.

A ***yellow*** LED is displayed above the button when the system is in the INTERLOCK mode.

NOTE

Activating this switch aborts all motion and disables Motor Power. All axes are affected.

Keypad Connector

This receptacle allows the user to plug-in the *optional* Hand Held alphanumeric Keypad. This Hand Held Keypad allows the user to access the full command set of the ESP7000 without use of a host terminal (e.g., computer). (See Section 7.1 of this manual). **Figure 1.3** shows the Keypad Connector.

Direction Function Buttons

The direction function keys have two predominant uses:

1. As an alternative to the numeric keypad for entering data values:
2. As a means of manual control for a selected positioner.

When used as a means to manually control a positioner, the specific button (either up/down OR left/right) will be selected in the **JOG** Menu (See paragraph **Menu Section**).

When used as an alternative to the numeric keypad, the up/down buttons will increment/decrement the value selected with the right/left buttons (See **Figure 1.5**).

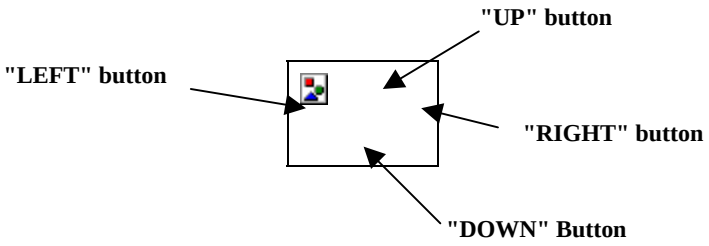


Figure 1.5: Direction Function Buttons

Numeric Keypad

The Numeric Keypad on the Front Panel allows the user to enter numeric values onto the display screen. See **Figure 1.6.**

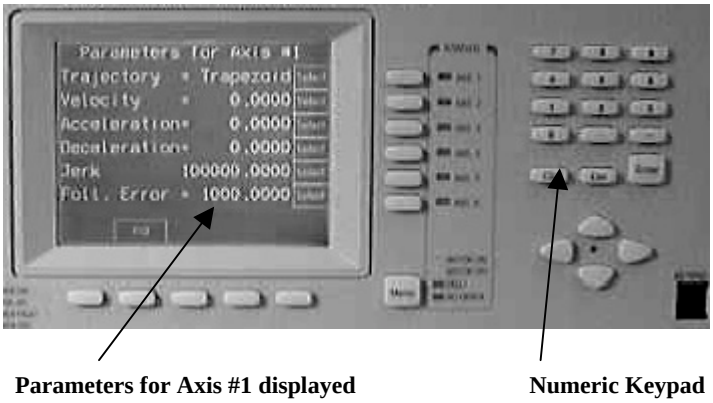


Figure 1.6: Numeric Keypad to enter specific Parameters

Menu Button

When the user presses the Menu button on the Front Panel, it allows the user to go from a sub-menu screen up one level in the menu hierarchy.

Menu Selection Buttons

These *five* Menu Selection push buttons below the display window, allows the user to navigate through the menu system. See paragraph called **Menu Section** and the **Menu Matrix.**

Power Button

The BLACK Power button has an associated ***four*** state LED's: Power On (green), Standby (yellow), Power Fault (red), and Power Off (gray).

POWER ON indicates the main power switch is ON, and the system is active.

STAND BY indicates the main power switch is on and the system is INACTIVE.

POWER FAULT indicates a malfunction of the power supply.

POWER OFF indicates the ESP7000 main power switch is OFF.

MENU SECTION

The ESP7000 features ***five*** push buttons below the display window on the front panel. This menu access allows the user to setup and use the motion system without a computer interface.

When the black POWER button is pressed, the display shows the ***ESP7000 Product Screen*** then automatically displays the ***Top Menu*** which contains ***five*** selections: **MORE, MOTION CONFIGURATION, MOVE, PROGRAM** and **MOTOR POWER.** Additionally, it displays the type of stage and current position on each axis.

MORE Menu

The user should first select the **MORE** key from the **TOP MENU** screen. The **MORE** screen will give the user ***two*** choices: **SETUP** and **ERRORS** (See **Figure 1.7, Menu Matrix**).

Error

This function allows the user to view the **ERROR LIST** and clear the ERROR notation from the display (See Appendix A for listing of Error Messages).

When an error is encountered, a flashing "Error" in the lower right hand corner of the display will illuminate. Errors are buffered internally and held in memory until they are read.

To view the **ERROR LIST** from the "Top Menu", press the Menu Selection buttons to display the **MORE** screen. Select **ERRORS,** and an **ERROR LIST** will display.

An example would be:

Error #4: EMERGENCY STOP ACTIVATED

End of List

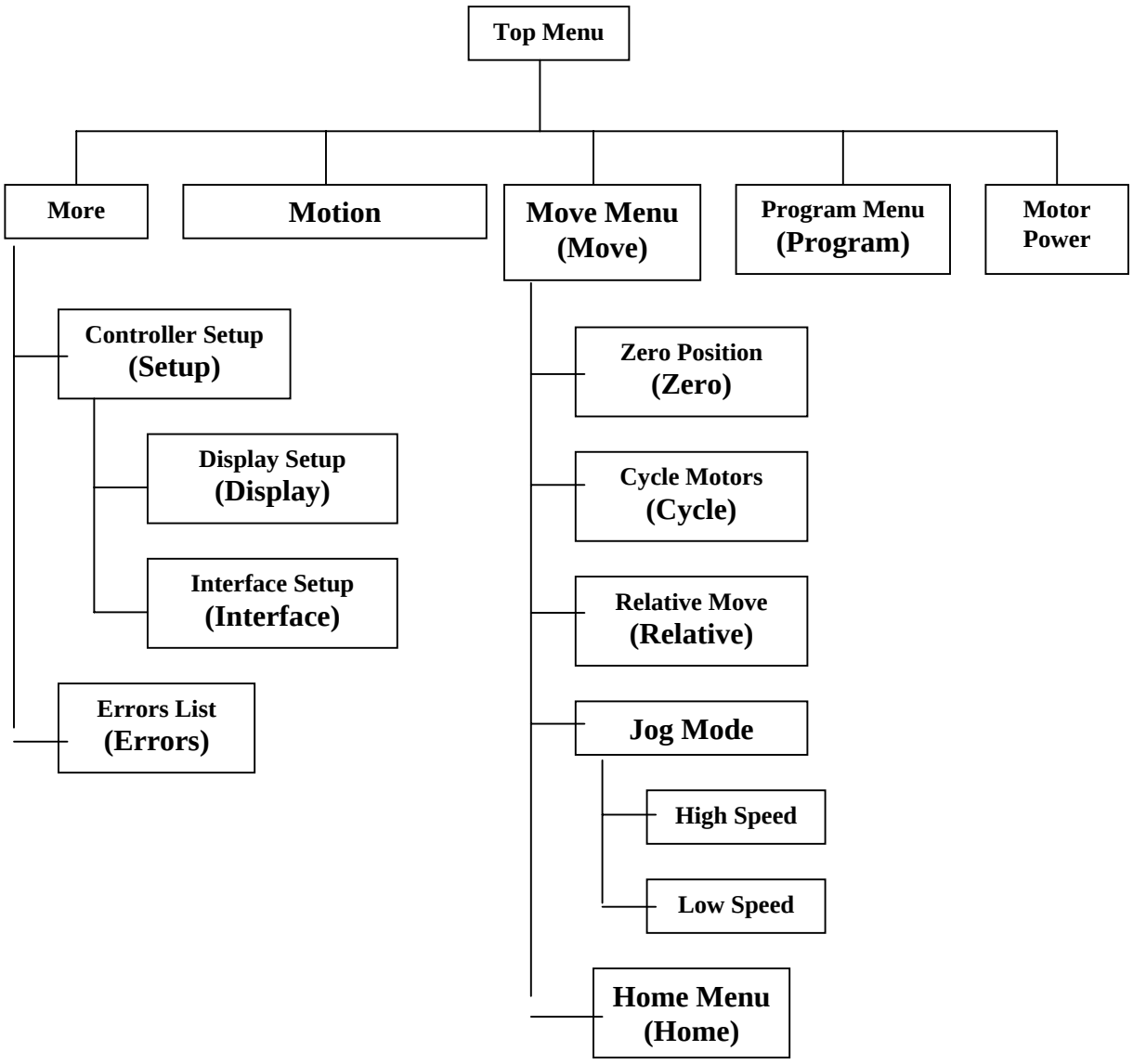


Figure 1.7: Menu Matrix

Error (Continued):

To exit **ERROR LIST** screen and clear Errors from the display, press **MENU** button, and the user returns to the **MORE** screen.

When the user presses **ERROR** button, an **ERROR LIST** screen displays, if there are No Errors, the screen will display NO ERRORS. Press the **MENU** button to exit to return to the **MORE** screen.

NOTE: The Error notation is gone from the **TOP MENU.**

Setup

When the user presses the **SETUP** button, a **CONTROLLER SETUP** screen displays, and indicates ***two*** choices: **DISPLAY** or **INTERFACE.**

1. When **DISPLAY** is selected, the **DISPLAY SETUP** screen is shown. There are ***two*** choices for Display Setup: **BackLight** or **Contrast.** The user can adjust these parameters by using the "Direction Function Buttons". The user should press **MENU** button to return to the **CONTROLLER SETUP** screen.
2. When **INTERFACE** is selected, the **INTERFACE SETUP** screen is displayed, and indicates ***two*** choices: **Serial Communication** or **IEEE488 Communication.**
3. When **SERIAL COMMUNICATION** is selected, the **RS-232 COMMUNICATION** screen is displayed. The user can select and modify BAUD RATE.

   When BAUD RATE is selected  +  arrows are displayed, so the user can increase or decrease the amount displayed, using the Direction Function Buttons.
4. When **IEEE488 COMMUNICATION** is selected, the **IEEE488 COMMUNICATION** screen is displayed. The user can select and modify: ADDRESS.

   Note: This follows the same process as RS-232 COMMUNICATION shown above.

NOTE: The user can press the MENU button to EXIT these Setup screens, after data has been revised.

MOTION CONFIGURATION MENU

When **MOTION CONFIGURATION** is selected from the **TOP MENU** screen, the **MOTION CONFIGURATION** screen is displayed. From this screen the user can change Parameters in the AXIS they select (Axis #1 through Axis #6). (See example in **Figure 1.8**).

Trajectory should be selected with the up/down Direction Function Buttons. Numeric values should be entered with the numeric keyboard.

Note: The ENTER button should be used to send the entered value into memory.

The user can also revise the PID values, by selecting **PID** from the PARAMETERS FOR AXIS #1 screen. The **PID** Values for AXIS #1 are shown in **Figure 1.9.**

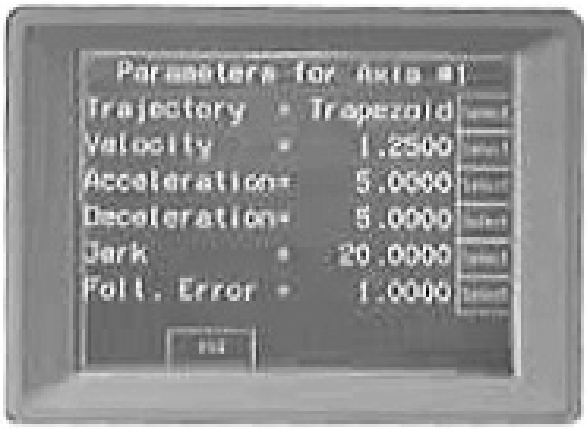


Figure 1.8: Parameters for Axis #1

From this screen the user can select new values for: **Kp, Kd, Ki, IL, Vff,** and **Aff,** by pressing the appropriate Axis Selection Buttons, entering the new value on the numeric keypad and pressing ENTER. (See example in **Figure 1.9**).

NOTE: The user can press the MENU button ***twice*** to EXIT the PID Values screen and the Parameters screen, and return to the **TOP MENU** screen.

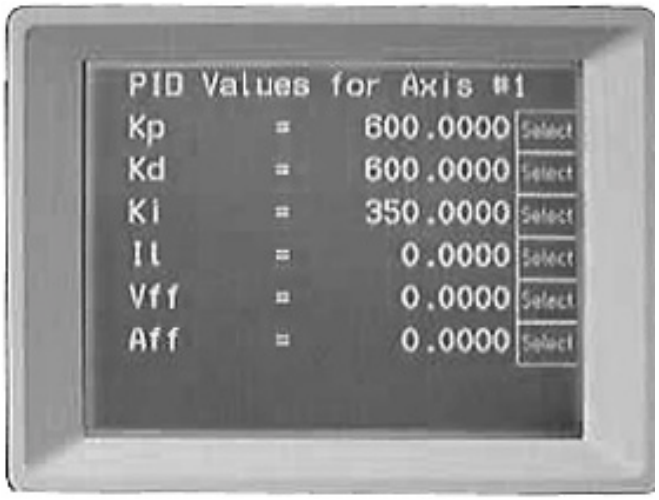


Figure 1.9: PID Values for Axis #1

MOVE Menu

When the user selects the **MOVE** button from the **TOP MENU** screen, the **MOVE MENU** displays, and gives the user ***five*** choices: **ZERO, CYCLE, RELATIVE, JOG,** and **HOME.**

1. When the user presses the **ZERO** button from the **MOVE MENU,** a **ZERO POSITION** screen displays. The user can zero an axis by pressing the related Axis Selection button for the axis desired.

   NOTE: The user can press **MENU** button to return to the **MOVE MENU.**

2. When the user presses the **CYCLE** button from the **MOVE MENU** screen, a **CYCLE MOTORS** screen displays. The user presses the axis button and types the required parameters from the numeric keypad on the front panel, and presses the <ENTER> key. NOTE: The parameters are displayed in the display window as they are typed. The ESP7000 changes the parameters on the **CYCLE MOTORS** screen after the Enter key is pressed (See example in **Figure 1.10**).

   NOTE: The screen indicates a MOVE selection. When the user selects **MOVE,** a **CYCLING** screen displays, with a **STOP ALL** selection. If **STOP ALL** is selected, the screen automatically returns to the **TOP MENU** screen.

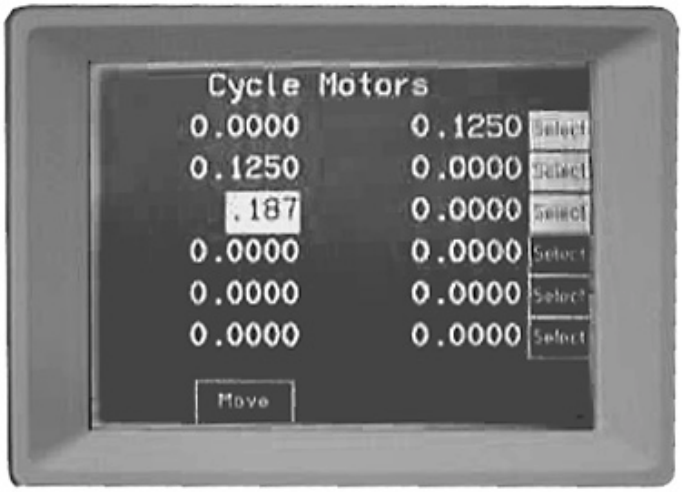


Figure 1.10: Cycle Motors Screen

3. When the user presses the **RELATIVE** button from the **MOVE MENU** screen, a **RELATIVE MOVE** screen displays. The user presses the axis button and enters the required parameters from the numeric keypad on the front panel, and presses the <ENTER> key. NOTE: The parameters are displayed in the display window as they are typed. The ESP7000 changes the parameters on the **RELATIVE MOVE** screen after the Enter key is pressed (See **Figure 1.11**).

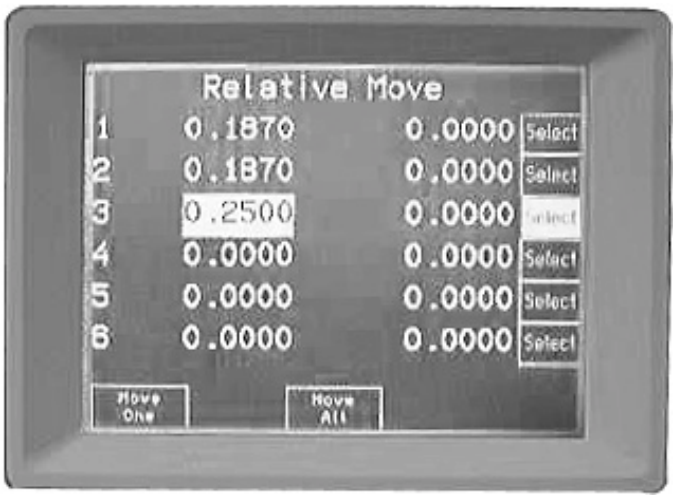


Figure 1.11: Relative Move Screen

NOTE: The screen indicates MOVE ONE and MOVE ALL selections. When the user selects **MOVE ONE** or **MOVE ALL,** the ***last*** Axis selected indicates the word **STOP** on the right side of the screen. The screen will automatically return to the **MOVE MENU.**

To STOP the movement before it completes, the user should press the Axis Selection Button next to STOP.

4. When the user selects the **JOG** button from the **MOVE MENU** screen, a **JOG MODE** screen displays using the Axis Selection Buttons. The user selects an axis and chooses the method of control from either the Direction Function Buttons or the Control Knob, depending on the number of times the Axis Selection Button is pressed. The appropriate icon will display the selected control method (e.g., left/right Direction Function Button, up/down Direction Function Button, or Control Knob) and its relative position direction (noted with an "+" sign (See **Figure 1.12**).

   NOTE: The screen indicates HIGH-SPEED and LOW SPEED selections. When the user selects **HIGH SPEED,** the **HIGH-SPEED** screen displays for ALL six Axes. The user can enter the High-Speed velocity values from the numeric keypad on the front panel, them press the <ENTER> key. The parameters are displayed in the display window as they are typed. The ESP7000 changes the parameter on the **HIGH-SPEED** screen. The user should press the **MENU** button to exit and returns the user to the **JOG MODE** screen.

   The **LOW SPEED** selection follows the same process as the HIGH SPEED. The user should press the **MENU** button to exit and returns the user to the **JOG MODE** screen.

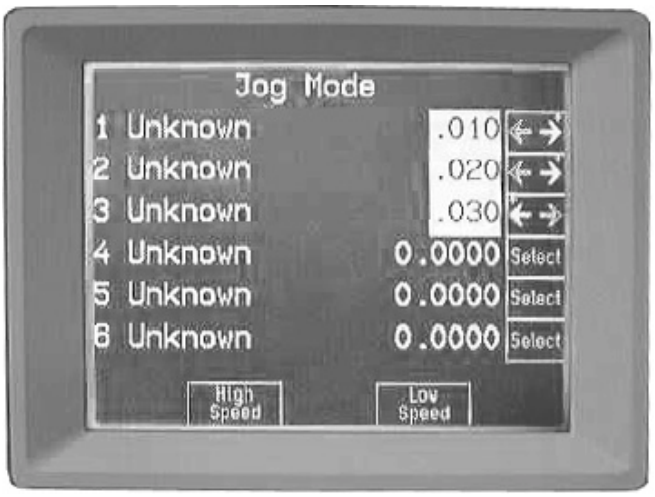


Figure 1.12: Jog Mode Screen

To toggle between the High and Low Speed loops (available only when using Direction Function buttons for control), the user must start jogging in one direction, and while jogging press, momentarily, the button to move in the opposite direction.

NOTE: If using the Control Knob, only low speed velocities are available.

5. When the user presses the **HOME** key from the **MOVE MENU** screen, the **HOME MENU** screen displays. The user chooses the desired axis by pressing the appropriate Axis Selection Button (See **Figure 1.13**).

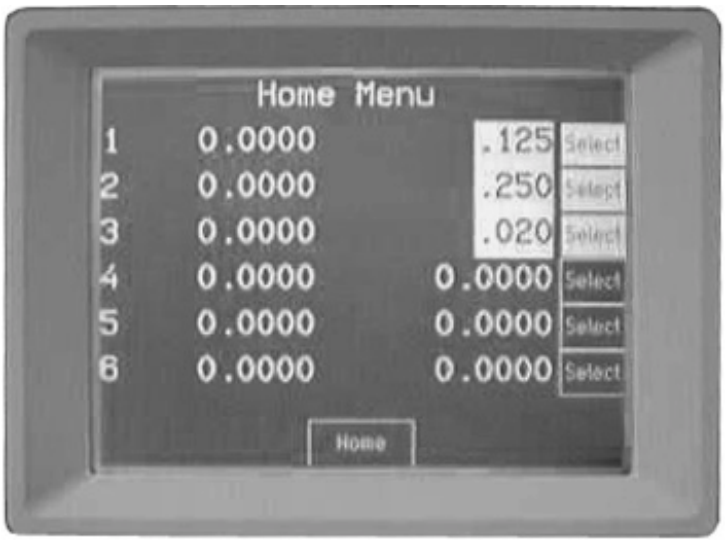


Figure 1.13: Home Menu Screen

Once an axis has been deleted the screen will display a **HOME** selection. Selecting **HOME** will execute the user's algorithm on the appropriate axes.

NOTE: ONLY ONE AXIS WILL **HOME** AT A TIME.

NOTE: While **HOME** is in progress, STOP will be displayed on the last selected axis. This allows the user to abort the **HOME** sequence if necessary.

NOTE: The user can press the **MENU** button to return to the **TOP MENU** screen.

PROGRAM Menu

When the user selects the **PROGRAM** button from the **TOP MENU** screen, the **PROGRAM MENU** screen displays, and **RUN STORED PROGRAM** (See **Figure 1.14: Program Menu Flow Diagram**).

1. When the user selects **FLOPPY DISK OPTIONS,** a **FLOPPY DISK** screen displays, giving the user ***four*** choices: **UPDATE DSP FIRMWARE, UPDATE CPU FIRMWARE, SAVE CONFIGURATION,** and **RESTORE CONFIGURATION.**

   1.1 Update DSP Firmware

   When the user selects this choice, the ESP7000 will display the task in process (Example: ***Erasing Flash***).

   1.2 Update CPU Firmware

   When the user selects this choice, the ESP7000 will display the task in process (Example: ***Updating CPU Firmware***).

   1.3 Save Configuration

   When the user selects this choice, the ESP7000 will display the task in process (Example: ***Saving Configuration***).

   1.4 Restore Configuration

   When the user selects this choice, the ESP7000 will display the task in process (Example: ***Restoring Configuration***).

2. When the user selects **RUN STORED PROGRAM,** the ESP7000 displays the **ENTER PROGRAM NUMBER** screen. The user should enter the PROGRAM #, and press the <ENTER> key.

The user can exit this screen by pressing the **MENU** button to return to **PROGRAM MENU** screen, or press the **MENU** button again to return to the **TOP MENU** screen.

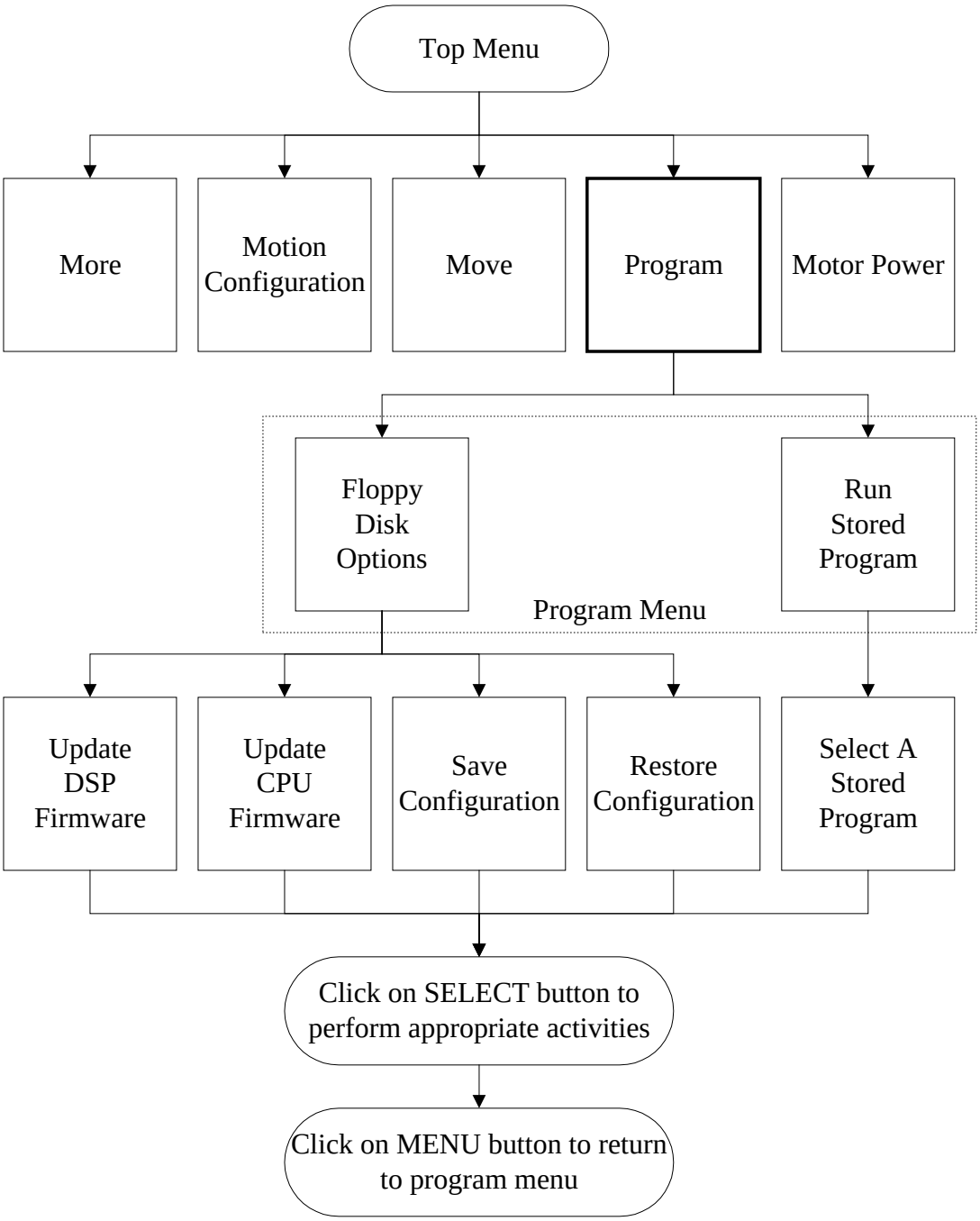


Figure 1.14: Program Menu Flow Diagram

MOTOR POWER

When the user selects the **MOTOR POWER** button from the **TOP MENU** screen, the **MOTOR POWER** screen displays, and gives the user a choice of selecting individual or ALL axes. (See Example in **Figure 1.15**).

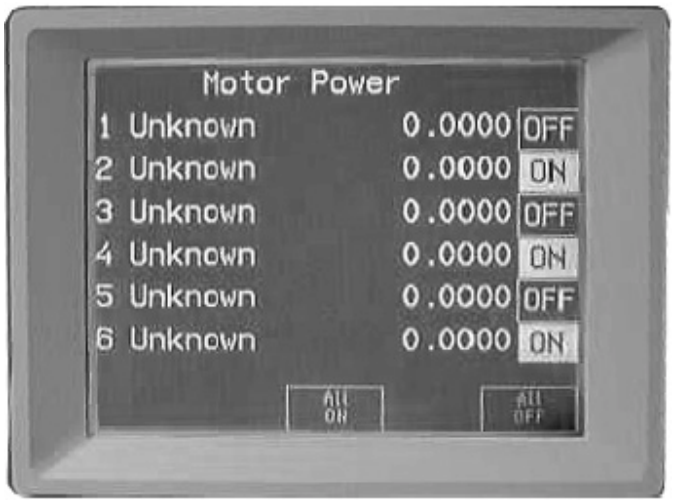


Figure 1.15: Motor Power Screen

The **MOTOR POWER** screen will appear, showing **OFF** displayed in the display window, for each axis.

1. If the user presses **ALL ON** button, the display window will show the word **ON** highlighted for each axis.
2. If the user selects a particular axis, the ESP7000 will indicate **ON** for each axis selected (See **Figure 1.15**).
3. The user can turn off the Motor Power by pressing the **ALL OFF** button, and the display window will show the word **OFF** for each axis.

1.4.4 Rear Panel Description

NOTE

For pin-outs see Appendix C.

Axis Connectors (AXIS 1 – AXIS 6)

Each installed axis driver card contains one 25-pin D-sub connector used to connect a motion device to the controller.

GPIO Digital I/O Connector

This is a 50 pin D-Sub connector used for general purpose Input/Output signals. A variety of commands are available to control these ports. See Section 3, Remote Mode and Appendix C for Connector Pin Outs.

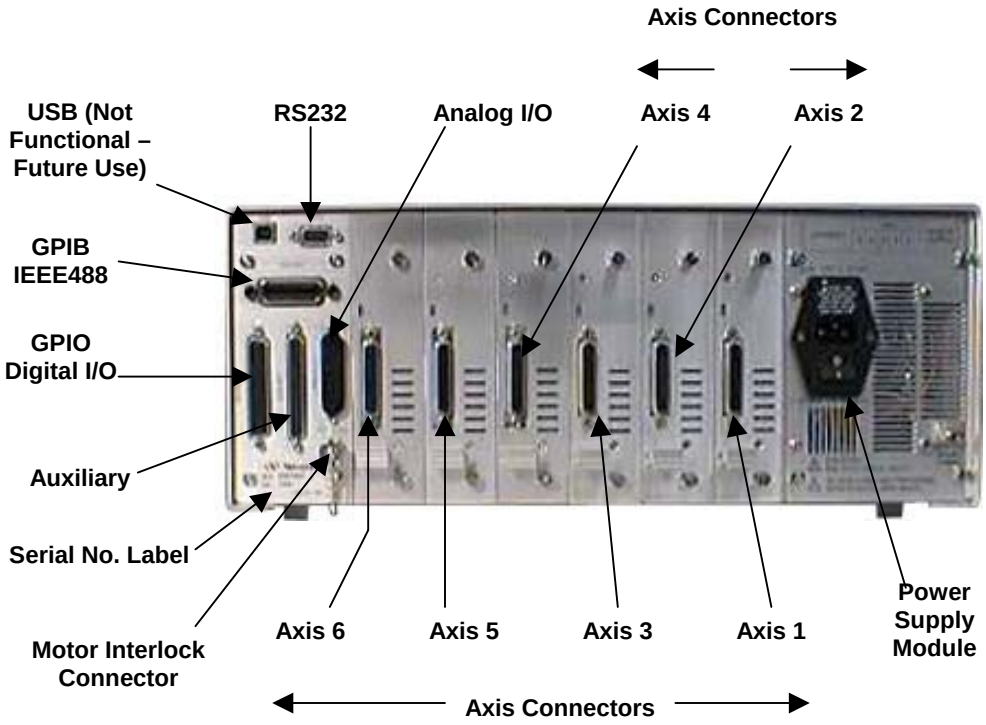


Figure 1.16: Rear Panel of the ESP7000

Motor Interlock Connector

The BNC connector provides remote motor power interlock capability. One or more external switches can be wired to remotely inhibit the motor power in a way similar to the **STOP ALL** key on the front panel.

The controller is shipped with a mating connector that provides the necessary wiring to enable proper operation without an external switch.

RS232-C Connector

The RS232-C interface to a host computer or terminal is made through this 9 pin D-Sub connector. The pin-out enables the use of an off-the-shelf, pin-to-pin cable.

GPIB IEEE488 Connector

This is a standard 24-pin connector to interface with a standard IEEE488 controller.

Power Supply Module

The power supply section on the right side of the rear panel provides a standard IEC 320 inlet, a fuse holder, and a binding post to ground the controller if the main power supply wiring does not provide earth ground terminals.

NOTE: The Main Power ON/OFF Switch is located below the IEC 320 inlet for the power cord.

Analog I/O Connector

This connector interfaces the ESP7000 controller to customer-defined Analog I/O devices. This 25-pin connector pin-outs are defined in **Section C.1.11** and **Table C.5.**

Auxiliary I/O Connector

The connector provides access to the ESP7000 Auxiliary I/O signals. The Auxiliary I/O connector provides access to:

- Additional quadrature encoder counters
- Digital I/O
- E-Stop input.

1.5 System Setup

This section guides the user through the proper set-up of the motion control system.

If not already done, carefully unpack and visually inspect the controller and stages for any damage.

Place all components on a flat and clean surface.

CAUTION

No cables should be connected to the controller at this point!

First, the controller must be configured properly before stages can be connected.

1.5.1 Power ON

Plug the AC line cord supplied with the ESP7000 into the power entry module on the rear panel.

Plug the AC line cord into the AC wall-outlet. Turn the Main Power Switch to ON (located on the Rear Panel).

After the main power is switched on, the ESP7000 will go into a STANDBY MODE (***Yellow*** LED illuminates above the Power button).

Blank Panel Version

After the power is switched on, the Blank Front Panel indicates the ON state with a ***green*** LED below the STOP ALL switch.

1.5.2 Connecting Stages

All ESP-compatible stages are electrically and physically compatible with the ESP7000 Controller. ESP-compatible stages are visually identified with a ***blue*** "ESP Compatible" sticker, on the stage. With ESP compatible stages, the configuration of each axis is identified automatically by the ESP7000 at power up. If an ES-compatible motion system was purchased, all necessary hardware to connect the stage with the ESP7000 Controller is included. The stage connects to the ESP7000 via a shielded custom cable that carries all power and control signals (encoder, limits, and home signals). The cable is terminated with a standard 25-pin D-Sub connector.

CAUTION

Never connect/disconnect stages while the ESP7000 controller is powered on.

CAUTION

Position stage(s) on a flat, stable surface before connection to the ESP7000 Controller.

With the power off, carefully connect one end of the supplied cables to the stage and the other end to the appropriate axis connector on the rear of the controller. Secure both connects with the locking thumbscrews. Refer to Appendix H, Factory Service to order replacement cables.

CAUTION

It is strongly recommended that the user read at least the System Setup (Section 1.5), before attempting to turn the controller or the motors on. Serious damage could occur if the system is not properly configured.

1.6 Quick Start

This section serves as a quick start for ESP7000 with front panel display only.

Push in the Power button on the lower left side of the Front Panel.

When the front panel Power button is pressed, a ***red*** LED illuminates briefly, then automatically changes to a ***green*** LED illuminated above the Power button.

NOTE:

The EMERGENCY STOP motion button (STOP ALL) is a latch type switch. As such, it is possible to – after powering up the ESP7000 – to have the STOP ALL MOTION disabled, as indicated by the associated LED being ***red.*** The controller can be enabled by pressing the STOP ALL button (the LED should then turn ***green***).

An error will be generated if the controller is powered up, with the STOP ALL MOTION button in the DISABLED position.

Once the **TOP MENU** is displayed, the user can **SETUP** the ESP7000 (See Section 1.4.3, MORE – SETUP).

NOTE

Any time the user calls for technical support, the firmware version is essential to trouble-shoot a problem. It is displayed every time the controller power is turned ON. User's of the Blank Front Panel can query the version with the 'VE' command (See Section 3, Remote Mode).

Users of the ESP7000 with blank front panel can skip to Section 3, Remote Mode.

The following paragraphs guide you through a quick tour of the LOCAL motion commands.

1.6.1 Motor ON

After the controller is properly configured and the stages connected as described, the motors can be powered on.

Make sure that the motion devices are placed on a flat surface and their full travel is not obstructed.

CAUTION

Be prepared to quickly turn the motor power off by pressing the STOP ALL button or power switch if any abnormal operation is observed.

After the power switch is pushed in, the controller performs the start-up sequence as described in Section 1.5.1: Power On. The default state after start-up is motor power OFF. The display indicates disabled motor power, by illuminating a ***yellow*** LED next to the appropriate Axis Selection Buttons.

To apply power to the motors, follow the procedures defined in **MOTOR POWER,** earlier in this section.

1.6.2 Homing Motion Devices

It is good practice to always home the motion device before executing any motion. As described in detail in Section 5, **Motion Control Tutorial,** *homing* a motion device means executing a special routine that locates a predetermined position.

The ESP7000 distinguishes between 3 different types of home:

1. The position defined by the location of a switch (mechanical, electro-optical, etc) mounted on the stage.
2. The position defined by the location of a switch mounted on the stage plus the occurrence of an index pulse after the switch is triggered.
3. The position that corresponds to a position count of 0.

Home position type 3 is referred to as a floating home because it can always be defined by simply moving to a specific location and re-setting the position number to 0 (See also **'DH'** command in Section 3, Remote Mode).

Before initiating a **HOME** search, the ESP7000 must be set for the type of **HOME** search to be performed. The default type at initial power up is a switch and index (type 2)!

See **'OR'** command in Section 3 (Search for home) for instructions to change the type of home search.

1.6.3 Jog

The user must display the **JOG MODE** screen.

By pressing the Axis Selection Button next to the appropriate axis, the user can **JOG** in one of several modes. **Figure 1.17** shows a different direction for each axis. For detailed description mode, see **MOVE MENU, JOG MODE** screen, paragraph 4 on Page 1-18.

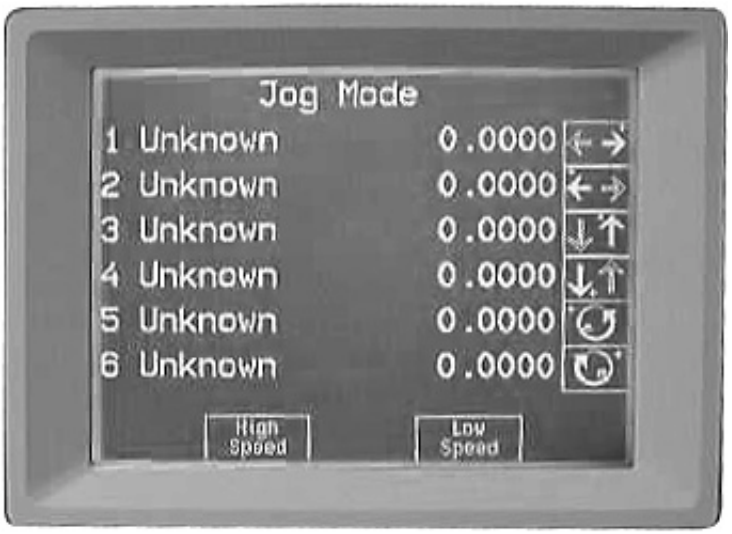


Figure 1.17: Jog Mode Screen using all directions

Example of the Sequence

1. From the **JOG MENU,** the user selects an axis by pushing the appropriate Axis Selection Button ONCE, → displays. The user then presses the RIGHT direction key ONCE, and the selected axis will STOP in the positive direction (pushing the LEFT button will move Stage in a negative direction).
2. Pushing the same Axis Selection Button AGAIN, the ← displays. The user then presses the LEFT direction button ONCE, and the selected axis will move slowly in the Positive direction (Pushing RIGHT direction button will move the stage in negative direction).
3. Pushing the same Axis Selection Button AGAIN, the ↑ displays. The user then presses the UP direction button ONCE, and the selected axis will move slowly in the positive direction (Pushing DOWN direction button will move the stage in a negative direction).

4. Pushing the same Axis Selection Button AGAIN, the user presses the button ONCE, the ↓ displays. The user then presses the DOWN direction button once, and the selected axis will move slowly in the positive direction (Pushing UP direction button will move the stage in a negative direction).
5. Pushing the same Axis Selection Button AGAIN, the ***counterclockwise rotation symbol*** displays. After the ***green*** LED illuminates above the CONTROL KNOB on the Front Panel, the user rotates the knob in a counterclockwise direction for positive stage movement, clockwise for negative stage movement.
6. Pushing the same Axis Selection Button AGAIN, the ***clockwise rotation symbol*** displays. After the ***green*** LED illuminates above the CONTROL KNOB on the Front Panel, the user rotates the knob in a clockwise direction for positive stage movement, counterclockwise for negative stage movement.

NOTE

Remember that only motions inside the software travel limits are allowed (see 'SR' command in Section 3, Remote Mode). Any move outside these limits will be ignored.

1.6.4 System Shut-Down

To shut down the system entirely, perform the following sequence:

- Wait for the stage(s) to complete their movement and come to a halt.
- Press the ***black*** Power button on the front panel to put the ESP7000 Controller in a STANDBY mode.

Section 2 – Modes of Operation

2.1 Overview of Operating Modes

The ESP7000 can be operated in one of two modes:

- LOCAL mode
- REMOTE mode

Following is an overview of these two modes of operation.

2.1.1 LOCAL Mode

This mode is applicable only if your unit is equipped with the optional front panel with display. If your ESP7000 is equipped with the blank front panel, you may skip to the REMOTE Mode, Section 3 of this manual.

In LOCAL Mode the user has access to a sub-set of the ESP7000 motion control functions through the use of Front Panel buttons and display menus.

This allows the user to set up a motion system without use of IEEE488, RS-232C Interfaces, or using a computer or terminal.

NOTE: Each of the display window screens is described in Section 1.4.3 (Description of Front Panel Versions, and Front Panel with Display).

2.1.2 REMOTE Mode

Operated remotely, the ESP7000 functions in two distinct modes. In *Remote Mode,* the ESP7000 receives motion commands through one of its interfaces (IEEE488, or RS-232C) using a computer or terminal. Additionally, an optional alphanumeric keypad with an LCD display enables the user to access the full command set of the ESP7000 without the use of a computer interface (See Section 7.2, of this document).

The ESP7000 employs a set of over 100 commands. Please refer to Section 3 (Remote Mode), for a detailed description of the ESP7000 command set.

In Program Execution Mode, internally stored programs are executed without any operator intervention (See Section 3.1).

2.2 Operating Options in the LOCAL Mode

This section defines the 'HOW TO' instructions for the functions of those Operating Options in the LOCAL Mode. Please remember that all functions can also be accessed with remote commands (See Section 3: Remote Mode). Typical functions or parameters that can be set are:

- Motor Power
- Motor Configuration
- Move functions
- Program execution

2.2.1 Motor Power

The Motor Power option is obtained from the TOP MENU screen. It allows the user to set the power ON/OFF for individual axes or all axes at once (See **MOTOR POWER** and **Figure 1.15**).

When the user presses the **MOTOR POWER** button from the TOP MENU screen, the **MOTOR POWER** screen will display. For each axis that has a stage connected and a valid configuration, a Menu Box will appear next to the corresponding Axis Selection Button. Inside each box, the label ON or OFF will indicate the state of the respective motor. If an axis does not detect a stage or it does not have valid configuration, it will not have a Menu Box.

By pressing an Axis Selection Button, the **MOTOR POWER** of the corresponding axis will switch state. If the **MOTOR POWER** was OFF previously, it will be turned ON, and vice versa.

To control all axes at once, two function buttons are active on the bottom of the display. One is labeled **ALL OFF** and the other **ALL ON.** By pressing one of them, all axes could be turned OFF or ON, respectively.

NOTE: The **MOTOR POWER** status of every axis is also constantly indicated by a set of six three-color LEDs located next to the Axis Selection Buttons. The color interpretation is as follows:

- Green — motor power ON
- Yellow — motor power OFF
- Red — driver fault
- Off (not lit) — stage missing or driver not configured

2.2.2 Motion Configuration

This mode is activated from the top menu. The **MOTION CONFIGURATION** screen first shows a list of all axes. The user must select an axis to configure by pressing one of the Axis Selection Buttons.

Once an axis is selected, the display shows the following list of motion parameter names and their corresponding values (See **Figure 1.6**):

- Trajectory = Trapezoid
- Velocity = 1.2500
- Acceleration = 5.0000
- Deceleration = 5.0000
- Jerk = 20.0000
- Foll. Error = 1.0000

A parameter can be selected by pressing one of the Axis Selection Buttons. Once selected, the value gets highlighted and can be modified by using the numeric keypad, the arrow buttons or the control knob.

Pressing the Menu button labeled PID, a list of all servo parameters and their values will appear on the display. They have the following meaning (See **Figure 1.6**):

- Kp proportional gain
- Kd derivative gain
- Ki integral gain
- IL integral limit
- Vff velocity feed-forward
- Aff acceleration feed-forward

Selecting one of them with the Axis Selection Buttons will let the user modify any parameter value, using the same controls and procedures described above.

To return to the **TOP MENU,** the user must press the **MENU BUTTON** two or three times, depending on the menu level the display is at.

2.2.3 Move – LOCAL Mode

The **MOVE MENU** screen displays the name and position of all axes. Through the Menu Selection Buttons, the user can select one of the five different types of motion – **CYCLE MOVE, RELATIVE MOVE, Manual JOG, HOME** – or to **ZERO POSITION** the counter or a particular axis.

ZERO

The **ZERO POSITION** screen is enabled by pressing the **ZERO** function button in the **MOVE MENU.** All installed axes will have an associated Axis Selection Button. Pressing any of the Axis Selection Buttons will zero the position of the respective axis.

NOTE: All configured axes can have their position zeroed, even if the **MOTOR POWER** is turned off.

Pressing the **MENU BUTTON** returns the display to the **MOVE MENU.**

CYCLE

The **CYCLE MOTORS** screen is enabled by pressing the **CYCLE** function button in the **MOVE MENU.** This screen displays two positions values. The first one represents Cycle Position "A", and the second represents Cycle Position "B".

Pressing any of the Axis Selection Buttons will highlight Cycle Position "A" of the respective axis. The value can be modified by using the numerical keypad.

Once all desired destinations have been made, the user can move all axes simultaneously. Pressing **MOVE BUTTON** will start a motion on the axis for which the **Selected** button is highlighted. Pressing the **RED STOP ALL** Button will STOP a motion on all axes.

NOTE: All axes start the motion at the same time but each axis moves with its specified motion parameters (velocity, acceleration, trajectory type, etc.). This means that the axes perform a non-coordinated motion, thus not reaching their respective target at the same time.

NOTE: Pressing the **RED STOP ALL** Button in the lower right corner of the front panel will initiate a motion inhibit sequence on all axes. This function is user programmable and could be set individually for each axis anywhere from *a hard abrupt stop* to a *do-nothing* action. The default setting is a controlled stop followed by a motor power off.

Pressing the **MENU BUTTON** returns the display to the **MOVE MENU.**

RELATIVE

The **RELATIVE MOVE** screen is enabled by pressing the **Relative** Menu Selection Button in the **MOVE MENU.** The screen displays two position values. The first one represents the desired motion increment and the second the current position.

To avoid a move on the axes for which no value has been entered when a **MOVE ALL** command is issued, the user should set increment value to zero.

All configured axes that have a motion device connected will have an associated Menu selection Button. Pressing any of the Menu Selection Buttons will highlight the desired motion increment of the respective axis. The values can be modified by using the numeric keypad.

Once all desired motion increments have been made, the user can select to move one or all axes simultaneously. Pressing **MOVE ONE** Menu Selection Button will start a motion on the axis for which the **Selected** button is highlighted. Pressing **MOVE ALL** Menu Selection Button will start a motion on all axes.

NOTE: All axes start the motion in the same time but each axis moves with its specified motion parameters (velocity, acceleration, trajectory type, etc). This means that the axes perform a non-coordinated motion, thus not reaching their respective target at the same time.

By pressing any of these STOP buttons, the user can abort the motion on any moving axis individually. If the **STOP ALL BUTTON** from the Menu Selection Buttons is pressed, all axes in motion will stop moving simultaneously with the specified deceleration.

NOTE: Pressing the **RED STOP ALL BUTTON** in the lower right corner of the front panel, will initiate a motion inhibit sequence on all axes. This function is user programmable and could be set individually for each axis anywhere from *a hard emergency stop* to a *do-nothing* action. The default setting is a controlled stop followed by a motor power off.

Pressing the **MENU BUTTON** returns the display to the **MOVE MENU.**

JOG

The **JOG MODE** screen is enabled by pressing the **JOG** Menu Selection Button in the **MOVE MENU.**

All configured axes that have a motion device connected will have an associated Axis selection Button. Pressing any of the Axis Selection Buttons will display a graphical jog symbol in the function button box and highlight it. There are a total of six different symbols that can be selected by pressing the same button repeatedly. The symbols and their meaning are, in order:

Select axis not yet selected for jogging

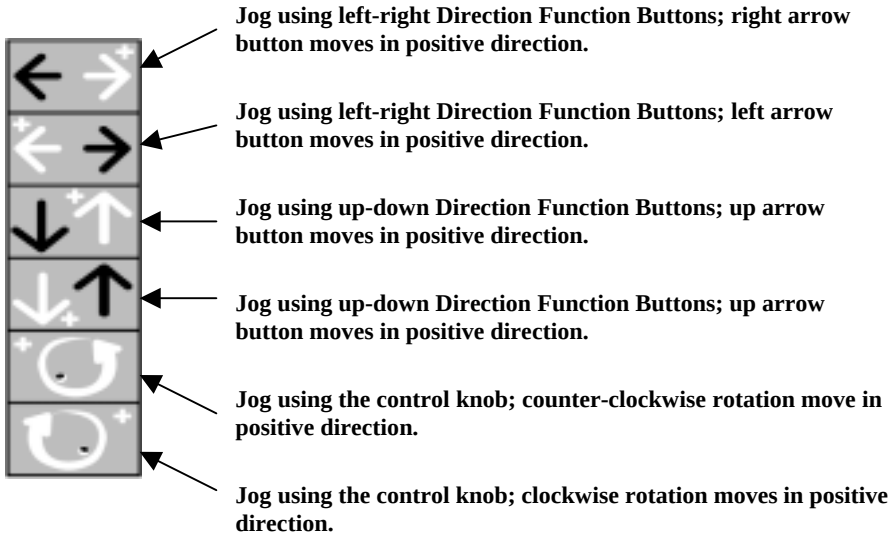


Figure 2.1: Jog Mode Direction Screen Images

NOTE: Multiple axes can be assigned to the same control (arrow buttons or control knob). When a control is activated, all associated axes move at the same time, in the specified direction.

NOTE: While the Direction Function Buttons function as typical jog buttons, the control knob acts as an encoder wheel to which the associated stage is slaved. This means that the motor will turn at a rate proportional to the control knob rotation. On the other hand, motors controlled by the Direction Function Buttons move at a constant velocity, predetermined by the jog velocity setting.

When the **JOG MODE** screen is displayed, two **JOG** velocities can be selected: **HIGH SPEED** or **LOW SPEED.**

When the user selects **HIGH SPEED** velocity for **JOG,** the **HIGH-SPEED** screen display all 6 axes. To change a **JOG** velocity, the user can enter the **HIGH SPEED** velocity value desired from the numeric keypad on the front panel, and press the <ENTER> key. The parameters are displayed in the window as they are typed.

NOTE:

- The Direction Function Buttons are used to select the direction for **HIGH-SPEED** or **LOW SPEED JOG** velocity.
- The control knob is used to select the direction for **LOW SPEED JOG** velocity.
- The numeric keypad is used to enter or change **JOG** velocities.

The user can toggle between the **HIGH** and **LOW SPEED** loops (available ONLY when using Direction Function Buttons as a method of control), and can start jogging in one direction. While jogging, press momentarily the opposite direction button changing speed.

Pressing the **MENU BUTTON** returns the display to the **JOG MODE** screen.

HOME

The **HOME MENU** screen is enabled by pressing the **HOME MENU** Selection Button in the **MOVE MENU.** This screen displays the name and position values of each axis.

All configured axes that have a motion device connected, will have an associated Axis Selection Button. Pressing one of these Menu Selection Buttons will highlight the corresponding Axis Selection Button box, indicating that the respective axis is selected to perform a **HOME** search routine. Pressing the Axis Function Button of an already selected axes will deselect it.

When the **Home** function button is pressed, the controller sequentially executes a **HOME** finding routine on all selected axes.

Once the **HOME** routine has started, **"STOP"** will be displayed next to all Axis Selection Buttons on whom **"HOME"** was selected. Pressing one of these buttons will step the **HOME** sequence on the selected axis.

NOTE: Pressing the **RED STOP ALL** button in the lower right corner of the front panel will initiate a motion inhibit sequence on all axes. This function is user programmable and could be set individually for each axis anywhere from *a hard emergency stop* to a *do-nothing* action. The default setting is a controlled stop followed by a motor power off.

Pressing the **MENU BUTTON** returns the display to the **MOVE MENU.**

Section 3 – Remote Mode

3.1 Programming Modes

The ESP is a command driven system. In general commands are a series of two letter ASCII characters preceded by an axis number and followed by parameters specific to the command. To communicate with the ESP controller, a host terminal has to transfer ASCII character commands according to the respective communication protocol (See Section 3.2 for IEEE488 or RS232 interfaces).

As briefly mentioned in Section 2, the ESP distinguishes between two different programming modes:

COMMAND MODE

In this mode, the ESP controller provides a command input buffer enabling the host terminal (e.g., PC) to download a series of commands and then proceed to other tasks while the ESP controller processes the commands.

As command characters arrive from the host terminal, they are placed into the command buffer. When a carriage-return (ASCII 13 decimal) terminator is received, the command is interpreted. If the command is valid and its parameter is within the specified range, it will be executed. If the command contains an error, it will not be executed and a corresponding error message will be stored in the error buffer.

NOTE

The ESP power up state is command mode.

An example of a typical command sequence is shown below:

Example 1:

1PA + 30	*move axis 1 to absolute position 30 units*
1WS	*wait for axis 1 to stop*

2PR-10 | *move axis 2 to relative position 10 units*

Assuming that axis 1 and 2 are configured, ***Example 1*** instructs the ESP controller to move axis 1 to absolute position +30 units, wait for it to stop, and then move axis 2 motor to relative –10 units.

Note that a command prefix identifies the axis or group that should execute a command. Commands received without an axis prefix generate an error. If a command is referenced to a non-existing axis, an error is also generated. See Section 3.4 for further details on the command syntax.

Also note that it is necessary to explicitly instruct the ESP controller with the WS (Wait for Stop) command to wait for axis 1 motion to stop. This is necessary because the ESP controller executes commands continuously as long as there are commands in the buffer unless a command is fetched from the buffer that instructs the controller to wait. Executing a move does not automatically suspend command execution until the move is complete. If the WS command were not issued in ***Example 1,*** the controller would start the second move immediately after the first move begins and simultaneously move axis 1 and axis 2.

NOTE

Unless instructed otherwise, the ESP controller executes commands in the order received without waiting for completion of previous commands.

Remember that commands must be terminated with a carriage-return (ASCII 13 decimal). Until a terminator is received, characters are simply kept in contiguous buffer space without evaluation.

Example 2:

1PA+30; 1WS; 2PR-10

Example #1 and ***Example #2*** perform the same operations. In ***Example #2*** however, semicolons are used in place of carriage-returns as command delimiters, keeping the ESP controller from interpreting any commands on that line until the carriage-return terminator is received at the very end of the string.

PROGRAM EXECUTION MODE

The ESP controller also implements an internal program execution mode that enables the user to store up to 100 programs in a 64kB non-volatile memory.

Even while executing stored programs, the ESP controller maintains open communication channels so that the host terminal can continue to direct the ESP to report any desired status, and even execute other motion commands.

Let's illustrate program execution mode using the previous example:

Example 3:

EP	*invoke program entry mode*
1PA+30	*enter program*
1WS	
2PR-10	
QP	*exit program entry mode*
1EX	*execute compiled program #1*

As shown above, the sequence of commands has to be downloaded into the ESP controller program memory without inadvertently executing them. To facilitate this, the system provides the EP (Enter Program) command; characters received thereafter are redirected to program memory. Command syntax and parameters are not evaluated (even after the carriage-return). Instead, they are treated as a series of characters to be stored in contiguous memory.

3.2 Remote Interfaces

In this manual, *Remote Interface* refers to the two communication interfaces that the controller can use to communicate with a computer or a terminal via commands in ASCII format. It is not called a *Computer Interface* since any device capable of sending ASCII characters can be interfaced with the controller.

The remote interface should not be confused with the General Purpose Input/Output (digital I/Os, a.k.a. GPIO).

3.2.1 RS-232C Interface

HARDWARE CONFIGURATION

The serial (RS-232C) communication interface on the ESP controller is accessed through the 9 pin Sub-D connector located on the rear panel. The pin out is designed to interface directly with an IBM PC or compatible computer, using a straight through cable.
Appendix **C** shows the pin out of the RS-232C connector and different cable types that may be used to interface to a computer.

COMMUNICATION PROTOCOL

The RS-232C interface must be properly configured on both devices communicating. A correct setting is one that matches **all** parameters (baud rate, number of data bits, number of stop bits, parity type and handshake type) for both devices.

The ESP RS-232C configuration is fixed at **8 data bits, no parity, and 1 stop bit.**

To prevent buffer overflow when data is transferred to the ESP controller input buffer, a CTS/RTS hardware handshake protocol is implemented. The host terminal can control transmission of characters from the ESP by enabling the Request To Send (RTS) signal once the controller's Clear To Send (CTS) signal is ready. Before sending any further characters, the ESP will wait for a CTS from the host.

As soon as its command buffer is full, the controller de-asserts CTS. Then, as memory becomes available because the controller reads and executes commands in its buffer, it re-asserts the CTS signal to the host terminal.

3.2.2 IEEE-488 Interface

HARDWARE CONFIGURATION

A typical IEEE-488 setup consists of a controller (host terminal) and several devices connected to the bus. All devices are connected in parallel to the data lines, data management and synchronization lines.

As a result of this type of connection, each device on the bus must have a unique address so that the controller can selectively communicate with it.

The address can be set through the optional front panel display or with the **SA** (set address) command. (note that the factory default is address 1)

COMMUNICATION PROTOCOL

The IEEE-488 interface is implemented on the motion controller somewhat differently from a typical instrument because the standard IEEE-488.2 command set and command format are inadequate for a complex motion control. Since the ESP controller has its own language and command set, the IEEE-488 interface is used only as a communication port. The extended protocol is not supported.

The ESP controller has an ASCII command set and also outputs system status in ASCII format. It features a command input buffer. If the buffer fills up, the ESP will not allow further communication until memory becomes available to accept new characters.

To send a command to the ESP controller, use the command specific to your IEEE-488 terminal [e.g., output (ASCII)].

If the host terminal asks the controller for a response [e.g., input (ASCII)] and no response is obtained, the controller will eventually will time-out.

USE OF SRQ LINE

The ESP controller can be instructed to generate an IEEE-488 service request (SRQ) upon processing the **RQ** command. This allows the user to generate SRQs anywhere within the ESP command stream thereby facilitating efficient event synchronization capability with the host computer.

The following example illustrates the use of the **RQ** command:

1PR10; 1WS100; 2PR10; 3PR10; 3WS100; RQ
In the above example, the SRQ line is asserted only after execution of the sequence preceding the **RQ** command is finished.

SERIAL POLL

When the IEEE-488 controller senses a service request on the bus, it creates an interrupt to the application program (if configured to do so). The application program must contain a service routine for this interrupt. First, the program must determine which device on the bus generated the service request. This is usually achieved with a function

called Serial Poll. The exact syntax for the serial poll command depends on the IEEE-488 controller.

Using that interrupt service routine, a serial poll command can be issued to each device. The device polled at each instance will respond with a status byte. Bit 6 of the status byte indicates whether a specific device (i.e., ESP controller) generated the service request or not. Bits 0 through 5 are under user control and are set with the **RQ** command. For example, command "**RQ5**" sets bits 0 and 2. This is useful in helping the application program determine which **RQ** in a program with multiple **RQ**s generated the SRQ.

3.3 Software Utilities

In order to communicate with the controller, the user must have a terminal or a computer capable of communicating through RS-232C or IEEE488. One approach is to use a computer with communications software that can emulate a terminal. Windows 95 provides an RS232 terminal emulation program named Hyper Terminal (HyperTrm.Exe) located in Accessories. HyperTrm allows the user to send ASCII commands to the motion controller. The user can even download text files with stored programs. Additionally, it can be used to download controller firmware for future upgrades.

For IEEE488 communications National Instruments Inc. provides a program named IBIC with their products that allow the user to send and receive ASCII characters and download files. This could be useful in determining that the interface is working.

3.4 Command Syntax

As mentioned previously, the ESP controller utilizes an ASCII command set and also outputs system status in ASCII format. Commands may be either upper or lower case characters.
The diagram below illustrates the ESP controller command syntax. As indicated in this diagram, a valid command consists of three main fields. The first field consists of a numerical parameter "xx", the second field consists of a two letter ASCII mnemonic, and the third field consists of numerical parameter "nn". The command is finally terminated by a carriage return. For example, 3PA10.0 is a valid command.

If a command does not require parameter “xx” and/or parameter “nn”, that field may be skipped by leaving a blank character (space). For example, BO1, 3WS, and AB are all valid commands.

If a command requires multiple parameters in the third field, all these parameters must be comma delimited. For example, 1HN1,2 is a valid command.

In a similar fashion, multiple commands can be issued on a single command line by separating the commands by a semi-colon (;). For example, 3MO; 3PA10.0; 3WS; 3MF is a valid command line.

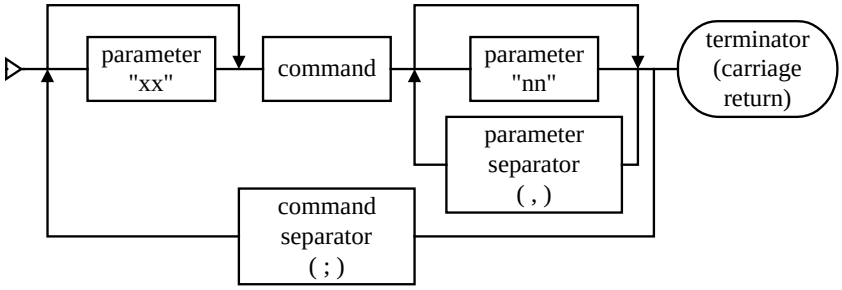


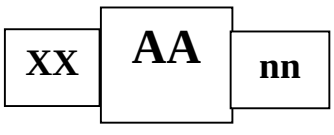
Figure 3.1: Command Syntax Diagram

NOTE

A controller command (or a sequence of commands) has to be terminated with a carriage return character. However, responses from the controller are always terminated by a carriage return/line feed combination. This setting may not be changed. If the IEEE interface is used, the IEEE controller has to be configured to terminate the input (read) function when it senses the line feed character.

3.4.1 Summary of Command Syntax

COMMAND FORMAT



The general format of a command is a two character mnemonic (AA). Both upper and lower case are accepted. Depending on the command, it could also have optional or required preceding (xx) and/or following (nn) parameters.

BLANK SPACES

Blank spaces are allowed and ignored between parameters and commands. For the clarity of the program and memory saving considerations, use blank spaces with restraint. The following two commands are equivalent.

2 PA 1000
2PA1000

but the first example is very confusing and uses more than twice the memory.

COMMAND LINE

Commands are executed line by line. A line can consist of one or a number of commands. The controller will interpret the commands in the order they are received and execute them sequentially. This means that commands issued on the same line are executed significantly closer to each other than if they would be issued on separate lines. The maximum number of characters allowed on a command line is 80.

SEPARATOR

Commands issued on the same line must be separated by semicolons (**;**).

Multiple parameters issued for the same command are separated by commas (,).

TERMINATOR

Each command line must end with a line terminator, i.e., carriage return.

3.5 Command Summary

The controller understands many commands. The following tables list all of them, sorted first by category and then alphabetically. The tables also show the operating modes in which each command can be used. The acronyms used in the tables have the following meaning:

IMM	IMMediate mode	Controller is idle and the commands will be executed immediately.
PGM	ProGraM mode	Controller does not execute but stores all commands as part of a program. EP activates this mode and **QP** exits it.
MIP	Motion In Progress	Controller executes command on the specified axis while in motion.

3.5.1 – Command List by Category (Table)

In a PDF format you may click on a page number to automatically be connected to the corresponding Command Page

GENERAL MODE SELECTION

Cmd.	Description	IMM	PGM	MIP	Page
BQ	Enable/disable DIO jog mode	♦	♦	♦	3-41
BR	Set serial communication speed	♦	♦	♦	3-42
DO	Set DAC offset	♦	♦	♦	3-56
FP	Set position display resolution	♦	♦	♦	3-65
MF	Power OFF	♦	♦	♦	3-102
MO	Power ON	♦	♦	♦	3-103
QD	Update Unidriver amplifier	♦	♦	♦	3-121
RS	Reset the controller	♦		♦	3-132
TJ	Set trajectory mode	♦	♦		3-149
ZA	Set amplifier configuration	♦	♦		3-167
ZB	Set feedback configuration	♦	♦		3-170
ZE	Set E-stop configuration	♦	♦		3-172
ZF	Set following error configuration	♦	♦		3-174
ZH	Set hardware limit configuration	♦	♦		3-176
ZS	Set software limit configuration	♦	♦		3-178
ZU	Get ESP system configuration	♦		♦	3-180
ZZ	Set system configuration	♦	♦		3-183

STATUS FUNCTIONS

Cmd.	Description	IMM	PGM	MIP	Page
DP	Get target position	♦		♦	3-57
DV	Get working speed	♦		♦	3-58
ID	Get stage model and serial number	♦		♦	3-91
MD	Get axis motion status	♦		♦	3-101
PH	Get hardware status	♦		♦	3-117
TB	Get error message	♦		♦	3-147
TE	Get error number	♦		♦	3-148
TP	Get position	♦		♦	3-150
TS	Get controller status	♦		♦	3-151
TV	Get velocity	♦		♦	3-152
TX	Get controller activity	♦		♦	3-153
VE	Get firmware version	♦		♦	3-159
XM	Get available program memory	♦		♦	3-165

MOTION & POSITION CONTROL

Cmd.	Description	IMM	PGM	MIP	Page
AB	Abort motion	♦		♦	3-21
DH	Define home	♦	♦		3-54
MT	Move to hardware travel limit	♦	♦	♦	3-104
MV	Move indefinitely	♦	♦	♦	3-105
MZ	Move to nearest index	♦	♦		3-107
OR	Origin searching	♦	♦	♦	3-111
PA	Move absolute	♦	♦	♦	3-113
PR	Move relative	♦	♦	♦	3-120
ST	Stop motion	♦	♦	♦	3-145

MOTION DEVICE PARAMETERS

Cmd.	Description	IMM	PGM	MIP	Page
FE	Set following error threshold	♦	♦	♦	3-54
FR	Full step resolution	♦	♦	♦	3-56
GR	Set gear ratio	♦	♦	♦	3-57
QG	Set gear constant	♦	♦		3-122
QI	Motor current	♦	♦		3-123
QM	Define motor type	♦	♦		3-124
QR	Torque reduction	♦	♦	♦	3-126
QS	Set microstep factor	♦	♦		3-127
QT	Define tachometer constant	♦	♦		3-128
QV	Set motor voltage	♦	♦		3-129
SI	Set master-slave jog update interval	♦	♦	♦	3-138
SK	Set slave axis jog velocity coefficients	♦	♦	♦	3-139
SL	Set left limit	♦	♦	♦	3-140
SN	Set units	♦	♦		3-142
SR	Set right limit	♦	♦		3-143
SS	Set master-slave relationship	♦	♦		3-144
SU	Set encoder resolution	♦	♦		3-146

PROGRAMMING

Cmd.	Description	IMM	PGM	MIP	Page
DL	**Define label**		♦		3-55
EO	Automatic execution on power on	♦		♦	3-59
EP	Enter program download mode	♦			3-60
EX	Execute stored program	♦	♦		3-63
JL	Jump to label		♦	♦	3-94

LP	List program	♦		♦	3-	100
QP	Quit program mode	♦			3-	125
SM	Save to non-volatile memory	♦			3-	141
XM	Get available program memory	♦		♦	3-	165
XX	Delete a stored program	♦		♦	3-	166

TRAJECTORY DEFINITION

Cmd.	Description	IMM	PGM	MIP	Page	
AC	Set acceleration	♦	♦	♦	3-	22
AE	Set e-stop deceleration	♦	♦	♦	3-	24
AG	Set deceleration	♦	♦	♦	3-	27
AU	Set maximum acceleration	♦	♦	♦	3-	31
BA	Set backlash compensation	♦	♦	♦	3-	32
CO	Set linear compensation	♦	♦	♦	3-	44
JH	Set jog high speed	♦	♦	♦	3-	92
JK	Set jerk rate	♦	♦	♦	3-	93
JW	Set jog low speed	♦	♦	♦	3-	95
OL	Set home search low speed	♦	♦	♦	3-	109
OH	Set home search high speed	♦	♦	♦	3-	108
OM	Set home search mode	♦	♦	♦	3-	110
SH	Set home preset position	♦	♦	♦	3-	137
UF	Update filter parameters	♦	♦	♦	3-	154
VA	Set velocity	♦	♦	♦	3-	157
VB	Set base velocity for step motors	♦	♦	♦	3-	158
VU	Set maximum speed	♦	♦	♦	3-	161

FLOW CONTROL & SEQUENCING

Cmd.	Description	IMM	PGM	MIP	Page	
DL	Define label		♦		3-	55
JL	Jump to label		♦	♦	3-	94
RQ	Generate service request	♦	♦	♦	3-	131
SA	Set device address	♦	♦	♦	3-	134
WP	Wait for absolute position crossing	♦	♦	♦	3-	162
WS	Wait for stop	♦	♦	♦	3-	163
WT	Wait for time	♦	♦	♦	3-	164

I/O FUNCTIONS

Cmd.	Description	IMM	PGM	MIP	Page	
AM	Set analog input mode	♦	♦	♦	3-	29
BG	Assign DIO bits to execute stored programs	♦		♦	3-	33

BK	Assign DIO bits to inhibit motion	♦	♦	♦	3-	34
BL	Enable DIO bits to inhibit motion	♦	♦	♦	3-	35
BM	Assign DIO bits to notify motion status	♦	♦	♦	3-	36
BN	Enable DIO bits to notify motion status	♦	♦	♦	3-	37
BO	Set DIO Port Direction	♦	♦	♦	3-	38
BP	Assign DIO for jog mode	♦	♦	♦	3-	40
BQ	Enable/disable DIO jog mode	♦	♦	♦	3-	41
DC	Setup data acquisition	♦		♦	3-	46
DD	Get data acquisition done status	♦		♦	3-	50
DE	Enable/disable data acquisition	♦		♦	3-	51
DF	Get data acquisition sample count	♦		♦	3-	52
DG	Get acquisition data	♦		♦	3-	53
ES	Define event action command string	♦		♦	3-	51
PC	Set position compare mode	♦	♦	♦	3-	114
RA	Read analog input	♦		♦	3-	130
SB	Set DIO state	♦	♦	♦	3-	135
UL	Wait for DIO bit low		♦		3-	156
UH	Wait for DIO bit high		♦		3-	155

GROUP FUNCTIONS

Cmd.	Description	IMM	PGM	MIP	Page	
HA	Set group acceleration	♦	♦	♦	3-	68
HB	Read list of groups assigned	♦		♦	3-	70
HC	Move group along an arc	♦	♦	♦	3-	71
HD	Set group deceleration	♦	♦	♦	3-	73
HE	Set group E-stop deceleration	♦	♦	♦	3-	75
HF	Group motor power OFF	♦	♦	♦	3-	76
HJ	Set group jerk	♦	♦	♦	3-	77
HL	Move group along a line	♦	♦	♦	3-	78
HN	Create new group	♦	♦		3-	80
HO	Group motor power ON	♦	♦	♦	3-	82
HP	Get group position	♦		♦	3-	83
HQ	Wait for group via point buffer near empty	♦	♦	♦	3-	84
HS	Stop group motion	♦	♦	♦	3-	85
HV	Set group velocity	♦	♦	♦	3-	86
HW	Wait for group motion to stop	♦	♦	♦	3-	87
HX	Delete a group	♦	♦	♦	3-	89
HZ	Get group size	♦		♦	3-	90

DIGITAL FILTERS

Cmd.	Description	IMM	PGM	MIP	Page
AF	Acceleration/Deceleration feed-forward gain	♦	♦	♦	3-26
CL	Set closed loop update interval	♦	♦	♦	3-43
DB	Set position deadband	♦	♦	♦	3-45
KD	Set derivative gain Kd	♦	♦	♦	3-96
KI	Set integral gain Ki	♦	♦	♦	3-97
KP	Set proportional gain Kp	♦	♦	♦	3-98
KS	Set saturation coefficient Ks	♦	♦	♦	3-99
UF	Update Filter Parameters	♦	♦	♦	3-154
VF	Set velocity feed-forward gain	♦	♦	♦	3-160

MASTER-SLAVE MODE DEFINITION

Cmd.	Description	IMM	PGM	MIP	Page
GR	Set master-slave Ratio	♦	♦	♦	3-57
SI	Set master-slave jog update interval	♦	♦	♦	3-138
SK	Set slave axis jog velocity coefficients	♦	♦	♦	3-139
SS	Set master-slave mode	♦	♦		3-144

3.5.2Command List – Alphabetical (Table)

In a PDF format you may click on a page number to automatically be connected to the corresponding Command Page

Cmd.	Description	IMM	PGM	MIP	Page
AB	Abort Motion	♦		♦	3- 21
AC	Set acceleration	♦	♦	♦	3- 22
AE	Set e-stop deceleration	♦	♦	♦	3- 24
AF	Set acceleration feed-forward gain	♦	♦	♦	3- 26
AG	Set deceleration	♦	♦	♦	3- 27
AM	Set analog input mode	♦	♦	♦	3- 29
AP	Abort program	♦	♦	♦	3- 30
AU	Set maximum acceleration and deceleration	♦	♦	♦	3- 31
BA	Set backlash compensation	♦	♦	♦	3- 32
BG	Assign DIO bits to execute stored programs	♦		♦	3- 33
BK	Assign DIO bits to inhibit motion	♦	♦	♦	3- 34
BL	Enable DIO bits to inhibit motion	♦	♦	♦	3- 35
BM	Assign DIO bits to notify motion status	♦	♦	♦	3- 36
BN	Enable DIO bits to notify motion status	♦	♦	♦	3- 37
BO	Set DIO port A, B, C direction	♦	♦	♦	3- 38
BP	Assign DIO bits for jog mode	♦	♦	♦	3- 40
BQ	Enable DIO bits for jog mode	♦	♦	♦	3- 41
BR	Set serial communication speed	♦	♦	♦	3- 42
CL	Set closed loop update interval	♦	♦	♦	3- 43
CO	Set linear compensation	♦	♦	♦	3- 44
DB	Set position deadband	♦	♦	♦	3- 45
DC	Setup data acquisition	♦		♦	3- 46
DD	Get data acquisition done status	♦		♦	3- 50
DE	Enable/disable data acquisition	♦		♦	3- 51
DF	Get data acquisition sample count	♦		♦	3- 52
DG	Get acquisition data	♦		♦	3- 53
DH	Define home	♦	♦	♦	3- 54
DL	Define label	♦	♦	♦	3- 55
DO	Set DAC offset	♦	♦	♦	3- 56
DP	Read desired position	♦		♦	3- 57
DV	Read desired velocity	♦		♦	3- 58
EO	Automatic execution on power on	♦		♦	3- 59
EP	Enter program mode	♦			3- 60
ES	Define event action command string	♦		♦	3- 61
EX	Execute a program	♦	♦		3- 63
FE	Set maximum following error threshold	♦	♦	♦	3- 64

TABLE 3.5.2 – Command List – Alphabetical (Continued)

In a PDF format you may click on a page number to automatically be connected to the corresponding Command Page

Cmd.	Description	IMM	PGM	MIP	Page
FP	Set position display resolution	♦	♦	♦	3- 65
FR	Set full step resolution	♦	♦	♦	3- 66
GR	Set master-slave reduction ratio	♦	♦	♦	3- 67
HA	Set group acceleration	♦	♦	♦	3- 68
HB	Read list of groups assigned	♦		♦	3- 70
HC	Move group along an arc	♦	♦	♦	3- 71
HD	Set group deceleration	♦	♦	♦	3- 73
HE	Set group e-stop deceleration	♦	♦	♦	3- 75
HF	Group motor power off	♦	♦	♦	3- 76
HJ	Set group jerk	♦	♦	♦	3- 77
HL	Move group along a line	♦	♦	♦	3- 78
HN	Create new group	♦	♦		3- 80
HO	Group on	♦	♦	♦	3- 82
HP	Read group position	♦		♦	3- 83
HQ	Wait for group command buffer level	♦	♦	♦	3- 84
HS	Stop group motion	♦	♦	♦	3- 85
HV	Set group velocity	♦	♦	♦	3- 86
HW	Wait for group motion stop	♦	♦	♦	3- 87
HX	Delete group	♦	♦	♦	3- 89
HZ	Read group size	♦	♦	♦	3- 90
ID	Read stage model and serial number	♦		♦	3- 91
JH	Set jog high speed	♦	♦	♦	3- 92
JK	Set jerk rate	♦	♦	♦	3- 93
JL	Jump to label		♦	♦	3- 94
JW	Set jog low speed	♦	♦	♦	3- 95
KD	Set derivative gain	♦	♦	♦	3- 96
KI	Set integral gain	♦	♦	♦	3- 97
KP	Set proportional gain	♦	♦	♦	3- 98
KS	Set saturation level of integral factor	♦	♦	♦	3- 99
LP	List program	♦		♦	3- 100
MD	Read motion done status	♦		♦	3- 101
MF	Motor power off	♦	♦	♦	3- 102
MO	Motor power on	♦	♦	♦	3- 103
MT	Move to hardware travel limit	♦	♦	♦	3- 104
MV	Move indefinitely	♦	♦	♦	3- 105
MZ	Move to nearest index	♦	♦	♦	3- 107
OH	Set home search high speed	♦	♦	♦	3- 108
OL	Set home search low speed	♦	♦	♦	3- 109
OM	Set home search mode	♦	♦	♦	3- 110

TABLE 3.5.2 – Command List – Alphabetical (Continued)

In a PDF format you may click on a page number to automatically be connected to the corresponding Command Page

Cmd.	Description	IMM	PGM	MIP	Page
OR	Search for home	♦	♦		3- 111
PA	Move to absolute position	♦	♦	♦	3- 113
PC	Set position compare mode	♦	♦	♦	3- 114
PH	Get hardware status	♦		♦	3- 117
PR	Move to relative position	♦	♦	♦	3- 120
QD	Update motor driver settings	♦	♦		3- 121
QG	Set gear constant	♦	♦		3- 122
QI	Set maximum motor current	♦	♦		3- 123
QM	Set motor type	♦	♦		3- 124
QP	Quit program mode	♦			3- 125
QR	Reduce motor torque	♦	♦	♦	3- 126
QS	Set microstep factor	♦	♦		3- 127
QT	Set tachometer gain	♦	♦		3- 128
QV	Set average motor voltage	♦	♦		3- 129
RA	Read analog input	♦		♦	3- 130
RQ	Generate service request	♦	♦	♦	3- 131
RS	Reset the controller	♦		♦	3- 132
SA	Set device address	♦	♦	♦	3- 134
SB	Set/get DIO port A, B, C bit status	♦	♦	♦	3- 135
SH	Set home preset position	♦	♦	♦	3- 137
SI	Set master-slave jog velocity update interval	♦	♦	♦	3- 138
SK	Set master-slave jog velocity scaling coefficients	♦	♦	♦	3- 139
SL	Set level travel limit	♦	♦	♦	3- 140
SM	Save settings to non-volatile memory	♦			3- 141
SN	Set axis displacement units	♦	♦		3- 142
SR	Set right travel limit	♦	♦		3- 143
SS	Define master-slave relationship	♦	♦		3- 144
ST	Stop motion	♦	♦	♦	3- 145
SU	Set encoder resolution	♦	♦		3- 146
TB	Read error message	♦		♦	3- 147
TE	Read error code	♦		♦	3- 148
TJ	Set trajectory mode	♦	♦		3- 149
TP	Read actual position	♦		♦	3- 150
TS	Get controller status	♦		♦	3- 151
TV	Get actual velocity	♦		♦	3- 152
TX	Get controller activity	♦		♦	3- 153
UF	Update servo filter	♦	♦	♦	3- 154
UH	Wait for DIO bit high		♦		3- 155

TABLE 3.5.2 – Command List – Alphabetical (Continued)

In a PDF format you may click on a page number to automatically be connected to the corresponding Command Page

Cmd.	Description	IMM	PGM	MIP	Page
UL	Wait for DIO bit low		♦		3-156
VA	Set velocity	♦	♦	♦	3-157
VB	Set base velocity for step motors	♦	♦	♦	3-158
VE	Read controller firmware version	♦		♦	3-159
VF	Set velocity feed-forward gain	♦	♦	♦	3-160
VU	Set maximum velocity	♦	♦	♦	3-161
WP	Wait for absolute position crossing	♦	♦	♦	3-162
WS	Wait for motion stop	♦	♦	♦	3-163
WT	Wait	♦	♦	♦	3-164
XM	Get available program memory	♦		♦	3-165
XX	Delete a stored program	♦		♦	3-166
ZA	Set amplifier I/O configuration	♦	♦		3-167
ZB	Set feedback configuration	♦	♦		3-170
ZE	Set E-stop configuration	♦	♦		3-172
ZF	Set following error configuration	♦	♦		3-174
ZH	Set hardware limit configuration	♦	♦		3-176
ZS	Set software limit configuration	♦	♦		3-178
ZU	Get ESP system configuration	♦		♦	3-180
ZZ	Set system configuration	♦	♦		3-183

3.6 Description of Commands

The extensive ESP controller command set exists to facilitate application development for wide range of application and needs. However, most simple positioning can be done with just a few commands:

VA – set velocity
AC – set acceleration
AG – set deceleration
PR – position relative
PA – position absolute
TP – tell position
WS – wait for stop

NOTE

Most of the commands take an axis number as a parameter (xx). For such commands, the valid range of axis number is from 1 to MAX AXES, where MAX AXES is dependant on the configuration of the ESP motion controller.

Commands related to coordinated motion and contouring (group commands) take a group number as a parameter. For such commands, the valid range of group number is from 1 to MAX GROUPS, where MAX GROUPS is one-half the MAX AXES.

AA – (command mnemonic) **(brief definition) (motor type) ***

(the squares mark which mode the command can be used in)

USAGE ■ **IMM** ■ **PGM** ■ **MIP**

(modes of operation)

SYNTAX xxAAnn (generic syntax format)

PARAMETERS

Description	**xx** [int]	--	(description of parameter)
	Nn [float]	--	(description of parameter)

(parameter could be **int**eger number, **float**ing point number, **char**acter or **str**ing)

Range	**xx**	--	**(minimum value** to **maximum value)**
	nn	--	**(minimum value** to **maximum value)**
Units	**xx**	--	(units description)
	nn	--	(units description)

Defaults

xx missing: (default or error if parameter is missing)
out of range: (default or error if parameter is out of range)

nn missing: (default or error if parameter is missing)
out of range: (default or error if parameter is out of range)

DESCRIPTION (detailed description of the command)

Note:
(notes, reminders and things to consider when using the command, if any)

RETURNS **(Type,** format and description of the return the command is Generating, if any)

ERRORS (Error Code) – (description of errors the command could Generate if misused)

REL. COMMANDS (brief definition of related commands)

EXAMPLE (Command Discussed) | *(description)*
(Other command) | *(description)*
(Controller return) | *(description)*

***(motor type)** – if the command is specific for a motor type (DC or stepping) it will be labeled here, otherwise this field is blank,

** The mode mnemonics has the following meanings:

IMMediate mode – controller is in idle mode and the commands are executed immediately.

Pro**G**ra**M** mode – controller does not execute but stores all commands as part of a program.

Motion **I**n **P**rogress – controller is executing a motion on all or the specified axis.

AB abort motion

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **AB**

PARAMETERS None.

DESCRIPTION This command should be used as an emergency stop. On reception of this command, the controller invokes emergency stop event processing for each axis as configured by **ZE** (e-stop event configuration) command.

By default axes are configured to turn motor power OFF, however, individual axes can be configured to stop using emergency deceleration rate set by **AE** command and maintain motor power.
It should be used only as an immediate command, not in a program.

Note
This command affects all axes, however the action taken is determined by each individual's axis **ZE** command configuration.

RETURNS none

REL. COMMANDS

ST	-	stop motion
AE	-	e-stop deceleration
ZE	-	e-stop deceleration
MF	-	motor OFF
MO	-	motor ON

EXAMPLE **AB** | used as an immediate command to stop all motion

AC set acceleration

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**AC**nn or xx**AC**?

PARAMETERS

Description			
	xx [int]	-	axis number
	nn [float]	-	acceleration value
Range	**xx**	-	**1** to **MAX AXES**
	N	-	**0** to the maximum programmed value in **AU** command or **?** to read current setting
Units	**xx**	-	none
	nn	-	predefined units / second^2
Defaults	**xx**	missing:	error 37, AXIS NUMBER MISSING
		out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error xx11, MAXIMUM ACCELERATION EXCEEDED

DESCRIPTION

This command is used to set the acceleration value for an axis. Its execution is immediate, meaning that the acceleration is changed when the command is processed, even while a motion is in progress.

It can be used as an immediate command or inside a program. If the requested axis is a member of a group, the commanded acceleration becomes effective only after the axis is removed from the group. (Refer to Advanced Capabilities section for a detailed description of grouping and related commands)

Avoid changing the acceleration during the acceleration or deceleration periods. For better predictable results, change acceleration only when the axis is not moving or when it is moving with a constant speed.

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

VA	-	set velocity
PA	-	execute an absolute motion
PR	-	execute a relative motion
AU	-	set maximum acceleration and deceleration
AG	-	set deceleration

EXAMPLE

2AU? | *read maximum allowed acceleration/deceleration of axis # 2*

10	*controller returns a value of 10 units/s^2*
2AC9	*set acceleration to 9 units/s^2*
2AG6	*set deceleration to 6 units/s^2*
2AU15	*set axis # 2 maximum acceleration/deceleration to 15 units/s^2*
2AU?	*read maximum allowed acceleration & deceleration of axis # 2*
15	*controller returns a value of 15 units*

AE set e-stop deceleration

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**AE**nn or xx**AE**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	e-stop deceleration value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	current normal deceleration value to **2e9 * encoder resolution** or **?** to read current setting
Units	**xx**	-	none
	nn	-	predefined units / second^2

Defaults	**xx**	missing:	error 37, AXIS NUMBER MISSING
		out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error 1, PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to set the e-stop deceleration value for an axis. Its execution is immediate, meaning that the e-stop deceleration value is changed when the command is processed, even while a motion is in progress.

It can be used as an immediate command or inside a program. If the requested axis is a member of a group, the commanded e-stop deceleration becomes effective only after the axis is removed from the group. (Refer to Advanced Capabilities section for a detailed description of grouping and related commands)

E-stop deceleration is invoked upon a local e-stop condition (e.g., front panel Stop All pushbutton, Interlock, etc..) has occurred, if configured to do so, or if the AB (abort motion) command is processed.

Note:
E-stop deceleration value cannot be set lower than the normal deceleration value. Refer the description of "**AG**" command for range of deceleration values.

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

VA	-	set velocity
PA	-	execute an absolute motion
PR	-	execute a relative motion

	AU -	set maximum acceleration and deceleration
	AG -	set deceleration
	AC -	set acceleration

EXAMPLE	2AE?	| *read e-stop deceleration of axis # 2*
	100	| *controller returns a value of 100 units/s^2*
	2AE150	| *set e-stop deceleration to 150 units/s^2*

AF set acceleration feed-forward gain

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**AF**nn or xx**AF**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	acceleration feed-forward gain factor
Range	**xx**	-	**1** to **MAX AXES**
	nn-	-	**0** to **2e9,** or **?** to read current setting
Units	**xx**	-	none
	nn	-	none

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command sets the acceleration feed-forward gain factor Af. It is active for any DC servo based motion device.

.

See the "**Feed-Forward Loops**" in Motion Control Tutorial section to understand the basic principles of feed-forward.

Note:
The command can be sent at any time but it has no effect until the UF (update filter) is received.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

KI	-	set integral gain factor
KD	-	set derivative gain factor
KP	-	set proportional gain factor
KS	-	set saturation gain factor
VF	-	set velocity feed-forward gain
UF	-	update filter

EXAMPLE

3VF1.5	*| set acceleration feed-forward gain factor for axis # 3 to 1.5*
3AF?	*| report present axis-3 acceleration feedforward setting*
0.9	*| controller returns a value of 0.9*
3AF0.8	*| set acceleration feed-forward gain factor for axis # 3 to 0.8*
3UF	*| update PID filter; only now the **AF** command takes effect*

AG set deceleration

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**AG**nn or xx**AG**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	acceleration value
Range	**xx**	-	**1** to **MAX AXES**
	Nn	-	to the maximum programmed value in **AU** command or **?** to read current setting
Units	**xx**	-	none
	Nn	-	predefined units / second^2
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx11, MAXIMUM ACCELERATION EXCEEDED

DESCRIPTION

This command is used to set the deceleration value for an axis. Its execution is immediate, meaning that the deceleration is changed when the command is processed, even while a motion is in progress.

It can be used as an immediate command or inside a program. If the requested axis is a member of a group, the commanded deceleration becomes effective only after the axis is removed from the group. (Refer to Advanced Capabilities section for a detailed description of grouping and related commands)

Avoid changing the deceleration during the acceleration or deceleration periods. For better predictable results, change deceleration only when the axis is not moving or when it is moving with a constant speed.

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

VA	-	set velocity
PA	-	execute an absolute motion
PR	-	xecute a relative motion
AU	-	set maximum acceleration and deceleration
AC	-	set acceleration

EXAMPLE	2AU?	*read maximum allowed acceleration/deceleration of axis # 2*
	10	*controller returns a value of 10 units/s^2*
	2AC9	*set acceleration to 9 units/s^2*
	2AG6	*set deceleration to 6 units/s^2*
	2AG?	*read maximum current deceleration of axis # 2*
	6	*controller returns a value of 6 units/s^2*

AM set analog input mode

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **AM**nn or **AM**?

PARAMETERS

Range	**nn**	-	**0** to **7,** or **?** to read current setting
Units	**nn**	-	none

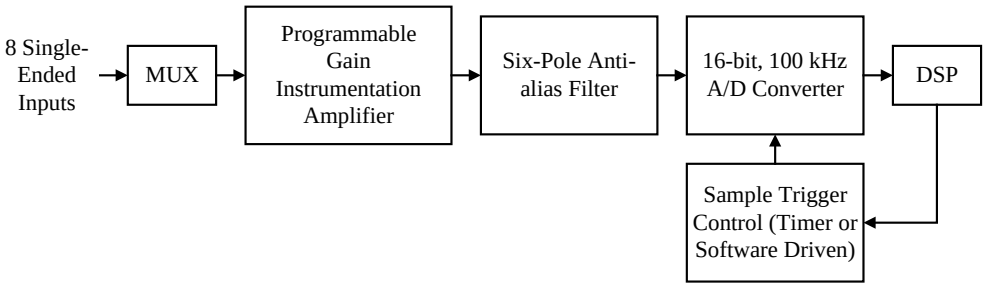
Defaults

nn	missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error 7, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to set all eight (8) ADC channels to any one of eight (8) analog input modes listed below.

- **nn** = 0: ±10 V input range
- **nn** = 1: ±5 V input range
- **nn** = 2: 0—10 V input range
- **nn** = 3: 0—5 V input range
- **nn** = 4: ±2.50 V input range
- **nn** = 5: ±1.25 V input range
- **nn** = 6: 0—2.50 V input range
- **nn** = 7: 0—1.25 V input range

ADC channels are located on the analog I/O connector on the controller card. The following block diagram illustrates the implementation of analog-to-digital conversion in ESP controller.



RETURNS If the "?" sign takes the place of **nn** value, this command reports the current analog input mode

REL. COMMANDS **RA** - read analog input

EXAMPLE

AM2	*| set 0 to 10 V analog range for all the ADC channels*
AM?	*| request the actual analog input mode*
2	*| controller returns a value of 2*

AP abort program

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **AP**

PARAMETERS none

DESCRIPTION This command is used to interrupt a motion program in execution. It will not stop a motion in progress. It will only stop the program after the current command line finished executing.

It can be used as an immediate command or inside a program.

Inside a program it is useful in conjunction with program flow control commands. It could, for instance, terminate a program on the occurrence of a certain external event, monitored by an I/O bit.

RETURNS none

REL. COMMANDS **EX** - execute a program

EXAMPLE

```
3EX          | execute program # 3
  .
  .
  .
AP           | stop program execution
```

AU set maximum acceleration and deceleration

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**AU**nn or xx**AU**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	acceleration value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **2e+9,** or **?** to read current setting
Units	**xx**	-	none
	nn	-	predefined units / second^2
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx11, MAXIMUM ACCELERATION EXCEEDED
			error xx1, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to set the maximum acceleration and deceleration value for an axis. This command remains effective even if the requested axis is member of a group. In this case, two error messages "GROUP MAXIMUM ACCELERATION EXCEEDED" or "GROUP MAXIMUM DECELERATION EXCEEDED" are generated if the commanded value is less than group acceleration or deceleration respectively. (Refer to Advanced Capabilities section for a detailed description of grouping and related commands)

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

VA	-	set velocity
PA	-	execute an absolute motion
PR	-	execute a relative motion
AG	-	set deceleration
AC	-	et acceleration

EXAMPLE

AU?	| *read maximum allowed acceleration/deceleration of axis # 2*
10	| *controller returns a value of 10 units/s^2*
2AC9	| *set acceleration to 9 units/s^2*
2AG6	| *set deceleration to 6 units/s^2*
2AU15	| *set axis # 2 maximum acceleration/deceleration to 15 units/s^2*
2AU?	| *read maximum allowed acceleration & deceleration of axis # 2*
15	| *controller returns a value of 15 units/s^2*

BA set backlash compensation

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**BA**nn or xx**BA**?

PARAMETERS

Description			
	xx [int]	-	axis number
	nn [float]	-	backlash compensation value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	to **distance equivalent to 10000 encoder counts**
Units	**xx**	-	none
	nn	-	user units

Defaults			
	xx	missing:	error 37, AXIS NUMBER MISSING
		out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error 7, PARAMETER OUT OF RANGE

DESCRIPTION This command initiates a backlash compensation algorithm when motion direction is reversed. The controller keeps track of the motion sequence and for each direction change it adds the specified **nn** correction. Setting **nn** to zero disables the backlash compensation.

NOTE: The command is affective only after a home search (**OR**) or define home (**DH**) is performed on the specified axis.

RETURNS If "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS None

EXAMPLE

1BA0.0012	|	*Set backlash compensation value for axis #1 to 0.0012 units*
1BA?	|	*Query backlash compensation value for axis #1*
0.0012	|	*Controller returns a value of 0.0012 units*
1OR	|	*Perform home search on axis #1*
1PA10	|	*Move axis #1 to absolute 10 units*
1PA0	|	*Move axis #1 to absolute 0 units*

BG assign DIO bits to execute stored programs

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **xxBGnn or xxBG?**

PARAMETERS

Description

xx [int]	-	bit number used to trigger stored program execution
nn [char]	-	name of stored program to be executed

Range

xx	-	**0** to **23**
nn	-	None or **?** to read current setting

Units None

Defaults

xx	missing:	error 7, PARAMETER OUT OF RANGE
	out of range:	error 7, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to assign DIO bits for initiating the execution of a desired stored program. Execution of the stored program begins when the specified DIO bit changes its state from HIGH to LOW logic level.

Note: Each DIO bit has a pulled-up resistor to +5V. Therefore, all bits will be at HIGH logic level if not connected to external circuit and configured as input.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS

BO	-	Set DIO port A, B, C direction
EP	-	Enter program mode
EX	-	Execute stored program
AP	-	Abort stored program execution

EXAMPLE

```
BO 04H      | Set DIO ports A and B to input and port C to output
0 BG 1      | Start execution of a stored program 1 when DIO bit #0 changes
            | state from HIGH to LOW
```

BK assign DIO bits to inhibit motion

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **xxBKnn1, nn2 or xxBK?**

PARAMETERS

Description

xx [int]	-	axis number
nn1 [int]	-	bit number for inhibiting motion
nn2 [int]	-	bit level when axis motion is inhibited

Range

xx	-	**1** to **MAX AXES**
nn1	-	**0** to **23**
nn2	-	**0 = LOW** and **1 = HIGH** or **?** to read current setting

Units None

Defaults

xx	missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
nn1	missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx1, PARAMETER OUT OF RANGE
nn2	missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx1, PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to assign DIO bits for inhibiting the motion of a selected axis. If the selected axis is already in motion, and DIO bit is asserted, e-stop is executed per E-stop configuration (Refer "**ZE**" command for further details). If the axis is not moving, any new move commands are refused as long as the DIO bit is asserted. In either case, "DIGITAL I/O INTERLOCK DETECTED" error is generated.

Note: The direction of the DIO port (A, B or C) the desired bit belongs to, should be set to "input" in order for the DIO bit to be read accurately. Refer "**BO**" command for further details.

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current assignment.

REL. COMMANDS

BL	-	Enable DIO bits to inhibit motion
BO	-	Set DIO port A, B, C direction
BM	-	Assign DIO bits to notify motion status

EXAMPLE

BO 04H	| *Set DIO ports A, B to input and C to output*
2BK 1, 1	| *Use DIO bit #1 to inhibit motion of axis #2. This DIO bit should be*
	| *HIGH when axis #2 motion is inhibited*
2BL 1	| *Enable inhibition of motion using DIO bits for axis #2*
2BK?	| *Query the DIO bit assignment for axis #2*
1, 1	| *The controller responds with the assigned values*

BL enable DIO bits to inhibit motion

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **xxBLnn or xxBL?**

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	disable or enable
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** = disable, and **1** = enable or **?** to read current setting
Units	None		
Defaults	**xx**	missing:	error 37, AXIS NUMBER MISSING
		out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error xx1, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to disable or enable motion inhibition of requested axes through DIO bits.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current status.

REL. COMMANDS

BK	-	Assign DIO bits to inhibit motion.
BO	-	Set DIO port A, B, C direction
BM	-	Assign DIO bits to notify motion status
BN	-	Enable DIO bits to notify motion status.

EXAMPLE

BO 04H	| *Set DIO ports A and B to input and port C to output*
2BK 1, 1	| *Use DIO bit #1 to inhibit motion of axis #2. This DIO bit should be*
	| *HIGH when axis #2 motion is inhibited*
2BL 1	| *Enable inhibition of motion using DIO bits for axis #2*
2BK?	| *Query the DIO bit assignment for axis #2*
1, 1	| *The controller responds with the assigned values*
2BL?	| *Query the status of inhibiting motion for axis #2 through DIO*
1	| *The controller responds with 1 indicating feature is enabled*

BM assign DIO bits to notify motion status

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **xxBMnn1, nn2 or xxBM?**

PARAMETERS

Description

xx [int]	-	axis number
nn1 [int]	-	bit number for notifying motion status
nn2 [int]	-	bit level when axis is not moving

Range

xx	-	**1** to **MAX AXES**
nn1	-	**0** to **23**
nn2	-	**0 = LOW** and **1 = HIGH** or **?** to read current setting

Units None

Defaults

xx	missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
nn1	missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx1, PARAMETER OUT OF RANGE
nn2	missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx1, PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to assign DIO bits for notifying the motion status – moving or not moving – of a selected axis. When the selected axis is not moving, the DIO bit state changes to the level specified with this command (refer parameter **nn2**).

NOTE: The direction of the DIO port (A, B or C) the desired bit belongs to, should be set to "output" in order for the DIO bit to be set accurately. Refer "**BO**" command for further details.

NOTE: If a motion feature, such as origin search, involves a sequence of moves, the motion status will be set to not moving only after the entire sequence of moves has completed.

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current assignment.

REL. COMMANDS

BN	-	Enable DIO bits to notify motion status
BO	-	Set DIO port A, B, C direction

EXAMPLE

BO 06H	*| Set DIO port A to input and B, C to output*
2BM 9, 1	*| Use DIO bit #9 to indicate motion status of axis #2. This DIO bit*
	| should be HIGH when axis #2 is not moving
2BN 1	*| Enable notification of motion using DIO bits for axis #2*
2BM?	*| Query the DIO bit assignment for axis #2*
9, 1	*| The controller responds with the assigned values*

BN enable DIO bits to notify motion status

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **xxBNnn or xxBN?**

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	disable or enable
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** = disable, and **1** = enable or **?** to read current setting
Units	None		

Defaults	**xx**	missing:	error 37, AXIS NUMBER MISSING
		out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error xx1, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to disable or enable notification of requested axis' motion status through DIO bits.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current status.

REL. COMMANDS	**BM**	-	Assign DIO bits to notify motion status
	BO	-	Set DIO port A, B, C direction
	BK	-	Assign DIO bits to inhibit motion
	BL	-	Enable DIO bits to inhibit motion

EXAMPLE

BO 06H	*| Set DIO port A to input and ports B, C to output*
2BM 9, 1	*| Use DIO bit #9 to indicate motion status of axis #2. This DIO bit*
	| should be HIGH when axis #2 is not moving
2BN 1	*| Enable notification of motion using DIO bits for axis #2*
2BM?	*| Query the DIO bit assignment for axis #2*
9, 1	*| The controller responds with the assigned values*
2BN?	*| Query the status of notifying motion status of axis #2 through DIO bits*
1	*| The controller responds with 1 indicating feature is enabled*

BO set DIO port A, B, C direction

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **BO**nn or **BO**?

PARAMETERS

Description	**nn** [int]	-	hardware limit configuration
Range	**nn**	-	**0** to **07H** (hexadecimal with leading zero(0)) on ESP6000 and ESP7000
		-	**0 to 05H (hexadecimal with leading zero(0)) on ESP100 and ESP300** or **?** to read current setting
Units	**nn**	-	None
Defaults	**nn**	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error 7, PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to set digital I/O (DIO) port A, B, and C direction where bit-0 corresponds to port A, bit-1 to port B, and bit-2 to port C. If any bit is set to zero(0) then its corresponding port will become an input only. If any bit is set to one(1) then its corresponding port will becomes an output only. Ports A and B only are available on the ESP100 and ESP300.

A DIO within a port configured as an input can only report its present HIGH or LOW logic level. Whereas a DIO bit within a port configured as an output can set(1) or clear(0) the corresponding DIO hardware to HIGH or LOW logic level. Reading the status of a port configured as output returns its present output status.

NOTE: All direction bits are automatically zeroed, or cleared, after a system reset. Therefore all DIO ports default to input by default.

NOTE: Each DIO bit has a pulled-up resistor to +5V. Therefore, all bits will be at HIGH logic level if not connected to external circuit and configured as input.

BIT#	VALUE	DEFINITION
*0	0	port A (DIO bit-0 through bit-7) assigned as input
0	1	port A (DIO bit-0 through bit-7) assigned as output
*1	0	port B (DIO bit-8 through bit-15) assigned as input
1	1	port B (DIO bit-8 through bit-15) assigned as output
*2	0	port C+ (DIO bit-16 through bit-23) assigned as input
2	1	port C+ (DIO bit-16 through bit-23) assigned as output
		* default setting after system reset
		+ if available on the controller

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting in hexadecimal notation.

REL. COMMANDS **SB -** set/clear DIO bits

EXAMPLE BO?

	| *read DIO port direction configuration*
0H	| *controller returns a value of 0H (all ports are input)*
BO 1H	| configure *DIO port A as output*
SB 0FFH	| *set all port A DIO output HIGH*

BP assign DIO bits for jog mode

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **xxBPnn1,nn2 or xxBP?**

PARAMETERS

Description

xx [int]	-	axis number
nn1 [int]	-	bit number for jogging in negative direction
nn2 [int]	-	bit number for jogging in positive direction

Range

xx	-	**1** to **MAX AXES**
nn$_i$	-	**0** to **23**

Units

xx	-	none
nn$_i$	-	none

Defaults

xx	missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
nn	missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to assign DIO bits for jogging axes in either negative or positive directions.

RETURNS If "?" sign is issued along with command, the controller returns the DIO bits used for jogging in negative and positive directions respectively.

REL. COMMANDS

BO	-	enable usage of DIO bits for jogging axes

EXAMPLE

1BP3, 4	| *set DIO bit #3 to jog axis #1 in negative direction and DIO bit #4 to jog axis #1 in positive direction*
1BP?	| *query the DIO bits assigned for jogging*
3,4	| *controller returns the bit assignment*
1BQ1	| *enable axis #1 jogging through DIO bits.*

BQ enable DIO bits for jog mode

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **xxBQ**nn or **BQ**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	disable or enable
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** = disable, and **1** = enable
Units	**xx**	-	none
	nn	-	one

Defaults	**xx**	missing:	error 37, AXIS NUMBER MISSING
		out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to disable or enable jogging of a requested axis through DIO bits.

RETURNS If "?" sign is issued along with command, the controller returns the status of jog through DIO bits.

REL. COMMANDS **BP** - assign DIO bits for jog mode

EXAMPLE

1BP3,4	*| set DIO bit #3 to jog axis #1 in negative direction and*
	| DIO bit #4 to jog axis #1 in positive direction
1BP?	*| query the DIO bits assigned for jogging*
3,4	***|*** *controller returns the bit assignment*
1BQ1	*| enable axis #1 jogging through DIO bits.*

BR set serial communication speed

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **BRnn or BR?**

PARAMETERS

Description	**nn** [int]	-	communication speed over serial interface
Range	**nn**	-	9600, 19200, 38400, 57600 or 115200 or **?** to read the current setting
Units	**nn**	-	bits per second (bps)
Defaults	**nn**	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error 7, PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to set the communication speed between a remote computer and the motion controller over the serial (RS-232) interface. Different speeds supported by the ESP motion controllers include: 9600, 19200, 38400, 57600 and 115200 bps. The default factory setting is 19200 bps.

The desired serial communication speed may be saved to non-volatile flash memory by issuing the ASCII command, **SM.** This will cause the DSP to automatically use the saved value after system reset or reboot.

NOTE: This command takes affect immediately after it is processed by the DSP. As a result, users are reminded that they cannot communicate with the controller (following a change in communication speed using this command) until they reinitialize the serial port on the remote computer with the same speed. Please see the example below.

NOTE: The software utilities, ESP-Download and ESP-Tuning, supplied with all ESP motion controllers (except ESP6000) support only 19200 bps communication speed. ESP-Terminal, however, supports all speeds.

CAUTION: ESP motion controllers do not do any form of error checking – CRC (Cyclic Redundancy Check) or Checksum – on the data that is being transmitted across the serial interface. As a result, it is highly recommended that the users select the communication speed judiciously based on the length of the serial cable, noise level of the work environment etc.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS None

EXAMPLE

BR? | *Query the serial communication speed*
19200 | *The controller responds with 19200 bps*
BR115200 | *Change the serial communication speed to 115200 bps*
Reinitialize the serial communication port on remote computer with 115200 bps
BR? | *Query the serial communication speed*
115200 | *The controller responds with 115200 bps*

CL set closed loop update interval

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **xxCLnn or xxCL?**

PARAMETERS

Description	**xx** { int]	-	axis number
	nn [int]	-	closed loop update interval
Range	**xx**	-	**0** to **MAX AXES**
	nn	-	**0** to **60000**
Units	**xx**	-	none
	nn	-	milliseconds
Defaults	**xx**	missing:	error 37, AXIS NUMBER MISSING
		out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error 7, PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to set the closed loop update interval for an axis. This will be the time duration between position error corrections during closed loop stepper positioning. Note that this command is effective only for steeper motors.

Furthermore, note that encoder feedback and closed loop positioning must be enabled for this command to be effective. Refer to feedback configuration (**ZB**) command for enabling these features in the case of stepper motors.

If "0" is used as an axis number, this command will set the specified interval to all the axes.

RETURNS

If "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS

ZB	-	set feedback configuration
DB	-	set position deadband value

EXAMPLE

3ZB300	| *enable encoder feedback and closed loop positioning of axis #3*
3DB1	| *set position deadband value to 1 encoder count*
3DB?	| *query deadband value*
1	| *controller returns a value of 1 encoder count*
3CL100	| *set closed loop update interval to 100 milliseconds*
3CL?	| *query closed loop update interval*
100	| *controller returns a value of 100 milliseconds*

CO set linear compensation

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**CO**nn or xx**CO**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	linear compensation value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **2e+9**
Units	**xx**	-	none
	nn	-	none

Defaults

xx missing: error 37, AXIS NUMBER MISSING
out of range: error 9, AXIS NUMBER OUT OF RANGE

nn missing: error 38, COMMAND PARAMETER MISSING
out of range: error 7, PARAMETER OUT OF RANGE

DESCRIPTION

This command allows users to compensate for linear positioning errors due to stage inaccuracies. Such errors decrease or increase actual motion linearly over the travel range.

The linear compensation value, **nn** is calculated according to the formula given below:

$$nn = \left(\frac{error}{travel} \right)$$

where,

travel = measured travel range
error = error accumulated over the measured travel range

NOTE: The command is affective only after a home search (**OR**) or define home (**DH**) is performed on the specified axis.

RETURNS

If "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS

None

EXAMPLE

If a stage has a travel range of 100 mm and it accumulates an error of 0.003 mm over the complete travel range,

$$nn = \left(\frac{0.003}{100} \right) = 0.00003$$

1CO0.00003	|	*Set linear compensation value for axis #1 to 0.00003*
1CO?	|	*Query linear compensation value for axis #1*
0.00003	|	*Controller returns a value of 0.00003*
1OR	|	*Perform home search on axis #1*
1PA10	|	*Move axis #1 to absolute 10 units*

DB set position deadband

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**DB**nn or xx**DB**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	deadband value
Range	**xx**	-	**0** to **MAX AXES**
	nn	-	to **2e9**
Units	**xx**	-	none
	nn	-	encoder counts
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING

DESCRIPTION This command is used to set the position deadband value for an axis. Since a majority of electro-mechanical systems have mechanical backlash or frictional hysterisis, closed-loop positioning can at times lead to oscillation or limit cycling of the systems around a desired position. In such situations, setting position deadband value judiciously can avoid limit cycling of the systems.

Note that this command is effective only during position regulation (holding position) as opposed to moving.

Furthermore, note that encoder feedback and closed loop positioning must be enabled for this command to be effective. Refer to feedback configuration (**ZB**) command for enabling these features in the case of stepper motors.

If "0" is used as an axis number, this command will set the specified deadband value to all the axes.

RETURNS If "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS

ZB	-	set feedback configuration
CL	-	set closed loop update interval

EXAMPLE

ZB300	|	*enable encoder feedback and closed loop positioning of axis#3*
3DB1	|	*set position deadband value to 1 encoder count*
3DB?	|	*query deadband value*
1	|	*controller returns a value of 1 encoder count*
3CL100	|	*set closed loop update interval to 100 milliseconds*
3CL?	|	*query closed loop update interval*
100	|	*controller returns a value of 100 milliseconds*

DC setup data acquisition

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **DC**nn1,nn2,nn3,nn4,nn5,nn6

PARAMETERS
Description

nn1	[int]	-	data acquisition mode
nn2	[int]	-	axis used to trigger data acquisition
nn3	[int]	-	data acquisition parameter 3
nn4	[int]	-	data acquisition parameter 4
nn5	[int]	-	data acquisition rate
nn6	[int]	-	number of data samples to be acquired

Range

nn1 -
0: Start data acquisition immediately
1: Start data acquisition when trigger axis starts motion
2: Start data acquisition when trigger axis reaches slew speed
3: Capture analog and/or encoder data on an external trigger
4-9: Reserved for future analog data acquisition modes
10: Start trace variable data acquisition immediately
11-29: Reserved for future analog data acquisition modes
30: Start streaming a fixed amount of data immediately
31: Start streaming a fixed amount of data when trigger axis starts motion
32: Start streaming a fixed amount of data when trigger axis reaches slew speed
33: Start streaming a fixed amount of data on an external trigger
34-35: Reserved for future analog data acquisition modes
36: Start streaming a continuous amount of data immediately
37: Start streaming a continuous amount of data when trigger axis starts motion
38: Start streaming a continuous amount of data when trigger axis reaches slew speed
39: Start streaming a continuous amount of data on an external trigger

nn2	-	**1** to **MAX AXES**
nn3	-	Refer table below
nn4	-	Refer table below
nn5	-	**0** to **1000**
nn6	-	to **1000**

Units None

Defaults **nn** missing: error 38, COMMAND PARAMETER MISSING

out of range: error 7, PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to setup data acquisition—analog data acquisition (ADC) as well as acquisition of certain trace variables—using ESP motion controller.

PARAMETER nn1: Data acquisition modes 0—9 and 30—39 support different ways in which analog data can be collected. On the other hand, mode 10 may be used to acquire trace variable data.

PARAMETER nn2: Data acquisition—analog or trace variable—is triggered by the motion of an axis specified through this parameter. Exceptions to this requirement are in the case of data acquisition modes 0, 10, 30 and 36. For these cases enabling data acquisition is sufficient to start the data acquisition process. For all other modes, two conditions—enabling of data acquisition and any mode dependent conditions such as trigger axis reaching slew speed—must be met in order to start the data acquisition process.

This parameter is of no consequence for data acquisition modes 3, 33 and 39, wherein the data acquisition process is triggered by an external source.

PARAMETER nn3 and nn4: Since the criteria used for analog data acquisition and trace variable data acquisition are different, parameters #3 and #4 are interpreted differently based on the type of data being acquired. Please refer to the table below for interpreting these two parameters accurately.

PARAMETER nn5: The rate at which data is to be acquired is specified through this parameter. The rate specified is in multiples of the servo cycle rate. For example, a value of 0 implies data acquisition every servo cycle, a value of 1 implies every other servo cycle, and so on.

This parameter is of no consequence for data acquisition modes 3, 33 and 39, wherein the data is acquired every time the controller detects a trigger from an external source.

PARAMETER nn6: The number of samples of data to be acquired is specified through this parameter. Data acquisition process is considered to be "done" only after the number of samples specified by this parameter are acquired by the controller. The status of data acquisition process may be found by issuing ASCII command, **DD**. Once the data acquisition is done, ASCII command, **DG** may be used to collect the data from the controller.

In the case of data acquisition modes 3 and 33, a sample size greater than 1 implies that data will be acquired by the controller every time an external trigger occurs until the specified number of samples are acquired.

This parameter is of no consequence in the case of data acquisition modes 36-39.

NOTE: Data acquisition mode 3 (nn1 = 3) is supported only on ESP6000 and ESP7000 motion controllers.
Data acquisition modes 30—39 (nn1 = 30—39) are supported only on ESP6000 motion controller. For these modes, the streamed data has to be collected by sending binary commands—esp_get_daq_data( ) or esp_daq_data_to_file( )—to the controller through the ESP6000.DLL.

NOTE: The controller responds with a servo cycle tick count along with every data sample collected. This feature is independent of the type of data—analog or trace variable data—collected.

NOTE: The external source for triggering data acquisition process (modes 3, 33 and 39) should be connected to pin #20 on the auxiliary I/O connector. Please refer to the appendix on Connector Pin Assignments for further details.

<table>
<tr><th>Data Acquisition Mode</th><th>Parameter #3</th><th colspan="3">Parameter #4</th></tr>
<tr><td>0-9 and
30-39:
Analog data acquisition</td><td>Analog channels involved in acquisition. (These channels are located on the analog I/O connector of the controller card.) The desired channels to be acquired are specified using the following bit assignment:
Bit 0: Channel 1
Bit 1: Channel 2
. . .
Bit 7: Channel 8</td><td colspan="3">Position feedback (encoder) channels involved in acquisition (This data is available from 8-quadrature decoders.) The desired channels to be acquired are specified using the following bit assignment:
Bit 0: Channel 1
Bit 1: Channel 2
. . .
Bit 7: Channel 8</td></tr>
<tr><td>10: Trace variable data acquisition</td><td>This parameter is used to specify the format in which controller should send the trace variable data acquired. Various formats supported by the controller include:
0: Texas Instruments floating point format & trigger axis is not configured as a slave axis
1: Scaled integer format & trigger axis is not configured as a slave axis
2: Texas Instruments floating point format & trigger axis is configured as a slave axis
3: Scaled integer format & trigger axis is configured as a slave axis
NOTE:
Trace variables such as velocity and acceleration are processed by ESP motion controller in TI floating point format. Users can have access to this data in its “native” format by choosing option 0. If this option is chosen, users must convert the data obtained into a format that is supported by host system. Since some users may have difficulty processing data in this format, a suggested alternative is to choose scaled integer format, option 1. If integer format is chosen, the floating point number is multiplied by 1e6 and converted to an integer. This option may be selected judiciously because the validity of data obtained depends on the range of values.

When the trigger axis is configured as a slave axis, the desired position for the slave is dependent on the gear ratio (GR), trajectory mode(TJ) and the master axis’ desired position. If one wishes to tune this slave axis without reconfiguring it as its own master, options 2 and 3 may be used to control the source of the trace variable data (self or master).</td><td colspan="3">This parameter is used to identify the trace variables to be acquired. The trace variables available for acquisition are dependent on the type of motor that drives the axis specified.

NOTE:
At the present time, these variables are not user selectable.

NOTE:
Users are suggested to issue a value of 1 for consistency with modes 0-9.
<table>
<tr><th>Servo motor driven axes</th><th>Stepper motor driven axes</th><th>Commutated stepper motor driven axes</th></tr>
<tr><td>Error Integral</td><td>Duration between pulses</td><td>Control (DAC) output #2</td></tr>
<tr><td>Trajectory phase</td><td>Trajectory phase</td><td>Trajectory phase</td></tr>
<tr><td>Following error</td><td>Following error</td><td>Following error</td></tr>
<tr><td>Control (DAC) output #1</td><td>Counts/servo cycle</td><td>Control (DAC) output #1</td></tr>
<tr><td>Desired acceleration</td><td>Desired acceleration</td><td>Desired acceleration</td></tr>
<tr><td>Desired velocity</td><td>Desired velocity</td><td>Desired velocity</td></tr>
<tr><td>Desired position</td><td>Desired position</td><td>Desired position</td></tr>
<tr><td>Actual position</td><td>Actual position</td><td>Actual position</td></tr>
</table></td></tr>
</table>

RETURNS None.

REL. COMMANDS

AM - set analog input mode
DD - get data-acquisition done status
DE - enable / disable data-acquisition
DF - get data-acquisition status – number of samples collected
DG - get data-acquisition data

EXAMPLE

DC10,1,1,1,0,1000 | *Acquire trace variable data for axis 1, in scaled integer*
| *format. Collect 1000 samples, one sample / servo cycle*
DE1 | *Enable trace variable data acquisition*
DD | *Query data-acquisition done status*
1 = true, 0 = false.
If true,
DE0 | *Disable trace variable data acquisition*
DG | *Get data collected*

DD get data acquisition done status

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **DD**

PARAMETERS **none**

DESCRIPTION This command returns the status of a data acquisition request.

RETURNS **aa,** where: **aa** = **1** for True, **0** for False

REL. COMMANDS

DC	**-**	setup data acquisition request
DG	**-**	get acquired data
DF	**-**	data acquisition status, returns # of samples collected
DE	**-**	enable / disable data acquisition

EXAMPLE

```
DC10,1,1,1,0,1000     | Acquire trace variable data for axis 1, in scaled integer
                      | format.  Collect 1000 samples, one sample / servo cycle
DE1                   | Enable trace variable data acquisition
DD                    | Query data-acquisition done status
                  1 = true, 0 = false.
If true,
DE0                   | Disable trace variable data acquisition
DG                    | Get data collected
```

DE enable/disable data acquisition

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **DE**nn

PARAMETERS **nn**

Description **nn** [int] - True | False

Range **nn** - **1** for True, **0** for False

DESCRIPTION This command is used to enable / disable the data acquisition request.

Note: This command cannot be issued when:
1. An axis is being homed (refer ASCII command, **OR**).
2. An axis is being moved to a travel limit (refer ASCII command, **MT**).
3. An axis is being moved to an index (refer ASCII command, **MZ**).

RETURNS None

REL. COMMANDS

DC	-	setup data acquisition request
DG	-	get acquired data
DF	-	data acquisition status, returns # of samples collected
DD	-	data acquisition done status

EXAMPLE

DC10,1,1,1,0,1000	| *Acquire trace variable data for axis 1, in scaled integer format. Collect 1000 samples, one sample / servo cycle*
DE1	| *Enable trace variable data acquisition*
DD	| *Query data-acquisition done status*
	1 = true, 0 = false.
If true,	
DE0	| *Disable trace variable data acquisition*
DG	| *Get data collected*

DF get data acquisition sample count

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **DF**

PARAMETERS **none**

DESCRIPTION This command returns the number of a data acquisition collected to the point of this request.

RETURNS **aa,** where: **aa** = **number of samples**

REL. COMMANDS

DC	**-**	setup data acquisition request
DG	**-**	get acquired data
DD	**-**	data acquisition done status
DE	**-**	enable / disable data acquisition

EXAMPLE

```
DC10,1,1,1,0,1000      | Acquire trace variable data for axis 1, in scaled integer
                       | format.  Collect 1000 samples, one sample / servo
                       cycle
DE1                    | Enable trace variable data acquisition
DD                     | Query data-acquisition done status
                   1 = true, 0 = false.
If true,
DE0                    | Disable trace variable data acquisition
DG                     | Get data collected
```

DG get acquisition data

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **DG**

PARAMETERS None

DESCRIPTION This command is used to retrieve data acquired from a data acquisition request.

RETURNS This command returns byte wide binary data. Each four bytes represents one DSP 32 bit word. The number of bytes returned depends on the setup request. (See **DC** command).

REL. COMMANDS

DC - setup data acquisition request
DE - enable / disable data acquisition
DF - data acquisition status, returns # of samples collected
DD - data acquisition done status

EXAMPLE

DC10,1,1,1,0,1000 | *Acquire trace variable data for axis 1, in scaled integer format. Collect 1000 samples, one sample / servo cycle*
DE1 | *Enable trace variable data acquisition*
DD | *Query data-acquisition done status*
1 = true, 0 = false.
If true,
DE0 | *Disable trace variable data acquisition*
DG | *Get data collected*

DH define home

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**DH**nn

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	position value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to ± **2e+9**
Units	**xx**	-	none
	nn	-	predefined units
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to define current position, HOME position. This means that the current position will be preset to the value defined by parameter ‘nn’.

RETURNS If the “?” sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS **OR** - execute a home search cycle

EXAMPLE

3OR1 | *perform a home search on axis # 3*

.
.
.

3DH | *define current position on axis # 3 HOME as 0 units*

.
.
.

3DH 20.0 | *define current position on axis # 3 HOME as 20.0 units*

DL define label

	IMM	PGM	MIP
USAGE		♦	

SYNTAX xx**DL**

PARAMETERS

Description	**xx** [int]	-	label number
Range	**xx**	-	**1** to **100**
Units	**xx**	-	none
Default	**xx** missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION

This command defines a label inside a program. In combination with JL (jump to label) command, they offer significant program flow control.

The operation of the DL / JL command pair is similar to commands in other computer languages that allow conditional jumps (or GOTO's) to predefined labels in a program.

Note:
This command does not generate an error when not used inside a program. Since it can not do any harm, it is only ignored.

RETURNS none

REL. COMMANDS **JL** - jump to label

EXAMPLE

```
3XX        | clear program 3 from memory, if any
3EP        |  create program 3
1DL        |  define label 1
.
.
.
1JL 5      | jump to label 1 five(5) times
QP         | end entering program and quit programming mode
3EX        | run stored program number 3
```

DO set dac offset

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**DO**nn or xx**DO**?

PARAMETERS

Description	**xx** [int]	-	DAC channel number
	nn [float]	-	DAC offset value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**-10.0** to **10.0** or **?** to read the current setting
Units	**xx**	-	None
	nn	-	Volts
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx16, MAXIMUM DAC OFFSET EXCEEDED

DESCRIPTION

This command is used to set the DAC offset compensation for the specified DAC channel. In the case of ESP6000 and ESP7000 motion controllers, there is only one DAC channel associated with every axis: DAC channel #1 is associated with axis #1, DAC channel #2 with axis #2 etc. In the case of ESP100 and ESP300 motion controllers, however, there are two DAC channels associated with every axis: DAC channels 1 and 2 are associated with axis #1, DAC channels 3 and 4 with axis #2 etc.

In order for the DAC offset to take affect, this command must be followed by the ASCII command, **UF** (Update Filter). This offset may be saved to non-volatile flash memory by issuing the ASCII command, **SM**. This will cause the DSP to automatically use the saved value after system reset or reboot.

NOTE: DAC offset compensation is necessary on servo axes to prevent motor drift during motor off conditions.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS None

EXAMPLE 1DO0.05 | *Set the offset for DAC channel #1 to 0.05V*

1UF | ***Update the filter settings***

SM | *Save parameters to non-dvolatile flash memory*

DP read desired position

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**DP?**

PARAMETERS

Description	**xx** [int]	-	axis number
Range	**xx**	-	**1** to **MAX AXES**
Units	**xx**	-	none
Defaults	**xx**	missing:	error 37, AXIS NUMBER MISSING
		out of range:	error 9, AXIS NUMBER OUT OF RANGE

DESCRIPTION

This command is used to read the desired positionIt returns the instantaneous desired position.

The command could be sent at any time but its real use is while a motion is in progress.

RETURNS

nn where **nn** = desired position, in pre-defined units

REL. COMMANDS

PA	-	move to an absolute position
PR	-	move to a relative position
TP	-	read actual position

EXAMPLE

3TP?	|	*read position on axis # 3*
5.32	|	*controller returns position 5.32 for axis # 3*
3PR2.2	|	*start a relative motion of 2.2 on axis # 3*
3DP?	|	*read desired position on axis # 3*
7.52	|	*controller returns desired position 7.52 for axis # 3*

DV read desired velocity

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX xx**DV**

PARAMETERS

Description			
	xx [int]	-	axis number
Range	**xx**	-	**1** to **MAX AXES**
Units	**xx**	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE

DESCRIPTION This command is used to read the desired velocity of an axis. The command can be sent at any time but its real use is while motion is in progress.

RETURNS **nn,** where **nn** = desired velocity of the axis in pre-defined units.

REL. COMMANDS

PA	-	move to an absolute position
PR	-	move to a relative position

EXAMPLE

3TP?	|	*read position on axis # 3*
5.32	|	*controller returns position 5.32 units for axis # 3*
3PR2.2	|	*start a relative motion of 2.2 units on axis # 3*
3DV	|	*read desired velocity on axis #3*
0.2	|	*controller returns velocity 0.2 units/sec for axis #3*
3DP?	|	*read desired position on axis # 3*
7.52	|	*controller returns desired position 7.52 units for axis # 3*

EO automatic execution on power on

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX xx**EO**nn or xx**EO**?

PARAMETERS

Description	**xx** [int]	- program number
	nn [int]	- number of times of execution
Range	**xx**	- **1** to **100**
	nn	- **1** to **2e9**
Units	**xx**	- none
	nn	- none
Defaults	**None**	

DESCRIPTION This command sets the program number that is automatically executed on power on. If **nn** is missing, the **xx** numbered program is executed once.

RETURNS If the sign **"?"** takes place of **nn** value, this command reports the number of the program that is executed on power on and the number of times of execution.

REL. COMMANDS

QP	-	quit programming mode
EX	-	execute stored program
AP	-	abort stored program execution
XX	-	erase program

EXAMPLE

3EO	| *set program #3 to be executed once on power on*
EO?	| *query the program number executed on power on*
EO,3,1	| *controller returns program #3 executed once on power on*
EO	| *Reset automatic program execution – no program is executed on power on*

EP enter program mode

	IMM	PGM	MIP
USAGE	♦		

SYNTAX xx**EP**

PARAMETERS

Description	**xx** [int]	-	program number
Range	**xx**	-	**1** to **100**
Units	**xx**	-	none
Defaults	**xx** missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error 7, PARAMETER OUT OF RANGE

DESCRIPTION

This command sets the controller in programming mode. All the commands following this one will not be executed immediately but stored in memory as part of program number **xx**. To exit program entry mode and return to immediate mode, use QP command.
Programs can be entered in any order. If a program already exists then it must be first deleted using XX command.
Note:
Programs are automatically stored into non-volatile memory when created.

RETURNS none

REL. COMMANDS

QP	-	quit programming mode
EX	-	execute stored program
AP	-	abort stored program execution
XX	-	erase program

EXAMPLE

```
3XX          | clear program 3 from memory, if any
3EP          | activate program mode and enter following commands as
             | program 3
    .
    .
    .
QP           | end entering program and quit programming mode
3EX          | run stored program number 3
```

ES define event action command string

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **ES**nn or **ES**?

PARAMETERS

Description	**nn** [string]	-	ASCII command string (184 characters max)
Range	**nn**	-	Limited to existing (non-query) command support or **?** to read the current setting
Units	None		
Defaults	**nn** missing:		To clear command string and disable event action

DESCRIPTION

This command is used to define (and enable) or disable (if null string) an event action command string to be performed when an external trigger occurs.

This command is effective only if it is used in conjunction with ASCII command, **DC** with data acquisition mode set to either 3 or 33. If the number of samples to be collected is specified to be 1 using DC command, the event action is initiated immediately following the detection of an external trigger. However, if the number of samples is specified to be greater than 1, the event action is initiated only after all the external triggers—equal to the number of samples required to complete the data acquisition—are detected by the controller.

NOTE: The external source for triggering data acquisition process (modes 3, 33 and 39) should be connected to pin #20 on the auxiliary I/O connector. Please refer to the appendix on Connector Pin Assignments for further details.

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current ASCII command string for event trigger command processing.

REL. COMMANDS

DC	-	setup data acquisition
DD	-	get data-acquisition done status
DE	-	enable / disable data-acquisition
DF	-	get data-acquisition status – number of samples collected
DG	-	get data-acquisition data

EXAMPLE

```
DC3,1,3,5,0,1   | Param. #1: Acquire analog data on an external trigger
                | Param. #2: No consequence for this data acquisition mode
                | Param. #3: Acquire analog channels 1 & 2 (3₁₀ = (11)₂ )
                | Param. #4: Acquire feedback channels 1 & 3 (5₁₀ = (101)₂ )
                | Param. #5: No consequence for this data acquisition mode
                | Param. #6: Acquire one sample of data
ES 1MF          | Turn OFF motor #1 when the external event trigger occurs
DE1             | Enable analog data acquisition
DD              | Query data-acquisition done status
        1 = true, 0 = false.
If true,
DE0             | Disable trace variable data acquisition
DG              | Get data collected
```

EX execute a program

	IMM	PGM	MIPO
USAGE	♦	♦	

SYNTAX xx**EX**nn

PARAMETERS

Description	**xx** [int]	-	program number
	nn [int]	-	number of times to execute the program
Range	**xx**	-	**1** to **100**
	nn	-	**1** to **2147385345**
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error 7, PARAMETER OUT OF RANGE
	nn missing:		1 assumed
	out of range:		error 7, PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to start executing a program. When the command is received the controller executes the program line by line or according to the flow control instructions.

During program execution, only commands that ask for information and that stop the motion are still allowed. Any of the following commands will terminate a program, in one way or another: AB, AP, MF, RS and ST. Most natural way to just stop a program execution is by using the AP command, the other ones having a more drastic effect.

RETURNS none

REL. COMMANDS

QP	-	quit programming mode
EP	-	enter program mode
AP	-	abort stored program execution
XX	-	erase program

EXAMPLE

```
3XX          | clear program 3 from memory, if any
3EP          | activate program mode and enter following commands as
             | program 3
    .
    .
    .
QP           | end entering program and quit programming mode
3EX          | run stored program number 3
```

FE set maximum following error threshold

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**FE**nn or xx**FE**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	maximum allowed following error
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to**(2e9 * encoder resolution),** or **?** to read current setting
Units	**xx**	-	none
	nn	-	predefined units
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION

This command sets the maximum allowed following error threshold for an axis. This error is defined as the difference between the real position and the theoretical position of a motion device. The real position is the one reported by the position sensing device (encoder, scale, etc.) and the theoretical position is calculated by the controller each servo cycle. If , for any axes and any servo cycle, the following error exceeds the preset maximum allowed following error, the controller invokes the following error event handling process which is defined with the **ZF** command. By default motor power is turned OFF.

Note

Using the **ZF** command each axis can be individually configured to either turn motor power OFF, abort motion using e-stop deceleration, or ignore the error.

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS

ZF - set following error event configuration

EXAMPLE

3FE ?	*| read maximum following error for axis # 3*
0.5	*| controller returns for axis # 3 following error of 0.5 unit*
3FE 1.0	*| set maximum following error for axis # 3 to 1 unit*

FP set position display resolution

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**FP**nn or xx**FP**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	display resolution
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **7**
			or **?** to read present setting
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to set the display resolution of position information. For instance, if **nn = 4**, the display will show values as low as 0.0001 units. If **nn = 7**, the display will show values in exponential form. If the user units (refer SN command) are in encoder counts or stepper increments, the position information is displayed in integer form, independent of the value set by this command.

RETURNS If "?" sign takes the place of **nn** value, this command reports current setting.

REL. COMMANDS None

EXAMPLE

1FP?	*read position display resolution for axis #1*
4	*controller returns a value of 4*
1TP	*read actual position of axis #1*
5.0001	*controller returns position value*
1FP2	*set position display resolution for axis #1 to 2*
1TP	*read actual position of axis #1*
5.00	*controller returns position value*
1FP7	*set position display resolution for axis #1 to 7*
1TP	*read actual position of axis #1*
5.000000E+0	*controller returns position value*

FR set encoder full-step resolution

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**FR**nn or xx**FR**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	encoder full step resolution
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**2e-9** to **2e+9** in user defined units or **?** to read present setting
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to set the encoder full step resolution for a Newport Unidrive compatible programmable driver with step motor axis.

RETURNS If "?" sign takes the place of **nn** value, this command reports current setting.

REL. COMMANDS
QS - set microstep factor
SU - set encoder resolution

EXAMPLE

2FR?	|	*read encoder full-step resolution setting of axis # 2*
0.0001	|	*controller returns a value of 0.0001 units for axis #2*
2FR0.0005	|	*set encoder full-step resolution to 0.0005 units for axis #2*
SM	|	*save all controller settings to non-volatile memory*

GR set master-slave reduction ratio

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**GR**nn or xx**GR**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	reduction ratio
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	±1,000,000
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION

This command sets the master-slave reduction ratio for a slave axis. The trajectory of the slave is the desired trajectory or actual position of the master scaled by reduction ratio. Refer to the **TJ** command to specify the desired trajectory mode for a slave axis.

Note:
Use this command very carefully. The slave axis will have its speed and acceleration in the same ratio as the position. Also, ensure that the ratio used for the slave axis does not cause overflow of this axis' parameters (speed, acceleration), especially with ratios greater than 1.

RETURNS

If "?" sign is issued along with command, the controller returns master-slave reduction ratio.

REL. COMMANDS

SS	-	define master-slave relationship

EXAMPLE

2SS1	*| set axis 2 to be the slave of axis 1*
2SS?	*| query the master axis number for axis 2*
1	*| controller returns a value of 1*
2TJ5	*| set axis 2 trajectory mode to 5*
2GR0.5	*| set the reduction ratio of axis 2 to 0.5*
2GR?	*| query the reduction ratio of axis 2*
0.5	*| controller returns a value of 0.5*

HA set group acceleration

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HA**nn or xx**HA**?

PARAMETERS

Description	**xx** [int]	-	group number
	nn [float]	-	vector acceleration value
Range	**xx**	-	**1** to **MAX GROUPS**
	nn	-	**0** to minimum of the maximum acceleration values of all axes assigned to this group.
Units	**xx**	-	none
	nn	-	predefined units / second^2

Defaults	**xx** missing:	error 13, GROUP NUMBER MISSING
	out of range:	error 14, GROUP NUMBER OUT OF RANGE
	not assigned:	error 15, GROUP NUMBER NOT ASSIGNED
	floating point:	truncated
	nn missing:	error 7, PARAMETER OUT OF RANGE
	negative:	error 22, GROUP PARAMETER OUT OF RANGE
	out of range:	error 24, GROUP MAXIMUM ACCELERATION EXCEEDED

DESCRIPTION

This command is used to set the vectorial acceleration value for a group. This value will be used during coordinated motion of axes assigned to the group. It will override any original acceleration values specified for individual axes using **AC** command. The axes' original values will be restored when the group to which they have been assigned is deleted.

This command takes effect immediately. It can be executed when controller is idling or motion is in progress or inside a program.

Note:
Avoid changing acceleration during acceleration or deceleration phases of a move. For better predictable results, change acceleration only when all the axes assigned to this group are not in motion.

RETURNS

If "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS

AU	-	set maximum acceleration and deceleration for an axis
HN	-	create a new group
HD	-	set vectorial deceleration for a group

EXAMPLE

1HN1,2	*| create a new group (#1) with physical axes 1 and 2*
1AU?	*| query maximum acceleration of axis #1*
50	*| controller returns a value of 50 units/second^2*
2AU?	*| query maximum acceleration of axis #2*
60	*| controller returns a value of 60 units/second^2*
1HA50	*| set vectorial acceleration of group #1 to 50 units/second^2*
1HA?	*| query vectorial acceleration of group #1*
50	*| controller returns a value of 50 units/second^2*

HB read list of groups assigned

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **HB**

PARAMETERS None

DESCRIPTION This command is used to read the group numbers that have already been created or assigned.

RETURNS This command reports the current setting. If no groups have been created, controller returns error number 15, GROUP NUMBER NOT ASSIGNED.

REL. COMMANDS

HN	-	create a new group
HX	-	delete a group

EXAMPLE

1HN1,2	*| create a new group (#1) with physical axes 1 and 2*
1HN?	*| read axes assigned to group #1*
1,2	*| controller returns the axes assigned to group #1*
2HN3,4	*| create a new group (#2) with physical axes 3 and 4*
2HN?	*| read axes assigned to group #2*
3,4	*| controller returns the axes assigned to group #2*
HB	*| read list of groups created*
1 2	*| controller returns 1 and 2*

HC move group along an arc

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HC**nn_1, nn_2, nn_3 **or** xx**HC**?

PARAMETERS

Description	**xx** [int]	-	group number
	nn_1 [float]	-	first coordinate of arc center
	nn_2 [float]	-	second coordinate of arc center
	nn_3 [float]	-	arc sweep angle
Range	**xx**	-	**1** to **MAX GROUPS**
	nn_1, nn_2	-	any position within the travel limits
	nn_3	-	any angle
Units	**xx**	-	none
	nn_1, nn_2	-	predefined units
	nn_3	-	degrees
Defaults	**xx** missing:		error 13, GROUP NUMBER MISSING
	out of range:		error 14, GROUP NUMBER OUT OF RANGE
	not assigned:		error 15, GROUP NUMBER NOT ASSIGNED
	floating point:		truncated
	nn_i		
	Missing parameter:		error 21, GROUP PARAMETER MISSING

DESCRIPTION

This command initiates motion of a group along an arc. It causes all axes assigned to the group to move with predefined vectorial (tangential) velocity, acceleration and deceleration along an arc. The group target position is determined based on the position of axes at the beginning of move, center of arc and sweep angle.

If this command is received while a group move is in progress, the new command gets enqueued into a “via point” buffer. Please refer to Advanced Capabilities section for a detailed description of via point buffer implementation. The enqueued commands get executed on a FIFO basis when the move already in progress has reached its destination. The group does not come to a stop at the end of last move. Instead, there will be a smooth transition to the new move command, just as if it were one compound move (combination of multiple moves).

Note:
Only trapezoid velocity profile is employed linear interpolation motion.

Note:

The transition from last move to new move will be smooth if tangential velocity at the end of last move is the same as that at the beginning of new move.

RETURNS If "?" sign takes the place of **nn** values, this command reports the commanded center position of arc and sweep angle.

REL. COMMANDS

HN	-	create a new group
HV	-	set vectorial velocity for a group
HA	-	set vectorial acceleration for a group
HD	-	set vectorial deceleration for a group
HO	-	enable a group
HF	-	disable a group
HL	-	move a group of axes to desired position along a line.

EXAMPLE

1HN1,2	*| create a new group (#1) with physical axes 1 and 2*
1HV10	*| set vectorial velocity of group #1 to 10 units/second*
1HA50	*| set vectorial acceleration of group #1 to 50 units/second^2*
1HD50	*| set vectorial deceleration of group #1 to 50 units/second^2*
1HO	*| enable group #1*
1HP?	*| query current group position*
50,50	*| controller returns axis #1 = 50 units and axis #2 = 50 units*
1HC40,60,180	| set axis #1 arc center = 40 units
	set axis #2 arc center = 60 units
	set sweep angle of arc = 180 degrees
1HC?	*| query target position of the commanded move*
40, 60, 180	*| controller returns axis #1 arc center = 40 units, axis #2 arc center = 70 units and arc sweep angle = 180 degrees*

HD set group deceleration

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HD**nn or xx**HD**?

PARAMETERS

Description			
	xx [int]	-	group number
	nn [float]	-	vector deceleration value
Range	**xx**	-	**1** to **MAX GROUPS**
	nn	-	**0** to minimum of the maximum deceleration values of all axes assigned to this group.
Units	**xx**	-	none
	nn	-	predefined units / second^2
Defaults	**xx** missing:		error 13, GROUP NUMBER MISSING
	out of range:		error 14, GROUP NUMBER OUT OF RANGE
	not assigned:		error 15, GROUP NUMBER NOT ASSIGNED
	floating point:		truncated
	nn missing:		error 7, PARAMETER OUT OF RANGE
	negative:		error 22, GROUP PARAMETER OUT OF RANGE
	out of range:		error 25, GROUP MAXIMUM DECELERATION EXCEEDED

DESCRIPTION

This command is used to set the vectorial deceleration value for a group. This value will be used during coordinated motion of axes assigned to the group. It will override any original deceleration values specified for individual axes using **AG** command. The axes' original values will be restored when the group to which they have been assigned is deleted.

This command takes effect immediately. It can be executed when controller is idling or motion is in progress or inside a program.

Note:
Avoid changing deceleration during acceleration or deceleration phases of a move. For better predictable results, change deceleration only when all the axes assigned to this group are not in motion.

RETURNS

If "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS

AU	-	set maximum acceleration and deceleration for an axis
HN	-	create a new group
HA	-	set vectorial acceleration for a group

EXAMPLE	1HN1,2	| *create a new group (#1) with physical axes 1 and 2*
	1AU?	| *query maximum deceleration of axis #1*
	50	| *controller returns a value of 50 units/second^2*
	2AU?	| *query maximum deceleration of axis #2*
	60	| *controller returns a value of 60 units/second^2*
	1HD50	| *set vectorial deceleration of group #1 to 50 units/second^2*
	1HD?	| *query vectorial deceleration of group #1*
	50	| *controller returns a value of 50 units/second^2*

HE set group e-stop deceleration

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HE**nn or xx**HE**?

PARAMETERS

Description	**xx** [int]	-	group number
	nn [float]	-	vector e-stop deceleration value
Range	**xx**	-	**1** to **MAX GROUPS**
	nn	-	maximum of deceleration values assigned to all axes in the group to **2e9 * encoder resolution.**
Units	**xx**	-	none
	nn	-	predefined units / second^2

Defaults	**xx** missing:	error 13, GROUP NUMBER MISSING
	out of range:	error 14, GROUP NUMBER OUT OF RANGE
	not assigned:	error 15, GROUP NUMBER NOT ASSIGNED
	floating point:	truncated
	nn missing:	error 7, PARAMETER OUT OF RANGE
	negative:	error 22, GROUP PARAMETER OUT OF RANGE
	out of range:	error 22, GROUP PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to set the vectorial e-stop deceleration value for a group. This value will be used during coordinated motion of axes assigned to the group. It will override any original e-stop deceleration values specified for individual axes using **AE** command. The axes' original values will be restored when the group to which they have been assigned is deleted.

This command takes effect immediately. It can be executed when controller is idling or motion is in progress or inside a program.

E-stop deceleration is invoked upon a local e-stop condition (e.g., front panel "Stop All" push button, interlock, etc…) has occurred, if configured to do so, or if the AB (abort motion) command is processed.

RETURNS If "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS

HN	-	create a new group
HV	-	set vectorial velocity for a group
HA	-	set vectorial acceleration for a group
HD	-	set vectorial deceleration for a group

EXAMPLE

1HN1,2	*| create a new group (#1) with physical axes 1 and 2*
1HE100	*| set vectorial e-stop deceleration of group #1 to 100 units/second^2*
1HE?	*| query vectorial e-stop deceleration of group #1*
100	*| controller returns a value of 100 units/second^2*

HF group off

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HF** or xx**HF**?

PARAMETERS

Description	**xx** [int]	-	group number
Range	**xx**	-	**1** to **MAX GROUPS**
Units	**xx**	-	none
Defaults	**xx** missing:		error 13, GROUP NUMBER MISSING
	out of range:		error 14, GROUP NUMBER OUT OF RANGE
	not assigned:		error 15, GROUP NUMBER NOT ASSIGNED
	floating point:		truncated

DESCRIPTION This command turns power OFF of all axes assigned to a group. Refer **MF** command to turn the power OFF of individual axes. The group power is assumed to be OFF if power to any one of the axes in the group is OFF.

RETURNS If "?" sign is issued along with command, the controller returns:

1	-	group power is ON
0	-	group power is OFF

REL. COMMANDS

HN	-	create a new group
HO	-	turn group power ON

EXAMPLE

1HN1,2	*| create a new group (#1) with physical axes 1 and 2*
1HO	*| turn group #1 power ON*
1HF?	*| query group #1 power status*
1	*| controller returns a value of 1*
1HF	*| turn group #1 power OFF*
1HF?	*| query group #1 power status*
0	*| controller returns a value of 0*

HJ set group jerk

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HJ**nn or xx**HJ**?

PARAMETERS

Description	**xx** [int]	-	group number
	nn [float]	-	vector jerk value
Range	**xx**	-	**1** to **MAX GROUPS**
	nn	-	**0** to **2e9**
Units	**xx**	-	none
	nn	-	predefined units / second^3

Defaults	**xx** missing:	error 13, GROUP NUMBER MISSING
	out of range:	error 14, GROUP NUMBER OUT OF RANGE
	not assigned:	error 15, GROUP NUMBER NOT ASSIGNED
	floating point:	truncated
	nn missing:	error 7, PARAMETER OUT OF RANGE
	negative:	error 22, GROUP PARAMETER OUT OF RANGE
	out of range:	error 22, GROUP PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to set the vectorial jerk value for a group. This value will be used during coordinated motion of axes assigned to the group. It will override any original jerk values specified for individual axes using **JK** command. The axes' original values will be restored when the group to which they have been assigned is deleted.

If vectorial jerk is set to zero, a trapezoid velocity profile is employed during motion. Otherwise, an S-curve velocity profile is employed.

This command takes effect immediately. It can be executed when controller is idling or motion is in progress or inside a program.

Note:
Avoid changing jerk during acceleration or deceleration phases of a move. For better predictable results, change jerk only when all the axes assigned to this group are not in motion.

RETURNS If "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS

HN	-	create a new group
HV	-	set vectorial velocity for a group
HA	-	set vectorial acceleration for a group
HD	-	set vectorial deceleration for a group
HK	-	set vectorial e-stop jerk for a group

EXAMPLE

1HN1,2	*| create a new group (#1) with physical axes 1 and 2*
1HJ50	*| set vectorial jerk of group #1 to 50 units/second^3*
1HJ?	*| query vectorial deceleration of group #1*
50	*| controller returns a value of 50 units/second^3*

HL move group along a line

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HL**nn_1, nn_2, …nn_i **or** xx**HL**?

PARAMETERS

Description			
	xx [int]	-	group number
	nn_1 [float]	-	target position of first axis in a group
	nn_2 [float]	-	target position of second axis in a group
	nn_i [float]	-	target position of *i*th axis in a group, where *i* can vary from 1 to 6
Range	**xx**	-	**1** to **MAX GROUPS**
	nn_i	-	any position within the travel limits
Units	**xx**	-	none
	nn_i	-	redefined units
Defaults	**xx** missing:		error 13, GROUP NUMBER MISSING
	out of range:		error 14, GROUP NUMBER OUT OF RANGE
	not assigned:		error 15, GROUP NUMBER NOT ASSIGNED
	floating point:		truncated
	nn_i		
	Missing parameter:		error 21, GROUP PARAMETER MISSING

DESCRIPTION

This command initiates motion of a group along a line. It causes all axes assigned to the group to move with predefined vectorial (tangential) velocity, acceleration and deceleration along a line. A trapezoid velocity profile is employed when vectorial jerk is set to zero. Otherwise, an S-curve velocity profile is employed.

If this command is received while a group move is in progress, the new command gets enqueued into a “via point” buffer. Please refer Advanced Capabilities section for a detailed description of via point buffer implementation. The enqueued commands get executed on a FIFO basis when the move already in progress has reached its destination. The group does not come to a stop at the end of last move. Instead, there will be a smooth transition to the new move command, just as if it were one compound move (combination of multiple moves).

Note:

The transition from last move to new move will be smooth if tangential velocity at the end of last move is the same as that at the beginning of new move.

RETURNS

If “?” sign takes the place of **nn** values, this command reports the target positions of axes assigned to the group.

REL. COMMANDS	**HN**	**-**	create a new group
	HV	**-**	set vectorial velocity for a group
	HA	**-**	set vectorial acceleration for a group
	HD	**-**	set vectorial deceleration for a group
	HO	**-**	enable a group
	HF	**-**	disable a group
	HC	**-**	move a group of axes to desired position along an arc.

EXAMPLE	1HN1,2	*| create a new group (#1) with physical axes 1 and 2*
	1HV10	*| set vectorial velocity of group #1 to 10 units/second*
	1HA50	*| set vectorial acceleration of group #1 to 50 units/second^2*
	1HD50	*| set vectorial deceleration of group #1 to 50 units/second^2*
	1HO	*| enable group #1*
	1HP?	*| query current group position*
	0,0	*| controller returns axis #1 = 0 units and axis #2 = 0 units*
	1HL50, 50	| move axis #1 to a target position = 50 units *move axis #2 to a target position = 50 units*
	1HL?	*| query target position of the commanded move*
	50,50	*| controller returns axis #1 = 50 units and axis #2 = 50 units*

HN create new group

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**HN**nn_1, nn_2, …nn_i **or** xx**HN**?

PARAMETERS

Description

xx	[int]	-	group number
nn_1	[int]	-	physical axis number to be assigned as first axis in this group
nn_2	[int]	-	physical axis number to be assigned as second axis in this group
nn_i	[int]	-	physical axis number to be assigned as *i*th axis in this group

Range

xx	-	**1** to **MAX GROUPS**
nn_i	-	**1** to **MAX AXES**

Units

xx	-	none
nn_i	-	none

Defaults

xx missing:	error 13, GROUP NUMBER MISSING
out of range:	error 14, GROUP NUMBER OUT OF RANGE
not assigned:	error 15, GROUP NUMBER NOT ASSIGNED
already assigned:	error 16, GROUP NUMBER ALREADY ASSIGNED
floating point:	truncated
nn_i	
out of range:	error 17, GROUP AXIS OUT OF RANGE
already assigned:	error 18, GROUP AXIS ALREADY ASSIGNED
duplicated:	error 19, GROUP AXIS DUPLICATED
Missing parameter:	error 21, GROUP PARAMETER MISSING

DESCRIPTION

This command is used to create a new group. A few rules are in place to facilitate easy management of groups.

- A group has to be created with at least two axes assigned to it before any command related to groups can be issued. The controller returns error 15, GROUP NUMBER NOT ASSIGNED, if, for instance, one tries to set velocity for group #1, before creating group #1.
- A group has to be deleted (refer **HX** command) before axes assigned to the group can be changed. The controller returns error 16, GROUP NUMBER ALREADY ASSIGNED, if one attempts to change axes assigned to a group already created. Please see the following table for correct method to change axes assigned to a group:

Correct Method	Incorrect Method
1HN1,2	1HN1,2
1HX	1HN2,3
1HN2,3	

- An axis cannot be a member of (or assigned to) different groups at the same time. The controller returns error 18, GROUP AXIS ALREADY ASSIGNED, if one

attempts to assign an axis under such circumstances. Refer HX command to delete a group.

- An axis cannot be assigned more than once in a group. The controller returns error 19, GROUP AXIS DUPLICATED, if one attempts to assign an axis more than once to a group.
- The order in which axes are assigned to a group is very important. This is because it specifies the frame of reference in which coordinated motion of axes takes place. For instance, the command **1HN1,2** assigns axis numbers 1 and 2 to group number 1, where axis #1 is equivalent to X-axis and axis #2 is equivalent to Y-axis in a traditional cartesian coordinate system. Reversing the ordering of axes (viz. **1HN2,1**) reverses the axis assignment.
- If a group has more than two axes assigned to it, and the group was commanded to make an arc (refer to HC command), the first two axes in the group are used to make the desired move.

RETURNS

If "?" sign takes the place of **nn** values, this command reports the axes assigned to the group in the order of their assignment.

REL. COMMANDS

HV	-	set vectorial velocity for a group
HA	-	set vectorial acceleration for a group
HD	-	set vectorial deceleration for a group
HO	-	enable a group
HF	-	disable a group
HC	-	move a group of axes to desired position along an arc.
HL	-	move a group of axes to desired position along a line.

EXAMPLE

1HN1,2	*| create a new group (#1) with physical axes 1 and 2*
1HN?	*| query axis assigned to group #1*
1,2	*| controller returns the axes assigned to group #1*
1HN2,3	*| create a new group (#1) with physical axes 1 and 2*
1HN?	*| query axis assigned to group #1*
1,2	*| controller returns the axes assigned to group #1*
TB?	*| read error message*
0, 450322, GROUP NUMBER ALREADY ASSIGNED	
1HX	*| delete group #1*
1HN2,3	*| create a new group (#1) with physical axes 1 and 2*
1HN?	*| query axis assigned to group #1*
2,3	*| controller returns the axes assigned to group #1*
2HN?	*| query axis assigned to group #2*
TB?	*| read error message*
0, 475322, GROUP NUMBER NOT ASSIGNED	
2HN3,4	*| create a new group (#2) with physical axes 3 and 4*
2HN?	*| query axis assigned to group #2*
TB?	*| read error message*
0, 500322, GROUP AXIS ALREADY ASSIGNED	
2HN4,4,5	*| create a new group (#2) with physical axes 4, 4 and 5*
2HN?	*| query axis assigned to group #2*
TB?	*| read error message*
0, 525322, GROUP AXIS DUPLICATED	

HO group on

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HO** or xx**HO**?

PARAMETERS

Description	**xx** [int]	-	group number
Range	**xx**	-	**1** to **MAX GROUPS**
Units	**xx**	-	none
Defaults	**xx** missing:		error 13, GROUP NUMBER MISSING
	out of range:		error 14, GROUP NUMBER OUT OF RANGE
	not assigned:		error 15, GROUP NUMBER NOT ASSIGNED
	floating point:		truncated

DESCRIPTION This command turns power ON of all axes assigned to a group. Refer **MO** command to turn the power ON of individual axes. The group power is assumed to be ON if power to all axes in the group is ON.

RETURNS If "?" sign is issued along with command, the controller returns:

1	-	group power is ON
0	-	group power is OFF

REL. COMMANDS

HN	-	create a new group
HF	-	turn group power OFF

EXAMPLE

1HN1,2	*| create a new group (#1) with physical axes 1 and 2*
1HO	*| turn group #1 power ON*
1HO?	*| query group #1 power status*
1	*| controller returns a value of 1*
1HF	*| turn group #1 power OFF*
1HO?	*| query group #1 power status*
0	*| controller returns a value of 0*

HP read group position

	IMM	PGM	MIP	
USAGE	♦		♦	

SYNTAX xx**HP**

PARAMETERS

Description	**xx** [int]	-	group number
Range	**xx**	-	**1** to **MAX GROUPS**
Units	**xx**	-	none
Defaults	**xx** missing:		error 13, GROUP NUMBER MISSING
	out of range:		error 14, GROUP NUMBER OUT OF RANGE
	not assigned:		error 15, GROUP NUMBER NOT ASSIGNED
	floating point:		truncated

DESCRIPTION This command is used to read the actual position, the instantaneous real position of all axes assigned to a group.

RETURNS $\mathbf{nn_1}$, $\mathbf{nn_2}$, … $\mathbf{nn_i}$ where $\mathbf{nn_i}$ = actual position of *i*th axis in the group.

REL. COMMANDS

HN	-	create a new group
HC	-	move a group of axes to desired position along an arc.
HL	-	move a group of axes to desired position along a line.

EXAMPLE

1HN1,2	*| create a new group (#1) with physical axes 1 and 2*
1HP	*| read position of group #1*
10,50	*| controller returns axis #1 = 10 units, axis #2 = 50 units*

HQ wait for group command buffer level

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HQ**nn or xx**HQ**?

PARAMETERS

Description

xx	[int]	-	group number
nn	[float]	-	level in group via point buffer

Range

xx	-	**1** to **MAX GROUPS**
nn	-	to **10** (default for maximum targets in via point buffer)

Units

xx	-	none
nn	-	milliseconds

Defaults

xx missing: error 13, GROUP NUMBER MISSING
out of range: error 14, GROUP NUMBER OUT OF RANGE
not assigned: error 15, GROUP NUMBER NOT ASSIGNED
floating point:truncated

nn
Missing parameter: error 21, GROUP PARAMETER MISSING

DESCRIPTION This command stops enqueuing new commands into the via point buffer until the buffer level equals **nn**. As commands in the buffer get executed on a FIFO basis and the buffer level equals **nn**, commands issued subsequent to this one get executed.

RETURNS If "?" sign takes the place of **nn** value, the controller returns the room available in via point buffer for more commands.

REL. COMMANDS

HN	-	create a new group
HL	-	move group to target position along a line
HC	-	move group to target position along an arc

EXAMPLE

```
1HN1,2      | create a new group (#1) with physical axes 1 and 2
1HV10       | set vectorial velocity of group #1 to 10 units/second
1HA50       | set vectorial acceleration of group #1 to 50 units/second^2
1HD50       | set vectorial deceleration of group #1 to 50 units/second^2
1HO         | enable group #1
1HL10,10    | move group #1 to target pos. 10,10 (ax. #1 = 10, #2 = 10 units)
1HL20,20    | move group #1 to target pos. 20,20 (ax. #1 = 20, #2 = 20 units).
            | This command gets enqueued in the via point buffer if it was
            | received prior completion of the previous move command.
1HL50,50    | move group #1 to target pos. 50,50 (ax. #1 = 50, #2 = 50 units).
1HQ10       | wait until the via point buffer level equals 10 commands
1HC40,60,180| move group #1 along an arc with center of arc at (40,60) units,
            | by a sweep angle of 180 deg. from current position.
```

HS stop group motion

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HS or** xx**HS**?

PARAMETERS

Description	**xx** [int]	-	group number
Range	**xx**	-	**1** to **MAX GROUPS**
Units	**xx**	-	none
Defaults	**xx** missing:		error 13, GROUP NUMBER MISSING
	out of range:		error 14, GROUP NUMBER OUT OF RANGE
	not assigned:		error 15, GROUP NUMBER NOT ASSIGNED
	floating point:		truncated

DESCRIPTION This command stops the motion of all axes assigned to a group using vector deceleration set using **HD** command.

RETURNS If "?" sign is supplied along with the command, the controller returns:

1 - group motion is stopped
0 - group motion is in progress

REL. COMMANDS

HN - create a new group
HC - move a group of axes to desired position along an arc.
HL - move a group of axes to desired position along a line.

EXAMPLE

1HN1,2	*| create a new group (#1) with physical axes 1 and 2*
1HV10	*| set vectorial velocity of group #1 to 10 units/second*
1HA50	*| set vectorial acceleration of group #1 to 50 units/second^2*
1HD50	*| set vectorial deceleration of group #1 to 50 units/second^2*
1HO	*| enable group #1*
1HP?	*| query current group position*
0,0	*| controller returns axis #1 = 0 units and axis #2 = 0 units*
1HL50, 50	*| move axis #1 to a target position = 50 units* *move axis #2 to a target position = 50 units*
1HS?	*| query if motion of group #1 is stopped*
0	*| controller returns 0, meaning group #1 is in motion*
1HS	*| stop motion of group #2*
1HS?	*| query if motion of group #1 is stopped*
1	*| controller returns 1, meaning group #1 motion has stopped*

HV set group velocity

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HV**nn or xx**HV**?

PARAMETERS

Description	**xx** [int]	-	group number
	nn [float]	-	vector velocity value
Range	**xx**	-	**1** to **MAX GROUPS**
	nn	-	**0** to minimum of the maximum velocity values of all axes assigned to this group.
Units	**xx**	-	none
	nn	-	predefined units / second

Defaults	**xx** missing:	error 13, GROUP NUMBER MISSING
	out of range:	error 14, GROUP NUMBER OUT OF RANGE
	not assigned:	error 15, GROUP NUMBER NOT ASSIGNED
	floating point:	truncated
	nn missing:	error 7, PARAMETER OUT OF RANGE
	negative:	error 22, GROUP PARAMETER OUT OF RANGE
	out of range:	error 23, GROUP MAXIMUM VELOCITY EXCEEDED

DESCRIPTION

This command is used to set the vectorial velocity value for a group. This value will be used during coordinated motion of axes assigned to the group. It will override any original acceleration values specified for individual axes using **VA** command. The axes' original values will be restored when the group to which they have been assigned is deleted.

This command takes effect immediately. It can be executed when controller is idling or motion is in progress or inside a program.

Note:
Avoid changing velocity during acceleration or deceleration phases of a move. For better predictable results, change velocity only when all the axes assigned to this group are not in motion.

RETURNS

If "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS

VU	-	set maximum velocity for an axis
HN	-	create a new group

EXAMPLE

1HN1,2	*| create a new group (#1) with physical axes 1 and 2*
1VU?	*| query maximum velocity of axis #1*
10	*| controller returns a value of 10 units/second*
2VU?	*| query maximum velocity of axis #2*
15	*| controller returns a value of 15 units/second*
1HV10	*| set vectorial velocity of group #1 to 10 units/second*
1HV?	*| query vectorial velocity of group #1*
10	*| controller returns a value of 10 units/second*

HW wait for group motion stop

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HW**nn

PARAMETERS

Description	**xx** [int]	-	group number
	nn [float]	-	delay after group motion is complete
Range	**xx**	-	**1** to **MAX GROUPS**
	nn	-	**0** to **60000**
Units	**xx**	-	none
	nn	-	milliseconds
Defaults	**xx** missing:		error 13, GROUP NUMBER MISSING
	out of range:		error 14, GROUP NUMBER OUT OF RANGE
	not assigned:		error 15, GROUP NUMBER NOT ASSIGNED
	floating point:		truncated
	nn missing:		error 7, PARAMETER OUT OF RANGE
	negative:		error 22, GROUP PARAMETER OUT OF RANGE
	out of range:		error 26, MAXIMUM WAIT DURATION EXCEEDED

DESCRIPTION This command stops execution of any commands subsequent to it until the one prior to it has been completed. For instance, if a command preceding it is a group move command such as **HL** or **HC**, it stops execution of any commands following it until the group has reached target position. If **nn** is not equal to zero, the controller waits an additional **nn** milliseconds after the group motion is complete before executing any further commands.

RETURNS none

REL. COMMANDS

HN	-	create a new group
HL	-	move group to target position along a line

EXAMPLE

```
1HN1,2                          | create a new group (#1) with physical axes 1 and 2
2HN3,4                          | create a new group (#2) with physical axes 3 and 4
1HV10                           | set vectorial velocity of group #1 to 10 units/second
1HA50                           | set vectorial acceleration of group #1 to 50 units/second²
1HD50                           | set vectorial deceleration of group #1 to 50 units/second²
2HV10                           | set vectorial velocity of group #2 to 10 units/second
2HA50                           | set vectorial acceleration of group #2 to 50 units/second²
2HD50                           | set vectorial deceleration of group #2 to 50 units/second²
1HO                             | enable group #1
2HO                             | enable group #2
1HL50, 50; 1HW500; 2HL30,20     | move group #1 to a target position = 50, 50
                                | units (axis #1 = 50 units and axis #2 = 50 units), wait for the
```

| group to reach target position, wait an additional 500 ms, and
| then move group #2 to a target position = 30, 20 units (axis #3
| = 30 units and axis #4 = 20 units)

HX delete group

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HX**

PARAMETERS

Description			
	xx [int]	-	group number
Range	**xx**	-	**1** to **MAX GROUPS**
Units	**xx**	-	none
Defaults	**xx** missing:		error 13, GROUP NUMBER MISSING
	out of range:		error 14, GROUP NUMBER OUT OF RANGE
	not assigned:		error 15, GROUP NUMBER NOT ASSIGNED

DESCRIPTION This command deletes a group and makes available any axes that were assigned to it for future assignments.

RETURNS none

REL. COMMANDS **HN** - reate a new group

EXAMPLE

1HN1,2 | *create a new group (#1) with physical axes 1 and 2*

1HN? | *query axes assigned to group #1*
1,2 | *controller returns the axes assigned to group #1*
1HX | *delete group #1*
1HN? | *query axis assigned to group #1*
TB? | *read error message*
0, 475322, GROUP NUMBER NOT ASSIGNED

HZ read group size

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**HZ**

PARAMETERS

Description			
	xx [int]	-	group number
Range	**xx**	-	**1** to **MAX GROUPS**
Units	**xx**	-	none
Defaults	**xx** missing:		error 13, GROUP NUMBER MISSING
	out of range:		error 14, GROUP NUMBER OUT OF RANGE
	not assigned:		error 15, GROUP NUMBER NOT ASSIGNED

DESCRIPTION This command is used to read the number of axes assigned to a group.

RETURNS This command reports the current setting.

REL. COMMANDS

HN	-	create a new group
HX	-	delete a group

EXAMPLE

1HN1,2	*| create a new group (#1) with physical axes 1 and 2*
1HN?	*| read axes assigned to group #1*
1,2	*| controller returns the axes assigned to group #1*
1HZ	*| read size of group #1*
2	*| controller returns 2*

ID read stage model and serial number

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **ID ?**

PARAMETERS

Description	**xx** [int]	-	axis number
Range	**xx**	-	**1** to **MAX AXES**
Units	**xx**	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	timeout:		error 2, RS-232 COMMUNICATION TIME-OUT

DESCRIPTION This command is used to read Newport ESP compatible positioner (stage) model and serial number.

Note:
An important information needed when asking for help with the motion control system or when reporting a problem is the stage model and serial number. Use this command to determine the positioner model and serial number.

RETURNS **xx,yy**
where: **xx** = model number
yy = serial number

REL. COMMANDS none

EXAMPLE

1 ID ?	*read axis-1 positioner model and serial number*
TS50DC.5, SN1263	*controller returns model and serial number*

JH set jog high speed

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**JH**nn or xx**JH**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	high speed value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to maximum value allowed by VU command or **?** to read present setting
Units	**xx**	-	none
	nn	-	preset units/second
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx10, MAXIMUM VELOCITY EXCEEDED

DESCRIPTION This command is used to set the high speed for jogging an axis. Its execution is immediate, meaning that the value is changed when the command is processed, including when motion is in progress. It can be used as an immediate command or inside a program.

RETURNS If "?" sign takes the place of **nn** value, this command reports current setting.

REL. COMMANDS

JW	-	set jog low speed
VU	-	set maximum velocity

EXAMPLE

2VU?	| *read maximum velocity allowed axis # 2*
10	| *controller returns a value of 10.0 units/second for axis #2*
2JH7.5	**|** *set jog high speed to 7.5 units/second for axis #2*
2JH?	**|** *read jog high speed value for axis #2*
7.5	| *controller returns a value of 7.5 units/second for axis #2*

JK set jerk rate

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**JK**nn or xx**JK**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	jerk value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **2e9**
Units	**xx**	-	none
	nn	-	preset units / second^3
			or **?** to read current setting
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx15, MAXIMUM JERK EXCEEDED

DESCRIPTION

This command is used to set the jerk (i.e., rate of change in acceleration) value for an axis. Its execution is immediate, meaning that the jerk is altered when the command is processed and trajectory mode is set to S-curve, even while a motion is in progress. It can be used as an immediate command or inside a program.

Note
Avoid changing the jerk during the acceleration or deceleration periods. For better predictable results, change jerk only when the axis is not moving.

RETURNS none

REL. COMMANDS

AC	-	set acceleration
TJ	-	set trajectory mode
VA	-	set velocity

EXAMPLE

2**JK**?	| *read desired velocity of axis # 2*
10.5	| *controller returns a velocity value of 10.5 units/s^3*
2**JK**15	| set axis #2 jerk to 15 *units/s^3*

JL jump to label

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**JL**nn

PARAMETERS

Description

xx [int] - label number
nn [int] - loop count

Range

xx - **1** to **100**
nn - **1** to **65535**

Units

xx - none
nn - none

Default

xx missing: error 38, COMMAND PARAMETER MISSING
out of range: error xx2, PARAMETER OUT OF RANGE

nn missing: assume infinite
out of range: error xx2, PARAMETER OUT OF RANGE

DESCRIPTION

This command changes the flow of the program execution by jumping to a predefined label **xx**. This a flow control command that alters the normal sequential flow of a program. It must be used in conjunction with the DL command which defines a label.
Parameter **nn** determines the number of times to repeat the jump before allowing the program to flow passed.

RETURNS

none

REL. COMMANDS

JL - jump to label

EXAMPLE

```
3XX     | clear program 3 from memory, if any
3EP     | create program 3
1DL     | define label 1
   .
   .
   .
1JL 5   | jump to label 1 five(5) times
QP      | end entering program and quit programming mode
3EX     | run stored program number 3
```

JW set jog low speed

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**JW**nn or xx**JW**?

PARAMETERS

Description			
	xx [int]	-	axis number
	nn [float]	-	low speed value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to maximum value allowed by VU command or **?** to read present setting
Units	**xx**	-	none
	nn	-	reset units/second
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx10, MAXIMUM VELOCITY EXCEEDED

DESCRIPTION This command is used to set the low speed for jogging an axis. Its execution is immediate, meaning that the value is changed when the command is processed, including when motion is in progress. It can be used as an immediate command or inside a program.

RETURNS If "?" sign takes the place of **nn** value, this command reports current setting.

REL. COMMANDS

JH	-	set jog high speed
VU	-	set maximum velocity

EXAMPLE

2VU?	| *read maximum velocity allowed axis # 2*
10	| *controller returns a value of 10.0 units/second for axis #2*
2JW2.5	**|** *set jog low speed to 2.5 units/second for axis #2*
2JW?	**|** *read jog low speed value for axis #2*
2.5	| *controller returns a value of 2.5 units/second for axis #2*

KD set derivative gain

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**KD**nn or xx**KD**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	derivative gain factor Kd
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **2e9,** or **?** to read current setting
Units	**xx**	-	none
	nn	-	none

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command sets the derivative gain factor Kd of the PID closed loop. It is active for any DC servo based motion device that has been selected to operate in closed loop.
The command can be sent at any time but it has no effect until the UF (update filter) is received.

.

See the "**Servo Tuning**" chapter on how to adjust the PID filter parameters.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS	**KI**	-	set integral gain factor
	KP	-	set proportional gain factor
	KS	-	set saturation gain factor
	UF	-	update filter

EXAMPLE

3KD0.01 | *set derivative gain factor for axis # 3 to 0.01*
.
.
.
3UF | *update PID filter; only now the **KD** command takes effect*

KI set integral gain

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**KI**nn or xx**KI**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	integral gain factor Ki
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **2e9,** or **?** to read current setting
Units	**xx**	-	none
	nn	-	one

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx2, PARAMETER OUT OF RANGE

DESCRIPTION

This command sets the integral gain factor Ki of the PID closed loop. It is active for any DC servo based motion device that has been selected to operate in closed loop.
The command can be sent at any time but it has no effect until the UF (update filter) is received.

.

See the "**Servo Tuning**" chapter on how to adjust the PID filter parameters.

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

KD	-	set integral gain factor
KP	-	set proportional gain factor
KS	-	set saturation gain factor
UF	-	update filter

EXAMPLE

3KI 0.01 | *set integral gain factor for axis # 3 to 0.01*

.
.
.

3UF | *update PID filter; only now the **KI** command takes effect*

KP set proportional gain

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**KP**nn or xx**KP**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	roportional gain factor Kp
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **2e9,** or **?** to read current setting
Units	**xx**	-	none
	nn	-	none

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	rror xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command sets the proportional gain factor Kp of the PID closed loop. It is active for any DC servo based motion device that has been selected to operate in closed loop.
The command can be sent at any time but it has no effect until the UF (update filter) is received.

.

See the "**Servo Tuning**" chapter on how to adjust the PID filter parameters.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

KI	-	set integral gain factor
KD	-	set proportional gain factor
KS	-	set saturation gain factor
UF	-	update filter

EXAMPLE

3KP0.01 | *set proportional gain factor for axis # 3 to 0.01*

.
.
.

3UF | *update PID filter; only now the **KP** command takes effect*

KS set saturation level of integral factor

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**KS**nn or xx**KS**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	saturation level of integrator KS
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **2e9,** or **?** to read current setting
Units	**xx**	-	none
	nn	-	none

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command sets the saturation level of the integral factor of the PID closed loop and is useful for preventing integral wind-up. It is active for any DC servo based motion device that has been selected to operate in closed loop.
The command can be sent at any time but it has no effect until the UF (update filter) is received.

.

See the "**Servo Tuning**" chapter on how to adjust the PID filter parameters.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS	**KI**	-	set integral gain factor
	KP	-	set proportional gain factor
	KD	-	set derivative gain factor
	UF	-	update filter

EXAMPLE

3KS0.01 | *set saturation level for axis # 3 to 0.01*

.
.
.

3UF | *update PID filter; only now the **KS** command takes effect*

LP list program

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX xx**LP**

PARAMETERS

Description	**xx** [int]	-	rogram number
Range	**xx**	-	**1** to **100**
Units	**xx**	-	none
Defaults	**xx**	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error 7, PARAMETER OUT OF RANGE

DESCRIPTION

This command reads a specified program from non-volatile memory and sends it to the selected communication port (RS232 or IEEE488). During the transmission no other command should be sent to the controller.

Note

The program list always terminates with the word “END”

RETURNS program listing

REL. COMMANDS **EP** - enter program mode

EXAMPLE

```
3LP       | list program number 3
3MO       | enable axis 3 motor power
1DL       | define return label 1
3PR+10    | move axis 3  relative +10 units
3WS500    | wait 500ms after axis 3 stops
3PR-10    | move axis 3  relative -10 units
3WS500    | wait 500ms after axis 3 stops
1JL5      | jump to label 1 location 5 times
END       | end of program list
```

MD read motion done status

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX xx**MD?**

PARAMETERS

Description			
	xx [int]	-	axis number
Range	**xx**	-	**1** to **MAX AXES**
Units	**xx**	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE

DESCRIPTION This command is used to read the motion status for the specified axis **n**. The MD command can be used to monitor Homing, absolute, and relative displacement move completion status.

RETURNS **nn** - **0** or **1** where 0 = motion <u>not</u> done (FALSE)
1 = motion done (TRUE)

REL. COMMANDS

PA	-	move to an absolute position
PR	-	move to a relative position
OR	-	move to home position

EXAMPLE

3MD?	*| read axis #3 move done status*
1	*| controller returns status 1 (motion done) for axis # 3*
3PR2.2	*| start a relative motion of 2.2 on axis # 3*
3MD?	*|read axis #3 move done status*
0	*|controller returns status 0 (motion not done) for axis # 3*

MF motor off

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**MF** or xx**MF**?

PARAMETERS

Description			
	xx [int]	-	axis number
Range	**xx**	-	**1** to **MAX AXES**
Units	**xx**	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE

DESCRIPTION This command turns power OFF of the specified motor (axis).

RETURNS If "?" sign is issued along with command, the controller returns:

1	-	motor power is ON
0	-	motor power is OFF

REL. COMMANDS

AB	-	abort motion
ST	-	stop motion
MO	-	urn motor power ON

EXAMPLE

2MF	| *turn axis #2 motor power OFF*
2MF?	| *query axis #2 motor power status*
0	| *controller returns a value of 0*
2MO	| *turn axis #2 motor power ON*
2MF?	| *query axis #2 motor power status*
1	| *controller returns a value of 1*

MO motor on

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**MO** or xx**MO**?

PARAMETERS

Description	**xx** [int]	-	axis number
Range	**xx**	-	to **MAX AXES**
Units	**xx**	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE

DESCRIPTION This command turns power ON of the specified motor (axis).

CAUTION:

If the motor power was turned off by the controller detecting a fault condition, before turning the power back on, make sure that the cause of the fault was corrected.

RETURNS If "?" sign is issued along with command, the controller returns:

1	-	motor power is ON
0	-	motor power is OFF

REL. COMMANDS

AB	-	abort motion
ST	-	stop motion
MF	-	urn motor power OFF

EXAMPLE

MO	*| turn axis #2 motor power ON*
2MO?	*| query axis #2 motor power status*
1	*| controller returns a value of 1*
2MF	*| turn axis #2 motor power OFF*
2MO?	*| query axis #2 motor power status*
0	*| controller returns a value of 0*

MT move to hardware travel limit

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**MT**nn or xx**MT**?

PARAMETERS

Description			
	xx [int]	-	axis number
	nn [char]	-	direction of motion
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**+** for positive direction or **–** for negative direction
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		positive direction

DESCRIPTION This command is used to move an axis to its limit (positive or negative). It uses the home search speed during travel to hardware limit.

Note: This command cannot be issued after enabling DAQ (refer ASCII command, **DE**).

RETURNS If "?" sign takes the place of **nn** value, this command reports **1** if motion is done, or **0** if motion is in progress.

REL. COMMANDS			
	OR	-	home location search
	OH	-	set home search speed

EXAMPLE		
	3MT+	| *move axis #3 to positive travel limit*
	3MT?	| *query motion status*
	0	| *controller returns 0 indicating motion is in progress*

MV move indefinitely

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**MV**nn or xx**MV**?

PARAMETERS

Description

xx [int] - axis number
nn [char] - direction of motion

Range

xx - **1** to **MAX AXES**
nn - **+** for positive direction or **–** for negative direction

Units

xx - none
nn - none

Defaults

xx	missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
nn	missing:	positive direction
	out of range:	error xx04, POSITIVE HARDWARE LIMIT EXCEEDED
	out of range:	error xx05, NEGATIVE HARDWARE LIMIT EXCEEDED
	out of range:	error xx06, POSITIVE SOFTWARE LIMIT EXCEEDED
	out of range:	error xx07, NEGATIVE SOFTWARE LIMIT EXCEEDED

DESCRIPTION

This command initiates infinite motion. When received, the selected axis **xx** will move indefinitely, with the predefined acceleration and velocity, in the direction specified by **nn**. If the requested axis is member of a group, this command does not initiate the desired motion. Instead, error xx31, " COMMAND NOT ALLOWED DUE TO GROUP ASSIGNMENT" is generated. Refer HL and HC commands to move along a line or an arc.

If this command is issued when trajectory mode for this axis is not in trapezoidal or s-curve mode, the controller returns error xx32, “INVALID TRAJECTORY MODE FOR MOVING”.

Note:

Although the command is accepted while a motion is in progress, care should be taken not to reverse direction of motion.

RETURNS

If the “?” sign takes the place of **nn** value, this command reports the motion done status.

REL. COMMANDS	**PA**	**-**	move to absolute position
	PR	**-**	move to relative position
	ST	**-**	stop motion
	MD	**-**	move done status
EXAMPLE	**3MV+**		**|** *move axis #3 indefinitely in positive direction*
	3MV?		**|** *query status of move*
	0			*controller returns 0 meaning, motion is in progress*
	3ST			*stop axis #3 motion*
	3MV–		**|** *move axis #3 indefinitely in negative direction*

MZ move to nearest index

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**MZ**nn or xx**MZ**?

PARAMETERS

Description			
	xx [int]	-	axis number
	nn [char]	-	direction of motion
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**+** for positive direction or **–** for negative direction
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		ositive direction

DESCRIPTION This command is used to move an axis to its nearest index (positive or negative). It uses the home search speed during travel to nearest index.

Note: This command cannot be issued after enabling DAQ (refer ASCII command, **DE**).

RETURNS If "?" sign takes the place of **nn** value, this command reports **1** if motion is done, or **0** if motion is in progress.

REL. COMMANDS

OR	-	home location search
OH	-	set home search speed

EXAMPLE

3MZ+	*move axis #3 to nearest index in positive direction*
3MZ?	*query motion status*
0	*controller returns 0 indicating motion is in progress*

OH set home search high speed

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**OH**nn or xx**OH**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	high speed value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to maximum value allowed by VU command or **?** to read present setting
Units	**xx**	-	none
	nn	-	reset units/second
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx10, MAXIMUM VELOCITY EXCEEDED error xx24, SPEED OUT OF RANGE

DESCRIPTION This command sets the high speed used to search for home location for an axis. Its execution is immediate, meaning that the value is changed when the command is processed, including when motion is in progress. It can be used as an immediate command or inside a program.

RETURNS If "?" sign takes the place of **nn** value, this command reports current setting.

REL. COMMANDS

OR	-	search for home
OL	-	set home search low speed

EXAMPLE

3OH10	|	*set home search high speed of axis # 3 to 10 units/sec*
3OH?	|	*query home search high speed of axis #3*
10	|	*controller returns a value of 10.0 units/second*

OL set home search low speed

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**OL**nn or xx**OL**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	low speed value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to maximum value allowed by VU command or **?** to read present setting
Units	**xx**	-	none
	nn	-	preset units/second
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx10, MAXIMUM VELOCITY EXCEEDED error xx24, SPEED OUT OF RANGE

DESCRIPTION This command sets the low speed used to search for home location for an axis. Its execution is immediate, meaning that the value is changed when the command is processed, including when motion is in progress. It can be used as an immediate command or inside a program.

RETURNS If "?" sign takes the place of **nn** value, this command reports current setting.

REL. COMMANDS

OR	-	search for home
OH	-	set home search high speed
OL	-	set home search low speed

EXAMPLE

3OL2	| *set home search low speed of axis # 3 to 2 units/sec*
3OL?	| *query home search low speed of axis #3*
2	| *controller returns a value of 2 units/second*

OM set home search mode

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	home search mode
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **6**
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION

This command selects the home search type without invoking the home search sequence (see the description of OR command for more information on home search). The seven home search types are +0 Position Count, Home Switch and Index Signals, Home Switch Signal, Positive Limit Signal, Negative Limit Signal, Positive Limit and Index Signals and Negative Limit and Index Signals.

If **nn** = **0** and the front panel HOME search push button is pressed, the axes will search for zero position count. If **nn** = **1** and the front panel HOME search push button is pressed, the axis will search for combined Home and Index signal transitions. The controller responds similarly for other values of **nn**.

The **nn** parameter is overwritten by the **OR** command parameter.

RETURNS

If "?" sign takes the place of **nn** value, this command reports current setting.

REL. COMMANDS

OR - search for home

EXAMPLE

3OM1 | *set axis #3 home search mode to 1*
3OR | *start home search on axis #3 using mode 1*

OR search for home

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**OR**nn

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	home mode
Range	**xx**	-	**0** to **MAX AXES**
	nn	-	**0** to **6** where: 0 = Find +0 Position Count 1 = Find Home and Index Signals 2 = Find Home Signal 3 = Find Positive Limit Signal 4 = Find Negative Limit Signal 5 = Find Positive Limit and Index Signals 6 = Find Negative Limit and Index Signals
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION

This command executes a Home search routine on the axis specified by **xx**. If **xx** = **0**, a home search routine is initiated sequentially on all installed axes. If **nn** is missing, the axes will search for home using the mode specified using **OM** command. If **nn** = **0**, the axes will search for zero position count. If **nn** = **1**, the axis will search for combined Home and Index signal transitions. If **nn** = **2**, the axes will search for Home signal transition only. If **nn = 3**, the axes will search for positive limit signal transition. If **nn = 4**, the axes will search for negative limit signal transition. If **nn = 5**, the axes will search for positive limit and index signal transition. If **nn = 6**, the axes will search for negative limit and index signal transition.

At the end of a home search routine, the position of axes is reset to the value specified using **SH** command.

The home search motion status can be monitored with the Motion Done (**MD**) status command. If a fault condition such as E-stop occurs while home search is in progress or if this command is issued to an axis before enabling it, the controller returns error xx20, “HOMING ABORTED”.

For a detailed description of the home search routine see the **Home - The Axis Origin** chapter in the **Motion Control Tutorial** section.

Note:
This command should be executed once every time the controller power is turned ON or the controller performs a complete system reset. There is no need to issue this command in any other case since the controller always keeps track of position, even when the motor power is OFF.

Note: This command cannot be issued after enabling DAQ (refer ASCII command, **DE**).

RETURNS none

REL. COMMANDS

DH	-	define home
OH	-	set home search speed
OM	-	set home search mode
MD	-	read motion done status
SH	-	set home preset position

EXAMPLE

3MO	|	*turn axis #3 motor power ON*
3SH0	|	*set axis #3 home position to 0 units*
3OR1	|	*perform a home search on axis # 3*
3MD?	|	*query axis #3 motion status*
1	|	*controller returns a value of 1, when motion is done*
3TP	|	*query axis #3 position*
0	|	*controller returns a value of 0 units*

PA move to absolute position

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**PA**nn or xx**PA**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	absolute position destination
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	any position within the travel limits and within ±2e9 * encoder resolution.
Units	**xx**	-	none
	nn	-	defined motion units

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx04, POSITIVE HARDWARE LIMIT EXCEEDED
	out of range:	error xx05, NEGATIVE HARDWARE LIMIT EXCEEDED
	out of range:	error xx06, POSITIVE SOFTWARE LIMIT EXCEEDED
	out of range:	error xx07, NEGATIVE SOFTWARE LIMIT EXCEEDED

DESCRIPTION

This command initiates an absolute motion. When received, the selected axis **xx** will move, with the predefined acceleration and velocity, to the absolute position specified by **nn**. If the requested axis is member of a group, this command does not initiate the desired motion. Instead, error xx31, " COMMAND NOT ALLOWED DUE TO GROUP ASSIGNMENT" is generated. Refer HL and HC commands to move along a line or an arc.

If this command is issued when trajectory mode for this axis is not in trapezoidal or s-curve mode, the controller returns error xx32, “INVALID TRAJECTORY MODE FOR MOVING”.

Note:

Even though the command is accepted while a motion is in progress, care should be taken not to reverse direction of motion. When this command is received, the controller verifies if it will produce a change of direction.

RETURNS

If the “?” sign takes the place of **nn** value, this command reports the current actual the same as TP?.

REL. COMMANDS

AC	-	set acceleration
PR	-	move to relative position
ST	-	stop motion
MD	-	move done status
VA	-	set velocity

EXAMPLE

3VA8 | *set velocity of axis #2 to 8 units / s*

3PA12.34 | *move axis #2 to absolute position 12.34*

PC set position compare mode

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **xxPCnn$_1$, nn$_2$**

PARAMETERS

Description	**xx** [int]	-	axis number
	nn1 [int]	-	compare modes
	nn2 [float]	-	compare absolute position or relative displacement
Range	**xx**	-	**1** to **2**
	nn1	-	**0** to **2** where 0 = disarm position compare feature 1 = arm absolute position compare feature 2 = arm relative position compare feature
	nn2	-	any position within the travel limits
Units	**xx**	-	none
	nn1	-	none
	nn2	-	defined motion units

Defaults	**xx**	missing:	error 37, AXIS NUMBER MISSING
		out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn1	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error xx1, PARAMETER OUT OF RANGE
	nn2	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error xx1, PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to configure the position compare mode. In this mode the controller is configured to output a TTL signal at precise position crossing(s). And because this feature is implemented in hardware (and not with CPU 'polling') the timing latency is negligible and can operate precisely even while the stage/positioner is moving at top speed. This feature can be used to accurately trigger an external device (e.g., data acquisition board) as a function of stage position -- regardless of travel speed.

If parameter **nn1** equals 0 then the position compare feature is in "disarmed" mode and no comparison will take place.

If **nn1** equals 1 then **absolute position** compare mode is "armed." During this mode the controller will output a TTL trigger signal whenever the stage feedback encoder crosses that absolute position define by **nn2**.

If **nn1** equals 2 then **relative position** compare mode is "armed." During this mode the controller will output a TTL trigger signal whenever the stage feedback encoder displaces **nn2** increments regardless of direction.

Note: If this command is issued when the desired axis is in motion, the accuracy of the position crossing capture depends upon the speed at which the axis was moving when the command was processed by the DSP. For instance, if the axis was moving at 40 mm/sec, and the encoder resolution is 10000 counts/mm, the **worst** case error in capturing a position crossing would be approximately 3 counts. Please see calculation below:

Worst case error	=	(Axis speed at setup time) * 7 micro seconds
	=	40 mm/sec * 10000 counts/mm * 7 micro seconds
	=	2.8 counts

However, if this command was issued when the axis is at standstill, the worst case error in capturing a position crossing is within ± 1 encoder count.

Note: The position compare hardware feature is only supported on ESP6000 and ESP7000 controllers axes 1 and 2.

Note: This feature is implemented using the auxiliary counter channels 7 and 8 in conjunction with axes counters 1 and 2. Therefore, if either axis 1 or 2 is configured in any compare mode then auxiliary counters channel 7 and 8 are used and not available. (e.g., cannot be used for trackball or master/slave modes) Counters 7 & 8 are released and made available when the compare mode is disarmed (e.g., “x PC 0”)

Note: A detailed TTL output timing diagram is available in the Advanced Capabilities section of the User’s Manual.

Note: Please find TTL compare output signal designations and electrical specifications in the Connector Pin Assignment Appendix section of the User’s Manual.

RETURNS

If the "?" sign takes the place of **nn1** value, this command reports the current assignment.

REL. COMMANDS

PA	-	Move to absolute position
PR	-	Move to relative position
WP	-	Wait for position

EXAMPLE #1 The following program will home (initialize) axis-1 and move to an absolute target position. Before the motion is started the controller will be configured to generate a single, precise TTL pulse when absolute position +12.34 is crossed.

```
1EP             // start program entry mode
1MO             // enable axis-1 motor power
1 OR 1          // home axis-1
1WS1000         // wait for home completion and dwell 1 second
1 PC 1, +12.34  // arm absolute position compare trigger output pulse at +12.34
1 PA +15.0      // start absolute position move to location +15.0
1 WS 0          // wait for home completion
1 PC 0, 0       // disarm compare trigger output pulse mode
QP              // end program entry mode
```

EXAMPLE #2

The following program will home (initialize) axis-1 and move to an absolute target position. Before the motion is started the controller will be configured to generate a precise TTL pulse every +1.00 units of relative distance.

```
2EP          // start program entry mode
1MO          // enable axis-1 motor power
1 OR 1       // home axis –1
1 WS 1000    // wait for home completion and dwell 1 second
1 PC2, +1.00 // arm relative position compare trigger pulse every +1.00 units
1 PA +15.0   // start absolute position move to location +15.0
1 WS 0       // wait for home completion
1 PC 0, 0    // disarm compare trigger output pulse mode
QP           // end program entry mode
```

PH get hardware status

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **PH**

PARAMETERS None

DESCRIPTION This command is used to get general hardware status for all axes. This routine allows user to observe the various digital input signals as they appear to the controller.

HARDWARE STATUS REGISTER #1

BIT#	VALUE	DEFINITION
0	0	axis 1 +hardware travel limit low
0	1	axis 1 +hardware travel limit high
1	0	axis 2 +hardware travel limit low
1	1	axis 2 +hardware travel limit high
2	0	axis 3 +hardware travel limit low
2	1	axis 3 +hardware travel limit high
3	0	axis 4 +hardware travel limit low
3	1	axis 4 +hardware travel limit high
4	0	axis 5 +hardware travel limit low
4	1	axis 5 +hardware travel limit high
5	0	axis 6 +hardware travel limit low
5	1	axis 6 +hardware travel limit high
6	0	reserved
6	1	reserved
7	0	reserved
7	1	reserved
8	0	axis 1 –hardware travel limit low
8	1	axis 1 –hardware travel limit high
9	0	axis 2 –hardware travel limit low
9	1	axis 2 –hardware travel limit high
10	0	axis 3 –hardware travel limit low
10	1	axis 3 –hardware travel limit high
11	0	axis 4 –hardware travel limit low
11	1	axis 4 –hardware travel limit high
12	0	axis 5 –hardware travel limit low
12	1	axis 5 –hardware travel limit high
13	0	axis 6 –hardware travel limit low
13	1	axis 6 –hardware travel limit high
14	0	reserved
14	1	reserved
15	0	reserved
15	1	reserved

16	0	axis 1 amplifier fault input low
16	1	axis 1 amplifier fault input high
17	0	axis 2 amplifier fault input low
17	1	axis 2 amplifier fault input high
18	0	axis 3 amplifier fault input low
18	1	axis 3 amplifier fault input high
19	0	axis 4 amplifier fault input low
19	1	axis 4 amplifier fault input high
20	0	axis 5 amplifier fault input low
20	1	axis 5 amplifier fault input high
21	0	axis 6 amplifier fault input low
21	1	axis 6 amplifier fault input high
22	0	reserved
22	1	reserved
23	0	reserved
23	1	reserved
24	0	reserved
24	1	reserved
25	0	reserved
25	1	reserved
26	0	reserved
26	1	reserved
27	0	100-pin emergency stop (unlatched) low
27	1	100-pin emergency stop (unlatched) high
28	0	auxiliary I/O emergency stop (unlatched) low
28	1	auxiliary I/O emergency stop (unlatched) high
29	0	100-pin connector emergency stop (latched) low
29	1	100-pin connector emergency stop (latched) high
30	0	auxiliary I/O conector emergency stop (latched) low
30	1	auxiliary I/O conector emergency stop (latched) high
31	0	100-pin cable interlock low
31	1	100-pin cable interlock high

HARDWARE STATUS REGISTER #2

BIT #	VALUE	DEFINITION
0	0	axis 1 home signal low
0	1	axis 1 home signal high
1	0	axis 2 home signal low
1	1	axis 2 home signal high
2	0	axis 3 home signal low
2	1	axis 3 home signal high
3	0	axis 4 home signal low
3	1	axis 4 home signal high
4	0	axis 5 home signal low
4	1	axis 5 home signal high
5	0	axis 6 home signal low
5	1	axis 6 home signal high

6	0	reserved
6	1	reserved
7	0	reserved
7	1	reserved
8	0	axis 1 index signal low
8	1	axis 1 index signal high
9	0	axis 2 index signal low
9	1	axis 2 index signal high
10	0	axis 3 index signal low
10	1	axis 3 index signal high
11	0	axis 4 index signal low
11	1	axis 4 index signal high
12	0	axis 5 index signal low
12	1	axis 5 index signal high
13	0	axis 6 index signal low
13	1	axis 6 index signal high
14	0	reserved
14	1	reserved
15	0	reserved
15	1	reserved
16	0	digital input A low
16	1	digital input A high
17	0	digital input B low
17	1	digital input B high
18	0	digital input C low
18	1	digital input C high
		• • •
31	0	reserved
31	1	reserved

RETURNS **T**his command reports the current status in hexadecimal notation.

REL. COMMANDS

ZU	-	get ESP system configuration
ZZ	-	get system configuration

EXAMPLE

PH	| *read hardware status*
18000404H, 4H	| *controller returns the status of the two hardware*
	| *regsisters*

PR move to relative position

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**PR**nn

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	relative motion increment
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	any value that will not cause exceeding the software limits and within 2e9 * encoder resolution.
Units	**xx**	-	none
	nn	-	defined motion units

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx04, POSITIVE HARDWARE LIMIT EXCEEDED
	out of range:	error xx05, NEGATIVE HARDWARE LIMIT EXCEEDED
	out of range:	error xx06, POSITIVE SOFTWARE LIMIT EXCEEDED
	out of range:	rror xx07, NEGATIVE SOFTWARE LIMIT EXCEEDED

DESCRIPTION

This command initiates a relative motion. When received, the selected axis **xx** will move, with the predefined acceleration and velocity, to a relative position **nn** units away from the current position. If the requested axis is member of a group, this command does not initiate the desired motion. Instead, error xx31, "COMMAND NOT ALLOWED DUE TO GROUP ASSIGNMENT" is generated. Refer HL and HC commands to move along a line or an arc.

If this command is issued when trajectory mode for this axis is not in trapezoidal or s-curve mode, the controller returns error xx32, “INVALID TRAJECTORY MODE FOR MOVING”.

Note:
Even though the command is accepted while a motion is in progress, care should be taken not to reverse direction of motion.

RETURNS none

REL. COMMANDS

AC	-	set acceleration
PA	-	move to absolute position
MD	-	move done status
ST	-	stop motion
VA	-	et velocity

EXAMPLE

3VA8	*| set velocity of axis # 3 to 8 units / s*
3PR2.34	*| move axis # 3 2.34 units away from the current position*

QD update motor driver settings

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**QD**

PARAMETERS

Description			
	xx [int]	-	axis number
Range	**xx**	-	**1** to **MAX AXES**
Units	**xx**	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	missing Unidrive:		error xx23, UNIDRIVE NOT DETECTED

DESCRIPTION This command is used to update Newport programmable driver (i.e., Unidrive) settings into working registers.

Note: This command should not be issued during motion since the motor power is automatically turned OFF.

RETURNS none

REL. COMMANDS

QS	-	set microstep factor
QG	-	set gear constant
QT	-	set tachometer gain
QV	-	set average motor voltage

EXAMPLE

2QI?	| *read maximum motor current setting of axis # 2*
1.6	| *controller returns a value of 1.6 Amp. for axis #2*
2QI 1.2	| *set maximum motor current to 1.2Amp. for axis #2*
2QD	| *update programmable driver with latest settings for axis #2*
SM	| *save all controller settings to non-volatile memory*

QG set gear constant

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**QG**nn or xx**QG**?

PARAMETERS

Description			
	xx [int]	-	axis number
	nn [float]	-	gear constant
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **2e9**
			or **?** to read present setting
Units	**xx**	-	none
	nn	-	evolution / unit of measure
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to set the gear constant for a Newport Unidrive compatible programmable driver for DC servo axis. This command should be used in conjunction with **QT** (tachometer gain) command.
The gear constant is defined as the number of revolutions the motor has to make for the motion device to move one displacement.
This command must to be followed by the **QD** update driver command to take affect.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

SN	-	set displacement units
QD	-	update driver
QS	-	set microstep factor
QI	-	set motor maximum current
QV	-	et average motor voltage

EXAMPLE

2QG?	| *read gear constant setting of axis # 2*
0.3937	| *controller returns a value of 0.3937 rev / unit for axis #2*
2QG 0.25	| *set gear constant to 0.25 rev / unit for axis #2*
2QT 7.0	| *set tachometer gain to 7 V/Krpm for axis #2*
2QD	| *update programmable driver with latest settings for axis #2*

QI set maximum motor current

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**QI**nn or xx**QI**?

PARAMETERS

Description			
	xx [int]	-	axis number
	nn [float]	-	motor current
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to maximum driver rating (see Specifications section) or **?** to read present setting
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to set the maximum motor current output for a Newport Unidrive compatible programmable driver axis. This command must to be followed by the **QD** update driver command to take affect.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

QG	-	set gear constant
QD	-	update driver
QS	-	set microstep factor
QT	-	set tachometer gain
QV	-	set average motor voltage

EXAMPLE

2QI?	| *read maximum motor current setting of axis # 2*
1.6	| *controller returns a value of 1.6 Amp. for axis #2*
2QI 1.2	| *set maximum motor current to 1.2Amp. for axis #2*
2QD	| *update programmable driver with latest settings for axis #2*
SM	| *save all controller settings to non-volatile memory*

QM set motor type

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**QM**nn or xx**QM**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	motor type
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **4** where 0 = motor type undefined (default) 1 = DC servo motor (single analog channel) 2 = step motor (digital control)* 3 = commutated step motor (analog control)+ 4 = commutated brushless DC servo motor or **?** to read current setting
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to set the motor type the for axis **xx**. Defining motor type is necessary because the ESP needs to apply different control algorithms for different motor types.

Note:
It will not be possible to control an axis if its motor type is undefined.

* **ESP300** motion controller does not support this motor type.
+ **ESP6000** and **ESP7000** motion controllers do not support this motor type.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

QV	-	set average motor voltage
QD	-	update driver
QI	-	set maximum motor current
QT	-	set tachometer gain
QG	-	et gear constant

EXAMPLE

2QM?	| *read motor type of axis # 2*
0	| *controller returns a value of 0 (motor undefined) for axis #2*
2QM 1	| *set motor type to value of 1 (DC servo motor) for axis #2*
2QD	| *update programmable driver with latest settings for axis #2*
SM	| *save all controller settings to non-volatile memory*

QP quit program mode

	IMM	PGM	MIP
USAGE	♦		

SYNTAX **QP**

PARAMETERS None

DESCRIPTION This command quits the controller from programming mode. All the commands following this one will be executed immediately.

RETURNS none

REL. COMMANDS

EX	-	execute stored program
AP	-	abort stored program execution
XX	-	erase program

EXAMPLE

```
3XX          | clear program 3 from memory, if any
3EP          | activate program mode and enter following commands as
             | program 3
    .
    .
    .
QP           | end entering program and quit programming mode
3EX          | run stored program number 3
```

QR reduce motor torque

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**QR**nn1,nn2 or xx**QR**?

PARAMETERS

Description

xx	[int]	-	axis number
nn1	[int]	-	delay period
nn2	[float]	-	motor current reduction percentage

Range

xx	-	**1** to **MAX AXES**
nn1	-	**0** to **60000**
nn2	-	**0** to **100**

Units

xx	-	none
nn1	-	milliseconds
nn2	-	percent of max. motor current

Defaults

xx missing:	error 37, AXIS NUMBER MISSING
missing parameter:	error 38, COMMAND PARAMETER MISSING
out of range:	error 9, AXIS NUMBER OUT OF RANGE

DESCRIPTION

This command automatically reduces the specified step motor's current (i.e., torque) output to the requested percentage **nn2** after motion has stopped and the specified time **nn1** has expired. The purpose of this command is to help reduce the motor heating typically generated by stepper motors. If **xx** is equal to 0, the torque reduction parameters get applied to all axes.

Note

This command does not affect DC servo motors and pulse stepper motors.

Note

Time parameter **nn1** is only effective for ESP300 motion controller

RETURNS

If "?" sign is issued along with command, the controller returns the torque reduction settings for the specified axis.

REL. COMMANDS

QM	-	set motor type
QI	-	et motor current

EXAMPLE

2QR1000,50	| *reduce motor #2 torque to 50%, 1000 msec after a move done*
2QR?	**|** *query motor #2 torque reduction settings*
1000,50	**|** *controller returns 1000 msec and 50%*

QS set microstep factor

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**QS**nn or xx**QS**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	microstep value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**1** to **250** for step motors **1** to **1000** for commutated step motors or **?** to read current setting
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to set the microstep factor for a Newport Unidrive compatible programmable driver with step motor axis. This command must be followed by the **QD** update driver command to take affect.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

QD	-	update driver
QI	-	set maximum motor current
QT	-	set tachometer gain
QG	-	set gear constant
QV	-	set average motor voltage

EXAMPLE

2QS?	| *read microstep factor of axis # 2*
100	| *controller returns a value of 100 for axis #2*
2QS 250	| *set microstep factor to 250 for axis #2*
2QD	| *update programmable driver with latest settings for axis #2*
SM	| *save all controller settings to non-volatile memory*

QT set tachometer gain

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**QT**nn or xx**QT**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	tachometer gain
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **20**
			or **?** to read present setting
Units	**xx**	-	none
	nn	-	Volts/Krpm
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to set the DC motor tachometer gain for a Newport Unidrive compatible programmable driver axis.
This command should be used in conjunction with **QG** (gear constant) command.
This command must to be followed by the **QD** update driver command to take affect.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

QD	-	update driver
QS	-	set microstep factor
QG	-	set gear constant
QI	-	set motor maximum current
QV	-	set average motor voltage

EXAMPLE

2QT?	*read tachometer gain setting of axis # 2*
7.0	*controller returns a value of 7.0 V/Krpm for axis #2*
2QT 6.5	*set tachometer gain value of 6.5 V/Krpm for axis #2*
2QG 0.3937	*set gear constant to 0.3937 rev / unit for axis #2*
2QD	*update programmable driver with latest settings for axis #2*
SM	*save all controller settings to non-volatile memory*

QV set average motor voltage

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**QV**nn or xx**QV**?

PARAMETERS

Description			
	xx [int]	-	axis number
	nn [float]	-	motor voltage
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to maximum driver rating (see Specifications section) or **?** to read present setting
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to set the average motor voltage output for a Newport Unidrive compatible programmable driver axis. This command must to be followed by the **QD** update driver command to take affect.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

QD	-	update driver
QI	-	set maximum motor current
QG	-	set gear constant
QS	-	set microstep factor
QT	-	set tachometer gain

EXAMPLE

2QV?	| *read average motor voltage setting of axis # 2*
48.0	| *controller returns a value of 48Volts for axis #2*
2QV 12	| *set average motor voltage to 12 Volts for axis #2*
2QD	| *update programmable driver with latest settings for axis #2*
SM	| *save all controller settings to non-volatile memory*

RA read analog input

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX xx**RA**

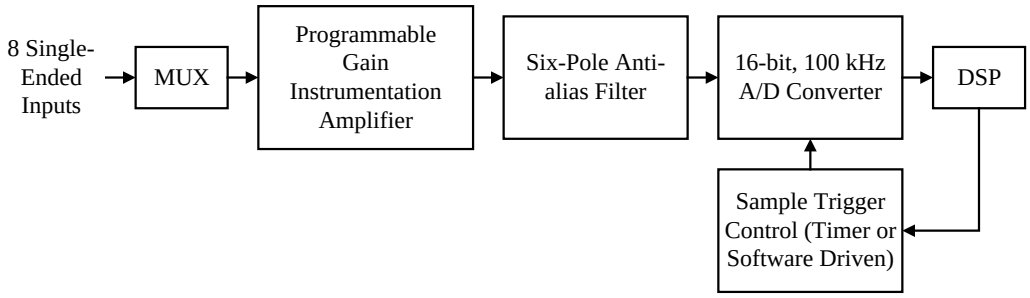
PARAMETERS

Description			
	xx [int]	-	ADC channel number
Range	**xx**	-	**0** to **8**
Units	**xx**	-	none
Defaults	**xx** missing:		0
	out of range:		error 7, PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to read the analog-to-digital converter channel (1-8) specified. If xx is missing or set to 0, the controller returns the values found in all eight ADC converters.

ADC channels are located on the analog I/O connector on the controller card. The following block diagram illustrates the implementation of analog-to-digital conversion in ESP controller.



RETURNS read ADC channel

REL. COMMANDS **AM** - set analog input mode

EXAMPLE

1RA | *read the value of ADC channel #1*
2.3456, 12456 | *controller returns a value of 2.3456V and servo clock tick number when conversion was done*
0RA | *read the value of all eight ADC channels*
2.3456, 3.2240, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 224578 | *controller returns values of all ADC channels and servo clock tick number when conversion was done*

RQ generate service request (SRQ)

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **RQ**nn

PARAMETERS

Description	**nn** [int]	-	interrupt number
Range	**nn**	-	**0** to **31**
Units	**nn**	-	none
Defaults	**nn**	missing:	0
		out of range:	error 7, PARAMETER OUT OF RANGE

DESCRIPTION This command generates an interrupt service request to the host computer. The parameter **nn** is used to identify the RQ command which generated the interrupt. Upon receiving the interrupt, the host computer interrupt service routine should perform an IEEE 488 serial poll. If the interrupt was as a result of the **RQ** command, then bit 6 of the response is 1 and the lower five bits equal the parameter **nn**.

This command can be used to notify the host computer of the progress or flow of command execution in the motion controller.

RETURNS None

REL. COMMANDS **SA** - set device address

EXAMPLE 2PR200;2WS;1PR100;1WS;**RQ3** | *generate interrupt when RQ command is encountered and set bit 0 and 2*

RS reset the controller

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **RS**

PARAMETERS None

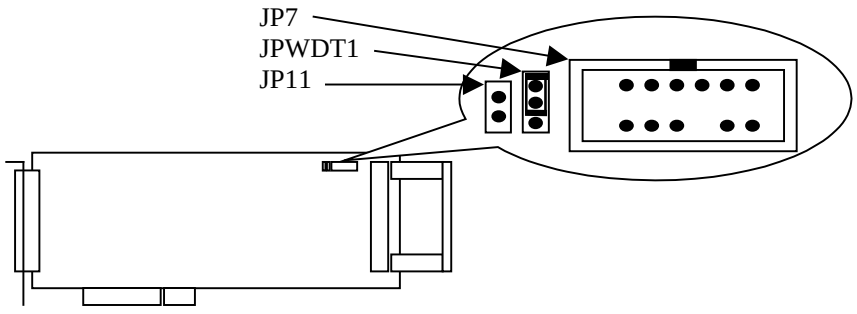
DESCRIPTION This command is used to perform a hardware reset of the controller. It performs the following preliminary tasks before resetting the controller:

1. Stop all the axes that are in motion. The deceleration value specified using the command **AG** is used to stop the axes.
2. Wait for 500 ms to allow the axes to settle.
3. Disable all the axes by turning the power OFF.
4. Reset to the controller card.

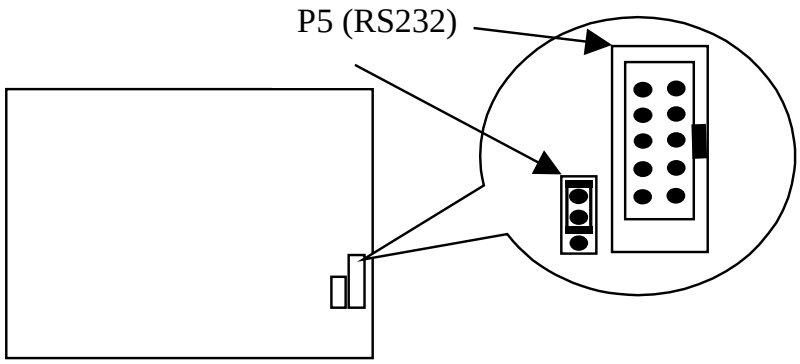
Once the command to reset the controller is detected by the DSP, the controller will stay in reset for a minimum of 200 ms. After the reset condition has occurred (i.e., after the 200 ms reset time), the controller firmware reboots the controller. At this point, all the parameters last saved to the non-volatile flash memory on the controller will be restored. Furthermore, the controller will detect any stages (ESP compatible or otherwise) and drivers connected to the controller. This process can take anywhere up to 20 seconds depending upon the controller configuration.

NOTE: This command is affective only when the watchdog timer is enabled through appropriate jumper setting on the controller card (default factory setting is "enabled"). The following figure illustrates the jumper settings to enable the watchdog timer.

NOTE: If this command is issued over the PCI bus interface (as in the case of ESP6000 motion controller), all communication between the host PC and the controller over this bus will be interrupted. As a result, **this command is not recommended for use over the PCI bus interface**. Use the binary command, esp_init_system( ) instead.**CAUTION:** Use this command judiciously! It is not intended to be a substitute for an e-stop condition.



*For **ESP6000** and **ESP7000** motion controllers*



*For **ESP100** and **ESP300** motion controllers*

RETURNS None

REL. COMMANDS None

EXAMPLE **RS** | *Reset the controller*

SA set device address

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **SA**nn or **SA?**

PARAMETERS

Description	**nn** [int]	-	address number
Range	**nn**	-	**1** to **30**
Units	**nn**	-	none
Defaults	**nn** missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error 7, PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to set and report the device (i.e., ESP controller) address for use with IEEE-488 or USB communications (if equipped).
The address change takes affect immediately after the command is processed.

Note:
Use the SM command to save new address setting to non-volatile memory.

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS

none

EXAMPLE

SA 3	| *set device address to 3*
SA ?	| *read present device address setting*
3	| *controller returns device address #3*
SM	| *save all settings to non-volatile memory*

SB set / get DIO port A, B, C bit status

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **SB**nn or **SB**?

PARAMETERS
Description

nn [int] - hardware limit configuration

Range

nn - **0** to **0FFFFFFH** (hexadecimal with leading zero(0))
or **?** to read current setting

Units

nn - None

Defaults

nn missing: error 38, COMMAND PARAMETER MISSING
out of range: error 7, PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to either set all digital I/O (DIO) port A, B, and C logic level or read its present status. Bits 0-7 correspond to port A, bits 8-15 to port B, and bits 16-23 to port C. Each 8-bit port can be set as either input or output with the **BO** command.

A DIO within a port configured as an input can only report its present HIGH or LOW logic level. Whereas a DIO bit within a port configured as an output can set(1) or clear(0) the corresponding DIO hardware to HIGH or LOW logic level. Reading the status of a port configured as output returns its present output status.

NOTE: All direction bits are automatically zeroed, or cleared, after a system reset. Therefore all DIO ports default to input by default.

NOTE: Each DIO bit has a pulled-up resistor to +5V. Therefore, all bits will be at HIGH logic level if not connected to external circuit and configured as input.

BIT#	VALUE	DEFINITION
0	0	port A bit-0 at logic level 0 (LOW)
*0	1	port A bit-0 at logic level 1 (HIGH)
1	0	port A bit-1 at logic level 0 (LOW)
*1	1	port A bit-1 at logic level 1 (HIGH)
2	0	port A bit-2 at logic level 0 (LOW)
*2	1	port A bit-2 at logic level 1 (HIGH)
3	0	port A bit-3 at logic level 0 (LOW)
*3	1	port A bit-3 at logic level 1 (HIGH)
4	0	port A bit-4 at logic level 0 (LOW)
*4	1	port A bit-4 at logic level 1 (HIGH)

5	0	port A bit-5 at logic level 0 (LOW)
*5	1	port A bit-5 at logic level 1 (HIGH)
6	0	port A bit-6 at logic level 0 (LOW)
*6	1	port A bit-6 at logic level 1 (HIGH)
7	0	port A bit-7 at logic level 0 (LOW)
*7	1	port A bit-7 at logic level 1 (HIGH)
		• • •
23	0	port C bit-23 at logic level 0 (LOW)
*23	1	port C bit-23 at logic level 1 (HIGH)
		* default setting

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting in hexadecimal notation.

REL. COMMANDS

BO - set DIO port direction

EXAMPLE

BO? | *read DIO port direction configuration*
0H | *controller returns a value of 0H (all ports are input)*
BO 1H | configure *DIO port A as output*
SB 0FFH | *set all port A DIO output HIGH*

SH set home preset position

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**SH**nn or xx**SH?**

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	home preset position
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	any position within the travel limits
Units	**xx**	-	none
	nn	-	defined motion units

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx01, PARAMETER OUT OF RANGE

DESCRIPTION This command defines the value that is loaded in the position counter when home is found. The default value for all motion devices is 0. This means that unless a new value is defined using this command, the home position will be set to 0 when a home search is initiated using the OR command or from the front panel (if available).

Note:
The change takes effect only when a subsequent home search routine is performed. To make the change permanent, use the SM command to save it in the non-volatile memory.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting.

REL. COMMANDS **DH** - define home

EXAMPLE

3MO	|	*turn axis #3 motor power ON*
3SH75.0	|	*set axis #3 home position to 75.0 units*
3OR1	|	*perform a home search on axis # 3*
3MD?	|	*query axis #3 motion status*
1	|	*controller returns a value of 1, when motion is done*
3TP	|	*query axis #3 position*
75.0	|	*controller returns a value of 75.0 units*

SI set master-slave jog velocity update interval

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **SI**nn or **SI**?

PARAMETERS

Description	**nn** [int]	-	jog velocity update interval
Range	**nn**	-	**1** to **1000**
Units	**nn**	-	milliseconds
Defaults	**nn** missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error 7, PARAMETER OUT OF RANGE

DESCRIPTION This command sets the jog velocity update interval for slave axis. The jog velocity of slave axis is computed once every interval using user specified scaling coefficients and the master axis velocity at the time of computation. Refer **SK** command to specify slave jog velocity scaling coefficients. Note that appropriate trajectory mode has to be specified using **TJ** command before this command becomes effective.

RETURNS If "?" sign is issued along with command, the controller returns slave axis jog velocity update interval.

REL. COMMANDS

SS	-	define master-slave relationship
SK	-	set slave axis jog velocity scaling coefficients

EXAMPLE

2SS1	*| set axis 2 to be the slave of axis 1*
2SS?	*| query the master axis number for axis 2*
1	*| controller returns a value of 1*
2TJ6	*| set axis 2 trajectory mode to 6*
SI10	*| set the jog velocity update interval of slave axis to 10 msec*
SI?	*| query the jog velocity update interval of slave axis*
10	*| controller returns a value of 10*
SK0.5,0	*| set the jog velocity scaling coefficients to 0.5 and 0*
SK?	*| query the jog velocity scaling coefficients*
0.5,0	*| controller returns 0.5 and 0*

SK set master-slave jog velocity scaling coefficients

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **SK**nn1, nn2 or **SK**?

PARAMETERS

Description	**nn$_i$** [float]	-	jog velocity scaling coefficients
Range	**nn$_i$**	-	none
Units	**nn$_i$**	-	none
Defaults	**nn$_i$**	missing:	error 38, COMMAND PARAMETER MISSING

DESCRIPTION This command sets the jog velocity scaling coefficients for slave axis. The jog velocity of slave axis is computed once every interval using user specified scaling coefficients and the master axis velocity at the time of computation. The user specified coefficients are used as follows:

$$\dot{x}_s = A\dot{x}_m + B\dot{x}_m^2 \ \text{sgn}(\dot{x}_m)$$

where $\dot{x}_s$ is the jog velocity of the slave and $\dot{x}_m$ is the velocity of the master axis.
Refer **SI** command to specify slave jog velocity update interval.

Note: Appropriate trajectory mode has to be specified using **TJ** command before this command becomes effective.

RETURNS If "?" sign is issued along with command, the controller returns slave axis jog velocity scaling coefficients.

REL. COMMANDS

SS	-	define master-slave relationship
SI	-	set slave axis jog velocity update interval

EXAMPLE

2SS1	*| set axis 2 to be the slave of axis 1*
2SS?	*| query the master axis number for axis 2*
1	*| controller returns a value of 1*
2TJ6	*| set axis 2 trajectory mode to 6*
SI10	*| set the jog velocity update interval of slave axis to 10 msec*
SI?	*| query the jog velocity update interval of slave axis*
10	*| controller returns a value of 10*
SK0.5,0	*| set the jog velocity scaling coefficients to 0.5 and 0*
SK?	*| query the jog velocity scaling coefficients*
0.5,0	*| controller returns 0.5 and 0*

SL set left travel limit

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**SL**nn or xx**SL**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	left (negative) software limit
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**-2e9** ∗ **encoder resolution** to 0
Units	**xx**	-	none
	nn	-	predefined motion units

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx2, PARAMETER OUT OF RANGE

DESCRIPTION

This command defines the value for the negative (left) software travel limit. It should be used to restrict travel in the negative direction to protect the motion device or its load. For instance, if traveling full range, a stage could push its load into an obstacle. To prevent this, the user can reduce the allowed travel by changing the software travel limit.
Since a motion device must be allowed to find its home position, the home switch and/or sensor must be inside the travel limits. This means that both positive and negative travel limits cannot be set on the same side of the home position. A more obvious restriction is that the negative limit cannot be greater than the positive limit. If any of these restrictions is not respected, the controller will return PARAMETER OUT OF RANGE.

Note:
If the command is issued for an axis in motion, the new limit should not be set inside the current travel.

Note:
Be careful when using this command. The controller does not know the real hardware limits of the motion device. Always set the software limits inside the hardware limits (limit switches). In normal operation, a motion device should never hit a limit switch.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS	**OR**	-	search for home
	SR	-	set right software limits

EXAMPLE **1SL41.4** | *set negative travel limit of axis #1 to 41.4 units*

SM save settings to non-volatile memory

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX **SM**

PARAMETERS none

DESCRIPTION This command is used to save system and axis configuration settings from RAM to non-volatile flash memory. It should be used after modifying system and/or axis parameters and settings to assure that the new data will not be lost when the controller is powered off.

Note:

User programs created with EP command are automatically saved to non-volatile memory.

RETURNS none

REL. COMMANDS none

EXAMPLE

```
3VA12.5      | set axis 3 velocity to 12.5 units/sec
3AC50.0      | set axis 3 acceleration to 50 unit/sec^2
    .
    .
    .
SM           | save changes to non-volatile memory
```

SN set axis displacement units

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**SN**nn or xx**SN**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	displacement units
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **10** where 0 = encoder count, 1 = motor step, 2 = millimeter, 3 = micrometer, 4 = inches, 5 = milli-inches, 6 = micro-inches, 7 = degree, 8 = gradian, 9 = radian, 10 = milliradian, 11 = microradian or **?** to read present setting
Units	**xx**	-	none
	nn	-	none

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to set the displacement units for the for axis **xx**.
Note:
The unit of measure as used with this controller is intended as a label only. It is the user's responsibility to convert and resend all affected parameters (e.g., velocity, acceleration, etc…) when switching from one unit of measure to another.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS	**FR**	-	set full-step resolution
	SU	-	set encoder resolution

EXAMPLE

2SN	| *read displacement unit setting of axis # 2*
2	| *controller returns a value 2 (millimeter) for axis #2*
2SN 0	| *set displacement unit to 0 (encoder count) for axis #2*

SR set right travel limit

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**SR**nn or xx**SR**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	right (positive) software limit
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**+2e9** ∗ **encoder resolution** to 0
Units	**xx**	-	none
	nn	-	defined motion units
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION

This command defines the value for the positive (right) software travel limit. It should be used to restrict travel in the positive direction to protect the motion device or its load. For instance, if traveling full range, a stage could push its load into an obstacle. To prevent this, the user can reduce the allowed travel by changing the software travel limit.
Since a motion device must be allowed to find its home position, the home switch and/or sensor must be inside the travel limits. This means that both positive and negative travel limits cannot be set on the same side of the home position. A more obvious restriction is that the negative limit cannot be greater than the positive limit. If any of these restrictions is not respected, the controller will return PARAMETER OUT OF RANGE

Note:
If the command is issued for an axis in motion, the new limit should not be set inside the current travel.

Note:
Be careful when using this command. The controller does not know the real hardware limits of the motion device. Always set the software limits inside the hardware limits (limit switches). In normal operation, a motion device should never hit a limit switch.

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

OR	-	search for home
SL	-	set left software limit

EXAMPLE **1SR41.4** | *set positive travel limit of axis #1 to 41.4 units*

SS define master-slave relationship

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**SS**nn or xx**SS**?

PARAMETERS

Description	**xx** [int]	-	axis number to be defined as a slave
	nn [int]	-	axis number to be defined as a master
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**1** to **MAX AXES**
Units	**xx**	-	none
	xx	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE

DESCRIPTION

This command defines master-slave relationship between any two axes. A few rules are in place for ease of use.

- The trajectory mode for slave has to be appropriately defined before that axis follows master in a desired fashion.
- An axis cannot be assigned as its own slave if it is already in a trajectory mode that is specific to master-slaving.
- A slave axis cannot be moved individually using **PA** or **PR** commands if its trajectory mode is specific to master-slaving.

This command gets executed immediately, and can also be called from within a program.

RETURNS

If "?" sign is issued along with command, the controller returns master axis number.

REL. COMMANDS

TJ	-	set trajectory mode
GR	-	set master-slave reduction ratio

EXAMPLE

2SS1	| *set axis 2 to be the slave of axis 1*
2SS?	| *query the master axis number for axis 2*
1	**|** *controller returns a value of 1*
2TJ5	**|** *set axis 2 trajectory mode to 5*
2GR1.0	**|** *set the reduction ratio of axis 2 to 1.0*
1MO	**|** *turn axis 1 motor power ON*
2MO	**|** *turn axis 2 motor power ON*
1PA10	**|** *move axis 1 to absolute 10 units*
2PA20	**|** *move axis 2 to absolute 10 units*
TB	**|** *read error messages*
232, 242000, AXIS-2 INVALID TRAJECTORY MODE FOR MOVING	
	| controller returns appropriate error message

ST stop motion

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**ST**

PARAMETERS

Description			
	xx [int]	-	axis number
Range	**xx**	-	**1** to **MAX AXES**
Units	**xx**	-	none
Defaults	**xx**	out of range:	error 9, AXIS NUMBER OUT OF RANGE

DESCRIPTION This command stops a motion in progress using deceleration rate programmed with **AG** (set deceleration) command on the specified axes. If the ST command is sent with no axis parameter, all axes are stopped.

RETURNS none

REL. COMMANDS

AB	-	abort motion
AG	-	set deceleration
MF	-	motor power off

EXAMPLE

```
2PA40      | move axis # 2 to absolute position 40
2ST        | stop motion on axis # 2
```

SU set encoder resolution

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**SU**nn or xx**SU**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	encoder resolution
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**2e-9** to **2e+9** in user defined units or **?** to read present setting
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command is used to set the encoder resolution for axis **xx**.
Note:
The encoder resolution can only be changed when encoder feedback is enabled. See ZB command.

RETURNS If "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

FR	-	set full-step resolution
SU	-	set encoder resolution
QD	-	update driver
ZB	-	set feedback configuration

EXAMPLE

2SU?	*read encoder resolution setting of axis # 2*
0.0001	*controller returns a value of 0.0001 units for axis #2*
2SU0.0005	*set encoder resolution to 0.0005 units for axis #2*
2QD	*update programmable driver with latest settings for axis #2*
SM	*save all controller settings to non-volatile memory*

TB read error message

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **TB**?

PARAMETERS questions mark (?)

Defaults

DESCRIPTION This command is used to read the error code, timestamp, and the associated message.
The error code is one numerical value up to three(3) digits long. (see Appendix for complete listing)
In general, non-axis specific errors numbers range from 1-99. Axis-**1** specific errors range from **1**00-**1**99, Axis-**2** errors range from **2**00-**2**99 and so on.

The timestamp is in terms of servo cycle ticks accumulated since the last System Reset, incrementing at the servo interrupt interval (400us default). The message is a description of the error associated with it. All arguments are separated by commas.

Note:
Errors are maintained in a FIFO buffer ten(10) elements deep. When an error is read using TB or TE, the controller returns the last error that occurred and the error buffer is cleared by one(1) element. This means that an error can be read only once, with either command.

RETURNS **aa, bb, cc**
where: **aa** = error code **cc** = error message
bb = timestamp (see Appendix for complete listing)

REL. COMMANDS **TE** - read error code

EXAMPLE

TB?	*| read error message*
	0, 451322, NO ERROR DETECTED
	| controller returns no error
8PA12.3	*| move axis #8 to position 12.3*
TB?	*| read error message*
	9, 451339, AXIS NUMBER NOT AVAILABLE
	| controller returns error code, timestamp, and description

TE read error code

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **TE?**

PARAMETERS questions mark (?)

Defaults timeout: error 2, RS-232 COMMUNICATION TIME-OUT

DESCRIPTION This command is used to read the error code.

The error code is one numerical value up to three digits long. (see Appendix for complete listing)
In general, non-axis specific errors numbers range from 1-99. Axis-**1** specific errors range from **1**00-**1**99, Axis-**2** errors range from **2**00-**2**99 and so on.

Note:
Errors are maintained in a FIFO buffer ten(10) elements deep. When an error is read using TB or TE, the controller returns the last error that occurred and the error buffer is cleared by one(1) element. This means that an error can be read only once, with either command.

RETURNS **aa**

where: **aa** = error code number (see Appendix for complete listing)

REL. COMMANDS **TB** - read error message

EXAMPLE

TE?	*read error message*
0	*controller returns no error*
8PA12.3	*move axis #8 to position 12.3*
TE?	*read error message*
9	controller returns error code 9 meaning incorrect axis number

TJ set trajectory mode

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**TJ**nn or xx**TJ**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	home mode
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**1** to **6** where 1 = trapezoidal mode 2 = s-curve mode 3 = jog mode 4 = slave to master's desired position (trajectory) 5 = slave to master's actual position (feedback) 6 = slave to master's actual velocity for jogging
Units	**xx**	-	none
	nn	-	none

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx2, PARAMETER OUT OF RANGE
	during motion:	error xx26, PARAMETER CHANGE NOT ALLOWED DURING MOTION

DESCRIPTION This command sets the trajectory mode **nn** on the axis specified by **xx.**

Changing trajectory during motion is not allowed. Change trajectory mode only when the axis is not moving.

If the requested axis is member of a group, the controller returns error xx31, "COMMAND NOT ALLOWED DUE TO GROUP ASSIGNMENT".

For a detailed description of motion profiles see the **Motion Control Tutorial** section.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS	**JK**	-	set s-curve jerk rate
	GR	-	set master/slave gear ratio

EXAMPLE	**1TJ?**	| *report current trajectory mode setting on axis #1*
	1	| *controller returns trajectory mode 1 (trapezoidal) for axis #1*
	1TJ2	| *set trajectory mode on axis #1 to 2 (s-curve)*

TP read actual position

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX xx**TP**

PARAMETERS

Description	**xx** [int] -	axis number
Range	**xx** -	**1** to **MAX AXES**
Units	**xx** -	none
Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE

DESCRIPTION This command is used to read the actual position. It returns the instantaneous real position of the specified axis.

RETURNS **nn** where: **nn** = actual position, in pre-defined units

REL. COMMANDS

PA	-	move to an absolute position
PR	-	move to a relative position
DP	-	read instantaneous desired position

EXAMPLE

3TP	| *read real position on axis # 3*
5.322	| controller returns real position 5.322 for axis # 3

TS read controller status

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **TS**

PARAMETERS None

DESCRIPTION This command is used to read the controller status byte. The byte returned is in the form of an ASCII character. The value of each bit in the status byte can be deduced after converting the ASCII character into a binary value. Each bit of the status byte represents a particular controller parameter, as described in the following table.
Note:
Please refer to the Appendix for a complete ASCII to binary conversion table.

INTERPRETATION OF LEFT MOST ASCII CHARACTER:

Bit #	Function	Meaning for	
		Bit LOW	Bit HIGH
0	Axis #1 motor state	Stationary	In motion
1	Axis #2 motor state	Stationary	In motion
2	Axis #3 motor state	Stationary	In motion
3	Axis #4 motor state	Stationary	In motion
4	Motor power of at least one axis	OFF	ON
5	Reserved	Default	—
6	Reserved	—	Default
7	Reserved	Default	—

INTERPRETATION OF RIGHT MOST ASCII CHARACTER:

Note: This ASCII character is returned only if the motion controller supports more than four (4) axes.

Bit #	Function	Meaning for	
		Bit LOW	Bit HIGH
0	Axis #5 motor state	Stationary	In motion
1	Axis #6 motor state	Stationary	In motion
2	Reserved	Default	—
3	Reserved	Default	—
4	Motor power of at least one axis	OFF	ON
5	Reserved	Default	—
6	Reserved	—	Default
7	Reserved	Default	—

RETURNS ASCII character representing the status byte.

REL. COMMANDS **TX** - read controller activity

EXAMPLE
TS | *read controller status*
[P | *controller returns characters [and P indicating axes 1, 2 and 4 are*
| *in motion, and motor power of at least one axis is ON.*

TV read actual velocity

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX xx**TV**

PARAMETERS

Description			
	xx [int]	-	axis number
Range	**xx**	-	**1** to **MAX AXES**
Units	**xx**	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE

DESCRIPTION This command is used to read the actual velocity of an axis. The command can be sent at any time but its real use is while motion is in progress.

RETURNS **nn,** where **nn** = actual velocity of the axis in pre-defined units.

REL. COMMANDS

PA	-	move to an absolute position
PR	-	move to a relative position

EXAMPLE

3TP?	|	*read position on axis # 3*
5.32	|	*controller returns position 5.32 units for axis # 3*
3PR2.2	|	*start a relative motion of 2.2 units on axis # 3*
3DV	|	*read desired velocity on axis #3*
0.2	|	*controller returns velocity 0.2 units/sec for axis #3*
3TV	|	*read actual velocity on axis #3*
0.205	|	*controller returns velocity 0.205 units/sec for axis #3*
3DP?	|	*read desired position on axis # 3*
7.52	|	*controller returns desired position 7.52 units for axis # 3*

TX read controller activity

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **TX**

PARAMETERS None

DESCRIPTION This command is used to read the controller activity register. The byte returned is in the form of an ASCII character. The value of each bit in the status byte can be deduced after converting the ASCII character into a binary value. Each bit of the status byte represents a particular parameter, as described in the following table.

Note:
Please refer to the Appendix for a complete ASCII to binary conversion table.

Bit #	Function	Meaning for	
		Bit LOW	Bit HIGH
0	At least one program is executing	NO	YES
1	Wait command is executing	NO	YES
2	Manual jog mode is active	NO	YES
3	Local mode is inactive	Default	—
4	At least one trajectory is executing	NO	YES
5	Reserved	Default	—
6	Reserved	—	Default
7	Reserved	Default	—

RETURNS ASCII character representing the status byte.

REL. COMMANDS **TS** - read controller status

EXAMPLE
TX | *read controller activity*
P | *controller returns character P indicating at least one trajectory is executing*

UF update servo filter

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **UF**

PARAMETERS None.

DESCRIPTION This command is used to make active the latest entered PID parameters. Any new value for Kp, Ki, Kd and maximum following error are not being used in the PID loop calculation until UF command is received. This assures that the parameters are loaded simultaneously, without any transitional glitches in the loop.

If the axis specifier **xx** is missing or set to 0 , the controller updates the filters for all axes. If **xx** is a number between 1 and 4, the controller updates only the filter for the specified axis.

RETURNS none

ERRORS none

REL. COMMANDS

FE	-	set maximum following error
KD	-	set derivative gain factor
KI	-	set integral gain factor
KP	-	set proportional gain factor

EXAMPLE

3KP0.05	|	*set proportional gain factor of axis # 3 to 0.05*
3KD0.07	|	*set derivative gain factor of axis # 3 to 0.07*
3UF	|	update servo loop of axis # 3 with the new parameters

UH wait for DIO bit high

	IMM	PGM	MIP
USAGE		♦	

SYNTAX Xx**UH**

PARAMETERS

Description	**xx** [int]	-	DIO bid number
Range	**xx**	-	**0** to **23** (for ESP6000 and ESP7000 controllers) **0** to **15** (for ESP100 and ESP300 controllers)
Units	**xx**	-	none
Defaults	**xx**	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error 7, PARAMETER OUT OF RANGE

DESCRIPTION This command causes a program to wait until a selected I/O input bit becomes high. It is level, not edge sensitive. This means that at the time of evaluation, if the specified I/O bit **xx** is high already, the program will continue to execute subsequent commands.

Note:
All DIO bits are pulled high on the board. Therefore, a missing signal will cause the wait to complete and subsequent commands will continue to be executed.

RETURNS none

REL. COMMANDS **UL** - Wait for DIO bit low

EXAMPLE

1EP	| *Enter stored program #1*
1MO	| *Turn axis #1 motor power ON*
1MV+	| *Move axis #1 indefinitely in positive direction*
13UH	| *Wait for DIO bit #13 to go HIGH before executing any subsequent commands*
1^{ST}	| *Stop axis #1*
WT500	| *Wait for 500 ms*
1MV-	| *Move axis #1 indefinitely in negative direction*
QP	| *Quit program mode*

UL wait for DIO bit low

	IMM	PGM	MIP
USAGE		♦	

SYNTAX Xx**UL**

PARAMETERS

Description	**xx** [int]	-	DIO bid number
Range	**xx**	-	**0** to **23** (for ESP6000 and ESP7000 controllers) **0** to **15** (for ESP100 and ESP300 controllers)
Units	**xx**	-	none
Defaults	**xx**	missing: out of range:	error 38, COMMAND PARAMETER MISSING error 7, PARAMETER OUT OF RANGE

DESCRIPTION This command causes a program to wait until a selected I/O input bit becomes low. It is level, not edge sensitive. This means that at the time of evaluation, if the specified I/O bit **xx** is low already, the program will continue to execute subsequent commands.

RETURNS none

REL. COMMANDS

UL	-	Wait for DIO bit low

EXAMPLE

1EP	| *Enter stored program #1*
1MO	| *Turn axis #1 motor power ON*
1MV+	| *Move axis #1 indefinitely in positive direction*
13UL	| *Wait for DIO bit #13 to go LOW before executing any subsequent commands*
1^{ST}	| *Stop axis #1*
WT500	| *Wait for 500 ms*
1MV-	| *Move axis #1 indefinitely in negative direction*
QP	| *Quit program mode*

VA set velocity

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**VA**nn or xx**VA**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	velocity value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to maximum value allowed by VU command or **?** to read current setting
Units	**xx**	-	none
	nn	-	preset units / second
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx10, MAXIMUM VELOCITY EXCEEDED

DESCRIPTION

This command is used to set the velocity value for an axis. Its execution is immediate, meaning that the velocity is changed when the command is processed, even while a motion is in progress.

It can be used as an immediate command or inside a program. If the requested axis is member of a group, the commanded velocity becomes effective only after the axis is removed from the group. (Refer the Advanced Capabilities section for detailed description of grouping and related commands).

Avoid changing the velocity during the acceleration or deceleration periods. For better predictable results, change velocity only when the axis is not moving or when it is moving with a constant speed.

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

AC	-	set acceleration
VU	-	set maximum velocity
PA	-	execute an absolute motion
PR	-	execute a relative motion

EXAMPLE

2VA?	| *read desired velocity of axis # 2*
10	| *controller returns a velocity value of 10 units/s*
2PA15	| *move to absolute position 15*
WT500	| *wait for 500ms*
2VA4	| *set axis # 2 velocity to 4 units/s*
2VA?	| *read velocity of axis # 2*
4	| *controller returns a velocity value of 4 units/s*

VB set base velocity for step motors

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**VB**nn or xx**VB**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	base velocity value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to maximum value allowed by VU command or **?** to read current setting
Units	**xx**	-	none
	nn	-	preset units / second
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx10, MAXIMUM VELOCITY EXCEEDED error xx01, Axis-xx PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to set the base velocity, also referred to as start/stop velocity value for a step motor driven axis. Its execution is immediate, meaning that the velocity is changed when the command is processed, even while a motion is in progress. It can be used as an immediate command or inside a program.

Avoid changing the velocity during the acceleration or deceleration periods. For better predictable results, change velocity only when the axis is not moving or when it is moving with a constant speed.

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

AC	-	set acceleration
VA	-	set velocity
VU	-	set maximum velocity
PA	-	execute an absolute motion
PR	-	execute a relative motion

EXAMPLE

2VB?	| *read desired base velocity of axis # 2*
5	| *controller returns a velocity value of 5 units/s*
2VB10	| *set axis # 2 base velocity to 10 units/s*
2VB?	| *read base velocity of axis # 2*
10	| *controller returns a velocity value of 10 units/s*

VE read controller firmware version

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **VE ?**

PARAMETERS none

Defaults timeout: error 2, RS-232 COMMUNICATION TIME-OUT

DESCRIPTION This command is used to read the controller type and version.

Note:
Important information needed when asking for technical support for the motion control system or when reporting a problem is the controller version. Use this command to determine the controller type and in particular, the firmware version.

RETURNS **ESP300 Version xx.yy**
where: **xx.yy** = version and release number

REL. COMMANDS none

EXAMPLE

VE?	*read controller firmware version*
ESP300 Version 3.0.1 6/1/99	*controller returns model ESP300 version 3.0 and release date 6/1/99*

VF set velocity feed-forward gain

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**VF**nn or xx**VF**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	velocity feed-forward gain factor Vf
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **2e9,** or **?** to read current setting
Units	**xx**	-	none
	nn	-	none

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command sets the velocity feed-forward gain factor Vf. It is active for any DC servo based motion device.

.

See the "**Feed-Forward Loops**" in Motion Control Tutorial section to understand the basic principals of feed-forward.

Note:
The command can be sent at any time but it has no effect until the UF (update filter) is received.

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS	**KI**	-	set integral gain factor
	KS	-	set saturation gain factor
	KD	-	set derivative gain factor
	KP	-	set proportional gain factor
	AF	-	set acceleration feed-forward gain
	UF	-	update filter

EXAMPLE

3AF0.8	*| set acceleration feed-forward gain factor for axis # 3 to 0.8*
3VF?	*| report present axis-3 velocity feedforward setting*
1.4	*| controller returns a value of 1.4*
3VF1.5	*| set acceleration feed-forward gain factor for axis # 3 to 1.5*
3UF	*| update PID filter; only now the **VF** command takes effect*

VU set maximum velocity

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**VU**nn or xx**VU**?

PARAMETERS

Description			
	xx [int]	-	axis number
	nn [float]	-	velocity value
Range	**xx**	-	to **MAX AXES**
	nn	-	**0** to **2e+9,** or **?** to read current setting
Units	**xx**	-	none
	nn	-	predefined units / second
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx10, MAXIMUM VELOCITY EXCEEDED
			error xx2, Axis-xx PARAMETER OUT OF RANGE

DESCRIPTION This command is used to set the maximum velocity value for an axis. This command remains effective even if the requested axis is member of a group. In this case an error message, "GROUP MAXIMUM VELOCITY EXCEEDED", is generated if the commanded value is less than group velocity. (Refer to Advanced Capabilities section for a detailed description of grouping and related commands).

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting

REL. COMMANDS

VA	-	set velocity
PA	-	execute an absolute motion
PR	-	execute a relative motion
AG	-	set deceleration
AC	-	set acceleration

EXAMPLE

2VU?	| *read maximum allowed velocity of axis # 2*
10	| *controller returns a value of 10 units/s*
2VU8	| *set axis # 2 maximum maximum to 8 units/s*
2VA6	| *set axis #2 working velocity to 6 units/s*

WP wait for position

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**WP**nn

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [float]	-	position value
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	starting position to destination of axis number **xx**
Units	**xx**	-	none
	nn	-	predefined units
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error 7, PARAMETER OUT OF RANGE

DESCRIPTION

This command stops program execution until a user specified position is reached. The program continues executing any subsequent commands only after axis **xx** has reached position **nn**.

Note:
Ensure that position **nn** is within the travel range of axis **xx**. The controller cannot always detect if a value is outside the travel range of an axis to flag an error, especially while making coordinated motion of multiple axes.

Wait commands are primarily intended for use in internal program execution or in combination with the RQ command. If used in command mode, it is important to note that input command processing is suspended until the wait condition has been satisfied.

RETURNS None

REL. COMMANDS

WT	-	wait
WS	-	wait for motion stop

EXAMPLE

2PA-10; 2WS	| *move axis # 2 to position –10 units and wait for stop*
2PA10; **2WP0**; 3PA5	| *move axis #2 to position 10 units, wait for axis #2 to reach* position 0 units and then move axis #3 to position 5 units

WS wait for motion stop

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX xx**WS**nn

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	delay after motion is complete
Range	**xx**	-	**0** to **MAX AXES**
	nn	-	**0** to **60000**
Units	**xx**	-	none
	nn	-	milliseconds

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx2, PARAMETER OUT OF RANGE

DESCRIPTION This command stops the program execution until a motion is completed. The program is continued only after axis **xx** reaches its destination. If **xx** is not specified, the controller waits for all motion in progress to end. If **nn** is specified different than 0, the controller waits an additional **nn** milliseconds after the motion is complete and then executes the next commands.

Note:
Wait commands are primarily intended for use in internal program execution or in combination with the RQ command. If used in command mode, it is important to note that input command processing is suspended until the wait condition has been satisfied.

RETURNS none

REL. COMMANDS

WT	-	wait
WP	-	wait for position

EXAMPLE 2PA10;**2WS500**;3PA5 | *move axis # 2 to position 10 units, wait for axis # 2 to reach destination, wait an additional 500ms and then move* axis # 3 to position 5 units

WT wait

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **WTnn**

PARAMETERS

Description	**nn** [int]	-	wait time (delay)
Range	**nn**	-	**0** to **60000**
Units	**nn**	-	milliseconds
Defaults	**xx**	missing:	error 37, AXIS NUMBER MISSING
		out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn	missing:	error 38, COMMAND PARAMETER MISSING
		out of range:	error xx2, PARAMETER OUT OF RANGE

DESCRIPTION

This command causes the controller to pause for a specified amount of time. This means that the controller will wait **nn** milliseconds before executing the next command.

Note:
Even though this command can be executed in immediate mode, its real value is as a flow control instruction inside programs.

Wait commands are primarily intended for use in internal program execution or in combination with the RQ command. If used in command mode, it is important to note that input command processing is suspended until the wait condition has been satisfied.

RETURNS none

REL. COMMANDS

WS - wait for stop
WP - wait for position

EXAMPLE

2MO;**WT400**;2PA2.3 | turn axis motor ON, *wait an additional*
| 400 ms and then move axis 2 to position 2.3 units

XM read available memory

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **XM**

PARAMETERS None

DESCRIPTION This command reports the amount of unused program memory. The controller has 61440 bytes of non-volatile memory available for permanently storing programs. This command reports the amount not used.

Note:
Available memory space is updated only after the stored program memory is purged using **XX** command.

RETURNS Available storage space

REL. COMMANDS

EP	-	enter program download mode
EX	-	execute a stored program
LP	-	list stored program
XX	-	delete a stored program

EXAMPLE

XM	| *read available memory*
Available storage space = 61440	| *controller reports available storage space*

XX erase program

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX xx**XX**

PARAMETERS

Description	**xx** [int]	-	program number
Range	**xx**	-	**1** to **100**
Units	**xx**	-	none
Defaults	**xx** missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error 7, PARAMETER OUT OF RANGE

DESCRIPTION

This command makes the program **xx** loaded in controller's non-volatile memory unavailable to user. It does not delete the program from memory. Consequently, the program space does not become available to user immediately after deleting the program. It becomes available to user only after the entire stored program memory is purged by issuing the command "**0xx**".

Note:
Purging the stored program memory takes approximately 3 seconds for completion.

RETURNS None

REL. COMMANDS

EP	-	enter program download mode
EX	-	execute a stored program
LP	-	list stored program
XM	-	read available memory

EXAMPLE

1XX | *delete program #1*
XM | *read available memory*
Available storage space = 60228| *controller reports available storage space*
2XX | *delete program #2*
XM | *read available memory*
Available storage space = 60228| *controller reports available storage space*
0XX | *purge stored program memory*
XM | *read available memory*
Available storage space = 61440| *controller reports available storage space*

ZA set amplifier I/O configuration

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**ZA**nn or xx**ZA**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	amplifier I/O configuration
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **0FFFFH** (hexadecimal with leading zero(0)) or **?** to read current setting
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE
	critical setting:		error xx17, ESP CRITICAL SETTINGS ARE PROTECTED
	during motion:		error xx26, PARAMETER CHANGE NOT ALLOWED DURING MOTION

DESCRIPTION

This command is used to set the amplifier I/O polarity, fault checking, and event handling for axis specified with **xx**.

NOTE: If bit-0 or both bits-1 and -2 are set to zero(0) then no action will be taken by the controller.

NOTE: The controller always interprets the **nn** value as a hexadecimal number, even when the letter "H" is not appended to the desired value. Since **nn** is a hexadecimal number, it is possible that the most significant character (left most character) is an alphabet (A—F) depending on the choice of values for various bits. In order for the controller to distinguish between an ASCII command and its value, it is recommended that the users always add a leading zero (0) to the nn value. See table below for clarification:

Example:

Command Issued	Controller Interpretation
1ZA123H	nn = 123H = (0001 0010 0011)Binary
1ZA123	nn = 123H = (0001 0010 0011)Binary
1ZA0F25H	nn = F25H = (1111 0010 0101) Binary
1ZAF25H	Invalid command

BIT #	VALUE	DEFINITION
0	0	disable amplifier fault input checking
*0	1	enable amplifier fault input checking
1	0	do not disable motor on amplifier fault event
*1	1	disable motor on amplifier fault event
*2	0	do not abort motion on amplifier fault event
2	1	abort motion on amplifier fault event
3	0	reserved
3	1	reserved
4	0	reserved
4	1	reserved
5	0	amplifier fault input active low
*5	1	amplifier fault input active high
*6	0	configure step motor control outputs for STEP / DIRECTION
6	1	configure step motor control outputs for +STEP/-STEP
*7	0	configure STEP output as active low
7	1	configure STEP output as active high
8	0	configure DIRECTION output as active low for negative move
*8	1	configure DIRECTION output as active high for negative move
*9	0	do not invert servo DAC output polarity
9	1	invert servo DAC output polarity
*10	0	amplifier enable output active low
10	1	amplifier enable output active high
*11	0	+stepper motor winding is FULL
11	1	+ stepper motor winding is HALF
		• • •
31	0	reserved
31	1	reserved
		* default setting
		+ This bit assignment is effective only on ESP100 and ESP300 motion controllers. Also, any change in motor winding takes affect only when the controller is reset or power cycled. As a result, amplifier I/O configuration must be saved to memory and controller must be reset for this change to take affect.

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting in hexadecimal notation.

REL. COMMANDS

ZB - set feedback configuration
ZE - set e-stop configuration
ZF - set following error configuration
ZH - set hardware limit configuration
ZS - set software limit configuration
ZZ - set general system configuration

EXAMPLE

2ZA?	*read amplifier I/O configuration of axis # 2*
123H	*controller returns a value of 123H for axis #2*
	123H = (0001 0010 0011)Binary
	Bits 0, 1, 5, 8 = 1. All other bits = 0
2ZA 125H	*set amplifier I/O configuration to 125H for axis #2*
	125H = (0001 0010 0101)Binary
	Bits 0, 2, 5, 8 = 1. All other bits = 0
SM	*save all controller settings to non-volatile memory*

Please refer the table above to interpret the affect of these bit values.

ZB set feedback configuration

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**ZB**nn or xx**ZB**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	feedback configuration
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **0FFFFH** (hexadecimal with leading zero(0)) or **?** to read current setting
Units	**xx**	-	none
	nn	-	none

Defaults	**xx** missing:	error 37, AXIS NUMBER MISSING
	out of range:	error 9, AXIS NUMBER OUT OF RANGE
	nn missing:	error 38, COMMAND PARAMETER MISSING
	out of range:	error xx2, PARAMETER OUT OF RANGE
	critical setting:	error xx17, ESP CRITICAL SETTINGS ARE PROTECTED
	during motion:	error xx26, PARAMETER CHANGE NOT ALLOWED DURING MOTION

DESCRIPTION

This command is used to set the feedback configuration , fault checking, and event handling, as well as stepper closed-loop positioning for axis specified with **xx**.

NOTE: If bit-0 or both bits-1 and -2 are set to zero(0) then no action will be taken by the controller.

NOTE: The controller always interprets the **nn** value as a hexadecimal number, even when the letter "H" is not appended to the desired value. Since **nn** is a hexadecimal number, it is possible that the most significant character (left most character) is an alphabet (A—F) depending on the choice of values for various bits. In order for the controller to distinguish between an ASCII command and its value, it is recommended that the users always add a leading zero (0) to the nn value. See table below for clarification:

Example:

Command Issued	Controller Interpretation
1ZB123H	nn = 123H = (0001 0010 0011)Binary
1ZB123	nn = 123H = (0001 0010 0011)Binary
1ZB0F25H	nn = F25H = (1111 0010 0101) Binary
1ZBF25H	Invalid command

BIT#	VALUE	DEFINITION
*0	0	disable feedback error checking
0	1	enable feedback error checking
*1	0	do not disable motor on feedback error event
1	1	disable motor on feedback error event
*2	0	do not abort motion on feedback error event
2	1	abort motion on feedback error event
*3	0	Reserved
3	1	Reserved
*4	0	Reserved
4	1	Reserved
*5	0	do not invert encoder feedback polarity
5	1	invert encoder feedback polarity
*6	0	reserved
6	1	reserved
*7	0	reserved
7	1	reserved
8	0	do not use encoder feedback for positioning
*8	1	use encoder feedback for stepper positioning
9	0	disable stepper closed-loop positioning
*9	1	enable stepper closed-loop positioning
10	0	reserved
10	1	reserved
		• • •
31	0	reserved
31	1	reserved
		* default setting

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting in hexadecimal notation.

REL. COMMANDS

ZA - set amplifier I/O configuration
ZE - set e-stop configuration
ZF - set following error configuration
ZH - set hardware limit configuration
ZS - set software limit configuration
ZZ - set general system configuration

EXAMPLE

2ZB? | *read amplifier I/O configuration of axis # 2*
100H | *controller returns a value of 100H for axis #2*
2ZB 105H | *set amplifier I/O configuration to 105H for axis #2*
SM | *save all controller settings to non-volatile memory*

ZE set e-stop configuration

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**ZE**nn or xx**ZE**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	e-stop configuration
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **0FFFFH** (hexadecimal with leading zero(0)) or **?** to read current setting
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx17, ESP CRITICAL SETTINGS ARE PROTECTED

DESCRIPTION This command is used to set the emergency stop (e-stop) configuration , fault checking, and event handling for axis specified with **xx**.

NOTE: If bit-0 or both bits-1 and -2 are set to zero(0) then no action will be taken by the controller.

NOTE: The controller always interprets the **nn** value as a hexadecimal number, even when the letter "H" is not appended to the desired value. Since **nn** is a hexadecimal number, it is possible that the most significant character (left most character) is an alphabet (A—F) depending on the choice of values for various bits. In order for the controller to distinguish between an ASCII command and its value, it is recommended that the users always add a leading zero (0) to the nn value. See table below for clarification:

Example:

Command Issued	Controller Interpretation
1ZE123H	nn = 123H = (0001 0010 0011)Binary
1ZE123	nn = 123H = (0001 0010 0011)Binary
1ZE0F25H	nn = F25H = (1111 0010 0101) Binary
1ZEF25H	Invalid command

BIT #	VALUE	DEFINITION
0	0	disable E-stop checking
*0	1	enable E-stop checking
*1	0	do not disable motor power on E-stop event
1	1	disable motor power on E-stop event
2	0	do not abort motion on E-stop event
*2	1	abort motion on E-stop event
3	0	reserved
3	1	reserved
4	0	reserved
4	1	reserved
5	0	reserved
5	1	reserved
6	0	reserved
6	1	reserved
7	0	reserved
7	1	reserved
		• • •
31	0	reserved
31	1	reserved
		* default setting

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting in hexadecimal notation.

REL. COMMANDS

ZA - set amplifier I/O configuration
ZB - set feedback configuration
ZF - set following error configuration
ZH - set hardware limit configuration
ZS - set software limit configuration
ZZ - set general system configuration

EXAMPLE

2ZE? | *read e-stop configuration of axis # 2*
3H | *controller returns a value of 3H for axis #2*
2ZE 5H | *set e-stop configuration to 5H for axis #2*
SM | *save all controller settings to non-volatile memory*

ZF set following error configuration

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**ZF**nn or xx**ZF**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	following error configuration
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **0FFFFH** (hexadecimal with leading zero(0)) or **?** to read current setting
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE
	critical setting:		error xx17, ESP CRITICAL SETTINGS ARE PROTECTED

DESCRIPTION

This command is used to set the following error configuration , fault checking, and event handling for axis specified with **xx**.

NOTE: If bit-0 or both bits-1 and -2 are set to zero(0) then no action will be taken by the controller.

NOTE: The controller always interprets the **nn** value as a hexadecimal number, even when the letter "H" is not appended to the desired value. Since **nn** is a hexadecimal number, it is possible that the most significant character (left most character) is an alphabet (A—F) depending on the choice of values for various bits. In order for the controller to distinguish between an ASCII command and its value, it is recommended that the users always add a leading zero (0) to the nn value. See table below for clarification:

Example:

Command Issued	Controller Interpretation
1ZF123H	nn = 123H = (0001 0010 0011)Binary
1ZF123	nn = 123H = (0001 0010 0011)Binary
1ZF0F25H	nn = F25H = (1111 0010 0101) Binary
1ZFF25H	Invalid command

BIT#	VALUE	DEFINITION
0	0	disable motor following error checking
*0	1	enable motor following error checking
1	0	do not disable motor power on following error event
*1	1	disable motor power on following error event
*2	0	do not abort motion on following error event
2	1	abort motion on following error event
3	0	reserved
3	1	reserved
4	0	reserved
4	1	reserved
5	0	reserved
5	1	reserved
6	0	reserved
6	1	reserved
7	0	reserved
7	1	reserved
		• • •
31	0	reserved
31	1	reserved
		* default setting

RETURNS If the "?" sign takes the place of **nn** value, this command reports the current setting in hexadecimal notation.

REL. COMMANDS

ZA - set amplifier I/O configuration
ZB - set feedback configuration
ZE - set e-stop configuration
ZH - set hardware limit configuration
ZS - set software limit configuration
ZZ - set general system configuration
FE - set following error threshold

EXAMPLE

2ZF? | *read following error configuration of axis # 2*
3H | *controller returns a value of 3H for axis #2*
2ZF 5H | *set following error configuration to 5H for axis #2*
SM | *save all controller settings to non-volatile memory*

ZH set hardware limit configuration

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**ZH**nn or xx**ZH**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	hardware limit configuration
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **0FFFFH** (hexadecimal with leading zero(0)) or **?** to read current setting
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE
	critical setting:		error xx17, ESP CRITICAL SETTINGS ARE PROTECTED

DESCRIPTION

This command is used to set the hardware limit checking, polarity, and event handling for axis specified with **xx**.

NOTE: If bit-0 or both bits-1 and -2 are set to zero(0) then no action will be taken by the controller.

NOTE: The controller always interprets the **nn** value as a hexadecimal number, even when the letter "H" is not appended to the desired value. Since **nn** is a hexadecimal number, it is possible that the most significant character (left most character) is an alphabet (A—F) depending on the choice of values for various bits. In order for the controller to distinguish between an ASCII command and its value, it is recommended that the users always add a leading zero (0) to the nn value. See table below for clarification:

Example:

Command Issued	Controller Interpretation
1ZH123H	nn = 123H = (0001 0010 0011)Binary
1ZH123	nn = 123H = (0001 0010 0011)Binary
1ZH0F25H	nn = F25H = (1111 0010 0101) Binary
1ZHF25H	Invalid command

BIT #	VALUE	DEFINITION
0	0	disable hardware travel limit error checking
*0	1	enable hardware travel limit error checking
*1	0	do not disable motor on hardware travel limit event
1	1	disable motor on hardware travel limit event
2	0	do not abort motion on hardware travel limit event
*2	1	abort motion on hardware travel limit event
3	0	reserved
3	1	reserved
4	0	reserved
4	1	reserved
5	0	hardware travel limit input active low
*5	1	hardware travel limit input active high
6	0	reserved
6	1	reserved
7	0	reserved
7	1	reserved
		• • •
31	0	reserved
31	1	reserved
		* default setting

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting in hexadecimal notation.

REL. COMMANDS

ZA	-	set amplifier I/O configuration
ZE	-	set e-stop configuration
ZF	-	set following error configuration
ZB	-	set feedback configuration
ZS	-	set software limit configuration
ZZ	-	set general system configuration

EXAMPLE

2ZH?	| *read hardware limit configuration of axis # 2*
25H	| *controller returns a value of 25H for axis #2*
2ZH 23H	| *set hardware limit configuration to 23H for axis #2*
SM	| *save all controller settings to non-volatile memory*

ZS set software limit configuration

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX xx**ZS**nn or xx**ZS**?

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	hardware limit configuration
Range	**xx**	-	**1** to **MAX AXES**
	nn	-	**0** to **0FFFFH** (hexadecimal with leading zero(0)) or **?** to read current setting
Units	**xx**	-	none
	nn	-	none
Defaults	**xx** missing:		error 37, AXIS NUMBER MISSING
	out of range:		error 9, AXIS NUMBER OUT OF RANGE
	nn missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE
	critical setting:		error xx17, ESP CRITICAL SETTINGS ARE PROTECTED

DESCRIPTION

This command is used to set the software limit checking and event handling for axis specified with **xx**.

NOTE: If bit-0 or both bits-1 and -2 are set to zero(0) then no action will be taken by the controller.

NOTE: The controller always interprets the **nn** value as a hexadecimal number, even when the letter "H" is not appended to the desired value. Since **nn** is a hexadecimal number, it is possible that the most significant character (left most character) is an alphabet (A—F) depending on the choice of values for various bits. In order for the controller to distinguish between an ASCII command and its value, it is recommended that the users always add a leading zero (0) to the nn value. See table below for clarification:

Example:

Command Issued	Controller Interpretation
1ZS123H	nn = 123H = (0001 0010 0011)Binary
1ZS123	nn = 123H = (0001 0010 0011)Binary
1ZS0F25H	nn = F25H = (1111 0010 0101) Binary
1ZSF25H	Invalid command

BIT #	VALUE	DEFINITION
0	0	disable software travel limit error checking
*0	1	enable software travel limit error checking
*1	0	do not disable motor on software travel limit event
1	1	disable motor on software travel limit event
2	0	do not abort motion on software travel limit event
*2	1	abort motion on software travel limit event
3	0	reserved
3	1	reserved
4	0	reserved
4	1	reserved
5	0	reserved
5	1	reserved
6	0	reserved
6	1	reserved
7	0	reserved
7	1	reserved
		• • •
31	0	reserved
31	1	reserved
		* default setting

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting in hexadecimal notation.

REL. COMMANDS

ZA - set amplifier I/O configuration
ZE - set e-stop configuration
ZF - set following error configuration
ZB - set feedback configuration
ZH - set hardware limit configuration
ZZ - set general system configuration
SL - set left limit
SR - set right limit

EXAMPLE

2ZS? | *read software limit configuration of axis # 2*
4H | *controller returns a value of 4H for axis #2*
2ZS 5H | *set software limit configuration to 5H for axis #2*
SM | *save all controller settings to non-volatile memory*

ZU get ESP system configuration

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **ZU**

PARAMETERS None

DESCRIPTION This command is used to get the present ESP system stage/driver configuration. After each system reset or initialization the ESP motion controller detects the presence of Universal drivers and ESP-compatible stages connected.

BIT #	VALUE	DEFINITION
0	0	axis-1 universal driver not detected
0	1	axis-1 universal driver detected
1	0	axis-2 universal driver not detected
1	1	axis-2 universal driver detected
2	0	axis-3 universal driver not detected
2	1	axis-3 universal driver detected
3	0	axis-4 universal driver not detected
3	1	axis-4 universal driver detected
4	0	axis-5 universal driver not detected
4	1	axis-5 universal driver detected
5	0	axis-6 universal driver not detected
5	1	axis-6 universal driver detected
6	0	reserved
6	1	reserved
7	0	reserved
7	1	reserved
8	0	axis-1 ESP-compatible motorized positioner not detected
8	1	axis-1 ESP-compatible motorized positioner detected
9	0	axis-2 ESP-compatible motorized positioner not detected
9	1	axis-2 ESP-compatible motorized positioner detected
10	0	axis-3 ESP-compatible motorized positioner not detected
10	1	axis-3 ESP-compatible motorized positioner detected
11	0	axis-4 ESP-compatible motorized positioner not detected
11	1	axis-4 ESP-compatible motorized positioner detected
12	0	axis-5 ESP-compatible motorized positioner not detected
12	1	axis-5 ESP-compatible motorized positioner detected
13	0	axis-6 ESP-compatible motorized positioner not detected
13	1	axis-6 ESP-compatible motorized positioner detected
14	0	reserved
14	1	reserved
15	0	reserved
15	1	reserved
		• • •
31	0	reserved
31	1	reserved

RETURNS **T**his command reports the current setting in hexadecimal notation.

REL. COMMANDS

ZA	**-**	set amplifier I/O configuration
ZB	**-**	set feedback configuration
ZE	**-**	set e-stop configuration
ZF	**-**	set following error configuration
ZH	**-**	set hardware limit configuration
ZS	**-**	set software limit configuration
ZZ	**-**	set system configuration

EXAMPLE

ZU	**|** *read ESP system configuration*
150015H	**|** *controller returns a value of 150015H*

ZZ set system configuration

	IMM	PGM	MIP
USAGE	♦	♦	

SYNTAX **ZZ**nn or **ZZ**?

PARAMETERS

Description	**nn** [int]	-	system configuration
Range	**nn**	-	**0** to **0FFFFH** (hexadecimal with leading zero(0)) or **?** to read current setting
Units	**nn**	-	none
Defaults	**nn** missing:		error 38, COMMAND PARAMETER MISSING
	out of range:		error xx2, PARAMETER OUT OF RANGE

DESCRIPTION

This command is used to configure system fault checking, event handling and general setup for all axes.

NOTE: If bit-0 or both bits-1 and -2 are set to zero(0) then no action will be taken by the controller.

NOTE: The controller always interprets the **nn** value as a hexadecimal number, even when the letter "H" is not appended to the desired value. Since **nn** is a hexadecimal number, it is possible that the most significant character (left most character) is an alphabet (A—F) depending on the choice of values for various bits. In order for the controller to distinguish between an ASCII command and its value, it is recommended that the users always add a leading zero (0) to the nn value. See table below for clarification:

Example:

Command Issued	Controller Interpretation
1ZZ123H	nn = 123H = (0001 0010 0011)Binary
1ZZ123	nn = 123H = (0001 0010 0011)Binary
1ZZ0F25H	nn = F25H = (1111 0010 0101) Binary
1ZZF25H	Invalid command

BIT#	VALUE	DEFINITION
0	0	disable 100-pin interlock error checking
*0	1	enable 100-pin interlock error checking
1	0	do not disable all axes on 100-pin interlock error event
*1	1	disable all axes on 100-pin interlock error event
2	0	reserved
2	1	reserved
3	0	reserved
3	1	reserved
4	0	configure interlock fault as active low
*4	1	configure interlock fault as active high
5	0	reserved
5	1	reserved
6	0	reserved
6	1	reserved
*7	0	route auxiliary I/O encoder signals to counter channels **MAX AXES** + 1 and **MAX AXES** + 2
7	1	route axis 1 and 2 encoder feedback to counter channels **MAX AXES** + 1 and **MAX AXES** + 2
8	0	unprotect ESP system-critical settings
*8	1	protect ESP system-critical settings
*9	0	Enable queue purge on time expiration
9	1	Disable queue purge on time expiration
*10	0	Do not display units along with certain responses
10	1	Display units along with certain responses
*11	0	Enable timeout during homing
11	1	Disable timeout during homing
		• • •
31	0	reserved
31	1	reserved
		* default setting

RETURNS

If the "?" sign takes the place of **nn** value, this command reports the current setting in hexadecimal notation.

REL. COMMANDS

ZA	-	set amplifier I/O configuration
ZB	-	set feedback configuration
ZE	-	set e-stop configuration
ZF	-	set following error configuration
ZH	-	set hardware limit configuration
ZS	-	set software limit configuration
ZU	-	get ESP system configuration

EXAMPLE

ZZ?	**|** *read system configuration*
113H		*controller returns a value of 113H*
ZZ 13H	**|** set system configuration to 13H

Section 4 – Advanced Capabilities

4.1 Data Acquisition

4.1.1 Introduction – Data Acquisition

ESP series of motion controllers combine high-performance data acquisition and a diverse motion control feature set on one controller card. This enables physical integration of the two functions, eliminating the problems normally associated with integrating different circuit boards, enhancing acquisition synchronization, and alleviating power and space constraints.

While the sub-sections 4.2 and 4.3 outline some of the advanced motion related features, this section is dedicated to the data acquisition capabilities of the motion controller. ESP controllers support acquisition of two different types of data: analog data and trace variable data. This data can be collected using one of two different modes: fixed and continuous. The following sub-section detail acquisition of analog data and the subsequent sub-section explains acquisition of trace variable data.

4.1.2 Analog Data Acquisition

The ESP controller supports acquisition of up to eight (8) analog channels, and eight (8) position feedback (encoder) channels, sampling rate, and number of samples to be collected can easily be made using ASCII command, **DC.** Furthermore, the initiation of data acquisition is very flexible and can be configured using the same command.

The controller supports eight (8) different ways of selecting the analog input mode (ADC gain and polarity) through ASCII command, **AM.** Through this command, users can select gains of 1.25, 2.5, 5.0, or 10.0 volts in either uni-polar (0 to +) or bi-polar (- to +) format for all analog inputs.

The controller card provides a multiplexed, eight (8) channel, 16-bit Analog-To-Digital converter (ADC) for processing analog signals from its analog I/O connector. This connector can be accessed via

Newport's optional analog cable. The cable attaches to an open slot at the back of a personal computer for convenient hook-up to customer provided devices. A simplified block diagram of the converter and associated circuitry is shown in **Figure 4.1,** and described in the following paragraphs.

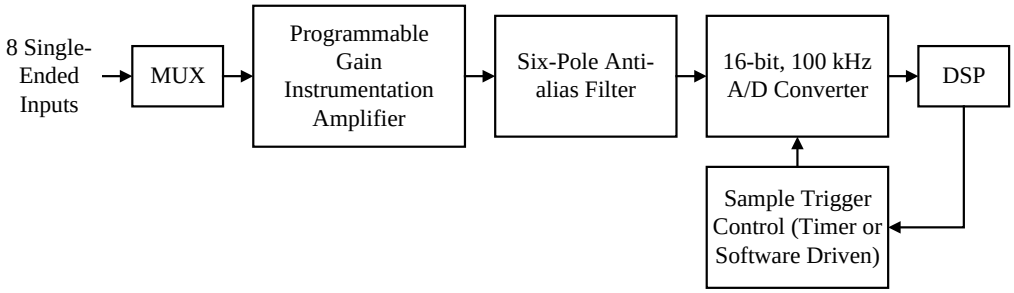


Figure 4.1: Analog-To-Digital Flow Diagram

One of the eight (8) analog voltage input signals is initially selected by the input multiplexer (MUX) based on the data acquisition setup command (**DC**) issued by user. The signal then passes to a Programmable Gain Instrumentation Amplifier (PGIA), which must be pre-set to accept a designated voltage range using the analog input mode command (**AM**). The amplified signal is routed through an anti-alias filter, which filters input at the servo cycle frequency. The filtered signal is then converted into digital form using a 16-bit, A/D converter for processing by the Digital Signal Processor (DSP).

The A/D converter can read from one to eight channels, one-at-a-time, during a servo cycle (about 400 milliseconds). Sampling is not instantaneous, but occurs at 30 micro-second intervals between each input signal. In BIPOLAR mode the accuracy of the digitized signal is ± 2.5 millivolts of the input signal. BIPOLAR mode is generally recommended over UNIPOLAR mode. In UNIPOLAR mode there may be considerable variation in the reported digitized signal as compared to the input signal depending on the gain selection.

A typical acquisition sampling tray is shown in **Table 4.1.**

First Element						Last Element
Ch 1	Ch 3	Ch 6	Axis 2	Axis 4	Axis 5	CLK

Ch xx = value read at analog channel number xx

Axis xx = position value for axis xx

CLK = servo clock counter

Table 4.1: Acquisition Tray

Steps	ASCII Command	Action by Controller
1. Define analog input mode	**AM2**	Set 0 to 10V analog range for all the ADC channels
2. Setup data acquisition	**DC1, 1, 37, 26, 0, 1000**	Start analog data acquisition when trigger axis #1 starts motion. Acquire analog data for channels 2, 4, 5. Collect 1000 samples, one sample every servo cycle.
3. Enable data acquisition	**DE1**	Controller enables data acquisition. Data acquisition process will start when trigger axis starts motion.
4. Query data acquisition done status	**DD**	Controller responds with the acquisition status. A response = 1 implies data acquisition is done. A response = 0 implies data acquisition is in progress.
5. Query number of samples collected	**DF**	Controller responds with the number of data samples acquired at the time of processing this command.
6. Users may query the data acquisition status until the controller responds a value of 1, i.e. data acquisition is DONE.		
7. Disable data acquisition	**DE0**	Controller disables data acquisition.
8. Get data collected	**DG**	Controller responds with the data collected.

Table 4.2: An Example of Analog Data Acquisition

4.1.3 Trace Variable Data Acquisition

The ESP controller supports acquisition of certain trace variables to monitor motion controller performance. Similar to the analog data acquisition setup, the sampling rate, and number of trace samples to be collected can be configured using the ASCII command, **DC.** However, at the current time, the acquisition of trace variable data is initiated immediately after enabling the data acquisition process.

The trace variables supported include:

1. Desired position
2. Actual position (if encoder feedback is available)
3. Desired velocity
4. Desired acceleration
5. Trajectory phase (acceleration, constant velocity, etc.)
6. Following error

7. Error integral
8. Control (DAC) output #1
9. Control (DAC) output #2 (for Commutated steppers only)
10. Number of stepper pulses/servo cycle
11. Duration between stepper pulses

The controller also supports four (4) different options in which controller can send the collected trace variable data. They are:

1. Texas Instruments floating point format data for an axis that is **not** configured as a slave axis.
2. Scaled integer format for an axis that is **not** configured as a slave axis.
3. Texas Instruments floating point format data for an axis that **is** configured as a slave axis.
4. Scaled integer format for an axis that **is** configured as a slave axis.

Please refer to the description of data acquisition setup command (**DC**) for further details, correct syntax, etc.

Steps	ASCII Command	Action by Controller
1. Setup data acquisition.	**DC10, 1, 1, 1, 0, 1000**	Start trace variable data acquisition when trigger axis #1 starts motion. Collect 1000 samples, one sample every servo cycle.
2. Enable data acquisition	**DE1**	Controller enables data acquisition. Data acquisition process will start when trigger axis starts motion.
3. Query data acquisition done status.	**DD**	Controller responds with the acquisition status. A response = 1 implies data acquisition is done. A response = 0 implies data acquisition is in progress.
4. Query number of samples collected.	**DF**	Controller responds with the number of data samples acquired at the time of processing this command.
5. Users may query the data acquisition status until the controller responds a value of 1, i.e. data acquisition is DONE.		
6. Disable data acquisition	**DE0**	Controller disables data acquisition.
7. Get data collected	**DG**	Controller responds with the data collected.

Table 4.3: An Example of Trace Variable Data Acquisition

Commands related to data acquisition – analog or trace variable – are listed in **Table 4.4** (refer to Section 3, Remote Mode for additional details):

Command	Description
AM	Set analog input mode
DC	Setup data acquisition
DD	Query data acquisition done status
DE	Enable or disable data acquisition
DG	Get acquired data

Table 4.4: Data Acquisition Commands (ASCII)

4.2 Grouping

4.2.1 Introduction - Grouping

Coordinated motion of multiple axes is required to produce a desired contour in a multi-dimensional space. For instance, if we want to move from one point to another along a line or along a circle, or a combination of both line and circle, we require coordinated motion of multiple axes. One way to facilitate such coordinated motion is *"**grouping**"* the axes involved in producing the desired motion. This is akin to defining the coordinate system in which the desired contour is being made.

Coordinated motion on a 2-D plane, therefore, requires a group comprised of any two axes, while a similar motion in a 3-D space requires a group consisting of any three axes. For sake of simplicity, all further discussion of coordinated motion will be restricted to a 2-D plane.

The procedure for defining a group and all the group parameters required for making coordinated motion is described in Section 4.2.2. Section 4.2.3 discusses the commands that actually make the coordinated motion. The procedure for making "long" moves or contours that involve a combination of circular and linear moves is described in Section 4.2.4. Miscellaneous grouping commands are discussed in Section 4.2.5.

4.2.2 Defining a Group and Group Parameters

This subsection discusses the method for defining a group and all the group parameters.

4.2.2.1 Creating a Group

The ASCII command used to create a new group is **HN.** For instance, the command **1HN2, 3** assigns axis numbers 2 and 3 to group number 1. One such group must be defined first before those axes can be moved in a coordinated fashion. A group can comprise of axes anywhere from one to six. If a group has only one axis assigned to it, a linear motion of the group is similar to moving that axis from one point to another. Circular motion of a group with only one axis cannot be made. If a group has more than two axes assigned to it, circular motion of the group is made using the first two axes in the group.

The order in which axes are assigned to a group is very important. This is because it specifies the frame of reference in which coordinated motion of axes takes place.

For instance, the command **IHN2, 3** assigns axis numbers 2 and 3 to group number 1, where axis #2 is equivalent to X-axis and axis #3 is equivalent to Y-axis in a traditional Cartesian coordinate system. Reversing the order of axes (E.G., **IHN3, 2**) reverses the axis assignment.

A few rules that are in place for easy management of group are as follows:

- An axis cannot be a member of different groups at the same time.
- An axis cannot be assigned more than once in a group.
- A group has to be deleted before axes assigned to it can be changed.
- An axis assigned to a group cannot be moved individually using commands such as **PA** and **PR.** Use group linear move commands instead.

Refer to the description of the command in the *commands section* (See Section 5: Programming) for correct syntax, parameter ranges, etc.

4.2.2.2 Defining Group Parameters

Group parameters such as velocity, acceleration, deceleration, jerk, and e-stop deceleration must be defined for every group following the creation of that group. These parameters are used to produce the desired coordinated motion of the group. They override any original values specified for individual axes. The axes' original values are restored when the group to which they have been assigned is deleted. Refer to the description of **HV, HA, HD, HJ,** and **HE** commands in the *commands section* (See Section 3: Remote Mode) for correct syntax, parameter ranges, etc.

4.2.3 Making Linear and Circular Moves

This subsection discusses the method for making linear and circular moves of groups. While coordinated motion of axes with different motor types and different encoder resolutions is supported, it is assumed that all axes have the same units of measure.

4.2.3.1 Making Linear Move

Once a group has been defined and all group parameters have been specified, the ASCII command **HL** is used to move the group from an initial position to a final position along the line. The current position of axes is the initial position of linear move. The desired final position is specified along with this command.

This command makes all axes assigned to the group move with predefined group (tangential) velocity, acceleration and deceleration along a line. A trapezoid velocity profile is employed when a group jerk is set to zero. Otherwise, an S-curve velocity profile is employed.

The linear move is a true linear interpolation, meaning:

$$(y - y_0) = m(x - x_0)$$

$$m = \frac{(y_f - y_0)}{(x_f - x_0)}$$

where: X_0 and Y_0 represent initial position of the group.

X_f and Y_f represent desired final position of the group.

4.2.3.2 Making Circular Move

Once a group has been defined and all group parameters have been specified, the ASCII command **HC** can be used to move the group from an initial position to a final position along a circle. The current position of axes is the initial position of circular move. The final position of move is calculated based on the desired center of circle and sweep angle specified along with this command. All sweep angles are measured in degrees. The sign of angles follow the trigonometric convention: positive angles are measured counterclockwise.

This command makes all axes assigned to the group move with predefined group (tangential) velocity, acceleration and deceleration along a circle. A trapezoid velocity profile is employed to produce the desired motion.

The circular move is a true arc of a circle, meaning:

$$r = \sqrt{(x_0 - x_c)^2 + (y_0 - y_c)^2}$$

$$\theta = a\tan 2\left(\frac{(y_0 - y_c)}{(x_0 - x_c)}\right)$$

$$\theta_0\,[axis\#2] = \theta$$

$$\theta_0\,[axis\#3] = \theta - \frac{\pi}{2}$$

$$\theta_f\,[axis\#2] = \theta_0 + \theta_{cmd}$$

$$\theta_f\,[axis\#3] = \theta_0 + \theta_{cmd}$$

$$x_f = r * \cos(\theta_f\,[axis\#2]) + x_c$$

$$y_f = r * \cos(\theta_f\,[axis\#3]) + y_c$$

where: X_0 and Y_0 represent initial position of the group.

X_c and Y_c represent desired center position of the circular move.

X_f and Y_f represent calculated final position of the group.

r is radius of the circle.

θ_0 is the base initial angle of an axis.

θ_0 is the final angle of an axis, which is dependant on the sweep angle, θ_{cmd}.

Both **HL** and **HC** can initiate the desired motion if they are received while the group is holding position.

On the other hand, if they are received while a group move is in progress, the new commands get queued into a "via-point" buffer. The queued commands are executed on a FIFO basis when the move already in progress has reached its destination. The group does not come to a stop at the end of last move. Instead, there will be a smooth transition to the new move command, just as if it were one compound move (combination of multiple moves). The next section details the procedure for making contours or "long" moves using "via point" buffers.

Refer to the description of **HL** and **HC** commands in the *commands section* (See Section 3: Remote Mode) for correct syntax, parameter ranges, etc.

4.2.4 Making Contours

This subsection discusses the method for making contours. Contouring is the process of making complex trajectories or "long" moves that may involve linear and circular move segments.

These move segments can be sequenced in any order. Arcs can be followed by arcs or lines, and lines by arcs or other lines as shown in the following figures. Since there is no pre-processing of move segments involved in making a contour, the user must ensure that there is no change in tangential velocity at the transition from one move to another. If this constraint is not satisfied, the transition from one move segment to another may cause excessive accelerations and shocks that could damage the stages.

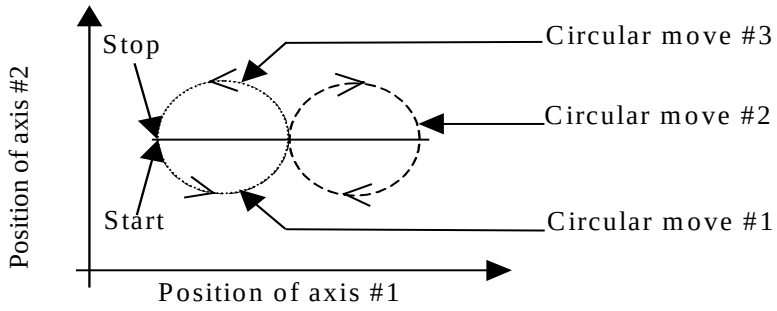


Figure 4.2: A Contour with Multiple Circular Moves

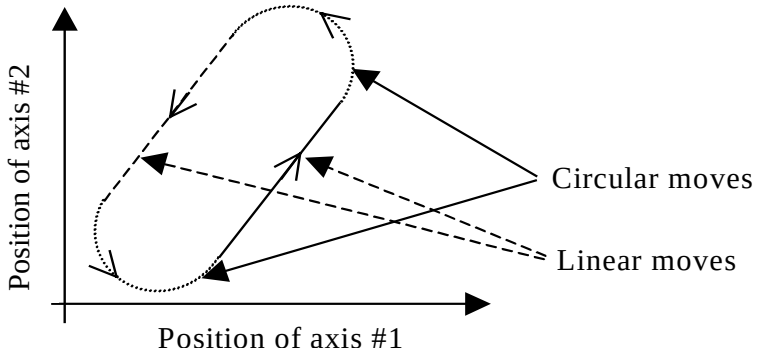


Figure 4.3: A Contour with Multiple Linear and Circular Moves

In order to store the multiple move segment commands needed to make a contour, we make use of a "via point" buffer. This "via point" buffer contains group move commands essential to make a new move segment upon completion of the move segment currently in progress. The new move commands are pulled out of the buffer on a FIFO basis. The "via point" buffer can hold a maximum of 10 group move commands.

If more than 10 group move commands are issued by a user, the excess commands are flow-controlled by the firmware. This mechanism will block the portal through which the commands were issued until all the commands issued have been executed.

It is, therefore, recommended that the user take advantage of ASCII command, **HQ** which tells the number of commands that can be put in the "via point" buffer at any given time. This allows a user to control the flow of commands manually, while ensuring the availability of that portal for other commands such as **HP, TP,** etc.

The trajectory generator checks if the "via point" buffer has a new target position (i.e., any new move segments pending?) while the current move is in progress. If "via point" buffer is empty, the group comes to a stop upon completion of current move segment. Otherwise, it begins a new move segment without stopping after completing the current move. The group transitions from current move segment to a new move segment smoothly if the tangential velocity at the transition is ensured to be constant.

The ASCII command **HQ** is used to query the available "via point" buffer space. The commands **HL** and **HC** are used to queue linear move or circular move commands into the "via point" buffer. Refer to the description of these commands in the *commands section* (See Section 3: Remote Mode) for correct syntax, parameter ranges, etc.

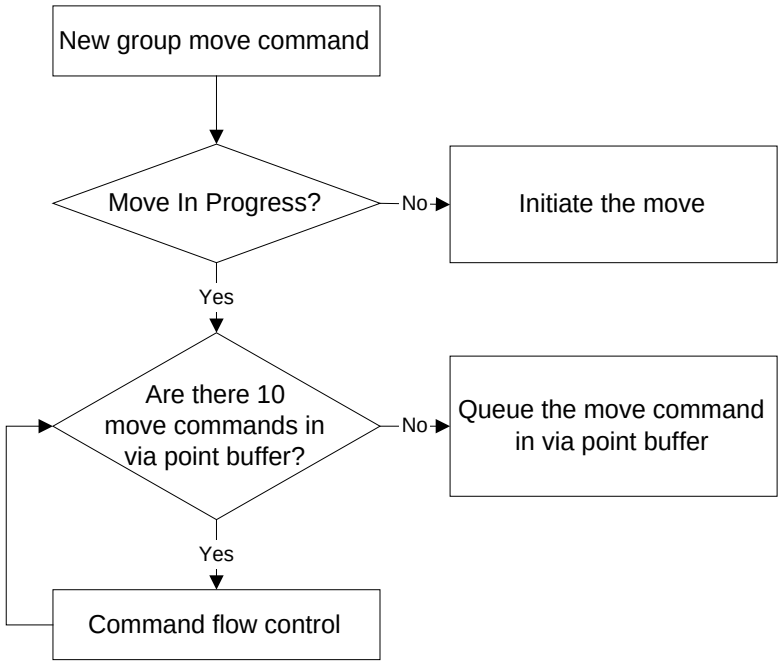


Figure 4.4: Block Diagram of Via Point Data Handling by Command Processor

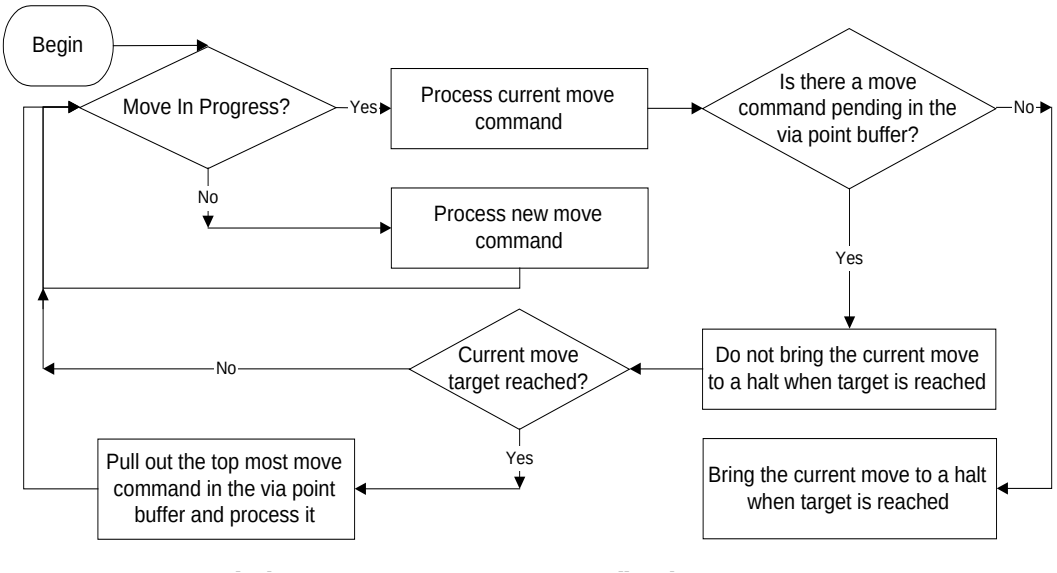


Figure 4.5: Block Diagram of Via Point Data Handling by Trajectory Generator

4.2.5 Miscellaneous Commands

The following commands are available to operate on a group of axes simultaneously:

- **HO** and **HF:** These commands are used to turn ON and turn OFF the power to all axes in a group respectively. The axes assigned to a group can be powered ON or OFF individually using **MO** and **MF** commands also. A group is considered to be ON if all axes assigned to that group are ON.
- **HP:** This command is used to read the actual position of all axes in a group.
- **HS:** This command is used to stop the group motion.
- **HW:** This command is used to wait for the group motion to stop and a user settable delay period thereafter.
- **HX:** This command is used to delete a group.
- **HZ:** This command is used to read the size or the number of axes assigned to a group.

4.3 Slaving a Stage to Trackball, Joystick, or a Different Stage

4.3.1 Introduction – Slaving a Stage

ESP series of motion controllers allow three different ways in which a slave axis can respond to a master axis. They are:

1. Slave to master's desired position (trajectory).
2. Slave to master's actual position (feedback).
3. Slave to master's actual velocity for jogging.

The first two ways may be used when absolute or relative move commands can be issued to the master. This is the situation when both master and slave axes are driven by valid motor types. The third way may be used when move commands cannot be issued to the master. This is the situation when the slave axis is driven by a valid motor type, but the master, such as trackball or joystick, is not.

In any case, a series of preliminary commands have to be issued before the desired master-slave response is obtained. These include defining master-slave relationship, appropriate constants and trajectory mode.

The next section outlines the steps to be taken for a slave axis to follow master's position. The subsequent section outlines the steps to be taken for a slave axis to follow master's velocity. The final section outlines the steps to be taken to jog an axis based on inputs from a digital joystick.

4.3.2 Slave to A Different Stage

The following steps may be taken for a slave axis to follow master's position. This mode may be chosen exclusively when absolute or relative move commands can be used to the master.

Steps	Move Command	Action by Move Command
1. Define master-slave relationship.	**2SS1**	Axis #2 is the slave of axis #1
2. Define master-slave reduction ratio	**2GR0.5**	Master's position is scaled by 0.5 to obtain slave's position
3. Define slave axis trajectory mode	**2TJ4 (or 5)**	Set slave axis trajectory mode
4. Define master axis trajectory mode	**1TJ1 (or 2)**	Set master axis trajectory mode
5. Issue move commands to master axis	**1PA10** **1PR10**	Move master to absolute 10 units. Move master by relative 10 units

Table 4.5: Slave to A Different Stage Steps

4.3.3 Slave to a Trackball

The following steps may be taken for a slave axis to follow master's velocity. This mode may be chosen exclusively when absolute or relative move commands cannot be issued to the master.

In this case, when the master is moved by the user, the slave axis responds by jogging in proportion to the master's velocity. The slave axis jog velocity update interval and the scaling coefficients can be defined by the user.

Steps	Move Command	Action by Move Command
1. Define master-slave relationship	**2SS4**	Axis #2 is the slave of axis #4
2. Define slave axis jog vel. Update interval	**2SI100**	Update slave axis jog velocity every 100 milliseconds
3. Define slave axis scaling coefficients	**2SK0.5, 0**	Specify scaling coefficients
4.Define slave axis trajectory mode	**2TJ6**	Set slave axis trajectory mode
5. Move the master axis physically		

Table 4.6: Slave to a Trackball Steps

4.3.4 Slave to a Joystick

If the slave axis is required to jog based on a DIO bit status (such as through joystick), follow these steps:

Steps	Move Command	Action by Move Command
1. Assign DIO bits for jogging slave axis	**2BP0.1**	Jog axis #2 in negative direction if DIO bit #0 is low. Jog axis #2 in positive direction if DIO bit #1 is low.
2. Enable DIO bits for jog mode	**2BQ1**	
3. Define slave axis jog vel. Update interval	**2SI100**	Update slave axis jog velocity every 100 milliseconds
4. Define slave axis scaling coefficients	**2SK0.5, 0**	Specify scaling coefficients
5. Define slave axis trajectory mode	**2TJ6**	Set slave axis trajectory mode
6. Change DIO bit value physically		

Table 4.7: Slave to a Joystick Steps

Refer to the description of the ASCII commands in Section 3: Remote Mode, for additional description, correct syntax, parameter ranges, etc.

4.4 Closed Loop Stepper Motor Positioning

4.4.1 Introduction – Closed Loop Stepper

Most of the electro-mechanical systems are subjected to phenomena such as backlash and friction. Due to such physical attributes, a significant position error can be generated when systems are moved from one position to another by stepper motors without any closed loop control mechanism. This error can be further accentuated by micro-stepping and non-collection of encoders (necessary to have closed loop control) and motors. ESP series of motion controller's support closed loop positioning of stepper motors to eliminate such errors.

The next subsection details the implementation of this feature in ESP controllers.

4.4.2 Feature Implementation

While closed loop control of stepper motors can be done during tracking as well as regulation, ESP controllers' closed loop stepping feature is effective only during regulation, i.e., desired motion is completed and the motor is holding position. This was done in order to avoid tuning of control gains such as proportional (Kp), integral (Ki), derivative (Kd) gains, etc. Users need to only enable the feature and define two (2) parameters – desired deadband and closed loop update interval.

The following block diagram illustrates this feature. When the desired motion is completed, the controller calculates position error and evaluates if the error is within the user-specified deadband. If it is, no further corrective actions are commanded. On the other hand, if the error is larger than the desired value, the controller starts the closed loop update interval timer and issues commands to make necessary corrections.

It, then, waits for the timer to reset before checking the position error again. This process is repeated until the position error reduces to the desired value (deadband). The corrective actions taken by the controller to reduce positioning error are dependent upon the way in which the stepper motors are controlled: digital (pulse generation) or analog (sinusoidal commutation).

Figure 4.6: Block Diagram of Closed Loop Stepper Motor Positioning

In the case of digitally controlled stepper motors, new corrective move commands are internally issued by the controller. In the case of Commutated stepper motors, the electrical angle is adjusted.

The following steps (See **Table 4.10**) may be followed to setup the closed loop stepper motor positioning.

Steps	ASCII Command	Action by Controller
1. Set feedback configuration.	**1ZB300**	Enable encoder feedback and closed loop positioning of stepper motors for axis #1.
2. Specify deadband value.	**1DB1**	Set deadband value for axis #1 to 1 encoder count.
3. Specify closed loop update interval.	**1CL50**	Set closed loop update interval for axis #1 to 50 milliseconds.

Table 4.8: An Example of Closed Loop Stepper Motor Positioning Setup

Commands related to closed loop stepper positioning are listed in **Table 4.11** (refer to Section 3: Remote Mode, for additional details):

Command	Description
ZB	Set feedback configuration.
DB	Specify deadband value.
CL	Specify closed loop update interval.

Table 4.9: Closed Loop Stepper Positioning Commands

4.5 Synchronize Motion to External and Internal Events

4.5.1 Introduction – Synchronize Motion

Certain applications require the use of inputs from an external source to command the motion controller to perform certain tasks. These tasks can be to either initiate motion of desired axes (written in a user's stored program) or to inhibit motion of desired axes or, more simply, to just monitor the motion status of these axes. ESP series of motion controller's address these issues by taking advantage of the digital I/O interface available on the controller.

The 24 digital I/O (DIO) bits are divided into three (3) ports: A, B and C (ESP100 and ESP300 motion controllers have access to only ports A and B). Port A covers DIO bits 0 – 7, port B covers bits 8 – 15 and port C covers bits 16 – 23.

The direction of each port can be setup to be either input or output. If a port is configured to be an input, the DIO bits that belong to that port can only report the state -–HIGH or LOW logic level – of the corresponding DIO hardware. On the other hand, if the port is configured to be an output, the DIO bits in that port can be used to either set or clear the state of the corresponding hardware. Each DIO bit has a pull-up resistor to +5V. As a result, all bits will be at HIGH logic if not connected to external circuit and configured as input. Furthermore, the direction of all the ports is set to input by default following a controller reset.

The next section details the way in which these DIO bits can be used to initiate the motion of desired axes through stored programs. The subsequent sections outline the way to inhibit the motion of desired axes and to monitor the motion status of these axes using DIO bits.

4.5.2 Using DIO to Execute Stored Programs

ESP series of motion controllers can synchronize the initiation of any motion profile to external events. In order to accomplish this task, users must write their desired motion profile as a stored program and assign this stored program to a desired DIO bit.

The direction of the DIO port this DIO bit belongs to must then be set to "input" in order for the controller to detect the external event. Once these preliminaries are completed, the controller will execute the user specified stored program whenever it detects a change in the state – HIGH to LOW logic level – of the corresponding DIO hardware. Please review the examples below for further clarifications.

Example 1:

```
EP ABS0MM          | Define stored program called "Abs0mm"
1MO; 2MO           | Turn axes 1,2 ON
1TJ1; 2TJ1         | Set trajectory mode for axes 1,2 to TRAPEZOID
1PA0; 2PA0         | Move axes 1,2 to absolute 0 units
1WS100; 2WS100     | Wait for axes 1,2 motion to complete
QP                 | End of program
0BG ABS0MM         | Assign DIO #0 to run stored program called
                   | "Abs0mm"
BO 04H             | 04H = (0100)Binary
                   | Set DIO ports A,B to input and port C to output
                   | i.e., set bits 0 – 15 to input and 16 – 23 to output
```

After the above commands are sent to the controller, the controller will execute the stored program called "Abs0mm" when DIO bit #0 changes its state from HIGH to LOW logic level.

Example 2:

```
EP CYC2MM          | Define stored program called "Cyc2mm"
1MO; 2MO           | Turn axes 1,2 ON
1TJ1; 2TJ1         | Set trajectory mode for axes 1,2 to TRAPEZOID
1PA0; 2PA0         | Move axes 1,2 to absolute 0 units
1WS100; 2WS100     | Wait for axes 1,2 motion to complete
DL LOOP            | Define a label called "LOOP"
1PR2; 2PR2         | Move axes 1,2 by relative 2 units
1WS100; 2WS100     | Wait for axes 1,2 motion to complete
1PR-2; 2PR-2       | Move axes 1,2 by relative –2 units
1WS100; 2WS100     | Wait for axes 1,2 motion to complete
JL LOOP,10         | Jump to label called "LOOP" 10 times
QP                 | End of program
1BG CYC2MM         | Assign DIO #1 to run stored program called
                   | "Cyc2mm"
BO 04H             | 04H = (0100)Binary
                   | Set DIO ports A, B  to input and port C to output
                   | i.e., set bits 0 – 15 to input and 16 – 23 to output
```

After the above commands are sent to the controller, the controller will execute "Cyc2mm" stored program when DIO bit #1 changes its state from HIGH to LOW logic level.

4.5.3 Using DIO to Inhibit Motion

ESP series of motion controllers can inhibit the motion of any axis in response to external events. In order to accomplish this task, users must define the DIO bit to be employed to inhibit the motion of a desired axis and the logic state in which that bit should be in order to inhibit motion. Once this done, the feature has to be enabled. Furthermore, the direction of the DIO port this DIO bit belongs to must be set to "input" in order for the controller to detect the external event.

At this point, if the selected axis is already in motion, and DIO bit is asserted, e-stop is executed per E-stop configuration (Refer "**ZE**" command for further details). If the axis is not moving, any new move commands are refused as long as the DIO bit is asserted. In either case, "AXIS-XX DIGITAL I/O INTERLOCK DETECTED" error is generated, where XX is the axis whose motion is inhibited through DIO. Please review the example below for further clarifications.

Example 3:

```
2BK1,1      | Use DIO bit#1 to inhibit motion of axis#2. This DIO bit
            | should be HIGH when axis #2 motion is inhibited
2BL1        | Enable inhibition of motion using DIO bits for axis#2
BO 04H      | 04H = (0100)Binary
            | Set DIO ports A,B to input and port C to output
            | i.e., set bits 0 – 15 to input and 16 – 23 to output
```

After the above commands are sent to the controller, the controller will inhibit the motion of axis #2 when DIO bit is at a HIGH logical level, and generate appropriate error message.

4.5.4 Using DIO to Monitor Motion Status

User's applications can monitor motion status – desired axis is in motion or standstill – through ESP motion controller's DIO. This status bit can in turn be used to drive external processes such as turning on/off a mechanical brake, for instance. In order to accomplish this task, users must define the DIO bit to be employed to monitor the motion status of a desired axis and the logic state in which that bit should be in when the axis is not in motion. Once this is done, the feature has to be enabled. Furthermore, the direction of the DIO port this DIO bit belongs to must be set to "output" in order for the controller to report the motion status.

At this point, if the selected axis is not in motion, the DIO bit changes its state to the level specified as described earlier. Please review the example below for further clarifications.

Example 3:

```
2BM9,1      | Use DIO bit #9 to indicate motion status of axis #2. This DIO
            | bit will be set to HIGH when axis #2 is not in motion
2BN1        | Enable notification of motion status using DIO for axis #2
BO 06H      | 06H = (0110)Binary
            | Set DIO port A, to input and ports B, C to output
            | i.e., set bits 0 – 7 to input and 8 – 23 to output
```

After the above commands are sent to the controller, the controller will set DIO bit #9 to a HIGH logical level when axis #2 is not in motion.

Commands related to utilizing DIO for initiating/inhibiting motion of desired axis and notifying motion status of these axes are listed in the table below (refer to Section 5: Programming, for additional details):

Command	Description
BG	Assign DIO bits to execute stored programs
BK	Assign DIO bits to inhibit motion
BL	Enable DIO bits to inhibit motion
BM	Assign DIO bits to notify motion status
BN	Enable DIO bits to notify motion status
BO	Set DIO port A, B, C direction

Table 4.10: Commands to Synchronize Motion to External Events

4.6 Position Compare Output Triggering

4.6.1 Introduction – Position Compare Output Triggering

Certain applications require the triggering of an external event when a desired axis crosses a specified position. On other occasions, an external event may have to be triggered every time the axis is displaced by a specified value. This feature can be advantageously employed in applications such as laser machining. Wherein the firing of a laser has to be synchronized to a certain position crossing or a laser has to be fired at equal distances.

ESP300 motion controllers accomplish this feature by taking advantage of the sophisticated hardware available on the controller. Since this feature is implemented in hardware, as opposed to CPU polling, the timing latency is virtually negligible. As a result, the external events can be triggered precisely even while a stage/positioner is moving.

The next section outlines the way in which user applications can setup this feature through one ASCII command. The subsequent section details the hardware required.

4.6.2 Feature Setup – Position Compare Output Triggering

User applications must issue an ASCII command, PC, with appropriate parameter values in order for the controller to trigger external events. This ASCII command has two parameters: **nn1** and **nn2.** The parameter **nn1** is used to specify the data acquisition mode.

- **nn1** = 0: disarm (disable) the feature
- **nn1** = 1: trigger an external event whenever an axis' encoder crosses the absolute position defined by **nn2.**
- **nn1** = 2: trigger an external event whenever an axis' encoder displaces by a relative distance defined by **nn2** independent of direction.

Mode #1 is applicable, for instance, when a single laser firing has to be synchronized to a specified position crossing. Mode #2 is applicable when a laser has to be fired at equal distances.

If this command is issued when the desired axis is at standstill, the worst-case error in capturing a position crossing is within ± 1 encoder count.

However, if this ASCII command is issued when the desired axis is in motion, the accuracy of the position crossing in absolute mode (nn1 = 1) depends upon the speed at which the axis was moving when the command was processed by the DSP.

For instance, if the axis was moving at 40 mm/sec, and the encoder resolution is 10000 counts/mm, the **worst**-case error in capturing a position would be approximately 3 counts. Please see calculation below:

Example 1:

Worst case error = ± (Axis speed at setup time) * 7 μs
= ± 40 mm/sec * 10000 counts/mm * 7 μs
= ± 2.8 counts

The timing latency between the registration of a position crossing by the hardware and sending out a trigger (TTL) pulse is 320ns. The pulse width duration can be between 30 and 60ns. Please see the timing diagram below for further details.

where Td = propagation delay and Tp = output pulse width

Figure 4.7: Timing Diagram of a TTL Pulse Generation Synchronized to Position Crossing

Please refer to the following examples for further clarifications.

Example 2

The following program will home (initialize) axis #1 and move to an absolute target position. Before the motion is started the controller will be configured to generate a single, precise TTL pulse when absolute +12.34 is crossed.

```
EP ABSCOMPARE  // start program entry mode
1MO            // enable axis-1 motor power
1 OR 1         // home axis-1
1 WS 1000      // wait for home completion and dwell 1 second
1 PC 1, +12.34 // arm absolute position compare trigger output pulse
               // at +12.34
1 PA +15.0     // start absolute position move to location +15.0
1 WS 0         // wait for home completion
1 PC 0, 0      // disarm compare trigger output pulse mode
QP             // end program entry mode
```

Example 3:

The following program will home (initialize) axis #1 and move to an absolute target position. Before the motion is started, the controller will be configured to generate a precise TTL pulse every +1.00 units of relative distance.

```
EP RELCOMPARE  // start program entry mode
1 MO           // enable axis-1 motor power
1 OR 1         // home axis-1
1WS 1000       // wait for home completion and dwell 1 second
```

```
1 PC 2, +1.00       // arrm relative position compare trigger pulse every
                    // +1.00 units
1 PA +15.0          // start absolute position move to location +15.0
1 WS 0              // wait for home completion
1 PC 0, 0           // disarm compare trigger output pulse mode
QP                  // end program entry mode
```

NOTE: This feature is implemented using the auxiliary counter channels 7 and 8 in conjunction with axes counters 1 and 2. Therefore, if either axis 1 or 2 are configured in any compare mode then auxiliary counters channel 7 and 8 are used and not available (e.g., cannot be used for trackball or master/slave modes). Counters 7 & 8 are released and made available when the compare mode is disarmed (e.g., "xxPC 0").

4.6.3 Hardware Required – Position Compare Output Triggering

The trigger pulses generated by the motion controller can be accessed through Auxiliary I/O connector pins 17 and 18. Pin #17 is used in conjunction with axis #1 and pin #18 in conjunction with axis #2. Both of these outputs are buffered by IC P/N SN74F21. As a result, they can source 1mA (maximum) or sink 20mA (maximum). Please refer to the description of these pins in the **Auxiliary I/O (40-Pin) JP5 Connector** section in Appendix C.

Commands related to generating a trigger pulse in synchronization with axis position are listed in the table below (refer to Section 3, Remote Mode, for additional details):

Command	Description
PC	Set position compare mode

Table 4.11: Commands to Synchronize External Events to Axis Position

4.7 Position Capture Input Triggering

4.7.1 Introduction – Position Capture Input Triggering

Certain applications require the capture of desired axes' position in response to an input from an external source. Applications may also require certain internal events to be triggered following the position capture. The internal events can be, for instance, commanding an axis to move to a specific location, or executing a stored program. This feature can be advantageously employed, for instance, in metrology applications wherein the controller can be commanded to capture the position of desired axes when a probe touches a desired object.

ESP300 notion controllers accomplish this feature by taking advantage of the sophisticated hardware available on the controller. Since this feature is implemented in hardware, the timing latency is virtually negligible. As a result, the position of desired axes can be captured very accurately even while a stage/positioner is moving at high speed.

The next section outlines the way in which user applications can setup this feature through a few ASCII commands. The subsequent section details the hardware required.

4.7.2 Feature Setup – Position Capture Input Triggering

User applications must issue an ASCII command, **DC,** with appropriate parameter values in order for the controller to capture the position of desired axes. This command is used to setup the data acquisition mode. If the controller has to trigger any internal events after the position capture, the ASCII command, **ES,** has to be issued to specify the desired event action. Note that **ES** command is optional, and it is required only if the controller has to take a particular action following the position capture. After this command is issued, the data acquisition process must be enabled by issuing the ASCII command, **DE.** Once the data acquisition is enabled, the specified axes' position will be captured by the controller and necessary actions initiated as defined by commands **DC** and **ES.** The ASCII command **DD** can be used to monitor if all the desired position samples have been collected by the controller. If the command **DD** responds with a value of one (1) meaning data acquisition is done, all the samples collected by the controller can be retrieved by issuing the ASCII command, **DG.** Please review the examples below for further clarifications.

Example 1:

The following example sets up the data acquisition mode to acquire two analog channels (1 and 2) and two position channels (1 and 3) data when an external event trigger is detected by the controller. Furthermore, it turns OFF axis #1 motor power after data has been captured.

```
DC3,1,3,5,0,1
                Param. #1: Acquire analog/position data on an external trigger
                // Param. #2: No consequence for this DAQ mode
                // Param. #3: Acquire analog channels 1 & 2 (3₁₀ = (101)₂ )
                // Param. #5: No consequence for this DAQ mode
                // Param. #6: Acquire one sample of data
ES 1MF          // Turn OFF motor #1 when the external event trigger occurs
DE1             // Enable analog data acquisition
DD              // Query data-acquisition done status
   1 = true, 0 = false.
```

If true,
DEO // *Disable trace variable data acquisition*
DG // *Get data collected*

The user specified analog and/or encoder (position) data will be captured 60ns after falling edge of position capture input trigger. If there is a de-bounce on the input signal, the data is captured 20ns after the last falling edge. Please see the timing diagram below for further details.

Figure 4.8: Timing Diagram of Position Capture Synchronized to Extend input Trigger

4.7.3 Hardware Required – Position Capture Input Triggering

The external input required to trigger the capture of desired axes' position must be connected to Auxiliary I/O connector pin 20. This input is pulled-up to +5 volts with a 1KΩ resistor. The input pulse must be a low going pulse, with a width of at least 80ns. Please refer to the description of these pins in the **Auxiliary I/O (40-Pin) JP5 Connector** section in Appendix C.

Commands related to generating a trigger pulse in synchronization with axis position are listed in **Table 4.12** (refer to Section 3, Remote Mode, for additional details):

Command	Description
DC	Setup data acquisition
DD	Query data acquisition done status
DE	Enable or disable data acquisition
DG	Get acquired data
ES	Define event action

Table 4.12: Commands to Synchronize Position Capture to External Inputs

Section 5 – Motion Control Tutorial

5.1 Motion Systems

A schematic of a typical motion control system is shown in **Figure 5.1.**

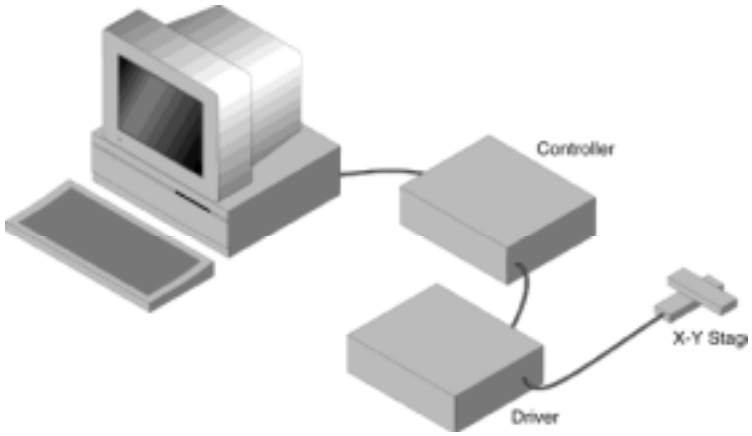


Figure 5.1: Typical Motion Control Systems

Its major components are:

Controller

An electronic device that receives motion commands from an operator directly or via a computer, verifies the real motion device position and generates the necessary control signals.

Driver

An electronic device that converts the control signals to the correct format and power needed to drive the motors.

Motion Device

An electro-mechanical device that can move a load with the necessary specifications.

Cables

Needed to interconnect the other motion control components.

If the user is like most motion control users, they started by selecting a motion device that matches certain specifications needed for an application. Next, the user should choose a controller that can satisfy the motion characteristics required. The chances are that the user is less interested in how the components look or what their individual specifications are, but want to be sure that together they perform reliably according to their needs.

We mentioned this to make a point. A component is only as good as the system lets (or helps) it to be.

For this reason, when discussing a particular system performance specification, we will also mention which components affect performance the most and, if appropriate, which components improve it.

5.2 Specification Definitions

People mean different things when referring to the same parameter name. To establish some common ground for motion control terminology, here are some general guidelines for the interpretation of motion control terms and specifications.

- As mentioned earlier, most motion control performance specifications should be considered system specifications.
- When not otherwise specified, all error-related specifications refer to the position error.
- The servo loop feedback is position-based. All other velocity, acceleration, error, etc. parameters are derived from the position feedback and the internal clock.
- To measure the absolute position, we need a reference, a measuring device, that is significantly more accurate than the device tested. In our case, dealing with fractions of microns (0.1 μm and less), even a standard laser interferometer becomes unsatisfactory. For this reason, all factory measurements are made using a number of high precision interferometers, most of them connected to a computerized test station.

- To avoid unnecessary confusion and to more easily understand and troubleshoot a problem, special attention must be paid to avoid bundling discrete errors in one general term. Depending on the application, some discrete errors are not significant. Grouping them in one general parameter will only complicate the understanding of the system performance in certain applications.

5.2.1 Following Error

The Following Error is not a specifications parameter but, because it is at the heart of the servo algorithm calculations and of other parameter definitions, it deserves our attention.

As will be described later in Section 5.3: Control Loops, a major part of the servo controller's task is to make sure that the actual motion device follows as close as possible an ideal trajectory in time. The user can imagine having an imaginary (ideal) motion device that executes exactly the motion profile they are requesting. In reality, the real motion device will find itself deviating from this ideal trajectory. Since most of the time the real motion device is trailing the ideal one, the instantaneous error is called Following Error.

To summarize, the Following Error is the instantaneous difference between the actual position as reported by the position feedback device and the ideal position, as seen by the controller. A negative following error means that the load is trailing the ideal motion device.

5.2.2 Error

Error has the same definition as the Following Error with the exception that the ideal trajectory is not compared to the position feedback device (encoder) but to an external precision-measuring device.

In other words, the Following Error is the instantaneous error perceived by the controller while the Error is the one perceived by the user.

5.2.3 Accuracy

The accuracy of a system is probably the most common parameter users want to know. Unfortunately, due to its perceived simplicity, it is also the easiest to misinterpret.

The Accuracy is a static measure of a point-to-point positioning error. Starting from a reference point, the user should command the controller to move a certain distance. When the motion is completed, the user should measure the actual distance traveled with an external precision-measuring device. The difference (the Error) represents the positioning Accuracy for that particular motion.

Because every application is different, the user needs to know the errors for all possible motions. Since this is practically impossible, an acceptable compromise is to perform the following test.

Starting from one of travel, the user can make small incremental moves and at every stop, the user should record the position Error. The user performs this operation for the entire nominal travel. When finished, the Error data is plotted on a graph similar to **Figure 5.2.**

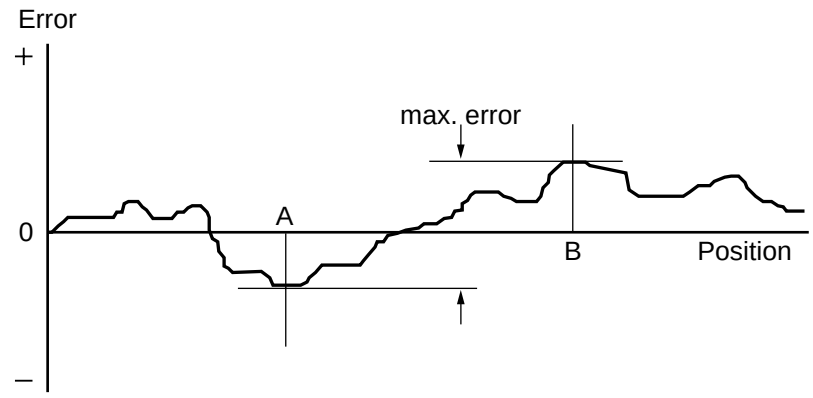


Figure 5.2: Position Error Test

The difference between the highest and the lowest points on the graph is the maximum possible Error that the motion device can have. This worst-case number is reported as the positioning Accuracy. It guaranties the user that for any application, the positioning error will not be greater than its value.

5.2.4 Local Accuracy

For some applications, it is important to know not just the positioning Accuracy over the entire travel but also over a small distance. To illustrate this case, **Figure 5.3a** and **Figure 5.3b** shows two extremes cases.

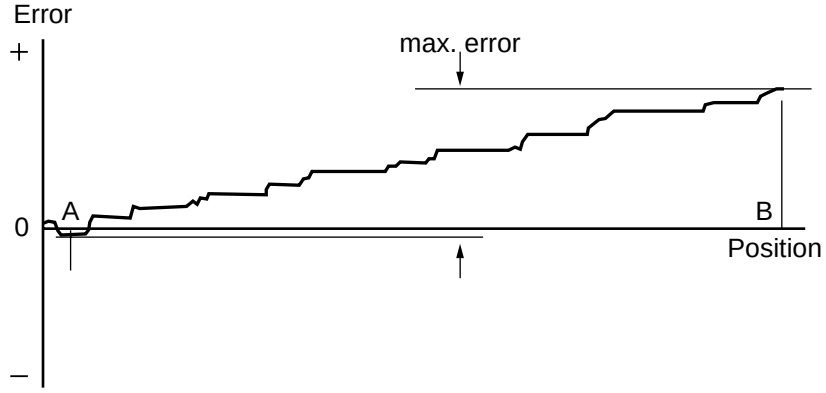


Figure 5.3a: High Accuracy for Small Motions

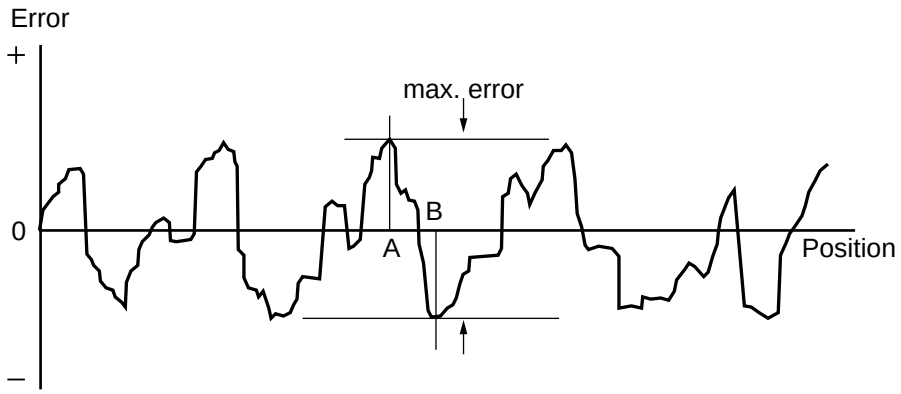


Figure 5.3b: Low Accuracy for Small Motions

Both error plots from **Figure 5.3a** and **Figure 5.3b** have a similar maximum Error. But, if the user compares the maximum Error for small distances, the system in **Figure 5.3b** shows significantly larger values.

For applications requiring high accuracy for small motions, the system in **Figure 5.3a** is definitely preferred.

"Local Error" is a relative term that depends on the application; usually no Local Error value is given with the system specifications. The user should study the error plot supplied with the motion device and determine the approximate maximum Local error for the specific application.

5.2.5 Resolution

Resolution is the smallest motion that the controller attempts to make. For all DC motor and most all standard stepper motor driven stages supported by the ESP7000, this is also the resolution of the encoder.

Keeping in mind that the servo loop is a digital loop, the Resolution can be also viewed as the smallest position increment that the controller can handle.

5.2.6 Minimum Incremental Motion

The Minimum Incremental Motion is the smallest motion that a device can reliably make, measured with an external precision-measuring device. The controller can, for instance, execute a motion equal to the Resolution (one encoder count) but in reality, the load may not move at all. The cause for this is in the mechanics.

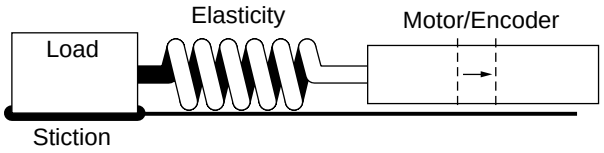


Figure 5.4: Effect of Stiction and Elasticity on Small Motions

Figure 5.4 shows how excessive stiction and elasticity between the encoder and the load can cause the motion device to deviate from ideal motion when executing small motions.

The effect of these two factors has a random nature. Sometimes, for a small motion step of the motor, the load may not move at all. Other times, the accumulated energy in the spring will cause the load to jump a larger distance. The error plot will be similar to **Figure 5.5.**

Figure 5.5: Error Plot

Once the Minimum Incremental Motion is defined, the next task is to quantify it. This is more difficult for two reasons: one is its random nature and the other is in defining what a *completed motion* represents.

Assume that the user has a motion device with a 1 μm resolution. If every time the user commands a 1 μm motion the measured error is never greater than 2%, the user will probably be very satisfied and declare that the Minimum Incremental Motion is better than 1 μm, If, on the other hand, the measured motion is sometimes as small as 0.1 μm (a 90% error), the user could not say that 1 μm is a reliable motion step. The difficulty is in drawing the line between acceptable and unacceptable errors when performing a small motion step. The most common value for the maximum acceptable error for small motions is 20%, but each application ultimately has its own standards.

One way to solve the problem is to take a large number of measurements (a few hundred at minimum) for each motion step size and present them in a format that an operator can use to determine the Minimum Incremental Motion by its own standards.

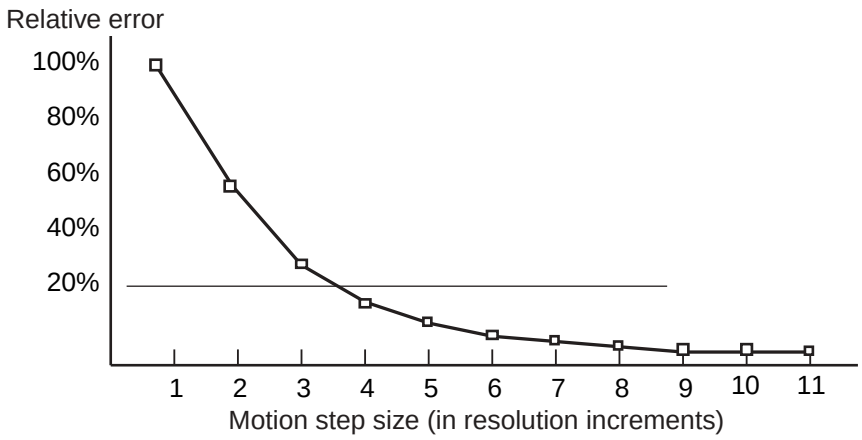


Figure 5.6: Error vs. Motion Step Size

Figure 5.6 shows an example of such a plot. The graph represents the maximum relative error for different motion step sizes. In this example, the Minimum Incremental Motion that can be reliably performed with a maximum of 20% error is one equivalent to 4 resolution (encoder) increments.

5.2.7 Repeatability

Repeatability is the positioning variation when executing the same motion profile. Assuming that the user has a motion sequence that stops at a number of different locations, the Repeatability is the maximum variation in position all targets when the same motion sequence is repeated a large number of times. It is a relative, not absolute, error between identical motions.

5.2.8 Backlash (Hysteresis)

For all practical purposes, Hysteresis and Backlash have the same meaning for typical motion control systems; the error caused by approaching a point from a different direction. The difference is that Hysteresis refers to the compliance of the mechanical components, while Backlash represents the "play", or looseness, in the mechanical drive train.

All parameters discussed up to now that involve the positioning Error assumed that all motions were performed in the same direction. If the user tries to measure the positioning error of a certain target (destination), approaching the destination from difficult directions could make a significant difference.

In generating the plot in **Figure 5.2** we said that the motion device will make a large number of incremental moves, from one end of travel to the other. If the user commands the motion device to move back and stop at the same locations to take a position error measurement, the user would expect to get an identical plot, superimposed on the first one. In reality, the result could be similar to **Figure 5.7.**

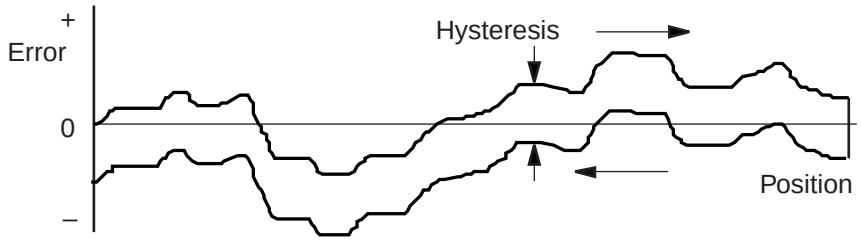


Figure 5.7: Hysteresis Plot

The error plot in reverse direction is identical with the first one but seems to be shifted down by a constant error. This constant error is the Hysteresis of the system.

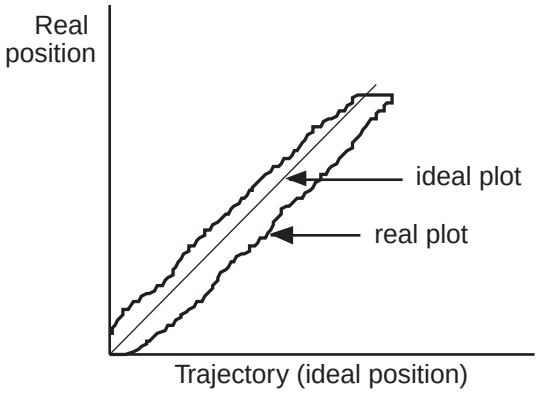


Figure 5.8: Real vs. Ideal Position

To justify a little more why we call this Hysteresis, lets do the same graph in a different format (**Figure 5.8**). Plotting the real versus the ideal position will give the user a familiar hysteresis shape.

5.2.9 Pitch, Roll and Yaw

These are the most common angular error parameters for linear translation stages. They are pure mechanical errors and represent the rotational error of a stage carriage around the three axes. A perfect stage should not rotate around any of the axes, thus the Pitch, Roll and Yaw should be zero.

The commonly used representation of the three errors is shown in **Figure 5.9.** Pitch is rotation around the Y axis, Roll is rotation around the X axis, and Yaw is rotation around the Z axis.

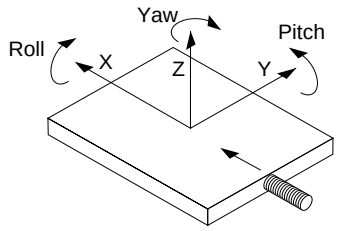


Figure 5.9: Pitch, Roll and Yaw Motion Axes

The problem with this definition is that, through correct, it is difficult to remember. A more graphical representation is presented in **Figure 5.10.** Imagine a tiny carriage driven by a giant leadscrew. When the carriage rolls sideways on the lead screw, we call it a roll. When it rides up and down on the lead screw pitch, we call that Pitch. And, when the carriage deviates left or right from the straight direction(on an imaginary Y trajectory), we call it yaw.

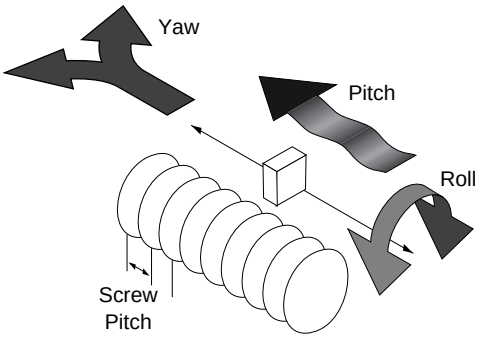


Figure 5.10: Pitch, Yaw and Roll

5.2.10 Wobble

This parameter applies only to rotary stages. It represents the deviation of the axis of rotation during motion. A simple form of wobble is a constant one, where the rotating axis generates a circle (**Figure 5.11**).

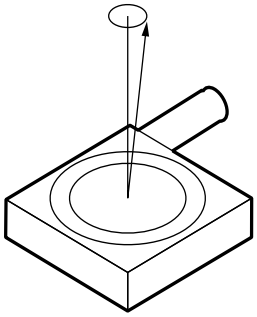


Figure 5.11: Wobble Form

A real rotary stage may have a more complex Wobble, where the axis of rotation follows a complicated trajectory. This type of error is caused by the imperfections of the stage machining and/or ball bearings.

5.2.11 Load Capacity

There are two types of loads that are of interest for motion control applications: static and dynamic loads.

The static :pad Capacity represents the amount of load that can be placed on a stage without damaging or excessively deforming it. Determining the Load Capacity of a stage for a particular application is more complicated than it may first appear. The stage orientation and the distance from the load to the carriage play a significant role. For a detailed description on how to calculate the static Load Capacity, please consult the motion control catalog tutorial section.

The dynamic Load Capacity refers to the motor's effort to move the load. The first parameter to determine is how much load the stage can push or pull. In some cases the two values could be different due to internal mechanical construction.

The second type of dynamic Load Capacity refers to the maximum load that the stage could move with the nominal acceleration.. This parameter is more difficult to specify because it involves defining an acceptable following error during acceleration.

5.2.12 Maximum Velocity

The Maximum Velocity that could be used in a motion control system is determined by both motion device and driver. Usually it represents a lower value than the motor or driver are capable of. In most cases, including the ESP7000, the default Maximum Velocity should be increased. The hardware and firmware are tuned for a particular maximum velocity that cannot be exceeded.

5.2.13 Minimum Velocity

The Minimum Velocity usable with a motion device depends on the motion control system but also on the acceptable velocity regulation. First, the controller sets the slowest rate of motion increments it can make. The encoder resolution determines the motion increment size and then, the application sets a limit on the velocity ripple.

To illustrate this, take the example of a linear stage with a resolution of 0.1 μm. If the user sets the velocity to 0.5 μm/sec, the stage will move 5 encoder counts in one second. But, properly tuned servo loop could move the stage 0.1 μm in about 20 ms. The position and velocity plots are illustrated in **Figure 5.12.**

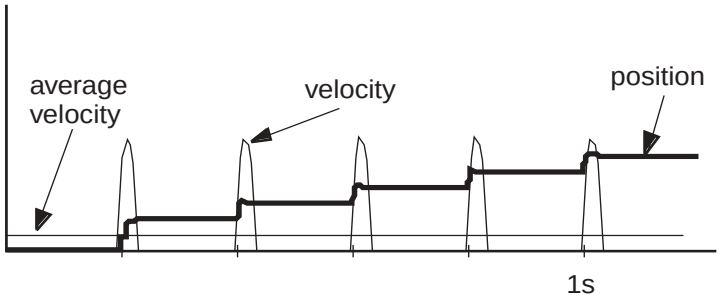


Figure 5.12: Position, Velocity and Average Velocity

The average velocity is low but the velocity ripple is very high. Depending on the application, this may be acceptable or not. With increasing velocity, the ripple decreases and the velocity becomes smoother.

This example is even more true in the case of a stepper motor driven stage. The typical noise comes from a very fast transition from one step position to another. The velocity ripple in that case is significantly higher.

In the case of a DC motor, adjusting the PID parameters to get a softer response will reduce the velocity ripple but care must be taken not to negatively affect other desirable motion characteristics.

5.2.14 Velocity Regulation

In some applications, for example scanning, it is important for the velocity to be very constant. In reality, there are a number of factors besides the controller that affect the velocity.

As described in the Minimum Velocity definition, the speed plays a significant role in the amount of ripple generated, especially at low values.

Even if the controller does a perfect job by running with zero following error, imperfections in the mechanics (friction, variation, transmission ripple, etc.) will generate some velocity ripple that can be translated to Velocity Regulation problems.

Depending on the specific application, one motor technology can be preferred over the other.

As far as the controller is concerned, the stepper motor version is the ideal case for a good average Velocity Regulation because the motor inherently follows precisely the desired trajectory. The only problem is the ripple caused by the actual stepping process.

The best a DC motor controller can do is to approach the stepper motor's performance in average Velocity Regulation, but it has the advantage of significantly reduced velocity ripple, inherently and through PID tuning. If the DC motor implements a velocity closed loop through the use of a tachometer, the overall servo performance increases and one of the biggest beneficiary is the Velocity Regulation.

5.2.15 Maximum Acceleration

The Maximum Acceleration is a complex parameter that depends as much on the motion control system as it does on application requirements. For stepper motors, the main concern is not to loose steps 9or synchronization) during the acceleration. Besides the motor and driver performance, the load inertia plays a significant role.

For DC motor systems the situation is different. If the size of the following error is of no concern during the acceleration, high Maximum Acceleration values can be entered. The motion device will move with the highest natural acceleration it can (determined by the motor, driver, load inertia, etc.) and the errors will consist of just a temporary larger following error and a velocity overshoot.

In any case, special consideration should be given when setting the acceleration. Though in most cases no harm will be done in setting a high acceleration value, avoid doing so if the application does not require it. The driver, motor, motion device and load undergo maximum stress during high acceleration.

5.2.16 Combined Parameters

Very often a user looks at an application and concludes that they need a certain overall accuracy. This usually means that the user is combining a number of individual terms (error parameters) into a single one. Some of this combined parameters even have their own name, even though not all people mean the same thing by them: Absolute Accuracy, Bi-directional Repeatability, etc. The problem with these generalizations is that, unless the term is well defined and the testing closely simulates the application, the numbers could be of little value.

The best approach is to carefully study the application, extract from the specification sheet the applicable discrete error parameters and combine them (usually add them) to get the worst-case general error applicable to the specific case. This method not only offers a more accurate value but also gives a better understanding of the motion control system performance and helps pinpoint problems.

Also, due to integrated nature of the ESP7000 system, many basic errors can be significantly corrected by another component of the loop. Backlash, Accuracy and Velocity Regulation are just a few examples where the controller can improve motion device performance.

5.3 Control Loops

When talking about motion control systems, one of the most important questions is the type of servo loop implemented. The first major distinction is between open and closed loops. Of course, this is of particular interest when driving stepper motors. As far as the DC servo loops, the PID type is by far the most widely used.

The ESP7000 implements a PID servo loop with velocity and acceleration feed-forward.

The basic diagram of a servo loop is shown in **Figure 5.13.** Besides the command interpreter, the main two parts of a motion controller are the trajectory generator and the servo controller. The first generates the desired trajectory and the second one controls the motor to follow it as closely as possible.

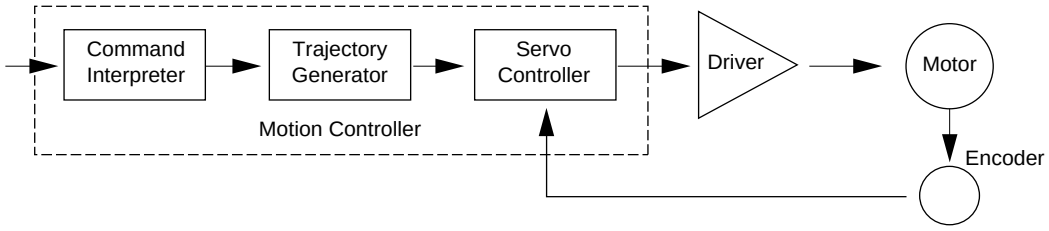


Figure 5.13: Servo Loop

5.3.1 PID Servo Loops

The PID term comes from the proportional, integral and derivative gain factors that are at the basis of the control loop calculation. The common equation given for it is:

$$K_p \cdot e + K_i \cdot \int e\, dt + K_d \cdot \frac{de}{dt}$$

where: K_p = Proportional gain factor
K_i = Integral gain factor
K_d = Derivative gain factor
e = Instantaneous following factor

The problem for most users is to get a feeling for this formula, especially when trying to tune the PID loop. Tuning the PID means changing its three gain factors to obtain a certain system response, task quite different to achieve without some understanding of its behavior.

The following paragraphs explain the PID components and their operation.

P Loop

Lets start with the simplest type of closed loop, the P (proportional) loop. The diagram in **Figure 5.14** shows its configuration.

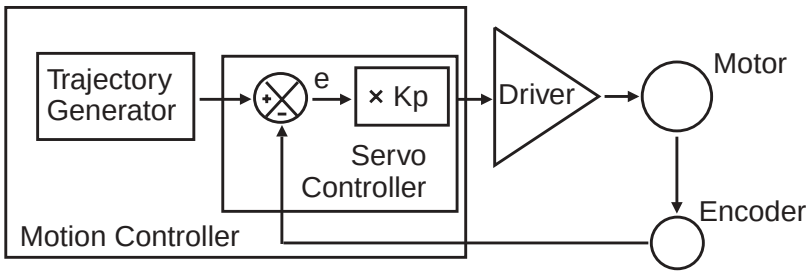


Figure 5.14: P Loop Configuration

Every servo cycle, the actual position, as reported by the encoder, is compared to the desired position generated by the trajectory generator. The difference **e** is the positioning error (the following error). Amplifying it (multiplying it by $\mathbf{K_p}$) generates a control signal that, converted to an analog signal, is sent to the motor driver.

There are a few conclusions that could be drawn from studying this circuit:

- The motor control signal, thus the motor voltage, is proportional to the following error.
- There must be a following error in order to drive the motor.
- Higher velocities need higher motor voltages and thus higher following errors.
- At stop, small errors cannot be corrected if they don't generate enough voltage for the motor to overcome friction and stiction.
- Increasing the $\mathbf{K_p}$ gain reduces the necessary following error but too much of it will generate instabilities and oscillations.

PI Loop

To eliminate the error at stop and during long constant velocity motions (usually called steady-state error), an integral term can be added to the loop. This term integrates (adds) the error every servo cycle and the value, multiplied by the $\mathbf{K_i}$ gain factor, is added to the control signal (**Figure 5.15**).

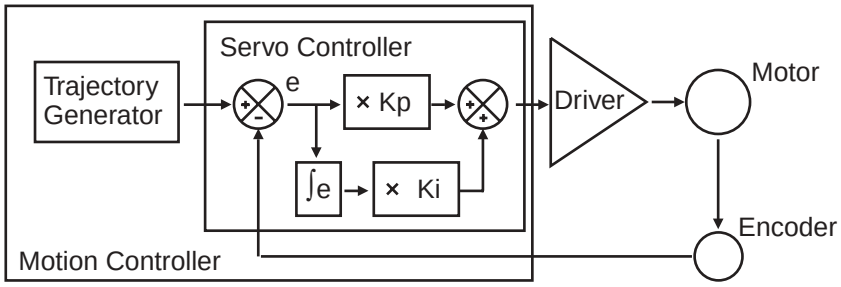


Figure 5.15: PI Loop Configuration

The result is that the integral term will increase until it drives the motor by itself, reducing the following error to zero. At stop, this has the very desirable effect of driving the positioning error to zero. During a long constant-velocity motion it also brings the following error to zero, an important feature for some applications.

Unfortunately, the integral term also has a negative side, a severe de-stabilizing effect on the servo loop. In the real world, a simple PI Loop is usually undesirable.

PID Loop

The third term of the PID Loop is the derivative term. It is defined as the difference between the following error of the current servo cycle and of the previous one. If the following error does not change, the derivative term is zero.

Figure 5.16 shows the PID servo loop diagram. The derivative term is added to the proportional and integral one. All three process the following error in their own way and, added together, form the control signal.

The derivative term adds a damping effect that prevents oscillations and position overshoot.

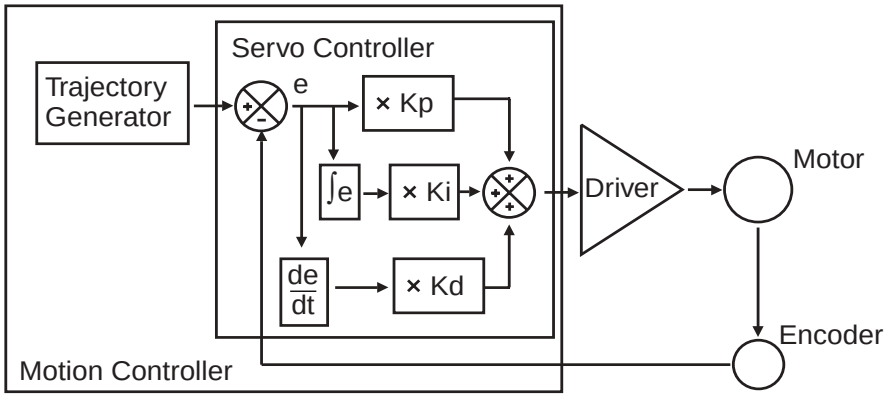


Figure 5.16: PID Loop Configuration

5.3.2 Feed-Forward Loops

As described in the previous paragraph, the main driving force in a PID loop is the proportional term. The other two correct static and dynamic errors associated with the closed loop.

Taking a closer look at the desired and actual motion parameters and at the characteristics of the DC motors, some interesting observations can be made. For a constant load, the velocity of a DC motor is approximately proportional with the voltage. This means that for a trapezoidal velocity profile, for instance, the motor voltage will have also a trapezoidal shape (**Figure 5.17**).

The second observation is that the desired velocity is calculated by the trajectory generator and is known ahead of time. The obvious conclusion is that we could take this velocity information, scale it by $\mathbf{K_{vff}}$ factor and feed it to the motor driver. If the scaling is done properly, the right amount of voltage is sent to the motor to get the desired velocities, without the need for a closed loop. Because the signal is derived from the velocity profile and it is being sent directly to motor driver, the procedure is called velocity feed-forward.

Of course, this looks like an open loop, and it is (**Figure 5.18**). But, adding this signal to the closed loop has the effect of significantly reducing the "work" the PID has to do, thus reducing the overall following error. The PID now has to correct only for the residual error left over by the feed-forward signal.

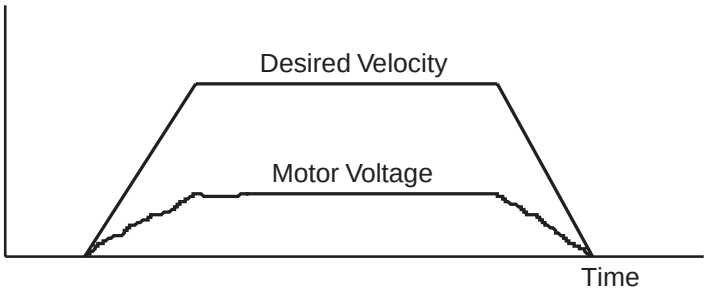


Figure 5.17: Trapezoidal Velocity Profile

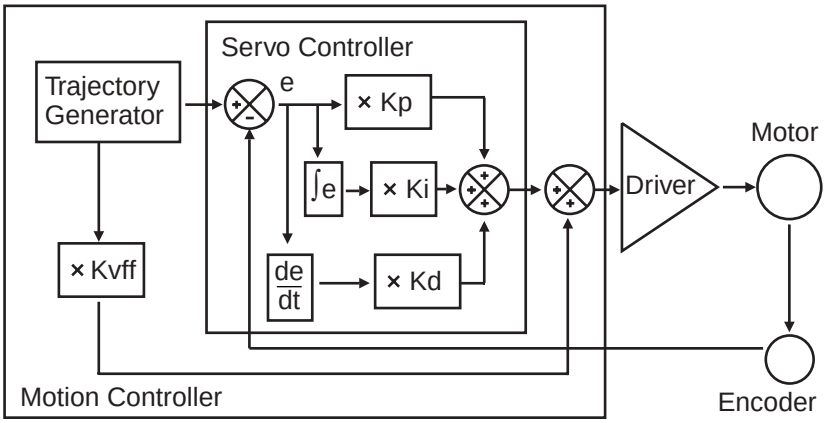


Figure 5.18: PID Loop with Feed-Forward

There is another special note that has to be made about the feed-forward method. The velocity is approximately proportional to the voltage and only for constant loads, but this true only if the driver is a simple voltage amplifier or current (torque) driver. A special case is when the driver has its own velocity feedback loop from a tachometer (**Figure 5.19**).

The tachometer is a device that outputs a voltage proportional with the velocity. Using its signal, the driver can maintain the velocity to be proportional to the control signal.

If such a driver is used with a velocity feed-forward algorithm, by properly tuning the $\mathbf{K_{vff}}$ parameter, the feed-forward signal could perform an excellent job, leaving very little for the PID loop to do.

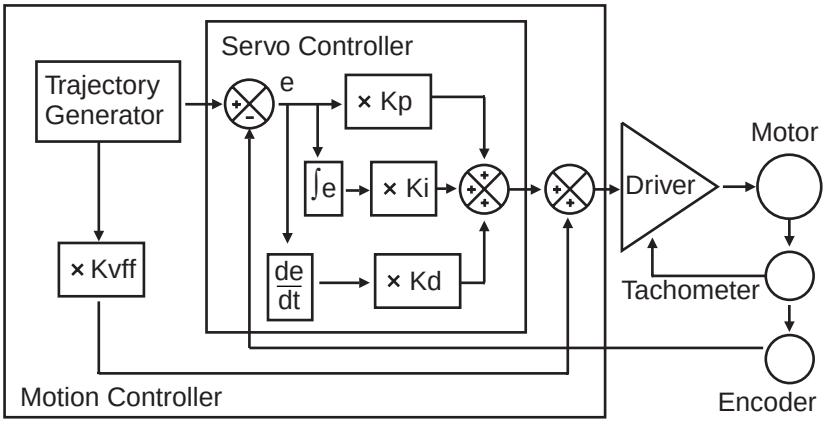


Figure 5.19: Tachometer-driven PIDF Loop

5.4 Motion Profiles

When talking about motion commands we refer to certain strings sent to a motion controller that will initiate a certain action, usually a motion. There are a number of common motion commands that are identified by name. The following paragraphs describe a few of them.

5.4.1 Move Motion

A move is a point-to-point motion. On execution of a move motion command, the motion device moves from the current position to a desired destination. The destination can be specified either as an absolute position or as a relative distance from the current position.

When executing a move command, the motion device will accelerate until the velocity reaches a pre-defined value. Then, at the proper time, it will start decelerating so that when the motor stops, the device is at the correct position. The velocity plot of this type of motion will have a trapezoidal shape (**Figure 5.20**). For this reason, this type of motion is called a trapezoidal motion.

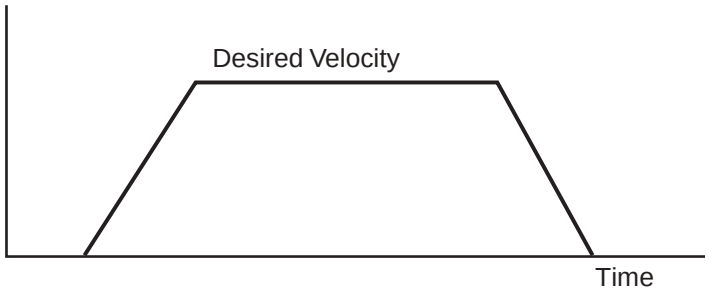


Figure 5.20: Trapezoidal Motion Profile

The position and acceleration profiles relative to the velocity are shown in **Figure 5.21.**

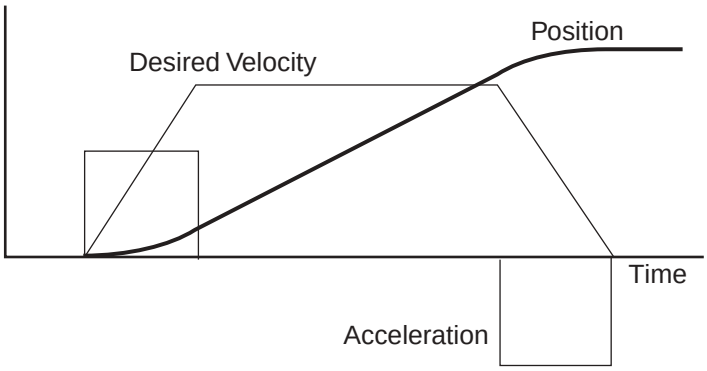


Figure 5.21: Position and Acceleration Profiles

Besides the destination, the acceleration and the velocity of the motion (the constant portion of it) can be set by the user before every move command. Advanced controllers like the ESP7000 allow the user to change them even during the motion.

5.4.2 Jog Motion

When setting up an application, it is often necessary to move stages manually while observing motion. The easy way to do this without resorting to specialized input devices such as joysticks or track-wheels is to use simple push-button switches. This type of motion is called a jog. When a jog button is pressed the selected axis starts moving with a pre-defined velocity. The motion continues only while the button is pressed and stops immediately after its release.

The ESP7000 offers two jog speeds. Both high speed and low speed are used programmable. The jog acceleration is also ten times smaller than the programmed maximum acceleration values.

5.4.3 Home Search

Home search is a specific motion routine that is useful for most types of applications. Its goal is to find a specific point in travel relative to the mounting base of the motion device very accurately and repeatable. The need for this absolute reference point is twofold. First, in many applications it is important to know the exact position in space in space, even after a power-off cycle. Secondly, to protect the motion device from hitting a travel obstruction set by the application (or its own travel limits), the controller uses programmable software limits. To be efficient though, the software limits must be placed accurately in space before running the application.

To achieve this precise position referencing, the ESP7000 motion control system executes a unique sequence of moves.

First, lets look at the hardware required to determine the position of a motion device. The most common (and the one supported by the ESP7000) are incremental encoders. By definition, these are encoders that can tell only relative moves, not absolute position. The controller keeps track of position by incrementing or decrementing a dedicated counter according to the information received from the encoder. Since there is no absolute position information, position "zero" is where the controller was powered on (and the position counter reset).

To determine an absolute position, the controller must find a "switch" that is unique to the entire travel, called a home switch or origin switch. An important requisition is that this switch must be located with the same accuracy as the encoder pulses.

If the motion device is using a linear scale as position encoder, the home switch is usually placed on the same scale and read with the same accuracy.

If on the other hand, a rotary encoder is used, the problem becomes more complicated. To have the same accuracy, a mark on the encoder disk could be used (called index pulse) but because it repeats itself every revolution, it does not define a unique point over the entire travel. An origin switch, on the other hand, placed in the travel of the motion device is unique but not accurate (repeatable) enough. The solution is to use both, following a search algorithm.

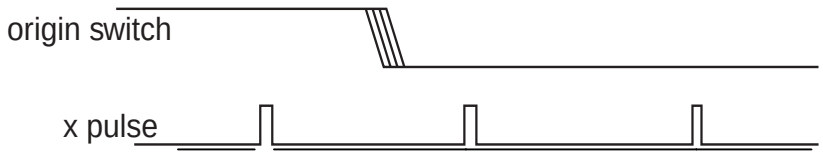


Figure 5.22: Origin Switch and Encoder Index Pulse

An origin switch (**Figure 5.22**) separates the entire travel in two areas: one for which it has a high level and one for which is low. The most important part of it is the transition between the two areas. Also, looking at the origin switch level, the controller knows on which side of the transition it currently is and which way to move to find it.

The task of the home search routine is to identify one unique index pulse as the absolute position reference. This is done by the first finding the origin switch transition and then the very first index pulse (**Figure 5.23**).

So far, we can label the two motion segments D and E. During D the controller is looking for the origin switch transition and during E for the index pulse. To guarantee the best accuracy possible, both D and E segments are performed at a very low speed and without a stop in – between. Also, during E the display update is suppressed to eliminate any unnecessary overhead.

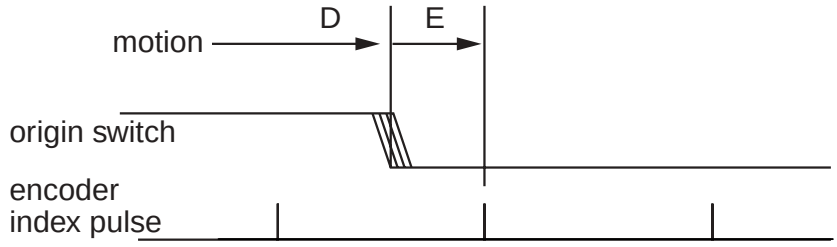


Figure 5.23: Slow-Speed Origin Switch Search

The routine described above could work but has one problem. Using the low speeds, it could take a very long time if the motion device happens to start from the opposite end of travel. To speed things up, we can have the motion device move fast in the vicinity of the origin switch and then perform the two slow motions, D and E. The new sequence is shown in **Figure 5.24.**

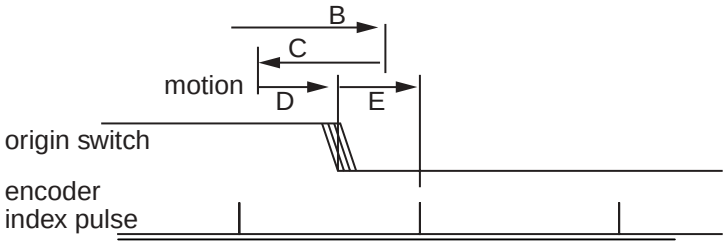


Figure 5.24: High/Low-Speed Origin Switch Search

Motion segment B is performed at high speed, with the pre-programmed home search speed. When the origin switch transition is encountered, the motion device stops (with an overshoot), reverses direction and looks for it again, this time with half the velocity (segment C).

Once found, it stops again with an overshoot, reverses direction and executes D and E with one tenth of the programmed home search speed.

In the case when the motion device starts from the other side of the origin switch transition, the routine will look like **Figure 5.25.**

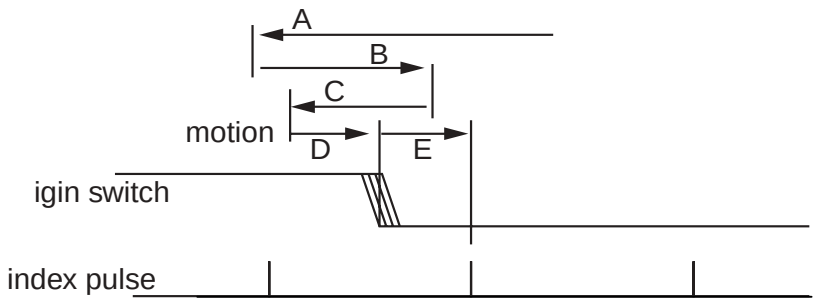


Figure 5.25: Origin Search From Opposite Direction

The ESP7000 moves at high speed up to the origin switch transition (segment A) and then executes B, C, D and E.

All home search routines are run so that the last segment, E, is performed in the positive direction of travel.

CAUTION

The home search routine is very important for the positioning accuracy of the entire system and it requires full attention from the controller. Do not interrupt or send other commands during its execution, unless it is for emergency purposes.

5.5 Encoders

PID closed-loop motion control requires a position sensor. The most widely used technology by far are incremental encoders.

The main characteristic of an incremental encoder is that it has a 2-bit gray code output, more commonly known as quadrature output (**Figure 5.26**).

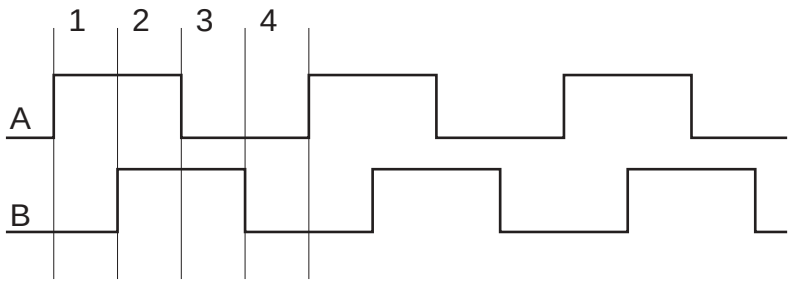


Figure 5.26: Encoder Quadrature Output

The output has two signals, commonly known as channel A and channel B. Some encoders have analog outputs (sine-cosine signals) but the digital type are more widely used. Both channels have a 50% duty cycle and are out of phase by 90°. Using both phases and an appropriate decoder, a motion controller can identify four different areas within one encoder cycle. This type of decoding is called X4 (or quadrature decoding), meaning that the encoder resolution is multiplied by 4. For example, and encoder with 10 μm phase period can offer a 2.5 μm resolution when used with a X4 type decoder.

Physically, and encoder has two parts: a *scale* and a *read head.* The scale is an array of precision placed marks that are read by the head. The most commonly used encoders, optical encoders, have a scale made out of a series of transparent and opaque lines placed on a glass substrate or etched in a thin metal sheet (**Figure 5.27**).

The encoder read head has three major components: a light source, a mask and a detector (**Figure 5.28**). The mask is a small scale-like piece, having identically spaced transparent and opaque lines.

Combining the scale with the read head, when one moves relative to another, the light will pass through where the transparent areas line up or blocked when they do not line up (**Figure 5.29**).

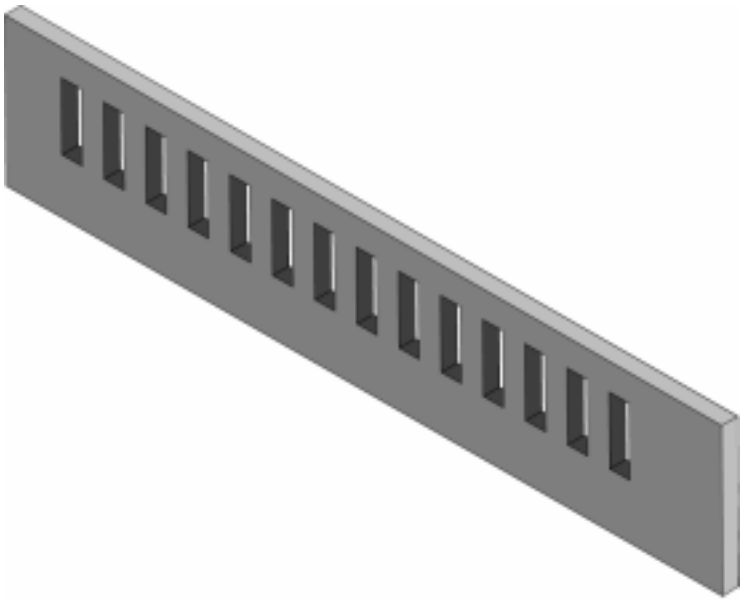


Figure 5.27: Optical Encoder Scale

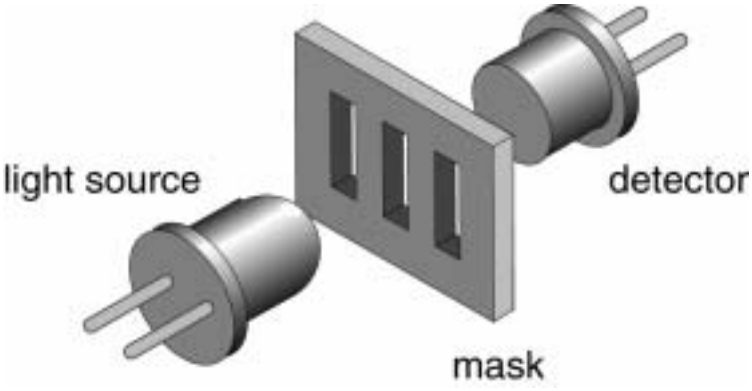


Figure 5.28: Optical Encoder Read Head

The detector signal is similar to a sine wave. Converting it to a digital waveform, the user will get the desired encoder signal. But, this is only one phase, only half of the signal needed to get position information. The second channel is obtained the same way but from a mask that is placed 90% out of phase relative to the first one (**Figure 5.30**).

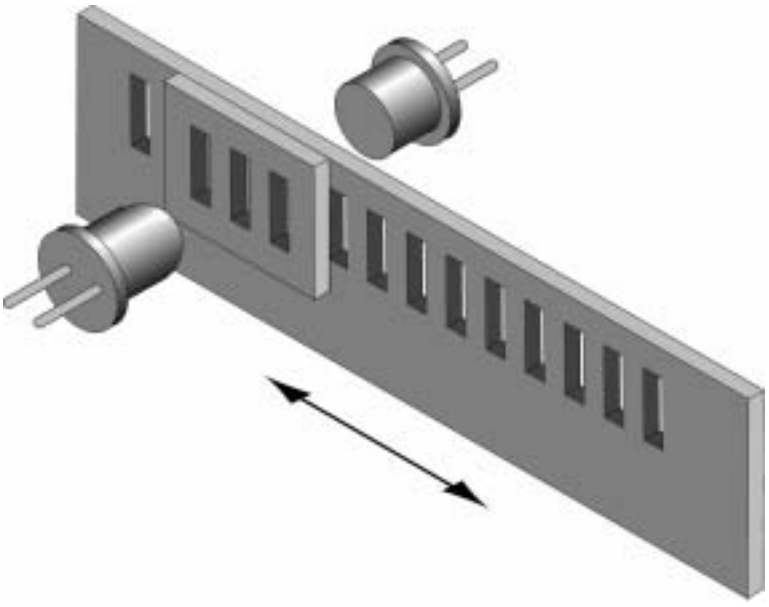


Figure 5.29: Single-Channel Optical Encoder Scale and Read Head Assembly

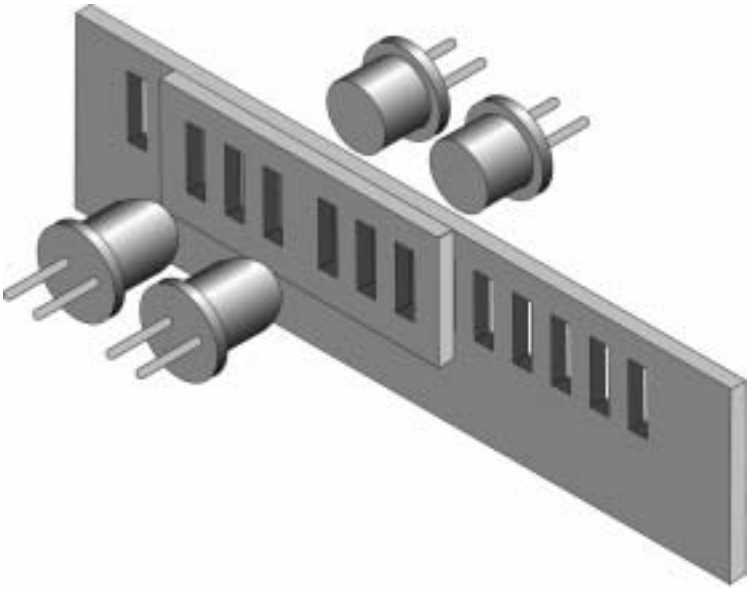


Figure 5.30: Two-Channel Optical Encoder Scale and Read Head Assembly

There are two basic types of encoders: *linear* and *rotary.* The linear encoders, also called linear scales, are used to measure linear motion directly. This means that the physical resolution of the scale will be the actual positioning resolution. This is their main drawback since technological limitations prevent them from having better resolutions than a few microns. To get higher resolutions in linear scales, a special delicate circuitry must be added, called scale interpolator.

Other technologies like interferometry or holography can be used but they are significantly more expansive and need more space.

The most popular encoders are rotary. Using gear reduction between the encoder and the load, significant resolution increases can be obtained at low cost. But the price paid for this added resolution is higher backlash.

In some cases, rotary encoders offer high resolution without the backlash penalty. For instance, a linear translation stage with a rotary encoder on the lead screw can easily achieve 1 μm resolution with negligible backlash.

NOTE

For rotary stages, a rotary encoder measures the output angle directly. In this case, the encoder placed on the rotating platform has the same advantages and disadvantages of the linear scales.

5.6 Motors

There are many different types of electrical motors, each one being best suitable for certain kind of applications. The ESP7000 supports two of the most popular types: *stepper motors* and *Dc motors.*

Another way to characterize motors is by the type of motion they provide. The most common ones are rotary but in some applications, linear motors are preferred.

5.6.1 Stepper Motors

The main characteristic of a stepper motor is that each motion cycle has a number of stable positions. This means that, if current is applied to one of its windings 9called phases), the rotor will try to find one of these stable points and stay here. In order to make a motion, another phase must be energized which, in turn, will find a new stable point, thus making a small incremental move – a step.

Figure 5.31 shows the basics of a stepper motor. When the winding is energized, the magnetic flux will turn the rotor until the rotor and stator teeth line up. This true of the rotor core is made out of soft iron. Regardless of the current polarity, the stator will try to pull-in the closest rotor tooth.

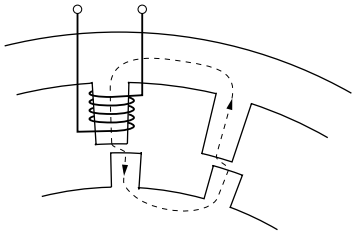


Figure 5.31: Stepper Motor Operation

But, if the rotor is a permanent magnet, depending on the current polarity, the stator will pull or push the rotor tooth. This is a major distinction between two different stepper motor technologies: variable reluctance and permanent magnet motors. The variable reluctance motors are usually small, low cost, large step angle stepper motors. The permanent magnet technology is used for larger, high precision motors.

The stepper motor advances to a new stable position by means of several stator phases that have the teeth slightly offset from each other. To illustrate this, **Figure 5.32** shows a stepper motor with four phases and, to make it easier to follow, it is drawn in a linear fashion (as a linear stepper motor).

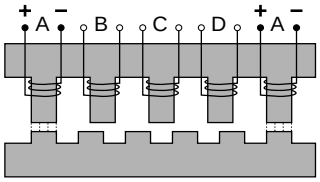


Figure 5.32: Four-Phase Stepper Motor

The four phases, from A to D, are energized one at a time (phase A is shown twice). The rotor teeth line up with the first energized phase, A. If the current to phase A is turned off and B is energized next, the closest rotor tooth to phase B will be pulled in and the motor moves one step forward.

If, on the other hand, the next energized phase is D, the closest rotor tooth is in the opposite direction, thus making the motor to move in reverse.

Phase C cannot be energized immediately after A because it is exactly between two teeth, so the direction of movement is indeterminate.

To move in one direction, the current in the four phases must have the following timing diagram (**Figure 3.33**).

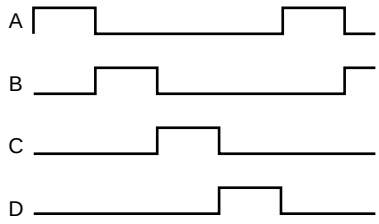


Figure 5.33: Phase Timing Diagram

One phase is energized after another, in a sequence. To advance one full rotor tooth the user needs to make a complete cycle of four steps. To make a full revolution, the user needs a number of steps four times the number of rotor teeth. These steps are called full steps. They are the largest motion increment the stepper motor can make. Running the motor in this mode is called full-stepping.

What happens if the user energizes two neighboring phases simultaneously (**Figure 5.34**).

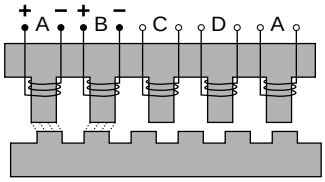


Figure 5.34: Energizing Two Phases Simultaneously

Both phases will pull equally on the motor will move the rotor only half of the full step. If the phases are always energized two at a time, the motor still makes full steps. But, if the user alternates one and two phases being activated simultaneously, the result is that the motor will move only half a step at a time. This method of driving a stepper motor is called half-stepping. The advantage is that we can get double the resolution from the same motor with very little effort on the driver's side. The timing diagram for half-stepping is shown in **Figure 5.35.**

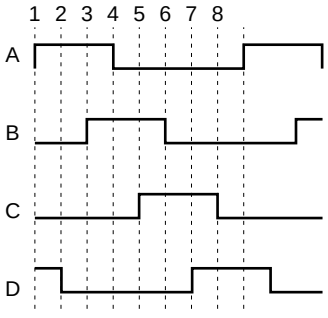


Figure 5.35: Timing Diagram, Half-Stepping Motor

Now, what happens if we energize the same two phases simultaneously but with different currents? For example, lets say that phase A has the full current and phase B only half. This means that phase A will pull the rotor tooth twice as strongly as B does. The rotor tooth will stop closer to A, somewhere between the full step and the half step positions (**Figure 5.36**).

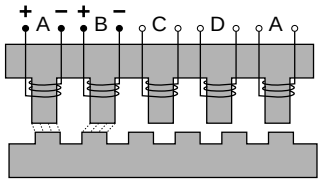


Figure 5.36: Energizing Two Phases with Different Intensities

The conclusion is that, varying the ratio between the currents of the two phases, the user can position the rotor anywhere between the two full step locations. To do so, the user needs to drive the motor with analog signals, similar to **Figure 5.37.**

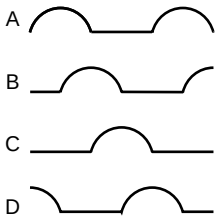


Figure 5.37: Timing Diagram, Continuous Motion (Ideal)

But a stepper motor should be stepping. The controller needs to move it in certain known increments. The solution is to take the half-sine waves and digitize them so that for every step command, the currents change to some new pre-defined levels, causing the motor to advance one small step (**Figure 5.38**).

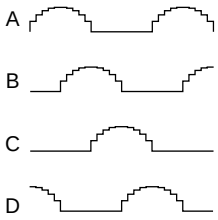


Figure 5.38: Timing Diagram, Mini-Stepping

This driving method is called mini-stepping or micro-stepping. For each step command, the motor will move only a fraction of the full-step. Motion steps are smaller so the motion resolution is increased and the motion ripple (noise) is decreased.

However, mini-stepping comes at a price. First, the driver electronics are significantly more complicated. Secondly, the holding torque or one step is reduced by the mini-stepping factor. In other words, for a x10 mini-stepping, it takes only 1/10 of the full-step holding torque to cause the motor to have a positioning error equivalent to one step (a mini-step).

To clarify a little what this means, lets take a look at the torque produced by a stepper motor. For simplicity, lets consider the case of a single phase being energized (**Figure 5.39**).

Once the closest rotor tooth has been pulled in, assuming that the user doesn't have any external load, the motor does not develop any torque. This is a stable point.

If external forces try to move the rotor (**Figure 5.40**), the magnetic flux will fight back. The more teeth misalignment exists, the larger the generated torque.

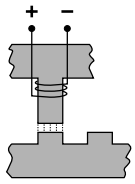


Figure 5.39: Single Phase Energization

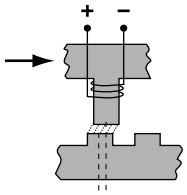


Figure 5.40: External Force Applied

If the misalignment keeps increasing, at some point, the torque peaks and then starts diminishing again such that, when the stator is exactly between the rotor teeth, the torque becomes zero again (**Figure 5.41**).

This is an unstable point and any misalignment or external force will cause the motor to move one way or another. Jumping from one stable point to another is called missing steps, one of the most critiqued characteristics of stepper motors.

The torque diagram versus teeth misalignment is shown in **Figure 5.42.** The maximum torque is obtained at one quarter of the tooth spacing, which is equivalent to one full step.

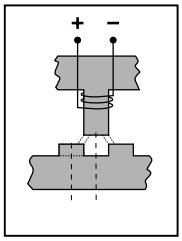


Figure 5.41: Unstable Point

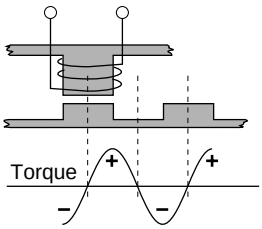


Figure 5.42: Torque and Tooth Alignment

This torque diagram is accurate even when the motor is driven with half-, mini-, or micro-steps. The maximum torque is still one full step away from the stable (desired) position.

Stepper Motor Types

To simplify the explanation, the examples above are based on a variable reluctance stepper motor. The main characteristic of these motors is that their rotors have no permanent magnets. The variable reluctance motors are easy and inexpensive to make but suffer from higher inefficiency and require a unipolar driver. They are used in low cost, low power applications.

Permanent magnet motors have each "tooth" made out of a permanent magnet, each one having alternate polarity. They are more efficient but the step size is very large due to the physical size of the pole "teeth". They are also being used in low cost and, in particular, miniature applications.

The most common type of stepper motor is the Hybrid stepper motor. It has the fine "teeth" and stepping angle of a variable reluctance motor and the efficiency of the permanent magnet motor. The rotor is made out of one or more stacks that consist of a pair of magnetically opposite polarized sections. These motors offer the best combination of efficiency and fine stepping angles and can be driven by both unipolar and bipolar drivers.

Advantages

Stepper motors are primarily intended to be used for low cost microprocessor controlled positioning applications. Due to some of their inherent characteristics, they are preferred in many industrial and laboratory applications. Some of their main advantages are:

- Low cost full-step, open loop implementation
- No servo tuning required
- Good position lock-in
- No encoder necessary
- Easy velocity control
- Retains some holding torque even with power off
- No wearing or arcing commutators
- Preferred for vacuum and explosive environments.

Disadvantages

Some of the main disadvantages of the stepper motors are:

- Could loose steps (synchronization) in open loop operation
- Requires current (dissipates energy) even at stop
- Generates higher heat levels than other types of motors
- Moves from one step to another are made with sudden motions
- Large velocity ripples, especially at low speeds, causing noise and possible resonances
- Load torque must be significantly lower than the motor holding torque to prevent stalling and missing steps
- Limited high speed.

5.6.2 DC Motors

A DC motor is similar to a permanent magnet stepper motor with an added internal phase commutator (**Figure 5.43**).

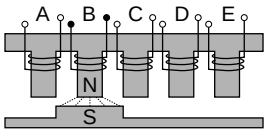


Figure 5.43: DC Motor

Apply current to phase B pulls in the rotor pole. If, as soon as the pole gets there, the current is switched to the next phase C, the rotor will not stop but continue moving to the next target. Repeating the current switching process will keep the motor moving continuously. The only way to stop a DC motor is not to apply any current to its windings. Due to the permanent magnets, reversing the current polarity will cause the motor to move in the opposite direction.

Of course, there is a lot more to the DC motor theory but this description gives the user a general idea on how they work.

A few other characteristics to keep in mind are:

- For a constant load, the velocity is approximately proportional to the voltage applied to the motor
- For accurate positioning, DC motors need a position feed-back device
- Constant current generates approximately constant torque
- If DC motors are turned externally (manually, etc) they act as generators.

Advantages

DC motors are preferred in many applications for the following reasons:

- Smooth, ripple-free motion at any speed
- High torque per volume
- No risk of loosing position (in a closed loop)
- Higher power efficiency than stepper motors
- No current requirement at stop
- High speeds can be obtained than with other types of motors.

Disadvantages

Some of the DC motor's disadvantages are:

- Requires a position feedback encoder and servo loop controller
- Requires servo loop tuning
- Commutator may wear out in time
- Not suitable for high vacuum application due to the commutator arcing
- Hardware and setup are more costly than for an open loop stepper motor (full stepping).

5.7 Drivers

Motor drivers must not be overlooked when judging a motion control system. They represent an important part of the loop that in many cases could increase or reduce the overall performance.

The ESP7000 is an integrated controller and driver. The controller part is common for any configuration but the driver section must have the correct hardware for each motor driven. The driver hardware is one driver card per axis that installs easily in the rear of the controller. Each card has an end-plate with the 25-pin D-Sub motor connector

and an identifying label. Always make sure that the motor specified on the driver card label matches the label on the motion device.

There are important advantages to having an integrated controller/driver. Besides reducing space and cost, integration also offers tighter coordination between the two units so that the controller can more easily monitor and control the driver's operation.

Driver types and techniques vary widely. In the following paragraphs, we will discuss only those implemented in the ESP300.

5.7.1 Stepper Motor Drivers

Driving a stepper motor may look simple at first place. For a motor with four phases, the most widely used type, the user will need only four switches (transistors) controlled directly by a CPU (**Figure 5.44**).

This driver works fine for simple, low performance applications. But, if high speeds are required, having to switch the current fast in inductive loads becomes a problem. When voltage is applied to a winding, the current (and thus the torque) approaches its normal value exponentially (**Figure 5.45**).

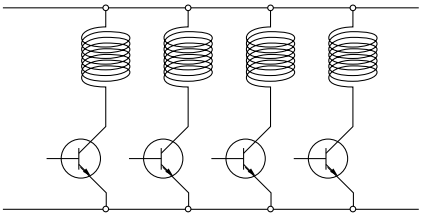


Figure 5.44: Simple Stepper Motor Driver

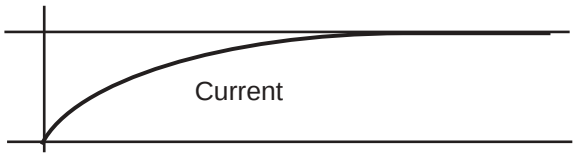


Figure 5.45: Current Build-up in Phase

When the pulse rate is fast, the current does not have time to reach the desired value before it is turned off and the total torque generated is only a fraction of the normal one (**Figure 5.46**).

How fast the current reaches its normal value depends on three factors: the winding's inductance, resistance and the voltage applied to it.

The inductance cannot be reduced. But the voltage can be temporarily increased to bring the current to its desired level faster. The most widely used technique is a high voltage chopper.

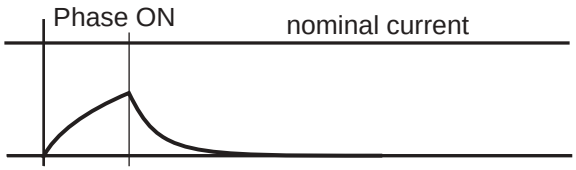


Figure 5.46: Effect of a Short ON Time on Current

If, for instance, a stepper motor requiring only 3V to reach the nominal current is connected momentarily to 30V, it will reach the same current in only 1/10 of the time (**Figure 5.47**).

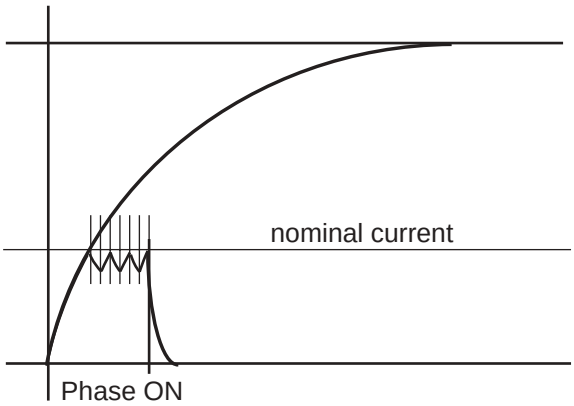


Figure 5.47: Motor Pulse with High Voltage Chopper

Once the desired current value is reached, a chopper circuit activates to keep the current close to the nominal value..

5.7.2 Unipolar – Bipolar Drivers

In the examples described in Section 5.7.1: Stepper Motor Drivers, each phase has its own commutator (transistor) to control the current that flows through it. Having one end permanently connected to the power source, the current will flow through each phase always in the same direction. For this reason, these types of drivers are called **Unipolar.**

On the other hand, **Figure 5.48** shows a **Bipolar** Driver built in a dual H-bridge configuration. The name "H-Bridge" comes from the topology of the transistors controlling one load (coil). In this case, by turning on diagonally transistors (1-4 or 2-3), the current could be made to flow either way through the coil. This means that the driver can control not just the intensity of the magnetic field generated by the stator, but also its polarity. Implicitly, the only stepper motors that can be used with such a driver are the ones with polarized rotors, the Permanent Magnet, and the Hybrid types.

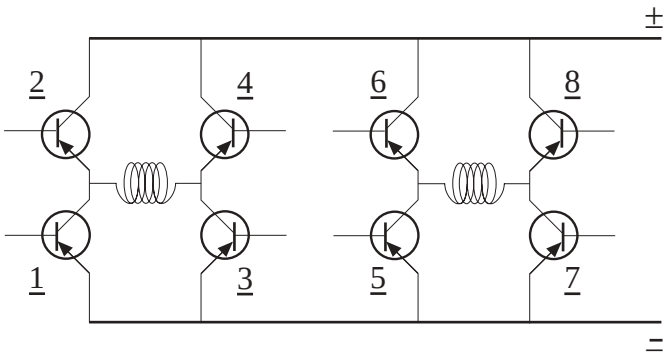


Figure 5.48: Dual H-Bridge Driver

The question that arises from this driver configuration is how to connect a four phase stepper motor to a driver that drives only two coils. This could be accomplished in three different ways, each one with its own advantages and disadvantages:

1. Use only two adjacent phases (e.g., phase #1 and #2).
   - **Advantage** – simplicity
   - **Disadvantage** – lower efficiency since only half the windings are being used.
2. Connect the two opposing phases (1-3 and 2-4) in series.
   - **Advantage** – the motor does not require more than the nominal current.
   - **Disadvantage** – the driver will see twice the nominal motor inductance that will reduce the motor's torque performance at higher speeds.
3. Connect the two opposing phases (1-3 and 2-4) in parallel.
   - **Advantage** – the motor inductance does not increase, allowing it to perform well at higher speeds.
   - **Disadvantages** – requires the driver to supply twice the motor's nominal current.

5.7.3 DC Motor Drivers

There are three major categories of DC motor drivers. The simplest one is a voltage amplifier (**Figure 5.49**).

The driver amplifies the standard ±10 V control signal to cover the motor's nominal voltage range while also supplying the motor's nominal current.

This type of driver is used mostly in low cost applications where following error is not a great concern. The controller does all the work in trying to minimize the following error but load variations make this task very difficult.

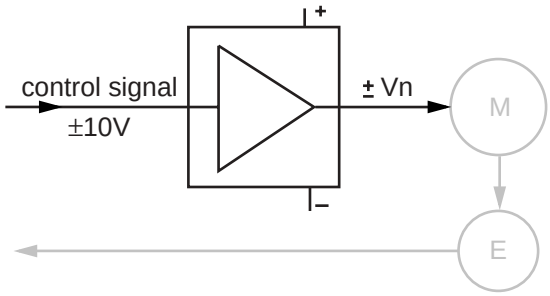


Figure 5.49: DC Motor Voltage Amplifier

The second type of DC motor driver is the current driver, also called a torque driver (**Figure 5.50**).

In this case, the control signal voltage defines the motor current. The driver constantly measures the motor current and always keeps it proportional to the input voltage. This type of driver is usually preferred over the previous one in digital control loops, offering a stiffer response and thus reduces the dynamic following error.

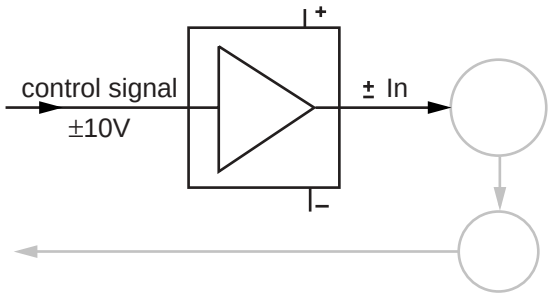


Figure 5.50: DC Motor Current Driver

But, when the highest possible performance is required, the best choice is always the velocity feedback driver. This type of driver requires a tachometer, an expensive and sometimes difficult to add device (**Figure 5.51**).

The tachometer, connected to the motor's rotor, outputs a voltage directly proportional with the motor velocity. The circuit compares this voltage with the control signal and drives the motor so that the two are always equal. This creates a second closed loop, a velocity loop. Motions performed with such a driver are very smooth at high and low speeds and have a similar dynamic following error.

General purpose velocity feedback drivers have usually two adjustments: tachometer gain and compensation (**Figure 5.52**).

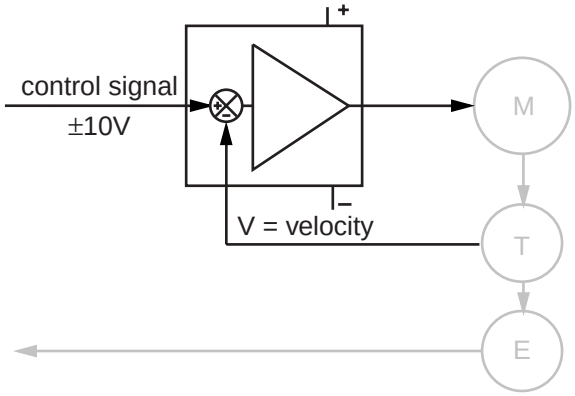


Figure 5.51: DC Motor Velocity Feedback Driver

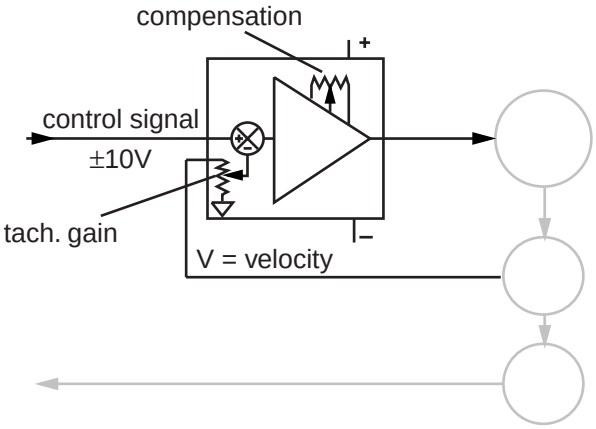


Figure 5.52: DC Motor Tachometer Gain and Compensation

The tachometer gain is used to set the ratio between the control voltage and the velocity. The compensation adjustment reduces the bandwidth of the amplifier to avoid oscillations of the closed loop.

PWM Drivers

Even though linear amplifiers are similar and cleaner (do not generate noise), their low efficiency makes them impractical to be used with medium and larger motors. The most common types of DC drivers use some kind of PWM (**P**ulse-**W**idth **M**odulation) techniques to control the current and/or voltage applied to the motor. This allows for a more efficient and compact driver design.

Section 6 – Servo Tuning

6.1 Tuning Principles

The ESP controller uses a PID servo loop with feed-forward. Servo tuning set the **Kp, Ki,** and **Kd,** and feed-forward parameters of the digital PID algorithm, also called the PID filter.

Tuning PID parameters requires a reasonable amount of closed-loop system understanding. First review the Control Loops paragraph in the Motion Control Tutorial Section. If needed, consult additional servo control theory books.

Start the tuning process using the default values supplied with the stage. These values are usually very conservative, favoring safe, oscillation-free operation. To achieve the best dynamic performance possible, the system must be tuned for the specific application. Load, acceleration, stage orientation, and performance requirements all affect how the servo loop should be tuned.

6.2 Tuning Procedures

Servo tuning is usually performed to achieve better motion performance (such as reducing the following error statically and/or dynamically) or because the system is malfunctioning (oscillating and/or shutting off due to excessive following error).

Acceleration plays a significant role in the magnitudes of the following error and overshoot, especially at start and stop. Rapid velocity changes represent very high acceleration, causing large following errors and overshoot. Use the smallest acceleration the application can tolerate to reduce overshoot and make tuning the PID filter easier.

NOTE

In the following descriptions, it is assumed that a software utility is being used to capture the response of the servo loop during a motion step command, and to visualize the results.

6.2.1 Hardware And Software Requirements

Hardware Requirements

Tuning is best accomplished when the system response can be measured. This can be done with external monitoring devices but can introduce errors.

The ESP controller avoids this problem by preventing an internal *trace* capability. When *trace* mode is activated, the controller records a number of different parameters. The parameters can include real instantaneous position, desired position, desired velocity, desired acceleration, DAC output value, etc.

The sample interval can be set to one servo cycle or any multiple of it and the total number of samples can be up to 1000.

This is a powerful feature that the user can take advantage of, to get maximum performance out of the motion system.

Software Requirements

Users can write their own application(s) or use the ESP-tune-exe Windows utility.

6.2.2 Correcting Axis Oscillation

There are three parameters that can cause oscillation. The most likely to induce oscillation is **Ki,** followed by **Kp** and **Kd.** Start by setting **Ki** to zero and reducing **Kp** and **Kd** by 50%.

If oscillation does not stop, reduce **Kp** again.

When the axis stops oscillating, system response is probably very soft. The following error may be quite large during motion and non-zero at stop. Continue tuning the PID with the procedures described in the next paragraph.

6.2.3 Correcting Following Error

If the system is stable and the user wants to improve performance, start with the current PID parameters. The goal is to reduce following error during motion and to eliminate it at stop.

Guidelines for further tuning (based on performance starting point and desired outcome) are provided in the following paragraphs.

Following Error Too Large

This is the case of a soft PID loop caused by low values for **Kp** and **Kd.** It is especially common after performing the procedures described in paragraph 6.2.2.

First increase **Kp** by a factor of 1.5 to 2. Repeat this operation while monitoring the following error until it starts to exhibit excessive ringing characteristics (more than 3 cycles after stop). To reduce ringing, add some damping by increasing the **Kd** parameter.

Increase it by a factor of 2 while monitoring the following error. As **Kd** is increased, overshoot and ringing will decrease almost to zero.

NOTE

Remember that if acceleration is set too high, overshoot cannot be completely eliminated with Kd.

If **Kd** is further increased, at some point oscillation will reappear, usually at a higher frequency. Avoid this by keeping **Kd** at a high enough value, but not so high as to re-introduce oscillation.

Increase **Kp** successively by approximately 20% until signs of excessive ringing appear again.

Alternately increase **Kd** and **Kp** until **Kd** cannot eliminate overshoot and ringing at stop. This indicates **Kp** is larger than its optional value and should be reduced. At this point, the PID loop is very tight.

Ultimately, optimal values for **Kp** and **Kd** depend on the stiffness of the loop and how much ringing the application can tolerate.

NOTE

The tighter the loop, the greater the risk of instability and oscillation when load conditions change.

Errors At Stop (Not In Position)

If you are satisfied with the dynamic response of the PID loop but the stage does not always stop accurately, modify the integral gain factor **Ki.** As described in the Motion Control Tutorial section, the **Ki** factor of the PID works to reduce following error to near zero. Unfortunately it can also contribute to oscillation and overshoot. Change this parameter carefully, and if possible, in conjunction with **Kd.**

Start with the integral limit (**II**) set to a high value and **Ki** value at least two orders of magnitude smaller than **Kp.** Increase its value by 50% at a time and monitor overshoot and final position at stop.

If intolerable overshoot develops, increase the **Kd** factor. Continue increasing **Ki** and **Kd** alternatively until an acceptable loop response is obtained. If oscillation develops, immediately reduce **Ki.**

Remember that any finite value for **Ki** will eventually reduce the error at stop. It is simply a matter of how much time is acceptable for the application. In most cases it is preferable to wait a few extra milliseconds to get to the stop in position rather than have overshoot or run the risk of oscillations.

Following Error During Motion

This is caused by a **Ki** value that is too low. Follow the procedures in the previous paragraph, keeping in mind that it is desirable to increase the integral gain factor as little as possible.

6.2.4 Points To Remember

- Use the Windows-based "ESP_tune.exe" utility to change PID parameters and to visualize the effect. Compare the results and parameters used with the previous iteration.
- The ESP controller uses a servo loop based on the PID with velocity and acceleration feed-forward algorithm.
- Use the lowest acceleration the application can tolerate. Lower acceleration generates less overshoot.
- Use the default values provided with the system for all standard motion devices as a starting point.
- Use the minimum value for **Ki** that gives acceptable performance. The integral gain factor can cause overshoot and oscillations.

A summary of servo parameter functions is listed in **Table 6.1.**

Parameter	Function	Value Set Too Low	Value Set Too High
Kp	Determines stiffness of servo loop	Servo loop too soft with high following errors	Servo loop too light and/or causing oscillation
Kd	Main damping factor, used to eliminate oscillation	Uncompensated oscillation caused by other parameters being high	Higher-frequency oscillation and/or audible noise in the motor caused by large ripple in the motor voltage
Ki	Reduces following error during long motions and at stop	Stage does not reach or stay at the desired stop position	Oscillations at lower frequency and higher amplitude
Vff	Reduces following error during the constant velocity phase of a motion	Negative following error during the constant velocity phase of a motion. Stage lags the desired trajectory.	Positive following error during the constant velocity phase of a motion. Stage is ahead of the desired trajectory.
Aff	Reduces following error during the acceleration and deceleration phases of a motion	Negative following error during the acceleration phase of a motion. Stage lags the desired trajectory.	Positive following error during the acceleration phase of a motion. Stage is ahead of the desired trajectory.

Table 6.1: Servo Parameter Functions

Section 7 – Optional Equipment

7.1 Hand-held Keypad

An optional alphanumeric keypad (see **Figure 7.1** below) allows the user to access the full command set of the ESP7000 without the use of a host computer. The keypad features a backlit LCD display that echoes each character typed on the keypad. Additionally, status messages are echoed to the display in certain cases (e.g., error codes).

Four macro keys on the top row of the keypad permit execution of previously stored programs on the push of one button (see **EP, EX** commands in Section 3, Remote Mode, for details).

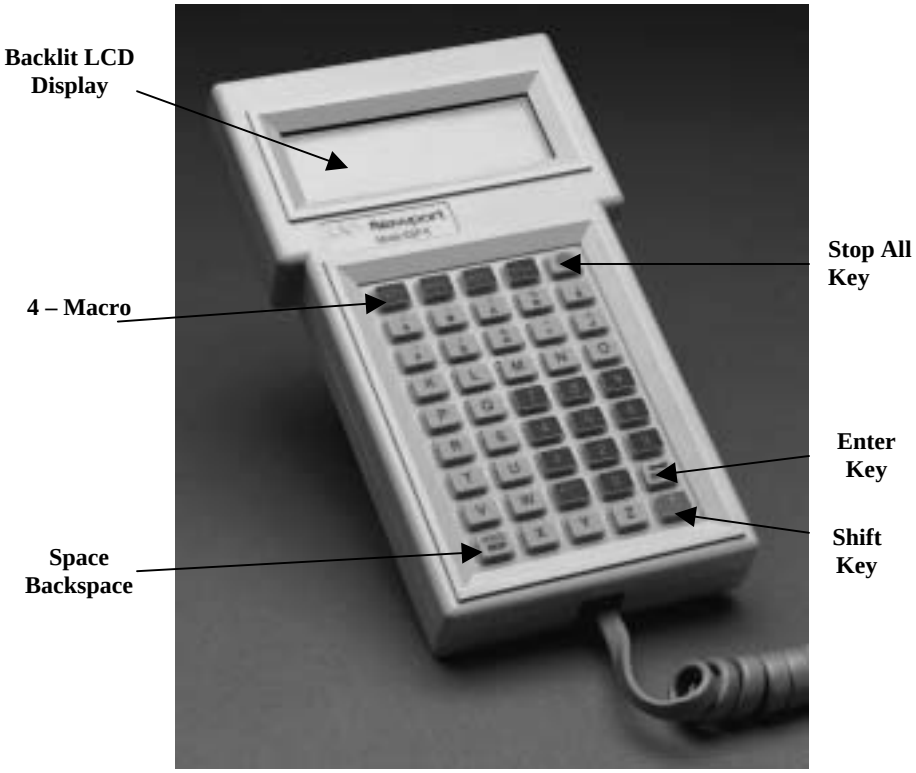


Figure 7.1: Hand-held Keypad Image

7.1.1 Description of Keys (see **Figure 7.1**)

STOP ALL

When this button is activated, all motion is aborted. All axes are affected. The function of STOP ALL on the keypad is equivalent to STOP ALL on the ESP7000 front panel.

MACRO1, MACRO2, MACRO3, MACRO4

Activating either MACRI # button results in execution of a previously stored program. See **EP** command in Section 3, Remote Mode, for details on creating programs.

SHIFT↑

Pressing this key in connection with a double function key selects the upper symbol, e.g. ***/F*** (this symbol is selected).

↵ ENTER

When this key is pressed, all previously typed characters are sent to the ESP7000.

SPACE BKSP

Pressing this key deletes a preceding character. Pressing it in connection with the SHIFT key inserts a space in a given line.

7.1.2 Activating the Keypad

Plug one end of the cable that was supplied with the keypad into the receptacle on the bottom side of the keypad and the other end into the receptacle on the lower right corner of the ESP7000 front panel. The ESP7000 will detect the presence of the keypad. No user settings are required.

Now, the keypad is ready for use. An easy way to verify proper operation of the keypad is by sending the **VE** command (type: **VE** and hit ENTER). The ESP7000 should respond with the current version.

7.2 Rack Mount Brackets

This option provides a means of mounting the ESP7000 controller in a 19-inch rack. There is no disassembly required for this installation. The brackets are attached by putting supplied screws through exterior screw-holes and into permanent nuts inside the enclosure. Installation is shown in **Figure 7.2.**

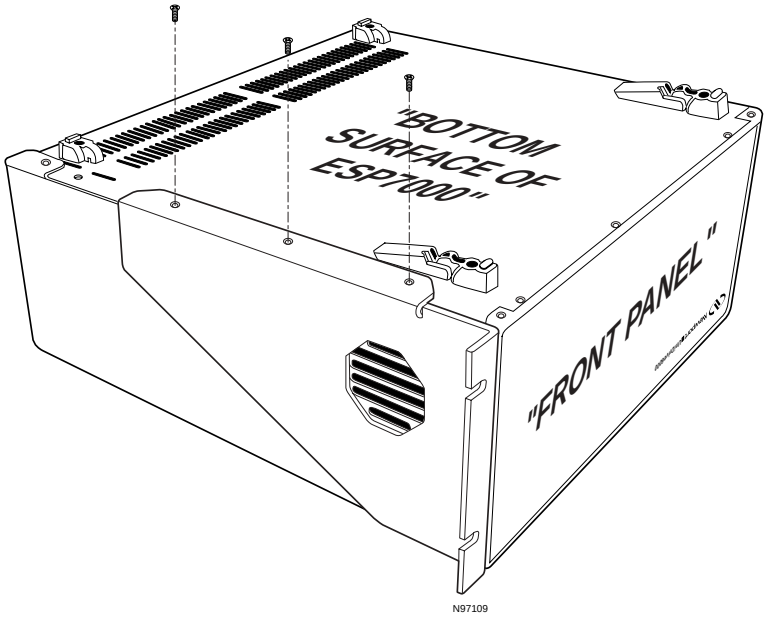


Figure 7.2: Rack-Mount Bracket Installation

7.3 Motor Driver Card

The ESP7000 controller card can be upgraded to operate with up to six stages by installing a separate driver card for each additional stage.

Procedures for adding a driver card are provided in the following paragraphs.

WARNING

Power off all equipment and unplug AC power cord(s) before installing any equipment.

CAUTION

The ESP7000 controller and driver cards are sensitive to static electricity. Wear a properly grounded anti-static strap when handling equipment.

Shut-down all stage operations.

Power down all equipment (refer to the Controls and indicators paragraph in System Setup) and unplug AC power cord(s).

Loosen the upper and lower thumbscrews on one of the blank cut-out panels at the rear of the ESP7000 controller, and remove the panel from the slot.

Carefully remove the driver card to be installed from its packaging.

Inspect the driver card for loose components or other problems (see **Figure 7.3**). Refer to Appendix H, Factory Service, to report discrepancies.

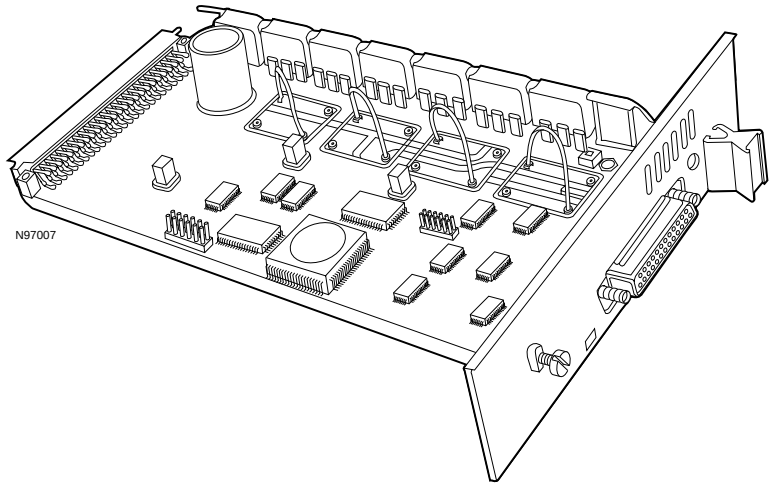


Figure 7.3: Driver Card

Insert the driver card into the black guides (top and bottom) in the slot (see **Figure 7.4**).

Figure 7.4: Driver Card Installation

Push gently until the edge connector at the back of the card mates with the motherboard chassis connector.

Tighten the thumbscrews on the driver card.

Connect the stage to the newly installed axis. Plug in the ESP7000 power cord and power up the controller.

Appendix A – Error Messages

The ESP controller has an elaborate command interpreter and system monitor. Every command is analyzed for syntax and correct format after it is received. The result of the analysis is stored in an output buffer in plain English. System inputs are monitored while motion is in progress and holding position, and any change is reported to the user via the output buffer. To read the contents of the output buffer, send command **TB** (tell buffer).

For more compact error messages, use the **TE** command. The ESP controller response to this command is a one byte; binary coded error number, e.g., 33.

For the sake of convenience, error messages are divided into two categories – non-axis specific error messages and axis specific error messages. Below is a list of all possible ESP controller error messages that are not axis specific:

0 NO ERROR DETECTED
No errors exist in the output buffer.

1 PCI COMMUNICATION TIME-OUT
A communication transfer was initiated through PCI bus interface and was never completed.

2 Reserved for future use

3 Reserved for future use

4 EMERGENCY STOP ACTIVATED
An emergency stop was executed because the motion controller received a '#' character or "STOP ALL AXES" button was pressed.

5 Reserved for future use

6 COMMAND DOES NOT EXIST
The issued command does not exist. Check the ASCII mnemonic.

7 PARAMETER OUT OF RANGE
The specified parameter is out of range. Refer to the description of issued command for valid parameter range.

8 CABLE INTERLOCK ERROR
The 100-pin cable between motion controller board and driver is disconnected.

9 AXIS NUMBER OUT OF RANGE
The specified axis number is out of range. Refer to the description of issued command for valid axis number range.

10 Reserved for future use

11 Reserved for future use

12 Reserved for future use

13 GROUP NUMBER MISSING
Group number is not specified. The issued command requires a valid group number. Refer to the description of issued command for valid group number range.

14 GROUP NUMBER OUT OF RANGE
The specified group number is out of range. Refer to the description of issued command for valid group number range.

15 GROUP NUMBER NOT ASSIGNED
The specified group has not been assigned. Refer to the description of **HN** command to create a new group with this group number.

16 GROUP NUMBER ALREADY ASSIGNED
The specified group number has already been assigned. Refer to the description of **HB** command to query the list of group numbers already assigned.

17 GROUP AXIS OUT OF RANGE
At least one of the axis numbers specified to be a member of this group is out of range. Refer to the description of **HN** command for valid range of axis numbers that can be assigned to a group.

18 GROUP AXIS ALREADY ASSIGNED
At least one of the axis numbers specified to be a member of this group is already a member of a different group.

19 GROUP AXIS DUPLICATED
At least one of the axis numbers is specified to be a member of this group more than once.

20 DATA ACQUISITION IS BUSY
Data acquisition is not yet complete.

21 DATA ACQUISITION SETUP ERROR
An error occurred during data acquisition setup. Ensure that data acquisition is disabled and all parameters are within valid range before issuing the command. Refer to the command description for valid range of parameters.

22 DATA ACQUISITION NOT ENABLED
Data acquisition is not yet enabled.

23 SERVO CYCLE TICK FAILURE
There was a failure to increment the servo tick in the Interrupt Service Routine (ISR) that manages motion control.

24 Reserved for future use

25 DOWNLOAD IN PROGRESS
Firmware download is in progress.

26 STORED PROGRAM NOT STARTED
An attempt was made to execute a stored program and the program could not be started.

27 COMMAND NOT ALLOWED
The issued command is not valid in the context in which it was issued.

28 STORED PROGRAM FLASH AREA FULL
The flash area reserved for stored programs is full.

29 GROUP PARAMETER MISSING
At least one parameter is missing. Refer to the description of issued command for valid number of parameters.

30 GROUP PARAMETER OUT OF RANGE
The specified group parameter is out of range. Refer to the description of issued command for valid range of parameter.

31 GROUP MAXIMUM VELOCITY EXCEEDED
The specified group velocity exceeds the minimum of the maximum velocities of members of this group. Refer to the description of **HV** command for more details.

32 GROUP MAXIMUM ACCELERATION EXCEEDED
The specified group acceleration exceeds the minimum of the maximum acceleration of members of this group. Refer to the description of **HA** command for more details.

33 GROUP MAXIMUM DECELERATION EXCEEDED
The specified group deceleration exceeds the minimum of the maximum decelerations of members of this group. Refer to the description of **HD** command for more details.

34 GROUP MOVE NOT ALLOWED DURING HOMING
Cannot make a coordinated move when one of the members of the group is being "homed".

35 PROGRAM NOT FOUND
The issued command could not be executed because the stored program requested is not available.

36 Reserved for future use

37 AXIS NUMBER MISSING
Axis number not specified. The issued command requires a valid axis number. Refer to the description of issued command for valid axis number range.

38 COMMAND PARAMETER MISSING
At least one parameter associated with this command is missing. Refer to the description of issued command for valid number of parameters.

39 PROGRAM LABEL NOT FOUND
The issued command could not be executed because the requested label within a stored program is not available.

40 LAST COMMAND CANNOT BE REPEATED
An attempt was made to repeat the last (previous) commanded by just sending a carriage return. This feature is not allowed for commands that carry strings in addition to the two-letter ASCII mnemonic. Issue the last command again.

41 MAX NUMBER OF LABELS PER PROGRAM EXCEEDED
The number of labels used in the stored program exceeds the allowed value.

Below is a list of all possible error messages that are axis specific. Here, *"x"* represents the axis number.

x00 MOTOR TYPE NOT DEFINED
A valid motor type was not defined for the requested axis. Refer to the description of **QM** command to define a motor type.

x01 PARAMETER OUT OF RANGE
The specified parameter is out of range. Refer to the description of issued command for valid parameter range.

x02 AMPLIFIER FAULT DETECTED
There was an amplifier fault condition.

x03 FOLLOWING ERROR THRESHOLD EXCEEDED
The real position of specified axis was lagging the desired position by more encoder counts than specified with the **FE** command. Refer to the description of **ZF** command to configure the motion controller tasks upon encountering a following error.

x04 POSITIVE HARDWARE LIMIT DETECTED
The motion controller sensed a high level at its positive travel limit input. Refer to the description of **ZH** command to configure the motion controller tasks upon encountering a hardware limit.

x05 NEGATIVE HARDWARE LIMIT DETECTED
The motion controller sensed a high level at its negative travel limit input. Refer to the description of **ZH** command to configure the motion controller tasks upon encountering a hardware limit.

x06 POSITIVE SOFTWARE LIMIT DETECTED
The motion controller sensed that the axis has reached positive software travel limit. Refer to the description of **SR** command to specify the desired positive software travel limit. Also, refer to the description of **ZS** command to configure the motion controller tasks upon encountering a software limit.

x07 NEGATIVE SOFTWARE LIMIT DETECTED
The motion controller sensed that the axis has reached negative software travel limit. Refer to the description of **SL** command to specify the desired negative software travel limit. Also, refer to the description of **ZS** command to configure the motion controller tasks upon encountering a software limit.

x08 MOTOR/STAGE NOT CONNECTED
The specified axis is not connected to the driver.

x09 FEEDBACK SIGNAL FAULT DETECTED
There was a feedback signal fault condition. Ensure that the encoder feedback is relatively noise free.

x10 MAXIMUM VELOCITY EXCEEDED
The specified axis velocity exceeds maximum velocity allowed for the axis. Refer to the description of **VU** command or set maximum velocity for the axis.

x11 MAXIMUM ACCELERATION EXCEEDED
The specified axis acceleration exceeds maximum acceleration allowed for the axis. Refer to the description of **AU** command to query or set maximum acceleration or deceleration for the axis.

x12 Reserved for future use

x13 MOTOR NOT ENABLED
A command was issued to move an axis that was not powered ON. Refer to the description of **MO** and **MF** commands to turn the power to an axis ON or OFF respectively.

x14 Reserved for future use

x15 MAXIMUM JERK EXCEEDED
The specified axis jerk exceeds maximum jerk allowed for the axis. Refer to the description of **JK** command for valid jerk range.

x16 MAXIMUM DAC OFFSET EXCEEDED
The specified axis DAC offset exceeds maximum value allowed for the axis. Refer to the description of issued command for valid range.

x17 ESP CRITICAL SETTINGS ARE PROTECTED
An attempt was made to modify parameters that are specific to smart stages or "Unidriver".

x18 ESP STAGE DEVICE ERROR
An error occurred while reading a smart stage.

x19 ESP STAGE DATA INVALID
Smart stage data is invalid.

x20 HOMING ABORTED
Axis home search was aborted. This message is obtained when home search was not completed either due to an axis not being enabled or due to the occurrence of a fault condition. Refer to the description of **OR** command for information related to locating the home position of an axis.

x21 MOTOR CURRENT NOT DEFINED
Maximum current for the motor is not specified. Refer to the description of **QI** command to query or set the maximum motor current for an axis.

x22 UNIDRIVE COMMUNICATIONS ERROR
There was no communication between motion controller and the Unidriver.

x23 UNIDRIVE NOT DETECTED
Unidrive could not be detected by the motion controller.

x24 SPEED OUT OF RANGE
The specified home search speed is out of range. Refer to the description of **OH** command for valid home search speed range.

x25 INVALID TRAJECTORY MASTER AXIS
The specified trajectory mode is not valid for a master axis. Refer to the description of **TJ** command to specify a valid trajectory mode for a master axis.

x26 PARAMETER CHANGE NOT ALLOWED
The specified parameter cannot be changed while the axis is in motion. Wait until the axis motion is complete, and issue this command again. Refer to the description of **MD** command to determine if motion is done.

x27 INVALID TRAJECTORY MODE FOR HOMING
The specified trajectory mode is not valid for locating the home position of the axis. Refer to the description of **TJ** command to specify a valid trajectory mode for locating the home position of this axis.

x28 INVALID ENCODER STEP RATIO

The specified full step resolution is invalid. Refer to the description of **FR** command for valid range of full step resolution.

x29 DIGITAL I/O INTERLOCK DETECTED

A DIO interlock was asserted.

x30 COMMAND NOT ALLOWED DURING HOMING

The command issued was not executed because locating the home position of this axis is in progress. Refer to the description of the issued command for further details.

x31 COMMAND NOT ALLOWED DUE TO GROUP ASSIGNMENT

The specified command was not executed because this axis is member of a group. Refer to the description of issued command for further details.

x32 INVALID TRAJECTORY MODE FOR MOVING

The specified trajectory mode is invalid to make absolute or relative moves. Refer to the description of **PA** and **PR** commands for valid trajectory modes to initiate motion.

x33 CLOSED-LOOP ATTEMPTS EXCEEDED

The axis was unable to reach a desired position even after a fixed number of closed-loop attempts. Refer to the description of **CL** command for further description.

Appendix B – Trouble-shooting and Maintenance

There are no user-serviceable parts or user adjustments to be made to the ESP7000 Stand Alone Motion Controller.

WARNING

Procedures are to be performed only by qualified service personnel. Qualified service personnel should be aware of the shock hazards involved when instrument covers are removed and should observe the following precautions before proceeding.

- **Turn off power switch and unplug the unit from its power source**
- **Disconnect cables if their function is not understood**
- **Remove jewelry from hands and wrist**
- **Expect hazardous voltages to be present in any unknown circuits.**

WARNING

Fuse replacement involves removing an enclosure panel that can expose you to terminals having hazardous voltages in excess of 250 VAC at various locations.

CAUTION

Verify proper alignment before inserting cables into connectors. Do not force.

Refer to Appendix G, Factory Service, for information about repair or other hardware corrective action.

B.1 Trouble-Shooting Guide Information

Most of the time a blown fuse is the result of a more serious problem. Fixing the problem should include not only correcting the effect (blown fuse) but also the cause of the failure. Analyze the problem carefully to avoid repeating it in the future.

A list of the most common problems and their corrective actions is provided in Table B.1. Use it as a reference but remember that a perceived error is usually an operator error or has some other simple solution.

PROBLEM	CAUSE	CORRECTIVE ACTION
Display LCDs do not come on.	Power switch is turned off.	Turn on the main power switch located on the front panel.
Power LED does not illuminate green when power on button is pressed.	No electrical power.	Verify with an adequate tester or another electrical device (lamp, etc.) that power is present in the outlet. If not, contact an electrician to correct the problem.
	Power cord not plugged in.	Plug the power cord in the appropriate outlet. Observe all caution notes and procedures described in the System Setup section.
	Blown fuse.	Replace the line fuse as described in the System Setup section. Beware that the fuse blows only when a serious problem arises. If the fuse blows again, contact Newport for service.
Error message or physically present stage is declared unconnected.	Bad connection.	Turn power off and verify the motion device cable connection.
A connected axis is declared unconnected.	Bad component.	Turn power off and swap the motor cable with another axis (if cables are identical) to locate the problem. Contact Newport for cable replacement or motion device service.

Table B.1: Trouble-Shoot Guide

PROBLEM	CAUSE	CORRECTIVE ACTION
Safety control connector is not connected	Safety control connector on the rear of the ESP7000 is missing.	Plug connector in. If the connector was lost, you can either build one as shown in System Setup in Section 1, or call Newport for a replacement.
Motor can not be turned on.	Motor type is not defined.	Enter correct stage parameters, including motor type.
Excessive following error	- -	Verify that all setup parameters correspond to the actual motion device installed.
		Verify that the load specifications for the motion device are not being exceeded.
Axis does not move.	Incorrect connection.	Verify that the motion device is connected to the correct diver card, as specified by the labels.
	Incorrect parameters.	Verify that all relevant parameters (PID, velocity, etc.) are set properly.
System performance below.	Incorrect connection.	Verify that the motion device is connected to the correct driver card, as specified by the labels.
	Incorrect parameters.	Verify that all relevant parameters (PID, velocity, etc.) are set properly.
Motor excessively hot.	Incorrect connection.	Verify that the motion device is connected to the correct driver card, as specified by the labels.
Move command not executed.	Software travel limit.	The software limit (See **SL** command) if the specified direction was reached. If limits are set correctly, do not try to move past them.
	Incorrect parameters.	Verify that all relevant parameters (PID, velocity, etc.) are set properly.

Table B.1: Trouble-Shoot Guide (Continued)

PROBLEM	CAUSE	CORRECTIVE ACTION
Home search not completed.	Faulty origin or index signals.	Carefully observe and record the motion sequence by watching manual knob rotation, if available. With the information collected, call Newport for assistance.
	Wrong line terminator.	Make sure that the computer and the controller use the same line terminator.
	No remote communication, wrong communication port.	Verify that the controller is set to communicate on the right port, RS-232, IEEE488 or VSB.
	Wrong communication parameters.	Verify that all communication parameters match between the computer and the controller.

Table B.1: Trouble-Shoot Guide (Continued)

NOTE: Many problems are detected by the controller and reported on the display and/or in the error register. Consult Appendix A, Error Messages, and for a complete list and description.

B.2 Fuse Replacement

WARNING

Power-down equipment and unplug AC power cord before replacing fuses.

At the rear of the ESP7000, pop out the fuse holder cover plate with a small, thin-blade screwdriver (see **Figure B.1**) and ease the fuse holder out of the AC plug receptacle.

NOTE: The fuse holder has a dual function. It is also a voltage selector. The voltage selection function is not being used in the ESP7000. The power supply has universal input and can work with 115VAC/230VAC. The Voltage Selector/Fuse Holder can be plugged in either way (the arrow pointing toward 120V or 220V)!

After easing out the fuse holder, remove and inspect the (2) fuses. Replace as needed with 10A, T, 250V fuses.

Re-insert into the AC plug receptacle by pushing in the fuse holder, until the cover plate is flush with the outside surface of the Power Entry Module.

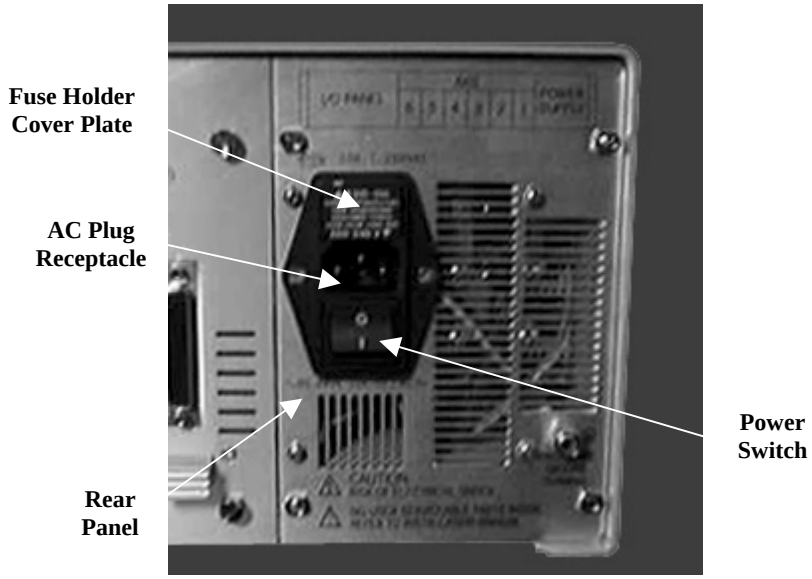


Figure B.1: Rear Power Line Panel Fuse Replacement

Re-connect and power up the system to verify that the problem has been corrected.

B.3 Cleaning

Clean the exterior metallic surfaces of the ESP7000 with water and a clean, lint-free cloth. Clean external cable surfaces with alcohol, using a clean, lint free cloth.

WARNING

Power-down all equipment before cleaning.

CAUTION

Do not expose connectors, fans, LEDs, or switches to alcohol or water.

Appendix C – Connector Pin Assignments

C.1 ESP7000 Rear Panel

C.1.1 Digital I/O Connector (50-Pin D-Sub)

This connector provides access to the ESP7000 Opto22 compatible 24-bit digital I/O interfaces. Connector pin-outs are listed in **Table C.1,** and functionally described in **Section C.1.2.**

C.1.2 Signal Descriptions (Digital I/O, 50-Pin Connector) +5V, 100mA (maximum)

+5V supply available from the PC.

Digital I/O

The digital I/O can be programmed to be either input or output (in 8-bit blocks) via software and is pulled-up to +5 volts with a 4.7KΩ resistor. When configured as output, each bit can source 64mA (maximum). When configured as input, each bit can sink 32mA (maximum).

Ground

Ground reference used for all digital signals.

C.1.3 Motor Driver Card (25-Pin) I/O Connector

This connector interfaces an ESP7000 driver card to motorized stages. Cabling to the connector is provided with the applicable stage. Connector pin-outs are listed in **Table C.2,** and are functionally described in **Section C.1.4.**

C.1.4 Signal Descriptions (Motor Driver Card, 25-Pin I/O Connector)

DC Motor Phase(+) Output

This output must be connected to the positive lead of the DC motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48V DC.

Pin #	Description
1	Port C, Bit-7
2	Ground
3	Port C, Bit-6
4	Ground
5	Port C, Bit-5
6	Ground
7	Port C, Bit-4
8	Ground
9	Port C, Bit-3
10	Ground
11	Port C, Bit-2
12	Ground
13	Port C, Bit-1
14	Ground
15	Port C, Bit-0
16	Ground
17	Port B, Bit-7
18	Ground
19	Port B, Bit-6
20	Ground
21	Port B, Bit-5
22	Ground
23	Port B, Bit-4
24	Ground
25	Port B, Bit-3
26	Ground
27	Port B, Bit-2
28	Ground
29	Port B, Bit-1
30	Ground
31	Port B, Bit-0
32	Ground
33	Port A, Bit-7
34	Ground
35	Port A, Bit-6
36	Ground
37	Port A, Bit-5
38	Ground
39	Port A, Bit-4
40	Ground
41	Port A, Bit-3
42	Ground
43	Port A, Bit-2
44	Ground
45	Port A, bit-1
46	Ground
47	Port A, Bit-0
48	Ground
49	+5V, 250mA (maximum)
50	Ground

Table C.1: Digital Connector Pin-Outs

Pins	Stepper Motor	DC Motor
1	Stepper Phase 1	Tacho Generator (+)
2	Stepper Phase 1	Tacho Generator (+)
3	Stepper Phase 2	Tacho Generator (-)
4	Stepper Phase 2	Tacho Generator (-)
5	Stepper Phase 3	DC Motor Phase (+)
6	Stepper Phase 3	DC Motor Phase (+)
7	Stepper Phase 4	DC Motor Phase (-)
8	Stepper Phase 4	DC Motor Phase (-)
9	Not Connected	Not Connected
10	Not Connected	Not Connected
11	Not Connected	Not Connected
12	Not Connected	Not Connected
13	Home Signal	Home Signal
14	Shield Ground	Shield Ground
15	Encoder Index (+)	Encoder Index (+)
16	Limit Ground	Limit Ground
17	Travel Limit (-) Input	Travel limit (+) Input
18	Travel Limit (-) Input	Travel Limit (-) Input
19	Encoder Channel A (+)	Encoder Channel A (+)
20	Encoder Channel B (+)	Encoder Channel B (+)
21	Encoder Supply: +5V	Encoder Supply: +5V
22	Encoder Ground	Encoder Ground
23	Encoder Channel A (-)	Encoder Channel A (-)
24	Encoder Channel B (-)	Encoder Channel B (-)
25	Encoder Index (-)	Encoder Index (-)

Table C.2: Driver Card Connector Pin-Outs

Signal Descriptions (Motor Driver Card, 25-Pin I/O Connector) – CONTINUED

DC Motor Phase(-) Output

This output must be connected to the negative lead of the DC motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48V DC.

Stepper Motor Phase 1 Output

This output must be connected to Winding A+ lead of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48V DC.

Stepper Motor Phase 2 Output

This output must be connected to Winding A- lead of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48V DC.

Stepper Motor Phase 3 Output

This output must be connected to Winding B+ lead of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48V DC.

Stepper Motor Phase 4 Output

This output must be connected to Winding B- lead of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48V DC.

Tacho Generator(+) Input

This input can be connected to the positive lead of a tachometer. The maximum input voltage range is ± 10V.

Tacho Generator(-) Input

This input can be connected to the negative lead of a tachometer. The maximum input voltage range is ± 10V.

Travel Limit(+) Input

This input is pulled-up to +5V with a 4.7KΩ resistor by the controller and represents the stage negative direction hardware travel limit. The active true state is user-configurable. The default is active HIGH.

Travel Limit(-) Input

This input is pulled-up to +5V with a 4.7KΩ resistor by the controller and represents the stage negative direction hardware travel limit. The active true state is user-configurable (default is active HIGH).

Encoder A(+) Input

The A(+) input is pulled-up to +5V with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The A(+) encoded signal originates from the stage position feedback circuitry and is used for position tracking.

Encoder A(-) Input

The A(-) input is pulled-up to +5V and pulled down to ground with 1KΩ resistors. This facilitates both single- and double- ended signal handling into a 26LS32 differential receiver. The A(-) encoded signal originates from the stage position feedback circuitry and is used for position tracking.

Encoder B(+) Input

The B(+) input is pulled-up to +5V with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The B(+) encoded signal originates from the stage position feedback circuitry and is used for position tracking.

Encoder B(-) Input

The B(-) input is pulled-up to +5V and pulled down to ground with 1KΩ resistors. This facilitates both single- and double- ended signal handling into a 26LS32 differential receiver. The B(-) encoded signal originates from the stage position feedback circuitry and is used for position tracking.

Encoder Ground

Ground reference for encoder feedback.

Home Input

This input is pulled-up to +5V with a 1KΩ resistor by the controller. The Home signal originates from the stage and is used for homing the stage to a repeatable location.

Index(+) Input

The (+) Index input is pulled-up to +5V with a 1KΩ resistor by the controller and is buffered with a 26LS32 differential receiver. The (+) Index signal originates from the stage and is used for homing the stage to a repeatable location.

Index (-) Input

The (-) Index input is pulled-up to +5V and pulled down to ground with 1KΩ resistors by the controller. This facilitates both single- and double-ended signal handling into a 26LS32 differential receiver. The (-) Index signal originates from the stage and is used for homing the stage to a repeatable location.

Encoder Supply: +5V, 250mA (Maximum)

+5V supply is available from the ESP7000. The standard supply configuration is +5V (250mA maximum). This supply is provided for stage home, index, travel limit, and encoder feedback circuitry.

Limit Ground

Ground for stage travels limit signals. Limit ground is combined with digital ground at the controller side.

Shield Ground

Motor cable shield ground.

C.1.5 Auxiliary I/O (37-Pin Connector)

This connector provides access to the ESP7000 auxiliary I/O signals. The auxiliary I/O connector provides access to:

(a) additional quadrature encoder counters
(b) digital I/O
(c) E-Stop input.

Connector pin-outs are listed in **Table C.3,** and functionally described in **Section C.1.6.**

C.1.6 Signal Descriptions (Auxiliary I/O, 37-Pin Connector)

+5V, 250mA (maximum)

+5V supply available from the PC.

+12V, 250mA (maximum)

+12V supply available from the PC.

-12V, 250mA (maximum)

-12V supply available from the PC.

Pin #	Description
1	Auxiliary Ch. 7 Input A (+)
2	Auxiliary Ch. 7 Input B (+)
3	Reserved
4	Auxiliary Ch. 8 Input A (+0
5	Auxiliary Ch. 8 Input B (+)
6	Reserved
7	Axis-6 Encoder Input A
8	Reserved
9	Axis-1 position compare output
10	Digital Ground
11	Port A, Bit-0
12	Port A, Bit-2
13	Port B, Bit-0
14	Port B, Bit-2
15	E-Stop
16	+5V, 250mA (maximum)
17	+12V, 250mA (maximum)
18	-12V, 250mA (maximum)
19	Digital Ground
20	Auxiliary Ch. 7 Input A (-)
21	Auxiliary Ch. 7 Input B (-)
22	Reserved
23	Auxiliary Ch. 8 Input A (-)
24	Auxiliary Ch. 8 Input B (-)
25	Reserved
26	Axis-6 Encoder Input B
27	Reserved
28	Axis-2 position compare output
29	Global position latch input
30	Port A, Bit-1
31	Port A, Bit-3
32	Port B, Bit-1
33	Port B, Bit-3
34	Reserved
35	DSP Reset Output
36	Reserved
37	Reserved

Table C.3: Auxiliary I/O Connector Pin-Outs

Axis-1 Position Compare Output

The position compare output for axis #1 is a TTL-buffered output. This active low output pulse is used to trigger external devices/events whenever the crossing of a desired position is detected by the ESP motion controller hardware. The output pulse width is in the range of 30 to 60ns. Since this output is buffered by IC SN74F21, it can source 1mA (maximum) or sink 20mA (maximum). Please refer to ***Position Compare Output Triggering*** section in the Advanced Capabilities Tutorial for further details.

Axis-2 Position Compare Output

The position compare output for axis #2 is a TTL-buffered output. This active low output pulse is used to trigger external devices/events whenever the crossing of a desired position is detected by the ESP motion controller hardware. The output pulse width is in the range of 30 to 60ns. Since this output is buffered by IC SN74F21, it can source 1mA (maximum) or sink 20mA (maximum). Please refer to ***Position Compare Output Triggering*** section in the Advanced Capabilities Tutorial for further details.

Axis Ch. 6 Input A

This input is hard-wired to the 50-pin Axis-6 encoder channel A (+) input. Therefore, this input can only be used if five (5) (or fewer) axes of motion are incorporated. The single-ended A input is pulled-up to +5V with a 1KΩ resistor. The signal is buffered with a 26LS32 receiver. The A quadrature encoded signal originates from external feedback circuitry and is used for position tracking.

Axis Ch. 6 Input B

This input is hard-wired to the 50-pin Axis-6 encoder channel B(+) input. Therefore, this input can only be used if five (5) (or fewer) axes of motion are incorporated. The single-ended B input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 receiver. The B quadrature encoded signal originates from external feedback circuitry and is used for position tracking.

Auxiliary Ch. 7 Input A (+0

The A(+) input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The A (+) quadrature encoded signal originates from external feedback circuitry and is used for position tracking.

Auxiliary Ch. 7 Input A (-)

The A(-) input is pulled-up to +5 volts and pulled down to ground with 1KΩ resistors.

This facilitates both single- and double-ended signal handling into a 26LS32 differential receiver. The A(-) quadrature encoded signal originates from external feedback circuitry and is used for position tracking.

Auxiliary Ch. 7 Input B (+)

The B(+) input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The B(-) quadrature encoded signal originates from external feedback circuitry and is used for position tracking.

Auxiliary Ch. 7 Input B (-)

The B(-) input is pulled-up to +5 volts and pulled down to ground with 1KΩ resistors. This facilitates both single- and double-ended signal handling into a 26LS32 differential receiver. The B(-) quadrature encoded signal originates from external feedback circuitry and is used for position tracking.

Auxiliary Ch. 8 Input A(+)

The A(+) input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The A(+) quadrature encoded signal originates from external feedback circuitry and is used for position tracking.

Auxiliary Ch. 8 Input A(-)

The A(-) input is pulled-up to +5 volts and pulled down to ground with 1KΩ resistors. This facilitates both single- and double-ended signal handling into a 26LS32 differential receiver. The A(-) quadrature encoded signal originates from external feedback circuitry and is used for position tracking.

Auxiliary Ch. 8 Input B(+)

The B(+) input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The B(+) quadrature encoded signal originates from external feedback circuitry and is used for positioning tracking.

Auxiliary Ch. * Input B(-)

The B(-) input is pulled-up to –5 volts and pulled down to ground with 1KΩ resistors. This facilitates both single- and double-ended signal handling into a 26LS 32 differential receiver. The B(-) quadrature encoded signal originates from external feedback circuitry and is used for position tracking.

Digital Ground

Ground reference used for all digital signals.

Digital I/O

The digital I/O can be programmed to be either input or output (in 8-bit blocks) via software and is pulled-up to +5 volts with a 1KΩ resistor. In the auxiliary connector only 4-bits of digital I/O from each 8-bit block are made available so that users can program 4-bits as output and 4-bits of input. Bits 0-3 are controlled by Port A API calls and bits 8-11 by Port B API calls. Note that these digital I/O signals are internally hard-wired to digital I/) connector signals. When configured as output, each bit can source 64mA (maximum). When configured as input, each bit can sink 32mA (maximum).

DSP Reset Output

The Reset output is a TTL-buffered output that represents ESP7000 hardware reset status of the controller itself. When the controller is held in a reset state this output is a logical LOW. This output can be used to reset external devices whenever the ESP7000 DSP is reset.

E-Stop Input

The Emergency Stop (E-Stop) input is pulled-up to +5 volts with a 1KΩ resistor. The incoming signal to this input must be a low-going, TTL-compatible digital pulse with a minimum 10 microsecond duration. This signal should be debounced so as not to generate multiple E-Stop within a 100 millisecond time period. When this signal is asserted the controller will perform an Emergency Stop procedure as configured by the user. When used with the ESP7000 motor driver, this signal is coupled to the Stop All front panel pushbutton.

Global Position Capture Input

The position capture input is pulled-up to +5 volts with a 1KΩ resistor. The ESP motion controller can be setup to initiate analog and/or encoder data acquisition when this trigger is detected. The input pulse must be a low going pulse, with a width of at least 80ns. The user specified analog and/or encoder (position) data will be captured 60ns after falling edge of global position capture input trigger. If there is a de-bounce on the input signal, the data is captured 60ns after the last falling edge. Please refer to ***Position Capture Input Triggering*** section in the Advanced Capabilities Tutorial for further details.

C.1.7 IEEE488 Interface Connector (24 Pin)

The IEEE488 Interface Connector has a standard configuration, as shown in **Table C.4.**

Description	Pin #	Pin #	Description
DIO1	1	13	DIO5
DIO2	2	14	DIO6
DIO3	3	15	DIO7
DIO4	4	16	DIO8
EOI	5	17	REN
DAV	6	18	GND
NRFD	7	19	GND
NDAC	8	20	GND
IFC	9	21	GND
SRQ	10	22	GND
ATN	11	23	GND
SHIELD	12	24	SIG. GND

Table C.4: IEEE488 Interface Connector

C.1.8 RS-232C Interface Connector (9-Pin D-Sub Female)

The RS-232C interface uses a 9-pin Sub-D Female connector. The back panel connector pin-out is shown in **Figure C.1.**

Pin No. Description

1 ---------- -12V
2 ---------- TXD
3 ---------- RXD
4 ---------- N/C
5 ---------- GND
6 ---------- -12V
7 ---------- CTS
8 ---------- RTS
9 ---------- N/C

Figure C.1: RS-232C Connector Pin-Out

C.1.9 RS-232C Interface Cable

Figure C.2 shows a simple straight through, pin-to-pin cable with 9 conductors that can be used to connect to a standard 9 pin RS232 host.

Pin No. Pin No.

1 ---------- 1
2 ---------- 2
3 ---------- 3
4 ---------- 4
5 ---------- 5
6 ---------- 6
7 ---------- 7
8 ---------- 8
9 ---------- 9

9-Pin D-Sub Male Connector on Controller Side

9-Pin D-Sub Female Connector on Computer Side

Figure C.2: Conductor, pin-to-pin RS-232C Interface Cable

C.1.10 Motor Interlock Connector (BNC)

This connector is provided for the wiring of one or more remote Emergency Stop switches. They will have the same effect as the front panel STOP ALL button.

The switch has to be normally closed for operation. If more than one switch is installed, they should be connected in series. The minimum rating for the switches should be **50mA** at **5V.**

The ESP7000 is supplied with a dust cap that automatically provides the proper connection for operation if no switch is connected.

Pin #	**Description**
Center Pin	Input Emergency Stop must always be connected to the shell (GND) during normal controller operation. An open circuit is equivalent to pressing STOP ALL on the front panel.
Connector Shield	Provides GND for switch

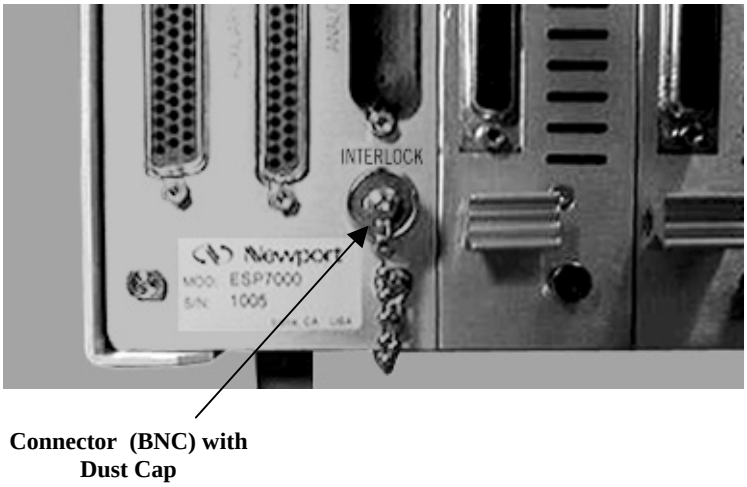


Figure C.3: Motor Interlock Connector (BNC) Image

C.1.11 Analog I/O (25-Pin) Connector

This connector interfaces the ESP7000 controller to customer-defined analog I/O devices. Connector pin-outs are listed in **Table C.5,** and functionally described in **Section C.1.12.**

C.1.12 Signal Descriptions (Analog I/O, 25-Pin Connector)

+5V, 250mA (maximum)

+5V supply available from the PC.

+12V, 250mA (maximum)

+12V supply available from the PC.

-12V, 250mA (maximum)

-12V supply available from the PC.

Analog Ground

Analog-to-Digital Converter (ADC) signal ground.

Analog Input 0-7

Refer to Section 4.3, Data Acquisition for information.

Function	DB 25 Pins
Digital Ground	13
E-Stop Interrupt to DSP	25
-12V, 250mA (maximum)	12
Reserved	24
+12V, 250mA (maximum)	11
Reset Output from DSP	23
+5V, 250mA (maximum)	10
Reserved	22
Reserved	9
Reserved	21
Analog Ground	8
Axis-6 Servo DAC Output	20
Analog Ground	7
Axis-5 Servo DAC Output	19
Analog Ground	6
Reserved	18
Analog Ground	5
Analog Input 0	17
Analog Input 1	4
Analog input 2	16
Analog Input 3	3
Analog Input 4	15
Analog Input 5	2
Analog Input 6	14
Analog Input 7	1

Table C.5: Analog Connector Pin-Outs

Signal Descriptions (Analog I/O, 25-Pin Connector) – CONTINUED

Axis-5, 6 Servo DAC Output

The servo Digital-to-Analog Converter (DAC) output is the ± 10 volt, 18-bit resolution control signal used to control DC servo motors. These signals are made available on this connector for special applications where the users need to observe the actual control signal output to the amplifier for analysis purposes.

Digital Ground

Ground reference used for all digital signals.

E-Stop Interrupt to DSP

This input is normally high. Pulling it low will interrupt the ESP7000 Controller.

Reset Output from DSP

This signal is a buffered active-low signal connected to the ESP7000 Controller reset signal.

Appendix D – Binary Conversion Table

Some of the status reporting commands return an ASCII character that must be converted to binary. To aid with the conversion process, the following table converts all character used and some other common ASCII symbols to decimal to decimal and binary. To also help in working with the I/O port related commands, the table is extended to a full byte, all 256 values.

Number (decimal)	ASCII Code	Binary Code
0	*Null*	00000000
1	*Soh*	00000001
2	*Stx*	00000010
3	*Etx*	00000011
4	*Eot*	00000100
5	*Enq*	00000101
6	*Ack*	00000110
7	*Bel*	00000111
8	*Bs*	00001000
9	*Tab*	00001001
10	*Lf*	00001010
11	*Vt*	00001011
12	*Ff*	00001100
13	*Cr*	00001101
14	*So*	00001110
15	*Si*	00001111
16	*Dle*	00010000
17	*Dc1*	00010001
18	*Dc2*	00010010
19	*Dc3*	00010011
20	*Dc4*	00010100
21	*Nak*	00010101
22	*Syn*	00010110
23	*Etb*	00010111
24	*Can*	00011000
25	*Em*	00011001
26	*Eof*	00011010
27	*Esc*	00011011
28	*Fs*	00011100
29	*Gs*	00011101
30	*Rs*	00011110
31	*Us*	00011111
32	*Space*	00100000

Table D.1: Binary Conversion Table Listing

Number (decimal)	ASCII Code	Binary Code
33	**!**	00100001
34	**"**	00100010
35	**#**	00100011
36	**S**	00100100
37	**%**	00100101
38	**&**	00100110
39	**'**	00100111
40	**(**	00101000
41	**)**	00101001
42	*****	00101010
43	**+**	00101011
44	**,**	00101100
45	**-**	00101101
46	**.**	00101110
47	**/**	00101111
48	**0**	00110000
49	**1**	00110001
50	**2**	00110010
51	**3**	00110011
52	**4**	00110100
53	**5**	00110101
54	**6**	00110110
55	**7**	00110111
56	**8**	00111000
57	**9**	00111001
58	**:**	00111010
59	**;**	00111011
60	<	00111100
61	**=**	00111101
62	>	00111110
63	**?**	00111111
64	**@**	01000000
65	**A**	01000001
66	**B**	01000010
67	**C**	01000011
68	**D**	01000100
69	**E**	01000101
70	**F**	01000110
71	**G**	01000111
72	**H**	01001000
73	**I**	01001001
74	**J**	01001010
75	**K**	01001011
76	**L**	01001100
77	**M**	01001101
78	**N**	01001110
79	**O**	01001111
80	**P**	01010000
81	**Q**	01010001
82	**R**	01010010

Table D.1: Binary Conversion Table Listing (Continued)

Number (decimal)	ASCII Code	Binary Code
83	**S**	01010011
84	**T**	01010100
85	**U**	01010101
86	**V**	01010110
87	**W**	01010111
88	**X**	01011000
89	**Y**	01011001
90	**Z**	01011010
91	**[**	01011011
92	\	01011100
93	**]**	01011101
94	**^**	01011110
95	**_**	01011111
96	**'**	01100000
97	**A**	01100001
98	**B**	01100010
99	**C**	01100011
100	**D**	01100100
101	**E**	01100101
102	**F**	01100110
103	**G**	01100111
104	**H**	01101000
105	**I**	01101001
106	**J**	01101010
107	**K**	01101011
108	**L**	01101100
109	**M**	01101101
110	**N**	01101110
111	**O**	01101111
112	**P**	01110000
113	**Q**	01110001
114	**R**	01110010
115	**S**	01110011
116	**T**	01110100
117	**U**	01110101
118	**V**	01110110
119	**W**	01110111
120	**X**	01111000
121	**Y**	01111001
122	**Z**	01111010
123	**{**	01111011
124	**|**	01111100
125	**}**	01111101
126	**~**	01111110
127		01111111
128		10000000
129		10000001
130		10000010
131		10000011
132		10000100

Table D.1: Binary Conversion Table Listing (Continued)

Number (decimal)	ASCII Code	Binary Code
133		10000101
134		10000110
135		10000111
136		10001000
137		10001001
138		10001010
139		10001011
140		10001100
141		10001101
142		10001110
143		10001111
144		10010000
145		10010001
146		10010010
147		10010011
148		10010100
149		10010101
150		10010110
151		10010111
152		10011000
153		10011001
154		10011010
155		10011011
156		10011100
157		10011101
158		10011110
159		10011111
160		10100000
161		10100001
162		10100010
163		10100011
164		10100100
165		10100101
166		10100110
167		10100111
168		10101000
169		10101001
170		10101010
171		10101011
172		10101100
173		10101101
174		10101110
175		10101111
176		10110000
177		10110001
178		10110010
179		10110011
180		10110100
181		10110101
182		10110110

Table D.1: Binary Conversion Table Listing (Continued)

Number (decimal)	ASCII Code	Binary Code
183		10110111
184		10111000
185		10111001
186		10111010
187		10111011
188		10111100
189		10111101
190		10111110
191		10111111
192		11000000
193		11000001
194		11000010
195		11000011
196		11000100
197		11000101
198		11000110
199		11000111
200		11001000
201		11001001
202		11001010
203		11001011
204		11001100
205		11001101
206		11001110
207		11001111
208		11010000
209		11010001
210		11010010
211		11010011
212		11010100
213		11010101
214		11010110
215		11010111
216		11011000
217		11011001
218		11011010
219		11011011
220		11011100
221		11011101
222		11011110
223		11011111
224		11100000
225		11100001
226		11100010
227		11100011
228		11100100
229		11100101
230		11100110
231		11100111
232		11101000

Table D.1: Binary Conversion Table Listing (Continued)

Number (decimal)	ASCII Code	Binary Code
233		11101000
234		11101001
235		11101010
236		11101011
237		11101100
238		11101101
239		11101110
240		11101111
241		11110001
242		11110010
243		11110011
244		11110100
245		11110101
246		11110110
247		11110111
248		11111000
249		11111001
250		11111010
251		11111011
252		11111100
253		11111101
254		11111110
255		11111111

Table D.1: Binary Conversion Table Listing (Continued)

Appendix E – System Upgrades

The modular design of the ESP7000 makes it easy for qualified individuals to upgrade the unit in the field. Upgrade kits to add more axis, IEEE488 or display option are available upon request. Call Newport for details.

NOTE: ALL UPGRADES TO THE ESP7000 ARE TO BE MADE FROM THE REAR PANELS, BY SLIDING OUT AN AXIS DRIVER. *Do not remove the Top Cover or the Front Panel*.

This section describes how to upgrade an ESP7000 from 4 to 5 axes. Other axes upgrades can be performed accordingly.

WARNING

Opening or removing covers will expose you to hazardous voltages.

Refer all servicing internal to this controller enclosure to qualified service personnel who should observe the following precautions before proceeding:

- **Turn power OFF and unplug the unit from its power source**
- **Disconnect all cables**
- **Remove any jewelry from hands and wrists**
- **Use only insulated hand tools**
- **Maintain grounding by wearing a wrist strap attached to instrument chassis.**

CAUTION

The ESP7000 contains static sensitive devices. Exercise appropriate caution when handling ESP7000 boards, cables and other internal components.

CAUTION

Do not install anything into your ESP7000 except items provided by Newport specifically for installation into the ESP7000.

E.1 Adding Axes

1. Turn the power off and unplug the power cord from the controller. Disconnect all cables from the controller.
2. Turn and loosen the (2) thumb screws as shown below. See **Figure E.1,** which shows how to remove an Axis Driver Module.

Figure E.1: Removal of an Axis Driver Module

3. Carefully slide out the axis Driver Module.
4. Insert the driver module for axis 5. The connector of the driver module is keyed to prevent insertion with improper polarity. Make sure the keys line up properly before you try to insert the module (See **Figure E.2**).
5. Attach the driver panel to the rear panel of the unit with the (2) supplied thumb screws.

The unit is now ready for use.

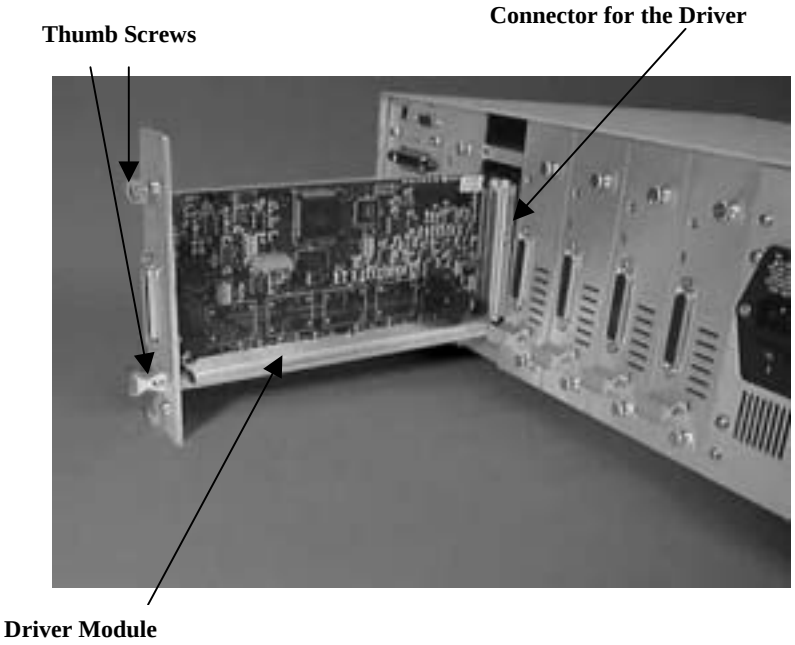


Figure E.2: Interior of the Axis Driver Module showing connector

E.2 Adding IEEE488

1. Follow steps 1-3 adding axes.
2. Insert the IEEE-488 driver module in the connector. The connector of the module is keyed to prevent insertion with improper polarity. Make sure the keys line up properly before you try to insert the module.
3. Attach the IEEE-488 panel to the rear panel of the unit with the (2) supplied thumb screws.

The unit is now ready for use.

Appendix F – ESP Configuration Logic

Each time a stage or stages are disconnected/re-connected, or a system is powered down and then back up, the ESP7000 controller card verifies the type of stage(s) present and re-configures its own flash memory if necessary (i.e., new stage). The controller card in the ESP7000 system configuration, the stage motor and the current type are defined, the controller card will configure the specific axis. Specific ESP logic is shown in **Figure F.1.**

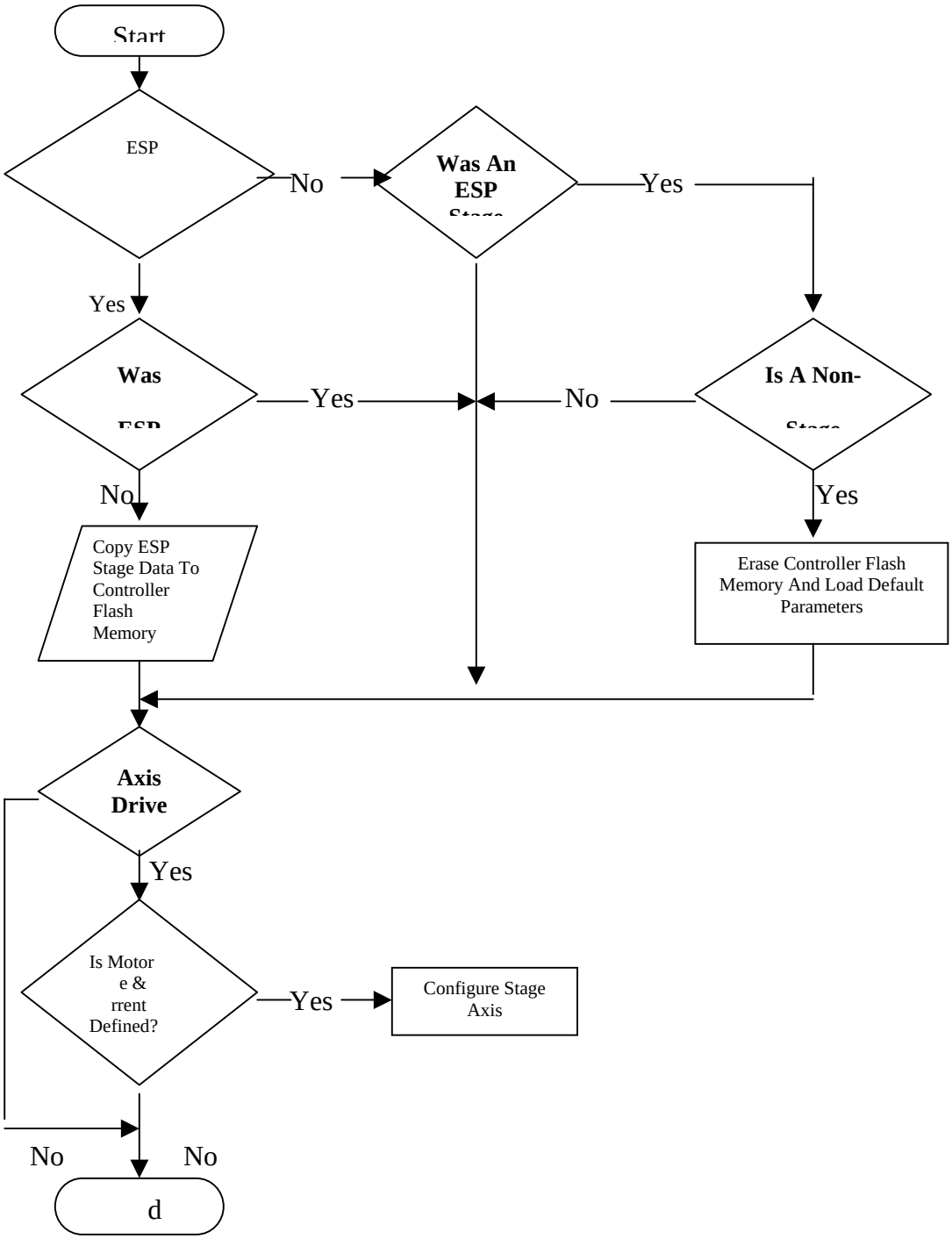


Figure F.1: Configuration Logic

Appendix G – Programming Non-ESP Compatible Stages

Newport positioners, or stages, with integrated configuration memory devices are said to be "ESP Compatible". It is not necessary to manually enter individual stage parameters (E. G., motor current, maximum velocity, etc.) with an ESP compatible stage. All necessary configuration settings will be automatically loaded after system reset with ESP compatible positioners.

When a positioner is said to be ESP incompatible, all that really means is that it does not have the integrated memory device that allows for automatic configuration. Therefore, it will have to be configured manually – as is customary with all non-ESP controllers.

There are two (2) basic levels of positioner configuration. The first level assumes that the stage is Newport controller compatible. That is to say that hardware travel limits, encoder feedback counting, etc… are designed to operate with Newport controllers. In this case only a certain amount of commands are necessary to setup the axis before moving the stage. The second level is when a stage is not a standard Newport stage and various compatibility issues need to be addressed. This scenario requires additional command configurations.

The following are examples of how to configure an ESP controller axis for a standard Newport stage that is not equipped with the "ESP Compatible" memory device (i.e., level 1):

Example #1: DC Servo on axis 1

1qm1	//set motor type to DC servo
1qi0.15	//set motor maximum current to 0.15 amps
1qv30	//set motor voltage to 30 volts
1sn7	//set user units to degrees
1su0.005	//set resolution to 0.005 degrees
1vu15	//set maximum velocity to 15 deg/sec
1va7	//set working velocity to 7 deg/sec
1oh7	//set Homing speed to 7 deg/sec
1jh7	//set jog high speed to 7 deg/sec
1jw1	//set jog low speed to 1 deg/sec
1au40	//set maximum acceleration to 40 deg/sec^2

1ac20	//set working acceleration to 20 deg/sec^2
1ag25	//set deceleration to 2.5 deg/sec^2
1fe0.5	//set following error threshold to 0.5 deg
1kp600	//set PID proportional gain to 600
1kd600	//set PID derivative gain to 600
1ki350	//set PID integral gain to 350
1ks300	//set PID integral saturation gain to 300
1tj1	//set trajectory mode to trapezoidal
1qd	//update motor driver configuration
sm	//save configuration to non-volatile memory

Example #2: Stepper stage on axis 1

1qm3	//set motor type to Commutated stepper (ESP300 only)
1qi 1	//set motor maximum current to 1 amp
1qv30	//set motor voltage to 30 volts
1sn2	//set user units to millimeters
1su 0.001	//set resolution to 1 micron
1fr0.01	//set stepper motor full step resolution to 10 micron
1qs100	//set micro-stepping resolution to 100x
1vu20	//set maximum velocity to 20 mm/sec
1va10	//set working velocity to 10 mm/sec
1jh10	//set jog high velocity to 10 mm/sec
1jw1	//set jog low velocity to 1 mm/sec
1oh10	//set Homing velocity to 10 mm/sec
1au50	//set maximum acceleration to 50 mm/sec^2
1 ac 30	//set acceleration to 30 mm/sec^2
1ag30	//set deceleration to 30 mm/sec^2
1fe1	//set following error threshold to 1 mm
1tj1	//set trajectory mode to trapezoidal
1qd	//update motor driver configuration
sm	//save configuration to non-volatile memory

The following commands should be reviewed for proper axis compatibility when connecting to a non-Newport stage – assuming that it is electrically compatible with the controller (i.e., level 2):

ZA	//set amplifier configuration
ZB	//set feedback configuration
ZH	// set hardware limit configuration

Appendix H – Factory Service

This section contains information regarding factory service for the ESP7000 System. The user should not attempt any maintenance or service of the system or optional equipment beyond the procedures outlined in the Trouble-Shooting appendix of this manual. Any problem that cannot be resolved should be referred to Newport Corporation. Technical Customer Support contact information is listed in **Table H-1.**

Telephone	1-800-222-6440
Fax	1-949-253-1479
Email	rma.service@newport.com
Web Page URL	www.newport.com/srvc/service.html

Table H.1: Technical Customer Support Contacts

Contact Newport to obtain information about factory service. Telephone contact number(s) are provided on the Service Form (see next page). Please have the following information available:

- Equipment model number (ESP7000)
- Equipment serial number (for the ESP7000)
- Distribution revision number from a floppy disk
- Problem description (document using the Service Form, on the following page).

If the instrument is to be returned for repair, you will be given a Return Authorization Number that should be referenced in your shipping documentation. Complete a copy of the Service Form on the next page and include it with your shipment.

Service Form

Newport Corporation
U.S.A. Office: 949/863-3144
FAX: 949/253-1800

RETURN AUTHORIZATION # _____
(Please obtain prior to return of item)

Name _____

Company _____

Address _____ Date _____

Country _____ Phone Number _____

P.O. Number _____ FAX Number _____

Item(s) Being Returned:

Model # _____ Serial # _____

Description: _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____