ESP6000 UNIDRIVE6000
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, the user should write or call your their Newport office or representative, or contact Newport headquarters in Irvine, California. The user 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 other 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 June 2000

Copyright 2000 by Newport Corporation, Irvine, CA. All rights reserved. No part of this manual may be reproduced or copied without the prior written approval of Newport Corporation.

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.

© 2000 Newport Corporation

1791 Deere Ave.

Irvine, CA 92606

(949) 863-3144

P/N 22945-01, **Rev. E**

IN-04971 (10/00)

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: **ESP6000 UNIDRIVE6000**

Year CE mark affixed:

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

Section	Title	Page
Warranty		ii
Limitation of Warranty		ii
EU Declaration of Conformity		iii
List of Figures		xii
List of Tables		xvii
Section 1 – Introduction		**1-1**
1.1	Scope	1-1
1.2	Safety Considerations	1-2
1.3	Conventions and Definitions	1-4
1.3.1	Definitions and Symbols	1-4
1.3.2	Terminology	1-5
1.4	System Overview	1-6
1.4.1	Features	1-7
1.4.2	Specifications	1-8
	ESP6000 Controller Card	1-8
	UniDrive6000 Universal Motor Driver	1-9
	Environmental Limits	1-10
Section 2 – System Set-Up		**2-1**
2.1	Unpacking	2-1
2.2	PC Hardware and Software Requirements	2-2
2.3	Equipment Controls and Indicators	2-2
2.4	Installation and Connection	2-5
2.4.1	Installing the ESP6000 Controller Card and Software Driver	2-5
2.4.2	Installing Windows Software	2-8
2.4.3	Verifying Communication Between the ESP6000 Card and the PC	2-15
2.4.4	Selecting the UniDrive6000 Line Voltage	2-15
2.4.5	Connecting Stages	2-17
2.4.6	Connecting the UniDrive6000 to the ESP6000 Controller Card	2-18
Section 3 – Quick Start		**3-1**
3.1	General Description	3-1
3.2	Motor On	3-1

3.3 Homing a Stage 3-3
3.4 Jog.....3-4
3.5 System Shut-Down 3-6

Section 4 – Windows Utilities..... 4-1

4.1 Motion Utility 4-1
4.1.1 General Description 4-1
4.1.2 Features.....4-1
4.1.3 Operation 4-1
4.1.3.1 Communication Initialization.....4-3
4.1.3.2 File Menu.....4-4
Reset System.....4-5
Save To Flash 4-5
Save To File.....4-5
Restore From File 4-5
Convert to ASCII Command File 4-5
Demo Mode 4-5
Exit.....4-6
4.1.3.3 Setup Menu.....4-6
Motion 4-6
Faults 4-11
Hardware 4-13
Firmware.....4-17
UniDrive 4-17
4.1.3.4 Motion Menu 4-18
Stop.....4-18
Home 4-19
Jog.....4-19
Cycle.....4-20
Enable 4-20
4.1.3.5 Status Menu 4-21
Position 4-21
Hardware 4-22
4.1.3.6 Window Menu 4-23
4.1.3.7 Help Menu 4-24
About ESP 6000 4-24
Internet Resources 4-24
ESP6000 Commands 4-25
4.2 Servo Tuning Utility.....4-25
4.2.1 General Description.....4-25
4.2.2 Features.....4-25
4.2.3 Operation 4-25
Communication Initialization 4-25

Display....4-27
Motion....4-27
PID....4-29
Setup....4-29
Terminal....4-30
About....4-31

Section 5 – Programming....5-1

5.1 Introduction....5-1
5.2 How to Communicate with the Controller....5-1
5.3 General Description....5-3
5.3.1 Windows Programming....5-3
5.3.2 How to Use the Dynamic Link Library....5-3
5.3.3 Minimum Software Requirements....5-3
5.4 Description of Commands....5-4
5.4.1 ASCII Commands....5-4
Command Mode....5-4
Program Execution Mode....5-5
5.4.2 Binary Commands....5-6
5.5 Commands (**ASCII**)....5-7
5.5.1 Command Syntax....5-7
5.5.2 Summary of Command Syntax....5-8
5.5.3 Command Summary....5-9
5.5.3.1 Command List by Category....5-15
5.5.3.2 Command List – Alphabetical....5-21
5.5.4 Description of Commands....5-24
5.6 Commands (Binary)....5-210
5.6.1 Command Summary....5-210
5.6.2 Command List....5-213
Initialization....5-215
ADCII Command Interface....5-221
General Mode Selection....5-227
Status Functions....5-251
Motion and Position Control....5-265
Trajectory Definition....5-272
Flow Control and Sequencing....5-284
Motion Device Parameters....5-288
Programming....5-309
Digital Filters....5-311
I/O Function....5-324
5.6.3 User Programming....5-356
5.6.3.1 Visual C....5-356

5.6.3.2 Visual Basic 5-356
5.6.3.3 LabView 5-356
5.6.4 Error Handling 5-358
5.6.5 Multiple Board Support 5-359
5.6.5.1 Introduction 5-359
5.6.5.2 ESP-Util 5-359
5.6.5.3 ESP-Tuning 5-359
5.6.5.4 API Calls Supporting Multiple ESP6000 Boards 5-360

Section 6 – Motion Control Tutorial..... 6-1

6.1 Motion Systems 6-1
6.2 Specification Definitions 6-2
6.2.1 Following Error 6-3
6.2.2 Error 6-3
6.2.3 Accuracy 6-3
6.2.4 Local Accuracy 6-4
6.2.5 Resolution 6-6
6.2.6 Minimum Incremental Motion 6-6
6.2.7 Repeatability 6-7
6.2.8 Backlash (Hysteresis) 6-8
6.2.9 Pitch, Roll, and Yaw 6-9
6.2.10 Wobble 6-10
6.2.11 Load Capacity 6-10
6.2.12 Maximum Velocity 6-11
6.2.13 Minimum Velocity 6-11
6.2.14 Velocity Regulation 6-12
6.2.15 Maximum Acceleration 6-12
6.2.16 Combined Parameters 6-13
6.3 Control Loops 6-13
6.3.1 PID Servo Loops 6-14
P Loop 6-15
PI Loop 6-16
PID Loop 6-16
6.3.2 Feed-Forward Loops 6-17
6.4 Motion Profiles 6-19
6.4.1 Move 6-19
6.4.2 Jog 6-20
6.4.3 Home Search 6-21
6.5 Encoders 6-24
6.6 Motors 6-27
6.6.1 Stepper Motors 6-28
6.6.2 DC Motors 6-34
6.7 Drivers 6-35
6.7.1 Stepper Motor Drivers 6-35

6.7.2 Unipolar-Bipolar Drivers.....6-37
6.7.3 DC Motor Drivers.....6-38
PWM Drivers.....6-41

Section 7 – Servo Tuning 7-1

7.1 Tuning Principles.....7-1
7.2 Tuning Procedures.....7-1
7.2.1 Hardware And Software Requirements.....7-2
7.2.2 Correcting Axis Oscillation.....7-2
7.2.3 Correcting Following Error 7-3
7.2.4 Points To Remember 7-4

Section 8 – Optional Equipment 8-1

8.1 ESP6000 Controller Card.....8-1
8.1.1 Terminal Block Board 8-2
8.1.2 Analog I/O Cable.....8-3
8.1.3 Digital I/O Cable 8-4
8.1.4 Auxiliary I/O Cable 8-6
8.1.5 Driver Interface (100-100 pin) Cable.....8-7
8.1.6 Motor/Driver (100-68 pin) Cable 8-7
8.2 UniDrive6000 8-8
8.2.1 Motor Driver Card.....8-8
8.2.2 Rack-Mount Ears.....8-9

Section 9 – Advanced Capabilities..... 9-1

9.1 Motion Control Software Overview 9-1
9.1.1 Introduction 9-1
9.1.2 Control API.....9-1
9.1.2.1 System Initialization 9-1
9.1.2.2 Configuration.....9-1
9.1.2.3 Axis Control 9-2
9.1.3 Trajectory Control Process 9-2
9.2 Data Acquisition.....9-3
9.2.1 Introduction 9-3
9.2.2 Analog Data Acquisition 9-3
9.2.3 Trace Variable Data Acquisition 9-5
9.2.4 Streaming Data Acquisition.....9-8
9.2.4.1 Introduction 9-8
9.2.4.2 Basic Description.....9-8
9.2.4.3 Streaming Error Conditions 9-8

9.2.4.4 A Programming Example ..9-9
9.3 PCI Bus Overview 9-10
9.4 Grouping 9-12
9.4.1 Introduction 9-12
9.4.2 Defining a Group & Group Parameters 9-13
9.4.2.1 Creating a Group 9-13
9.4.2.2 Defining Group Parameters 9-14
9.4.3 Making Linear and Circular Moves..9-14
9.4.3.1 Making Linear Move 9-14
9.4.3.2 Making Circular Move 9-15
9.4.4 Making Contours 9-16
9.4.5 Miscellaneous Commands 9-18
9.5 Slaving a Stage to Trackball, Joystick, or a Different Stage 9-19
9.5.1 Introduction 9-19
9.5.2 Slave to a Different Stage 9-20
9.5.3 Slave to a Trackball 9-20
9.5.4 Slave to a Joystick 9-20
9.6 Closed Loop Stepper Motor Positioning 9-21
9.6.1 Introduction 9-21
9.6.2 Feature Implementation 9-22
9.7 Synchronize Motion to External and Internal Events 9-23
9.7.1 Introduction 9-23
9.7.2 Using DIO to Execute Stored Programs 9-24
9.7.3 Using DIO to Inhibit Motion 9-25
9.7.4 Using DIO to Monitor Motion Status 9-26
9.8 Position Compare Output Triggering 9-27
9.8.1 Introduction 9-27
9.8.2 Feature Setup 9-27
9.8.3 Hardware Required 9-29
9.9 Position Capture Input Triggering 9-30
9.9.1 Introduction 9-30
9.9.2 Feature Setup 9-30
9.9.3 Hardware Required 9-32

Appendix A – Error Messages A-1

Appendix B – Trouble-Shooting and Maintenance B-1

B.1 Trouble-Shooting Guide B-2
B.2 Fuse Replacement B-3
B.2.1 Replacing Fuses On The UniDrive Rear Power Line Panel B-3
B.2.2 Replacing Fuses On The UniDrive Motor Power Supply Board B-4
B.3 Cleaning B-7

Appendix C – Connector Pin Assignments C-1

C.1 ESP6000 Controller Card C-1
C.1.1 Main I/O (100-Pin) Connector C-3
C.1.2 Motor/Driver Interface (100-to-68 Pin) Cable C-8
C.1.3 Digital I/O (50-Pin) JP4 Connector C-11
C.1.4 Auxiliary I/O (40-Pin) JP5 Connector C-13
C.1.5 Analog I/O (26-Pin) JP2 Connector C-19
C.2 UniDrive6000 Universal Motor Driver C-21
C.2.1 Controller Input Connector C-21
C.2.2 Motor Driver Card 25-Pin I/O Connector C-21
C.3 Terminal Block Board C-26
C.3.1 MD4 15-Pin Connector C-27
C.3.2 Eighteen-Lead Connector C-29
C.3.3 Optional Power Supply Connector . C-32
C.3.4 Nine-Lead Connector C-33
C.3.5 Jumpers C-35

Appendix D – Binary Conversion Table D-1

Appendix E – System Upgrades E-1

E.1 ESP6000 Controller Card E-1
E.1.1 Installing New Software E-1
E.1.2 Installing New Firmware E-4

Appendix F – ESP Configuration Logic F-1

Appendix G – Programming Non-ESP Compatible Stages G-1

Appendix H – Factory Service H-1

H.1 Service Form.....H-2

List of Figures

Figure 1.4.1: ESP6000 Controller Card.....1-6
Figure 1.4.2: ESP Configuration 1-7
Figure 2.3.1: ESP6000 Controller Card.....2-3
Figure 2.3.2: UniDrive6000 Front and Rear View.....2-4
Figure 2.4.1: Enclosure Removal.....2-6
Figure 2.4.2: ESP6000 Controller Card Insertion Orientation.....2-7
Figure 2.4.3: Controller Card Device Driver Prompt (Representative Screen Only) 2-7
Figure 2.4.4: Install From Disk Message (Representative Screen Only) 2-8
Figure 2.4.5: System Settings Change Message (Representative Screen Only) 2-8
Figure 2.4.6: Windows 95 Start-Up Screen 2-9
Figure 2.4.7: Control Panel Menu.....2-9
Figure 2.4.8: Add/Remove Programs Properties Menu.2-10
Figure 2.4.9: Install Program From Floppy Disk Or CD-ROM Screen.....2-11
Figure 2.4.10: Run Installation Program 2-11
Figure 2.4.11:ESP Welcome Screen 2-12
Figure 2.4.12: ESP Select Destination Directory Screen 2-12
Figure 2.4.13: ESP Ready To Install Screen.....2-13
Figure 2.4.14: Installing Message Screen.....2-13
Figure 2.4.15: Insert New Disk Message Screen 2-14
Figure 2.4.16: ESP Installation Completed Screen 2-14
Figure 2.4.17: ESP Initialization Screen 2-15
Figure 2.4.18: Line Voltage Select Switch.....2-16
Figure 2.4.19: Stage To UniDrive Connection 2-18
Figure 2.4.20: UniDrive To Controller Card Connection.....2-19
Figure 3.2.1: Motion Drop-Down Menu.....3-2
Figure 3.2.2: Motor Power Menu 3-2
Figure 3.3.1: Home Stages Menu.....3-3
Figure 3.4.1: Jog Menu.....3-4
Figure 3.4.2: Speed Menu 3-5
Figure 3.4.3: Set X(Y) Speed Menu.....3-5
Figure 4.1.1: Main Menu 4-3
Figure 4.1.2: Communication Initialization.....4-4
Figure 4.1.3: File Menu 4-4
Figure 4.1.4: Setup Menu.....4-6
Figure 4.1.5: Setup Resolution Sub-Menu 4-7

Figure 4.1.6: Setup Motion Sub-Menu: Trajectory Generation Tab 4-8
Figure 4.1.7: Setup Motion Sub-Menu: Motion Control Tab 4-8
Figure 4.1.8: Setup Interpolation Sub-Menu: Trajectory Generation Tab 4-9
Figure 4.1.9: Setup Interpolation Sub-Menu: Coordinated Motion Tab 4-10
Figure 4.1.10: Setup Master-Slaving Sub-Menu 4-10
Figure 4.1.11: Setup Error Compensation Sub-Menu ... 4-11
Figure 4.1.12: Setup Faults Sub-Menu: Enable Checking Tab 4-12
Figure 4.1.13: Setup Faults Sub-Menu: Abort Motion Tab 4-12
Figure 4.1.14: Setup Faults Sub-Menu: Disable Motion Tab 4-13
Figure 4.1.15: Setup Faults Sub-Menu: System Faults Tab 4-13
Figure 4.1.16: Setup Hardware: Amplifier I/O Tab 4-14
Figure 4.1.17: Setup Hardware: Analog I/O Tab 4-14
Figure 4.1.18: Setup Hardware: Digital I/O Tab 4-15
Figure 4.1.19: Setup Hardware:Servo DAC Offset Tab 4-15
Figure 4.1.20: Setup Hardware: Travel Limit Tab 4-16
Figure 4.1.21: Setup Hardware: Feedback Tab 4-16
Figure 4.1.22: Setup UniDrive Sub-Menu 4-17
Figure 4.1.23: Motion Menu 4-18
Figure 4.1.24: Stop Button 4-18
Figure 4.1.25: Home Stages Sub-Menu 4-19
Figure 4.1.26: Jog Sub-Menu 4-19
Figure 4.1.27: Cycle Motors Sub-Menu 4-20
Figure 4.1.28: Motor Power Sub-Menu 4-21
Figure 4.1.29: Status Menu 4-21
Figure 4.1.30: Position Status Sub-Menu 4-22
Figure 4.1.31: Hardware Status Sub-Menu 4-23
Figure 4.1.32: Window Menu 4-23
Figure 4.1.33: Help Menu 4-24
Figure 4.1.34: About ESP6000 Sub-Menu 4-24
Figure 4.2.1: Select Communication Port 4-26
Figure 4.2.2: Controller Initialization Completed 4-26
Figure 4.2.3: Servo Tuning Main Screen 4-27
Figure 4.2.4: Servo Tuning-Motion Tab 4-28
Figure 4.2.5: Servo Tuning – PID Tab 4-29
Figure 4.2.6: Servo Tuning – Setup Tab 4-30
Figure 4.2.7: Servo Tuning – Terminal Tab 4-31
Figure 4.2.8: Servo Tuning – About Tab 4-32
Figure 5.2.1: Communication with Controller 5-2

Figure 5.5.1.1: Command Syntax.....5-7
Figure 5.6.3.3: VI Front Panel.....5-357
Figure 5.6.5.2: Select Target Board.....5-359
Figure 6.1.1: Typical Motion Control System.....6-1
Figure 6.2.1: Position Error Test.....6-4
Figure 6.2.2: High Accuracy for Small Motions.....6-5
Figure 6.2.3: Low Accuracy for Small Motions.....6-5
Figure 6.2.4: Effect of Stiction and Elasticity on Small Motions.....6-6
Figure 6.2.5: Error Plot.....6-6
Figure 6.2.6: Error vs. Motion Step Size 6-7
Figure 6.2.7: Hysteresis Plot 6-8
Figure 6.2.8: Real vs. Ideal Position.....6-8
Figure 6.2.9: Pitch, Yaw, and Roll Motion Axes.....6-9
Figure 6.2.10: Pitch, Yaw and Roll.....6-9
Figure 6.2.11: Wobble.....6-10
Figure 6.2.12: Position, Velocity, and Average Velocity.....6-11
Figure 6.3.1: Servo Loop 6-14
Figure 6.3.2: P Loop.....6-15
Figure 6.3.3: PI Loop.....6-16
Figure 6.3.4: PID Loop.....6-17
Figure 6.3.5: Trapezoidal Velocity Profile 6-18
Figure 6.3.6: PID Loop with Feed-Forward.....6-18
Figure 6.3.7: Tachometer-Driven PIDF Loop.....6-19
Figure 6.4.1: Trapezoidal Motion Profile.....6-20
Figure 6.4.2:Position and Acceleration Profiles 6-20
Figure 6.4.3: Origin Switch and Encoder Index Pulse ..6-22
Figure 6.4.4: Slow-Speed Origin Switch Search.....6-22
Figure 6.4.5: High/Low-Speed Origin Switch Search....6-23
Figure 6.4.6: Origin Search From Opposite Direction .6-23
Figure 6.5.1: Encoder Quadrature Output 6-24
Figure 6.5.2: Optical Encoder Scale.....6-25
Figure 6.5.3: Optical Encoder Read Head 6-25
Figure 6.5.4: Single-Channel Optical Encoder Scale and Read Head Assembly 6-26
Figure 6.5.5: Two-Channel Optical Encoder Scale and Read Head Assembly 6-26
Figure 6.6.1: Stepper Motor Operation 6-27
Figure 6.6.2: Four-Phase Stepper Motor.....6-29
Figure 6.6.3: Phase Timing Diagram 6-29
Figure 6.6.4: Energizing Two Phases Simultaneously...6-30
Figure 6.6.5: Timing Diagram, Half-Stepping Motor....6-30
Figure 6.6.6: Energizing Two Phases with Different Intensities.....6-30

Figure 6.6.7: Timing Diagram, Continuous Motion (Ideal) 6-31
Figure 6.6.8: Timing Diagram, Mini-Stepping 6-31
Figure 6.6.9: Single Phase Energization 6-32
Figure 6.6.10: External Force Applied 6-32
Figure 6.6.11: Unstable Point 6-32
Figure 6.6.12: Torque and Tooth Alignment 6-33
Figure 6.6.13: DC Motor 6-34
Figure 6.7.1: Simple Stepper Motor Driver 6-36
Figure 6.7.2: Current Build-up in Phase 6-36
Figure 6.7.3: Effect of a Short ON Time on Current 6-36
Figure 6.7.4: Motor Pulse with High Voltage Chopper 6-37
Figure 6.7.5: Dual H-Bridge Driver 6-38
Figure 6.7.6: DC Motor Voltage Amplifier 6-39
Figure 6.7.7: DC Motor Current Driver 6-39
Figure 6.7.8: DC Motor Velocity Feedback Driver 6-40
Figure 6.7.9:DC Motor Tachometer Gain and Compensation 6-40
Figure 8.1.1:Terminal Block Board 8-2
Figure 8.1.2: Analog I/O Cable 8-4
Figure 8.1.3: Digital I/O Cable 8-5
Figure 8.1.4: Auxiliary I/O Cable Connections 8-6
Figure 8.1.5: Driver Interface (100-100 Pin) Cable 8-7
Figure 8.1.6: Motor/Driver (100-68 Pin) Cable 8-7
Figure 8.2.1: Driver Card 8-9
Figure 8.2.2: Driver Card Installation 8-9
Figure 8.2.3: Rack-Mount Ear Installation 8-10
Figure 9.2.2: Analog-To-Digital Flow Diagram 9-4
Figure 9.4.1: A Contour with Multiple Circular Moves 9-16
Figure 9.4.2: A Contour with Multiple Linear and Circular Moves 9-17
Figure 9.4.3: Block Diagram of Via Point Data Handling by Command Processor 9-18
Figure 9.4.4: Block Diagram of Via Point Data Handling by Trajectory Generator 9-18
Figure 9.6.2: Block Diagram of Closed Loop Stepper Motor Positioning 9-22
Figure 9.8.2: Timing Diagram of a TTL Pulse Generation Synchronized to Position Crossing 9-28
Figure 9.9.2: Timing Diagram of Position Capture Synchronized to External Input Trigger ... 9-31
Figure B.2.1: Rear Power Line Panel Fuse Replacement B-4
Figure B.2.2: Rear Power Line Board Removal B-5
Figure B.2.3: Power Supply Board Removal B-6

Figure B.2.4: Power Supply Board Fuse Replacement... B-6
Figure C.1.1: Main I/O (100-Pin) Connector Orientation.....C-1
Figure C.1.2: One-Hundred to Sixty-Eight Pin Cable Connector Orientation.....C-2
Figure C.1.3: JP2/JP4/JP5 Connector Orientation.....C-2
Figure C.2.1: UniDrive Controller Input Connector Orientation.....C-21
Figure C.2.2: Driver Card Connector Orientation.....C-22
Figure C.3.1: Terminal Block Board Connector Orientation.....C-27
Figure E.1.1: Add/Remove Programs Properties Screen E-2
Figure E.1.2: Select Uninstall Method Screen.....E-3
Figure E.1.3:Perform Uninstall Screen E-3
Figure E.1.4: ESP6000 Setup Menu E-4
Figure E.1.5: Update Firmware Message Screen.....E-4
Figure E.1.6: Open Screen.....E-5
Figure E.1.7: Firmware Update Screen.....E-5
Figure E.1.8: ESP6000 Initialization Screen.....E-6
Figure F-1: Configuration Logic F-2

List of Tables

Table 2.3.1: ESP6000 Controls And Indicators.....2-3
Table 2.3.2: UniDrive6000 Controls And Indicators.....2-4
Table 4.1.1: Stage (Motor) Type Settings.....4-2
Table 4.1.2: Stage (Motor) Trajectory Settings.....4-2
Table 5.3.3.1: Software Versions 5-3
Table 5.4.2.1: API Function Categories 5-6
Table 5.5.3: Binary Commands vs. ASCII Commands.....5-9
Table 5.5.3.1: Command List By Category 5-16
Table 5.5.3.2: Command List – Alphabetical.....5-21
Table 5.6.1: Commands.....5-210
Table 5.6.5.4: BoardUsed by an Application.....5-360
Table 7.2.1: Servo Parameter Functions 7-5
Table 8.1: Optional Equipment.....8-1
Table 8.1.1: Terminal Block Board Functions.....8-3
Table 8.1.2: Analog I/O Cable Connections 8-4
Table 8.1.3: Digital I/O Cable Connections 8-5
Table 8.1.4: Auxiliary I/O Connections 8-6
Table 9.2.2.1: Acquisition Tray.....9-4
Table 9.2.2.2: An Example of Analog Data Acquisition ..9-5
Table 9.2.3.1: An Example of Trace Variable Data Acquisition.....9-6
Table 9.2.3.2: Data Acquisition Commands (ASCII).....9-7
Table 9.2.3.3: Data Acquisition Commands (Binary).....9-7
Table 9.3.1: PCI Design Goals 9-11
Table 9.5.2:Slave to a Different Stage 9-20
Table 9.5.3: Slave to a Trackball 9-20
Table 9.5.4: Slave to a Joystick.....9-21
Table 9.6.2.1: An Example of Closed Loop Stepper Motor Positioning Setup.....9-23
Table 9.6.2.2: Closed Loop Stepper Positioning Commands 9-23
Table 9.7.4: Commands to Synchronize Motion to External Events.....9-26
Table 9.8.3: Commands to Synchronize External Events To Axis Position.....9-29
Table 9.9.3: Commands to Synchronize Position Capture To External Inputs 9-32
Table A.1: Error Messages A-1
Table B.1.1: Trouble-Shooting Guide B-2
Table C.1.1: Main I/O Connector Pin-Outs..... C-3
Table C.1.2: Motor/Driver Interface (100-to-68) Cable Connector Pin-Outs..... C-8
Table C.1.3: Digital Connector Pin-Outs C-11

Table C.1.4: Auxiliary Connector Pin-Outs *C-13*
Table C.1.5: Analog Connector Pin-Outs *C-19*
Table C.2.1: Driver Card Connector Pin-Outs *C-22*
Table C.3.1: MD4 Connector Pin-Outs *C-27*
Table C.3.2: Eighteen-Lead Upper Connector Pin-Outs *C-30*
Table C.3.3: Eighteen-Lead Lower Connector Pin-Outs *C-31*
Table C.3.4: Optional Power Supply Connector Pin-Outs *C-33*
Table C.3.5: Nine-Pin Connector Pin-Outs *C-34*
Table C.3.6: Jumpers *C-35*
Table D.1: Binary Conversion Table *D-1*
Table H-1: Technical Customer Support Contacts *H-1*

Section 1 – Introduction

1.1 Scope

This manual provides descriptions and operating procedures for the Enhanced System Performance (ESP) motion system, consisting of the ESP6000 controller card, UniDrive6000 universal motor driver, and various stages.

Safety considerations, conventions and definitions, and a system overview are provided in Section 1, Introduction.

Procedures for unpacking the equipment, hardware and software requirements, descriptions of controls and indicators, and setup procedures are provided in Section 2, System Setup.

Instructions for configuring and powering up the UniDrive and stage motors, for home and jog motions, and for system shut-down are provided in Section 3, Quick Start.

Features and operation of the Windows motion and tuning utilities are described in Section 4.

Newport-provided commands (BINARY and ASCII), language-specific information, and error-handling procedures are provided in Section 5, Programming.

An overview of motion parameters and equipment is provided in Section 6, Motion Control Tutorial.

Servo tuning principles and procedures are given in Section 7.

Procedures for ordering, installing, and using optional equipment are provided in Section 8.

The motion control software, data acquisition, Peripheral Component Interconnect (PCI) bus structure, grouping, and Slaving a Stage to Trackball, Joystick, or a Different Stage, are described in Section 9, Advanced Capabilities.

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
- 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 power cord in the following circumstances:

- If the 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 UniDrive6000 except to replace the rear power line panel fuses and power supply board fuse (see Appendix B, Trouble Shooting). There are no user-serviceable parts inside the UniDrive.
- Do not make modifications or parts substitutions.
- Return equipment to Newport Corporation for service or repair.
- Do not touch, directly or with other objects, live circuits inside the unit.
- Do not operate the unit in an explosive atmosphere.
- 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 (>90% humidity).

WARNING

All attachment plug receptacles in the vicinity of this unit 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, they should contact an electrician to perform the necessary alterations to assure that the green (green-yellow) wire is attached to earth ground.

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.

WARNING

Opening or removing covers will expose the user to hazardous voltages. Observe the following precautions:

- **Turn power OFF and unplug the unit from its power source;**
- **Disconnect all cables;**
- **Remove jewelry from hands and wrists;**
- **Use insulated hand tools only;**
- **Maintain grounding by wearing a wrist strap attached to instrument chassis**.

1.3 Conventions And Definitions

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

1.3.1 Definitions and Symbols

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

WARNING

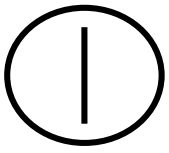
Calls attention to a procedure, practice or condition which, if not correctly performed or adhered to, could result in injury or death.

CAUTION

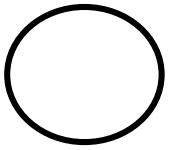
Calls attention to a procedure, practice, or condition which, if not correctly performed or adhered to, could result in damage to equipment.

NOTE

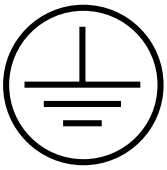
Calls attention to a procedure, practice, or condition that is considered important to remember in the context.



This symbol indicates the principal on/off switch is in the ON position.



This symbol indicates the principal on/off switch is in the OFF position.



A terminal that must be connected to earth ground prior to making any other connections to the equipment.

1.3.2 Terminology

The following is a brief description of the terms specific to motion control and the ESP6000 controller card and UniDrive6000 universal motor driver equipment.

API – Application Programmer Interface

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

DLL – Dynamic Link Library

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

ESP – Enhanced System Performance motion system comprised of ESP6000 controller card, UniDrive universal motor driver, and compatible stage(s). ESP is synonymous with a plug-and-play motion system.

ESP6000 – the ESP6000 controller card

ESP-compatible – refers to Newport Corporation stage with its own firmware-based configuration parameters. Newport stages or other stages without this feature are referred to as being not ESP-compatible and must be uniquely 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

Motion device – electro-mechanical equipment. Used interchangeably with stage and positioner

Move – a motion to a destination, initiated manually

Origin – used interchangeably with home

PCI – Peripheral Component Interconnect (type of personal computer bus)

PID – a closed loop algorithm using proportional, integral, and derivative gain factors

Positioner – used interchangeably with stage and motion device

Stage – used interchangeably with motion device and positioner

UniDrive6000 – the universal motor driver used with the ESP6000 controller card.

1.4 System Overview

The Enhanced System Performance (ESP) architecture consists of the ESP6000 controller card, UniDrive6000 universal motor driver, and ESP-compatible stages. The ESP6000 controller card (See **Figure 1.4.1**) is designed for convenient installation in the user's own PC.

The ESP6000's Windows-based setup utility provides a full range of functions for configuring and operating from one to six axes.

The system is designed to operate with Newport Corporation's ESP-compatible stages, but can be configured to function with other stages also. A typical PC-based ESP6000 system configuration with the UniDrive6000 and one stage is shown in **Figure 1.4.2.**

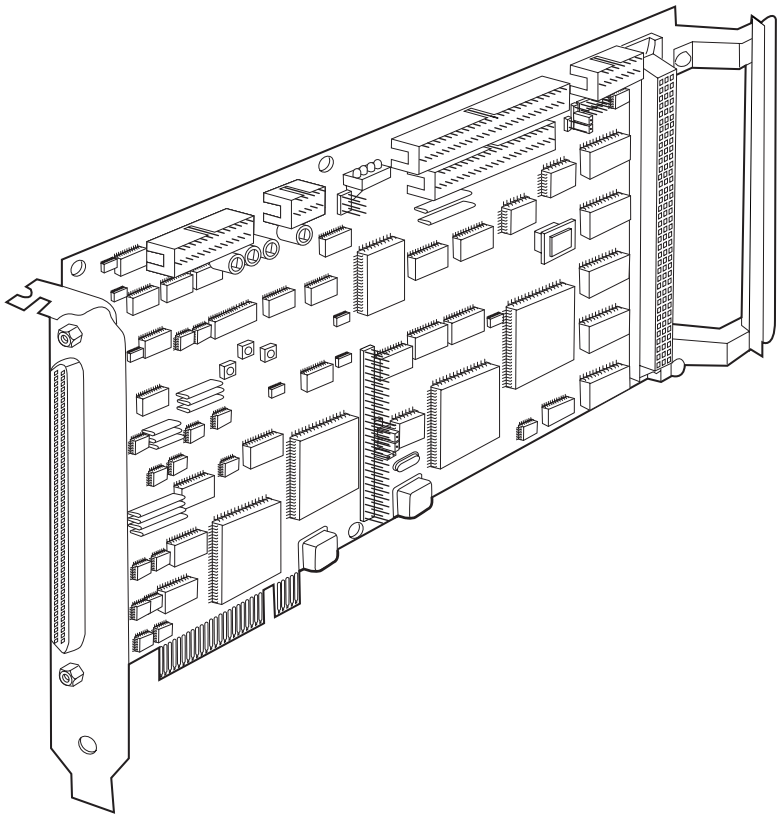


Figure 1.4.1: ESP6000 Controller Card

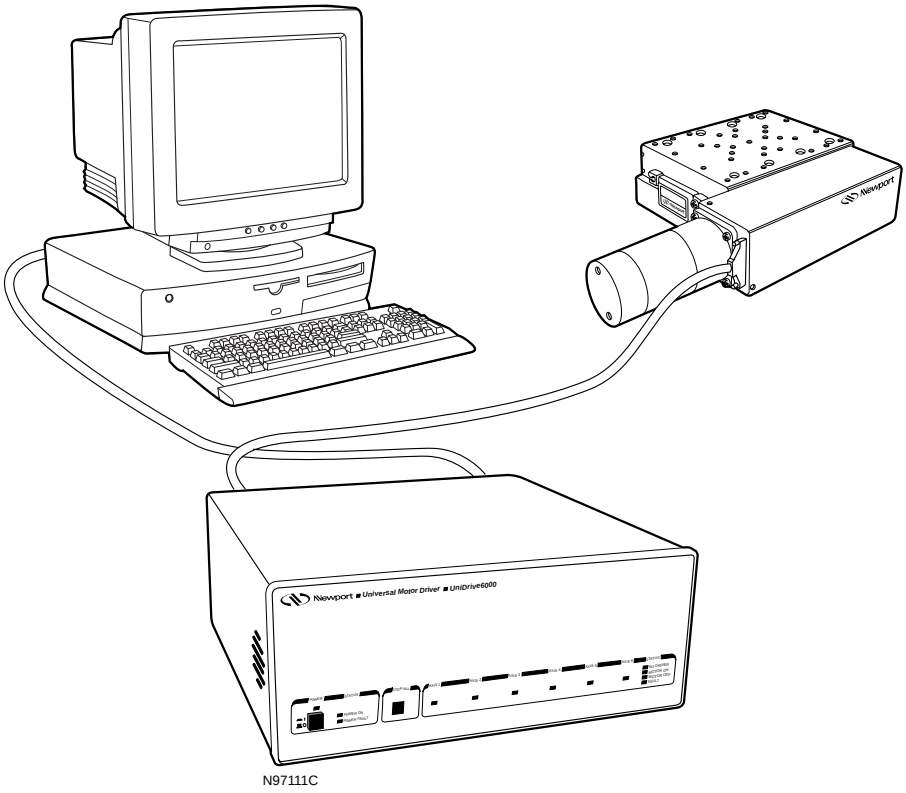


Figure 1.4.2: ESP Configuration

1.4.1 Features

Many advanced features make ESP6000 the preferred system for precision motion applications:

- Combined data acquisition and motion control
- DSP based 6-axis stepper / DC servo control
- PCI 'plug and play' host interface
- Newport stage ESP 'plug and play' compatible
- Trapezoidal and S-curve velocity trajectory profile
- Synchronized circular / linear interpolation motion
- Master / Slaving: motorized positioners slaved to other motorized positioners, trackball, or joystick
- Backlash and linear error compensation
- Digital PID-FF (feed-forward) servo loop
- Closed-loop stepper control
- 24 user programmable digital I/O Opto22™ compatible
- Instrument grade 8 channel, 16-bit A-to-D input
- Flash non-volatile memory technology

- Powerful motion programming capabilities in Visual Basic / C and LabVIEW languages with extensive set of commands
- Benchtop or rack-mount configuration for the UniDrive6000

1.4.2 Specifications

ESP6000 Controller Card

Trajectory Type:

- Non-synchronized motion
- Multi-axis synchronized motion
- S-curve velocity profile
- Trapezoidal velocity profile

DC Motor Control:

- 16-bit servo DAC resolution
- 16 MHz maximum encoder input frequency
- PID with velocity and acceleration feed-forward servo loop
- 0.4 ms digital servo cycle

Stepper Motor Control

- 2.5 MHz maximum pulse rate
- Open or closed-loop operation

Computer Interface:

- PCI bus interface

Utility Interfaces (Connectors):

- Analog I/O: 16 bit, 8 channel muxed analog inputs
- Digital I/O: 24-bit, Opto22™-compatible digital I/O
- Auxiliary I/O: 2 channel auxiliary encoder counters.

Programming:

- Visual BASIC
- Visual C/C++
- LabVIEW

Memory:

- 512KB firmware flash EPROM
- 128KB system configuration flash EPROM

Power:

- +5 Volts, 1.8 Amps (maximum)
- +12 Volts, 0.2 Amps (maximum)
- - 12 Volts, 0.2 Amps (maximum)

Physical:

- Width: 0.75"
- Height: 4.0" including face plate
- Depth: 14.0" including bracket
- Weight: 0.6 lb.

UniDrive6000 Universal Motor Driver

Number Of Motion Axes:

- 1 to 6, in any combination or order of DC and stepper motors

Stage Compatibility:

- ESP-compatible (Smart-Stage) devices, non-ESP compatible Newport stages, other stages.

DC Motor Control:

- 4 Amps, 60 Volts

Stepper Motor Control:

- 4 Amps, 60 Volts
- 250x micro-stepping resolution factor
- Full, half, and mini-step capability.

Power:

- Input voltage: 110/220V ±10%
- Frequency: 50/60Hz
- Current: 4 Amps (maximum input current)

Physical:

- Height: 7.00" including feet
- Width: 17.0"
- Depth: 17.0"
- Weight: 19.0 lb. (with 6 driver cards)

Fuses:

Location	Type
Rear Power Line Panel	4A, 250V (SLO-BLO)
Power Supply Board	3.15A, 250V

Environmental Limits

- Operating Temperature: 0° Centigrade to 40° Centigrade
- Operating Humidity: 90% non-condensing
- Storage Temperature: 60° Centigrade

Section 2 – System Set-Up

This section defines the hardware and software requirements for operation, the equipment controls and indicators, and the step-by-step procedures needed to prepare the system for use.

2.1 Unpacking

Before unpacking any components, inspect the shipping container(s) for evidence of damage. Notify the shipping carrier of damage.

CAUTION

All equipment is packaged in electrostatic material. Unpack carefully to avoid damage to equipment and packing materials.

Remove the packing list from the shipping container(s). Verify that the items listed on the packing slip are in the container(s). Refer to Appendix G, Factory Service, for reporting discrepancies.

CAUTION

The ESP6000 controller card and stages are sensitive to static electricity. Wear a properly grounded anti-static strap when handling equipment.

Further inspection of equipment should be made with the following precautions:

- Do not remove the ESP6000 controller card from the anti-static shipping bag until the user is ready to begin installation.
- Avoid touching components on the ESP6000 controller card.

All equipment is tested and inspected prior to shipment. Inspect the equipment, and refer to Appendix G, Factory Service, for reporting discrepancies.

2.2 PC Hardware and Software Requirements

PC hardware and software requirements include:

1. Personal computer with a 486DX or higher processor
2. PCI-compatible (full-length) card slot for ESP6000 board
3. Microsoft Windows 95 operating system or Microsoft Windows NT™ Workstation operating system 4.0 or Windows 2000.
4. 4 MB of memory for Windows 95 (8 MB recommended)
5. 12 MB of memory for Windows NT Workstation
6. 20 MB of hard disk space
7. VGA or higher-resolution video adapter
8. Microsoft Mouse or compatible pointing device
9. One interrupt line
10. 3-1/2" floppy disk drive (for installation only).

2.3 Equipment Controls and Indicators

Controls and Indicators for the ESP6000 controller card and UniDrive6000 as shown in **Figure 2.3.1** and **2.3.2,** and defined in **Table 2.3.1** and **2.3.2,** respectively. Refer to Appendix C for connector orientations.

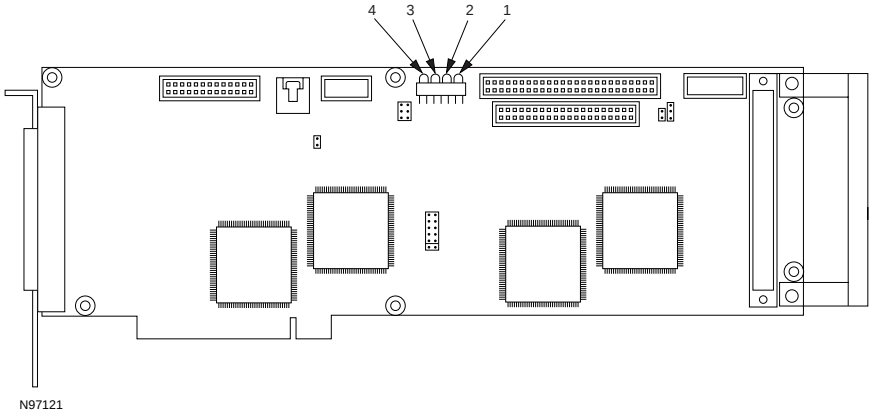


Figure 2.3.1: ESP6000 Controller Card

Reference Designation	Nomenclature	Description
1	Reset	LED state controlled by 'watchdog' timer and PCI interface. LED OFF indicates board is in reset state. Possible causes for reset state include: 1. Normal power-on reset 2. Application Programmer Interface (API) initialization command 3. +5V below operating range 4. Digital Signal Processor (DSP) not communicating with 'watchdog' timer.
2	Error	LED 2 will flash ON/OFF at approximately 0.5 second intervals as long as there is an unread error message on-board.
3	Motion	LED 3 will remain illuminated (ON) while any axis is in motion.
4	Communication	LED 4 will illuminate continuously when API commands are received.

Table 2.3.1: ESP6000 Controls and Indicators

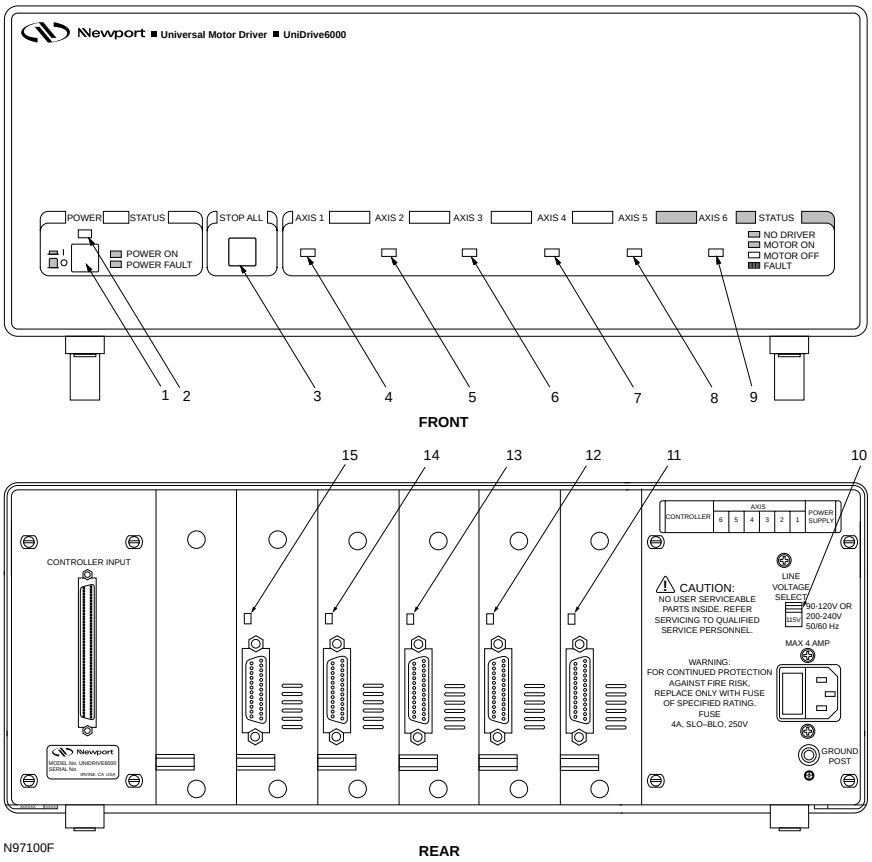


Figure 2.3.2: UniDrive6000 Front and Rear View

Item(s)	Nomenclature	Description	Normal Operating Condition/Position
1	Power on/off switch	Turns system power ON ┤ OFF O	In (depressed)
2	Power status LED	Green = UniDrive power on Red = UniDrive power fault	Illuminated (green)
3	STOP ALL switch	Turns off motor power	Out (not depressed)
4-9	LED, axis driver card status	Not illuminated = No driver (card not physically present) Green = Motor power on Yellow = Motor off, driver card installed Red = Driver fault	___
10	LINE SELECT VOLTAGE switch	Sets UniDrive to operate at 115 (range 90-120) or 230 (range 200-240) input VAC	___
11-15	LED, axis driver card status	Same as item numbers 4-9	___

Table 2.3.2: UniDrive6000 Controls And Indicators

2.4 Installation and Connection

2.4.1 Installing the ESP6000 Controller Card and Software Driver

Power down the computer. Refer to the computer user manual for procedures.

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.

WARNING

Opening or removing covers will expose the user to hazardous voltages. Observe the following precautions before proceeding:

- **Turn power OFF and unplug the unit from its power source;**
- **Disconnect all cables;**
- **Remove jewelry from hands and wrists;**
- **Use insulated hand tools only;**
- **Maintain grounding by wearing a wrist strap attached to instrument chassis.**

Remove the enclosure/housing from the computer. Removal of an enclosure from a representative PC is shown in **Figure 2.4.1.** Refer to computer user documentation or contact the computer dealer factory service department for disassembly/assembly procedures.

Locate an open PCI card slot in the chassis that is not obscured by adjacent cards. If other cards are protruding into the PCI slot space, it may be necessary to re-locate them in order to install the ESP6000 card. PCI spaces are identifiable by a double-row female edge connector on the PC motherboard.

Removing the retaining screw(s) on the inside of the cut-out panel for the PCI card slot, and remove the panel.

Due to varying card guide configurations of computers, it may be necessary to remove the bracket at the end of the ESP6000 card in order to install the card. Make a visual determination before attempting to continue installation.

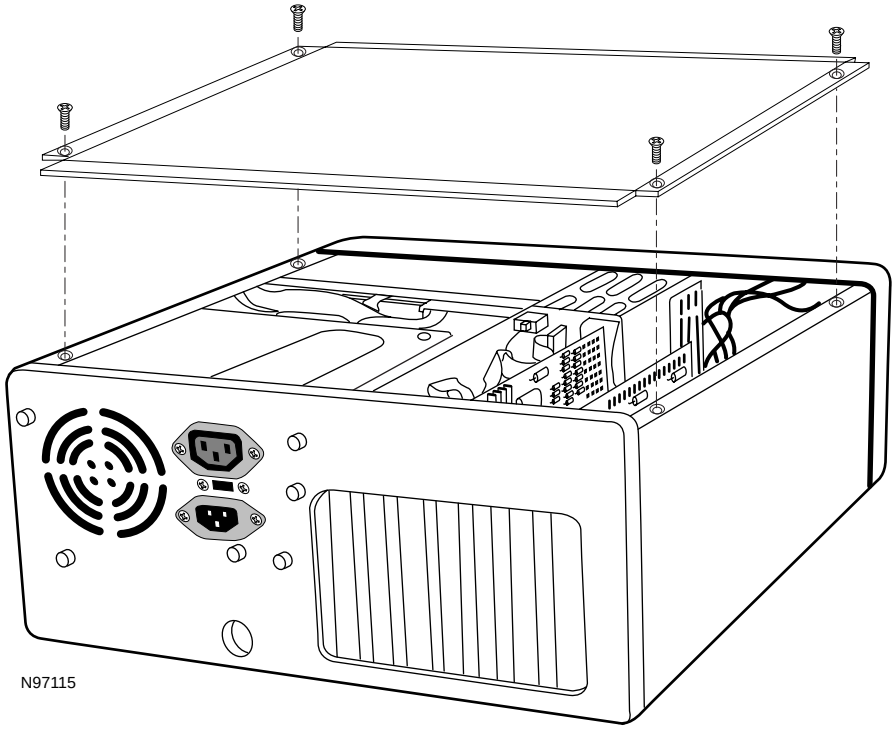


Figure 2.4.1: Enclosure Removal

Insert the 100-pin connector edge of the ESP6000 card through the open panel, and the rear of the card into the card guide at the back of the PCI card slot.

Orientation for card insertion is shown in **Figure 2.4.2.**

Push down gently until the edge connector on the bottom of the ESP6000 card mates with the connector at the bottom of the computer chassis.

Install the retaining screw into the bracket at the top of the ESP6000 card.

Re-attach the enclosure and plug in the computer AC power cord.

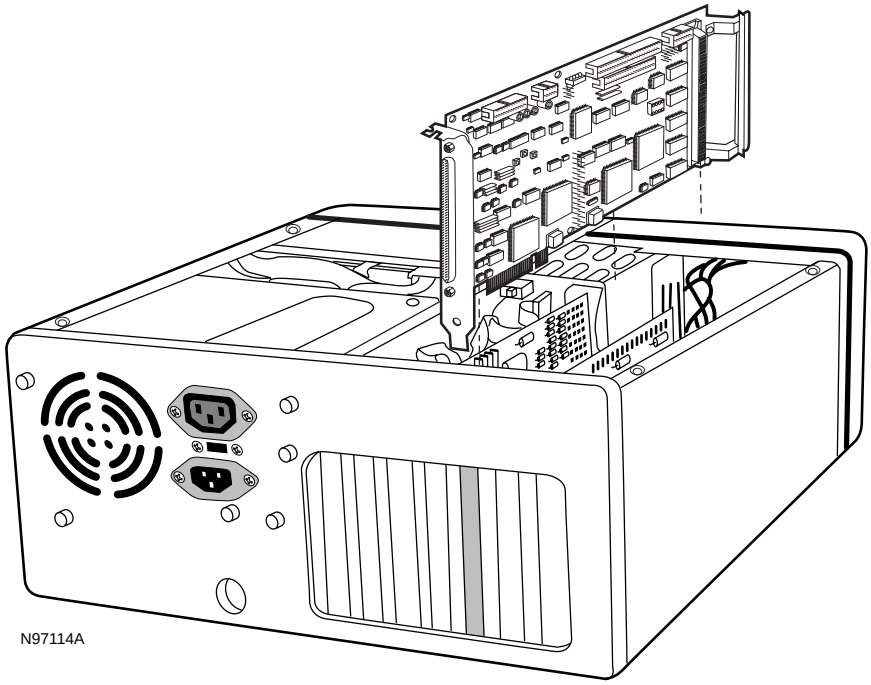


Figure 2.4.2: ESP6000 Controller Card Insertion Orientation

Turn on the computer. If the ESP6000 card is installed properly, the rear-most LED (1) at the top of the card should illuminate red continuously. Refer to Equipment Controls and Indicators, paragraph 2.3 of this section for detailed LED information.

After booting-up, a Windows 95 system will respond to the ESP6000 card installation with the following prompt (See **Figure 2.4.3**).

NEW HARDWARE FOUND

Select Which Driver You Want To Install For Your Hardware:

Windows Default Driver

Driver From Disk Provided By Hardware Manufacturer

Do Not Install A Driver (Windows will Not Prompt You Again)

Select From A List Of Alternate Drivers

OK **CANCEL** **HELP**

Figure 2.4.3:Controller Card Device Driver Prompt (Representative Screen Only)

Select **DRIVER FROM DISK PROVIDED BY HARDWARE MANUFACTURER** and press **OK.** Another Windows 95 prompt appears (See **Figure 2.4.4**).

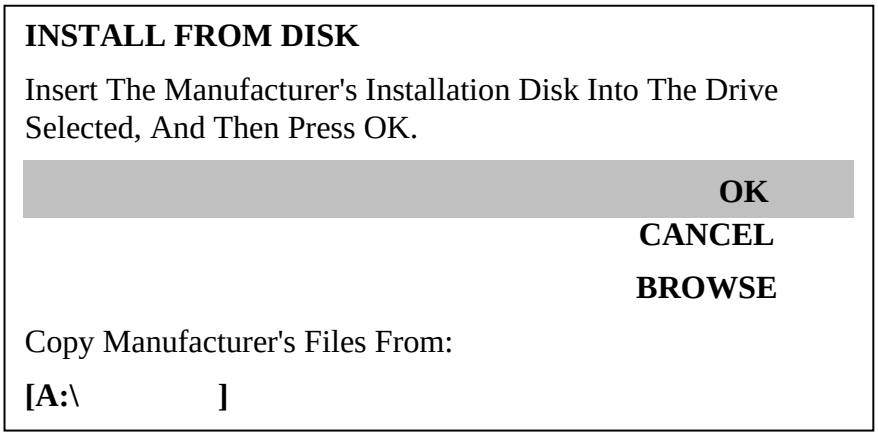


Figure 2.4.4: Install From Disk Message (Representative Screen Only)

Insert the disk labeled ESP6000 Windows 95 Device Driver into the computer floppy disk drive and select **OK.** Another Windows 95 prompt appears (See **Figure 2.4.5**).

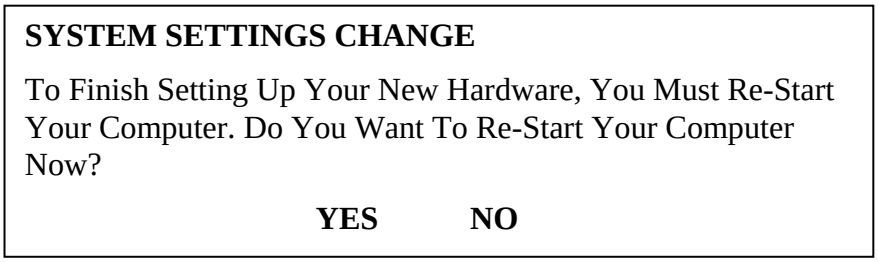


Figure 2.4.5: System Settings Change Message (Representative Screen Only)

Remove the device driver floppy disk.

Select **YES** to re-boot the system and actuate the driver.

2.4.2 Installing Windows Software

The Windows Interface Software disk set (which includes software utilities. DLL, and language libraries) can be installed any time after the ESP6000 controller card and driver software have been installed.

From the Windows 95 desk-top, press the start button and select **SETTINGS** and then **CONTROL PANEL.** The Windows 95 start-up screen appears (See **Figure 2.4.6**), followed by the Control Panel menu (See **Figure 2.4.7**).

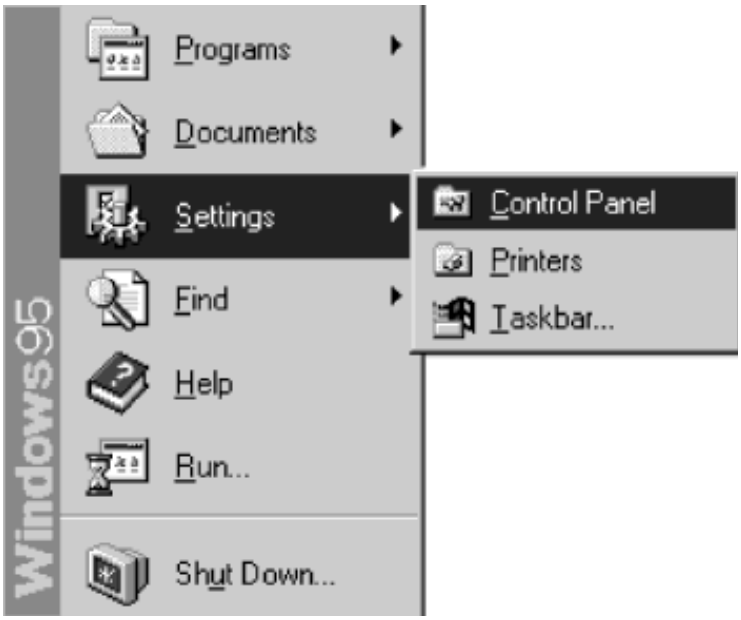


Figure 2.4.6: Windows 95 Start-Up Screen

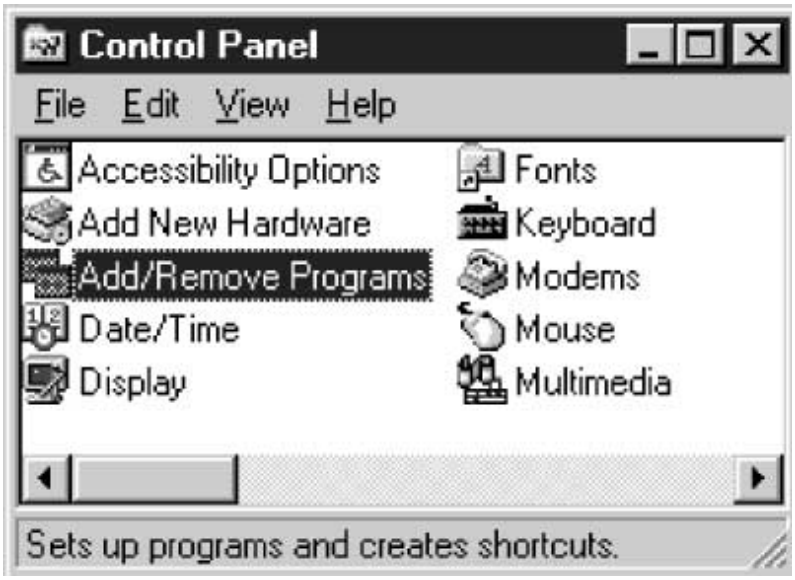


Figure 2.4.7: Control Panel Menu

Select **ADD/REMOVE PROGRAMS** from the Control Panel menu. The Add/Remove Programs Properties menu appears (See **Figure 2.4.8**).

Figure 2.4.8: Add/Remove Programs Properties Menu

Select the Install/Uninstall tab from the Add/Remove Programs Properties menu, then select **INSTALL.** The Install Program From Floppy Disk Or CD-ROM screen appears (See **Figure 2.4.9**).

Insert the disk labeled ESP6000 Windows Interface Software Disk 1 of 4 into the floppy drive and select **NEXT.** The Run Installation Program screen appears (**Figure 2.4.10**). Verify that the path appears as shown in the screen and select **FINISH.** The ESP Welcome screen appears (See **Figure 2.4.11**).

Figure 2.4.9: Install Program From Floppy Disk Or CD-ROM Screen

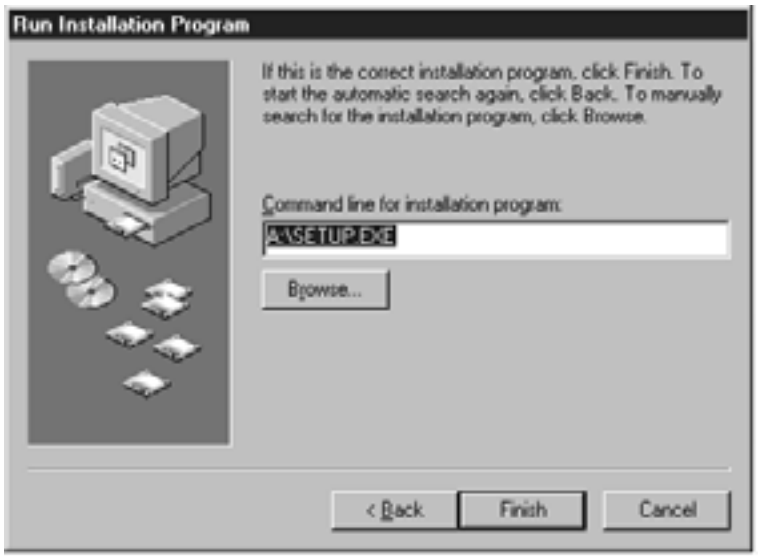


Figure 2.4.10: Run Installation Program

Figure 2.4.11: ESP Welcome Screen

Select **NEXT.** The ESP Select Destination Directory screen appears (See **Figure 2.4.12**).

Figure 2.4.12: ESP Select Destination Directory Screen

Select **NEXT.** The ESP Ready To Install screen appears (See **Figure 2.4.13**).

Figure 2.4.13: ESP Ready To Install Screen

Select **NEXT.** The Installing message screen appears (See **Figure 2.4.14**).

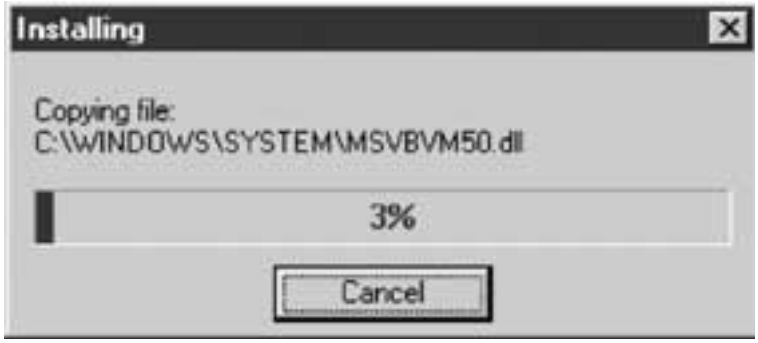


Figure 2.4.14: Installing Message Screen

When installation of the first disk is complete, the Insert New Disk message screen appears (See **Figure 2.4.15**) prompting the user to insert disk 2. Follow the prompts for disks 2, 3, and 4.

Figure 2.4.15: Insert New Disk Message Screen

After the last disk has been down-loaded, an Updating System Configuration message screen appears briefly.

When installation is complete, the ESP Installation Completed screen appears (See **Figure 2.4.16**), and the interface software is then available for use. Select FINISH to return to the Control panel menu.

Figure 2.4.16: ESP Installation Completed Screen

2.4.3 Verifying Communication Between the ESP6000 Card and the PC

The ESP 6000 software performs verification each time the system is booted-up. An ESP Initialization message screen appears that indicates the ESP6000 controller card status, the UniDrive axes configured, and ESP-compatible stages found during initialization (See **Figure 2.4.17**).

Figure 2.4.17: ESP Initialization Screen

2.4.4 Selecting The UniDrive6000 Line Voltage

Unplug the UniDrive6000 AC power cord from the power source, and disconnect it from any equipment.

Locate the line voltage selection switch at the right rear of the UniDrive (See **Figure 2.4.18**).

Use a flat-bladed screwdriver to move the switch to the upper position (115V), or to the lower position (230V), depending on the local line voltage. Input voltage ranges and frequencies are shown adjacent to the switch.

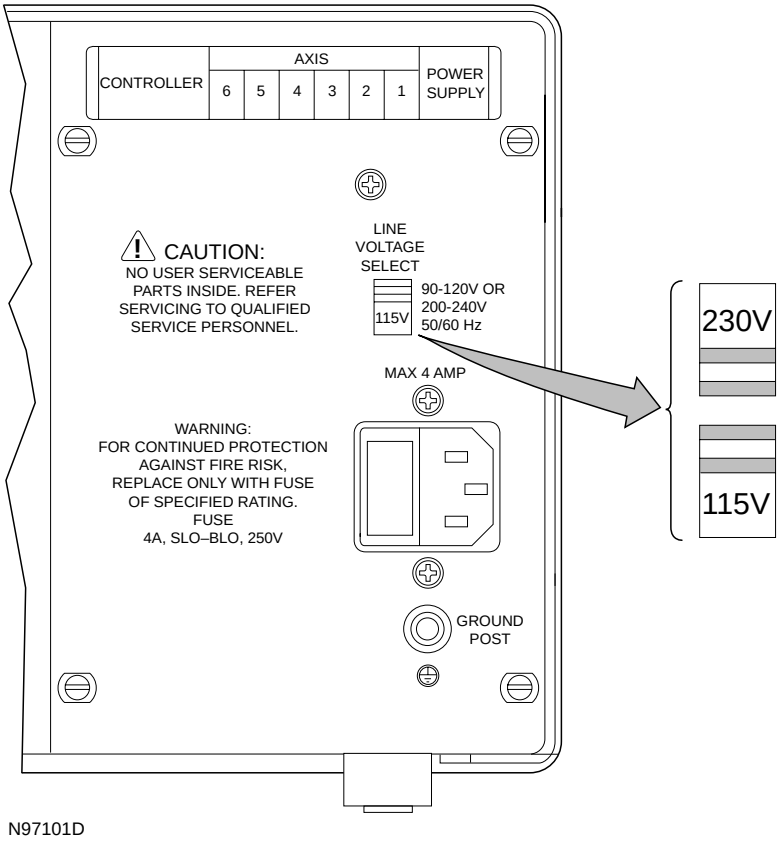


Figure 2.4.18: Line Voltage Select Switch

CAUTION

Verify that the UniDrive6000 power switch at the front of the unit is OFF before connecting the AC power cord.

CAUTION

Before applying AC power to the UniDrive6000, set the line voltage selection switch to the local AC line voltage.

2.4.5 Connecting Stages

All ESP-compatible stages are electrically and physically compatible with the UniDrive6000. ESP-compatible stages are recognizable by an ESP logo. Stages that are not ESP-compatible do not have an ESP logo. If an ESP-compatible motion system was purchased, all necessary hardware to connect the stage with the UniDrive6000 is included. The stage connects to the UniDrive6000 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

Make sure the UniDrive6000 and ESP6000 are powered off.

CAUTION

Position stage(s) on a flat, stable surface before connection to a rack-mounted UniDrive.

Carefully connect one end of the cable to the **stage** and the other end to a driver axis on the UniDrive6000 (See **Figure 2.4.19**). Secure both connectors by tightening the thumbscrews. Refer to Appendix G, Factory Service to order replacement cables (part number 52911).

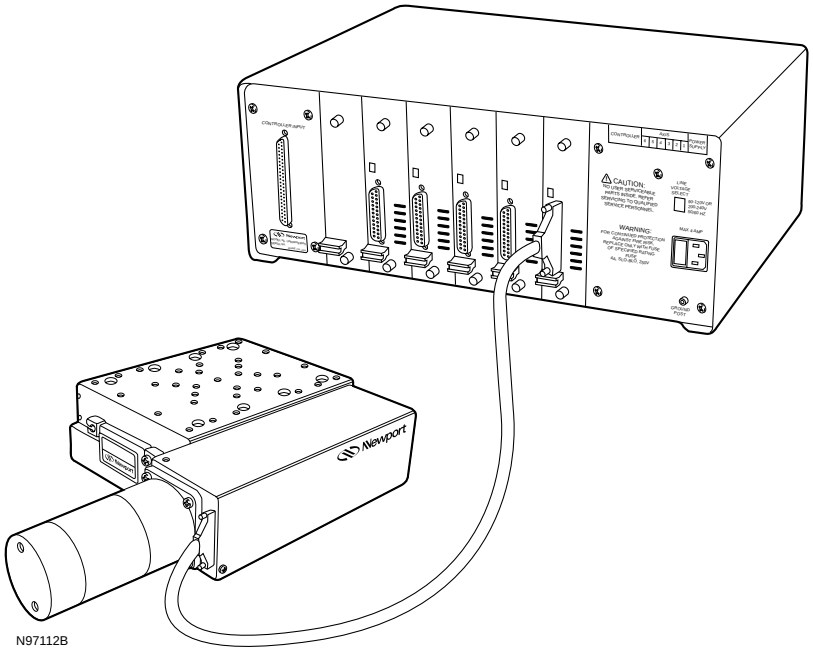


Figure 2.4.19: Stage To UniDrive Connection

2.4.6 Connecting the UniDrive6000 to the ESP6000 Controller Card

WARNING

All attachment plug receptacles in the vicinity of this unit 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 that the green (green-yellow) wire is attached to earth ground.

CAUTION

Verify proper alignment before inserting cables into connectors. Do not force.

With the UniDrive6000 and personal computer powered off, attach the supplied 100-pin cable to the connector labeled CONTROLLER INPUT at the rear of the UniDrive6000. Next attach the cable to the ESP6000 card connector at the rear of the personal computer (See **Figure 2.4.20**). Connector orientation is shown in Appendix C. Secure both connectors with the locking thumbscrews.

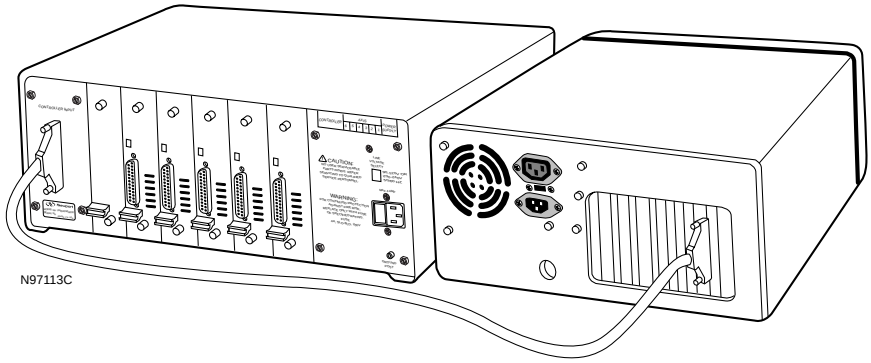


Figure 2.4.20: UniDrive To Controller Card Connection

NOTE

To ensure proper ventilation for the UniDrive, maintain a distance of at least 1" between a free-standing UniDrive and other equipment.

Section 3 – Quick Start

3.1 General Description

The following paragraphs cover procedures for operation after all equipment has been connected, and both the ESP6000 controller card driver and setup utility have been installed (See the System Setup section). Information includes how to power up the UniDrive and stage motor(s), activate home and jog motions, and to shut down the system. These procedures enable a user to verify that stages are functioning properly and to provide a limited overview of the Windows Motion Utility. Refer to the Windows Utilities and Programming sections for detailed operating procedures. Refer to the Setup UniDrive menu in the Windows Utilities section to configure the UniDrive for non-compatible (non-SmartStage) equipment operation.

3.2 Motor On

CAUTION

Place stages on a flat surface and move objects which could restrict their travel before turning on stage motors. Be prepared to turn motors off quickly if the user observes abnormal operation by pressing the red STOP ALL button on the UniDrive.

With the PC powered on and software installed, turn on the UniDrive by pressing the power switch at the front of the unit. The UniDrive is automatically configured for the appropriate motor at power-up. Select **PROGRAMS, ESP6000,** and then **ESP-UTIL** to boot-up the ESP-util.exe program.

For ESP6000-compatible stages, select **MOTION** from the ESP6000 Main Menu. The Motion drop-down menu appears (See **Figure 3.2.1**).

Figure 3.2.1: Motion Drop-Down Menu

Select **ENABLE.** The Motor Power menu appears (See **Figure 3.2.2**).

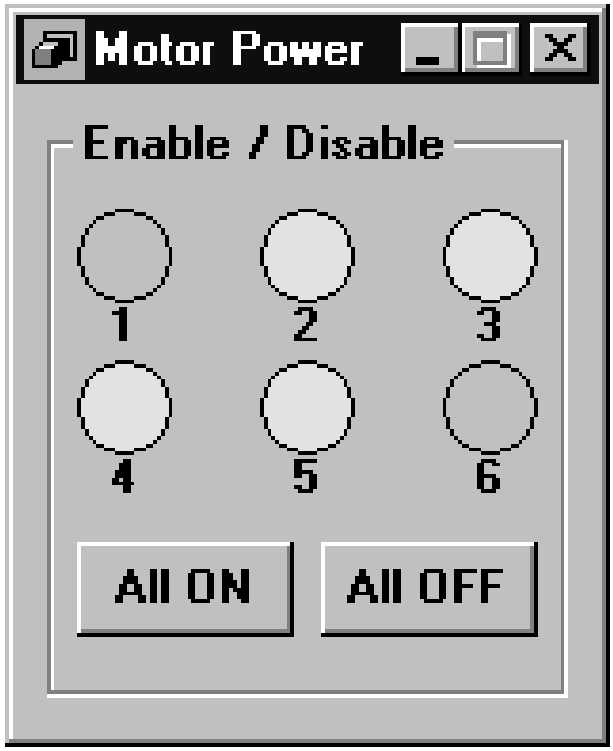


Figure 3.2.2: Motor Power Menu

Select numbered axes buttons one-by-one to turn on the motors the user has connected to the specific axes, or select **ALL ON** to enable the motors. Buttons for enabled and connected motors will illuminate green, and the axis LED on the UniDrive will turn green to indicate a motor-on condition. Click on an enabled button once to disable a motor, or select **ALL OFF** to disable all motors. Buttons for disabled motors will illuminate yellow.

If a motor type is not defined (because no ESP-compatible stages are detected) the button will not illuminate. If an axis is configured and motor type defined, but the stage is not installed for a numbered axis, the button will illuminate black. However, the user will not be able to select the axis.

3.3 Homing a Stage

From the Motion drop-down menu, select **HOME.** The Home Stages screen appears (See **Figure 3.3.1**).

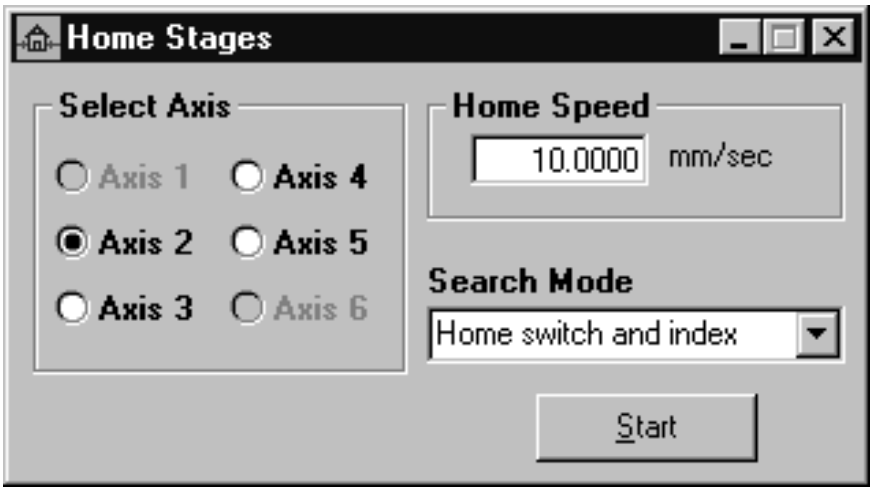


Figure 3.3.1: Home Stages Menu

Select an axis and search mode (Home Switch Only or Home Switch & Index), enter the number of units in the Home Speed input box, and Select **START** to home the stage(s).

Home each axis one by one. Verify that each stage carriage moves to its home (typically center) position.

NOTE

Enable stage(s) motor power before homing.

3.4 Jog

From the Motion drop-down menu, select **JOG** (See **Figure 3.2.1**) or select the **JOG** button from the tool bar. The **JOG** menu appears (See **Figure 3.4.1**).

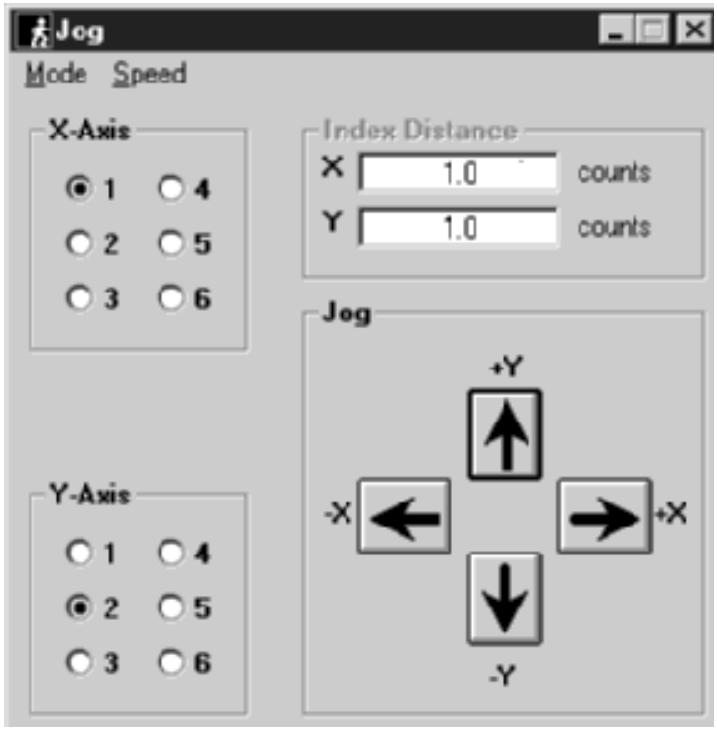


Figure 3.4.1: Jog Menu

NOTE

Enable stage(s) motor power before jogging.

Next select **SPEED** from the Jog menu (See **Figure 3.4.2**), then select the X or Y axis option from the Speed drop-down menu. The Set X/Y Speed menu appears (See **Figure 3.4.3**).

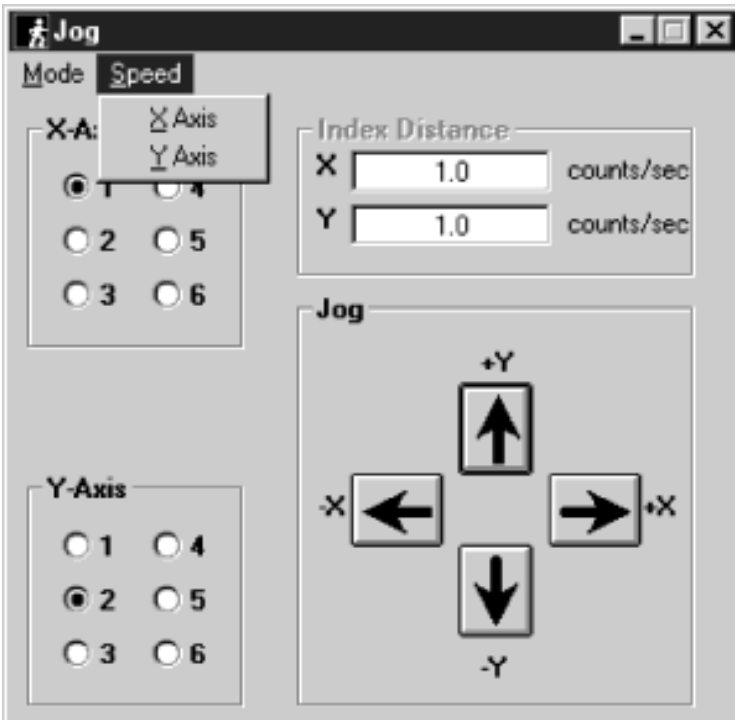


Figure 3.4.2: Speed Menu

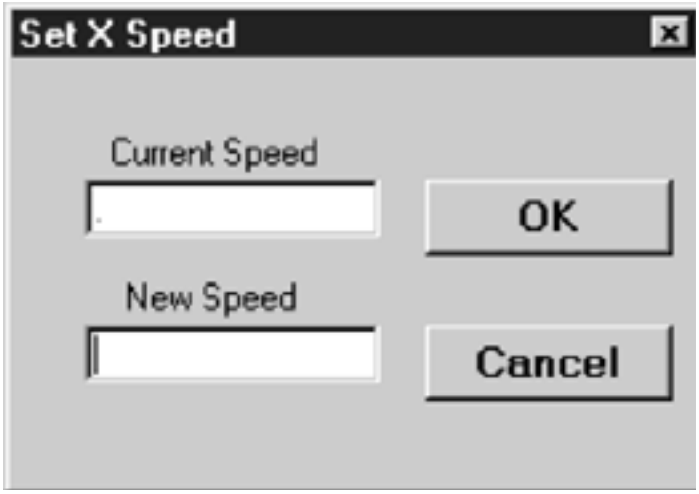


Figure 3.4.3: Set X(Y) Speed Menu

Enter a value in the New Speed text box and select **OK.** The Jog menu appears.

To execute a jog in the Free Run (unspecified motion length) mode, select a stage number (1 – 6) in the X or Y-axis sub-panels.

NOTE

The user cannot designate both the X and Y axes for the same stage axis number.

Press an X or Y arrow key to move a stage. Press and quickly release the arrow key for one-count (single) motions. Press and hold down an arrow key for continuous movement, and release the arrow key to stop movement. To execute a jog in the Index (specified motion length) mode, first select **MODE** from the Jog menu, then **INDEX** from the Mode drop-down menu. Input values for the X or Y axis in the Index Distance sub-panel, and press an arrow key to start a movement. Each depression of the arrow key will produce a movement specified by the entry in the Index Distance sub-panel.

NOTE

Stage hardware travel limits are determined by the stage itself and cannot be overridden.

3.5 System Shut-Down

To shut down the system entirely, perform the following:

Wait for the stage(s) to complete their movement and come to a halt.

Select **ENABLE** from the ESP-util Windows utility Motion menu and press **ALL OFF** to disable power.

Depress and release the UniDrive6000 power switch at the front of the unit.

Power off the personal computer.

Section 4 – Windows Utilities

4.1 Motion Utility

4.1.1 General Description

The ESP 6000 Windows utility, ESP-util.exe, is a 32-bit Window program designed to allow users to easily configure and test motion systems. The program is menu-driven and user-configurable to provide maximum flexibility for set-up and operation. A feature overview and detailed menu descriptions are provided in the following paragraphs.

4.1.2 Features

The ESP 6000 Windows motion utility, ESP-util.exe, provides users with helpful features and capabilities, including:

- Ability to exercise all six axes of motion
- Graphical interface for servo tuning
- Separate screens for jog and cycle motion, and for position status
- Set-up and monitor capability for analog-to-digital converter
- Set-up and monitor capability for digital I/O channels
- Software-driven upgrades of the ESP6000 firmware to flash EPROM (no component replacement required)
- User-defined parameters for motion set-up

4.1.3 Operation

At start up the program will load the Dynamic Link Library (DLL) and initialize the ESP6000 controller card. When the system is ready to operate a message window will identify ESP-compatible stages and motor drivers, and indicate that initialization status.

Commands to operate the motion utility are located on the Main Menu as drop-down menus or provided via tool bar buttons.

Default settings are provided for ESP-compatible stages, but settings for non-Newport stages must be individually configured from the Setup menu. The minimum setting requirement categories are listed in **Tables 4.1.1** and **4.1.2.**

Motor Type	Motor Current	PID	Micro-Step Factor	Full-Step Resolution
DC	X	X	_	_
Stepper	X	_	X	X

Table 4.1.1: Stage (Motor) Type Settings

Characteristic	Nominal	Maximum
Resolution/Units	X	_
Speed	X	X
Acceleration	X	X
Deceleration	X	X
Jerk	X	X

Table 4.1.2: Stage (Motor) Trajectory Settings

NOTE

Save configuration input on a systematic basis to ensure operating parameters are not lost.

CAUTION

Do not connect or disconnect stages while the personal computer and UniDrive are powered up.

The Main Menu is shown in **Figure 4.1.1,** and menu descriptions are provided in the following paragraphs.

Figure 4.1.1: Main Menu

4.1.3.1 Communication Initialization

When the ESP-Util application is launched, users are queried if they want the controller to be reset and UniDrive6000 amplifier, if attached, to be updated. Depending upon the user's response, the application resets the controller and updates the amplifier or just initializes communication with the controller. This process can take anywhere from 5 to 10 seconds. Once the communication is initialized, a window similar to the one in **Figure 4.1.2** appears identifying the presence of any stages. The information presented in this window may be interpreted as follows:

- **Axis-n UniDrive6000 Amplifier Detected:** A UniDrive6000 amplifier was detected on axis *#n*, where *n* can vary between 1 and 6. The amplifier was initialized with the values present at the time of initialization.
- **Axis-n XXXX Stage Detected:** A Newport stage with XXXX as stage ID was detected on axis *#n*, where *n* can vary between 1 and 6.
- **Axis-n No Stage Detected (Configured as DC Motor):** No stage was detected on axis *#n* (both positive and negative hardware limit signals were HIGH). However, this axis has DC motor type defined.
- **Axis-n No Stage Detected (Configured as Stepper Motor):** No stage was detected on axis *#n* (both positive and negative limit signals were HIGH). However, this axis has stepper motor type defined.
- **Axis-n Stage Unknown (Configured as DC Motor):** A stage with unknown stage ID was detected on axis *#n*, and this axis has DC motor type defined.

- **Axis-n Stage Unknown (Configured as Stepper Motor):** A stage with unknown stage ID was detected on axis *#n*, and this axis has stepper motor type defined.

Figure 4.1.2: Communication Initialization

4.1.3.2 File Menu

The file menu consists of a series of drop-down menus, as shown in **Figure 4.1.3.** Menu functions are described in the following paragraphs.

Figure 4.1.3: File Menu

Reset System

Select this option to reset the ESP6000 controller card and update UniDrive6000 (if attached) with existing parameters. The user will be prompted to verify the selection.

Save To Flash

Select this option to save existing controller parameters to non-volatile flash EPROM memory. The controller will automatically reload saved parameters from flash memory to working registers after a hardware reset. Please refer to Appendix F: ESP Configuration Logic to determine how the controller parameters are loaded into the working registers. The Save To File command updates controller card flash memory only. Stage memory parameters are not affected by this command.

Save To File

Select this option to save existing controller parameters to a text file. The controller parameters are saved as a series of 2-letter ASCII commands.

Restore From File

Select this option to restore controller parameters from a text file that contains the desired parameters as a series of 2-letter ASCII commands.

Convert To ASCII Command File

In order to restore controller parameters from a file that was generated by ESP-Util version 2.07 or below, the file in the old format must be converted first to the new ASCII command format. This may be accomplished by selecting this option. Users are prompted to first select a source file (file in old format). Subsequently, they are prompted to select a destination file (file in new format). Once the new file is generated, Restore From File option may be used to download the parameters to the controller.

Demo Mode

Select this option to view and exercise the utility software with no ESP6000 controller card installed.

Exit

Select this option to close this application and return to the Windows desktop.

4.1.3.3 Setup Menu

The Setup Menu consists of a series of drop-down menus, as shown in **Figure 4.1.4.** Menu functions are described in the following paragraphs.

Figure 4.1.4: Setup Menu

Motion

The Motion drop-down menu includes the following sub-menus:

Resolution:

Select **MOTION** and then **RESOLUTION** to access this sub-menu (See **Figure 4.1.5**).

Select the appropriate axis, resolution, and units of measurement. If the axis is driven by a stepper motor, enter a value for micro-step factor and full-step resolution also. Refer to the description of **SN, SU, FR** and **QS** commands in the Programming section for further details.

Axes 7 and 8 should be considered "virtual axes" only. This is because they are auxiliary encoder feedback channels with no direct stepper or DC servo motor control output association. However, they can be used in master/slave applications to indirectly control motor/slave position. Refer to the description of this feature in Advanced Capabilities section.

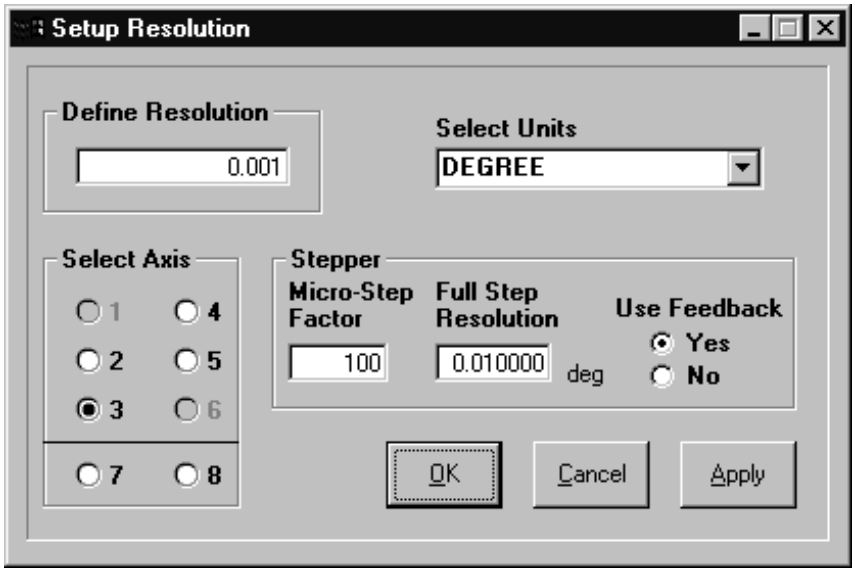


Figure 4.1.5: Setup Resolution Sub-Menu

Trajectory:

Select **MOTION** and then **TRAJECTORY** to access this sub-menu. This sub-menu includes tabs for defining trajectory and motion control. The tabs are shown in **Figures 4.1.6** and **4.1.7,** respectively.

Select the appropriate axis and enter parameters to set trajectory and motion control parameters. Refer to Servo Tuning, Section 7, for tuning guidelines before entering values. Refer to the description of **VU, VA, VB, AU, AC, AG, JK, TJ, KP, KI, KD, KS, VF, AF, ZB, DB, CL and FE** commands in the Programming section for further details.

NOTE

Default settings for ESP-compatible devices can be modified from the trajectory and motion control tabs. Select Save To Flash from the File menu to save the parameters to ESP6000 controller non-volatile flash EPROM memory.

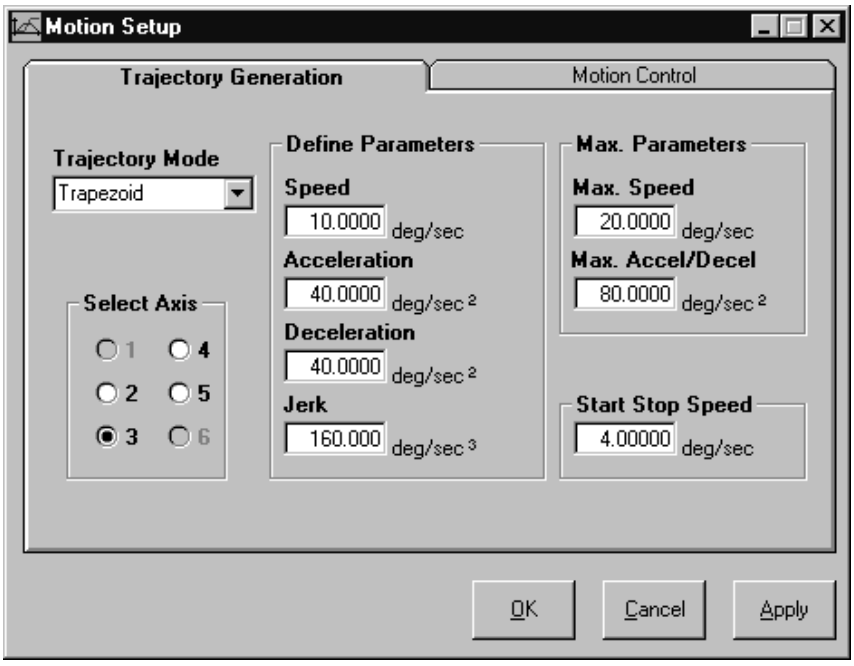


Figure 4.1.6: Setup Motion Sub-Menu: Trajectory Generation Tab

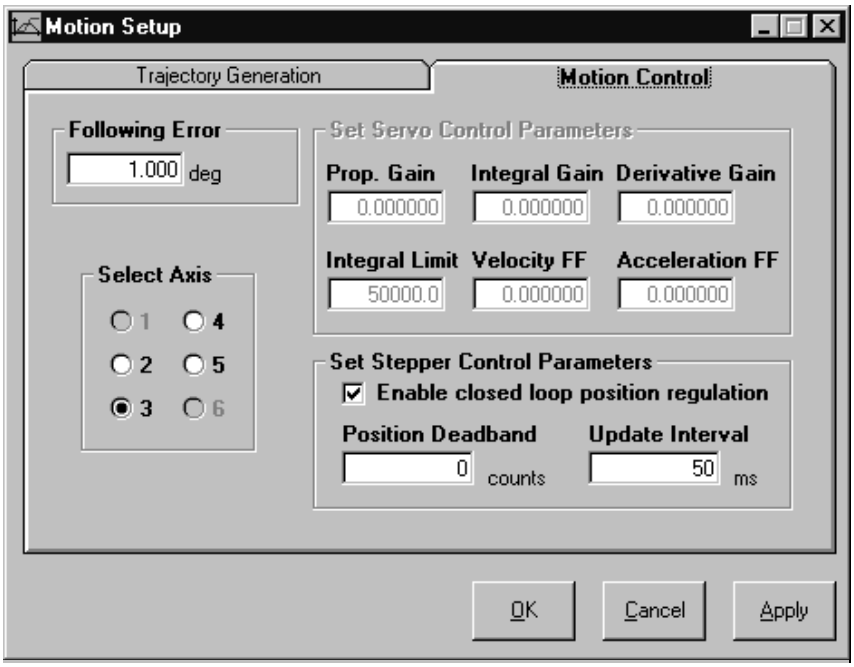


Figure 4.1.7: Setup Motion Sub-Menu: Motion Control Tab

Interpolation:

Select **MOTION** and then **INTERPOLATION** to access this sub-menu. This sub-menu includes tabs for defining group trajectory and commanding coordinated motion. The tabs are shown in **Figures 4.1.8** and **4.1.9,** respectively.

Select a desired group number and the axes that should belong to that group. Click on Unassign to remove any existing group assignments. Click on Assign button to make the group assignment. After the group assignment is made, specify group trajectory parameters such as velocity, acceleration, deceleration and jerk. Once these are defined, click on Coordinated Motion tab to make coordinated motion of the selected group. If linear interpolation is desired, specify the target positions of the individual axes and click on Move button to initiate the motion. If circular interpolation is desired, specify the center positions of the X and Y axes, the desired sweep angle and click on Move button to initiate the motion.

Refer to the description of this feature in Advanced Capabilities section for further details.

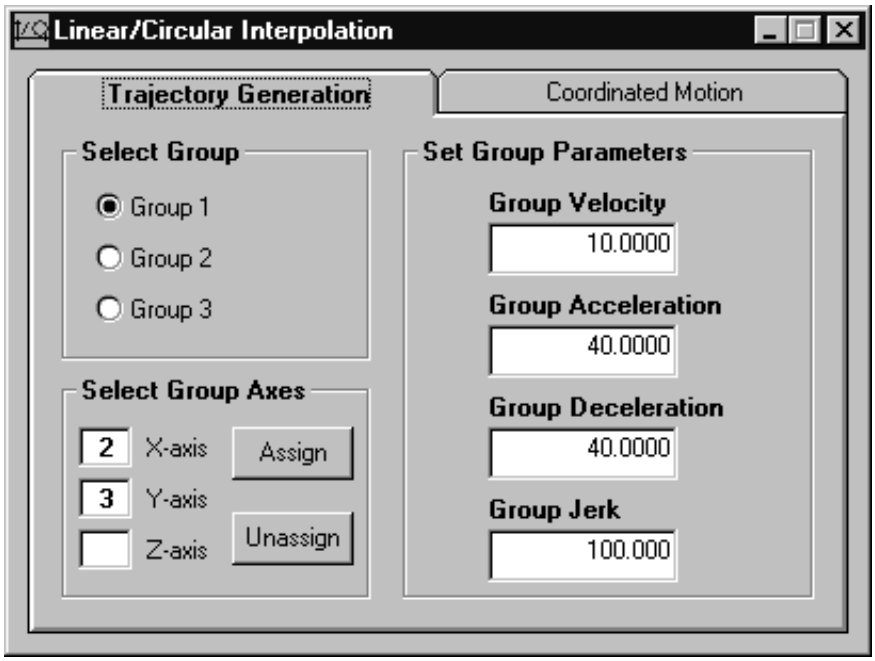


Figure 4.1.8: Setup Interpolation Sub-Menu: Trajectory Generation Tab

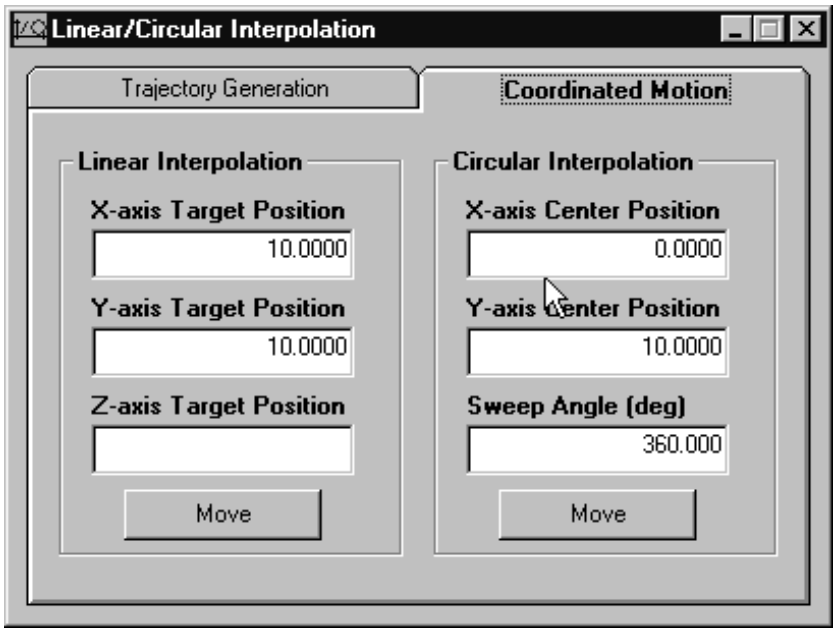


Figure 4.1.9: Setup Interpolation Sub-Menu: Coordinated Motion Tab

Master-Slaving:

Select **MOTION** and then **MASTER-SLAVING** to access this sub-menu (See **Figure 4.1.10**).

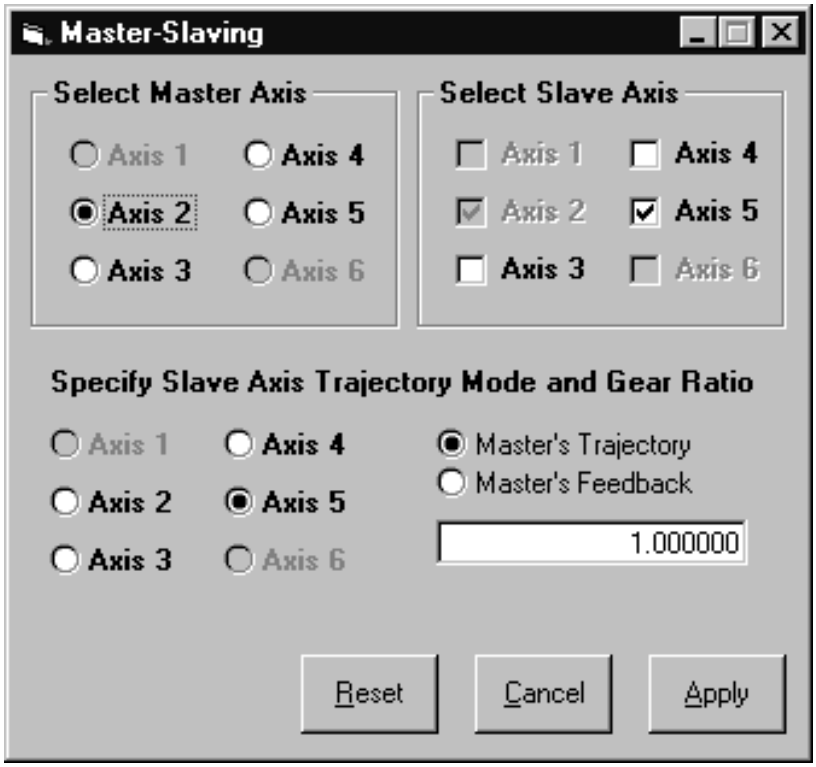


Figure 4.1.10: Setup Master-Slaving Sub-Menu

Select a master axis to which slave axes have to be assigned. Specify slave axes by checking off the slave axes. Once this assignment is done, specify the slave axis' trajectory mode and gear ratio. Click on Apply to send these changes to the controller. Click on Cancel to get back to master-slaving assignments at the time this window was opened. Click on Reset to remove all master-slave assignments.

Error Compensation:

Select **MOTION** and then **ERROR COMPENSATION** to access this sub-menu (See **Figure 4.1.11**).

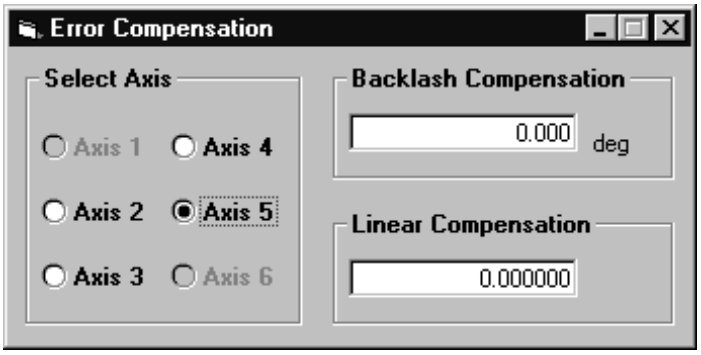


Figure 4.1.11: Setup Error Compensation Sub-Menu

Select the appropriate axis, desired backlash compensation, and linear compensation. Refer to the description of **BA** and **CO** commands in the Programming section for further details.

Faults

Select **FAULTS** to access this sub-menu (See **Figure 4.1.12** through **4.1.15**).

The Setup Faults window has four tabs to accomplish the following:

- Enable Checking: Enable/disable checking of certain faults (See **Figure 4.1.12**).
- Abort Motion: Abort motion of selected axis if a specified fault occurs (See **Figure 4.1.13**).
- Disable Motion: Disable motion of selected axis if a specified fault occurs (See **Figure 4.1.14**).
- System: Enable/disable checking of certain system faults: 100-pin cable interlock and overwriting of ESP critical settings (See **Figure 4.1.15**).

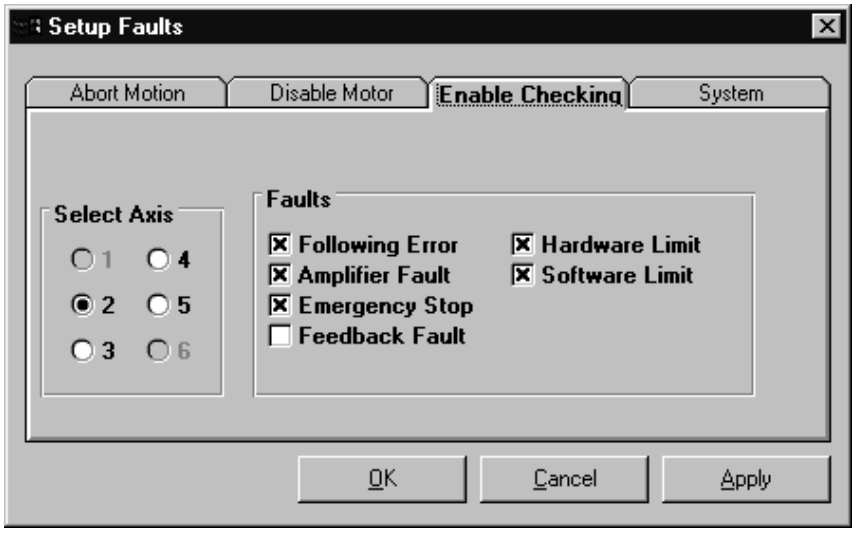


Figure 4.1.12: Setup Faults Sub-Menu: Enable Checking Tab

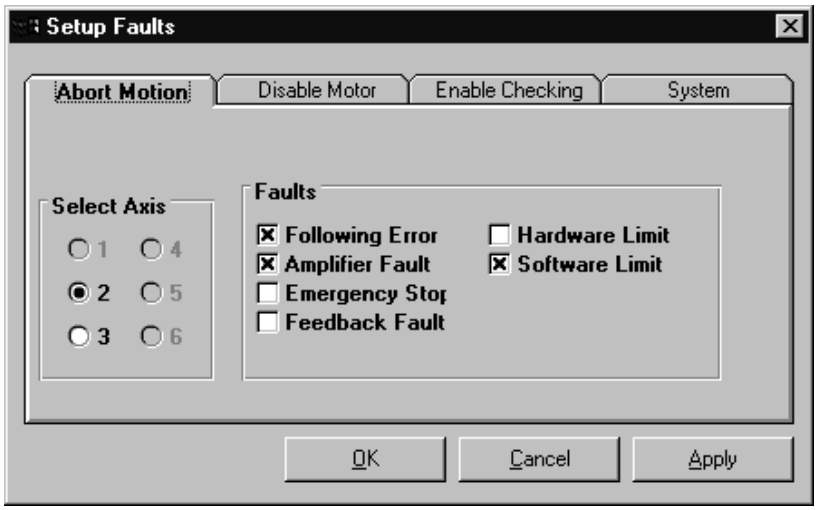


Figure 4.1.13: Setup Faults Sub-Menu: Abort Motion Tab

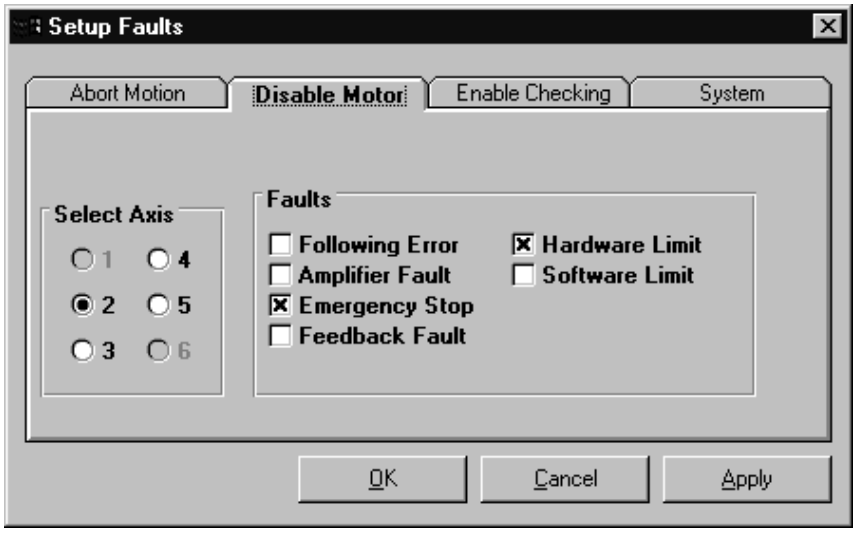


Figure 4.1.14: Setup Faults Sub-Menu: Disable Motion Tab

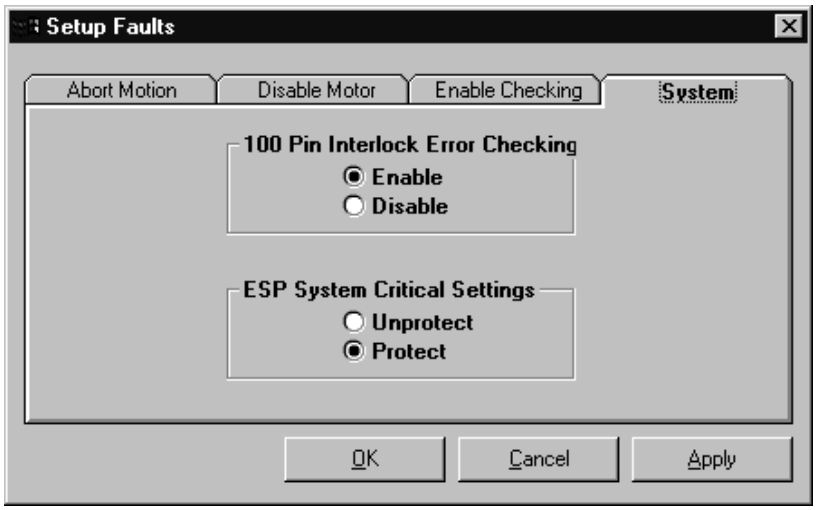


Figure 4.1.15: Setup Faults Sub-Menu: System Faults Tab

Hardware

Select **HARDWARE** to access the Setup Hardware sub-menu.

The Setup Hardware sub-menu includes tabs for defining amplifier I/O, analog I/O, digital I/O, servo DAC offset, travel limit, and feedback parameters. The tabs are shown in **Figures 4.1.16** through **4.1.21** and described in the following paragraphs.

Use the amplifier I/O tab to configure a non-ESP compatible motor amplifier. Select the axis and motor type, then designated settings. Set Amplifier Fault in the applicable Setup Faults sub-menu tab before selecting the Amplifier Fault Input option.

Figure 4.1.16: Setup Hardware: Amplifier I/O Tab

Use the Analog I/O tab to select gain and range settings for the analog-to-digital conversion. Refer to the Data Acquisition Overview material in Section 9, Advanced Capabilities, for ADC gain and range information.

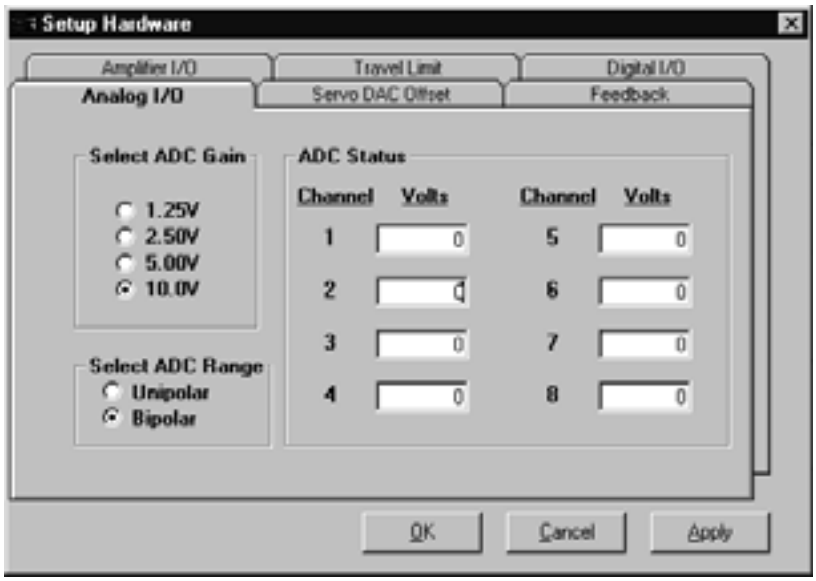


Figure 4.1.17: Setup Hardware: Analog I/O Tab

Use the Digital I/O tab to define port direction. Select bits in the Bit Settings sub-panel to designate the bits for output (red illumination indicates a HIGH logic level).

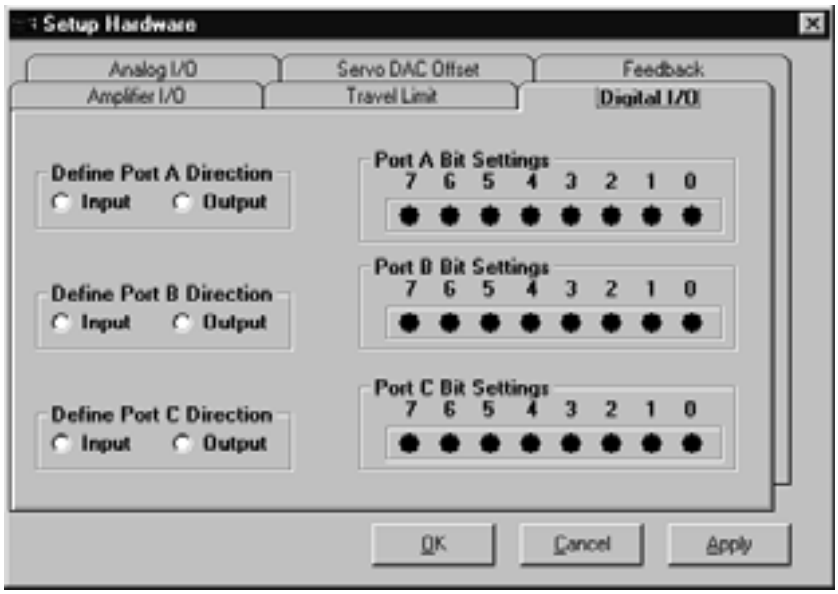


Figure 4.1.18: Setup Hardware: Digital I/O Tab

Use the Servo DAC Offset tab to designate voltage (± 1 volt maximum) offset on the servo digital-to-analog outputs for each axis, Input a value to set compensation.

Figure 4.1.19: Setup Hardware: Servo DAC Offset Tab

Use the Travel Limit tab to designate limit checks. Select the appropriate axis, enable or disable the limit checks, and set the hardware and software parameters.

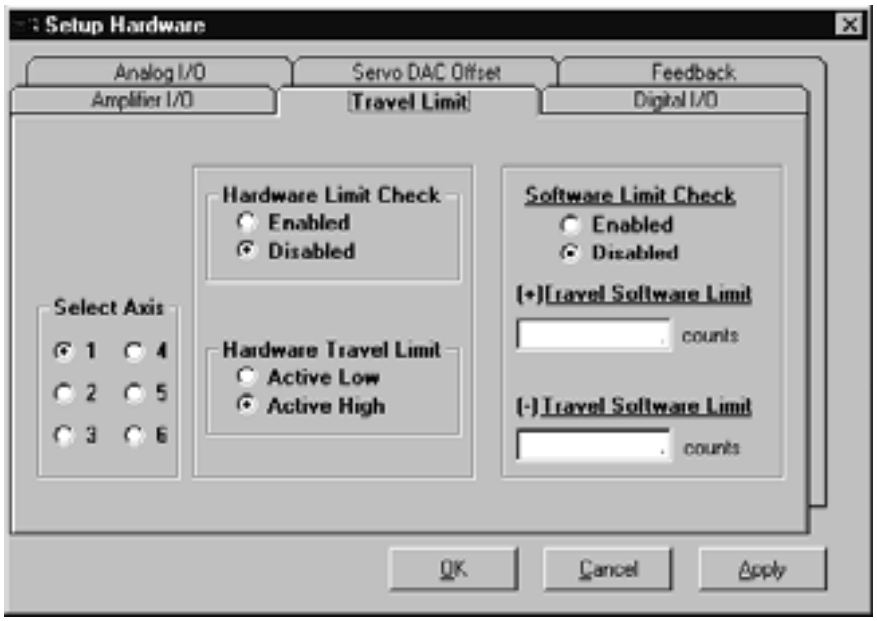


Figure 4.1.20: Setup Hardware: Travel Limit Tab

Use the Feedback tab to define encoder feedback polarity and to enable/disable use of encoder feedback for stepper motor positioning. Select the appropriate axis and make the desired choices.

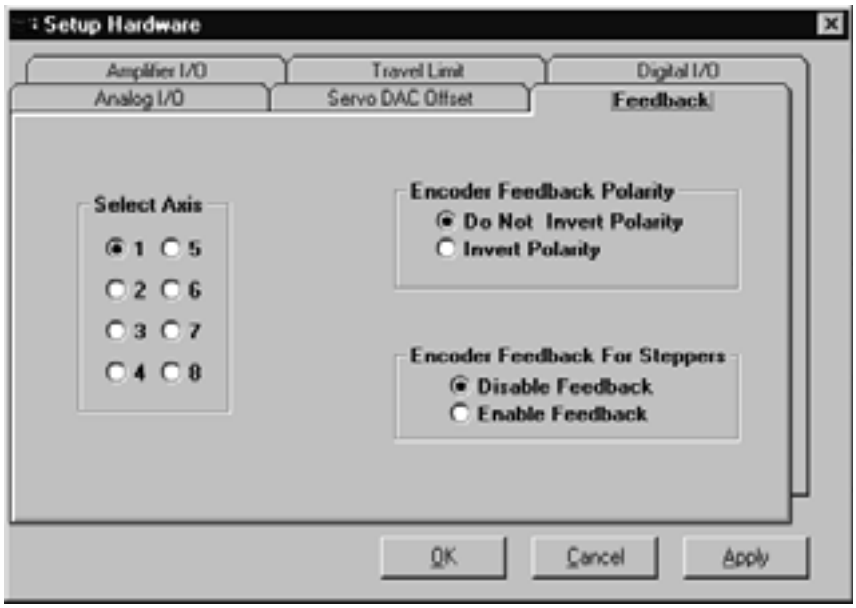


Figure 4.1.21: Setup Hardware: Feedback Tab

Firmware

The Setup menu includes a Setup Firmware sub-menu for downloading firmware to the ESP6000 controller card in the event that is becomes necessary (e.g., future upgrades).

Select **FIRMWARE** to begin the process of erasing existing firmware and downloading new firmware on the ESP6000 controller card. See Appendix E, System Upgrades, for detailed procedures.

NOTE

The ESP6000 controller card is factory equipped with firmware that is stored in non-volatile flash EPROM. Firmware downloading should be used for upgrades only.

UniDrive

The Setup menu includes a Setup UniDrive sub-menu for configuring a UniDrive6000 for operation with a non-ESP compatible stage(s).

Select **UNIDRIVE** to access the Setup UniDrive sub-menu (See **Figure 4.1.22**).

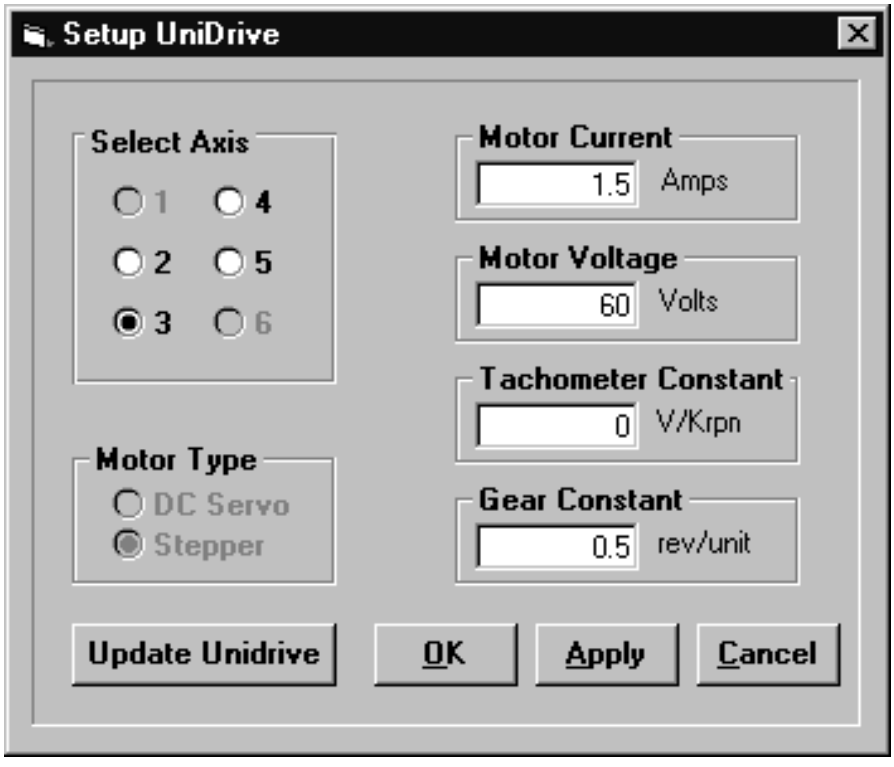


Figure 4.1.22: Setup UniDrive Sub-Menu

Select the appropriate axis and motor parameters. Verify stage current before entering a motor current value and activating a stage to prevent damage to the stage. Select **UPDATE UNIDRIVE** to transmit settings to all UniDrive axes.

4.1.3.4 Motion Menu

The Motion Menu consists of a series of drop-down menus, as shown in **Figure 4.1.23.** Menu functions are described in the following paragraphs.

Figure 4.1.23: Motion Menu

Stop

Select **STOP** to make the Stop Motion window appear (See **Figure 4.1.24**). Pressing this button halts motion movements in all axes.

Figure 4.1.24: Stop Button

Home

Select **HOME** to move the selected stage to Home reference position (See **Figure 4.1.25**).

Select the appropriate axis and enter the desired homing speed for that axis. Select a homing mode from the available choices and click on Start button to initiate home search. Refer to Section 3, Quick Start, for detailed procedures.

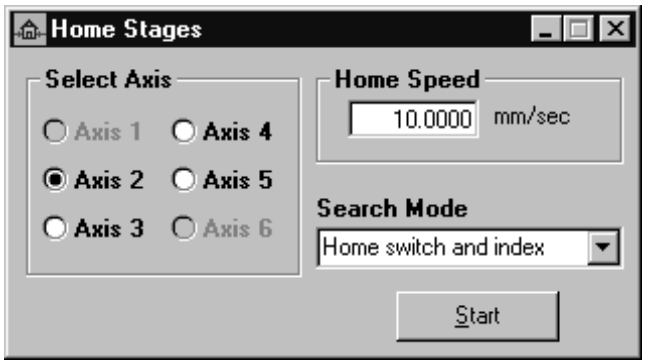


Figure 4.1.25: Home Stages Sub-Menu

Jog

Select **JOG** to initiate a jog movement (See **Figure 4.1.26**).

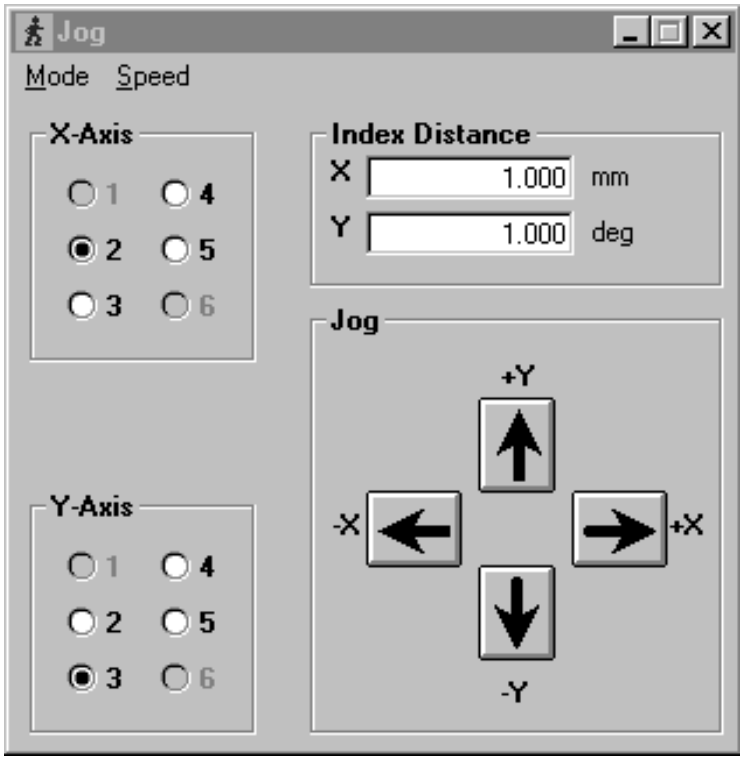


Figure 4.1.26: Jog Sub-Menu

Select the appropriate axis and the direction in which that axis must jog. Select a jog mode: index/free run. If index mode is selected, specify the index distance. Every click of the jog button will move the selected stage by the specified index distance. If free run mode is selected, the selected axis will jog as long as the jog button remains pressed. Click on speed menu to modify the jog speed of selected axis. Refer to Section 3, Quick Start, for detailed procedures.

Cycle

Select **CYCLE** to initiate point-to-point movement (See **Figure 4.1.27**).

Select all axes that should be cycled. Specify the two positions between which the axes must be cycled. Specify any dwell time between two consecutive moves. Click on Start/Stop button to initiate cycling.

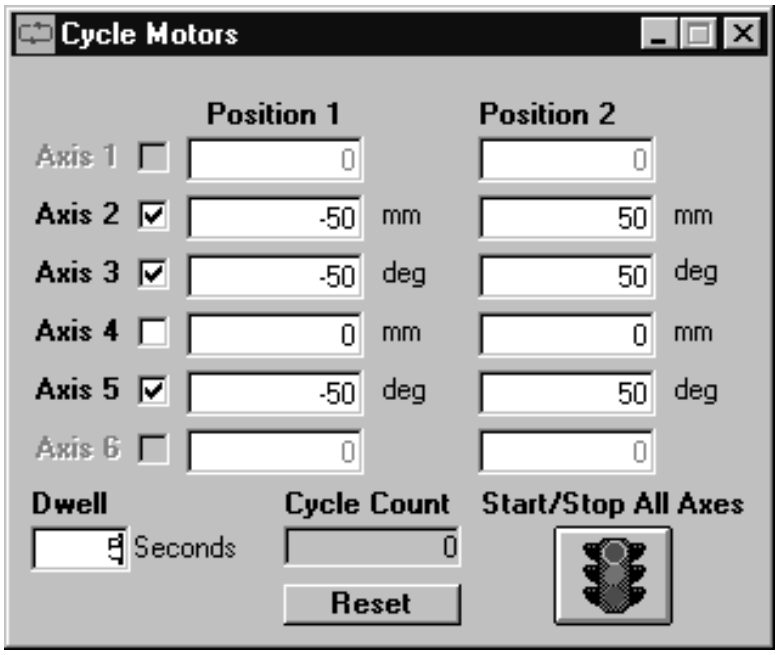


Figure 4.1.27: Cycle Motors Sub-Menu

Enable

Select **ENABLE** to power on stage motors (See **Figure 4.1.28**). Refer to Section 3, Quick Start for detailed procedures.

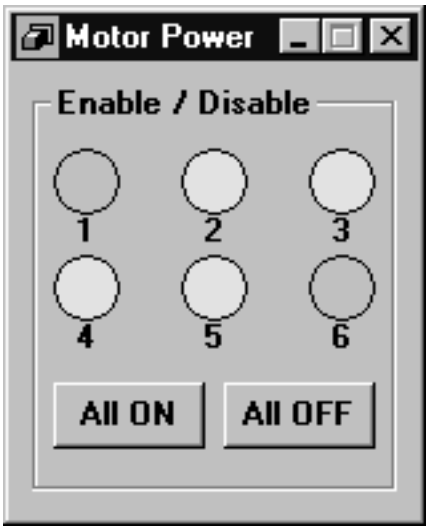


Figure 4.1.28: Motor Power Sub-Menu

4.1.3.5 Status Menu

The Status Menu consists of two drop-down menus, as shown in **Figure 4.1.29.** The Menu functions are described in the following paragraphs.

Figure 4.1.29: Status Menu

Position

Select **POSITION** to determine axis-by-axis position expressed in terms of user units. The Position Status sub-menu appears (See **Figure 4.1.30**).

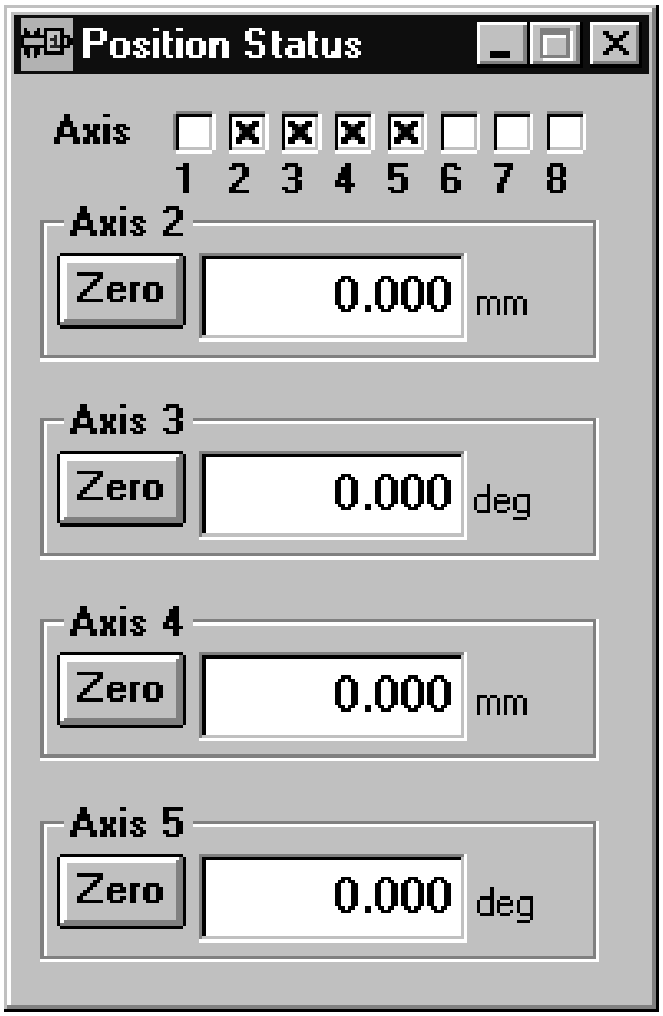


Figure 4.1.30: Position Status Sub-Menu

Select a single axis or multiple axes to display information. Axes 7 and 8 should be considered "virtual axes" only, because they are auxiliary encoder feedback channels with no direct stepper or DC servo motor control output association. However, they can be used in master/slave applications to indirectly control motor/slave position.

Hardware

Select **HARDWARE** to determine the hardware status of all axes (See **Figure 4.1.31**).

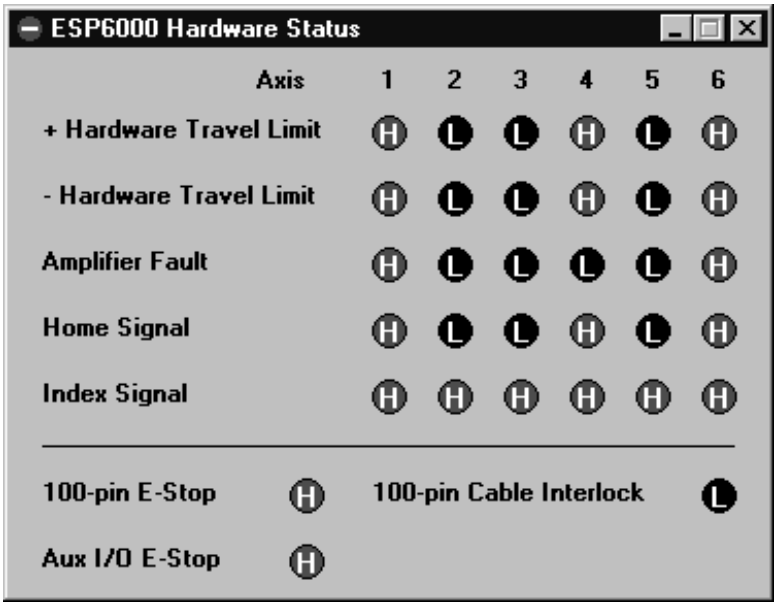


Figure 4.1.31: Hardware Status Sub-Menu

4.1.3.6 Window Menu

The Window Menu consists of two drop-down menus, as shown in **Figure 4.1.32.** Menu functions are described in the following paragraphs.

Figure 4.1.32: Window Menu

Click on Cascade sub-menu to cascade all ESP-Util windows that are open. Click on Top sub-menu to leave the all ESP-Util windows that are open to remain on the top of desktop.

4.1.3.7 Help Menu

The Help Menu consists of a series of drop-down menus, as shown in **Figure 4.1.33.** Menu functions are described in the following paragraphs.

Figure 4.1.33: Help Menu

About ESP 6000

Select **ABOUT ESP 6000** for revision information on active ESP-Util, DLL and firmware. The About ESP6000 screen appears (See **Figure 4.1.34**).

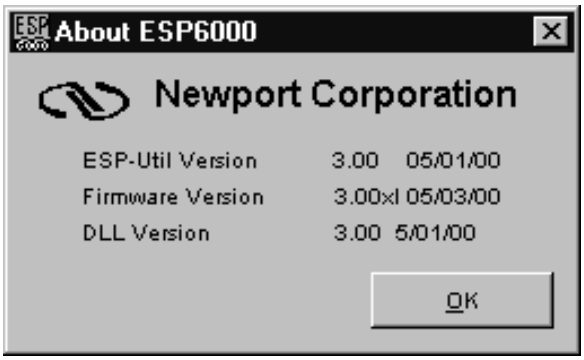


Figure 4.1.34: About ESP6000 Sub-Menu

Internet Resources

Select **INTERNET RESOURCES** to access help on ESP6000, UniDrive6000 or Motion Control Tutorial from Newport Corporation's web pages.

ESP6000 Commands

Select **ESP6000 COMMANDS** to access help on ASCII and Binary commands supported by ESP6000 motion controller.

4.2 Servo Tuning Utility

4.2.1 General Description

The ESP 6000 servo tuning utility (ESP-Tuning.exe) is a utility program developed to simplify and speed-up the servo tuning process. The program is user-configurable and consolidates input functions for efficient tuning.

4.2.2 Features

- Ability to input parameters for six axes of motion (with UniDrive 6000)
- Configurable data acquisition
- Graphical interface for value input and commands
- Plotting for comparison purposes.

4.2.3 Operation

Communication Initialization

When the ESP-Tuning application is launched, users are prompted to select a communication port (PCI – default) and press either Enter/Reset button. Press ENTER button if controller reset is not desired. Otherwise, press RESET to reset the controller and reset the UniDrive6000 amplifier, if attached. Depending upon the user's response, the application resets the controller and updates the amplifier or just initializes communication with the controller. This process can take anywhere from 5 to 10 seconds (See **Figure 4.2.1**).

Once the communication is initialized, a window similar to the one in **Figure 4.2.2** appears identifying the controller firmware version. Press the Continue button to do tuning of digital filter settings for servo motor driven stages (See **Figure 4.2.3**).

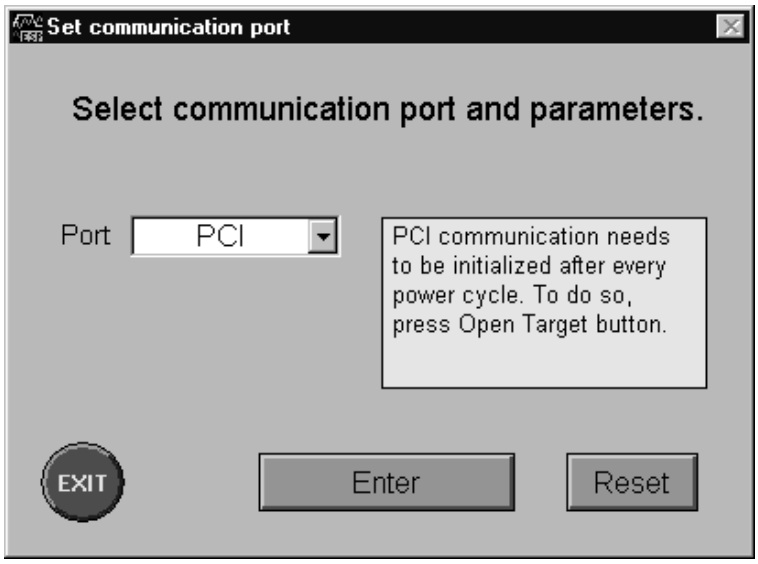


Figure 4.2.1: Select Communication Port

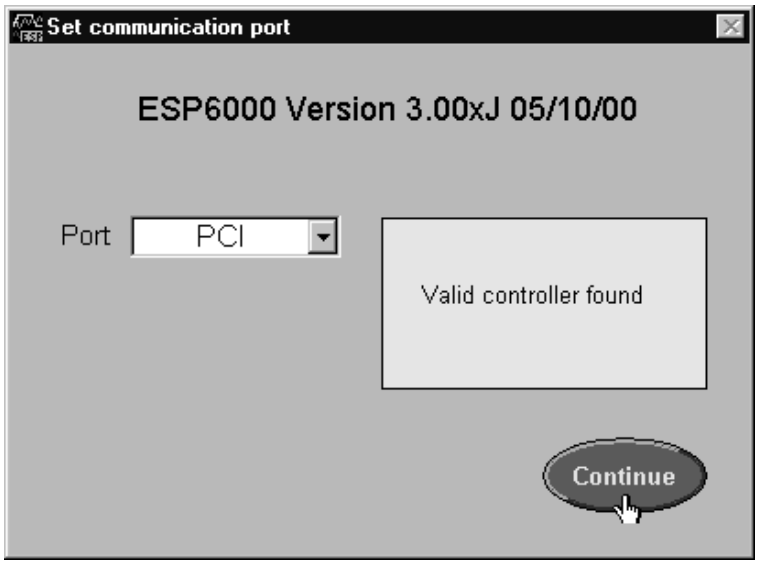


Figure 4.2.2: Controller Initialization Completed

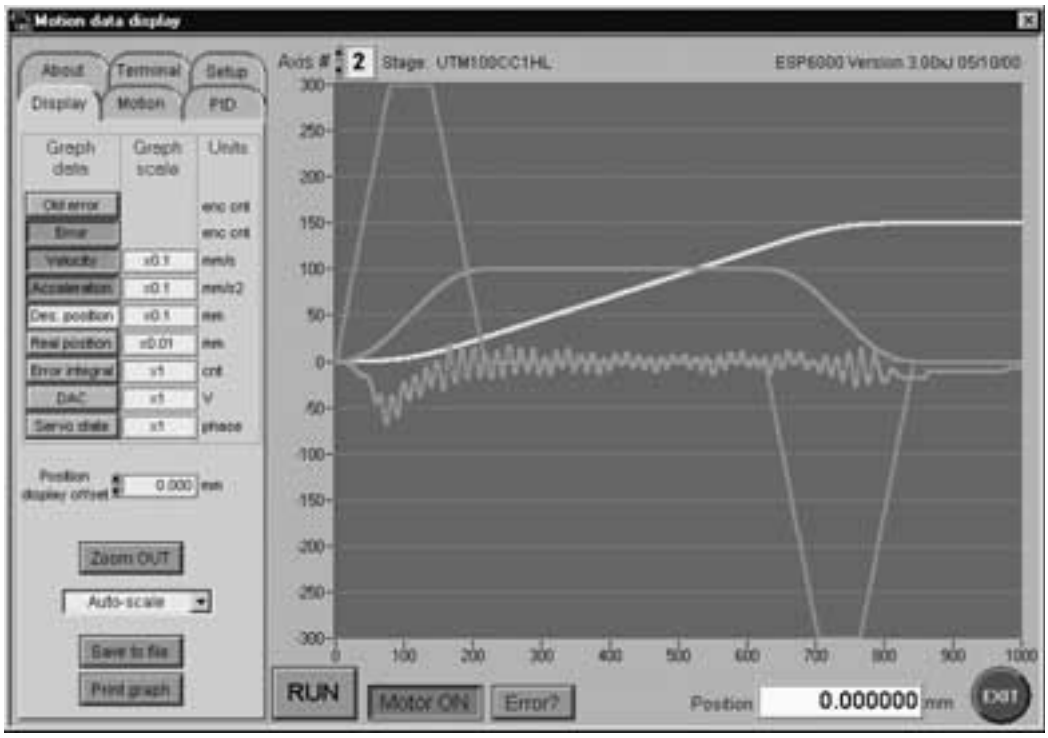


Figure 4.2.3: Servo Tuning Main Screen

There are 5 main tabs that allow users to specify motion parameters.

Display

Here user can select the parameters to be displayed in the graph, and their scale. One can also save the collected data to a file in MS-Excel format or save an image of the graph as a bitmap file or print the graph.

Motion

Here the users can specify motion parameters such as velocity, acceleration, trajectory mode, etc., in addition to the tuning data sampling interval and the number of samples to be collected (See **Figure 4.2.4**).

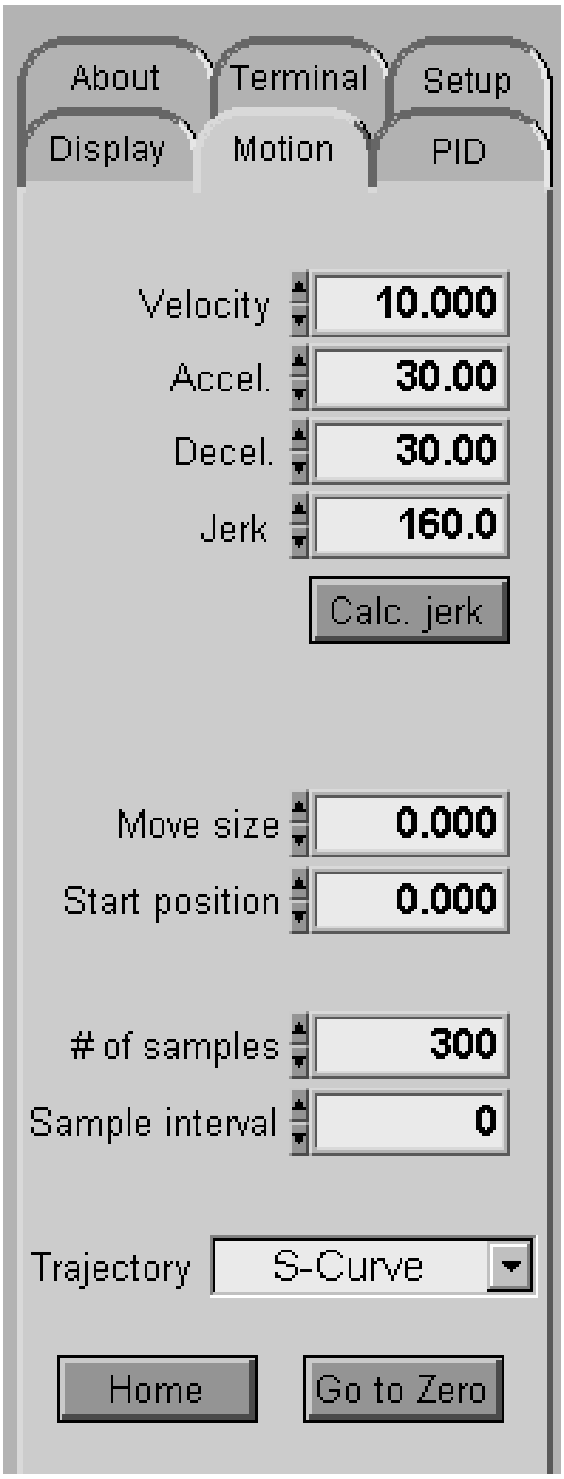


Figure 4.2.4: Servo Tuning – Motion Tab

PID

Here users can specify the digital filter – proportional, integral, derivative, velocity feedforward and acceleration feedforward – values to be used during a move. The application also keeps track of the previous four moves for performance comparison (See **Figure 4.2.5**).

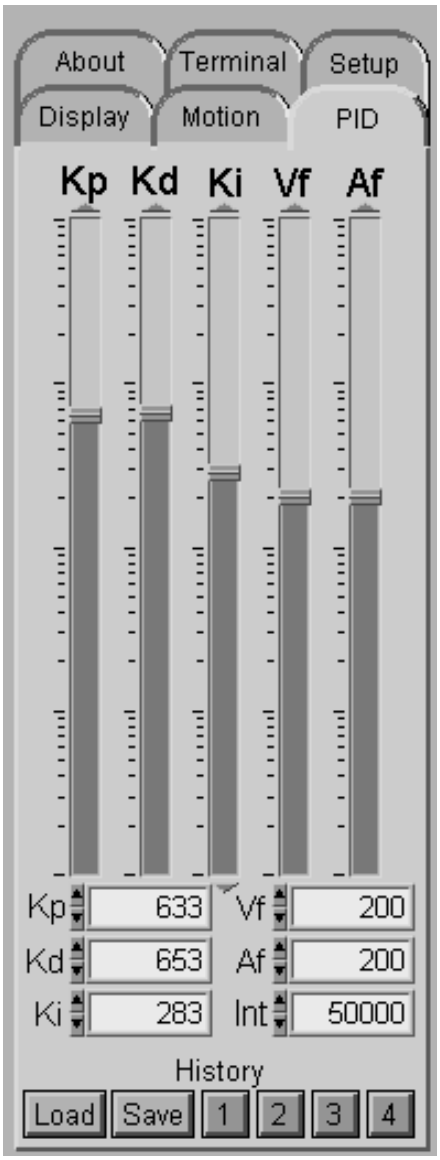


Figure 4.2.5: Servo Tuning – PID Tab

Setup

Here users can specify the parameters to setup a stage (See **Figure 4.2.6**).

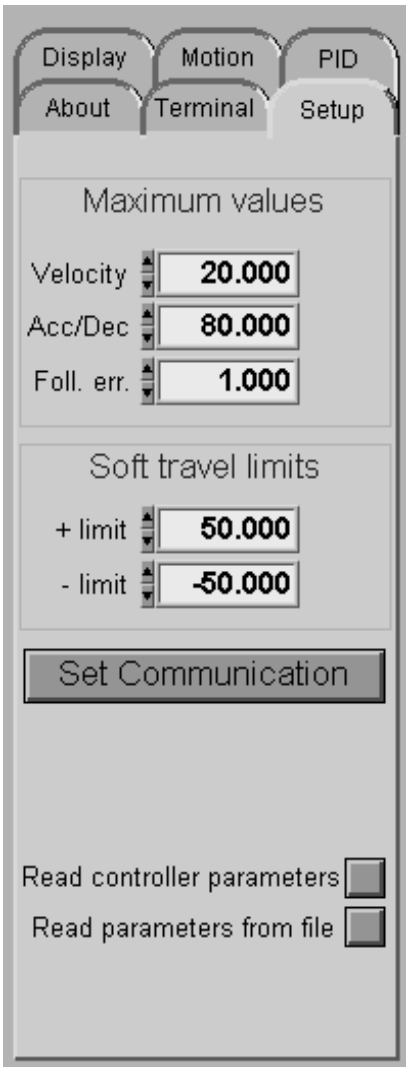


Figure 4.2.6: Servo Tuning – Setup Tab

Terminal

Here users can issue simple commands to the controller and obtain responses from the controller in case of queries. Note that this is not intended to download script files to the controller or for any advanced features ESP6000 motion controller has to offer. In order to use the advanced features users are recommended to use the ASCII terminal available in ESP-Util program (See **Figure 4.2.7**).

Figure 4.2.7: Servo Tuning – Terminal Tab

About

Here users can find the version of ESP-Tuning program (See **Figure 4.2.8**).

Refer to Section 7, Servo Tuning, for further guidance.

NOTE

Save configuration input on a systematic basis to ensure operating parameters are not lost.

CAUTION

Do not disconnect stages while the personal computer and UniDrive are powered up.

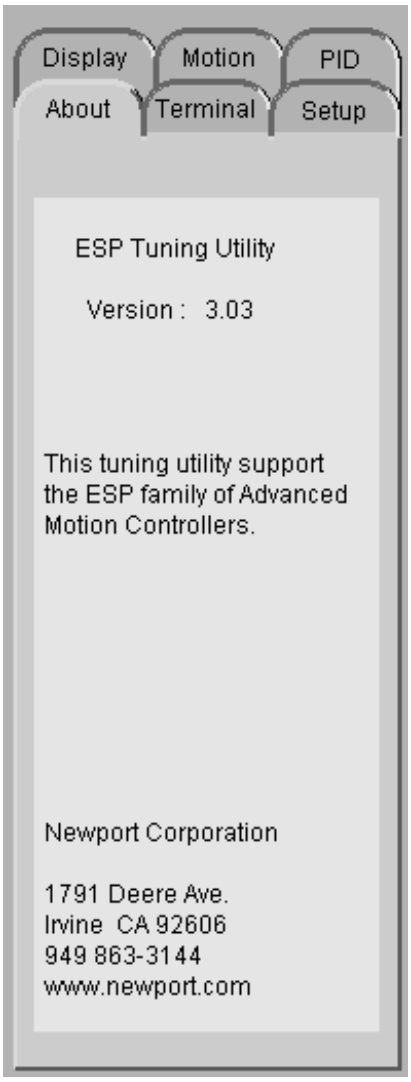


Figure 4.2.8: Servo Tuning – About Tab

Section 5 – Programming

5.1 Introduction

The objective of this section is to give an insight into the ESP6000 motion controller philosophy, and to detail ways in which users can take advantage of the rich features ESP6000 has to offer in solving their application needs.

ESP6000 controller card was designed such that users with minimal motion control background and average programming skills can get their system with advanced motion control features up-and-running with relative ease. Users can get their desired motion performance from the controller by writing a software application that calls various Application Programming Interface (API) commands contained in a Dynamic Link Library (DLL) supplied with the controller. This extensive list of ESP6000 API commands was developed to address a wide range of motion requirements: simple point-to-point motion of servo/stepper motors to linear/circular interpolation and contouring. Please refer the System Overview sub-section in Introduction for a brief listing of some of the key features ESP6000 motion controller has to offer.

The following section details different ways in which one can communicate with the controller. The subsequent sub-sections give a detailed description of all the API commands supplied with the controller.

5.2 How to Communicate with the Controller

ESP6000 has unique architecture that enables it to communicate directly with the host CPU through its data bus. As a result, users can send binary commands to the controller through PCI interface at a very fast rate.

This method of communicating with the controller is quite distinct from what many users are familiar with from there past experiences with standalone motion controllers: issue ASCII commands through serial/GPIB/other forms of communication interface.

While both approaches have their own advantages and disadvantages, ESP6000 provides for a clever way in which one can issue binary or ASCII commands to the controller via the PCI bus. Users can develop their application software in Visual C/C++, Visual Basic or LabVIEW programming environments to issue these commands (Binary/ASCII) to the controller. The commands are sent from the host to the controller through a Dynamic Link Library (DLL) that is supplied with the controller. The block diagram below illustrates the way to communicate with the controller.

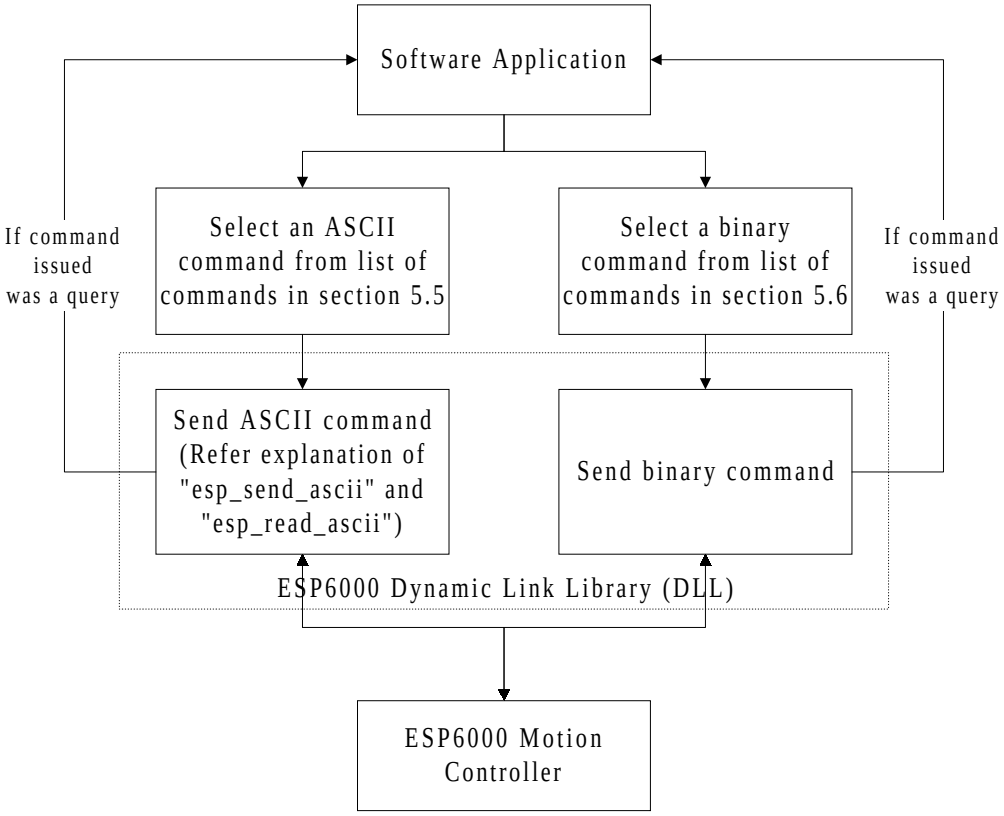


Figure 5.2.1: Communication with the Controller

5.3 General Description

The ESP6000 controller card and device driver must be installed correctly before programming can begin. The Dynamic Link Library (DLL) provides communication to the ESP6000 controller card via the PCI bus. When the system is initialized, the DLL makes a call to the device driver to open communications. The device driver responds with the address of the memory location, the DLL can use for shared memory. It is important to check the return value from the function esp_init_system (see the Commands paragraphs in this section) to insure that the shared memory was locked down.

5.3.1 Windows Programming

The libraries provided are intended to be used with the Windows 95 and Windows NT operating systems. If the user is not an experienced Windows programmer, they should consult a good book on the subject, "Programming Windows 95", by Charles Petzold, for example. Review the examples on the utility disk for familiarization with library usage.

5.3.2 How To Use The Dynamic Link Library

The user should make calls to the Dynamic Link Library, the same way they would call any other function. The library must be linked to the user's program either by using the LoadLibrary( ) function provided with the Windows 95 API or by using the import library included with the DLL.

5.3.3 Minimum Software Requirements

The minimum software version level requirements for ASCII as well as binary commands are listed in table below:

Language	Version
Visual C/C++	Any 32-bit compiler for Windows
Visual Basic	4.0
LabVIEW	4.0.1

Table 5.3.3.1: Software Versions

5.4 Description Of Commands

The ESP6000 provides various Application Programming Interface (API) commands for the user as an alternate to using the Windows setup utility (ESP-util.exe).

5.4.1 ASCII Commands

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 ESP6000 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 5.3.1 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 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.

5.4.2 Binary Commands

Commands are provided for Visual C/C++, Visual BASIC, and LabVIEW programming environments via a DLL. Minimum software version level requirements for commands are listed in **Table 5.3.3.1.**

General command categories are listed in **Table 5.4.2.1.** Command lists and information (C language representation only) are provided in the following paragraph.

Category
Initialization
ASCII Command Interface
General Mode Selection
Status Functions
Motion & Position Control
Trajectory Definition
Flow Control & Sequencing
Programming
Digital Filters
I/O Functions

Table 5.4.2.1: API Function Categories

5.5 Commands (ASCII)

5.5.1 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 semicolon (;). For example, 3MO; 3PA10.0; 3WS; 3MF is a valid command line.

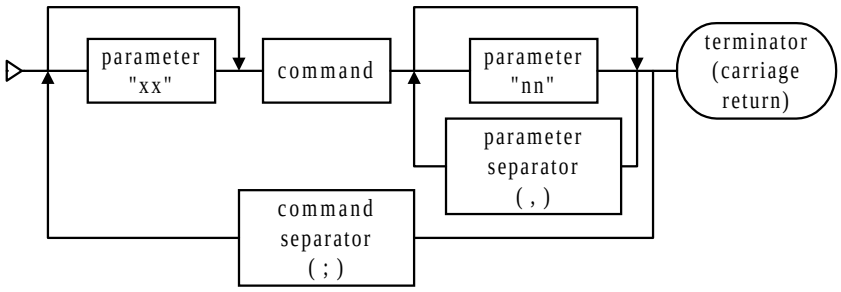


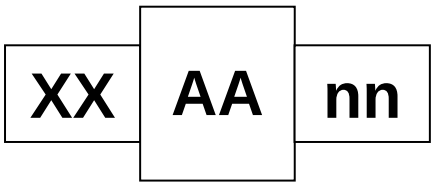
Figure 5.5.1.1: Command Syntax

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.

5.5.2 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 in any position, including inside a numerical value. 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.

5.5.3 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.
PBM	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.

The following table is a CROSS-REFERENCE listing comparing ***Binary*** commands, grouped by function, with the equivalent ***ASCII*** commands:

INITIALIZATION

Binary Command	Page	ASCII Command	Page
esp_init_system(void)	5-215	-	-
esp_open_system(void)	5-217	-	-
esp_running_nt	5-218	-	-
esp_get_num_boards	5-219	-	-
esp_set_target_board	5-220	-	-

Table 5.5.3: Binary Commands vs. ASCII Commands

ASCII COMMAND INTERFACE

Binary Command	Page	ASCII Command	Page
esp_send_ascii	5-221	-	-
esp_read_ascii	5-223	-	-
esp_send_read_ascii	5-225	-	-

Table 5.5.3: Binary Commands vs. ASCII Commands

GENERAL MODE SELECTION

Binary Command	Page	ASCII Command	Page
esp_update_unidrive(long axis)	5-227	QD	5-139
esp_get_sys_config(long *config)	5-228	ZU	5-206
esp_set_sys_fault_config(long config)	5-230	ZZ	5-208
esp_get_sys_fault_config(long config)	5-230	ZZ?	5-208
esp_set_followerr_config(long axis, long config)	5-232	ZF	5-200
esp_get_followerr_config(long axis, long *config)	5-232	ZF?	5-200
esp_set_hardlimit_config(long axis, long config)	5-234	ZH	5-202
esp_get_hardlimit_config(long axis, long *config)	5-234	ZH?	5-202
esp_set_softlimit_config(long axis, long config)	5-236	ZS	5-204
esp_get_softlimit_config(long axis, long *config)	5-236	ZS?	5-204
esp_set_ampio_config(long axis, long config)	5-238	ZA	5-192
esp_get_ampio_config(long axis, long *config)	5-238	ZA?	5-192
esp_set_feedback_config(long axis, long config)	5-240	ZB	5-195
esp_get_feedback_config(long axis, long *config)	5-240	ZB?	5-195
esp_set_e_stop_config(long axis, long config)	5-242	ZE	5-198
esp_get_e_stop_config(long axis, long *config)	5-242	ZE?	5-198
esp_set_dac_offset(long axis, float offset)	5-244	DO	5-68
esp_get_dac_offset(long axis, float *offset)	5-244	DO?	5-68
esp_set_traj_mode(long axis, long mode)	5-245	TJ	5-172
esp_get_traj_mode(long axis, long *mode)	5-245	TJ?	5-172
esp_enable_motor(long axis)	5-247	MO	5-120
esp_disable_motor(long axis)	5-249	MF	5-119

Table 5.5.3: Binary Commands vs. ASCII Commands(Continued)

STATUS FUNCTIONS

Binary Command	Page	ASCII Command	Page
esp_get_hardware_status(long *hardstat1, long *hardstat2)	5-251	PH	5-134
esp_move_done(long axis)	5-254	MD?	5-118
esp_home_done(long axis)	5-255	MD?	5-118
esp_get_motion_status(long *mstat)	5-257	TS	5-175
esp_get_motor_onoff_status(long *onoff)	5-258	MO?	5-120
esp_set_position(long axis, double *position)	5-260	DH	5-66
esp_get_position(long axis, double *position)	5-260	TP	5-174
esp_get_error_num(int *error, int *ServoTick)	5-261	TE	5-171
esp_get_error_string(char *ErrorString, int *error, int *ServoTick)	5-262	TB	5-170
esp_get_version(char *FirmVer, char *DllVer)	5-264	VE	5-183

Table 5.5.3: Binary Commands vs. ASCII Commands (Continued)

MOTION & POSITION CONTROL

Binary Command	Page	ASCII Command	Page
esp_move_absolute(long axis, double position)	5-265	PA	5-129
esp_move_relative(long axis, double position)	5-266	PR	5-137
esp_find_home(long axis, long mode)	5-267	OR	5-127
esp_stop(long axis)	5-269	ST	5-168
esp_stop_all(void)	5-270	AB	5-26
esp_set_position(long axis, double position)	5-271	DH	5-66
esp_get_position(long axis, double *position)	5-271	TP	5-174

Table 5.5.3: Binary Commands vs. ASCII Commands (Continued)

TRAJECTORY DEFINITION

Binary Command	Page	ASCII Command	Page
esp_set_speed(long axis, float speed)	5-272	VA	5-180
esp_get_speed(long axis, float *speed)	5-272	VA?	5-180
esp_set_max_speed(long axis, float speed)	5-273	VU	5-186
esp_get_max_speed(long axis, float *speed)	5-273	VU?	5-186
esp_set_accel(long axis, float accel)	5-274	AC	5-27

Table 5.5.3: Binary Commands vs. ASCII Commands (Continued)

TRAJECTORY DEFINITION (Continued)

Binary Command	Page	ASCII Command		Page
esp_get_accel(long axis, float *accel)	5-274	AC?		5-27
esp_set_decel(long axis, float decel)	5-275	AG		5-33
esp_get_decel(long axis, float *decel)	5-275	AG?		5-33
esp_set_max_accel(long axis, float accel)	5-276	AU		5-38
esp_get_max_accel(long axis, float *accel)	5-276	AU?		5-38
esp_set_jerk(long axis, float jerk)	5-277	JK		5-110
esp_get_jerk(long axis, float *jerk)	5-277	JK?		5-110
esp_set_home_speed(long axis, float speed)	5-279	OH		5-125
esp_get_home_speed(long axis, float *speed)	5-279	OH?		5-125
esp_set_startup_speed(long axis, float speed)	5-281	VB		5-182
esp_get_startstop_speed(long axis, float *speed)	5-281	VB?		5-182
esp_set_jog_speed(long axis, float speed)	5-282	High	JH	5-109
		Low	JW	5-112
esp_get_jog_speed(long axis, float *speed)	5-282	High	JH?	5-109
		Low	JW?	5-112

Table 5.3.3: Binary Commands vs. ASCII Commands (Continued)

FLOW CONTROL AND SEQUENCING

Binary Command	Page	ASCII Command	Page
esp_delay(float time)	5-284	WT	5-189
esp_delay_after_stop(long axis, float time)	5-286	WS	5-188

Table 5.3.3: Binary Commands vs. ASCII Commands (Continued)

MOTION DEVICE PARAMETERS

Binary Command	Page	ASCII Command		Page
esp_set_motor_type(long axis, long mtype)	5-288	QM		5-143
esp_get_motor_type(long axis, long *mtype)	5-288	QM?		5-143
esp_set_master_slave(long master, long slave)	5-290	SS		5-166
esp_get_master_slave(long *master, long slave)	5-290	SS?		5-166
esp_set_master_slave_ratio(long slave, float ratio)	5-292	GR		5-79
esp_get_master_slave_ratio(long slave, float *ratio)	5-292	GR?		5-79
esp_set_resolution(long axis, float resolution, long units)	5-294	Units	SN	5-163
		Resolution	SU	5-169
esp_get_resolution(long axis, float *resolution, long * units)	5-294	Units	SN?	5-163
		Resolution	SU?	5-169
esp_set_soft_limits(long axis, double neg, double pos)	5-296	Negative	SL	5-160
		Positive	SR	5-164
esp_get_soft_limits(long axis, double neg, double *position)	5-296	Negative	SL?	5-160
		Positive	SR?	5-164
esp_set_following_error(long axis, double error)	5-298	FE		5-76
esp_get_following_error(long axis, double *error)	5-298	FE?		5-76
esp_set_microstep_factor(long axis, long resolution)	5-299	QS		5-147
esp_get_microstep_factor(long axis, long *resolution)	5-299	QS?		5-147
esp_set_fullstep_resolution(float, fullstep)	5-301	FR		5-78
esp_get_fullstep_resolution(float, *fullstep)	5-301	FR?		5-78
esp_set_motor_current(long axis, float current)	5-303	QI		5-142
esp_get_motor_current(long axis, float *current)	5-303	QI?		5-142
esp_set_tach_constant(long axis, float tach)	5-305	QT		5-148
esp_get_tach_constant(long axis, float *tach)	5-305	QT?		5-148
esp_set_gear_constant(long axis, float gear)	5-307	QG		5-140
esp_get_gear_constant(long axis, float *gear)	5-307	QG?		5-140

Table 5.3.3: Binary Commands vs. ASCII Commands (Continued)

PROGRAMMING

Binary Command	Page	ASCII Command	Page
esp_save_parameters(void)	5-309	SM	5-162

Table 5.3.3: Binary Commands vs. ASCII Commands (Continued)

DIGITAL FILTERS

Binary Command	Page	ASCII Command	Page
esp_set_kp(long axis, float kp)	5-311	KP	5-115
esp_get_kp(long axis, float *kp)	5-311	KP?	5-115
esp_set_kd(long axis, float kd)	5-313	KD	5-113
esp_get_kd(long axis, float *kd)	5-313	KD?	5-113
esp_set_ki(long axis, float ki)	5-315	KI	5-114
esp_get_ki(long axis, float *ki)	5-315	KI?	5-114
esp_set_il(long axis, float il)	5-317	KS	5-116
esp_get_il(long axis, float *il)	5-317	KS?	5-116
esp_set_vel_feedforward(long axis, float vff)	5-319	VF	5-184
esp_get_vel_feedforward(long axis, float *vff)	5-319	VF?	5-184
esp_set_acc_feedforward(long axis, float aff)	5-321	AF	5-31
esp_get_acc_feedforward(long axis, float *aff)	5-321	AF?	5-31
esp_update_filter(void)	5-323	UF	5-179

Table 5.5.3: Binary Commands vs. ASCII Commands (Continued)

I/O FUNCTIONS

Binary Command	Page	ASCII Command	Page
esp_set_adc_gain(long gain)	5-324	AM	5-35
esp_get_adc_gain(long *gain)	5-324	AM?	5-35
esp_set_adc_range(long range)	5-326	AM	5-35
esp_get_adc_range(long *range)	5-326	AM?	5-35
esp_get_adc(long channel, float *conversion, long *TickCnt)	5-328	RA	5-150
esp_get_all_adc(float *channel, long *TickCnt)	5-330	RA	5-150
esp_set_daq_mode(long mode, long axis, long Adcs, long feedback, long rate, long Num)	5-332	DC	5-57
esp_enable_daq(void)	5-335	DE	5-63
esp_disable_daq	5-337	DE	5-63
esp_get_daq_status(long *count)	5-339	DF	5-64
esp_daq_done( )	5-341	DD	5-62
esp_get_daq_data(long *DataArray, long *Num, long *DaqStat)	5-343	DG	5-65

Table 5.3.3: Binary Commands vs. ASCII Commands (Continued)

I/O FUNCTIONS (Continued)

Binary Command	Page	ASCII Command	Page
esp_set_portabc_dir(long PortA, longPortB, long PortC)	5-346	BO	5-46
esp_get_portabc_dir(long *PortA, long *PortB, long *PortC)	5-346	BO?	5-46
esp_set_dio_porta(long data)	5-347	SB	5-155
esp_get_dio_porta(long *data)	5-347	SB?	5-155
esp_set_dio_portb(long data)	5-348	SB	5-155
esp_get_dio_portb(long *data)	5-348	SB?	5-155
esp_set_dio_portc(long data)	5-349	SB	5-155
esp_get_dio_portc(long *data)	5-349	SB?	5-155
esp_set_dio_inhibit	5-350	BL	5-43
esp_get_dio_inhibit	5-350	BL?	5-43
esp_set_dio_inhibit_cfg	5-351	BK	5-42
esp_get_dio_inhibit_cfg	5-351	BK?	5-42
esp_set_dio_motion	5-353	BN	5-45
esp_get_dio_motion	5-353	BN?	5-45
esp_set_dio_motion_cfg	5-354	BM	5-44
esp_get_dio_motion_cfg	5-354	BM?	5-44

Table 5.5.3: Binary Commands vs. ASCII Commands (Continued)

5.5.3.1 Command List By Category

The command lists for the ASCII commands are divided by various application categories:

- *General Mode Selection*
- *Status Functions*
- *Motion & Position Control*
- *Motion Device Parameters*
- *Programming*
- *Trajectory Definition*
- *Flow Control & Sequencing*
- *I/O Functions*
- *Group Functions*
- *Digital Filters*
- *Master-Slave Mode Definition*

GENERAL MODE SELECTION

Cmd.	Description	IMM	PGM	MIP	Page
BQ	Enable/disable DIO jog mode	♦	♦	♦	5-48
BR	Set serial communication speed	♦	♦	♦	5-49
DO	General Mode Selection	♦	♦	♦	5-68
FP	Set Position Display Resolution	♦	♦	♦	5-77
MF	Power OFF	♦	♦	♦	5-119
MO	Power ON	♦	♦	♦	5-120
OM	Set home search mode	♦	♦	♦	5-126
QD	Update Unidriver amplifier	♦	♦	♦	5-139
RS	Reset the controller	♦		♦	5-152
TJ	Set trajectory mode	♦	♦		5-172
ZA	Set amplifier configuration	♦	♦		5-192
ZB	Set feedback configuration	♦	♦		5-195
ZE	Set E-stop configuration	♦	♦		5-198
ZF	Set following error configuration	♦	♦		5-200
ZH	Set hardware limit configuration	♦	♦		5-202
ZS	Set software limit configuration	♦	♦		5-204
ZU	Get ESP system configuration			♦	5-206
ZZ	Set system configuration	♦	♦		5-208

Table 5.5.3.1: Command List By Category

STATUS FUNCTIONS

Cmd.	Description	IMM	PGM	MIP	Page
DP	Get target position	♦		♦	5-69
DV	Set working speed	♦		♦	5-70
ID	Get stage model and serial number	♦		♦	5-108
MD	Get axis motion status	♦		♦	5-118
PH	Get hardware status	♦		♦	5-134
TB	Get error message	♦		♦	5-170
TE	Get error number	♦		♦	5-171
TP	Get position	♦		♦	5-174
TS	Get Controller Status	♦		♦	5-175
TV	Get velocity	♦		♦	5-177
TX	Get controller activity	♦		♦	5-178
VE	Get firmware version	♦		♦	5-183
XM	Get available program memory	♦		♦	5-190

Table 5.5.3.1: Command List By Category (Continued)

MOTION & POSITION CONTROL

Cmd.	Description	IMM	PGM	MIP	Page
AB	Abort motion	♦		♦	5-26
DH	Define home	♦	♦		5-66
MT	Move to hardware travel limit	♦	♦	♦	5-121
MV	Move indefinitely	♦	♦	♦	5-122
MZ	Move to nearest index	♦	♦		5-124
OR	Origin searching	♦	♦	♦	5-127
PA	Move absolute	♦	♦	♦	5-129
PR	Move relative	♦	♦	♦	5-137
ST	Stop motion (Abort)	♦	♦	♦	5-168

Table 5.5.3.1: Command List By Category (Continued)

MOTION DEVICE PARAMETERS

Cmd.	Description	IMM	PGM	MIP	Page
FE	Set following error threshold	♦	♦	♦	5-76
FR	Set Encoder full-step resolution	♦	♦	♦	5-78
GR	Set gear ratio	♦	♦	♦	5-79
QG	Set gear constant	♦	♦		5-130
QI	Motor current	♦	♦		5-142
QM	Define motor type	♦	♦		5-143
QR	Torque reduction	♦	♦	♦	5-146
QS	Set microscope factor	♦	♦		5-147
QT	Define tachometer constant	♦	♦		5-148
QV	Set motor voltage	♦	♦		5-149
SI	Set master-slave jog update interval	♦	♦	♦	5-158
SK	Set slave axis jog velocity coefficients	♦	♦	♦	5-159
SL	Set left travel limit	♦	♦	♦	5-160
SN	Set units	♦	♦		5-163
SR	Set right limit	♦	♦		5-164
SS	Set master-slave relationship	♦	♦		5-166
SU	Set encoder resolution	♦	♦		5-169

Table 5.5.3.1: Command List By Category (Continued)

PROGRAMMING

Cmd.	Description	IMM	PGM	MIP	Page
DL	Define Label		♦		5-67
EO	Automatic execution on power on	♦		♦	5-71
EP	Enter program download mode	♦			5-72
EX	Execute stored program	♦	♦		5-75
JL	Jump to label		♦	♦	5-111
LP	List program	♦		♦	5-117
QP	Quit program mode	♦			5-144
SM	Save to non-volatile memory	♦			5-162
XM	Get available program memory	♦		♦	5-190
XX	Delete a stored program	♦		♦	5-191

Table 5.5.3.1: Command List By Category (Continued

TRAJECTORY DEFINITION

Cmd.	Description	IMM	PGM	MIP	Page
AC	Set acceleration	♦	♦	♦	5-27
AE	Set e-stop deceleration	♦	♦	♦	5-29
AG	Set deceleration	♦	♦	♦	5-33
AU	Set maximum acceleration	♦	♦	♦	5-38
BA	Set backlash compensation	♦	♦	♦	5-40
CO	Set linear compensation	♦	♦	♦	5-53
JH	Set jog high speed	♦	♦	♦	5-109
JK	Set jerk rate	♦	♦	♦	5-110
JW	Set jog low speed	♦	♦	♦	5-112
OH	Set home search high speed	♦	♦	♦	5-125
SH	Set home preset position	♦	♦	♦	5-157
UF	Update filter parameters	♦	♦	♦	5-179
VA	Set velocity	♦	♦	♦	5-180
VB	Set base velocity for step motors	♦	♦	♦	5-182
VU	Set maximum speed	♦	♦	♦	5-186

Table 5.5.3.1: Command List By Category (Continued)

FLOW CONTROL & SEQUENCING

Cmd.	Description	IMM	PGM	MIP	Page
DL	Define label		♦		5-67
JL	Jump to label		♦	♦	5-111
RQ	Generate service request	♦	♦	♦	5-151
SA	Set device address	♦	♦	♦	5-154
WP	Wait for absolute position crossing	♦	♦	♦	5-187
WS	Wait for stop	♦	♦	♦	5-188
WT	Wait for Time	♦	♦	♦	5-189

Table 5.5.3.1: Command List By Category (Continued)

I/O FUNCTIONS

Cmd.	Description	IMM	PGM	MIP	Page
AM	Set analog input mode	♦	♦	♦	5-35
BG	Assign DIO bits to execute stored programs	♦		♦	5-41
BK	Assign DIO bits to inhibit motion	♦	♦	♦	5-42
BL	Enable DIO bits to inhibit motion	♦	♦	♦	5-43
BM	Assign DIO bits to notify motion status	♦	♦	♦	5-44
BN	Enable DIO bits to notify motion status	♦	♦	♦	5-45
BO	Set DIO Port Direction	♦	♦	♦	5-46
BP	Assign DIO for jog mode	♦	♦	♦	5-47
BQ	Enable/disable DIO jog mode	♦	♦	♦	5-48
DC	Setup data acquisition	♦		♦	5-57
DD	Get data acquisition done status	♦		♦	5-62
DE	Enable/disable data acquisition	♦		♦	5-63
DF	Get data acquisition sample count	♦		♦	5-64
DG	Get acquisition data	♦		♦	5-65
ES	Define event action command string	♦		♦	5-73
PC	Set position compare mode	♦	♦	♦	5-131
RA	Read analog output	♦		♦	5-150
SB	Set DIO state	♦	♦	♦	5-155

Table 5.5.3.1: Command List By Category (Continued)

GROUP FUNCTIONS

Cmd.	Description	IMM	PGM	MIP	Page
HA	Set group acceleration	♦	♦	♦	5-80
HB	Read list of groups assigned	♦		♦	5-82
HC	Move group along an arc	♦	♦	♦	5-83
HD	Set group deceleration	♦	♦	♦	5-85
HE	Set group E-stop deceleration	♦	♦	♦	5-87
HF	Group motor power OFF	♦	♦	♦	5-89
HJ	Set group jerk	♦	♦	♦	5-90
HL	Move group along a line	♦	♦	♦	5-92
HN	Create new group	♦	♦		5-94
HO	Group motor power ON	♦	♦	♦	5-97
HP	Get group position	♦	♦	♦	5-98
HQ	Wait for group via point buffer near empty	♦	♦	♦	5-99
HS	Stop group motion	♦	♦	♦	5-101
HV	Set group velocity	♦	♦	♦	5-102
HW	Wait for group motion to stop	♦	♦	♦	5-104
HX	Delete a group	♦	♦	♦	5-106
HZ	Get group size	♦	♦	♦	5-107

Table 5.5.3.1: Command List By Category (Continued)

DIGITAL FILTERS

Cmd.	Description	IMM	PGM	MIP	Page
AF	Acceleration/Deceleration feed-forward gain	♦	♦	♦	5-31
CL	Set closed loop update interval	♦	♦	♦	5-51
DB	Set position deadband	♦	♦	♦	5-55
KD	Set derivative gain Kd	♦	♦	♦	5-113
KI	Set integral gain Ki	♦	♦	♦	5-114
KP	Set proportional gain Kp	♦	♦	♦	5-115
KS	Set saturation coefficient Ks	♦	♦	♦	5-116
UF	Update Filter Parameters	♦	♦	♦	5-179
VF	Set velocity feed-forward gain	♦	♦	♦	5-184

Table 5.5.3.1: Command List By Category (Continued)

MASTER-SLAVE MODE DEFINITION

Cmd.	Description	IMM	PGM	MIP	Page
GR	Set master-slave ratio	♦	♦	♦	5-79
SS	Set master-slave mode	♦	♦		5-166

Table 5.5.3.1: Command List By Category (Continued)

5.5.3.2 Command List - Alphabetical

The Alphabetical Command List indicates each command (TWO character mnemonic), the description name of the command, the mode the command can be used in (See sub-section 5.5.3), and the specific page the individual command is defined.

Cmd.	Description	IMM	PGM	MIP	Page
AB	Abort motion	♦		♦	5-26
AC	Set acceleration	♦	♦	♦	5-27
AE	Set e-stop deceleration	♦	♦	♦	5-29
AF	Set acceleration feed-forward gain	♦	♦	♦	5-31
AG	Set deceleration	♦	♦	♦	5-33
AM	Set analog input mode	♦	♦	♦	5-35
AP	Abort program	♦	♦	♦	5-37
AU	Set maximum acceleration and deceleration	♦	♦	♦	5-38
BA	Set backlash compensation	♦	♦	♦	5-40
BG	Assign DIO bits to execute stored programs	♦		♦	5-41
BK	Assign DIO bits to inhibit motion	♦	♦	♦	5-42
BL	Enable DIO bits to inhibit motion	♦	♦	♦	5-43
BM	Assign DIO bits to notify motion status	♦	♦	♦	5-44
BN	Enable DIO bits to notify motion status	♦	♦	♦	5-45
BO	Set DIO port A, B, C direction	♦	♦	♦	5-46
BP	Assign DIO bits for jog mode	♦	♦	♦	5-47
BQ	Enable DIO bits for jog mode	♦	♦	♦	5-48
BR	Set serial communication speed	♦	♦	♦	5-49
CL	Set closed loop update interval	♦	♦	♦	5-51
CO	Set linear compensation	♦	♦	♦	5-53
DB	Set position deadband	♦	♦	♦	5-55
DC	Setup data acquisition	♦		♦	5-57
DD	Get data acquisition done status	♦		♦	5-62
DE	Enable/disable data acquisition	♦		♦	5-63
DF	Get data acquisition sample count	♦		♦	5-64
DG	Get acquisition data	♦		♦	5-65
DH	Define home	♦	♦	♦	5-66
DL	Define label	♦	♦	♦	5-67
DO	Set dac offset	♦	♦	♦	5-68
DP	Read desired position	♦		♦	5-69

Table 5.5.3.2: Command List - Alphabetical

Cmd.	Description	IMM	PGM	MIP	Page
DV	Read desired velocity	♦		♦	5-70
EO	Automatic execution on power on	♦		♦	5-71
EP	Enter program mode	♦			5-72
ES	Define event action command string	♦		♦	5-73
EX	Execute a program	♦	♦		5-75
FE	Set maximum following error threshold	♦	♦	♦	5-76
FP	Set position display resolution	♦	♦	♦	5-77
FR	Set encoder full-step resolution	♦	♦	♦	5-78
GR	Set master-slave reduction ratio	♦	♦	♦	5-79
HA	Set group acceleration	♦	♦	♦	5-80
HB	Read list of groups assigned	♦	♦	♦	5-82
HC	Move group along an arc	♦	♦	♦	5-83
HD	Set group deceleration	♦	♦	♦	5-85
HE	Set group e-stop deceleration	♦	♦	♦	5-87
HF	Group motor power off	♦	♦	♦	5-89
HJ	Set group jerk	♦	♦	♦	5-90
HL	Move group along a line	♦	♦	♦	5-92
HN	Create new group	♦	♦		5-94
HO	Group on	♦	♦	♦	5-97
HP	Read group position	♦		♦	5-98
HQ	Wait for group command buffer level	♦	♦	♦	5-99
HS	Stop group motion	♦	♦	♦	5-101
HV	Set group velocity	♦	♦	♦	5-102
HW	Wait for group motion step	♦	♦	♦	5-104
HX	Delete group	♦	♦	♦	5-106
HZ	Read group size	♦	♦	♦	5-107
ID	Read stage model and serial number	♦	♦	♦	5-108
JH	Set jog high speed	♦	♦	♦	5-109
JK	Set jerk rate	♦	♦	♦	5-110
JL	Jump to label		♦	♦	5-111
JW	Set jog low speed	♦	♦	♦	5-112
KD	Set derivative gain	♦	♦	♦	5-113
KI	Set integral gain	♦	♦	♦	5-114
KP	Set proportional gain	♦	♦	♦	5-115
KS	Set saturation level of integral factor	♦	♦	♦	5-116
LP	List program	♦			5-117
MD	Read motion done status	♦			5-118
MF	Motor power off	♦	♦	♦	5-119
MO	Motor power on	♦	♦	♦	5-120
MT	Move to hardware travel limits	♦	♦	♦	5-121
MV	Move indefinitely	♦	♦	♦	5-122
MZ	Move to nearest index	♦	♦		5-124
OH	Set home search high speed	♦	♦	♦	5-125
OM	Set home search mode	♦	♦	♦	5-126
OR	Search for home	♦	♦		5-127
PA	Move to absolute position	♦	♦	♦	5-129
PC	Set position compare mode	♦	♦	♦	5-131
PH	Get hardware status	♦		♦	5-134

Table 5.5.3.2: Command List – Alphabetical (Continued)

Cmd.	Description	IMM	PGM	MIP	Page
PR	Move to relative position	♦	♦	♦	5-137
QD	Update motor driver settings	♦	♦		5-139
QG	Set gear constant	♦	♦		5-140
QI	Set maximum motor current	♦	♦		5-142
QM	Set motor type	♦	♦		5-143
QP	Quit program mode	♦			5-145
QR	Reduce motor torque	♦	♦	♦	5-146
QS	Set microstep factor	♦	♦		5-147
QT	Set tachometer gain	♦	♦		5-148
QV	Set average motor voltage	♦	♦		5-149
RA	Read analog output	♦		♦	5-150
RQ	Generate service request	♦	♦	♦	5-151
RS	Reset the controller	♦		♦	5-152
SA	Set device address	♦	♦	♦	5-154
SB	Set/get DIO port A, B, C bit status	♦	♦	♦	5-155
SH	Set home preset position	♦	♦	♦	5-157
SI	Set master-slave jog velocity update interval	♦	♦	♦	5-158
SK	Set master-slave jog velocity scaling	♦	♦	♦	5-159
SL	Set left travel limit	♦	♦	♦	5-160
SM	Save settings to non-volatile memory	♦			5-162
SN	Set axis displacement units	♦	♦		5-163
SR	Set right travel limit	♦	♦		5-164
SS	Define master-slave relationship	♦	♦		5-166
ST	Stop motion	♦	♦	♦	5-168
SU	Set encoder resolution	♦	♦		5-169
TB	Read error message	♦		♦	5-170
TE	Read error code	♦		♦	5-171
TJ	Set Trajectory mode	♦	♦		5-172
TP	Read actual position	♦		♦	5-174
TS	Get controller status	♦		♦	5-175
TV	Get actual velocity	♦		♦	5-177
TX	Get controller activity	♦		♦	5-178
UF	Update servo filter	♦	♦	♦	5-179
VA	Set velocity	♦	♦	♦	5-180
VB	Set base velocity for step motors	♦	♦	♦	5-182
VE	Read controller firmware version	♦	♦	♦	5-183
VF	Set velocity feed-forward gain	♦	♦	♦	5-184
VU	Set maximum velocity	♦	♦	♦	5-186
WP	Wait for absolute position crossing	♦	♦	♦	5-187
WS	Wait for motion stop	♦	♦	♦	5-188
WT	Wait	♦	♦	♦	5-189
XM	Get available program memory	♦		♦	5-190
XX	Delete a stored program	♦		♦	5-191
ZA	Set amplifier I/O configuration	♦	♦		5-192
ZB	Set feedback configuration	♦	♦		5-195
ZE	Set e-stop configuration	♦	♦		5-198
ZF	Set following error configuration	♦	♦		5-200
ZH	Set hardware limit configuration	♦	♦		5-202

Table 5.5.3.2: Command List – Alphabetical (Continued)

Cmd.	Description	IMM	PGM	MIP	Page
ZS	Set hardware limit configuration	♦	♦		5-204
ZU	Get ESP system configuration	♦		♦	5-206
ZZ	Set software limit configuration	♦	♦		5-208

Table 5.5.3.2: Command List – Alphabetical (Continued)

5.5.4 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 numbers is from 1 to MAX AXES, where MAX AXES is dependent 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 diamonds 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 configuration
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**
nn - **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/s²*
	2AC9	| *set acceleration to 9 units/s²*
	2AG6	| *set deceleration to 6 units/s²*
	2AU15	| *set axis # 2 maximum acceleration/deceleration to 15 units/s²*
	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** 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 7, 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/s²*
2AE150	*set e-stop deceleration to 150 units/s²*

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 Kf. 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 velocity 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 - **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 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 - execute 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 xx**AM**

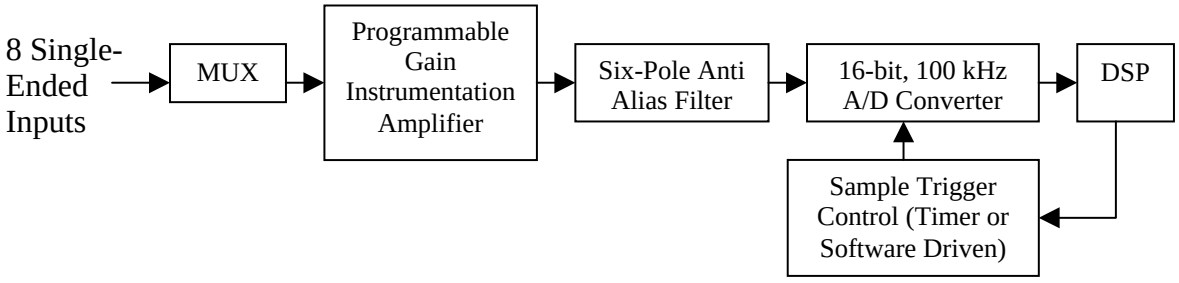
PARAMETERS

Description	**nn** [int]	-	analog input mode
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 of eight (8) analog input modes listed below:

- **nn** = 0: ±10V input range
- **nn** = 1: ±5V input range
- **nn** = 2: 0 – 10V input range
- **nn** = 3: 0 – 5V input range
- **nn** = 4: ±2.50V input range
- **nn** = 5: ±1.25V input range
- **nn** = 6: 0 – 2.50V input range
- **nn** = 7: 0 – 1.25V 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 10V 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 xx2, 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 - 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*

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	-	**0** to **distance equivalent to 1000 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 DETACHED" 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 "?" 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*
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	WIP
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 **03H** (hexadecimal with leading zero(0))
or **?** to read current setting

Units **nn** - none

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.

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

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

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

2BO?	*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 xx**BP**nn1,nn2 or xx**BP**?

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	-	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 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

USAGE

IMM	PGM	MIP
♦	♦	♦

SYNTAX xx**CL**nn or xx**CL**?

PARAMETERS

Description

xx [int]	-	axis number
nn [int]	-	closed loop update interval

Units

xx	-	**0** to **MAX AXES**
nn	-	**0** to **60000**
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 stepper 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*
	| 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

USAGE

IMM	PGM	MIP
♦	♦	♦

SYNTAX **xxCOnn** or **xxCO?**

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

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 **xxDBnn** or **xxDB?**

PARAMETERS

Description	**xx** [int]	-	axis number
	nn [int]	-	deadband value
Range	**xx**	-	**0** to **MAX AXIS**
	nn	-	**0** 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 hysteresis, 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

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*

DC setup data acquisitions

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **DCnn1,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 s 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**
nm6	-	**1** 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. These features are 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.

Data Acquisition Mode	Parameter #3	Parameter #4
0 – 9: Analog data acquisition	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	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

<table>
<tr><td>Data Acquisition Mode</td><td>Parameter #3</td><td colspan="3">Parameter #4</td></tr>
<tr><td>10: Trace variable data acquisition (Continued)</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.</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.</td></tr>
<tr><td></td><td></td><td>Servo motor driven axes</td><td>Stepper motor driven axes</td><td>Commutated stepper motor driven axes</td></tr>
<tr><td></td><td></td><td>Error Integral</td><td>Duration between pulses</td><td>Control (DAC) output #2</td></tr>
<tr><td></td><td></td><td>Trajectory phase</td><td>Trajectory phase</td><td>Trajectory phase</td></tr>
<tr><td></td><td></td><td>Following error</td><td>Following error</td><td>Following error</td></tr>
</table>

<table>
<tr><td>10: Trace variable</td><td>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>Control (DAC) output #1</td><td>Counts/servo cycle</td><td>Control (DAC) output #1</td></tr>
<tr><td></td><td></td><td>Desired acceleration</td><td>Desired acceleration</td><td>Desired acceleration</td></tr>
<tr><td></td><td></td><td>Desired velocity</td><td>Desired velocity</td><td>Desired velocity</td></tr>
<tr><td></td><td></td><td>Desired position</td><td>Desired position</td><td>Desired position</td></tr>
<tr><td></td><td></td><td>Actual position</td><td>Actual position</td><td>Actual position</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*
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 **DEnn**

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

USAGE	IMM	PGM	MIP
	♦		♦

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

USAGE

IMM	PGM	MIP
♦		♦

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).

Note: This command is not supported over the PCI bus, use esp_get_daq_data binary command instead.

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

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] - acceleration 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 **xxDOnn or xxDO?**

PARAMETERS

Description	**xx** [int]	-	DAC channel number
	nn [float]	-	DAC offset value
Range	**xx**	-	**1** to **6**
	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 effect, 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
SM        | Save parameters to non-volatile 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 position, the destination of a certain motion component.

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

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
            | controller returns desired position 7.52 units for
            for axis #3
```

EO automatic execution on power on

USAGE	IMM ♦	PGM	MIP ♦	
SYNTAX	**xxEOnn**		**or**	**xxEO?**
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. "0E0" disables automatic execution of stored programs at power ON.

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 **Esnn 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 [$3_{10} = (11)_2$]*
| *Param. #4: Acquire feedback channels 1 & 3 [$5_{10} = (101)_2$]*

| *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 **2e+9,** 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

USAGE

IMM	PGM	MIP
♦	♦	♦

SYNTAX **xxFPnn** **or** **xxFP?**

PARAMETERS

Description

xx [int]	-	axis number
nn [int]	-	display resolution

Range

xx	-	**1** to **MAX AXES**
nn	-	or **?** to read present setting

Units

xx	-	none
nn	-	none

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 - none

Units
xx - none
nn - none

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

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

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**

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 0*
1HF	*| turn group #1 power OFF*
1HF?	*| query group #1 power status*
0	*| controller returns a value of 1*

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 - predefined 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 [float] - physical axis number to be assigned as first axis in this group

nn_2 [float] - physical axis number to be assigned as second axis in this group

nn_i [float] - physical axis number to be assigned as *i*th axis in this group

Range

xx - **1** to **MAX GROUPS**

nn_i - **1** to **6**

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 one axis 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	*controller returns –1, implying that there was an error*
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
   -1         | controller returns –1, implying that there was an error
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
   -1         | controller returns –1, implying that there was an error
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
   -1         | controller returns –1, implying that there was an error
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 **nn_1, nn_2 … nn_I** where **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 - **0** 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/second²
1HD50       | set vectorial deceleration of group #1 to 50 units/second²
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/second²*
1HD50	*| set vectorial deceleration of group #1 to 50 units/second²*
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 EXTENDED

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
floating point: truncated

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

RETURNS none

REL. COMMANDS **HN** - create 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
floating point: truncated

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] - velocity 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 that 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 - 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 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 - 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 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] - proportional 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: error 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] - 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 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 not 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 - turn 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** - **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 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
MO - turn motor power ON

EXAMPLE

```
2MO      | 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) and to define that limit as home position.
Note: This command cannot be executed 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	♦	♦	

STNTAX **xxMZnn or xxMZ?**

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 it's nearest index (positive or negative). It uses the home search speed during travel to nearest index.

Note: This command cannot be executed 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 - 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 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

EXAMPLE

3OH10	*set home search high speed of axis # 2 to 10 units/sec*
3OH?	*query home search high speed of axis #2*
10	*controller returns a value of 10.0 units/second*

OM set home search mode

	IMM	PGM	MIP
USAGE	♦	♦	♦

SYNTAX **xxOMnn or xxOM?**

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 the "?" 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
USAGE	♦	♦

SYNTAX xx**OR**nn

PARAMETERS

Description

xx [int] - axis number
nn [int] - home mode

Range

xx - **1** 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 **nn** = **0**, a home search routine is initiated sequentially on all installed axes. If **nn** = **1** 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 cannot be issued after enabling DAQ (refer ASCII command, **DE**).

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.

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

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 **xxPCnn1,nn2**

PARAMETERS

Description	**xx** [int]	-	axis number
	nn1 [int]	-	compare modes
	nn2 [int]	-	compare absolute position or relative displacement
Range	**xx**	-	**1** 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

Worst case error	= ± (Axis speed at setup time) * 7 micro seconds
	= ± 40 mm/sec * 10000 counts/mm *7 micro seconds
	= ± 2.8 counts

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 crossing(s). And because this feature is implemented in hardware (and not with CPU 'polling') the timing latency is negligible (<50ns) 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 defined 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: 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., "xx 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.

```
1 EP              // start program entry mode
1 MO              // 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 #2:

The following example 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
1 MO            // enable axis-1 motor power
1 OR 1          // home axis-1
1 WS 1000       // wait for home completion and dwell 1 second
1 PC 2, +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

BIT#	VALUE	DEFINITION
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 connector emergency stop (latched) low
30	1	auxiliary I/O connector 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

BIT#	VALUE	DEFINITION
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 This 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 registers*

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

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 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 - set 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

ERRORS
A - unknown message code
B - incorrect axis number

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 **1**
or **?** to read present setting

Units
xx - none
nn - revolution / 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 - set 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.

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 - set 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.

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 - set 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 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
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	♦		♦

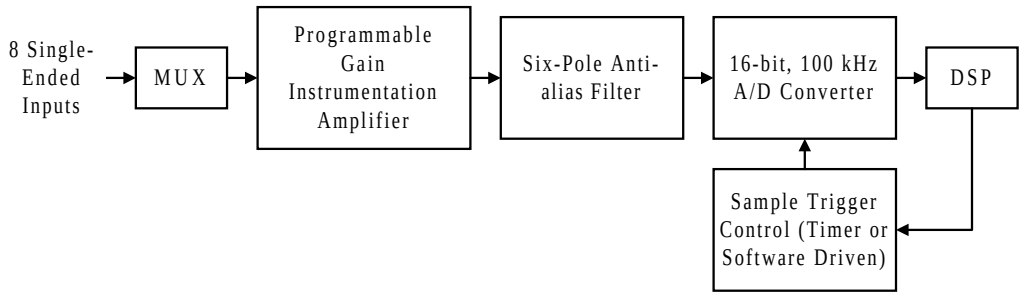
SYNTAX **xxRA**

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 (SRO)

USAGE

IMM	PGM	MIP
♦	♦	♦

SYNTAX **RQnn**

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 that 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.

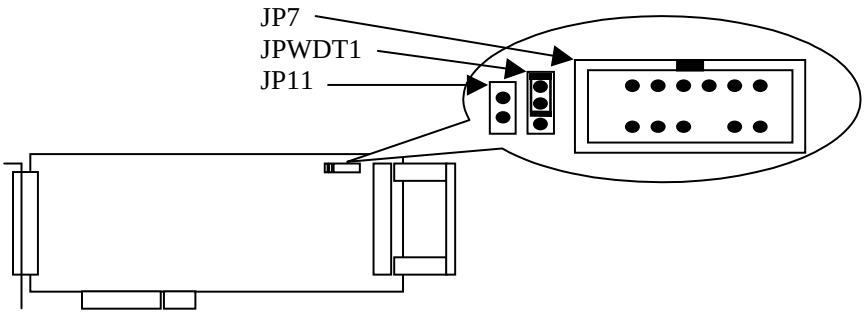
1. Stop all the axes that are in motion. The declaration value specified using the command **AG** is used to stop the axes.
2. Wait for 500 ms to allow the axes to settle.
3. Disable all the axes by turning the power OFF.
4. Reset to the controller card.

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.

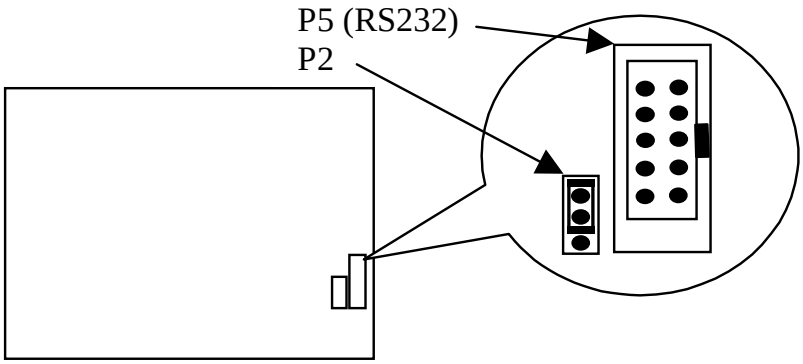
Note: This command is affective only when the watchdog timer is enabled through appropriate jumper setting on the controller card. The following figure illustrates the jumper settings to enable the watchdog timer.

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 controller*

RETURNS none

REL. COMMANDS none

EXAMPLE **RS** | *Reset the controller*

SA set device address

USAGE

IMM	PGM	MIP
♦	♦	♦

SYNTAX **Sann 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

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)

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

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

EXAMPLE

2BO?	*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

USAGE

IMM	PGM	MIP
♦	♦	♦

SYNTAX **xxSHnn** **or** **xxSH?**

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

DESCRIPTIONS

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 $\mathbf{nn_i}$ [float] - jog velocity scaling coefficients

Range $\mathbf{nn_i}$ - none

Units $\mathbf{nn_i}$ - none

Defaults $\mathbf{nn_i}$ 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 error C.

Note:
If the command is issued for an axis in motion, the new limit should not be set inside the current travel. If the motion in progress could reach the new desired software limit, the command is not accepted and the controller returns error D.

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/sec²
  ⋮
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 error C.

Note:
If the command is issued for an axis in motion, the new limit should not be set inside the current travel. If the motion in progress could reach the new desired software limit, the command is not accepted and the controller returns error D.

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
	floating point:	truncated

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** missing: error 37, AXIS NUMBER MISSING
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.

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**.

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

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 messages

	IMM	PGM	MIP
USAGE	♦		♦

SYNTAX **TB**?

PARAMETERS questions mark (?)

Defaults timeout: error 2, RS-232 COMMUNICATION TIME-OUT

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, the instantaneous real position of the specified axis.

RETURNS **nn** where: **nn** = actual position, in pre-defined units

REL. COMMANDS **TH** - read theoretical 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 notion 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**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, the instantaneous real position of the specified axis.

RETURNS **nn** where: **nn** = actual position, in pre-defined units

REL. COMMANDS **TH** - read theoretical position

EXAMPLE

```
3TP          | read real position on axis # 3
  5.322      | controller returns real position 5.322 for axis # 3
```

TX read controller activity

USAGE

IMM	PGM	MIP
♦		♦

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 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*

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: 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 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:
An important information needed when asking for help with 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 Kf. 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] - 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

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 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 to 8 units/s*
2VA6	*set axis #2 working velocity to 6 units/s*

WP wait for absolute position crossing

	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.

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.

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 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.

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

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 stored space = 61440  | controller reports available storage space
```

XX delete a stored 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
		● ● ●
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 effect 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 effect.

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*
2ZA 125H	*set amplifier I/O configuration to 125H for axis #2*
SM	*save all controller settings to non-volatile memory*

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 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
1ZB1213H	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 xx2, PARAMETER OUT OF RANGE
	critical setting:	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 defined
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 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
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 **ZZnn or ZZ?**

PARAMETERS

Description	**nn** [int]	-	system configuration
Range	**nn**	-	**0** to **OFFFFH** (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*

5.6 Commands (Binary)

5.6.1 Command Summary

Commands are categorized by function in **Table 5.6.1.**

Function	Command	Page
Initialization	esp_init_system(void)	5-215
	esp_open_system(void)	5-217
	esp_running_nt	5-218
	esp_get_num_boards	5-219
	esp_set_target_board	5-220
ASCII Command Interface	esp_send_ascii	5-221
	esp_read_ascii	5-223
	esp_send_read_ascii	5-225
General Mode Selection	esp_update_unidrive(long axis)	5-227
	esp_get_sys_config(long *config)	5-228
	esp_set_sys_fault_config(long config)	5-230
	esp_get_sys_fault_config(long config)	5-230
	esp_set_followerr_config(long axis, long config)	5-232
	esp_get_followerr_config(long axis, long *config)	5-232
	esp_set_hardlimit_config(long axis, long config)	5-234
	esp_get_hardlimit_config(long axis, long *config)	5-234
	esp_set_softlimit_config(long axis, long config)	5-236
	esp_get_softlimit_config(long axis, long *config)	5-236
	esp_set_ampio_config(long axis, long config)	5-238
	esp_get_ampio_config(long axis, long *config)	5-238
	esp_set_feedback_config(long axis, long config)	5-240
	esp_get_feedback_config(long axis, long *config)	5-240
	esp_set_e_stop_config(long axis, long config)	5-242
	esp_get_e_stop_config(long axis, long *config)	5-242
	esp_set_dac_offset(long axis, float offset)	5-244
	esp_get_dac_offset(long axis, float *offset)	5-244
	esp_set_traj_mode(long axis, long mode)	5-245
	esp_get_traj_mode(long axis, long *mode)	5-245
	esp_enable_motor(long axis)	5-247
	esp_disable_motor(long axis)	5-249
Status Functions	esp_get_hardware_status(long *hardstat1, long *hardstat2)	5-251
	esp_move_done(long axis)	5-254
	esp_home_done(long axis)	5-255

Table 5.6.1: Commands

Function	Command	Page
Status Functions (Con't.)	esp_get_motion_status(long *mstat)	5-257
	esp_get_motor_onoff_status(long *onoff)	5-258
	esp_set_position(long axis, double *position)	5-260
	esp_get_position(long axis, double *position)	5-260
	esp_get_error_num(init *error, int *ServoTick)	5-261
	esp_get_error_string(char *ErrorString, int *error, int *ServoTick)	5-262
	esp_get_version(char *FirmVer, char *DllVer)	5-264
Motion & Position Control	esp_move_absolute(long axis, double position)	5-265
	esp_move_relative(long axis, double position)	5-266
	esp_find_home(long axis, long mode)	5-267
	esp_stop(long axis)	5-269
	esp_stop_all(void)	5-270
	esp_set_position(long axis, double position)	5-271
	esp_get_position(long axis, double *position)	5-271
Trajectory Definition	esp_set_speed(long axis, float speed	5-272
	esp_get_speed(long axis, float *speed)	5-272
	esp_set_max_speed(long axis, float speed)	5-273
	esp_get_max_speed(long axis, float *speed)	5-273
	esp_set_accel(long axis, float accel)	5-274
	esp_get_accel(long axis, float *accel)	5-274
	esp_set_decel(long axis, float decel)	5-275
	esp_get_decel(long axis, float *decel)	5-275
	esp_set_max_accel(long axis, float accel)	5-276
	esp_get_max_accel(long axis, float *accel)	5-276
	esp_set_jerk(long axis, float jerk)	5-277
	esp_get_jerk(long axis, float *jerk)	5-277
	esp_set_home_speed(long axis, float speed)	5-279
	esp_get_home_speed(long axis, float *speed)	5-279
	esp_set_startstop_speed(long axis, float speed)	5-281
	esp_get_startstop_speed(long axis, float *speed)	5-281
	esp_set_jog_speed(long axis, float speed)	5-282
	esp_get_jog_speed(long axis, float *speed)	5-282
Flow Control and Sequencing	esp_delay(float time)	5-284
	esp_delay_after_stop(long axis, float time)	5-286

Table 5.6.1: Commands (Continued)

Function	Command	Page
Motion Device Parameters	esp_set_motor_type(long axis, long mtype)	5-288
	esp_get_motor_type(long axis, long *mtype)	5-288
	esp_set_master_slave(long master, long slave)	5-290
	esp_get_master_slave(long *master, long slave)	5-290
	esp_set_master_slave_ratio(long slave, float ratio)	5-292
	esp_get_master_slave_ratio(long slave, float *ratio)	5-292
	esp_get_resolution(long axis, float resolution, long units)	5-294
	esp_get_resolution(long axis, float *resolution, long *units)	5-294
	esp_set_soft_limits(long axis, double neg. double position)	5-296
	esp_get_soft_limits(long axis, double neg. double *position)	5-296
	esp_set_following_error(long axis, double error)	5-298
	esp_get_following_error(long axis, double *error)	5-298
	esp_set_microstep_factor(long axis, long resolution)	5-299
	esp_get_microstep_factor(long axis, long *resolution)	5-299
	esp_set_fullstep_resolution(float, fullstep)	5-301
	esp_get_fullstep_resolution(float, *fullstep)	5-301
	esp_set_motor_current(long axis, float current)	5-303
	esp_get_motor_current(long axis, float *current)	5-303
	esp_set_tach_constant(long axis, float tach)	5-305
	esp_get_tach_constant(long axis, float *tach)	5-305
	esp_set_gear_constant(long axis, float gear)	5-307
	esp_get_gear_constant(long axis, float *gear)	5-307
Programming	esp_save_parameters(void)	5-309
Digital Filters	esp_set_kp(long axis, float kp)	5-311
	esp_get_kp(long axis, float *kp)	5-311
	esp_set_kd(long axis, float kd)	5-313
	esp_get_kd(long axis, float *kd)	5-313
	esp_set_ki(long axis, float ki)	5-315
	esp_get_ki(long axis, float *ki)	5-315
	esp_set_il(long axis, float il)	5-317
	esp_get_il(long axis, float *il)	5-317
	esp_set_vel_feedforward(long axis, float vff)	5-319
	esp_get_vel_feedforward(long axis, float *vff)	5-319
	esp_set_acc_feedforward(long axis, float aff)	5-321
	esp_get_acc_feedforward(long axis, float *aff)	5-321
	esp_update_filter(void)	5-323

Table 5.6.1: Commands (Continued)

Functions	Command	Page
I/O Functions	esp_set_adc_gain(long gain)	5-324
	esp_get_adc_gain(long *gain)	5-324
	esp_set_adc_range(long range)	5-326
	esp_get_adc_range(long *range)	5-326
	esp_get_adc(long channel, Float *conversion, long *TickCnt)	5-328
	esp_get_all_adc(float *channel, long *TickCnt)	5-330
	esp_set_daq_mode(long mode, long axis, long Adcs, long feedback, long rate, long Num)	5-332
	esp_enable_daq(void)	5-335
	esp_disable_daq	5-337
	esp_get_daq_status(long *count)	5-339
	esp_daq_done( )	5-341
	esp_get_daq_data(long *DataArray, long *Num, long *DaqStat)	5-343
	esp_daq_data_to_file	5-345
	esp_set_portabc_dir(long PortA, longPortB, long PortC)	5-346
	esp_get_portabc_dir(long *PortA, long *PortB, long *PortC)	5-346
	esp_set_dio_porta(long data)	5-347
	esp_get_dio_porta(long *data)	5-347
	esp_set_dio_portb(long data)	5-348
	esp_get_dio_portb(long *data)	5-348
	esp_set_dio_portc(long data)	5-349
	esp_get_dio_portc(long *data)	5-349
	esp_set_dio_inhibit	5-350
	esp_get_dio_inhibit	5-350
	esp_set_dio_inhibit_cfg	5-351
	esp_get_dio_inhibit_cfg	5-351
	esp_set_dio_motion	5-353
	esp_get_dio_motion	5-353
	esp_set_dio_motion_cfg	5-354
	esp_get_dio_motion_cfg	5-354

Table 5.6.1: Commands (Continued)

5.6.2 Command List

Commands are provided in the following pages.

Initialization

- Commands start on Page 5-215.

ASCII Command Interface

- Commands start on Page 5-221.

General Mode Selection

- Commands start on Page 5-227.

Status Functions

- Commands start on Page 5-251.

Motion & Position Control

- Commands start on Page 5-265.

Trajectory Definition

- Commands start on Page 5-272.

Flow Control and Sequencing

- Commands start on Page 5-284.

Motion Device Parameters

- Commands start on Page 5-288.

Programming

- Commands start on Page 5-309.

Digital Filters

- Commands start on Page 5-311.

I/O Functions

- Commands start on Page 5-324.

Initialization

esp_init_system *Initialize ESP6000 and PCI Communications*

Synopsis:
```
#include "esp6000.h"
int esp_init_system(void)
```

Arguments: none

Library Location: \esp6000.dll

Description: **esp_init_system()** initializes the ESP6000 controller internals and PCI bus communication.

NOTE

The first time new stages are connected, the esp_init_system() API function call may take as long as 20 seconds to completely upload and parse all ESP-compatible stage data and configure UniDrive axes.

The esp_init_system( ) API function must be the first ESP function called in any application.

All motors are turned OFF and ESP6000 hardware is reset after an **esp_init_system( )** function call.

Returns: ESPOK, ESPERROR

Hint: Always evaluate returned value from `esp_init_system()`

Usage Example:
```
# include <stdio.h>
# include <stdlib.h>
# include "esp6000.h"

void tell_error(int esp_error)
{
char buffer[MAX_ERROR_LEN];

switch (esp_error)
{
case ESP_OK: /* ESP6000 and communications OK */
break;

default:
error_msg(error_code, buffer) ;
fprintf(stderr, "ERROR: %s
(%d).\n", buffer,
```

```
esp_error);
exit(1);
break;
}
}

int main()
{
int esp_error;
esp_error = esp_init_system();    /* initialize
ESP6000 */
tell_error(esp_error);       /* report errors (if
any) */
return (0);
}
```

See Also:

esp_open_system *Initialize PCI Communications*

Synopsis: #include “esp6000.h”
int esp_open_system(void)

Arguments: None

Library Location: \esp6000.dll

Description: **esp_open_system()** initializes the ESP6000 PCI bus communication, but does not invoke a board hardware reset.

Returns: ESPOK, ESPERROR

Hint: Always evaluate returned value from `esp_open_system()`

Usage Example:

```
# include <stdio.h>
# include <stdlib.h>
# include “esp6000.h”

void tell_error(int esp_error)
{
char buffer[MAX_ERROR_LEN];

switch (esp_error)
{
case ESP_OK: /* ESP6000 and communications OK */
break;

default:
error_msg(error_code, buffer) ;
fprintf(stderr, “ERROR: %s
(%d).\n”, buffer,
esp_error);
exit(1);
break;
}
}


int main()
{
int esp_error;

esp_error = esp_open_system();    /* initialize
ESP6000 */
tell_error(esp_error);  /* report errors (if any)
*/

return (0);
}
```

See Also: esp_init_system( )

esp_running_nt *Determine Which Operating System Is Running*

Synopsis: #include "esp6000.h"
int esp_running_nt (void)

Arguments: none

Library Location: \esp6000.dll

Description: esp_running_nt( ) queries the computer to determine if the operating system is Windows NT

Returns: 1 if the operating system is Windows NT or Windows 2000

0 if the operating system is Windows 95 or Windows 98

Hint: Multi-board support is offered only in Windows NT. The user should use this function if they are developing their application for both Windows NT and Windows 95

Usage Example:

```
#include "esp6000.h"

Main ()
{
long Boards, BdUsed;

if (esp_running_nt ())
{

esp_get_num_board (&Boards,
&BdsUsed);
     if (Boards == 2 && BdsUsed == 0)
     { // two board found. Init both of
          them
     esp_set_target_board(0);
     esp_init_system();
     esp_set_target_board(1);
     esp_init_system();
     }
}
```

See Also: esp_get_num_boards( ), esp_set_target_board( )

esp_get_num_boards *Determine How Many ESP6000 Boards are Installed In System*

Synopsis: #include "esp6000.h"
int esp_get_num_boards (long &Boards, long &BdsUsed)

Arguments: long Boards – number of boards installed in system
Long BdsUsed – The number of boards being used by another application

Library Location: esp_running_nt( ) queries the system to determine how Many esp6000 boards are installed in the computer and How many of those boards are presently being used by another application

Returns: ESPOK, ESPERROR

Hint: Multi-board support is offered only in Windows NT.

User Example:

```
#include "esp6000.h"

Main()
{
long Boards, BdUsed;

if (esp_running_nt())
{
esp_get_num_board(&Boards,
&BdsUsed);
if (Boards == 2 && BdsUsed == 0)
  {  // two board found. Init both of
        them
     esp_set_target_board(0);
     esp_init_system();
     esp_set_target_board(1);
     esp_init_system();
  }

}
```

See Also: esp_running_nt(), esp_set_target_board()

esp_set_target_board *Set Communication to the Specified ESP6000 Board*

Synopsis: #include "esp6000.h"
int esp_set_target_board(long target)

Arguments: long target – The ESP6000 board the user wishes to communicate to.

Library Location: \esp6000.dll

Description: If more than one ESP6000 board is installed in the user's system, they can use this command to specify which board they want to command.

Returns: ESPOK, ESPERROR

Hint: Multi-board support is offered only in Windows NT.

Usage Example:

```
#include "esp6000.h"

main()
long Boards, BdUsed;
{
if (esp_running_nt())
{
esp_get_num_board(&Boards,
&BdsUsed);
     if(Boards == 2 && BdsUsed == 0)
     { // two board found. Init both of
               them
          esp_set_target_board(0);
          esp_init_systme();
          esp_set_target_board(1);
          esp_init_system();

     }

}
```

See Also: esp_running_nt(), esp_get_num_boards()

ASCII Command Interface

esp_send_ascii *Send ASCII Command via Binary Interface*

Synopsis: #include "esp6000.h"
int esp_send_ascii(LPSTR string);

Arguments: char string
Any valid ASCII command

Library Location: \esp6000.dll

Description: esp_send_ascii( ) sends ASCII commands to the motion controller via binary interface.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include <stdlib.h>
#include <string.h>
#include "esp6000.h"

main()
{
static char cmdStr [255] = 0;
static char rspStr [255] = 0;
long mtnDone;

if (!esp_init_system())
{
    printF ("ESP6000 Not Initialized! \r\n");
    exit (-1);
}

/* Enable Axis 2 Motor Power */
strcpy (cmdStr, "2MO\r");
esp_send_ascii (cmdStr);

/* Set Axis 2 Home Speed */
```

```
strcpy (cmdStr, "20H20.0\r");

esp_send_ascii (cmdStr);

/* Begin Home Search on Axis 2 */
strcpy (cmdStr, "2OR1\r");
esp_send_ascii (cmdStr);

/* Wait For Home Search Completion */
while (!mtnDone)
{
    strcpy (cmdStr, "2MD?\r");
    esp_send_ascii (cmdStr);
    esp_read_ascii (rspStr);
    mtnDone = atoi (rspStr);
}

/* Check Error Status */
strcpy (cmdStr, "TB\r");
esp_send_ascii (cmdStr);
esp_read_ascii (rspStr);
}
```

See Also: esp_read_ascii( )
esp_send_read_ascii( )

esp_read_ascii *Read Response to an ASCII Command*

Synopsis: #include "esp6000.h"
int esp_read_ascii(char * string);

Arguments: char string

Library Location: \esp6000.dll

Description: esp_read_ascii( ) reads the response to an ASCII command. This function may be called only in case of queries (commands that have responses associated with them).

Returns: ESPOK, ESPERROR

Hint: esp_send_ascii( ) has to be called with a valid ASCII query prior to calling this function.

Usage Example:

```
#include <stdlib.h>
#include <string.h>
#include "esp6000.h"

main()
{
static char cmdStr [255] = 0;
static char rspStr [255] = 0;
long mtnDone;

if (!esp_init_system())
{
    pintf ("ESP6000 Not Initialized! \r\n");
    exit (-1);
}

/* Enable Axis 2 Motor Power */
strcpy (cmdStr, "2MO\r");
esp_send_ascii (cmdStr);

/* Set Axis 2 Home Speed */
strcpy (cmdStr, "2OH20.0\r");
esp_send_ascii (cmdStr);
```

```
/* Begin Home Search on Axis 2 */
strcpy (cmdStr, "2OR1\r");
esp_send_ascii (cmdStr);

/* Wait For Home Search Completion */
while (!mtnDone)
{
    strcpy (cmdStr, "2MD?\r");
    esp_send_ascii (cmdStr);
    esp_read_ascii (rspStr);
    mtnDone = atoi (rspStr);
}

/* Check error Status */
strcpy (cmdStr, "TB\r");
esp_send_ascii (cmdStr);
esp_read_ascii (rspStr);
}
```

See Also: esp_send_ascii( )
esp_send_read_ascii( )

esp_send_read_ascii *Send ASCII Command via Binary Interface and Obtain Response*

Synopsis: #include "esp6000.h"
int esp_send_read_ascii(char Command, char *Response);

Arguments: char Command
Any valid ASCII command
Char Response
Response to the ASCII command

Library Location: \esp6000.dll

Description: esp_send_read_ascii( ) sends ASCII commands to the motion controller via binary interface and reads responses from the controller. Note that this command is intended only for query (get) commands for which there are valid responses. If a non-query (set) command is issued using this function, the controller responds back with "No data available". For such commands, "esp_send_ascii( )" is recommended.

Returns: ESPOK, ESPERROR

Hint:

User Example:

```
#include <stdlib.h>
#include <string.h>
#include "esp6000.h"

main( )
{
static char cmdStr[255] = 0;
static char rspStr[255] = 0;
long mtnDone;

if (!esp_init_system( ) )
{
  printf ("ESP6000 Not Initialized! \r\n");
  exit (-1);
}

/* Enable Axis 2 Motor Power */
strcpy (cmdStr, "2MO\r");
esp_send_ascii (cmdStr);

/* Set Axis 2 Home Speed */
strcpy (cmdStr, "2OH20.0\r")
```

```
esp_send_ascii (cmdStr);
/* Begin Home Search on Axis 2 */
strcpy (cmdStr, "2OR1\r");
esp_send_ascii (cmdStr);

/* Wait For Home Search Completion */
while (!mtnDone)
{
  strcpy (cmdStr, "2MD?\r")
  esp_send_read_ascii (cmdStr, rspStr);
  mtnDone = atoi (rspStr);
}

/* Check Error Status */
strcpy (cmdStr, "TB\r");
esp_send_read_ascii (cmdStr, rspStr);
}
```

See Also: esp_send_ascii( )
esp_read_ascii( )

General Mode Selection

esp_update_unidrive *Update UniDrive Settings*

Synopsis: #include “esp6000.h”
int esp_update_unidrive(long axis)

Arguments: long axis
axis number from 1-6

Library Location: \esp6000.dll

Description: **esp_update_unidrive()** API function call resends all UniDrive6000 settings using the most recent motor configuration for the specified axis.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”
main()
{
long error, servotick;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* Change Axis-1 Motor Current */
esp_motor_current(1, 1.9);

/* Update Unidrive6000 Axis-1 */
esp_update_unidrive(1);

/* Save new settings to non-volatile memory */
esp_save_parameters();

/* check error status */
esp_get_error_num(&error,&ServoTick) ;
if (error) printf(“Error %d Reported!\r\n”,
error);
}
```

See Also: esp_motor_current( ), esp_save_parameters( )

esp_get_sys_config *Report ESP System Configuration*

Synopsis: #include “esp6000.h”
int esp_get_sys_config(long *config)

Arguments: long config
configuration register

Library Location: \esp6000.dll

Description: **esp_get_sys_config()** reports the present ESP system stage/driver configuration. After each system reset or initialization the ESP6000 detects the presence of UniDrive6000 driver channels and ESP-compatible stages connected.

BIT#	VALUE	DEFINITION
0	0	axis-1 UniDrive6000 not detected
0	1	xis-1 UniDrive6000 detected
1	0	axis-2 UniDrive6000 detected
2	0	axis-3 UniDrive6000 not detected
2	1	axis-3 UniDrive6000 detected
3	0	axis-4 UniDrive6000 not detected
3	1	axis-4 UniDrive6000 detected
4	0	axis-5 UniDrive6000 not detected
4	1	axis-5 UniDrive6000 detected
5	0	axis-6 UniDrive6000 not detected
5	1	axis-6 UniDrive6000 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

BIT#	VALUE	DEFINITION
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: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include "esp6000.h"

main()
{
long error, servotick, systconfig;

if (!esp_init_system())
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* Get ESP System Configuration */
esp_get_sys_config(&sysconfig);

/* print present system configuration */
printf("Configuration = %x \r\n", sysconfig);
}
```

See Also:

esp_set_sys_fault_config — *Set System Configuration Register*
esp_get_sys_fault_config — *Report System Configuration Register*

Synopsis: #include “esp6000.h”
int esp_set_sys_fault_config(long config)
int esp_get_sys_fault_config(long *config)

Arguments: long config
configuration register

Library Location: \esp6000.dll

Description: **esp_set_sys_fault_config()** is used to configure system fault checking,
event handling, and general setup for all axes.

esp_get_sys_fault_config() reports present setting.

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 interlock event
1	1	disable all axes on interlock 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 channel 7 & 8
7	1	route axis 1 & 2 encoder feedback to counter channel 7 & 8
8	0	unprotect ESP system-critical settings
8	1	protect ESP system-critical settings
9	0	reserved
9	1	reserved
10	0	reserved
10	1	reserved
11	0	reserved
11	1	reserved
12	0	reserved

12	1	reserved
13	0	reserved
13	1	reserved
14	0	reserved
14	1	reserved
15	0	reserved
15	1	reserved
	•	
	•	
	•	
31	0	reserved
31	1	reserved

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include "esp6000.h"

main()
{
if (!esp_init_system())
  {
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* Disable Motor Power & Flag Interlock Error */
status = esp_set_sys_fault_config(0x0b);
}
```

See Also:

esp_set_followerr_config *Set Following Error Configuration Register*
esp_get_followerr_config *Report Following Error Configuration Register*

Synopsis: #include “esp6000.h”
int esp_set_followerr_config(long axis, long config)
int esp_get_followerr_config(long axis, long *config)

Arguments: long axis
axis number from 1 to 6
long config
configuration register

Library Location: \esp6000.dll

Description: **esp_set_followerr_config()** is used to configure the motor following error
checking and event handling for the specified axis.

esp_get_followerr_config() reports the present configuration.

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

Returns: ESPOK, ESPERROR

Hint:

Usage Example:
```
#include “esp6000.h”
```

```
main()
{
if (!esp_init_system())
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* Disable Motor Power On Following Error */
esp_set_followerr_config(1,0x03);
}
```

See Also: esp_set_following_error()

esp_set_hardlimit_config — *Set Hardware Limit Configuration Register*
esp_get_hardlimit_config — *Report Hardware Limit Configuration Register*

Synopsis: #include “esp6000.h”
int esp_set_hardlimit_config(long axis, long config)
int esp_get_hardlimit_config(long axis, long *config)

Arguments: long axis
axis number from 1 to 6
long config
configuration register

Library Location: \esp6000.dll

Description: **esp_set_hardlimit_config()** is used to configure the hardware travel limit checking and event handling for the specified axis.

esp_get_hardlimit_config() reports present register setting.

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

Returns: ESPOK, ESPERROR

Hint: Newport stages use “active high” travel limit input setting.

Usage Example:
```
#include "esp6000.h"
```

```
main()
{
if (!esp_init_system())
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* Abort Motion & Flag Error On Hardware Limit */
esp_set_hardlimit_config(1,0x0d);
}
```

See Also:

esp_set_softlimit_config *Set Software limit Configuration Register*
esp_get_softlimit_config *Report Software Limit Configuration Register*

Synopsis: #include “esp6000.h”
int esp_set_softlimit_config(long axis, long config)
int esp_get_softlimit_config(long axis, long *config)

Arguments: long axis
axis number from 1 to 6
long config
configuration register

Library Location: \esp6000.dll

Description: **esp_set_softlimit_config()** is used to configure the software travel limit checking and event handling for the specified axis.

esp_get_softlimit_config() reports present register setting.

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

Returns: ESPOK, ESPERROR

Hint:

Usage Example:
```
#include “esp6000.h”
```

```
main()

{
if (!esp_init_system())
{   printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* Abort Motion & Flag Error On Software Limit */
esp_set_softlimit_config(1,0x0d);
}
```

See Also:

esp_set_ampio_config *Set Amplifier I/O Configuration Register*
esp_get_ampio_config *Report Amplifier I/O Configuration Register*

Synopsis: #include “esp6000.h”
int esp_set_ampio_config(long axis, long config)
int esp_get_ampio_config(long axis, long *config)

Arguments: long axis
axis number from 1 to 6
long config
configuration register

Library Location: \esp6000.dll

Description: **esp_set_ampio_config()** is used to configure the amplifier I/O polarity, fault checking, and event handling for the specified axis.

esp_get_ampio_config() reports the present setting.

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	reserved
11	1	reserved
12	0	reserved
12	1	reserved
13	0	reserved
13	1	reserved
14	0	reserved
14	1	reserved
15	0	reserved
15	1	reserved
16	0	reserved
16	1	reserved
17	0	reserved
17	1	reserved
18	0	reserved
18	1	reserved
	•	
	•	
	•	
31	0	reserved
31	1	reserved

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include "esp6000.h"

main()
{
if (!esp_init_system())
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* Disable Motor & Flag Error On Amp Fault */
esp_set_ampio_config(1,0x0b);
}
```

See Also:

esp_set_feedback_config — *Set Amplifier Input Configuration Register*
esp_get_feedback_config — *Report Amplifier Input Configuration Register*

Synopsis:
#include “esp6000.h”
int esp_set_feedback_config(long axis, long config)
int esp_get_feedback_config(long axis, long *config)

Arguments:
long axis
axis number from 1 to 6
long config
configuration register

Library Location: \esp6000.dll

Description: **esp_set_feedback_config()** is used to configure the feedback checking and event handling for the specified axis.

esp_get_feedback_config() reports present register setting.

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 stepper positioning
8	1	use encoder feedback for stepper positioning
9	0	reserved
9	1	reserved
10	0	reserved
10	1	reserved
11	0	reserved
11	1	reserved
12	0	reserved
12	1	reserved
13	0	reserved

13	1	reserved
14	0	reserved
14	1	reserved
15	0	reserved
15	1	reserved
	•	
	•	
	•	
31	0	reserved
31	1	reserved

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include "esp6000.h"

main()
{
if (!esp_init_system())
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* Set Flag On Feedback Error */
esp_set_feedback_config(1,0x09);
}
```

See Also:

esp_set_e_stop_config *Set Emergency Stop Configuration Register*
esp_get_e_stop_config *Report Emergency Stop Configuration Register*

Synopsis: #include “esp6000.h”
int esp_set_e_stop_config(long axis, long config)
int esp_get_e_stop_config(long axis, long *config)

Arguments: long axis
axis number from 1 to 6
long config
configuration register

Library Location: \esp6000.dll

Description: **esp_set_e_stop_config()** is used to configure the emergency stop checking and event handling for the specified axis.

esp_get_e_stop_config() reports present setting.

BIT#	VALUE	DEFINITION
0	0	disable E-Stop checking
0	1	enable E-Stop checking
1	0	do not disable power motor 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

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
if (!esp_init_system())
{
```

```
            printf("ESP6000 Not Initialized! \r\n");
            exit(-1);
            }

            /* Abort Motion On E-STOP Event */
            esp_set_e_stop_config(1,0x05);
            }
```

See Also:

esp_set_dac_offset *Set Axis DAC Offset Compensation*
esp_get_dac_offset *Report Axis DAC Offset Compensation Setting*

Synopsis:
#include "esp6000.h"
int esp_set_dac_offset(long axis, float offset)
int esp_get_dac_offset(long axis, float *offset)

Arguments:
long axis
axis number from 1-6
float offset
DAC offset from +1.0 volt to –1.0 volt

Library Location:
\esp6000.dll

Description:
esp_set_dac_offset( ) defines DAC offset compensation for the specified axis. DAC offset takes affect immediately after the command is processed by the DSP.

Save DAC offset to non-volatile flash memory with the **save_parameters( )** command. This will cause the DSP to automatically use this value after system reset or reboot.

esp_get_dac_offset( ) reports the present DAC offset compensation used for the specified axis.

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

Returns:
ESPOK, ESPERROR

Hint:

Usage Example:
```
# include "esp6000.h"
main ( )
{
if (!esp_init_system( ) )
{
print ("ESP6000 Not Initialized! \r\n");
exit (-1);
}

/* Set Axis 1 DAC Offset */
esp_set_dac_offset (1, -0.075);

/* Save Offset To Non-volatile memory */
esp_save_parameters ( );esp_save_parameters( )
```

See Also:
esp_save_parameters( )

esp_set_traj_mode *Set Axis Trajectory Mode*
esp_get_traj_mode *Report Axis Trajectory Mode Setting*

Synopsis: #include “esp6000.h”
int esp_set_traj_mode(long axis, long mode)
int esp_get_traj_mode(long axis, long *mode)

Arguments: long axis
axis number from 1-6
long mode
possible trajectory modes are: `TRAPEZOID, TRAPSTEP, SCURVE, SLAVEP, SLAVET, JOG`

Library Location: \esp6000.dll

Description: `esp_set_traj_mode()` defines the trajectory mode for the specified axis.

`esp_get_traj_mode()` reports the present trajectory mode setting for the specified axis.

`TRAPEZOID` - sets the axis into classical trapezoidal motion profile mode.

`TRAPSTEP` - sets stepper motor axis into trapezoidal motion profile mode with start/stop speed implementation to avoid inherent stepper motor resonant frequencies.

`SCURVE` - sets the axis into S-curve motion profile mode. This helps provide smoother, jerk-free motion when properly tuned. This mode is not available for stepper motor axes.

`JOG` - sets the axis into jog mode. When an axis has been placed in jog mode and
the axis is enabled, motion is started by setting the jog velocity with the `esp_set_jog_speed()` function. The axis will then accelerate to this specified velocity and continue until a stop command is issued, or a new velocity is specified.

`SLAVEP` - sets an axis into slave mode with respect to the master’s position feedback.

`SLAVET` - sets an axis into slave mode with respect to the master’s ideal trajectory output.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include "esp6000.h"

main()
{
long error, servotick;

if (!esp_init_system())
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* set S-Curve trajectory mode */
esp_set_traj_mode(1, SCURVE);

/* set axis 1 trajectory parameters */
esp_set_speed(1, 30.0);
esp_set_accel(1, 200.0);
esp_set_decel(1, 150.0);
esp_set_jerk(1, 10.0);

/* check error status */
esp_get_error_num(&error,&ServoTick) ;
if (error) printf("Error %d Reported!\r\n",
error);
}
```

See Also:

esp_enable_motor *Enable Motor Power*

Synopsis: #include “esp6000.h”
int esp_enable_motor(long axis)

Arguments: long axis
axis number from 1-6

Library Location: \esp6000.dll

Description: **esp_enable_motor()** enables motor power to specified axes. After this API call DC motors will servo on target position and stepper motors will have torque applied.

NOTE

The AMP ENABLE signal is set TRUE on the 100-pin motor I/O connector.

All axes are automatically disabled after system initialization or reset.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;
double position;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* enable motor power */
esp_enable_motor(2);

/* move axis 2 to absolute position -3.0 */
esp_move_absolute(2,-3.0);

while (!esp_move_done(2));

/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf(“Error %d Reported!”, error);
```

```
/* disable motor power */
esp_disable_motor(2);
}
```

See Also: esp_disable_motor(), esp_get_motor_onoff_status()

esp_disable_motor *Disable Motor Power*

Synopsis: #include “esp6000.h”
int esp_disable_motor(long axis)

Arguments: long axis
axis number from 1-6, axis = 0 disables all axes

Library Location: \esp6000.dll

Description: **esp_disable_motor()** disables motor power to specified axis. If axis parameter = 0 then all axes are disabled.

NOTE

The AMP ENABLE signal is set FALSE on the 100-pin motor I/O connector.

All axes are automatically disabled after system initialization or reset.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;
double position;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* enable motor power */
esp_enable_motor(2);

/* move axis 2 to absolute position -3.0 */
esp_move_absolute(2,-3.0);

while (!esp_move_done(2));

/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf(“Error %d Reported!”, error);
```

```
/* disable motor power */
esp_disable_motor(2);
}
```

See Also: esp_enable_motor()

Status Functions

esp_get_hardware_status *Report Hardware Status For All Axes*

Synopsis:

include "esp6000.h"

int esp_get_hardware_status(long *hardstat1, long *hardstat2)

Arguments:

long hardstat1

general hardware status register 1

long hardstat2

general hardware status register 2

Library Location:

\esp6000.dll

Description: (Register #1)

esp_get_hardware_status( ) is used to get general hardware status for all axes. This routine allows the 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 connector emergency stop (latched) low
30	1	auxiliary I/O connector 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 axis 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: ESPOK, ESPERROR

Hint:

Usage Example:

```
# include "esp6000.h"

main ( )
( long stat1, stat2 ;

if (! Esp_init_system( ) ) exit ( -1);

/* Get Board Hardware Status */
esp_get_hardware_status (&stat1, &stat2);

printf ("Hardware Status = %x, %x /n/r" ,
stat1, stat2)
)
```

See Also:

esp_move_done *Return Move Completion Status*

Synopsis: #include “esp6000.h”
int esp_move_done(long axis)

Arguments: long axis
axis number from 1-6, axis=0 tests for all axes

Library Location: \esp6000.dll

Description: **esp_move_done()** returns the moving/not moving status of the specified axis 1 - 6.
If axis parameter = 0 then the returned value is the combined logical OR of all axes.

This function returns `0` (ESPNOTDONE) while the axis has not completed the move
and a `1` (ESPDONE) when finished.

Returns: ESPDONE, ESPNOTDONE

Hint:

Usage Example:

```
#include “esp6000.h”
main()
{
long error, servotick;
double position;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}
/* enable motor power */
esp_enable_motor(2);

/* move axis 2 to absolute position -3.0 */
esp_move_absolute(2,-3.0);

while (!esp_move_done(2));

/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf(“Error %d Reported!\r\n”,
error);
}
```

See Also: esp_home_done( )

esp_home_done *Return Home Search Completion Status*

Synopsis: #include “esp6000.h”
bool esp_home_done(long axis)

Arguments: long axis
axis number from 1-6, axis = 0 tests for all axes

Library Location: \esp6000.dll

Description: **esp_home_done()** returns the home search status of the specified axis 1-6. If axis parameter = 0 then the returned value is the combined logical OR of all axes.

This function returns ESPNOTDONE (0) while the axis is homing. ESPDONE (1)
is returned when the axis is no longer in homing mode.

Returns: ESPDONE, ESPNOTDONE

Hint: Query for possible motion/system errors after homing.

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* Enable Axis 2 Motor Power */
esp_enable_motor(2);

/* Set Axis Home Speed */
esp_set_home_speed(2,20.0);

/* Begin Home Search On Axis 2 */
esp_find_home(2,1);

/* Wait For Home Search Completion */
while (!esp_home_done(2));

/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf(“Error %d Reported!\r\n”,
error);
}
```

See Also: esp_set_home_speed(), esp_find_home()

esp_get_motion_status *Report Motion Status For All Axes*

Synopsis: #include “esp6000.h”
int esp_get_motion_status(long *mstat)

Arguments: long mstat
motion status register

Library Location: \esp6000.dll

Description: **esp_get_motion_status()** is used to report motion status for all axes.

esp_get_motion_status() API call reports all axes’ motion status in binary format where axis-1 corresponds to bit-0 and axis-6 bit-5. If the corresponding bit is equal to ‘0’ then the axis is not being commanded to move. If the corresponding bit is equal to ‘1’ then the axis is in a move sequence.

NOTE

If motor type not previously defined then the corresponding status bit will equal zero (0).

BIT#	VALUE	DEFINITION
0	0	axis 1 positioner not moving
0	1	axis 1 positioner moving
1	0	axis 2 positioner not moving
1	1	axis 2 positioner moving
2	0	axis 3 positioner not moving
2	1	axis 3 positioner moving
3	0	axis 4 positioner not moving
3	1	axis 4 positioner moving
4	0	axis 5 positioner not moving
4	1	axis 5 positioner moving
5	0	axis 6 positioner not moving
5	1	axis 6 positioner moving
6	0	reserved
6	0	reserved
7	0	no error occurred (error buffer empty)
7	1	error occurred (error buffer contains message)
	•	
	•	
	•	
31	0	reserved
31	1	reserved

Returns: ESPOK, ESPERROR

Hint: Use **esp_move_done()** for simple “Motion Complete” testing.

esp_get_motor_onoff_status *Report Motor ON/OFF Status*

Synopsis: #include “esp6000.h”
int esp_get_motor_onoff_status(long *onoff)

Arguments: long *onoff
motor ON/OFF status where bits 0 - 5 correspond to axes 1-6

Library Location: \esp6000.dll

Description: `esp_get_motor_onoff_status()` reports all axes motor on/off status in binary format where axis-1 corresponds to bit-0 and axis-6 bit-5. If the corresponding bit is equal to ‘0’ then the axis is OFF (disabled). If the corresponding bit is equal to ‘1’ then the motor is ON (enabled).

NOTE

If motor type not previously defined then the corresponding status bit will equal zero (0).

BIT#	VALUE	DEFINITION
0	0	axis-1 motor not enabled
0	1	axis-1 motor enabled
1	0	axis-2 motor not enabled
1	1	axis-2 motor enabled
2	0	axis-3 motor not enabled
2	1	axis-3 motor enabled
3	0	axis-4 motor not enabled
3	1	axis-4 motor enabled
4	0	axis-5 motor not enabled
4	1	axis-5 motor enabled
5	0	axis-6 motor not enabled
5	1	axis-6 motor enabled
6	0	reserved
6	1	reserved
7	0	reserved
7	1	reserved
	•	
	•	
	•	
31	0	reserved
31	1	reserved

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

  main()
{
 long error, servotick, onoff;
```

```
if(!esp_init_system())
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

 /* enable motor power */
esp_enable_motor(2);

 esp_get_motor_onoff_status(&onoff);

/* test for axis 2 enabled */
 if(onoff& 0x02)
{
 printf("Axis 2 Motor Enabled! \r\n");
}
}
```

See Also: esp_disable_motor(), esp_enable_motor()

esp_set_position — *Set Position Count To Specified Value*
esp_get_position — *Report Position Count*

Synopsis: #include “esp6000.h”
int esp_set_position (long axis, double count)
int esp_get_position (long axis, double *count)

Arguments: long axis
Axis number from 1-8. Channels 7 & 8 are auxiliary counters.
double count
position count or steps in user units

Library Location: \esp6000.dll

Description: **esp_get_position()** reports the current position count or steps for the specified axis.
If bit-8 of **esp_set_feedback_config()** command is set then **esp_get_position()** reports encoder feedback in user units. If bit-8 is 0 and the axis motor type is stepper then **esp_get_position()** reports step count in user units.

esp_set_position() sets the current position count to the value specified for the selected axis.

Returns: ESPOK, ESPERROR

Hint: Position count is normally set to zero (0) after a home search.

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick; double position;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* report axis 1 position count */
esp_get_position(1,&position);
  • • •
/* zero axis 1 position count */
esp_set_position(1, 0.0);

}
```

See Also: esp_find_home()

esp_get_error_num *Report Error Number*

Synopsis: #include “esp6000.h”
int esp_get_error_num(long *error, long *ServoTick)

Arguments: long error
error number
long ServoTick
servo cycle timestamp (409µsec)

Library Location: \esp6000.dll

Description: **esp_get_error_num()** API function call reports ESP6000 error messages complete with error number and timestamp. The ESP6000 queues error messages in a 10-word FIFO buffer. The timestamp enables users to know the exact time of error posting.

NOTE
ESP6000 uses a 10-word FIFO buffer to queue error message.

Returns: ESPOK, ESPERROR

Hint: Check for errors after critical command sequences.

Usage Example:

```
#include “esp6000.h”
main()
{
long error, servotick;
double position;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* enable motor power */
esp_enable_motor(2);

/* move axis 2 to absolute position -3.0 */
esp_move_absolute(2,-3.0);
while (!esp_move_done(2));

/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf(“Error %d Reported!”, error);
}
```

See Also: esp_get_error_string( )

esp_get_error_string *Report Error String*

Synopsis: #include “esp6000.h”
int esp_get_error_string(char *ErrorStr, long *ErrorNum, long *ServoTick)

Arguments: char ErrorStrg
string containing error message
long ErrorNum
error number
long ServoTick
servo cycle timestamp with 409µsec. resolution

Library Location: \esp6000.dll

Description: **esp_get_error_string()** API function call reports ESP6000 error messages complete with error string, error number, and timestamp. The ESP6000 queues error messages in a 10-word FIFO buffer. The timestamp enables users to know the exact time of error posting.

NOTE
ESP6000 uses a 10-word FIFO buffer to queue error messages.

Returns: ESPOK, ESPERROR

Hint: Check for errors after critical command sequences.

Usage Example:

```
#include “esp6000.h”
main()
{
long error, ServoTick;
double position;
char String[100];

if (!esp_init_system()) exit(-1);

esp_enable_motor(2);          /* enable motor */
esp_move_absolute(2,-3.0);    /* move stage */

while (!esp_move_done(2));    /* wait for move
done */

/* prime the loop */
esp_get_error_string(String, &error, &ServoTick);
if(error == 0) return(0);
else printf(“Error: %s \r\n”, String);

while(error > 0)
```

```
{
esp_get_error_string(String, &error, &TickCount);
printf("Error: %s \r\n", String);
}
}
}
```

See Also: esp_get_error_num()

esp_get_version *Report ESP6000 Firmware and DLL Version*

Synopsis: #include “esp6000.h”
int esp_get_version(char *FirmwareVer, char *DllVer)

Arguments: char *FirmwareVer
pointer to start of character string containing firmware version number
char *DllVer
pointer to start of character string containing DLL version number

Library Location: \esp6000.dll

Description: **esp_get_version()** reports the ESP60000 firmware and dynamic link library (DLL) version.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”
int status;
char FirmwareVer[20];
char DllVer[20];

if (esp_init_system() )
{
/* Read Axis-1 SmartStage Data */
status = esp_get_version(FirmwareVer, DllVer);
}
```

See Also:

Motion & Position Control

esp_move_absolute *Move to Absolute Position*

Synopsis: #include "esp6000.h"
int esp_move_absolute(long axis, double position)

Arguments: long axis
axis number from 1-6
double position
target absolute position in user units

Library Location: \esp6000.dll

Description: **esp_move_absolute()** will move the specified axis to the absolute motor position
as referenced to position count zero (0).

Returns: ESPOK, ESPERROR

Hint: Remember to specify position parameter in user units (e.g., millimeters)

Usage Example:

```
#include "esp6000.h"
main()
{
long error, servotick; double position;

if (!esp_init_system())
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}
/* enable motor power */
esp_enable_motor(2);

/* move axis 2 to absolute position -3.0 */
esp_move_absolute(2,-3.0);

while (!esp_move_done(2));
/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf("Error %d Reported!", error);
}
```

See Also: esp_get_all_position(), esp_set_speed(), esp_set_accel(), esp_set_decel(), esp_move_relative(), esp_set_resolution(), esp_move_done( )

esp_move_relative *Move Relative Displacement*

Synopsis: #include “esp6000.h”
int esp_move_relative(long axis, double displacement)

Arguments: long axis
axis number from 1-6
double displacement
target displacement in user units

Library Location: \esp6000.dll

Description: **esp_move_relative()** will displace the selected axis relative to the present position.

For servo motor axes, “relative” displacements are with respect to present target absolute position. This helps avoid cumulative errors due to over- and/or under-shooting positioners.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* enable motor power */
esp_enable_motor(2);

/* move axis-2 relative -3 units */
esp_move_relative(2,-3);

/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf(“Error %d Reported!”, error);
}
```

See Also: esp_get_position_count(), esp_move_absolute(), esp_set_speed(), esp_set_accel(), esp_set_decel(), esp_set_resolution()

esp_find_home *Invoke Axis Home Search*

Synopsis: #include “esp6000.h”
int esp_find_home(long axis, mode)

Arguments: long axis
axis number from 1-6
long mode
define homing mode as HOMEONLY(0), HOMEINDEX (1)

Library Location: \esp6000.dll

Description: **esp_find_home()** invokes specified homing algorithm on the designated axis.

In all homing modes the controller attempts to reduce, or eliminate, mechanical backlash by always approaching the final position from the same direction.

HOMEONLY - During this mode the axis finds the home signal transition.

HOMEINDEX - During this mode the axis finds the index signal immediately following the home signal.

NOTE

Only one axis can be in homing mode at any one time.

At the end of a successful homing sequence the axis position counter is automatically set to position count zero (0). All absolute motion commands will then be referenced to this position.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* Enable Axis 2 Motor Power */
esp_enable_motor(2);
```

```
/* Set Axis Home Speed */
esp_set_home_speed(2,20.0);

/* Begin Home Search On Axis 2 */
esp_find_home(2,1);

/* Wait For Home Search Completion */
while (!esp_home_done(2));

/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf("Error %d Reported!\r\n",
error);
}
```

See Also: esp_set_home_speed(), esp_home_done()

esp_stop — *Stop Specified Axes*

Synopsis: #include “esp6000.h”
int esp_stop(long axis)

Arguments: long axis
axis number from 1-6, axis = 0 stops all axes

Library Location: \esp6000.dll

Description: **esp_stop()** causes all axes in motion to immediately decelerate using deceleration rate previously set by **esp_set_decel()** function call.

NOTE
All moving axes will be stopped when axis parameter is equal to zero (0).

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system()) exit(-1);

/* set axis 1 trajectory parameters */
esp_set_speed(1, 30.0);
esp_set_accel(1, 200.0);
esp_set_decel(1, 150.0);

esp_enable_motor(1);

/* set axis 1 to jog trajectory mode */
esp_set_traj_mode(1,JOG);

/* set axis 1 speed and direction */
esp_set_jog_speed(1, -20.0);
  • • •
/* stop motion */
esp_stop(1);
}
```

See Also: esp_stop_all()

esp_stop_all *Stop All Axes*

Synopsis: #include “esp6000.h”
int esp_stop_all(void)

Arguments: none

Library Location: \esp6000.dll

Description: **esp_ stop()** causes all axes in motion to immediately decelerate using deceleration rate previously set by **esp_set_decel()** function call.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system()) exit(-1);

esp_enable_motor(1);

/* set axis 1-3 to jog trajectory mode */
esp_set_traj_mode(1,JOG);
esp_set_traj_mode(2,JOG);
esp_set_traj_mode(3,JOG);

/* set axis 1 speed and direction */
esp_set_jog_speed(1, -20.0);
esp_set_jog_speed(1, +30.0);
esp_set_jog_speed(1, -25.0);
  • • •
/* stop all axes motion */
esp_stop_all();
}
```

See Also: esp_stop()

esp_set_position — *Set Position Count To Specified Value*
esp_get_position — *Report Position Count*

Synopsis: #include "esp6000.h"
int esp_set_position (long axis, double count)
int esp_get_position (long axis, double *count)

Arguments: long axis
Axis number from 1-8. Channels 7 & 8 are auxiliary counters.
double count
position count or steps in user units

Library Location: \esp6000.dll

Description: **esp_get_position()** reports the current position count or steps for the specified axis.
If bit-8 of **esp_set_feedback_config()** command is set then **esp_get_position()** reports encoder feedback in user units. If bit-8 is 0 and the axis motor type is stepper then **esp_get_position()** reports step count in user units.

esp_set_position() sets the current position count to the value specified for the selected axis.

Returns: ESPOK, ESPERROR

Hint: Position count is normally set to zero (0) after a home search.

Usage Example:

```
#include "esp6000.h"

main()
{
long error, servotick; double position;

if (!esp_init_system())
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* report axis 1 position count */
esp_get_position(1,&position);
  • • •
/* zero axis 1 position count */
esp_set_position(1, 0.0);

}
```

See Also: esp_find_home()

Trajectory Definition

esp_set_speed — *Set Axis Speed*
esp_get_speed — *Report Axis Speed Setting*

Synopsis: #include “esp6000.h”
int esp_set_speed(long axis, float speed)
int esp_get_speed(long axis, float *speed)

Arguments: long axis
axis number from 1-6
float speed
target speed <= maximum speed (in user units/second)

Library Location: \esp6000.dll

Description: **esp_set_speed()** sets the target slew (top) speed for the specified axis. Note that if acceleration is too shallow the axis may not reach the target speed.

esp_get_speed() reports target slew speed setting for the specified axis.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}


/* set axis 1 trajectory parameters */
esp_set_speed(1, 30.0);
esp_set_accel(1, 200.0);
esp_set_decel(1, 150.0);


/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf(“Error %d Reported!\r\n”,
error);
}
```

See Also: esp_set_accel( ), esp_set_decel( ), esp_move_absolute( ), esp_set_resolution( )

esp_set_max_speed *Set Axis Maximum Speed*
esp_get_max_speed *Get Axis Maximum Speed Setting*

Synopsis: #include “esp6000.h”
int esp_set_max_speed(long axis, float speed)
int esp_get_max_speed(long axis, float *speed)

Arguments: long axis
axis number from 1-6
long speed
maximum speed in user units/second

Library Location: \esp6000.dll

Description: **esp_set_max_speed()** sets the maximum permissible speed for the specified axis.

The controller will not accept speed parameters set by other functions (e.g., **esp_set_speed()**) which exceed maximum permissible speed.

esp_get_max_speed() reports the maximum permissible speed for the specified axis.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”
main()
{
long error, servotick;

if (!esp_init_system())
{
  printf(“ESP6000 Not Initialized! \r\n”);
  exit(-1);
}
/* set axis 1 trajectory parameters */
esp_set_max_speed(1, 100.0);
esp_set_max_accel(1, 500.0);
esp_set_max_jerk(1,100.0);

/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf(“Error %d Reported!\r\n”,
error);
}
```

See Also: esp_set_speed( ), esp_set_home_speed( ), esp_set_jog_speed( )

esp_set_accel — *Set Axis Acceleration Rate*
esp_get_accel — *Report Axis Acceleration Rate Setting*

Synopsis: #include “esp6000.h”
int esp_set_accel(long axis, float accel)
int esp_get_accel(long axis, float *accel)

Arguments: long axis
axis number from 1-6
float accel
target acceleration <= maximum accel (in user units/seconds^2)

Library Location: \esp6000.dll

Description: **esp_set_accel()** sets the target acceleration for the specified axis.

esp_get_accel() reports the target acceleration setting for the specified axis.

Returns: ESPOK, ESPERROR

Hint: Acceleration typically equal to deceleration.

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* set axis 1 trajectory parameters */
esp_set_speed(1, 30.0);
esp_set_accel(1, 200.0);
esp_set_decel(1, 150.0);

/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf(“Error %d Reported!\r\n”,
error);
}
```

See Also: esp_set_speed(), esp_set_decel(), esp_move_absolute(), esp_set_resolution()

esp_set_decel — *Set Axis Deceleration Rate*
esp_get_decel — *Get Axis Deceleration Rate*

Synopsis: #include “esp6000.h”
int esp_set_decel(long axis, float decel)
int esp_get_decel(long axis, float *decel)

Arguments: long axis
axis number from 1-6
float accel
target deceleration <= maximum accel (in user units/seconds^2)

Library Location: \esp6000.dll

Description: **esp_set_decel()** sets the target deceleration rate for the specified axis.

esp_get_decel() reports the target deceleration rate for the specified axis.

Returns: ESPOK, ESPERROR

Hint: Deceleration typically is set equal to acceleration.

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* set axis 1 trajectory parameters */
esp_set_speed(1, 30.0);
esp_set_accel(1, 200.0);
esp_set_decel(1, 150.0);

/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf(“Error %d Reported!\r\n”,
error);
}
```

See Also: esp_set_speed( ), esp_set_accel( ), esp_move_absolute( ), esp_set_resolution( )

esp_set_max_accel *Set Axis Maximum Acceleration Rate*
esp_get_max_accel *Report Axis Maximum Acceleration Rate Setting*

Synopsis: #include “esp6000.h”
int esp_set_max_accel(long axis, float accel)
int esp_get_max_accel(long axis, float *accel)

Arguments: long axis
axis number from 1-6
long accel
maximum acceleration in user units/seconds^2

Library Location: \esp6000.dll

Description: **esp_set_max_accel()** sets the maximum permissible acceleration for the specified axis.

The controller will not accept acceleration or deceleration parameters set by other functions (e.g., **esp_set_accel()**) which exceed maximum permissible acceleration.

esp_get_max_accel() reports the maximum permissible acceleration setting for the specified axis.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”
main()
{
long error, servotick;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}
/* set axis 1 trajectory parameters */
esp_set_max_speed(1, 100.0);
esp_set_max_accel(1, 500.0);
esp_set_max_jerk(1,100.0);

/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf(“Error %d Reported!\r\n”,
error);
}
```

See Also: esp_set_accel( ), esp_set_decel( )

esp_set_jerk — *Set Axis Jerk Rate*
esp_get_jerk — *Report Axis Jerk Rate Setting*

Synopsis: #include "esp6000.h"
int esp_set_jerk(long axis, float jerk)
int esp_get_jerk(long axis, float *jerk)

Arguments: long axis
axis number from 1-6
float jerk
set desired jerk rate in user units/seconds^3

Library Location: \esp6000.dll

Description: **esp_set_jerk()** sets the target jerk rate for the specified axis. This parameter should only be set while the axis is stopped. Jerk is the derivative of acceleration.

esp_get_jerk() reports the target jerk rate for the specified axis.

NOTE

The jerk parameter is only effective when the specified axis' trajectory mode is set to SCURVE.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include "esp6000.h"

main()
{
long error, servotick;

if (!esp_init_system())
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* set S-Curve trajectory mode */
esp_set_traj_mode(1, SCURVE);

/* set axis 1 trajectory parameters */
esp_set_speed(1, 30.0);
esp_set_accel(1, 200.0);
esp_set_decel(1, 150.0);
esp_set_jerk(1, 10.0);
```

```
/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf("Error %d Reported!\r\n",
error);
}
```

See Also: esp_get_jerk(), esp_set_speed(), esp_set_accel(), esp_set_decel(), esp_traj_mode(), esp_set_resolution()

esp_set_home_speed *Set Axis Home Speed*
esp_get_home_speed *Report Axis Home Speed Setting*

Synopsis: #include "esp6000.h"
int esp_set_home_speed(long axis, float speed)
int esp_get_home_speed(long axis, float *speed)

Arguments: long axis
axis number from 1-6
long speed
target home speed <= maximum speed (in user units/second)

Library Location: \esp6000.dll

Description: **esp_set_home_speed()** sets the target slew speed, or velocity, for the specified axis. The Homing algorithm uses the previously set acceleration, deceleration, and jerk settings.

esp_get_home_speed() reports the present target slew speed setting for the specified axis.

NOTE
If acceleration is too shallow the axis may not reach the target speed.

Returns: ESPOK, ESPERROR

Hint: During first time system testing set home speed to 1/10 of maximum speed.

Usage Example:

```
#include "esp6000.h"

main()
{
long error, servotick;

if (!esp_init_system())
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* Enable Axis 2 Motor Power */
esp_enable_motor(2);

/* Set Axis Home Speed */
esp_set_home_speed(2,20.0);

/* Begin Home Search On Axis 2 */
esp_find_home(2,1);
```

```
/* Wait For Home Search Completion */
while (!esp_home_done(2));

/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf("Error %d Reported!\r\n",
error);
}
```

See Also: esp_find_home(), esp_home_done()

esp_set_startstop_speed *Set Axis Start Speed (for Stepper Motors only)*
esp_get_startstop_speed *Report Axis Start Speed (for Stepper Motors only)*

Synopsis: #include "esp6000.h"
int esp_set_startstop_speed(long axis, float speed)
int esp_get_startstop_speed(long axis, float *speed)

Arguments: long axis
axis number from 1-6
float speed
target start/stop speed in user units/seconds^2

Library Location: \esp6000.dll

Description: **esp_set_startstop_speed()** sets the desired instantaneous start and stop speed for the specified stepper motor axis. This command is used to "skip over" the resonance frequency (typically around 1 motor rps) of stepper motors.

The axis has to be in TRAPSTEP trajectory mode (see `esp_set_traj_mode` function) for **esp_set_startstop_speed()** to take affect.

esp_get_startstop_speed() reports the present start speed setting for the specified axis.

Returns: ESPOK, ESPERROR

Hint: Most applications never need to change ESP-compatible default values.

Usage Example:

```
#include "esp6000.h"

main()
{
long error, servotick;

if (!esp_init_system())
{
  printf("ESP6000 Not Initialized! \r\n");
  exit(-1);
}

/* set axis 1 speed parameters */
esp_set_speed(1, 30.0);
esp_set_startstop_speed(1, 0.02);


/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf("Error %d Reported!\r\n",
error);
}
```

See Also: esp_set_speed( )

esp_set_jog_speed *Set Axis Jog Mode Speed*
esp_get_jog_speed *Report Axis Jog Mode Speed Setting*

Synopsis: #include "esp6000.h"
int esp_set_jog_speed(long axis, float speed)
int esp_get_jog_speed(long axis, float *speed)

Arguments: long axis
axis number from 1-6
float speed
target jog speed and direction <= maximum speed (in user units/second)

Library Location: \esp6000.dll

Description: **esp_set_jog_speed()** sets the target slew speed (velocity) for the specified axis. The sign of the jog speed determines the direction of the axis motion. For example, if the jog speed is defined as -2.0, then after the trajectory mode is set to jog (e.g., **esp_set_traj_mode**(1, JOG)) the specified axis will move in the negative direction at set speed.

esp_get_jog_speed() reports target slew speed setting for the specified axis.

In order for the axis to begin jog motion it first has to be placed in jog trajectory mode with the **esp_set_traj_mode()** command.

While in jog trajectory mode the axis will move indefinitely or until an **esp_stop(), esp_stop_all()** command is received.

Note that if acceleration is too shallow the axis may not reach the target speed.

Returns: ESPOK, ESPERROR

Hint: Use **esp_stop()** function to stop jog motion.

Usage Example:

```
#include "esp6000.h"

main()
{
long error, servotick;

if (!esp_init_system()) exit(-1);

esp_enable_motor(1);

/* set axis 1 to jog trajectory mode */
esp_set_traj_mode(1,JOG);

/* set axis 1 speed and direction */
esp_set_jog_speed(1, -20.0);
```

```
  • • •
/* stop motion */
esp_stop(1);
}
```

See Also: esp_set_traj_mode(), esp_stop(), esp_enable_motor()

Flow Control and Sequencing

esp_delay *Set Delay Time*

Synopsis: # include "esp6000.h"
int esp_delay(float time)

Arguments: float time
time in seconds (where maximum = 1000 seconds)

Library Location: \esp6000.dll

Description: **esp_delay( )** API call is used to delay continued command processing for the specified time. This API call uses the PC system timer that has a 55 milli-second resolution.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
# include "esp6000.h"

main ( )
(
long error, servotick;
double position;

if (! esp_init_system( ) )
{
printf ("ESP6000 Not Initialized! \r\n");
exit ( -1);
)

/* enable motor power */
esp_enable_motor (2);

/* move axis 2 to absolute position -3.0 */
esp_move_absolute (2, -3.0);

while ( ! esp_move_done (2) );

/* Wait 2.5 seconds */
esp_delay (2.5);

/* move axis 2 to absolute position 0.0 */
esp_move_absolute (2, 0.0);

while ( ! esp_move_done (2) );
```

```
/* check error status */
esp_get_error_num (&error, &ServoTick);
if (error) printf (*Error %d Reported !",
error);
}
```

See Also: esp_delay_after_stop( )

esp_delay_after_stop *Set Delay Time After Motion Stops*

Synopsis:
include "esp6000.h"
int esp_delay_after_stop(long axis, float time)

Arguments:
float axis
axis number from 1-6, axis = 0 tests all axes
float time
time in seconds (where maximum = 1000 seconds)

Library Location:
\esp6000.dll

Description:
esp_delay_after_stop( ) API call is used to delay continued command processing for the specified time after the specified axis has stopped. If the axis parameter is set to zero then the API will start the delay process after all axes have stopped.

This API call uses the PC system timer that has a 55 milli-second resolution.

Returns:
ESPOK, ESPERROR

Hint:

Usage Example:

```
# include "esp6000.h"

main ( )
{
long error, servotick;
double position;

if ( ! esp_init_system( ) )
{
printf ("ESP6000 Not Initialized! \r\n");
exit ( -1);
}

/* enable motor power */
esp_enable_motor (2);

/* move axis 2 to absolute position –3.0 */
esp_move_absolute (2, -3.0);

/* Wait 2.5 seconds */
esp_delay_after_stop (2, 2.5);

/* check error status */
esp_get_error_num (&error, &ServoTick);
```

```
if (error) printf ("Error %d Reported !",
error(;
}
```

See Also: esp_delay( )

Motion Device Parameters

esp_set_motor_type *Set Axis Motor Type*
esp_get_motor_type *Report Axis Motor Type Setting*

Synopsis:
#include "esp6000.h"
int esp_set_motor_type(long axis, long mtype)
int esp_get_motor_type(long axis, long *mtype)

Argument:
long axis
axis number from 1-6
long mtype
possible motor types are (0)UNDEFINED, (1)DCSERVO, (2)STEPPER

Library Location:
\esp6000.dll

Description:
esp_set_motor_type ( ) defines the motor type or technology used to control the specified axis.

Esp_get_motor_type ( ) reports the motor type or technology used to control the specified axis.

This is necessary because the ESP6000 needs to apply different control algorithms for different motor types. For example, DC servos are controlled via digital-to-analog converter (DAC) whereas stepper motors are positioned via digital rate multiplier.

NOTE

It will not be possible to control an axis if its motor type is undefined or equal to zero (0). Motor type is automatically set for ESP-compatible stages.

Returns:
ESPOK, ESPERROR

Hint:
Verify motor/stage is a defined motor type.

Usage Example:

```
#include "esp6000.h"

main()
{
```

```
if (!esp_init_system( )) exit (-1);
/* define axis 1 as DC servo */
esp_set_motor_type (1, DCSERVO);

/* Save new settings to non-volatile memory */
esp_save_parameters ();
}
```

See Also: esp_save_parameters( )

esp_set_master_slave *Assign Master/Slave Axes*
esp_get_master_slave *Report Master/Slave Axes Assignment*

Synopsis: #include “esp6000.h”
int esp_set_master_slave(long master, long slave)
int esp_get_master_slave(long *master, long slave)

Arguments: long master
master axis number from 1-8 (Note: axes 7 and 8 refer to auxiliary counters)
long slave
slave axis number from 1-6

Library Location: \esp6000.dll

Description: **esp_set_master_slave()** assigns master/slave relationship between axes.

esp_get_master_slave() reports the present master assignment to the specified (possible) slave axis.

NOTE

The slave's trajectory mode must be set to SLAVEP (slave to master encoder position) or SLAVET (slave to master trajectory) in order for master/slave mode to take effect.

The controller defaults to normal (non-master/slave) mode after system reset.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
if (!esp_init_system()) exit(-1);

/* assignment axis-2 (slave) to axis-1(master)*/
esp_set_master_slave(1,2);

/* assign master/slave ratio */
esp_set_master_slave_ratio(2,-0.5);

/* set slave to track master position (encoder)*/
esp_set_traj_mode(2, SLAVEP);

/* set master initial position */
esp_set_master_initial_position(1, 0.0);
```

```
/* set slave initial position */
esp_set_slave_initial_position(2, 0.0);
• • •
}
```

See Also:

esp_set_master_slave_ratio *Set Master/Slave Ratio*
esp_get_master_slave_ratio *Report Master/Slave Ratio*

Synopsis: #include “esp6000.h”
int esp_set_master_slave_ratio(long slave, float ratio)
int esp_get_master_slave_ratio(long slave, float ratio)

Arguments: long slave
slave axis number from 1-6
float ratio
master/slave ratio

Library Location: \esp6000.dll

Description: **esp_set_master_slave_ratio()** sets master-to-slave gear ratio. The sign of the ratio determines direction of gearing.

esp_get_master_slave_ratio() reports the present master-to-slave gear ratio.

NOTE

The controller defaults to normal (non-master/slave) mode after system reset.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
if (!esp_init_system()) exit(-1);

/* assignment axis-2 (slave) to axis-1(master) */
esp_set_master_slave(1,2);

/* assign master/slave ratio */
esp_set_master_slave_ratio(2,-0.5);

/* set slave to track master position (encoder)*/
esp_set_traj_mode(2, SLAVEP);

/* set master initial position */
esp_set_master_initial_position(1, 0.0);
```

```
/* set slave initial position */
esp_set_slave_initial_position(2, 0.0);
• • •
}
```

See Also:

esp_set_resolution *Set Axis Resolution*
esp_get_resolution *Report Axis Resolution Setting*

Synopsis: #include “esp6000.h”
int esp_set_resolution(long axis, float resolution, long units)
int esp_get_resolution(long axis, float *resolution, long *units)

Arguments: long axis
axis number from 1-6
float resolution
define mechanical resolution
long units
define user units as ENCODER_COUNT (0), MOTOR_STEP (1), MILLIMETER (2), MICROMETER (3), INCHES (4), MILLI_INCHES (5),
MICRO_INCHES (6), DEGREE (7), GRADIAN (8), RADIAN (9), MILLIRADIAN (10), MICRORADIAN (11)

NOTE

ESP6000 motion UNITS are treated as labels only for user convenience. No conversion is performed when units of measurement are changed from one unit to another. Users will have to re-enter all affected motion parameters (e.g., speed) when units are changed.

Resolution and units are automatically set the first time an ESP-compatible stage is detected on that axis.

Library Location: \esp6000.dll

Description: **esp_set_resolution()** defines the mechanical resolution for the specified axis.

Returns: ESPOK, ESPERROR

Hint: No need to change resolution with ESP-compatible stages present

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system()) exit(-1);

/* define axis 1 resolution equal to 0.001 mm  */
esp_set_resolution(1, 0.001, MILLIMETER);
```

```
/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf("Error %d Reported!\r\n",
error);

}
```

See Also: esp_set_speed(), esp_set_decel(), esp_set_accel(), esp_set_jerk()

esp_set_soft_limits *Define Software Travel Limits*
esp_get_soft_limits *Report Software Travel Limits Setting*

Synopsis: #include “esp6000.h”
int esp_set_soft_limit(long axis, double negative, double positive)
int esp_get_soft_limit(long axis, double *negative, double *positive)

Arguments: double leftpos
left software travel limit absolute position in user units
double rightpos
right software travel limit absolute position in user units

Library Location: \esp6000.dll

Description: **esp_set_soft_limit()** defines the right and left software travel limit for the specified axis.

Software travel limits are referenced to absolute position zero(0), or home position after homing has been performed.

Normally, after a system reset, the stage is first homed so that software travel limits are referenced to a known, repeatable location.

Software limits help prevent inadvertent travel into stage hardware limits.

esp_get_soft_limit() reports the right and left software travel limit setting for the specified axis.

NOTE

Software travel limits are ignored during homing mode.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”
main()
{
long error, servotick;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}
/* Define Axis 1 Software Travel Limits */
esp_set_soft_limit(1, -100.0, +100.0);

/* Abort Motion & Flag Error On Software Limit */
esp_set_softlimit_config(1, 0x0000000d);
```

```
/* Save Parameters To ESP6000 Flash EPROM */
esp_save_parameters();

/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf("Error %d Reported!\r\n",
error);
}
```

See Also: esp_set_softlimit_config(), esp_find_home()

esp_set_following_error *Set Motor Following Error Threshold*
esp_get_following_error *Report Motor Following Error Threshold Setting*

Synopsis: #include “esp6000.h”
int esp_set_following_error(long axis, double ferr)
int esp_get_following_error(long axis, double *ferr)

Arguments: long axis
axis number from 1-6
double ferr
maximum motor following error threshold in user units

Library Location: \esp6000.dll

Description: **esp_set_following_error()** sets the maximum motor following error threshold for the specified axis. A maximum moving error of ‘0’ disables this feature.

esp_get_following_error() reports the maximum motor following error threshold setting for the specified axis.

The **esp_set_followerr_config()** command determines what happens when the following error threshold is exceeded.

Returns: ESPOK, ESPERROR

Hint: If following error threshold is set too large then its purpose is defeated.

Usage Example:

```
#include “esp6000.h”
int status;

if (esp_init_system() )
{
/* Set Axis 1 Following Error Threshold */
status = esp_set_following_error(1,0.2);
}
```

See Also: esp_set_followerr_config()

esp_set_microstep_factor *Set Microstep Factor*
esp_get_microstep_factor *Report Microstep Factor Setting*

Synopsis: #include “esp6000.h”
int esp_set_microstep_factor(long axis, long factor)
int esp_get_microstep_factor(long axis, long *factor)

Arguments: long axis
axis number from 1-6
long factor
microstepping factor 1 - 255

Library Location: \esp6000.dll

Description: **esp_set_microstep_factor()** provides the controller with the step motor microstepping factor used on the motor driver. This API function call enables the ESP6000 to properly calculate the number of step pulses to output in order to achieve desired encoder-based position.

esp_get_microstep_factor() reports present microstepping factor setting in ESP6000 memory.

Returns: ESPOK, ESPERROR

Hint: Microstep factor is automatically set with ESP-compatible stepper stages.

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system() )
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* Define Axis-1 Microstep Resolution */
esp_set_microstep_factor(1, 10);

/* Full-step resolution */
esp_set_fullstep_resolution(1, 0.001);

/* Save new settings to non-volatile memory */
esp_save_parameters();

/* check error status */
esp_get_error_num(&error,&ServoTick);
```

```
if (error) printf("Error %d Reported!\r\n",
error);
}
```

See Also: esp_set_fullstep_resolution()

esp_set_fullstep_resolution *Set Full-Step Resolution*
esp_get_fullstep_resolution *Report Full-Step Resolution Setting*

Synopsis: #include “esp6000.h”
int esp_set_fullstep_resolution(float, fullstep)
int esp_get_fullstep_resolution(float, *fullstep)

Arguments: long axis
axis number from 1-6
float fullstep
full-step resolution in user units

Library Location: \esp6000.dll

Description: **esp_set_fullstep_resolution()** provides the controller with the step motor fullstep resolution in user units. This API function call enables the ESP6000 to properly calculate the number of step pulses to output in order to achieve desired encoder-based position.

esp_get_fullstep_resolution() reports present microstepping factor setting in ESP6000 memory.

Returns: ESPOK, ESPERROR

Hint: Fullstep resolution is automatically set with ESP-compatible stepper stages.

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system() )
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* Define Axis-1 Microstep Resolution */
esp_set_microstep_factor(1, 10);

/* Full-step resolution */
esp_set_fullstep_resolution(1, 0.001);

/* Save new settings to non-volatile memory */
esp_save_parameters();
```

```
/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf("Error %d Reported!\r\n",
error);

}
```

See Also: esp_set_fullstep_resolution()

esp_set_motor_current *Set UniDrive Axis Motor Current*
esp_get_motor_current *Get UniDrive Axis Motor Current*

Synopsis: #include “esp6000.h”
int esp_set_motor_current(long axis, float current)
int esp_get_motor_current(long axis, float current)

Arguments: long axis
axis number from 1-6
float current
motor current from 0 through 8.0 amps

Library Location: \esp6000.dll

Description: **esp_set_motor_current()** API function call is used to set the Unidrive6000 motor amplifier current for the specified axis. To take immediate effect this command should be followed with the **esp_update_unidrive()** command.

CAUTION

Motor damage can occur if current is set too high. Please refer to motor specifications before setting value.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”
main()
{
long error, servotick;

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* Change Axis-1 Motor Current */
esp_motor_current(1, 1.9);

/* Update Unidrive6000 Axis-1 */
esp_update_unidrive(1);

/* Save new settings to non-volatile memory */
esp_save_parameters();
```

```
/* check error status */
esp_get_error_num(&error,&ServoTick);
if (error) printf("Error %d Reported!\r\n",
error);

}
```

See Also: esp_update_unidrive(), esp_save_parameters()

esp_set_tach_constant *Set UniDrive Axis Motor Tachometer Constant*
esp_get_tach_constant *Get UniDrive Axis Motor Tachometer Constant*

Synopsis:

include "esp6000.h"

int esp_set_tach_constant(long axis, float tach)

int esp_get_tach_constant(long axis, float tach)

Arguments:

long axis
axis number from 1-6
float tach
motor tachometer constant (in volts/Krpm)

Library Location:

\esp6000.dll

Description:

esp_set_tach_constant( ) API call is used to set the UniDrive6000 motor amplifier tachometer constant for the specified servo axis and should be used in conjunction with **esp_set_gear_constant( ).** To take immediately effect this command should be followed with the **esp_ipdate_unidrive()** command.

Tachometer feedback provides improved servo stability.

CAUTION

Poor servo performance can occur if tachometer constant is inappropriately set. Please refer to tachometer specifications before setting value.

Returns:

ESPOK, ESPERROR

Hint:

Usage Example:

```
# include "esp6000.h"
main ( )

{
long error, servotick;

iff ( ! esp_init_system( ) )
{

printf ("ESP6000 Not Initialized! \r\n");
exit ( -1);
}

/* Set Axis-1 Motor Current */
```

```
esp_set_motor_current (1, 1.9);
/* Set Axis-1 Stage Gear Constant */
esp_set_gear_constant (1, 0.3);

/* Set Axis-1 Tachometer Constant */
esp_set_tach_constant (1, 3.1);

/* Update Unidrive6000 Axis-1 */
esp_update_unidrive (1);

/* Save new settings to non-volatile memory */
esp_save_parameters ( );

/* check error status */
esp_get_error_num (&error, &ServoTick);
if (error) printf ("Error %d Reported! \r\n",
error);

}
```

See Also: esp_set_gear_constant(), esp_update_unidrive(), esp_save_parameters()

esp_set_gear_constant *Set UniDrive Axis Motor Gear Constant*
esp_get_gear_constant *Report UniDrive Axis Motor Gear Constant*

Synopsis:
include "esp6000.h"
int esp_set_gear_constant(long axis, float gear)
int esp_get_gear_constant(long axis, float gear)

Arguments:
long axis
axis number from 1-6
float tach
motor/stage gear constant (in revolution / unit of measure)

Library Location:
\esp6000.dll

Description:
esp_set_gear_constant( ) API call is used to set the UniDrive6000 motor amplifier gear constant for the specified servo axis and should be used in conjunction with **esp_set_tach_constant( ).** The *gear constant* is defined as the number of revolutions the motor has to make for the motion device to move one displacement unit. To take immediate effect this command should be followed with the **esp_update_unidrive( )** command.

CAUTION
Poor servo performance can occur if gear constant is inappropriately set. Please refer to stage specifications before setting value.

Returns:
ESPOK, ESPERROR

Hint:

Usage Example:
```
# include "esp6000.h"
main ( )

{  long error, servotick;

if  (! Esp_init_system( ) )
{

printf ("ESP6000 Not Initialized! \r\n");
exit ( -1);
}

/* Set Axis-1 Motor Current */
esp_set_motor_current (1, 1.9);
```

```
/* Set Axis-1 Stage Gear Constant */
esp_set_gear_constant (1, 0.3);

/* Set Axis-1 Tachometer Constant */
esp_set_tach_constant (1, 3.1);

/* Update Unidrive6000 Axis-1 */
esp_update_unidrive (1);

/* Save new settings to non-volatile memory */
esp_save_parameters( );

/* check error status */
esp_get_error_num (&error, &ServoTick);
if (error) printf ("Error %d Reported! \r\n",
error);

}
```

See Also: esp_set_tach_constant( ), esp_update_unidriver( )

esp_save_parameters *Save Parameters To Flash EPROM Memory*

Synopsis:

include "esp6000.h"
int esp_save_parameters( )

Arguments:

none

Library Location:

\esp6000.dll

Description:

esp_save_parameters( ) API function call causes the ESP6000 to save present parameters to non-volatile flash EPROM memory. Parameters saved to flash EPROM are automatically restored to working registers after system initialization or reset.

NOTE

Saved axis settings are automatically overwritten when a different ESP-compatible stage is detected in the same axis channel.

All saved settings are automatically erased when new ESP6000 firmware is downloaded.

Returns:

ESPOK, ESPERROR

Hint:

Usage Example:

```
# include "esp6000.h"

main ( )
{
long error, servotick;

if (! Esp_init_system ( ) )
{
printf ("ESP6000 Not Initialized! \r\n");
exit (-1)
}

/* set PID gain */
esp_set_kp (1,100.0) ;
```

```
esp_set_kd (1,200.0) ;
esp_set_ki (1,50.0);
esp_set_il (1,50.0);
/* transfer PID to working registers */
esp_update_filter ( );

/* save parameters to ESP6000 Flash EPROM */
esp_save_parameters ( );

/* check error status */
esp_get_error_num (& error, &ServoTick) ;
if (error) printF ("Error %d Reported! ", error);
}
```

See Also:

Digital Filters

esp_set_kp — *Set PID Proportional Gain (Kp)*
esp_get_kp — *Report PID Proportional Gain (KP) Setting*

Synopsis:
#include "esp6000.h"
int esp_set_kp(long axis, float kp)
int esp_get_kp(long axis, float *kp)

Arguments:
long axis
axis number from 1-6
float kp
proportional gain

Library Location: \esp6000.dll

Description:
esp_set_kp() sets the proportional gain (kp) of the PID servo filter.
esp_get_kp() reports the proportional gain (kp) setting.

NOTE
If necessary, use the ESP-tune utility to optimize servo PID and feedforward parameters.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include "esp6000.h"

main()
{
long error, servotick;

if (!esp_init_system() )
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* set PID gain */
esp_set_kp(1,100.0);
esp_set_kd(1,200.0);
esp_set_ki(1,50.0);
esp_set_il(1,50.0);

/* transfer PID to working registers */
esp_update_filter();
```

```
/* save parameters to ESP6000 Flash EPROM */
esp_save_parameters();

/* check error status */
esp_get_error_num(&error,&ServoTick) ;
if (error) printf("Error %d Reported!", error);

}
```

See Also: esp_set_kd(), esp_set_ki(), esp_set_il(), esp_update_filter()

esp_set_kd — *Set PID Derivative Gain (Kd)*
esp_get_kd — *Report PID Derivative Gain (Kd) Setting*

Synopsis: #include “esp6000.h”
int esp_set_kd (long axis, float kd)
int esp_get_kd (long axis, float *kd)

Arguments: long axis
axis number from 1-6
float kp
derivative gain

Library Location: \esp6000.dll

Description: **esp_set_kd()** sets the derivative gain (kd) of the PID servo filter.
esp_get_kd() reports the derivative gain (kd) setting.

NOTE
If necessary, use the ESP-tune utility to optimize servo PID and feedforward parameters.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system() )
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* set PID gain */
esp_set_kp(1,100.0);
esp_set_kd(1,200.0);
esp_set_ki(1,50.0);
esp_set_il(1,50.0);

/* transfer PID to working registers */
esp_update_filter();

/* save parameters to ESP6000 Flash EPROM */
esp_save_parameters();
```

```
/* check error status */
esp_get_error_num(&error,&ServoTick) ;
if (error) printf("Error %d Reported!", error);
}
```

See Also: esp_set_kp(), esp_set_ki(), esp_set_il(), esp_update_filter()

esp_set_ki — *Set PID Integral Gain (Ki)*
esp_get_ki — *Report PID Integral Gain (Ki) Setting*

Synopsis: #include “esp6000.h”
int esp_set_ki(long axis, float ki)
int esp_get_ki(long axis, float *ki)

Arguments: long axis
axis number from 1-6
float kp
integral gain

Library Location: \esp6000.dll

Description: **esp_set_ki()** sets the integral gain (ki) of the PID servo filter.

esp_get_ki() reports the integral gain (ki) setting.

NOTE
If necessary, use the ESP-tune utility to optimize servo PID and feedforward parameters.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system() )
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* set PID gain */
esp_set_kp(1,100.0);
esp_set_kd(1,200.0);
esp_set_ki(1,50.0);
esp_set_il(1,50.0);

/* transfer PID to working registers */
esp_update_filter();

/* save parameters to ESP6000 Flash EPROM */
esp_save_parameters();
```

```
/* check error status */
esp_get_error_num(&error,&ServoTick) ;
if (error) printf("Error %d Reported!", error);
}
```

See Also: esp_set_kd(), esp_set_moving_kp(), esp_set_il(), esp_update_filter()

esp_set_il — *Set PID Integral Limit (il)*
esp_get_il — *Report PID Integral Limit (il) Setting*

Synopsis: #include “esp6000.h”
int esp_set_il(long axis, float il)
int esp_get_il(long axis, float *il)

Arguments: long axis
axis number from 1-6
float il
integral limit

Library Location: \esp6000.dll

Description: **esp_set_il()** sets the integral limit (il) of the PID servo filter.

esp_get_il() reports the integral limit (il) setting.

NOTE
If necessary, use the ESP-tune utility to optimize servo PID and feedforward parameters.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;

if (!esp_init_system() )
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* set PID gain */
esp_set_kp(1,100.0);
esp_set_kd(1,200.0);
esp_set_ki(1,50.0);
esp_set_il(1,50.0);

/* transfer PID to working registers */
esp_update_filter();

/* save parameters to ESP6000 Flash EPROM */
esp_save_parameters();
```

```
/* check error status */
esp_get_error_num(&error,&ServoTick) ;
if (error) printf("Error %d Reported!", error);
}
```

See Also: esp_set_kd(), esp_set_kp(), esp_set_ki(), esp_update_filter()

esp_set_vel_feedforward — *Set Velocity Feedforward Gain*
esp_get_vel_feedforward — *Report Velocity Feedforward Gain Setting*

Synopsis: #include "esp6000.h"
int esp_set_vel_feedforward(long axis, float vff)
int esp_get_vel_feedforward(long axis, float *vff)

Arguments: long axis
axis number from 1-6
float vff
velocity feedforward gain

Library Location: \esp6000.dll

Description: **esp_set_vel_feedforward()** sets the velocity feedforward gain for the specified servo axis.

esp_get_vel_feedforward() reports the present velocity feedforward gain setting for the specified servo axis.

NOTE

If necessary, use the ESP-tune utility to optimize servo PID and feedforward parameters.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include "esp6000.h"

main()
{
long error, servotick;

if (!esp_init_system() )
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* set PID gain */
esp_set_kp(1,100.0);
esp_set_kd(1,200.0);
esp_set_ki(1,50.0);
esp_set_il(1,50.0);
```

```
/* set velocity feedforward */
esp_set_vel_feedforward(1,75);

/* set acceleration feedforward */
esp_set_acc_feedforward(1,100);

/* transfer PID to working registers */
esp_update_filter();

/* save parameters to ESP6000 Flash EPROM */
esp_save_parameters();

/* check error status */
esp_get_error_num(&error,&ServoTick) ;
if (error) printf("Error %d Reported!", error);
}
```

See Also: esp_set_acc_feedforward(), esp_update_filter()

esp_set_acc_feedforward *Set Acceleration / Deceleration Feedforward Gain*
esp_get_acc_feedforward *Report Accel & Decel Feedforward Gain Settings*

Synopsis: #include "esp6000.h"
int esp_set_acc_feedforward(long axis, float aff)
int esp_get_acc_feedforward(long axis, float *aff)

Arguments: long axis
axis number from 1-6
float off
acceleration & deceleration feedforward gain

Library Location: \esp6000.dll

Description: **esp_set_acc_feedforward()** sets the acceleration and deceleration feedforward gain for the specified servo axis.

esp_get_acc_feedforward() reports the acceleration and deceleration feedforward gain setting for the specified servo axis.

NOTE
If necessary, use the ESP-tune utility to optimize servo PID and feedforward parameters.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include "esp6000.h"

main()
{
long error, servotick;

if (!esp_init_system() )
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}

/* set PID gain */
esp_set_kp(1,100.0);
esp_set_kd(1,200.0);
esp_set_ki(1,50.0);
esp_set_il(1,50.0);

/* set velocity feedforward */
esp_set_vel_feedforward(1,75);
```

```
/* set acceleration feedforward */
esp_set_acc_feedforward(1,100);

/* transfer PID to working registers */
esp_update_filter();

/* save parameters to ESP6000 Flash EPROM */
esp_save_parameters();

/* check error status */
esp_get_error_num(&error,&ServoTick) ;
if (error) printf("Error %d Reported!", error);
}
```

See Also: esp_set_vel_feedforward, esp_update_filter()

esp_update_filter *Update Servo PID and Feedforward Coefficients*

Synopsis: #include “esp6000.h”
int esp_update_filter(void)

Arguments:

Library Location: \esp6000.dll

Description: `esp_update_filter()` transfers all changed PID and feedforward parameters (i.e., acceleration and velocity) to working servo registers.

NOTE
If necessary, use the ESP-tune utility to optimize servo PID and feedforward parameters.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”
main()
{ long error, servotick;
if (!esp_init_system() )
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* set PID gain */
esp_set_kp(1,100.0);
esp_set_kd(1,200.0);
esp_set_ki(1,50.0);
esp_set_il(1,50.0);

/* transfer PID to working registers */
esp_update_filter();

/* save parameters to ESP6000 Flash EPROM */
esp_save_parameters();

/* check error status */
esp_get_error_num(&error,&ServoTick) ;
if (error) printf(“Error %d Reported!”, error);
}
```

See Also: esp_set_kd( ), esp_set_kp( ), esp_set_ki( ), esp_set_il( ), esp_update_filter( )

I/O Functions

esp_set_adc_gain — *Set Analog-To-Digital Converter Input Gain*
esp_get_adc_gain — *Report Analog-To-Digital Converter*

Synopsis:

```
# include "esp6000.h"
int esp_set_adc_gain(long gain)
int esp_get_adc_gain(long *gain)
```

Arguments:

long gain
ADC gain (V1_25, V2_5, V5, V10
0-3 (corresponding to gain of 1, 2, 4, or 8 respectively)

Library Location:

\esp6000.dll

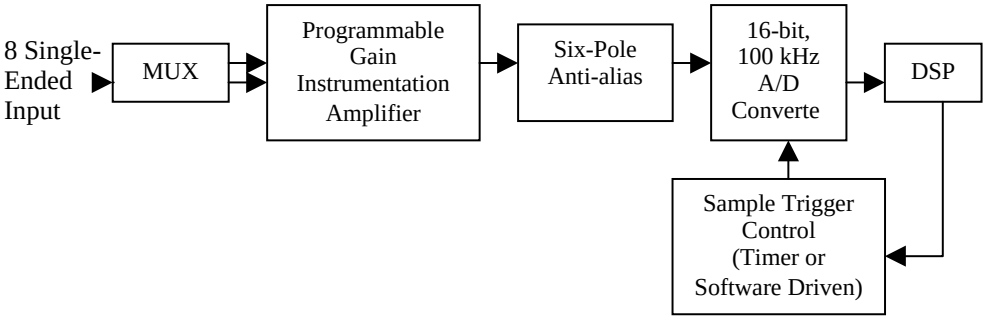
Description:

esp_set_adc_gain( ) will set the gain of all eight (8) ADC channels.

esp_get_adc_gain( ) reports the present gain setting of all eight (8) ADC channels.

The analog input range of each ADC channel is software-configurable for ranges of ±10V, ±5V, ±2.5V, and ±1.25V.

ADC channels are located on the analog I/O connector on the controller card.



Returns:

ESPOK, ESPERROR

Hint:

Usage Example:

```
# include "esp6000.h"
```

```
main( )
{

long timestamp; float volts;
if ( ! esp_init_system( ) ) exit (-1);

/* Set ADC Gain */
esp_set_adc_gain (V10);

/* Set ADC Range */
esp_set_adc_range (BIPOLAR) ;

/* Acquire ADC Channel 1 Analog Data */
esp_get_adc (1, &volts, &timestamp);
printf ("ADC = %f \n\r", volts);
}
```

See Also: esp_set_adc_gain( ), esp_get_adc( ), esp_get_all_adc( )

esp_set_adc_range *Set Analog-To-Digital Converter Input Range*
esp_get_adc_range *Report Analog-To-Digital Range Setting*

Synopsis:

```
# include "esp6000.h"
int esp_set_adc_range(long range)
int esp_get_adc_range(long *range)
```

Arguments:

long gain
ADC range 0 or 1 (BIPOLAR or UNIPOLAR respectively)

Library Location:

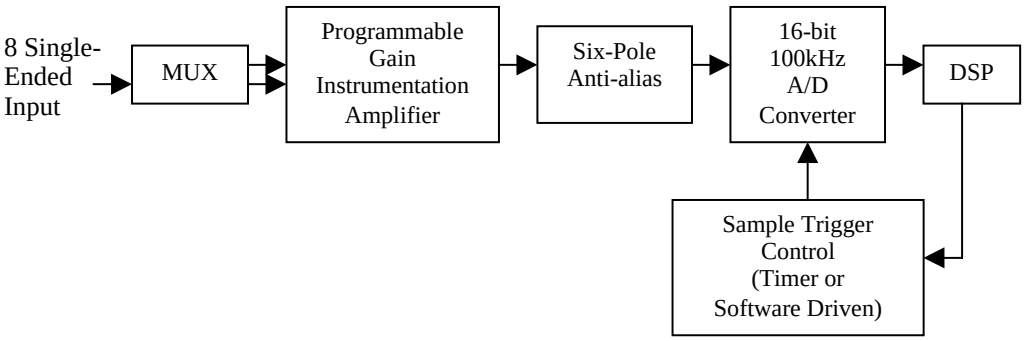
\esp6000.dll

Description:

esp_set_adc_range( ) will set the range of all eight (8) ADC channels whereas **esp_get_adc_range( )** will retrieve the current setting.

The analog input range of each ADC channel is software-configurable for ranges of ±10V, ±5V, ±2.5V, and ±1.25V for BIPOLAR. The input ranges are 0-10V, 0-5V, 0-2.5V, and 0-1.25V for UNIPOLAR.

ADC channels are located on the analog I/O connector on the controller card.



Returns:

ESPOK, ESPERROR

Hint:

Usage Example:

```
# include "esp6000.h"

main ( )
{
long timestamp;
float volts;

if ( ! esp_init_system ( ) exit (-1);
```

```
/* Set ADC Gain */
esp_set_adc_gain (V10) ;

/* Set ADC Range */
esp_set_adc_range (BIPOLAR);

/* Acquire ADC Channel 1 Analog Data */
esp_get_adc (1, &volts, &timestamp);
printf ("ADC = %f \n\r", volts);
}
```

See Also: esp_set_adc_gain( ), esp_get_adc( ), esp_get_all_adc( )

esp_get_adc *Read Analog-To-Digital Converter Channel*

Synopsis:
include "esp6000.h"
int esp_get_abc(long channel, float * volts, long *timestamp

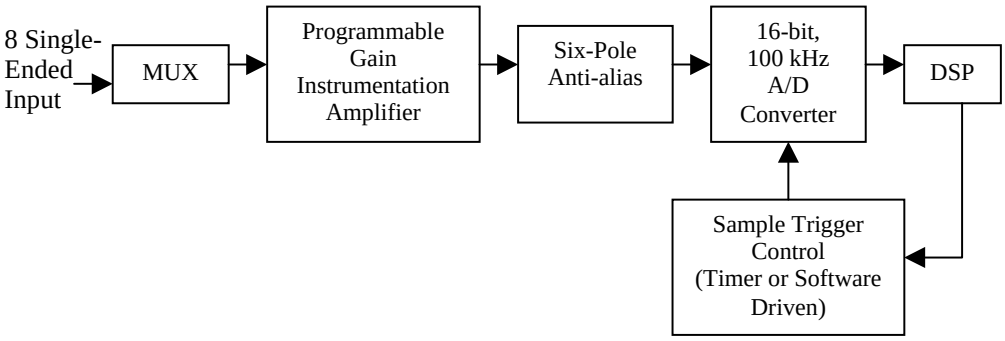
Arguments:
long channel
ADC input channel 1-8
long data
address of variable where ADC data is to be stored
long timestamp
current servo counter value at time of acquisition

Library Location:
\esp6000.dll

Description:
esp_get_adc( ) reads the analog-to-digital converter channel (1-8) specified.

The analog input range of each ADC channel is software-configurable for ranges of ±/10V, 0 through +10 through 0, ±/5V, ±/2.5V, and ±/1.25V.

ADC channels are located on the analog I/O connector on the controller card.



Returns:
ESPOK, ESPERROR

Hint:

Usage Example:

```
# include "esp6000.h"

main ( )
{
long timestamp;
float volts;
```

```
if ( ! esp_init_system( ) ) exit (-1);
/* Set ADC Gain */
esp_set_adc_gain (V10);

/* Set ADC Range */
esp_set_adc_range (BIPOLAR);

/* Acquire ADC Channel 1 Analog Data */
esp_get_adc (1, &volts, &timestamp);
printf ("ADC = %f \n\r", volts);
}
```

See Also: esp_set_adc_range( ), esp_set_adc_gain( ), esp_get_all_adc( )

esp_get_all_adc *Read All Analog-To-Digital Converter Channels*

Synopsis:

include "esp6000.h"
int esp_get_all_adc(float *DataArray, long *timestamp)

Arguments:

float DataArray
address of array where ADC data is to be stored
long timestamp
current servo counter value at time of acquisition

Library Location:

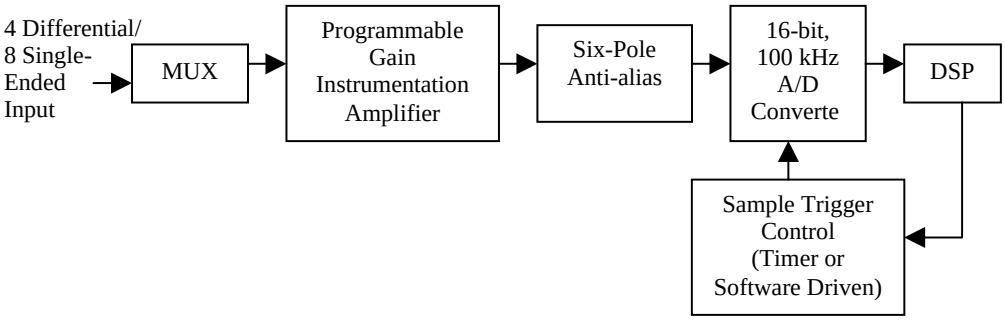
\esp6000.dll

Description:

esp_get_all_adc( ) reads all analog-to-digital converter channels 1-8.

The analog input range of each ADC channel is software-configurable for ranges of ±10V, ±5v, ±2.5V, and ±1.25V.

ADC channels are located on the analog I/O connector on the controller card.



Returns:

ESPOK, ESPERROR

Hint:

Usage Examples:

```
# include "esp6000.h"

main ( )
{
long timestamp;
float DataArray [8];

if ( ! esp_init_system ( ) ) exit (-1);

/* Set ADC Gain */
esp_set_adc_gain (V10);
```

```
/* Set ADC Range */
esp_set_adc_range (UNIPOLAR);
/* Acquire All ADC Channels Data */
esp_get_all_adc (DataArray, &timestamp);
}
```

See Also: esp_set_adc_range( ), esp_set_adc_gain( ), esp_get_adc( )

esp_set_daq_mode *Set Data Acquisition Mode and Parameters*

Synopsis:

include "esp6000.h"
int esp_set_daq_mode(long mode, long axis, long Adcs,
long feedback, long rate, long Num)

Arguments:

long mode
mode of acquisition where,
0 = start data collection when the daq is enabled.
1 = start data collection at the start of motion.
2 = start data when the specified axis reaches slew speed.
3 = Capture position on an external trigger.
4 – 29 Reserved

Fixed Streaming Modes:
30 = Start streaming a fixed amount of data when the daq is enabled.
31 = Start streaming a fixed amount of data at the start of motion.
32 = Start streaming a fixed amount of data when the specified axis reaches slew speed.
33 = Start streaming a fixed amount of position data on an external trigger.

Continuous Streaming Modes:
36 = Start a continuous stream of data when the daq is enabled.
37 = Start a continuous stream of data at the start of motion.
38 = Start a continuous stream of data when the specified axis reaches slew speed.
39 = Start a continuous stream of position data on an external trigger.

long axis
motion axis used to trigger acquisition

long Adcs
analog-to-digital channels involved in acquisition
where bit-0 = channel 1, bit-1 = channel 2, bit-2 = channel 3… bit-7 = channel 8

long feedback
position feedback (encoder) channels involved in acquisition
where bit-0 = channel 1, bit-1 = channel 2, bit-2 = channel 3… bit-7 = channel 8

long rate
acquisition rate which is a multiple of the servo cycle rate
where 0 = every servo cycle, 1 = every other, 2 = every 3rd, 3 = every 4th

long num
number of acquisition samples (where maximum = 512)

Library Location: \esp6000.dll

Description: **esp_set_daq_mode( )** API function call is used to set Data Acquisition (DAQ) mode. ESP6000 DAQ modes facilitate the capture of any combination of 16-bit, 8-channel analog-to-digital input and 8-quadrature encoded position data. Other commands like **esp_get_daq_data( )** retrieve stored information.

NOTE

During DAQ mode data is collected at servo cycle (409 msec.) intervals.

Returns: ESPOK, ESPERROR

Hint:

User Example:

```
# include "esp6000.h"

main ( )
{
long error, servicetick;
long Num, DaqStat, Mode, count;
float DataArray [512};

if ( ! esp_init_system ( ) )

{
printf ("ESP6000 Not Initialized! \r\n");
exit (-1);
}

/* Set ADC Gain and Range */
esp_set_adc_gain (V10);
esp_set_adc_range (BIPOLAR);

/* Set Acquisition Mode */
esp_set_daq_mode (1, 1, 1, 1, 2, 512);

esp_enable_daq( );

esp_enable_motor( );

esp_move_absolute (1, 50.0)

while ( ! esp_daq_done( ) )
/* Wait for DAQ End */
{
esp_get_daq_status (&count);
printf ("%d acquisitions collected.\r", count);
}
```

```
esp_disable_daq( );

/* Retrieve Data */
esp_get_daq_data (DataArray, &Num, &DaqStat);

/* check error status *
esp_get_error_num (&error, &ServoTick);
if (error) printf ("Error %d Reported ! \r\n",
error);
}
```

See Also: esp_get_daq_status( ), esp_enable_daq( ), esp_disable_daq( ), esp_get_daq_data( ), esp_daq_done( )

esp_enable_daq *Enable Data Acquisition Mode*

Synopsis: # include "esp6000.h"
int esp_enable_daq(void)

Arguments: none

Library Location: \esp6000.dll

Description: **esp_enable_daq( )** enables data acquisition mode.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
# include "esp6000.h"

main ( )
{
long error, servotick;;
long Num, DaqStat, Mode, count;
float DataArray [512];

if ( ! esp_init_system ( ) )
{
printf ("ESP6000 Not Initialized! \r\n");
exit (-1);
}

/* Set ADC Gain and Range */
esp_set_adc_gain (V10);
esp_set_adc_range (BIPOLAR);

/* Set Acquisition Mode */
esp_set_daq_mode (1, 1, 1, 1, 2, 512);

esp_enable_daq( );

esp_enable_motor (1);

esp_move_absolute (1, 50.0);

while (! Esp_daq_done( ) )
/* Wait for DAQ End */
{
esp_get_daq_status (&count(;
printf ("%d acquisitions collected.\r", count);
}

esp_disable_daq( );
```

```
/* Retrieve Data */
esp_get_daq_data (DataArray, &Num, &DaqStat);

/* check error status */
esp_get_error_num (&error, &ServoTick);
if (error) printf ("Error %d Reported! \r\n",
error);
```

See Also: esp_get_daq_status( ), esp_disable_daq( ), esp_get_daq_data( ), esp_daq_done( ), esp_set_daq_mode( )

esp_disable_daq *Disable Data Acquisition Mode*

Synopsis: #include “esp6000.h”
int esp_disable_daq(void)

Arguments: none

Library Location: \esp6000.dll

Description: **esp_disable_daq()** disables data acquisition mode.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
#include “esp6000.h”

main()
{
long error, servotick;
long Num, DaqStat, Mode, count;
float DataArray[512];

if (!esp_init_system())
{
printf(“ESP6000 Not Initialized! \r\n”);
exit(-1);
}

/* Set ADC Gain and Range*/
esp_set_adc_gain(V10);
esp_set_adc_range(BIPOLAR);

/* Set Acquisition Mode */
esp_set_daq_mode(1, 1, 1, 1, 2, 512);

esp_enable_daq();

esp_enable_motor(1);

esp_move_absolute(1,50.0);

while(!esp_daq_done()) /* Wait for DAQ End */
{
  esp_get_daq_status(&count);
  printf(“%d acquisitions collected.\r”,count);
}

esp_disable_daq();

/* Retrieve Data */

esp_get_daq_data(DataArray, &Num, &DaqStat);


/* check error status
*/esp_get_error_num(&error,&ServoTick) ;
if (error) printf(“Error %d Reported!\r\n”,
error);
```

```
}
```

See Also: esp_get_daq_status(), esp_disable_daq(), esp_get_daq_data(), esp_daq_done(), esp_set_daq_mode()

esp_get_daq_status *Retrieve Data Acquisition Status*

Synopsis: # include "esp6000.h"
int esp_get_daq_status(*count)

Arguments: long count
number of samples collected

Library Location: \esp6000.dll

Description: **esp_get_daq_status( )** returns the number of acquisitions collected. This function can be used to monitor the status of data acquisitions.

Returns: ESPOK, ESPERROR

Hint:

Usage Example:

```
# include "esp6000.h"

main ( )
{
long error, servotick;
long Num, DaqStat, Mode, count;
float DataArray [512];

if ( ! esp_init_system( ) )
{
printf ("ESP6000 Not Initialized! \r\n");
exit (-1);
}

/* Set ADC Gain and Range*/
esp_set_adc_gain (V10);
esp_set_adc_range (BIPOLAR);

/* Set Acquisition Mode */
esp_set_daq_mode (1, 1, 1, 1, 2, 512);

esp_enable_daq( );

esp_enable_motor (1);

esp_move_absolute (1, 50.0);

while ( ! esp_daq_done( ) )
/* Wait for DAQ End */
{

esp_get_daq_status (&count);
```

```
printf ("%d acquisitions collected. \r", count);
}

esp_disable_daq( );

/* Retrieve Data */
esp_get_daq_data (DataArray, &Num, &DaqStat);

/* check error status */
esp_get_error_num (&error, &ServoTick);
if (error) printf ("Error %d Reported! \r\n",
error);
}
```

See Also: esp_get_daq_done( ), esp_set_daq_mode( ), esp_enable_daq( ), esp_disable_daq( ), esp_get_daq_data( )

esp_daq_done — *Return Data Acquisition Completion Status*

Synopsis: #include “esp6000.h”
int esp_daq_done(void)

Arguments:

Library Location: \esp6000.dll

Description: **esp_daq_done()** API function call returns the present data acquisition completion status. During data acquisition modes esp_daq_done() is used to indicate the completion of data collection.

This function returns a value indicating:

DAQREADY(-1) meaning DAQ armed, but not yet triggered

DAQSTARTED(0) meaning DAQ acquiring

DAQDONE(1) meaning DAQ finished.

Returns: DAQREADY, DAQSTARTED, or DAQDONE

Hint:

Usage Example:

```
#include “esp6000.h”
main()
{
long error, servotick;
long Num, DaqStat, Mode, count;
float DataArray[512];
if (!esp_init_system())
{
  printf(“ESP6000 Not Initialized! \r\n”);
  exit(-1);
}
/* Set ADC Gain and Range*/
esp_set_adc_gain(V10);
esp_set_adc_range(BIPOLAR);
/* Set Acquisition Mode */
esp_set_daq_mode(1, 1, 1, 1, 2, 512);
esp_enable_daq();
esp_enable_motor(1);
esp_move_absolute(1,50.0);

while(!esp_daq_done())
/* Wait for DAQ End */
{
esp_get_daq_status(&count);
printf(“%d acquisitions collected.\r”,count);
}
```

```
esp_disable_daq();
/* Retrieve Data */
esp_get_daq_data(DataArray, &Num, &DaqStat);
/* check error status
*/esp_get_error_num(&error,&ServoTick);
if (error) printf("Error %d Reported!\r\n",
error);
}
```

See Also: esp_get_daq_status(), esp_set_daq_mode(), esp_enable_daq(), esp_disable_daq(), esp_get_daq_data()

esp_get_daq_data *Report Data Acquisition Results*

Synopsis: #include “esp6000.h”
int esp_daq_data(long *DataArray, long *Num, long *DaqStat)

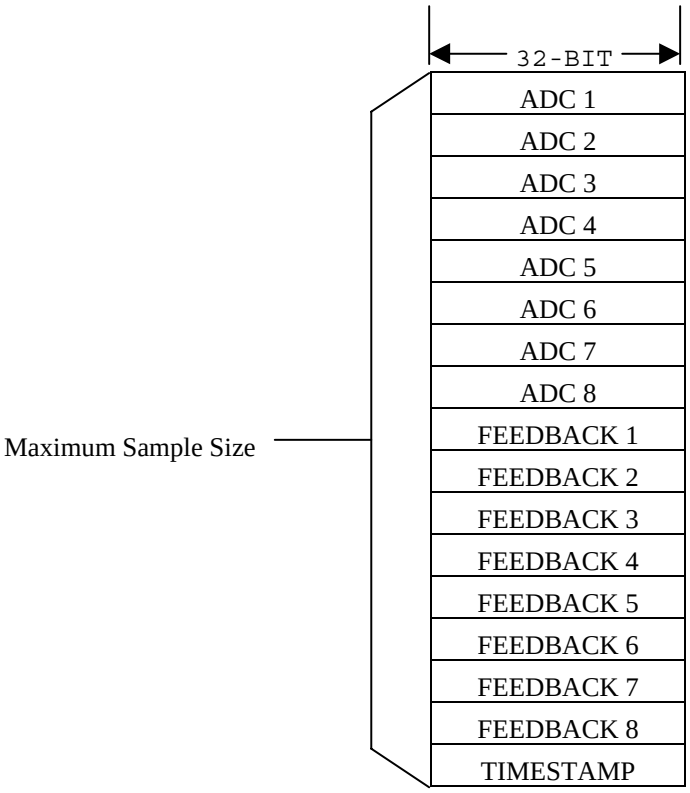
Arguments: long *DataArray – address of array where ADC data is to be stored.

Long *Num – returns the number of elements contained in the array. In streaming mode it is used to specify the amount of data to be copied to DataArray from the ring buffer. Upon return Num contains the number of elements copied to DataArray.

Long *DaqStat – contains data acquisition status: 0 = normal1 = overrun occurred.

Library Location: \esp6000.dll

Description: **esp_get_daq_data( )** retrieves data acquisition results. The sample size depends on the number of ADC and feedback channels tagged for acquisition. In streaming mode data is transferred to a ring buffer to be picked up by calls to esp_get_daq_data. To avoid errors data should be read at a fast enough rate to prevent the ring buffer from overflowing.



Returns: DAQREADY, DAQSTARTED, or DAQDONE

Hint:

Usage Example:

```
#include "esp6000.h"
main()
{
long error, servotick;
long Num, DaqStat, Mode, count;
float DataArray[512];

if (!esp_init_system())
{
printf("ESP6000 Not Initialized! \r\n");
exit(-1);
}
/* Set ADC Gain and Range*/
esp_set_adc_gain(V10);
esp_set_adc_range(BIPOLAR);

/* Set Acquisition Mode */
esp_set_daq_mode(1, 1, 1, 1, 2, 512);

esp_enable_daq();

esp_enable_motor(1);

esp_move_absolute(1,50.0);

while(!esp_daq_done())
/* Wait for DAQ End */
{

esp_get_daq_status(&count);
printf("%d acquisitions collected.\r",count);
}

esp_disable_daq();
/* Retrieve Data */
esp_get_daq_data(DataArray, &Num, &DaqStat);

/* check error status
*/esp_get_error_num(&error,&ServoTick);
if (error) printf("Error %d Reported!\r\n",
error);
}
```

See Also: esp_get_daq_status(), esp_set_daq_mode(), esp_enable_daq(), esp_disable_daq(), esp_get_daq_data()

esp_daq_data_to_file *Write Data Acquisition To File*

Synopsis:
#include "esp6000.h"
int esp_daq_data_to_file(char *FileName)

Arguments:
char *FileName
The name and path of the file to stream data to.

Library Location:
\esp6000.dll

Description:
esp_daq_data_to_file(char *FileName) allows streaming mode data acquisition to be directed to the specified file.

Returns:
ESPOK, ESPERROR

Hint:

Usage Example:

```
#include "esp6000.h"
main( )
{
      if(!esp_init_system( )
      {
      printf("ESP6000 Not Initialized");
      exit( -1);
      }
// Set ADC Gain and range
      esp_set_adc_gain(V10);
      esp_set_adc_range(BIPOLAR);

// setup the mode for continuous streaming
// start acquiring when enabled
// Data Acquisition on channels 1 & 2
// No position feedback
      esp_set_daq_mode(36,0,3,0,0,0);

// Turn on streaming to data file.
      esp_daq_data_to_file("c:\\data.dat);

// start streaming to file
      esp_enable_daq( );

// Collect data for 3 seconds
      wait(3);
// Stop Streaming and close the file
      esp_disable_daq( );
}
```

See Also:
esp_get_daq_done( ), esp_set_daq_mode( ), esp_enable_daq( ), esp_disable_daq( ), esp_get_daq-data( )

esp_set_portabe_dir — *Set Digital I/O Port A, B, & C Direction*
esp_get_portabe_dir — *Report Digital I/O Port A, B, & C Direction Setting*

Synopsis: #include “esp6000.h”
int esp_set_portabc_dir(long a, long b, long c)
int esp_get_portabc_dir(long *a, long *b, long *c)

Arguments: long a, b, c
port direction (1)PORT_OUTPUT or (0)PORT_INPUT

Library Location: \esp6000.dll

Description: **esp_set_portabc_dir()** defines digital I/O port direction. Digital I/O ports are located on both auxiliary I/O and digital I/O connectors on the controller card.

esp_get_portabc_dir() reports present port configuration.

Port A DIO signals are externally pulled-up via a 4.7KΩ resister to +5 volts.

NOTE
After system reset Ports A, B, and C are automatically configured as inputs.

Returns: ESPOK, ESPERROR

Hint: Define port direction with `esp_set_portabc_dir()` before using this function.

Usage Example:

```
#include “esp6000.h”

main()
{
if (!esp_init_system()) exit(-1);

/* Configure Ports A, B, C Directions */

esp_set_portabc_dir(PORT_OUTPUT, PORT_INPUT,
PORT_INPUT);
/* Set DIO_A Port */
esp_set_dio_porta(long 0x0FF);
}
```

See Also: esp_get_dio_porta(), esp_get_dio_portb(), esp_get_dio_portc(),

esp_set_dio_porta *Set Digital I/O Port A*
esp_get_dio_porta *Report Digital I/O Port A Status*

Synopsis: #include “esp6000.h”
int esp_set_dio_porta(long data)
int esp_get_dio_porta(long *data)

Arguments: long data
digital I/O Port A

Library Location: \esp6000.dll

Description: **esp_set_dio_porta()** writes specified value to digital I/O port A located on both auxiliary I/O and digital I/O connectors on the controller card. Port A is an 8-bit port starting from location bit-0 through bit-7.

esp_get_dio_porta() reports DIO port A status. Use function **esp_set_portabc_dir()** to define Port A as either an input or output.

Port A DIO signals are externally pulled-up via a 4.7KΩ resister to +5 volts.

PORT A							
Bit-7	Bit-8	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0

NOTE

After system reset Ports A, B, and C are automatically configured as inputs.

Returns: ESPOK, ESPERROR

Hint: Define port direction with **esp_set_portabc_dir()** before using this function.

Usage Example:

```
#include “esp6000.h”
main()
{
if (!esp_init_system()) exit(-1);

/* Configure Ports A, B, C Directions */
esp_set_portabc_dir(PORT_OUTPUT, PORT_INPUT,
PORT_INPUT);

/* Set DIO_A Port */
esp_set_dio_porta(long 0x0FF);
}
```

See Also: esp_get_dio_porta( ), esp_set_portabc_dir( )

esp_set_dio_portb *Write To Digital I/O Port B*
esp_get_dio_portb *Report Digital I/O Port B Status*

Synopsis: #include “esp6000.h”
int esp_set_dio_portb(long data)
int esp_get_dio_portb(long *data)

Arguments: long data
digital I/O Port B

Library Location: \esp6000.dll

Description: **esp_set_dio_portb()** writes specified value to digital I/O Port B located on both auxiliary I/O and digital I/O connectors on the controller card.

Port B is an 8-bit port starting from location bit-0 through bit-7.

Use function **esp_set_portabc_dir()** to define Port B as either an input or output.

esp_get_dio_portb() reports DIO port B status.

Port B DIO signals are externally pulled-up via a 4.7KΩ resister to +5 volts.

PORT B							
Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0

NOTE
After system reset Ports A, B, and C are automatically configured as inputs.

Returns: ESPOK, ESPERROR

Hint: Define port direction with `esp_set_portabc_dir()` before using this function. .

Usage Example:

```
#include “esp6000.h”
main()
{
if (!esp_init_system()) exit(-1);

/* Configure Ports A, B, C Directions */
esp_set_portabc_dir(PORT_OUTPUT, PORT_INPUT,
PORT_INPUT);
/* Set DIO_B Port */
esp_set_dio_portb(long 0x0FF);
}
```

See Also: esp_get_dio_porta( ), esp_get_dio_portc( ), esp_set_portabc_dir( )

esp_set_dio_portc *Write To Digital I/O Port C*
esp_get_dio_portc *Report Digital I/O Port C Status*

Synopsis: #include “esp6000.h”
int esp_set_dio_portc(long data)
int esp_get_dio_portc(long *data)

Arguments: long data
digital I/O Port C

Library Location: \esp6000.dll

Description: **esp_set_dio_portc()** writes specified value to digital I/O Port C located on both
auxiliary I/O and digital I/O connectors on the controller card.

Port A is an 8-bit port starting from location bit-0 through bit-7.

Use function **esp_set_portabc_dir()** to define Port C as either an input or output. **esp_get_dio_portc()** reports DIO port C status.

Port C DIO signals are externally pulled-up via a 4.7KΩ resister to +5 volts.

PORT C							
Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0

NOTE
After system reset Ports A, B, and C are automatically configured as inputs.

Returns: ESPOK, ESPERROR

Hint: Define port direction with `esp_set_portabc_dir()` before using this function. .

Usage Example:

```
#include “esp6000.h”
main()
{
if (!esp_init_system()) exit(-1);

/* Configure Ports A, B, C Directions */
esp_set_portabc_dir(PORT_OUTPUT, PORT_INPUT,
PORT_INPUT);

/* Set DIO_C Port */
esp_set_dio_portc(long 0x0FF);
}
```

See Also: esp_get_dio_porta(), esp_get_dio_portb(), esp_set_portabc_dir()

esp_set_dio_inhibit *Enable Or Disable The Motion Inhibit Bit*
esp_get_dio_inhibit *Report if Inhibit Bit is Enabled Or Disabled*

Synopsis: #include “esp6000.h”
int esp_set_dio_inhibit(long axis, long state)
int esp_get_dio_inhibit(long axis, long *state)

Arguments: long data
Axis number from 1-6

long state

1 – ENABLE

0 - DISABLE

Library Location: \esp6000.dll

Description: **esp_set_dio_inhibit()**

Enable or disable the motion inhibit function

Esp_get_dio_inhibit()

Indicates whether the inhibit function is enabled or disabled for the specified axis

Returns: ESPOK, ESPERROR

Hint: .

Usage Example:

```
#include “esp6000.h”
main()
{
int stat;

stat = esp_set_portabc_dir(PORT_OUTPUT,
PORT_INPUT, PORT_INPUT)
stat = esp_set_dio_inhibit_cfg(1,8,1)
stat = esp_set_dio_motion_cfg(1,0,1)
stat = esp_set_dio_inhibit(1,1)//Enable
stat = esp_set_dio_motion(1,1)//Enable
}
```

See Also: esp_get_dio_motion_cfg, esp_get_dio_motion_cfg, esp_set_dio_motion, esp_get_dio_motion

esp_set_dio_inhibit_cfg *Configure Motion Inhibit Digital I/O Bits*
esp_get_dio_inhibit_cfg *Get Digital I/O Motion Inhibit Configuration Information*

Synopsis: #include “esp6000.h”
int esp_set_dio_inhibit_cfg(long axis, long bit, long level)
int esp_get_dio_inhibit_cfg(long axis, long *bit, long *level)

Arguments: long axis

Axis number from 1-6.

long bit

bit number from 0 – 23

Digital I/O Bit Number that controls motion on the specified axis.

long level

Specifies the active level of the input bit

1 = Active High

0 = Active Low

Library Location: \esp6000.dll

Description: **esp_set_dio_inhibit_cfg()**

This function is used to configure the digital I/O to inhibit the motion when the active state of the specified I/O bit is set.

esp_get_dio_inhibit_cfg()

This function is used to obtain information on how the motion inhibit bits are set.

Returns: ESPOK, ESPERROR

Hint: .

Usage Example:

```
#include “esp6000.h”
main()
{
int stat;

stat = esp_set_portabc_dir(PORT_OUTPUT,
PORT_INPUT, PORT_INPUT)
stat = esp_set_dio_inhibit_cfg(1,8,1)
stat = esp_set_dio_motion_cfg(1,0,1)
stat = esp_set_dio_inhibit(1,1)//Enable
stat = esp_set_dio_motion(1,1)//Enable
```

```
}
```

See Also: esp_get_dio_motion_cfg, esp_get_dio_motion_cfg, esp_set_dio_inhibit, esp_get_dio_inhibit

esp_set_dio_motion *Enable Or Disable The Motion Complete Bit*
esp_get_dio_motion *Report if Motion Complete Bit is Enabled Or Disabled*

Synopsis: #include "esp6000.h"
int esp_set_dio_motion(long axis, long state)
int esp_get_dio_motion(long axis, long *state)

Arguments: long axis

Axis number from 1-6.

long state

1 – ENABLE

0 - DISABLE

Library Location: \esp6000.dll

Description: **esp_set_dio_motion()**

Enable or disable the motion complete function.

esp_get_dio_motion()

Indicates whether the motion complete function is enabled or disabled for the specified axis.

Returns: ESPOK, ESPERROR

Hint: .

Usage Example:

```
#include "esp6000.h"
main()
{
int stat;

stat = esp_set_portabc_dir(PORT_OUTPUT,
PORT_INPUT, PORT_INPUT)
stat = esp_set_dio_inhibit_cfg(1,8,1)
stat = esp_set_dio_motion_cfg(1,0,1)
stat = esp_set_dio_inhibit(1,1)//Enable
stat = esp_set_dio_motion(1,1)//Enable
}
```

See Also: esp_get_dio_motion_cfg, esp_get_dio_motion_cfg, esp_set_dio_inhibit, esp_get_dio_inhibit

esp_set_dio_motion_cfg *Configure Motion Complete Indication Digital I/O Bits*

esp_get_dio_motion_cfg *Report Motion Complete Indication Digital I/O Bits*

Synopsis: #include “esp6000.h”
int esp_set_dio_motion_cfg(long axis, long bit, long level)
int esp_get_dio_motion_cfg(long axis, long *bit, long *level)

Arguments: long axis

Axis number from 1-6.

long bit

bit number from 0 – 23

Digital I/O Bit Number that indicates motion complete status on the specified axis.

long level

Specifies the active level of the output bit

1 = Active High

0 = Active Low

Library Location: \esp6000.dll

Description: **esp_set_dio_motion_cfg()**

This function is used to configure the digital I/O to indicate the motion complete status. Motion is complete (axis is at standstill) when the specified I/O bit is in the active state.

esp_get_dio_motion_cfg()

This function is used to obtain information on how the motion complete status indication bits are set.

Returns: ESPOK, ESPERROR

Hint: .

Usage Example:

```
#include “esp6000.h”
main()
{
int stat;

stat = esp_set_portabc_dir(PORT_OUTPUT,
PORT_OUTPUT, PORT_INPUT)
stat = esp_set_dio_inhibit_cfg(1,8,1)
stat = esp_set_dio_motion_cfg(1,0,1)
```

```
stat = esp_set_dio_inhibit(1,1)//Enable
stat = esp_set_dio_motion(1,1)//Enable
}
```

See Also: esp_get_dio_motion_cfg, esp_get_dio_motion_cfg, esp_set_dio_inhibit, esp_get_dio_inhibit

5.6.3 User Programming

5.6.3.1 Visual C

Overview

The user must include the ESP6000 Dynamic Link Library, **esp6000.dll,** in their application program. Probably the easiest way is to statically link the import library to their project. The import library contains information that Windows uses to locate the code in the DLL. The user can also use the LoadLibrary( ) function in their project. If the user chooses too use this method, they should make sure that they check the return value from the LoadLibrary( ) function to verify that the library was found and is loaded into memory. Once the library is loaded the user is ready to start programming. The function esp_init_system( ) must be the first function called. This function establishes shared memory and opens the initial communications to the ESP6000 controller card. A complete list of "C" function prototypes is available in the **esp6000.h** header file. This file should be included in all the users "C" source code.

Examples

There are several examples included in the install disk in the "\newport\esp6000\vc" directory. If the user is using Microsoft Foundation Classes (MFC) they will want to look at the term485.cpp example.

5.6.3.2 Visual Basic

Overview

Visual basic is probably the easiest Windows Programming language to use. We have included a file called **esp6000.bas** in the …\ESP6000\apps\VB directory. The user should add this file to their project and then they are ready to start programming. As with any programming language, the user should be sure they call the esp_init_system( ) function first.

Examples

Refer to the utilities disk in the "VB" directory for examples they can use to get their project up and running.

5.6.3.3 LabVIEW

LabVIEW is a graphically-oriented programming tool designed for scientific and control applications.

Each ESP6000 LabVIEW API call is developed and saved as a virtual instrument (.vi) file. VI's are available through the ESP6000 LabVIEW library, ESP6000.LLB.

Example(s)

All VI's have two modes of operation, simulated (default) and real execution. VI's automatically return to simulated mode after execution. If applicable, a single VI may include both read (default) and set parameter functionality. VI's automatically return to read mode after execution. Input parameters are displayed on the left side of the panel and output parameters on the right side. A representative front panel is shown in **Figure 5.6.3.3.**

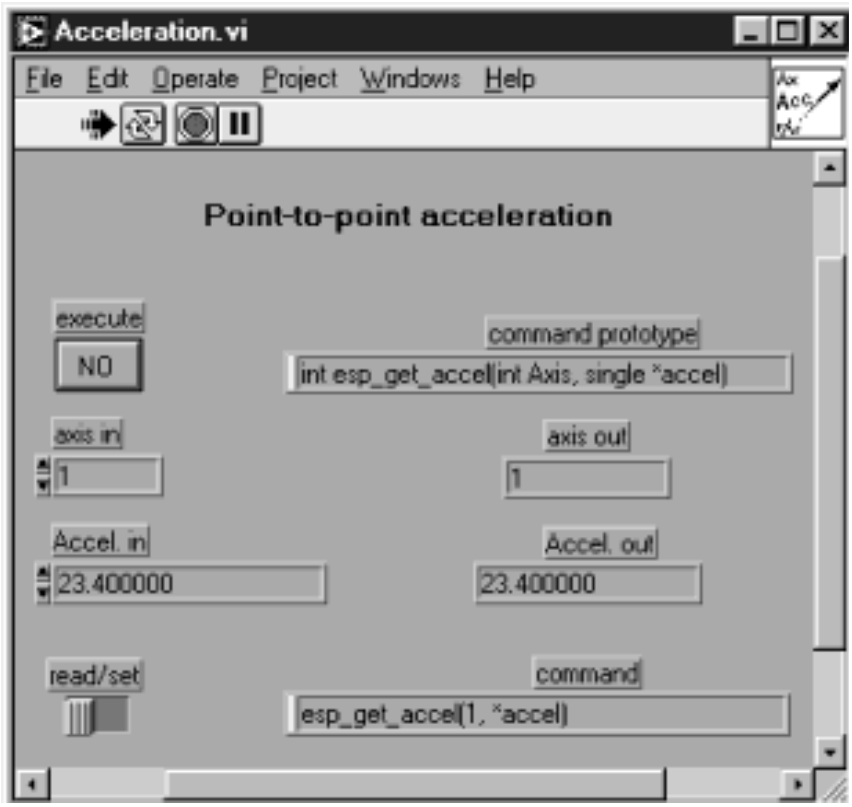


Figure 5.6.3.3: VI Front Panel

For all VI's, parameter boxes are located in the same relative position on both the wiring diagram(s) and the front-panel. Each VI is configured to handle up to 3 variables (input/output). If an axis number is a variable, it is always the first variable. If more than three variables are needed, they are grouped in arrays or bundles. Axis/channel parameters (when present) have inputs and outputs for easy flow control. In addition to parameters, each VI returns a command prototype and an actual command as presented to the DLL.

The returned command strings can be used for monitoring, troubleshooting, or tutorial purposes.

5.6.4 Error Handling

The ESP6000 maintains a 10-word, First-In, First-Out (FIFO) buffer error message. Errors encountered are stored in the order received along with a 'servo tick' time-stamp. The default servo tick resolution is 409msec.

Most ESP6000 API function calls return a value, -1 (error exists) or 0 (no error exists), which indicate whether an error has been detected or not. However, because the ESP6000 board queues commands, the returned value is not a direct indicator of function correctness. The returned value only indicates that an unread error now exists in the error buffer. The error may be the result of a previous function call.

NOTE

A returned –1 value only indicates that an unread error now exists in the error buffer. The error may be the result of a previous API function call.

There are two (2) dedicated functions for checking errors. The first is esp_get_string(char *Error, long *ErrorNum, long *ServoTick). ErrorNum (if non-zero) will contain the error number that has occurred. The ErrorString will contain a text message corresponding to the ErrorNum. The tick count is the value of the Servo Clock counter at the time of the error. The second function, esp_get_error_num(long *error, long *ServoTick) works like the previous function without the text message.

Error checking should be done after critical API function calls within the application program. Error checking is most commonly performed after 'block' API calls to verify that all commands were transmitted and properly executed.

When an error is detected (i.e., ErrorNum equals non-zero) the application should flush the error queue by calling the error-checking API functions calls until ErrorNum equals 0.

5.6.5 Multiple Board Support

5.6.5.1 Introduction

Under Windows NT Operating System and device driver version 3.00, users can access up to four ESP6000 cards (24-axes) with one application program. Newport supplies several unique API calls to facilitate the management of multiple boards.

Since ESP6000 boards are PCI based and do not have fixed addresses, it is not possible to predetermine which will be logical address 0, 1, 2, or 3 from one PC model to another. This is a function of the computer motherboard designer and/or BIOS. Typically, some simple testing is necessary to associate logical board number with physical board.

5.6.5.2 ESP-Util

Windows program **ESP-Util,** version 2.07 and later, supports multiple ESP6000 cards running in a Windows NT environment. The program automatically detects the presence of multiple cards and prompts the user to select the target board (See **Figure 5.6.5.2**).

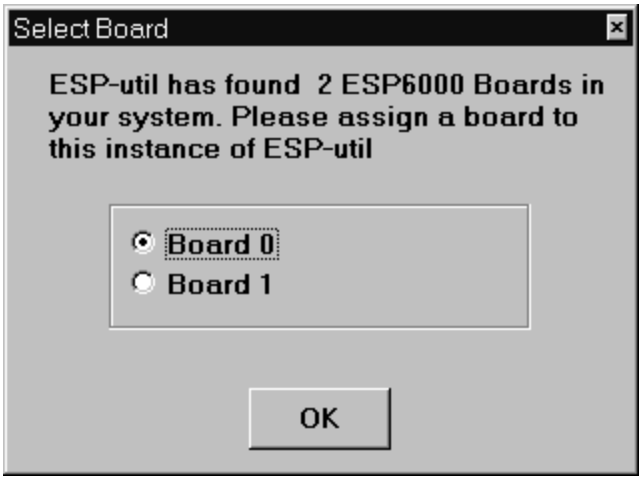


Figure 5.6.5.2: Select Target Board

5.6.5.3 ESP-Tuning

Windows program **ESP-Tuning,** version 2.07, does not support multiple ESP6000 cards. **ESP-Tuning** always defaults to board 0. Therefore, only one (1) ESP6000 controller card can be used to optimized PID servo settings with this program.

5.6.5.4 API Calls Supporting Multiple ESP6000 Boards

To communicate with multiple boards under Windows NT the user will need to call the API function **esp_set_target_board(int target)** to specify which board is to be addressed for the proceeding commands.

Example #1:

```
// Initialize ESP6000 board #0
esp_set_target_board(0);
status = esp_init_system();
if (status == -1)
     Error();

// Initialize ESP6000 Board #1
esp_set_target_board(1);
status = esp_init_system();
if (status == -1)
     Error();

// move axis 1 on board 0
esp_set_target_board(0); // address board 0
esp_enable_motor(1);     // turn ON power to axis1
esp_move_absolute(1,2.2) // move axis one to 2.2


// move axis 1 on board 1
esp_set_target_board(1); // address board 1
esp_enable_motor(1);     // move axis one to 5.0
```

This function call only needs to be made prior to addressing a different ESP6000.

The user can check to see how many boards are in the system by using the API function call **esp_get_num_boards(int *Boards,**

Int BoardsUsed);

The parameter **Boards** will tell the user how many ESP6000 boards were found in the system. The **BoardUsed** parameter returns a bit pattern that tells how many boards are in use by an Application.

Example #2:

BoardUsed	Meaning
0001	Board 0 in use
0010	Board 1 in use
0011	Boards 0 and 1 in use

Table 5.6.5.4: BoardUsed by an Application

Section 6 – Motion Control Tutorial

6.1 Motion Systems

A schematic of a typical motion control system is shown in **Figure 6.1.1.**

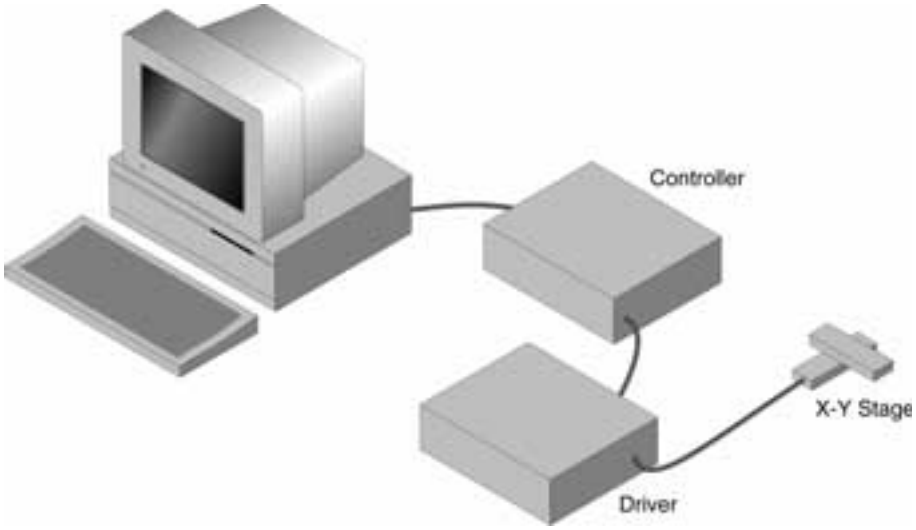


Figure 6.1.1: Typical Motion Control Systems

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 specs 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.

6.2 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.

6.2.1 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.

6.2.2 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.

6.2.3 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 end 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 range. When finished, the Error data is plotted on a graph similar to **Figure 6.2.1.**

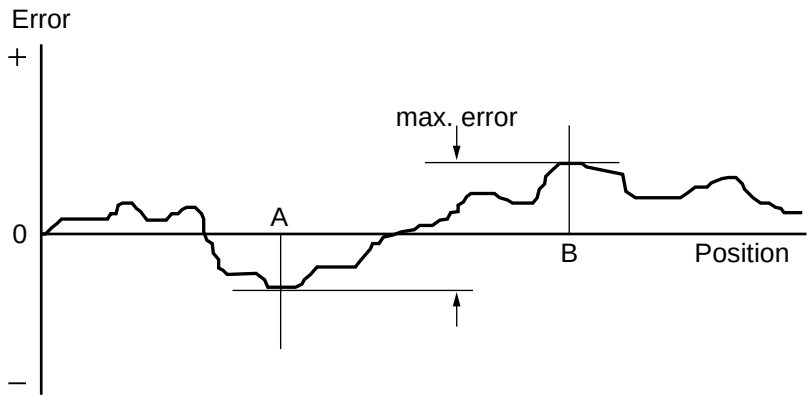


Figure 6.2.1: Position Error Test

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 this value.

6.2.4 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 6.2.2** and **Figure 6.2.3** shows two extreme cases.

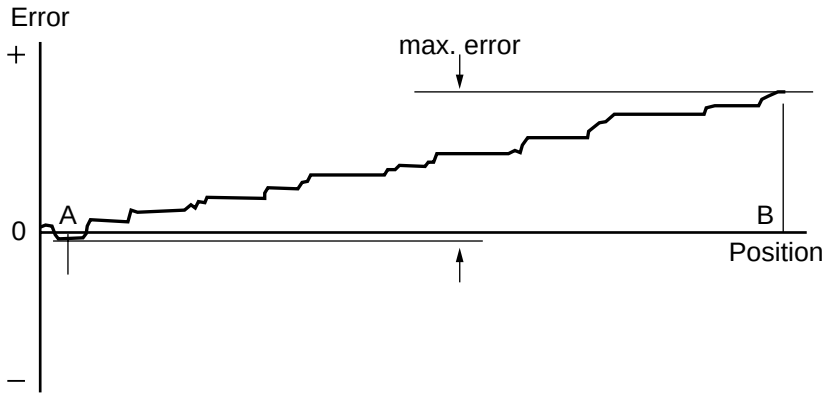


Figure 6.2.2: High Accuracy for Small Motions

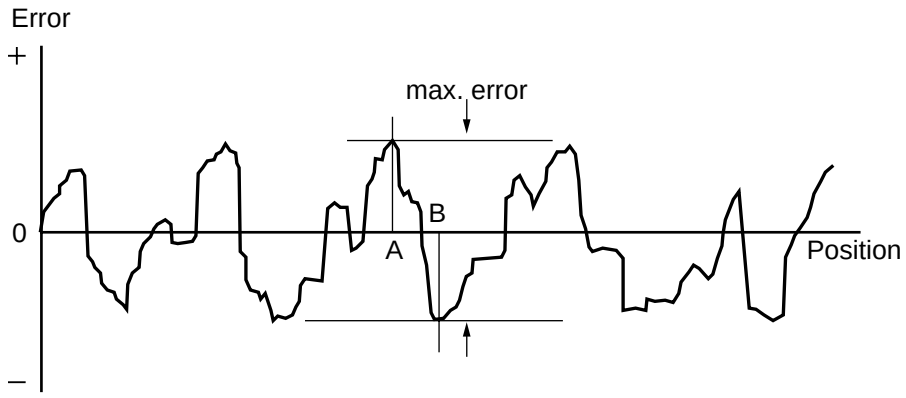


Figure 6.2.3: Low Accuracy for Small Motions

Both error plots from **Figure 6.2.2** and **Figure 6.2.3** have a similar maximum Error. But, if the user compares the maximum Error for small distances, the system in **Figure 6.2.3** shows significantly larger values.

For applications requiring high accuracy for small motions, the system in **Figure 6.2.2** 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.

6.2.5 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 ESP, this is also the resolution of the encoder.

Keeping in mind that the servo loop is a digital loop, the Resolution can be also be viewed as the smallest position increment that the controller can handle.

6.2.6 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.

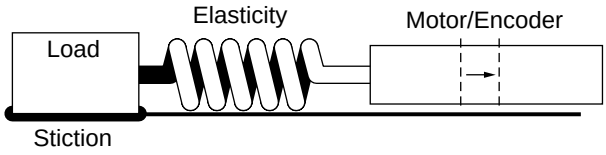


Figure 6.2.4: Effect of Stiction and Elasticity on Small Motions

Figure 6.2.4 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 6.2.5.**

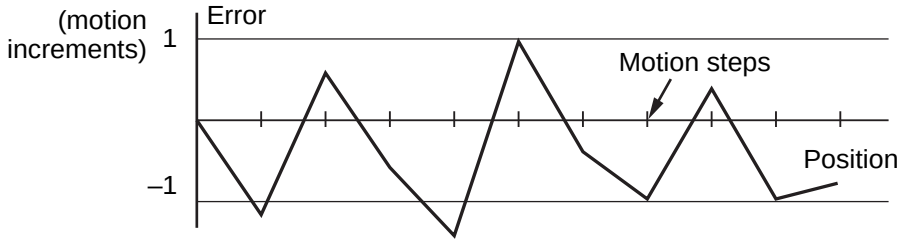


Figure 6.2.5: Error Plot

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.

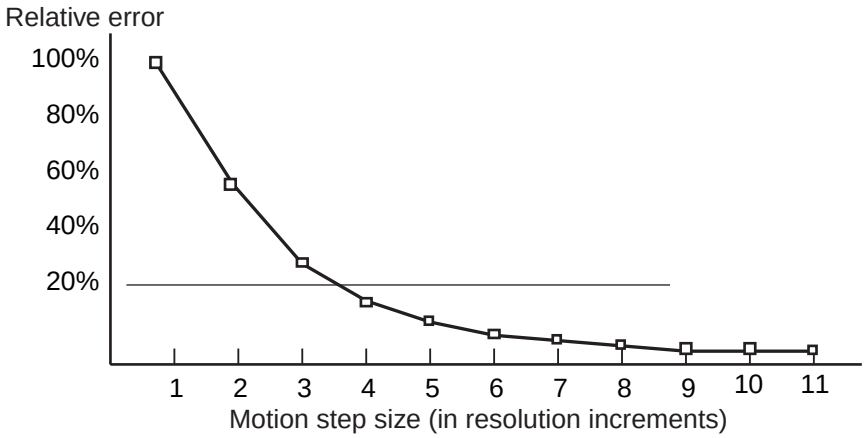


Figure 6.2.6: Error vs. Motion Step Size

Figure 6.2.6 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.

6.2.7 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 position variation of all targets when the same motion sequence is repeated a large number of times. It is a relative, not absolute, error between identical motions.

6.2.8 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 have involved 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 different directions could make a significant difference.

In generating the plot in **Figure 6.2.1** 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 6.2.7.**

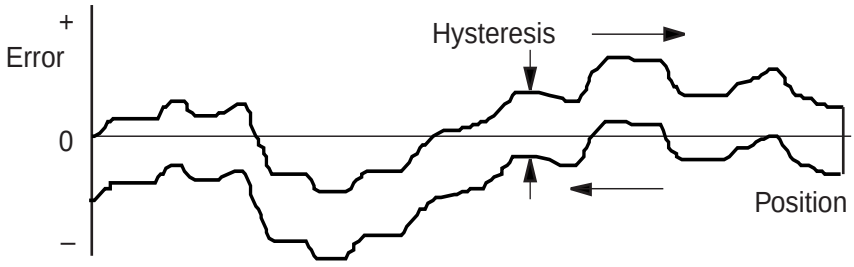


Figure 6.2.7: Hysteresis Plot

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.

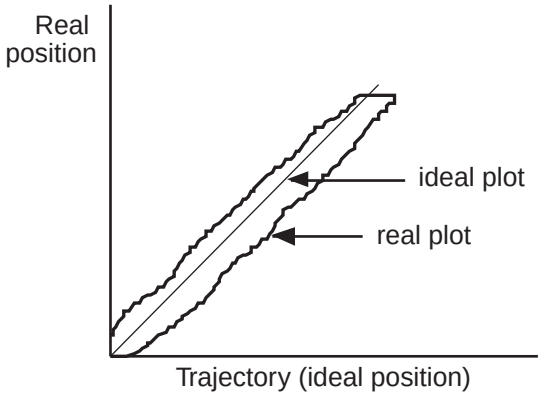


Figure 6.2.8: Real vs. Ideal Position

To justify a little more why we call this Hysteresis, lets do the same graph in a different format (**Figure 6.2.8**). Plotting the real versus the ideal position will give the user a familiar hysteresis shape.

6.2.9 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 6.2.9.** Pitch is rotation around the Y axis, Roll is rotation around the X axis, and Yaw is rotation around the Z axis.

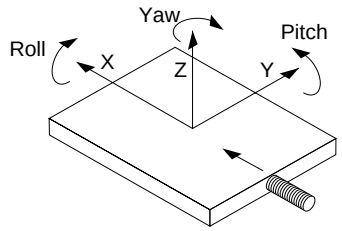


Figure 6.2.9: Pitch, Roll and Yaw Motion Axes

The problem with this definition is that, through correct, it is difficult to remember. A more graphical representation is presented in **Figure 6.2.10.** 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.

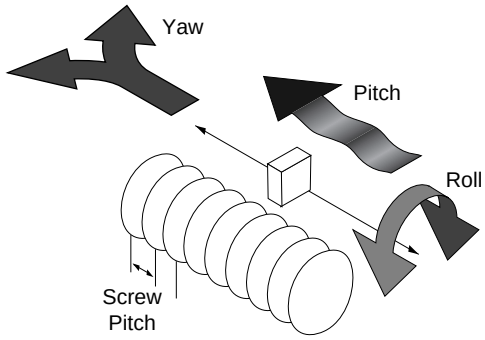


Figure 6.2.10: Pitch, Yaw and Roll Motion Axes

6.2.10 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 6.2.11**).

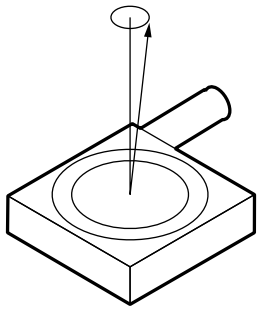


Figure 6.2.11: Wobble

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.

6.2.11 Load Capacity

There are two types of loads that are of interest for motion control applications: *static* and *dynamic loads.*

The static Load 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.

6.2.12 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 ESP, the default Maximum Velocity should be increased. The hardware and firmware are tuned for a particular maximum velocity that cannot be exceeded.

6.2.13 Minimum Velocity

The Maximum 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 a properly tuned servo loop could move the stage 0.1 µm in about 20 ms. The position and velocity plots are illustrated in **Figure 6.2.12.**

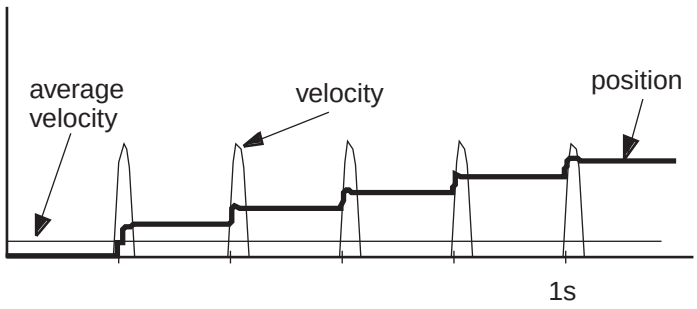


Figure 6.2.12: Position, Velocity and Average Velocity

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.

6.2.14 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.

6.2.15 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 (or 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. Through 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.

6.2.16 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 ESP 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.

6.3 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 ESP implements a PID servo loop with velocity and acceleration feed-forward.

The basic diagram of a servo loop is shown in **Figure 6.3.1.** 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.

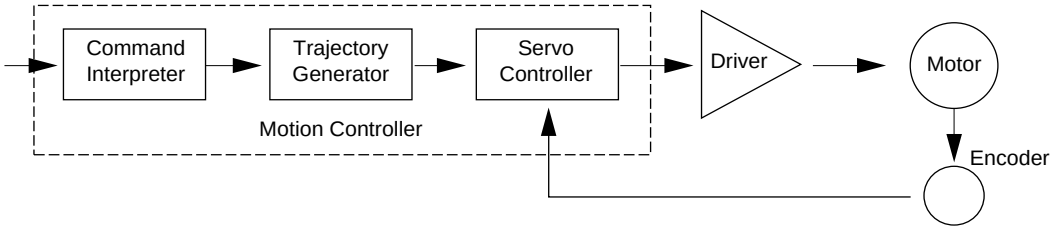


Figure 6.3.1: Servo Loop

6.3.1 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:

$$\mathbf{K_p \bullet e + K_I \int e\,dt + K_d \bullet \frac{de}{dt}}$$

where $\mathbf{K_p}$ = Proportional gain factor

$\mathbf{K_i}$ = integral gain factor

$\mathbf{K_d}$ = derivative gain factor

$\mathbf{E}$ = 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 difficult to achieve without some understanding of its behavior of servo loops.

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 6.3.2** shows its configuration.

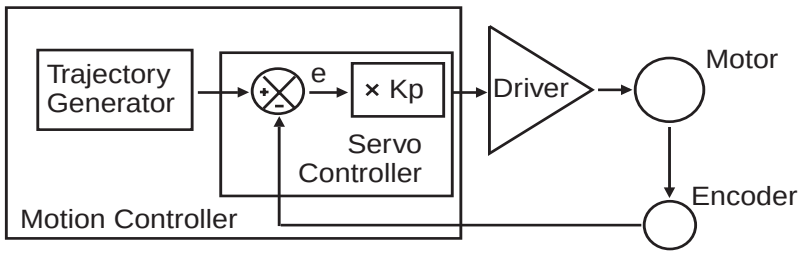


Figure 6.3.2: P Loop

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 (See **Figure 6.3.3**).

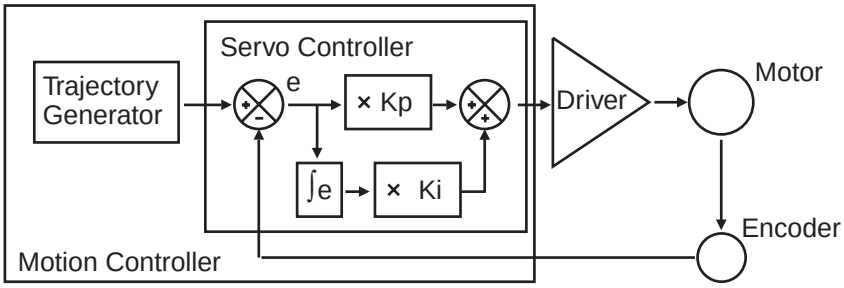


Figure 6.3.3: PI Loop

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 6.3.4 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.

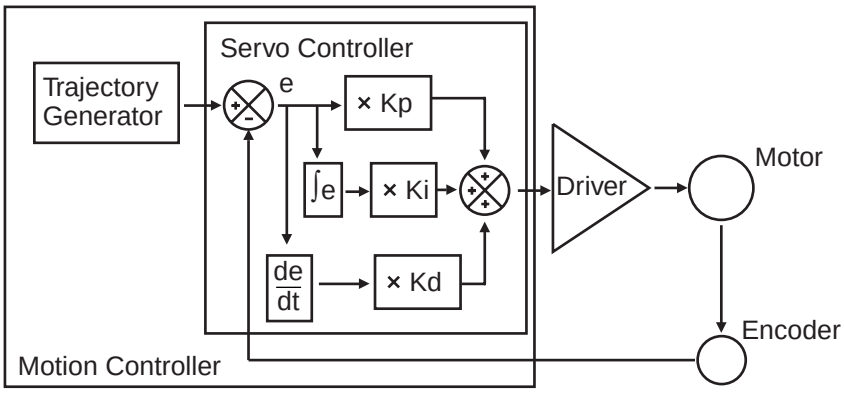


Figure 6.3.4: PID Loop

6.3.2 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 a trapezoidal shape (See **Figure 6.3.5**).

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 6.3.6**). But, by adding this signal to the closed loop has the effect of significantly reducing the work "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.

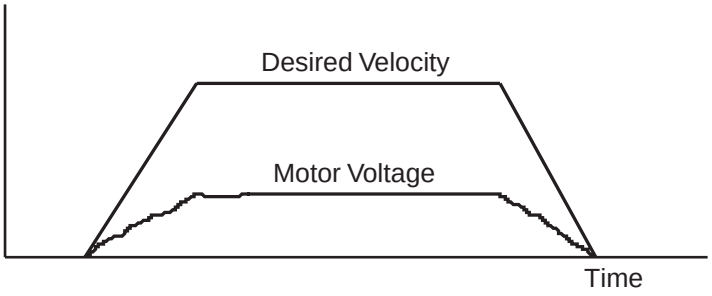


Figure 6.3.5: Trapezoidal Velocity Profile

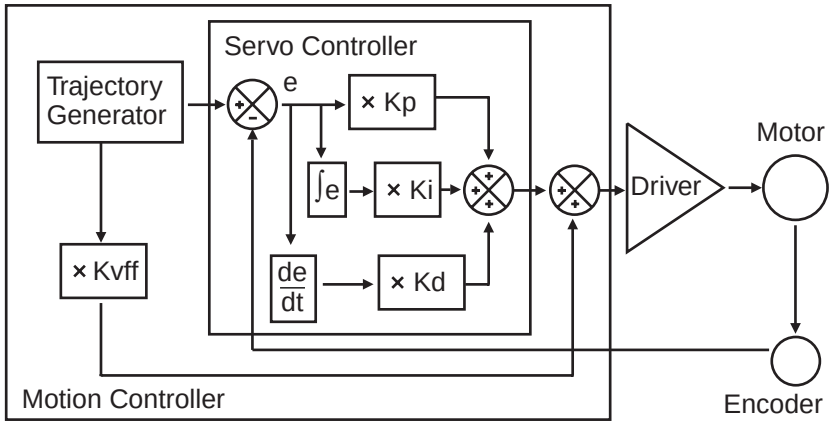


Figure 6.3.6: PID Loop with Feed-Forward

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 is 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. (See **Figure 6.3.7**).

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.

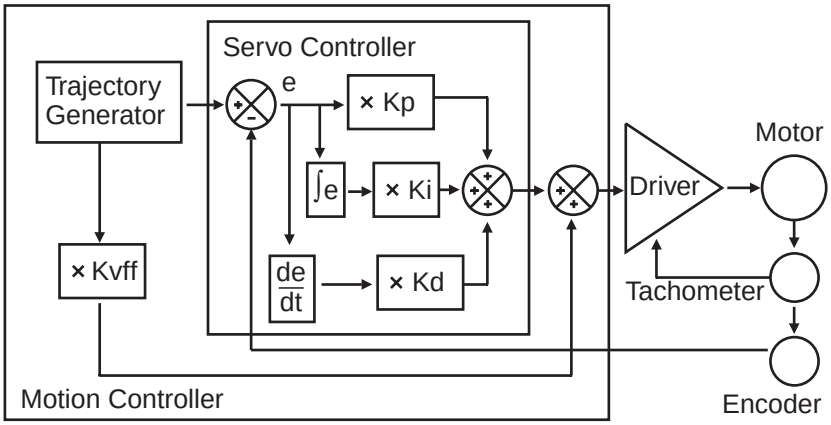


Figure 6.3.7: Tachometer-Driven PIDF Loop

6.4 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.

6.4.1 Move

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 6.4.1**). For this reason, this type of motion is called a trapezoidal motion.

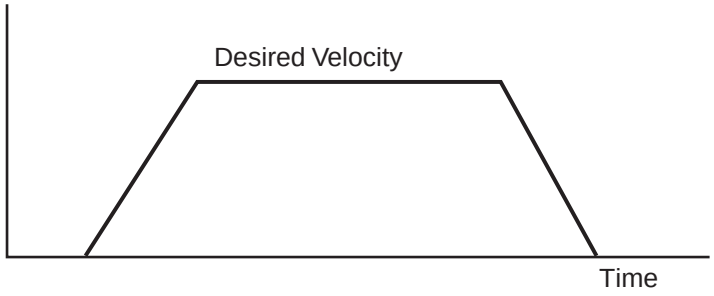


Figure 6.4.1: Trapezoidal Motion Profile

The position and acceleration profiles relative to the velocity are shown in **Figure 6.4.2.**

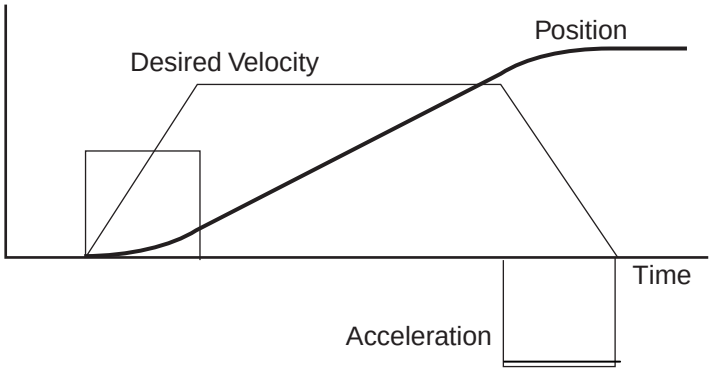


Figure 6.4.2: Position and Acceleration Profiles

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 ESP allow the user to change them even during the motion.

6.4.2 Jog

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 ESP offers two jog speeds. Both high speed and low speed are user programmable. The jog acceleration is also ten times smaller than the programmed maximum acceleration values.

6.4.3 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, 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 replaced accurately in space before running the application.

To achieve this precise position referencing, the ESP 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 ESP) 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 6.4.3: Home (Origin) Switch and Encoder Index Pulse

A Home switch (**Figure 6.4.3**) 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 in 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 home switch transition and then the very first index pulse (**Figure 6.4.4**).

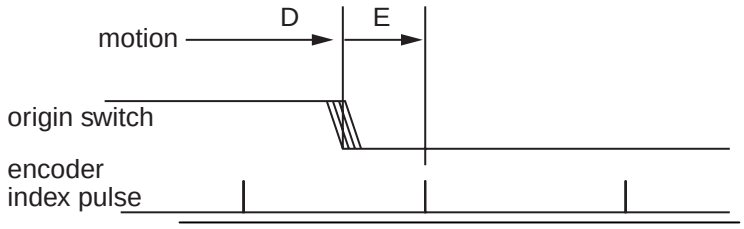


Figure 6.4.4: Slow-Speed Origin Switch Search

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.

The routine described above could work but has one problem. Using the low speeds, it could take a very long time if the stage happens to start from the opposite end of travel. To speed things up, we can have the stage 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 6.4.5.**

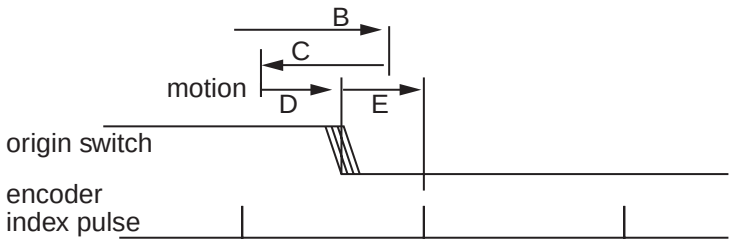


Figure 6.4.5: High/Low-Speed Home (Origin) Switch Search

Motion segment B is performed at high speed, with the pre-programmed home search speed. When the home 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 home switch transition, the routine will look like **Figure 6.4.6.**

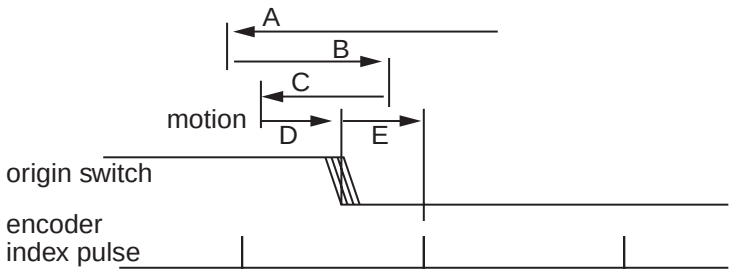


Figure 6.4.6: Home (Origin) Search from Opposite Direction

The ESP moves at high speed up to the home 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 spend other commands during its execution, unless it is for emergency purposes.

6.5 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 6.5.1**).

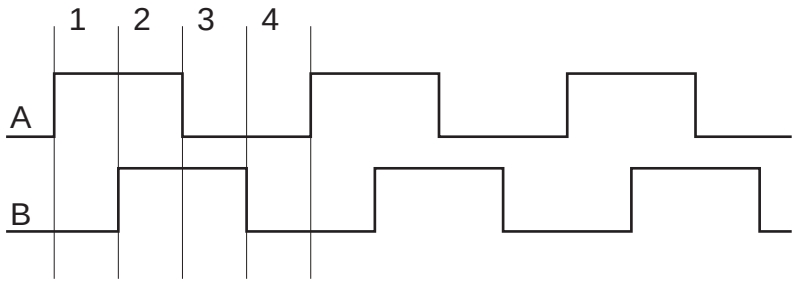


Figure 6.5.1: Encoder Quadrature Output

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, an encoder with 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 6.5.2**).

The encoder read head has three major components: a light source, a mask and a detector (**Figure 6.5.3**). The mask is a small-like piece, having identically spaced transparent and opaque lines.

If there is movement by either the scale or read head, light will be blocked by any overlay, or pass through otherwise (**Figure 6.5.4**).

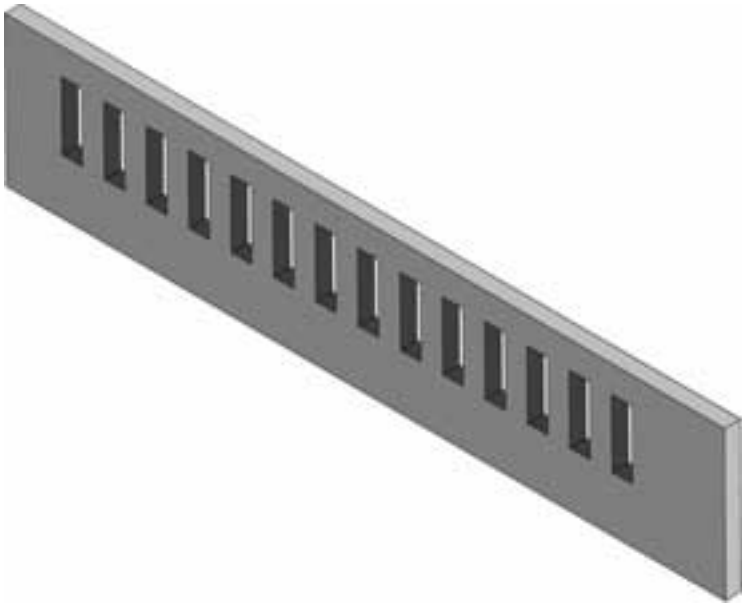


Figure 6.5.2: Optical Encoder Scale

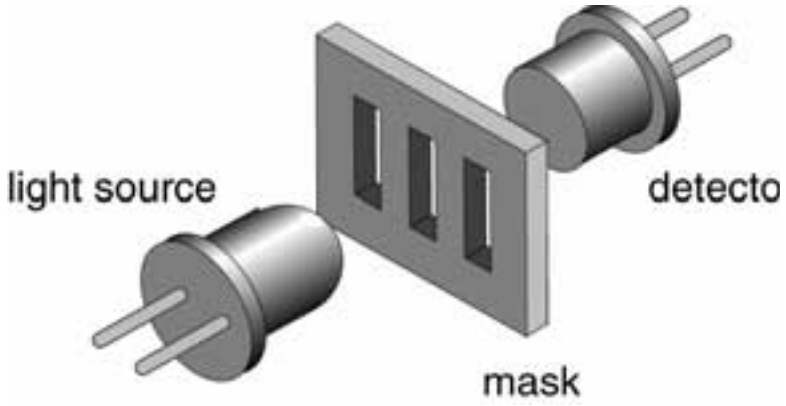


Figure 6.5.3: Optical Encoder Read Head

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 6.5.5**).

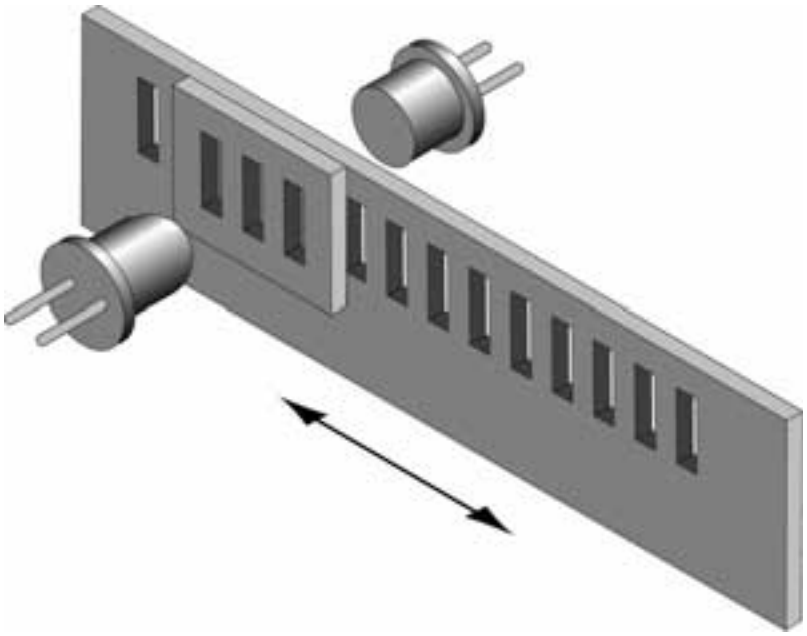


Figure 6.5.4: Single-Channel Optical Encoder Scale and Read Head Assembly

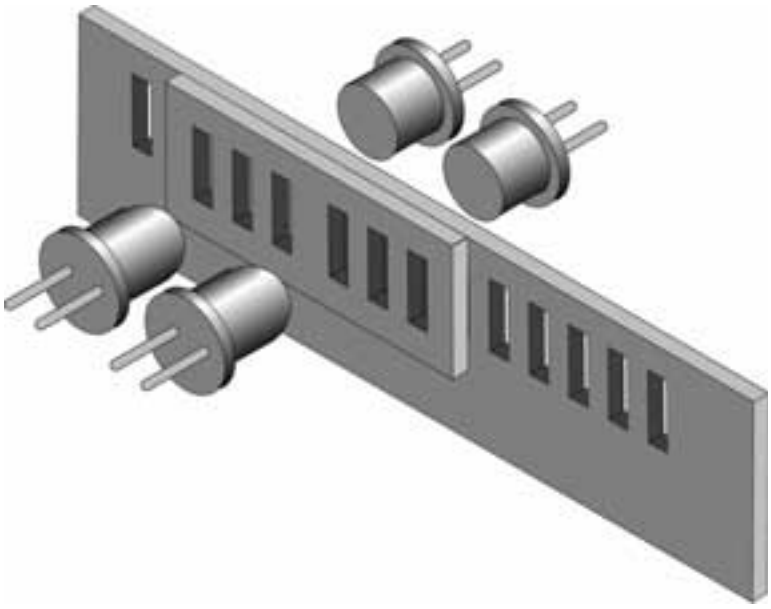


Figure 6.5.5: Two-Channel Optical Encoder Scale and Read Head Assembly

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 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.

6.6 Motors

There are many different types of electrical motors, each one being best suitable for certain kind of applications. The ESP supports two of the most popular types: *stepper motors* and *DC motors.*

Other technologies like interferometry or holography can be used but they are significantly more expansive and need more space.

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.

6.6.1 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 (called phases), the rotor will try to find one of these stable points and stay there. 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 6.6.1 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.

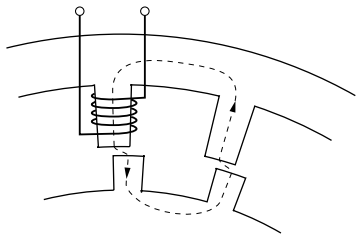


Figure 6.6.1: Stepper Motor Operation

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 advantages to a new stable position by means of several stator phases that have the teeth slightly offset from each other. To illustrate this, **Figure 6.6.2** 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).

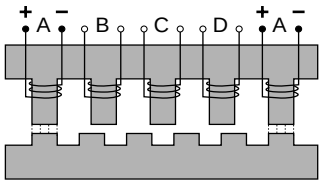


Figure 6.6.2: Four-Phase Stepper Motor

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 6.6.3**):

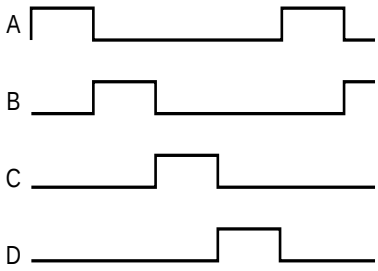


Figure 6.6.3: Phase Timing Diagram

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 6.6.4**).

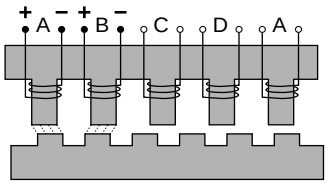


Figure 6.4.4: Energizing Two Phases Simultaneously

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 6.6.5.**

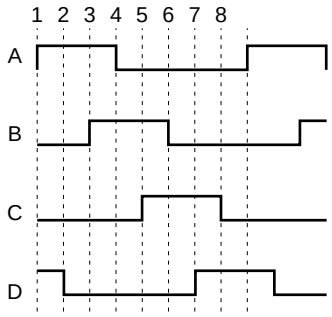


Figure 6.6.5: Timing Diagram, Half-Stepping Motor

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 teeth 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 6.6.6**).

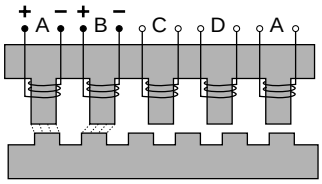


Figure 6.6.6: Energizing Two Phases with Different Intensities

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 6.6.7.**

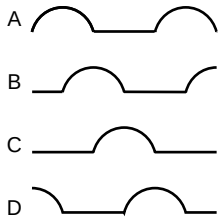


Figure 6.6.7: Timing Diagram, Continuous Motion (Ideal)

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 6.6.8**).

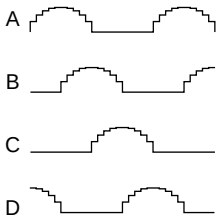


Figure 6.6.8: Timing Diagram, Mini-Stepping

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 similar 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 6.6.9**).

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 a stable point.

If external forces try to move the rotor (**Figure 6.6.10**), the magnetic flux will fight back. The more teeth misalignment exists, the larger the generated torque.

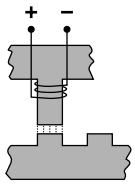


Figure 6.6.9: Single Phase Energization

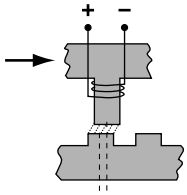


Figure 6.6.10: External Force Applied

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 6.6.11**).

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 6.6.12.** The maximum torque is obtained at one quarter of the tooth spacing, which is equivalent to one full step.

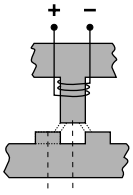


Figure 6.6.11: Unstable Point

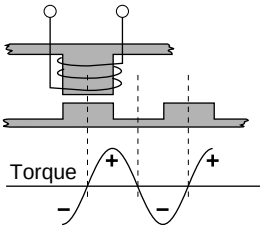


Figure 6.6.12: Torque and Tooth Alignment

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. When mini- and micro-stepping motors are used in open-loop applications there is inherent error, but advanced controllers like the ESP6000 can control the stepper motors with closed loop operation to eliminate this problem.

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.

6.6.2 DC Motors

A DC motor is similar to a permanent magnet stepper motor with an added internal phase commutator (**Figure 6.6.13**).

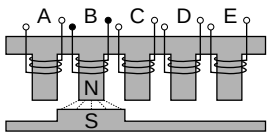


Figure 6.6.13: DC Motor

Applying 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 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
- Higher 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).

6.7 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 ESP 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 ESP.

6.7.1 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 6.7.1**).

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 6.7.2**).

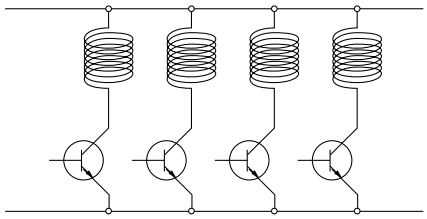


Figure 6.7.1: Simple Stepper Motor Driver

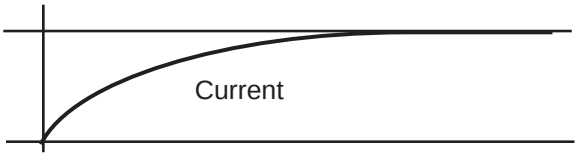


Figure 6.7.2: Current Build-up in Phase

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 nominal one (**Figure 6.7.3**).

How fast the current reaches its nominal value depends on three factors: the winding's inductance, resistance and the voltage applied to it.

The inductance cannot be reached. But the voltage can be temporarily to bring the current to its desired level faster. The most widely used technique is a high voltage chopper.

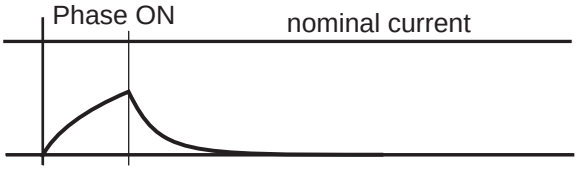


Figure 6.7.3: Effect of a Short ON Time on Current

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 6.7.4**).

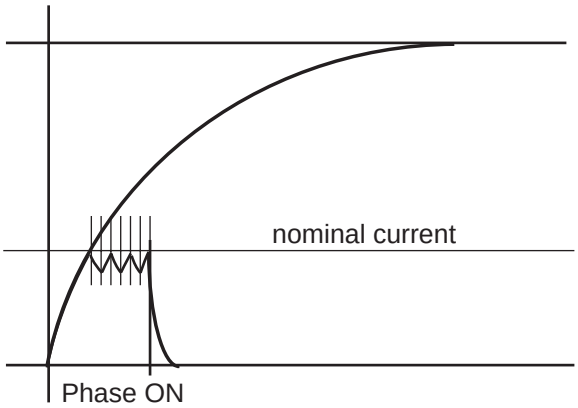


Figure 6.7.4: Motor Pulse with High Voltage Chopper

Once the desired current value is reached, a chopper circuit activates to keep the current close to the nominal value.

6.7.2 Unipolar – Bipolar Drivers

In the examples described in Section 6.7.1: 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 6.7.5** 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.

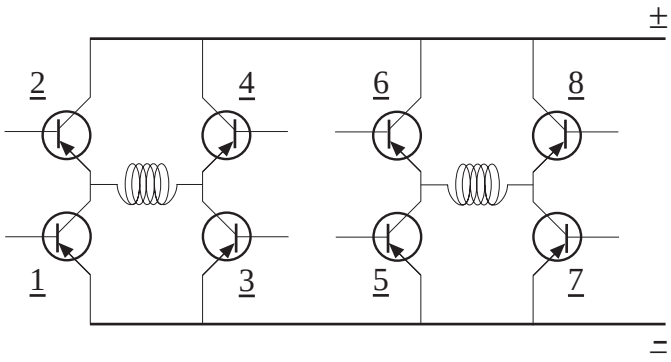


Figure 6.7.5: Dual H-Bridge Driver

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.

6.7.3 DC Motor Drivers

There are three major categories of DC motor drivers. The simplest one is a voltage amplifier (**Figure 6.7.6**).

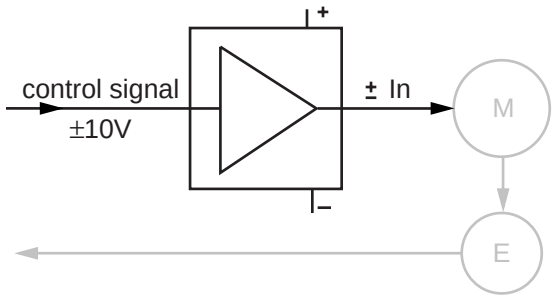


Figure 6.7.6: DC Motor Voltage Amplifier

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.

The second type of DC motor driver is the current driver, also called a torque driver (**Figure 6.7.7**).

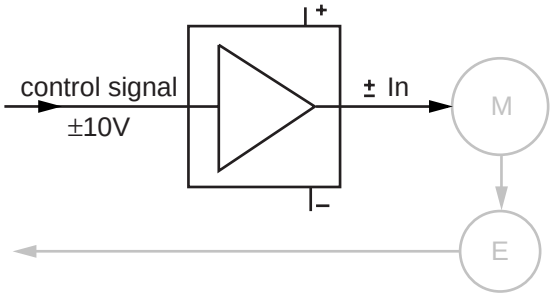


Figure 6.7.7: DC Motor Current Driver

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.

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 6.7.8**).

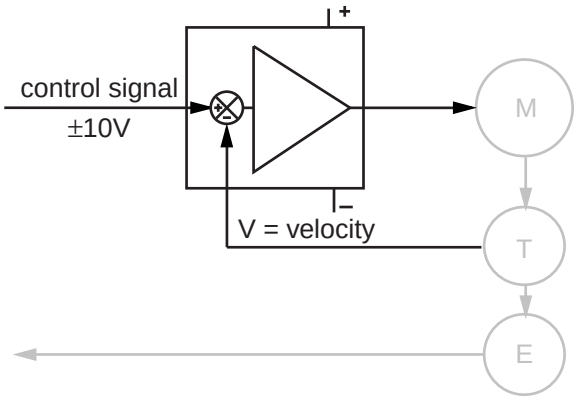


Figure 6.7.8: DC Motor Velocity Feedback Driver

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 6.7.9**).

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.

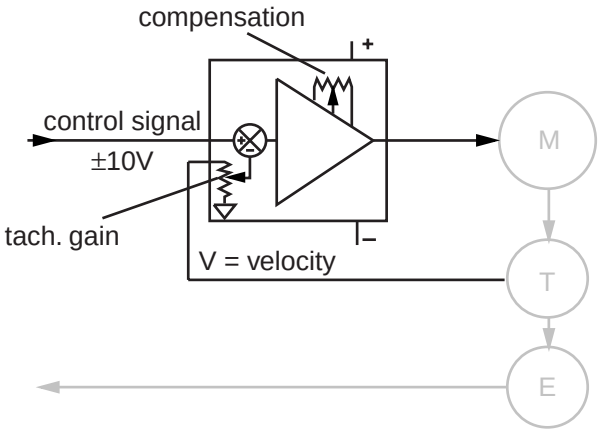


Figure 6.7.9: DC Motor Tachometer Gain and Compensation

The UniDrive can be configured either as a current driver or a velocity driver. When used with an ESP-compatible stage, the ESP6000 controller card will configure the UniDrive for optimum stage performance.

PWM Drivers

Even though linear amplifiers are simpler 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 7 – Servo Tuning

7.1 Tuning Principles

The ESP6000 controller uses a PID servo loop with feed-forward. Servo tuning sets the **K_p, K_I,** and **K_d,** 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.

7.2 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.

7.2.1 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 ESP6000 controller avoids this problem by providing 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. Refer to Section 4 for a detailed description of the utility and its operation.

7.2.2 Correcting Axis Oscillation

There are three parameters that can cause oscillation. The most likely to induce oscillation is $\mathbf{K_I}$, followed by $\mathbf{K_p}$ and $\mathbf{K_d}$. Start by setting $\mathbf{K_I}$ to zero and reducing $\mathbf{K_p}$ and $\mathbf{K_d}$ by 50%.

If oscillation does not stop, reduce $\mathbf{K_p}$ 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.

7.2.3 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 $\mathbf{K_p}$ and $\mathbf{K_d}$**.** It is especially common after performing the procedures described in paragraph 7.2.2.

First increase $\mathbf{K_p}$ 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 $\mathbf{K_d}$ parameter.

Increase it by a factor of 2 while monitoring the following error. As $\mathbf{K_d}$ 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 K_d.

If $\mathbf{K_d}$ is further increased, at some point oscillation will reappear, usually at a higher frequency. Avoid this by keeping $\mathbf{K_d}$ at a high enough value, but not so high as to re-introduce oscillation.

Increase $\mathbf{K_p}$ successively by approximately 20% until signs of excessive ringing appear again.

Alternatively increase $\mathbf{K_d}$ and $\mathbf{K_d}$ cannot eliminate overshoot and ringing at stop. This indicates $\mathbf{K_p}$ is larger than its optimal value and should be reduced. At this point, the PID loop is very tight.

Ultimately, optimal values for $\mathbf{K_p}$ and $\mathbf{K_d}$ 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 the user is satisfied with the dynamic response of the PID loop but the stage does not always stop accurately, modify the integral gain factor $\mathbf{K_i}$. As described in the Motion Control Tutorial section, the $\mathbf{K_I}$ 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 $\mathbf{K_d}$.

Start with the integral limit (**II**) set to a high value and $\mathbf{K_I}$ value at least two orders of magnitude smaller than $\mathbf{K_p}$. Increase its value by 50% at a time and monitor overshoot and final position at stop.

If intolerable overshoot develops, increase the $\mathbf{K_d}$ factor. Continue increasing $\mathbf{K_I}$ and $\mathbf{K_d}$ alternatively until an acceptable loop response is obtained. If oscillation develops, immediately reduce $\mathbf{K_i}$.

Remember that any finite value for $\mathbf{K_I}$ 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 $\mathbf{K_I}$ 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.

7.2.4 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 ESP6000 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 $\mathbf{K_i}$ that gives acceptable performance. The integral gain factor can cause overshoot and oscillations.

A summary of servo parameter functions is listed in **Table 7.2.1.**

Parameter	Function	Value Set Too Low	Value Set Too High
$\mathbf{K_p}$	Determines stiffness of servo loop	Servo loop too soft with high following errors	Servo loop too tight and/or causing oscillation
$\mathbf{K_d}$	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
$\mathbf{K_I}$	Reduces following error during long motions and at stop	Stage does not reach or stay at the desired stop position	Oscillation at lower frequency and higher amplitude
$\mathbf{V_{ff}}$	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.
$\mathbf{A_{ff}}$	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 7.2.1: Servo Parameter Functions

Section 8 – Optional Equipment

Options for the ESP6000 controller card consist of a terminal block board, three utility interface cables, and the 100-to-100 and 100-to-68 pin interface cables which connect to the UniDrive and Universal Interface Box (UIB), respectively. Each utility interface cable plugs into a connector on the controller card and has an integral faceplate for attaching to an individual PC slot. Optional equipment for the UniDrive6000 universal motor driver consists of motor driver card(s) and rack-mount ears.

Refer to paragraph 8.2 for motor driver card and rack-mount ear installation. Optional equipment is listed in **Table 8.1,** and described in the following paragraphs. Refer to Appendix G, Factory Service for ordering information.

Description	Part/Order Number
Terminal Block Board	ESP6000-INTF-BD
Analog I/O Cable	22895-01
Digital I/O Cable	22897-01
Auxiliary Cable	22896-01
Driver Interface (100-100 pin) Cable	22947-01
Motor/Driver Interface (100-68 pin) Cable	23657-01
Motor Driver Card	UNIDRIV-BD
Rack-Mount Option	UNIDRIVER-RACK

Table 8.1: Optional Equipment

8.1 ESP6000 Controller Card

CAUTION

Verify proper alignment before inserting cables into connectors. Do not force.

8.1.1 Terminal Block Board

The main (100-pin) connector of the ESP6000 controller card is intended to directly attach to the UniDrive6000. If a UniDrive is not used in a particular application, the signals from the 100-pin connector must be accessed individually and routed to appropriate amplifiers, motors, stages, etc. The terminal block board provides a means of accessing the signals from the ESP6000 controller card. The terminal block board is shown in **Figure 8.1.1.**

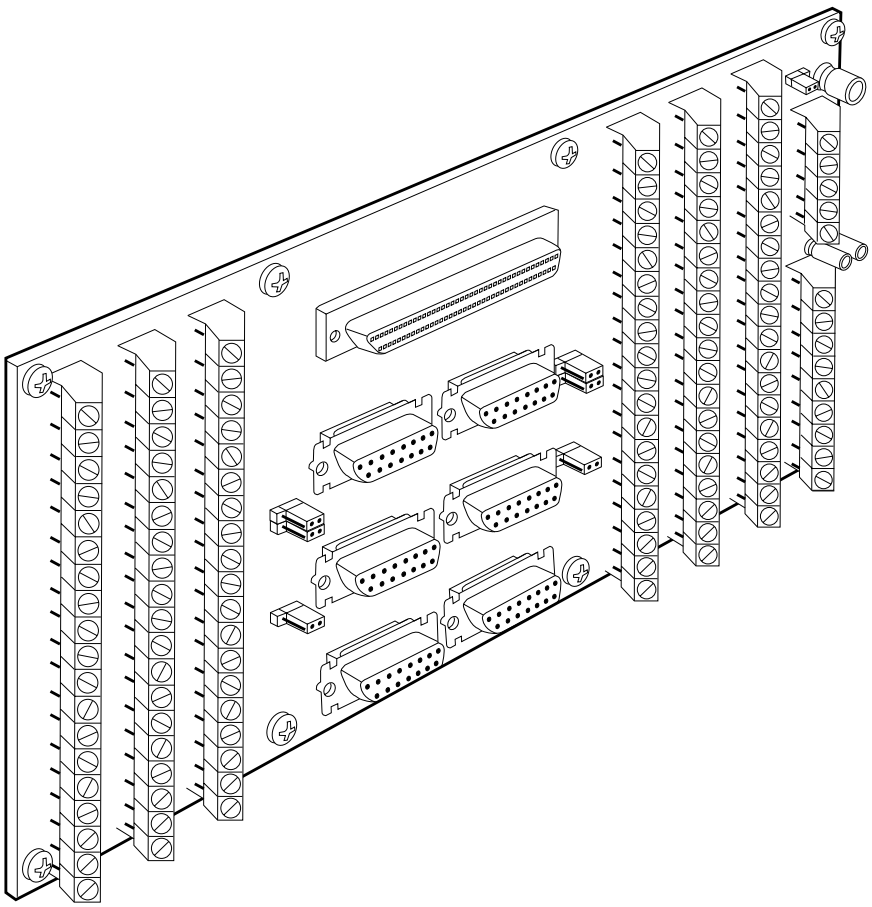


Figure 8.1.1: Terminal Block Board

Connector and lead functions are listed in **Table 8.1.1.** Refer to

Appendix C for connector orientations and jumper settings.

For systems configured with a personal computer, the terminal block board draws power from the personal computer power supply through the 100-pin cable. If an external power supply is connected to the terminal block board, the board will draw power from the external source instead of the PC.

Reference Designation	Description	To Device (Connector)	Cable Part Number(s)
J1-J6	15-pin connectors	MD4 motion device (input connector)	Supplied with device
J7	100-pin connector	ESP6000 controller card (controller connector)	22947-01
J8	Leads for optional power supply	Customer-specified power supply	Supplied with device
J9	Leads for non-motion signals	Customer-specified	Customer-provided
J11A/B-J16A/B	Leads for stages (axes 1-6)	Customer-specified	Customer-provided

Table 8.1.1: Terminal Block Board Functions

CAUTION

The ESP6000 controller card, terminal block board, and stages are sensitive to static electricity. Wear a properly grounded anti-static strap when handling equipment.

WARNING

Power-down personal computer or external power supply before connecting any equipment.

For PC-based configurations, the terminal block board can be connected any time after installation of the ESP6000 controller card, controller card driver software, and Windows motion utility software is complete. First power off the PC, then connect the terminal block board and cables/leads and device(s).

8.1.2 Analog I/O Cable

The analog I/O cable is a 26-25 ribbon cable designed for use with the ESP6000 controller card. The cable has an 8-channel analog signal capacity for interfacing devices.

Refer to the Windows Motion Utility Setup Hardware sub-menu, Analog I/O tab to configure devices for use. The cable is shown in **Figure 8.1.2,** connections are listed in **Table 8.1.2,** and connector pin-outs and descriptions are provided in Appendix C.

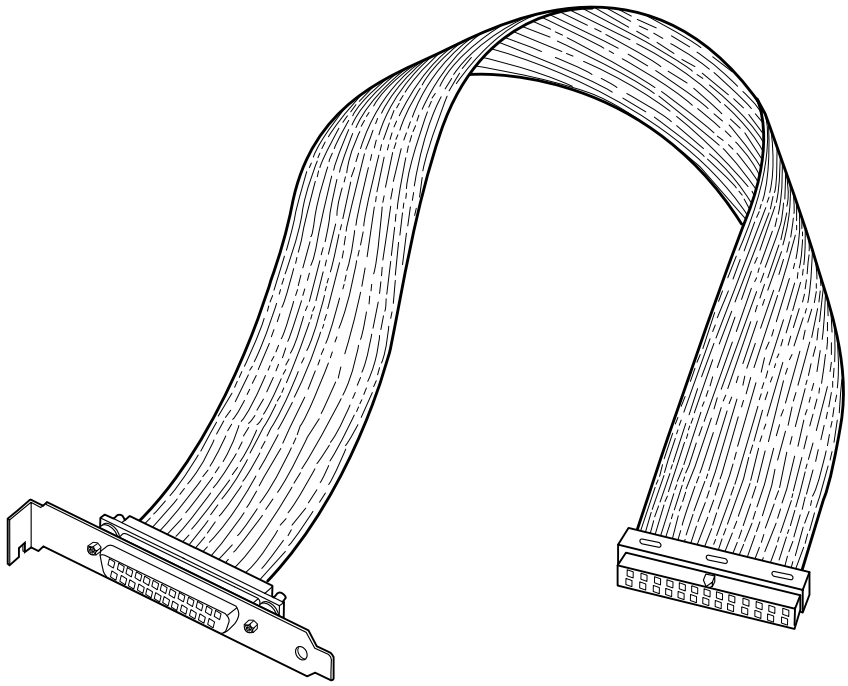


Figure 8.1.2: Analog I/O Cable

ESP6000 Controller Card Reference Designation	**To Device (D-Sub Connector)**
JP-2	Customer-specified

Table 8.1.2: Analog I/O Cable Connections

WARNING

Power-down personal computer or external power supply before connecting any equipment.

8.1.3 Digital I/O Cable

The digital I/O cable I/O cable is a 50-50 ribbon cable designed for use with the ESP6000 controller card. The cable has a 24-channel digital signal capacity for interfacing devices.

Refer to the Windows Motion Utility Setup Hardware sub-menu, Digital I/O tab to configure devices for use.

The cable is shown in **Figure 8.1.3,** connections are listed in **Table 8.1.3,** and connector pin-outs are provided in Appendix C.

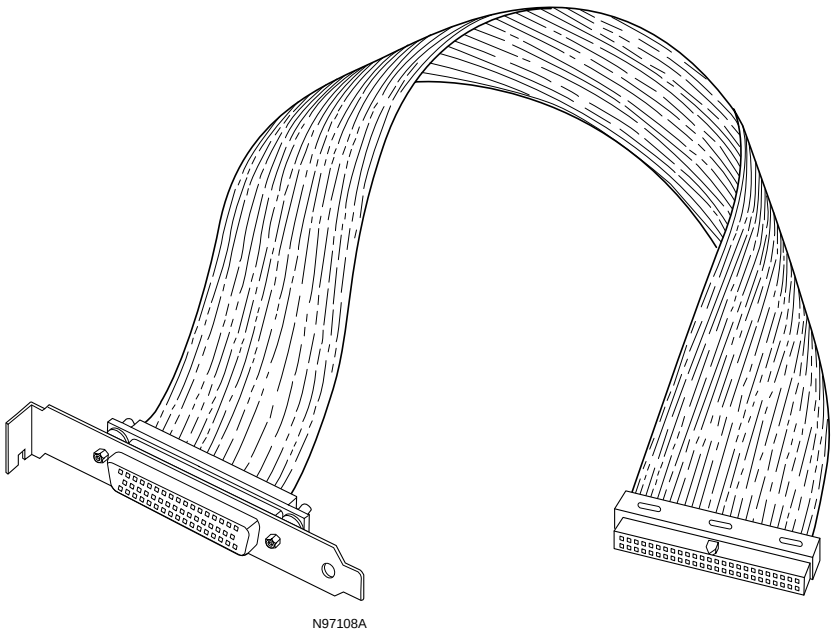


Figure 8.1.3: Digital I/O Cable

ESP6000 Controller Card Reference Designation	**To Device (D-Sub Connector)**
JP4	Customer-specified

Table 8.1.3: Digital I/O Cable Connections

WARNING

Power-down personal computer or external power supply before connecting any equipment.

8.1.4 Auxiliary I/O Cable

The auxiliary I/O cable is a 40-37 ribbon cable designed for accessing signals from the ESP6000 controller card for customer-defined applications/usage.

The cable is shown in **Figure 8.1.4,** connections are listed in **Table 8.1.4,** and connector pin-outs are provided in Appendix C.

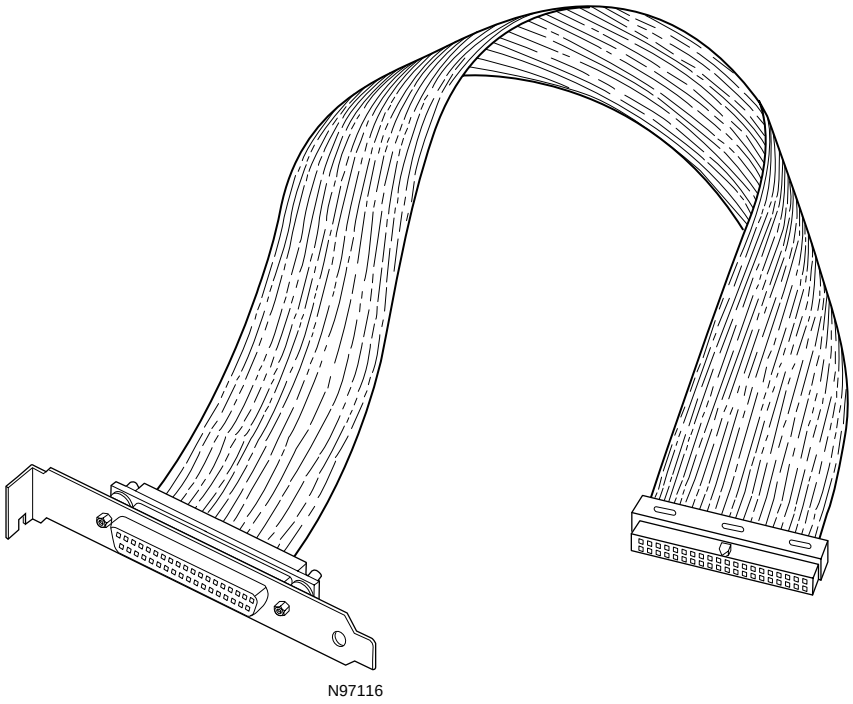


Figure 8.1.4: Auxiliary I/O Cable Connections

ESP6000 Controller Card Reference Designation	**To Device (D-Sub Connector)**
JP5	Customer-specified

Table 8.1.4: Auxiliary I/O Connections

WARNING

Power-down personal computer or external power supply before connecting any equipment.

8.1.5 Driver Interface (100-100 pin) Cable

The driver interface cable connects the ESP6000 controller card to the UniDrive6000. The cable is shown in **Figure 8.1.5** and connector pin-outs and orientation are provided in Appendix C.

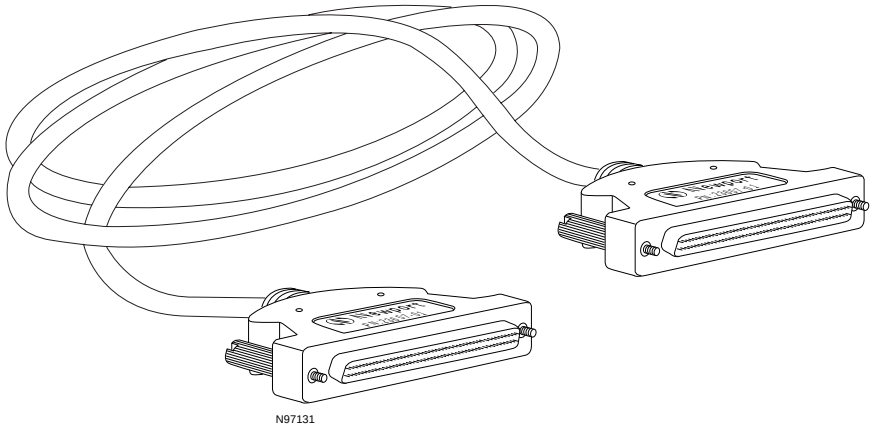


Figure 8.1.5: Driver Interface (100-100 pin) Cable

8.1.6 Motor/Driver (100-68 pin) Cable

The motor/driver cable connects the ESP6000 controller card to the UIB. The cable is shown in **Figure 8.1.6** and connector pin-outs and orientation are provided in Appendix C.

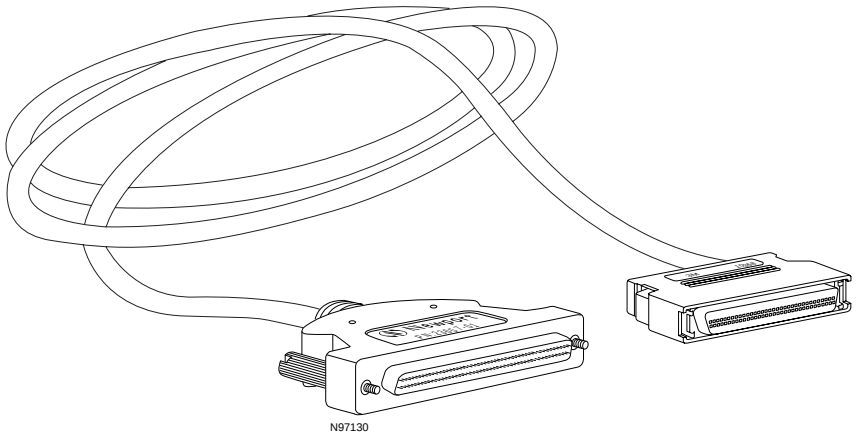


Figure 8.1.6: Motor/Driver (100-68 pin) Cable

8.2 UniDrive6000

8.2.1 Motor Driver Card

The UniDrive6000 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 UniDrive6000 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 UniDrive, 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 8.2.1**). Refer to Appendix G, Factory Service, to report discrepancies.

Insert the driver card into the black guides (top and bottom) in the slot (See **Figure 8.2.2**).

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 UniDrive AC power cord and power up the UniDrive.

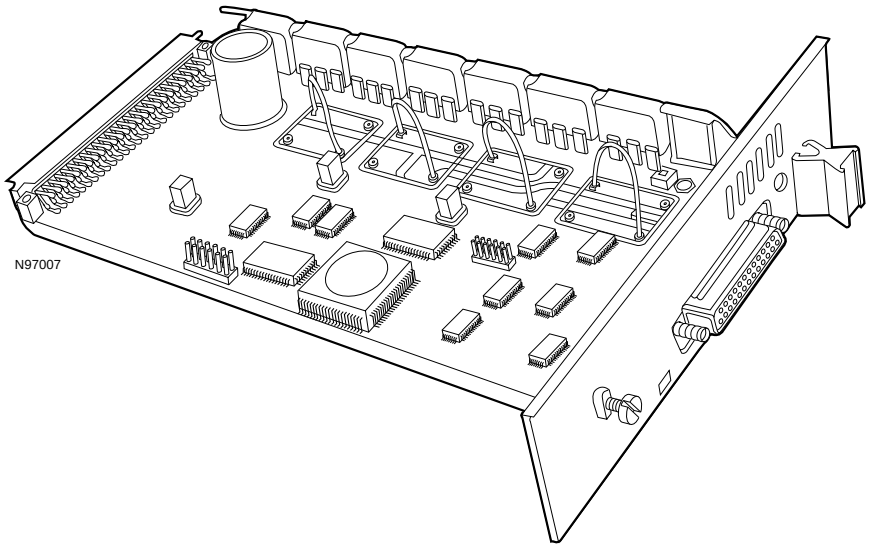


Figure 8.2.1: Driver Card

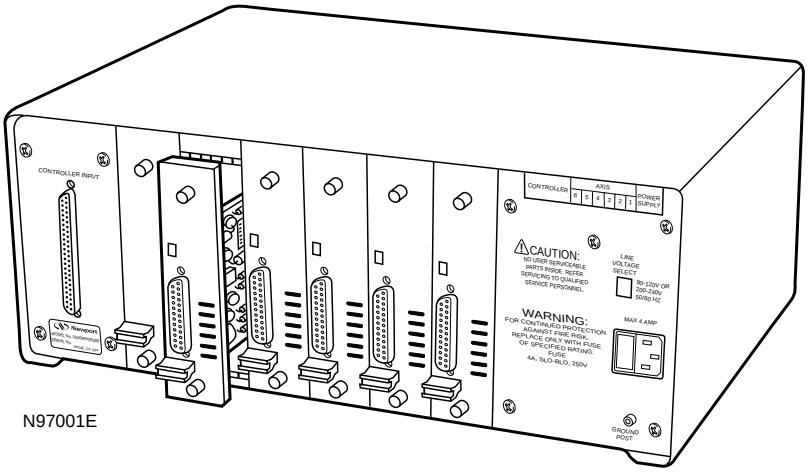


Figure 8.2.2: Driver Card Installation

8.2.2 Rack-Mount Ears

This option provides a means of mounting the UniDrive6000 in a 19-inch rack. There is no disassembly required for installation. The ears are attached by putting supplied screws through exterior screw-holes and into permanent nuts inside the enclosure. Installation is shown in **Figure 8.2.3.**

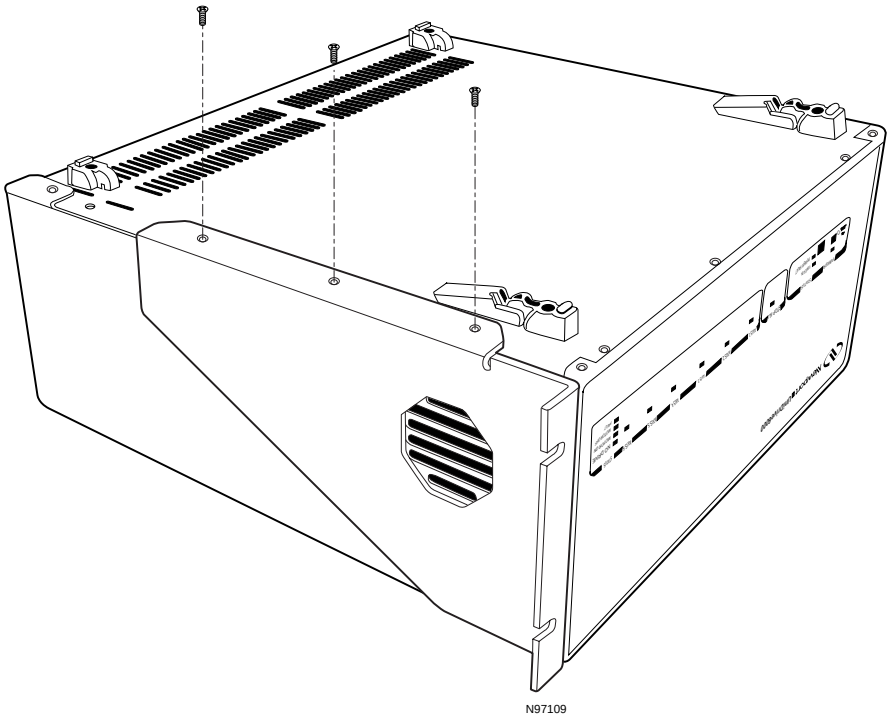


Figure 8.2.3: Rack-Mount Ear Installation

Section 9 – Advanced Capabilities

9.1 Motion Control Software Overview

9.1.1 Introduction

The ESP system motion control software provides the core functionality required for motion control applications. The motion control kernel consists of the servo system and the trajectory generator. A set of functions provide an Application Programming Interface (API) for configuring and controlling the software modules. Functional descriptions are provided in the following paragraphs.

9.1.2 Control API

The ESP system API functions are used to customize the motion control performance for a specific application. These functions can be divided into three fundamental categories: (1) initialization, (2) configuration, and (3) control. Initialization functions are used at start-up to set the initial values for all the ESP system data structures. The ESP system behavior is then customized using the configuration functions. After initialization and configuration, the axes are sent to specific locations by setting the axis control parameters. API functions are briefly described in the following paragraphs.

9.1.2.1 System Initialization

Initialization is the first required step in using the ESP system. Initialization API calls typically invoke a board-level hardware reset, initialize data structures with previously stored parameters from non-volatile flash memory (or ESP-compatible stage if new) and establish communication.

9.1.2.2 Configuration

Axis-specific parameters must be configured after initialization. These parameters consist of:

- General axis configuration: for servo or stepper motor designation
- Servo parameters: for configuring control algorithm gains. Parameters include digital-to-analog converter offset, proportional gain, derivative gain, integral gain, integrator limit, velocity feed-forward, and acceleration feed-forward.
- Trajectory parameters: trapezoidal velocity profile, S-curve velocity profile, and master-slave profile.
- Limit parameters: maximum software travel, maximum velocity, maximum acceleration, maximum jerk, and motor following error.

9.1.2.3 Axis Control

Once the axes have been configured the ESP system can move motion devices. Devices must first be enabled, then targets can be specified from trapezoidal or S-curve profiles, or jog mode can be activated.

9.1.3 Trajectory Control Process

The trajectory generation module operates independently from the servo generation module and data is passed from one to the other only in the servo interrupt service routine. Trajectory control processes are therefore not affected by servo generation processes.

Since trajectories are generated in real time, parameter changes can be processed in real time as they are received from the API. The following parameters may be changed while the axis is moving: target point, slew velocity (speed), jog velocity, acceleration, and deceleration.

For all other trajectory control parameters, it is recommended that the application programmer not submit parameter changes to the API unless the affected axes are stopped. In general, the software holds these parameter change requests as pending until the axes are stopped.

NOTE

Binary and/or ASCII Commands, described in Section 5: Programming, may be used for both configuration and trajectory control process.

9.2 Data Acquisition

9.2.1 Introduction

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 9.1, 9.3, 9.4 and 9.5 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.

9.2.2 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 9.2.2,** and described in the following paragraphs.

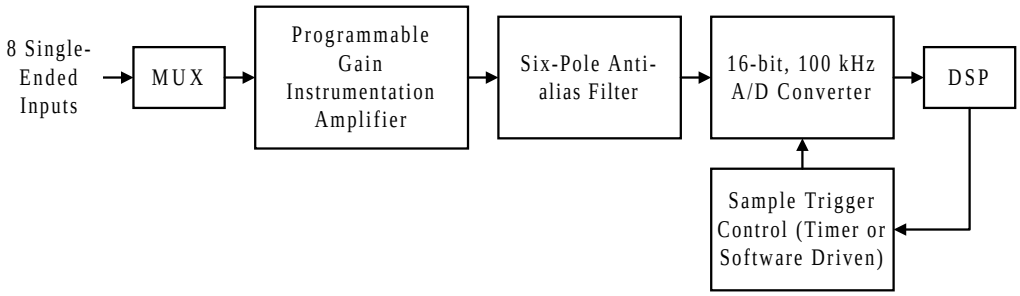


Figure 9.2.2: Analog-To-Digital Flow Diagram

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 9.2.2.1.**

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 9.2.2.1: 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 9.2.2.2: An Example of Analog Data Acquisition

9.2.3 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 9.2.3.1: An Example of Trace Variable Data Acquisition

Commands related to data acquisition – analog or trace variable – are listed in **Table 9.2.3.2** (refer to Section 5, Programming, 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 9.2.3.2: Data Acquisition Commands (ASCII)

Equivalent Binary data acquisition commands are listed in **Table 9.2.3.3** (refer to Section 5, Programming, for additional details).

Command	Description
esp_set_adc_gain*	Sets gain
esp_get_adc_gain	
esp_set_adc_range*	Sets polarity (uni-polar or bi-polar)
esp_get_adc_range	
esp_get_adc*	Selects channels, samples (one or more channels, one or more samples for each channel)
esp_get_all_adc*	Selects all eight channels at once
esp_set_daq_mode	Sets the following: (1) channel: select x channels of eight (2) mode: immediate execution, when an axis begins movement, or when an axis reaches a specific velocity (3) number: number of samples, expressed as integer value (4) position on axis designates axis to be reported on (5) timing: sets frequency/how often sampling is to occur (6) trigger on axis: sets axis top be triggered on.
esp_enble_daq	Enables data acquisition
esp_daq-done	Provides completion status for the number of samples requested (0 = DAC armed/ready but not triggered, 1 = finished, -1 = acquiring)

Table 9.2.3.3: Data Acquisition Commands (Binary)

Command	Description
esp_get_daq_data	Returns an array as specified
esp_disable_daq	Disables data acquisition

* = Immediate execution-type command, does not require setup before execution

Table 9.2.3.3: Data Acquisition Commands Binary (Continued)

9.2.4 Streaming Data Acquisition

9.2.4.1 Introduction

The ESP6000 Motion Controller has the capability of streaming analog and trace variable data to a user application at a maximum rate of 200K bytes per second. Streaming data acquisition works very similar to fixed mode data acquisition as described in the previous sections, with the following exceptions:

1. Binary interface must be used to collect the data.
2. The application program written by users must work efficiently enough to accept the amount of data at the specified rate.

9.2.4.2 Basic Description

Streaming data acquisition is setup using the command esp_set_daq_mode. The system can be set to stream data at a rate of every servo cycle (400 microseconds) down to every 10^{th} servo cycle.

Data can be streamed to one of two places:

1. To a user specified data file by using the command esp_daq_data_to_file.
2. To a ring buffer located within ESP6000.dll. The command esp_get_daq_data is used to retrieve a specified amount of data from the ring buffer.
3. Streaming can be discontinued by making a call to esp_disable_daq.

9.2.4.3 Streaming Error Conditions

1. *Ring Buffer Overflow:* This error condition occurs when ring buffer becomes full and new data is being written over data that hasn't been collected. The typical cause for this error is that the user application is collecting data from the ring buffer slower than it is being stored.

2. *Data out of Sequence:* For speed and efficiency data being streamed from the ESP6000 to the PC is being streamed to two separate buffers. The error occurs when the procedure within ESP6000.dll gets out of sync and reads data from the wrong buffer. This error indicates that the data being read is out of sequence.
3. *DSP Data Overflow:* This error signifies that the ESP6000 motion controller is writing data faster than the PC can accept it. This error is extremely rare and should only occur on a PC with limited processing power.
4. *Ring Buffer Allocation error:* This error signifies that there was not enough free memory on the PC for the ring buffer.

9.2.4.4 A Programming Example

```
#include "mm6000.h"
void main(void)
{
int status, I;
DWORD daqLength, daqStat, daqArray[1000];

// This example sets up data acquisition mode
as follows:
// 1. Constant streaming, start acquiring when
enabled
// 2. ADC data only, no axis needed
// 3. Collect data from channels 1 & 2
// 4. No position feedback
// 5. Sample at a rate of every servo cycle
// 6. Constantly streaming. No need to request
a number of samples

status =
esp_set_daq_modes(STREAM_UNCONDITIONAL,0,3,0,0,
0);
if(status == -1)
ErrorReport( );

// After the mode has been setup enable the daq
to start the data collection
status = esp_enable_daq( );
if(status == -1)
ErrorReport( );

// we are going to be streaming two analog
channels plus the servo tick at a rate of 400
µsecs
```

```
// we need to start collecting data at a fast
rate
// request 1000 elements from the ring buffer
each iteration through the loop for(;;)
{
daqLength = 1000;
esp_get_daq_data(daqArray, &daqLength,
&daqStat);

// daqLength will return with the number of
data points it was able to put
// into daqArray up to 1000
// We are getting raw data. It needs to be
converted to a value between -10 and +10
// then just print it out

for(I = 0; I <daqLength;I++)
print("%ld\n", daqArray[I] * 10 / 32767);

// Lets check to see if the streaming has
stopped. If it has, then we need to
// look at the error and get out of the loop
if(esp_daq_done( ))
{
ErrorReport( );
Break;
}
}
```

9.3 PCI Bus Overview

Newport Corporation's ESP system employs the Compact Peripheral Interconnect Component (PCI) bus for its high-performance motion control and data acquisition applications. The ESP PCI bus structure is described in the following paragraphs.

The PCI bus was designed for population with adapters requiring fast access to each other and/or system memory and that can be accessed by the host processor at speeds approaching that of the processor's full native bus speed. All read and write transfers over the PCI bus are burst transfers. The length of the burst is negotiated between the initiator and target devices and may be of any length. **Table 9.3.1** lists some of the major PCI design goals.

Features	Description
Address spaces	Full definition of three address spaces: memory, I/O, and configuration.
Auto configuration	Full bit-level specification of the configuration registers necessary to support automatic peripheral detection and configuration.
Burst read and write transfers	Burst mode for all read and write transfers. Supports 132 Mbytes-per-second peak transfer rate for both read and write transfers.
Bus master support	Full support of PCI bus initiators allows peer-to-peer PCI bus access, as well as access to main memory and expansion bus devices through PCI and expansion bus bridges. In addition, a PCI master can access a target that resides on another PCI bus lower in the bus hierarchy.
Bus speed	Revision 2.0 specification supports PCI bus speeds up to 33 MHz.
Bus width	Full definition of a 64-bit extension
Concurrent bus operation	High-end bridges support full bus concurrency with host bus, PCI bus, and the expansion bus simultaneously in use.
Expansion card definition	The specification includes a definition of PCI connectors and add-in cards.
Expansion card size	The specification defines three card sizes: long, short, and variable-height cards.
Fast access	As fast as 60ns (at a bus speed of 33 MHz when an initiator parked on the PCI bus is writing to a PCI target.
Hidden bus arbitration	Arbitration for the PCI bus can take place while another bus master is in possession of the PCI bus. This eliminates latency encountered during bus arbitration on buses other than PCI.
Low pin-count	Economical use of bus signals allows implementation of a functional PCI target with 47 pins and a functional PCI bus initiator with 49 pins.
Low power consumption	A major design goal of the PCI specification is the creation of a system design that draws as little current as possible.

Table 9.3.1: PCI Design Goals

Feature	Description
Number of PCI buses supported	The specification provides support for up to 256 PCI buses.
PCI functional devices	Although a typical PCI bus implementation supports approximately ten supported electrical loads, each PCI device package may contain up to eight separate PCI functions. The PCI bus logically supports up to 32 physical PCI device packages, for a total of 256 possible PCI functions per PCI bus.
Processor independence	Components designed for the PCI bus are PCI-specific, not processor-specific, thereby isolating device design from processor-upgrade treadmill.
Software transparency	Software drivers utilize same command set and status definition when communicating with PCI device or its expansion bus-oriented cousin.
Transaction integrity check	Parity checking on the address, command, and data.

Table 9.3.1: PCI Design Goals (Continued)

9.4 Grouping

9.4.1 Introduction

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 9.4.2. Section 9.4.3 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 9.4.4. Miscellaneous grouping commands are discussed in Section 9.4.5.

9.4.2 Defining a Group and Group Parameters

This subsection discusses the method for defining a group and all the group parameters.

9.4.2.1 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.

9.4.2.2 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 5: Programming) for correct syntax, parameter ranges, etc.

9.4.3 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.

9.4.3.1 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.

9.4.3.2 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 5: Programming) for correct syntax, parameter ranges, etc.

9.4.4 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.

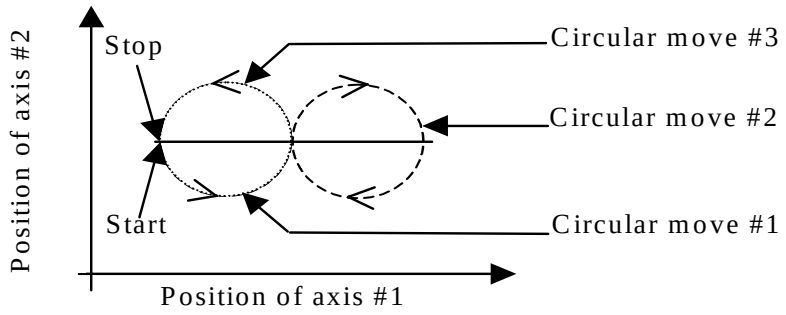


Figure 9.4.1: A Contour with Multiple Circular Moves

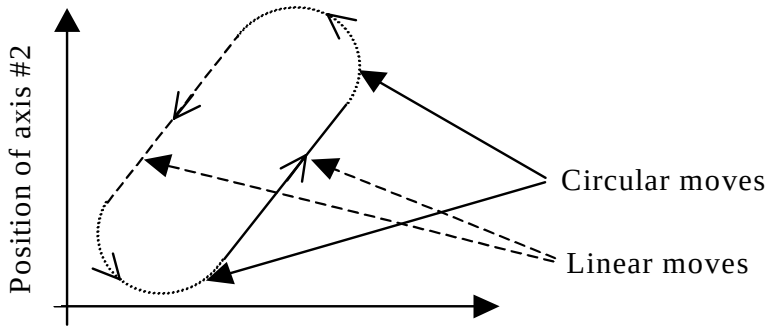


Figure 9.4.2: A Contour with Multiple Linear and Circular Moves

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 5: Programming) for correct syntax, parameter ranges, etc.

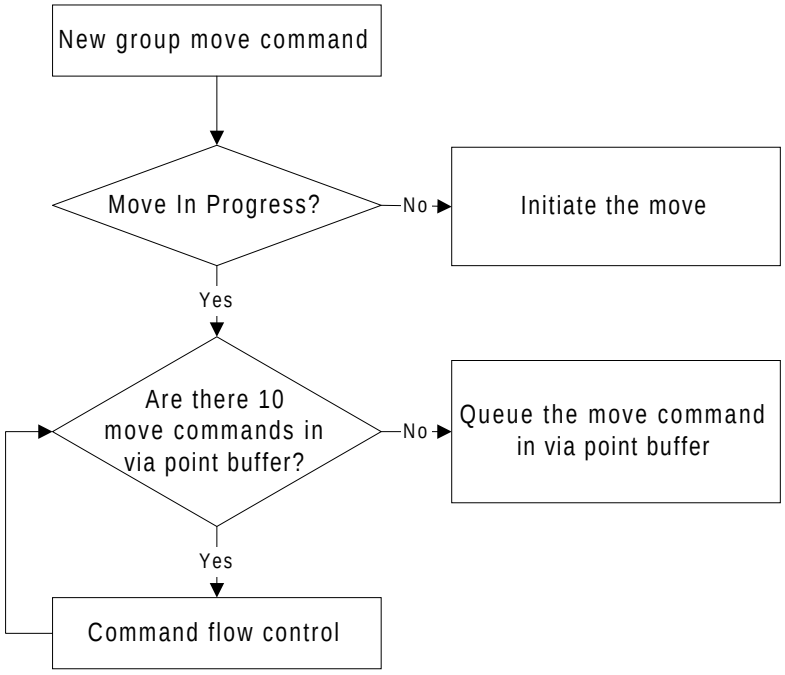


Figure 9.4.3: Block Diagram of Via Point Data Handling by Command Processor

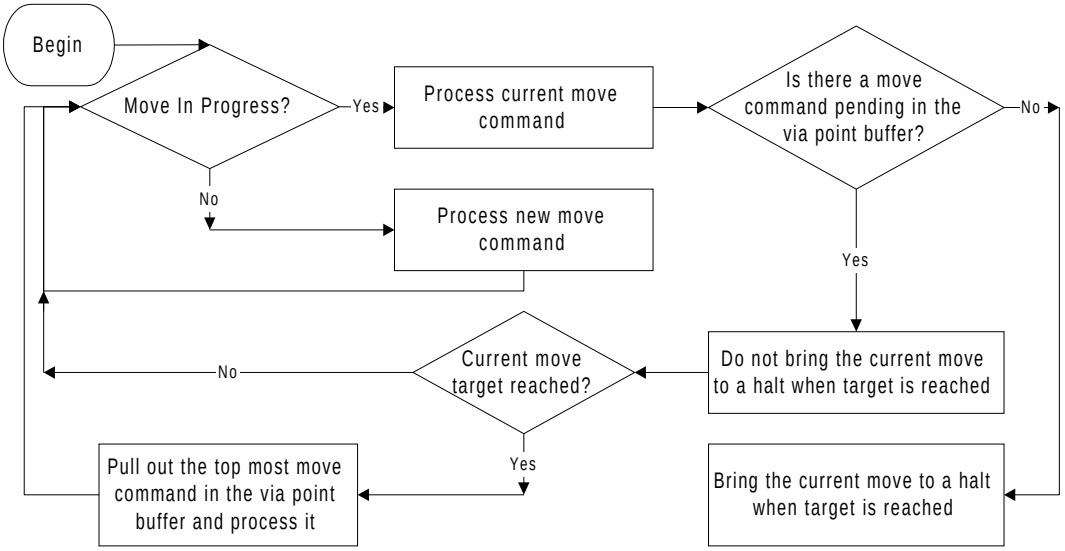


Figure 9.4.4: Block Diagram of Via Point Data Handling by Trajectory Generator

9.4.5 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.

9.5 Slaving a Stage to Trackball, Joystick, or a Different Stage

9.5.1 Introduction

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.

9.5.2 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 9.5.2: Slave to A Different Stage

9.5.3 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 9.5.3: Slave to a Trackball

9.5.4 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 9.5.4: Slave to a Joystick

Refer to the description of the ASCII commands in Section 5: Programming, for additional description, correct syntax, parameter ranges, etc.

9.6 Closed Loop Stepper Motor Positioning

9.6.1 Introduction

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.

9.6.2 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.

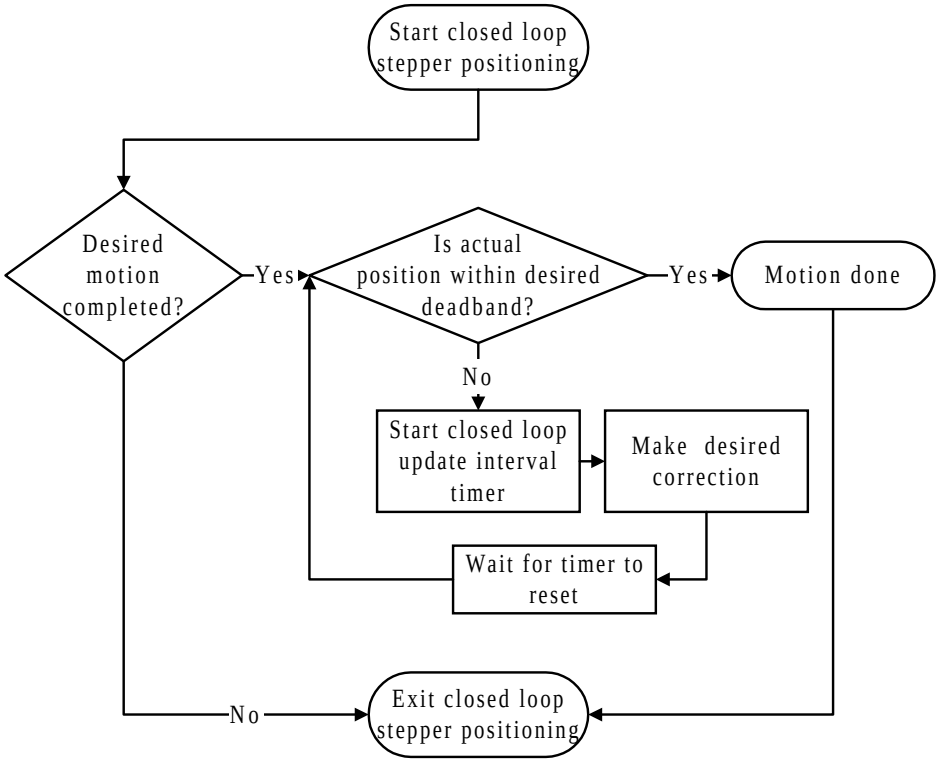


Figure 9.6.2: Block Diagram of Closed Loop Stepper Motor Positioning

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).

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 9.6.2.1**) 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 9.6.2.1: An Example of Closed Loop Stepper Motor Positioning Setup

Commands related to closed loop stepper positioning are listed in **Table 9.6.2.2** (refer to Section 5, Programming, for additional details):

Command	Description
ZB	Set feedback configuration.
DB	Specify deadband value.
CL	Specify closed loop update interval.

Table 9.6.2.2: Closed Loop Stepper Positioning Commands

9.7 Synchronize Motion to External and Internal Events

9.7.1 Introduction

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.

9.7.2 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.

9.7.3 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.

9.7.4 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 9.7.4: Commands to Synchronize Motion to External Events

9.8 Position Compare Output Triggering

9.8.1 Introduction

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.

ESP6000 and ESP7000 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.

9.8.2 Feature Setup

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.

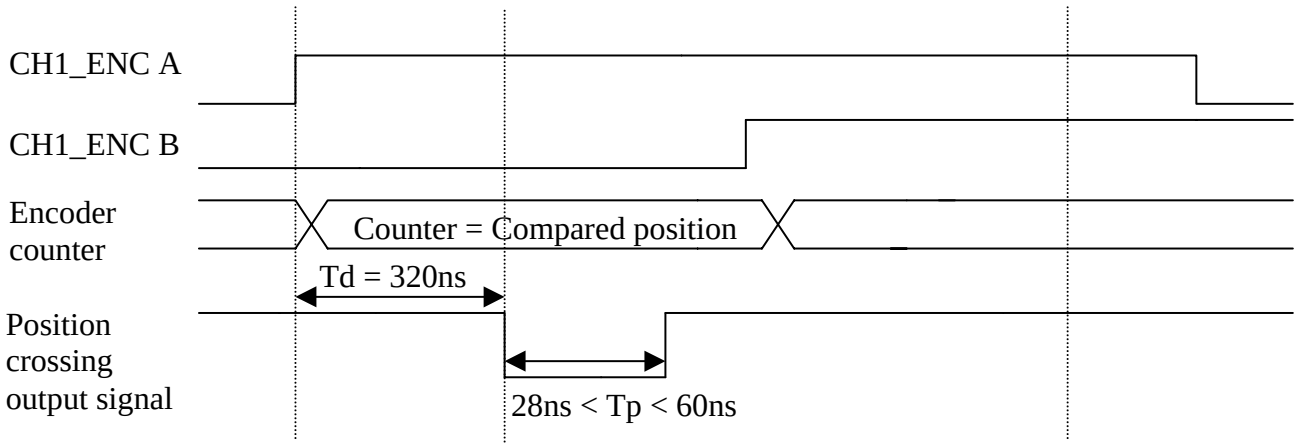
However, if this ASCII command is issued when the desired axis is in motion, the accuracy of the position crossing capture in absolute mode (nn1 = 1) depends upon the speed at which the axis was moving when the command was processed by the DSP.

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 9.8.2: 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  // 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
```

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").

9.8.3 Hardware Required

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 9.8.3: Commands to Synchronize External Events to Axis Position

9.9 Position Capture Input Triggering

9.9.1 Introduction

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.

ESP6000 and ESP7000 motion 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.

9.9.2 Feature Setup

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 (3_10 = (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.

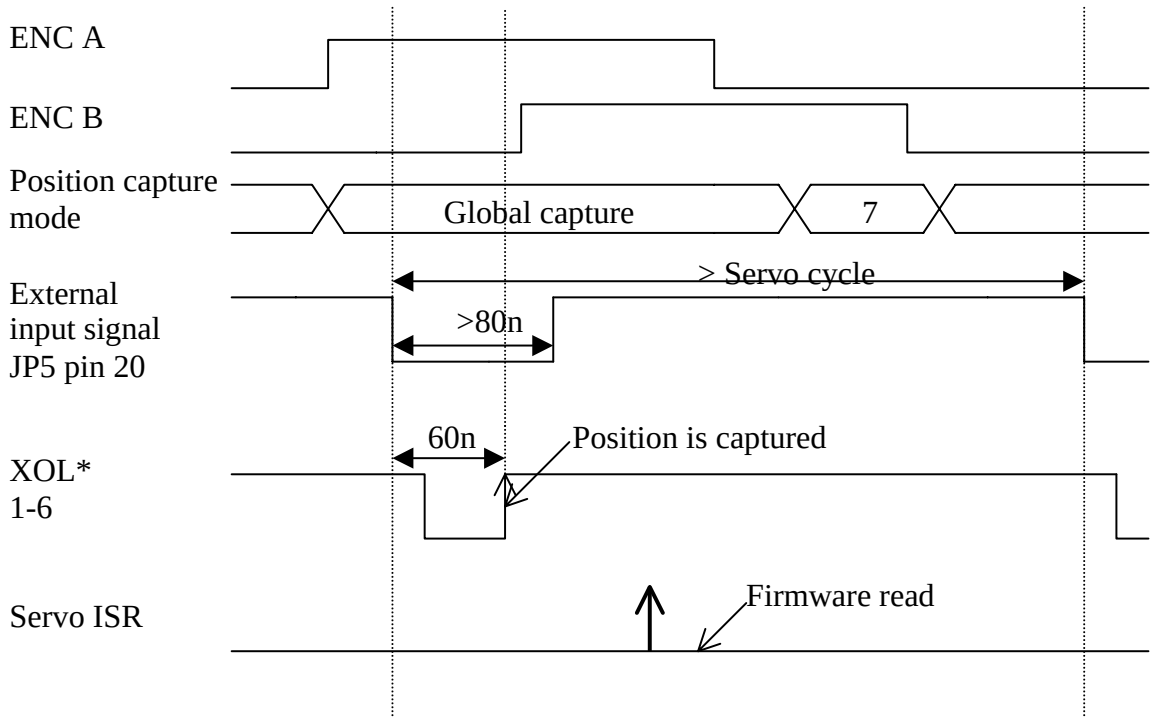


Figure 9.9.2: Timing Diagram of Position Capture Synchronized to External Input Trigger

9.9.3 Hardware Required

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 the table below (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 9.9.3: Commands to Synchronize Position Capture to External Inputs

Appendix A – Error Messages

The ESP6000 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. During moves and while, idle system inputs are monitored and any change is reported to the user via the output buffer. To read the contents of the output buffer, send the command **TB** (tell buffer).

For more compact error messages, use the **TE** command. The ESP6000 controller response to this command is a one byte; binary coded error number, e.g., 33.

Users can also access these messages through API calls, such as:

"esp_get_error_string( )" or "esp_get_error_num( )".

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, in **Table A-1,** of all possible ESP6000 controller error messages that are ***not axis specific:***

Error Number	Message Text	Message Text Description	Possible Cause/Solution
0	No Error Detected	No errors exist in the output buffer.	----
1	PCI Communication Timeout	Windows DLL was not able to successfully communicate with the ESP6000 controller card.	ESP may not be installed at all or properly seated.
2	Reserved for future use	----	----
3	Reserved for future use	----	----

Error Numbering Legend:
1-99 = Non-axis specific
Y00-Y99 = Axis-specific, Y = axis number
1000 or greater = Originated by DLL

Table A.1: Error Messages

Error Number	Message Text	Message Text Description	Possible Cause/Solution
4	Emergency Stop Activated	E-Stop input signal was detected true and event was processed.	Stop All button on UniDrive6000 pressed or E-Stop input on auxiliary I/O pulled low.
5	Insufficient Memory	ESP6000 did not have enough available memory to process/execute command.	Too many commands in queue, or command(s) being processed are presently using all available memory.
6	Command Does Not Exist	ESP6000 Windows DLL attempted to call a command that does not exist in ESP6000 firmware.	DLL and firmware not compatible.
7	Parameter Out Of Range	Command parameter(s) not within allowable range.	Refer to the Command arguments (Section 5) for appropriate parameter range.
8	100-pin Cable Interlock Error	100-pin cable interlock error detected.	100-pin cable not properly connected at ESP6000 and/or UniDrive6000 end.
9	Axis Number Not Available (Out Of Range)	Axis number specified in command is out of range or not available.	Axis number greater than 6 will cause this error.
10	ADC Gain Is Out Of Range	Specified analog-to-digital converter 'gain' setting is not within allowable range.	Valid ADC gain values are 0, 1, 2, and 4.
11	ADC Range Is Out Of Range	Specified analog-to-digital converter 'range' setting is not within allowable range.	Valid ADC range values are 0 (UNIPOLAR) and 1 (BIPOLAR).
12	ESP Critical Settings Are Protected	ESP compatible stage and UniDrive critical settings are protected.	Modification to critical ESP system settings (e.g., motor type) was not allowed.
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.

Error Numbering Legend:
1-99 = Non-axis specific
Y00-Y99 = Axis-specific, Y = axis number
1000 or greater = Originated by DLL

Table A.1: Error Messages (Continued)

Error Number	Message Text	Message Text Description	Possible Cause/Solution
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	1. New acquisition was attempted before current acquisition complete. 2. Current acquisition still in progress.	1. An attempt was made to execute an ADC-related command while a DAQ-related command was in progress. 2. An attempt was made to execute a DAQ-related command while a DAQ-related command was in progress.
21	DAQ Setup Error	Data acquisition setup failed.	Possible causes: A data acquisition setup command was issued while in the process of data acquisition. Or, an invalid parameter was specified in the setup command.
22	DAQ Not Enabled	Data acquisition not setup and enabled.	The application must setup for and enable acquisition before attempting to acquire data.

Error Numbering Legend:
1-99 = Non-axis specific
Y00-Y99 = Axis-specific, Y = axis number
1000 or greater = Originated by DLL

Table A.1: Error Messages (Continued)

Error Number	Message Text	Message Text Description	Possible Cause/Solution
23	Servo Cycle Tick Failure	Servo cycle interrupt failure was detected.	This error message appears if a servo cycle interrupt fails to occur within the required time period. Motor control is not possible without the servo cycle interrupt. *Possible causes include:* 1. Jumper JP "SRVO" not installed. 2. Jumper JP14 not installed. 3. Hardware failure.
24	Reserved for future use	----	----
25	Download In Progress	Firmware download in progress.	----
26	Stored Program Not Started	A 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 Move Not Allowed During Homing	Cannot make a coordinated move when one of the members of the group is being "homed".	----

Error Numbering Legend:
1-99 = Non-Axis Specific
Y00-Y99 = Axis-specific, Y = axis number
1000 or greater = Originated by DLL

Table A.1: Error Messages (Continued)

Error Number	Message Text	Message Text Description	Possible Cause/Solution
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 axis.
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.	----

Error Numbering Legend:
1-99 = Non-axis specific
Y00-Y99 = Axis-specific, Y = axis number
1000 or greater = Originated by DLL

Table A.1: Error Messages (Continued)

Below is a list of all possible error messages ***that are axis specific.*** Here, "x" represents the axis number.

Error Number	Message Text	Message Text Description	Possible Cause/Solution
y00	Axis-**y** Motor Type Not Defined	Axis motor type (i.e., stepper or DC servo) not presently defined.	Axis 'motor type' must be defined before any motion commands can be executed. The default motor type is UNDEFINED for unconfigured axes. To achieve motion, axes must be defined as either DCSERVO or STEPPER. Axes with ESP-compatible stages are automatically configured.
y01	Axis-**y** Parameter Out Of Range	Command parameter(s) not within allowable range.	Refer to the Command arguments (Section 5) for appropriate parameter range.
y02	Axis-**y** Amplifier Fault Detected	Amplifier fault input signal is asserted.	This may be caused by a true failure in the UniDrive itself. Most often this error is generated because the UniDrive was turned OFF while the controller was in a motor ON state.
y03	Axis-**y** Following Error Threshold Exceeded	Servo axis is not accurately following desired position or trajectory and has exceeded maximum threshold setting.	Possible causes for this problem are: 1. PID servo parameters not optimally tuned. 2. Following error threshold parameter itself is too small and not realistic. Trajectory parameters (e.g., velocity) too high for motor/load setup.
y04	Axis-**y** Positive Hardware Limit Detected	Positive travel stage hardware limit encountered	This error is caused by traversing further than is mechanically possible (or allowed) in the positive direction.
y05	Axis-**y** Negative Hardware Limit Detected	Negative travel stage hardware limit encountered.	This error is caused by traversing further than is mechanically possible (or allowed) in the negative direction.

Error Numbering Legend:
1-99 = Non-axis specific
Y00-Y99 = Axis-specific, Y = axis number
1000 or greater = Originated by DLL

Table A.1: Error Messages (Continued)

Error Number	Message Text	Message Text Description	Possible Cause/Solution
y06	Axis-**y** Positive Software Limit Detected	Positive travel stage software limit encountered.	This error is caused by traversing further than is allowed (via software setting) in the positive direction.
y07	Axis-**y** Negative Software Limit Detected	Negative travel stage software limit encountered.	This error is caused by traversing further than is allowed (via software setting) in the positive direction.
y08	Axis-**y** Motor / Stage Not Connected	Motor/stage are not connected to controller.	If both travel negative and positive direction limits become active simultaneously then it is assumed that the motor/stage is *not* connected to the controller. This is most typically caused by a disconnected motor cable.
y09	Axis-**y** Feedback Signal Fault Detected	Encoder feedback signal error detected.	Possible causes for this error are: 1. Disconnecting and reconnecting motor/stages. 2. Glitches on encoder signals.
y10	Axis-**y** Maximum Velocity Exceeded	Velocity (speed) setting exceeds maximum allowed for that axis.	Each axis has a maximum allowed velocity (speed) setting. If new velocity commands received exceed this value then an error will appear.
y11	Axis-**y** Maximum Acceleration Exceeded	Acceleration or deceleration setting exceeds maximum allowed for that axis.	Each axis has a maximum allowed acceleration/deceleration setting. If new acceleration or deceleration commands received exceed this value then an error will appear.
y12	Reserved for future use	----	----

Error Numbering Legend:
1-99 = Non-axis specific
Y00-Y99 = Axis-specific, Y = axis number
1000 or greater = Originated by DLL

Table A.1: Error Messages (Continued)

Error Number	Message Text	Message Text Description	Possible Cause/Solution
y13	Axis-**y** Motor Not Enabled	Axis motor control and power are not enabled.	When the controller receives a 'motor enable' command it then sets the AMPLIFIER ENABLE output signal to the active true state thereby enabling UniDrive power to the motor/stage. If motion commands are received prior to the motor being enabled then this error will persist and no motion will occur. NOTE: Motors are disabled by a system reset.
y14	Axis-**y** Parameter Modification Denied	Change to desired parameter or mode was not permitted.	Specified axis must be stopped before changing desired parameter)e.g., trajectory mode).
y15	Axis-**y** Maximum Jerk Exceeded	Jerk setting exceeds maximum allowed for that axis.	Each axis has a minimum allowed jerk (derivative of acceleration) setting. If new jerk commands received exceed this value then an error will appear.
y16	Axis-**y** Maximum DAC Offset Exceeded	Commanded servo control DAC output offset adjustment setting exceeds maximum allowable setting.	The ESP6000 can software compensate for DAC output offsets. Offsets are typically tens of milli-volts. The maximum offset adjustment permitted is ±1.0 volt. Off offset command exceeds ±1.0 volt then this error will occur.
y17	Axis-**y** Critical Settings Are Protected.	ESP-compatible stage and UniDrive critical settings are protected.	Modification to critical ESP system settings (e.g., motor type) were not allowed.
y18	Axis-**y** ESP Stage Device Error	Communication with serial EPROM failed.	Possible causes: cable connection, motor enabled or component enabled or component failure. Check cable connection and ensure motor not enabled. Call technical support.
y19	Axis-**y** ESP Stage Data Invalid	Some key parameter data in the SmartStage is invalid.	Possible invalid parameters; motor-type, stepper-related data, units of measure or encoder type.

Error Numbering Legend:
1-99 = Non-axis specific
Y00-Y99 = Axis-specific, Y = axis number
1000 or greater = Originated by DLL

Table A.1: Error Messages (Continued)

Error Number	Message Text	Message Text Description	Possible Cause/Solution
y20	Axis-**y** ESP Homing Aborted	Failed to complete the homing function.	All axes must be stopped before homing can begin. Or, some error condition exists that is inhibiting motion. Clear error condition and try again.
y21	Motor Current Not Defined	Maximum for the motor is not specified.	Refer to the described of **QI** command to query or set the maximum motor current for an axis.
y22	UniDrive Communications Error	There was no communication between motion controller and the UniDriver.	----
y23	UniDrive Not Detected	UniDrive could not be detected by the motion controller.	----
y24	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.
y25	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.
y26	Parameter Change Not Allowed During Motion	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.
y27	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.
y28	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.
y29	Digital I/O Interlock Detected	A DIO interlock was asserted.	----
y30	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.
y31	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.

Error Numbering Legend:
1-99 = Non-axis specific
Y00-Y99 = Axis-specific, Y = axis number
1000 or greater = Originated by DLL

Table A.1: Error Messages (Continued)

Error Number	Message Text	Message Text Description	Possible Cause/Solution
y32	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.
1000	ESP Communication Not Allowed	ESP Initialization API must be called prior to communicating with board.	ESP controller card was not initialized. Possible cause includes initialization API functions not called.
1001	Device Driver Failed To Lock Shared Memory	ESP Windows device driver could not secure PC RAM memory needed for communication.	Not enough Windows memory was available for the ESP initialization. Possible causes include: 1. Device driver not installed. 2. Insufficient PC memory.

Error Numbering Legend:
1-99 = Non-axis specific
Y00-Y99 = Axis-specific, Y = axis number
1000 or greater = Originated by DLL

Table A.1: Error Messages (Continued)

Appendix B – Trouble-Shooting and Maintenance

There are no user-serviceable parts or user adjustments to be made to the ESP6000 controller card or UniDrive6000. Fuse replacement procedures for the UniDrive are provided in this section.

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 the user 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

Most of the time, a blown fuse or an error reported by the ESP6000 controller card is the result of a more serious problem. Fixing the problem should include not only correcting the effect (blown fuse, limit switch, etc.) but also the cause of the failure. Analyze the problem carefully to avoid repetition. A list of the most common problems and their corrective actions is provided in **Table B.1.1.** 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
UniDrive Power LED does not illuminate green when power on button is pressed.	No electrical power.	Use a tester or other device (lamp, etc.) to verify that power is present in the outlet. Contact an electrician if not.
UniDrive Power LED does not illuminate green when power on button is pressed.	Power cord not plugged in.	Connect UniDrive power cord to the appropriate outlet. Refer to the System Setup section for procedures.
UniDrive Power LED does not illuminate green when power on button is pressed.	Rear power line panel fuse(s) blown.	Replace fuse(s) as described in this section. Refer to Appendix G, Factory Service if fuse blows again.
Error message or physically present stage is declared unconnected.	Bad connection.	Power down the UniDrive and PC and verify the stage cable connection.
Error message or other indication: physically present stage is declared unconnected.	Bad component.	Power down the UniDrive and PC and replace cable. Refer to Appendix G, Factory Service, for cable replacement or stage service.
UniDrive Power LED green does not illuminate.	Excessive following error.	Verify that all setup parameters corresponding to the installed stage.

Table B.1.1: Trouble-Shooting Guide

Problem	Cause	Corrective Action
Stage does not move.	Incorrect connection	Verify that the stage is connected to the correct driver card.
Stage does not move	Incorrect parameters	Verify that relevant parameters are set properly.
System performance below expectations	Incorrect parameters	Verify that relevant parameters are set properly.
System will not respond to move command.	Software travel limit.	The software limit in the specified direction was reached. Verify that limits are set correctly and do not try to exceed them.
System will not respond to move command	Incorrect parameters	Verify that all relevant parameters are set properly.
Home search not completed	Faulty origin or index signals	Carefully observe and record the motion sequence by watching manual knob rotation, if available. Refer to Appendix G, Factory Service for assistance.

Table B.1.1: Trouble-Shooting Guide (Continued)

NOTE

Many other types of problems are detected by the ESP6000 controller card and reported via error messages. Refer to Appendix A for a complete list and descriptions.

B.2 Fuse Replacement

B.2.1 Replacing Fuses On The UniDrive Rear Power Line Panel

WARNING

Power-down equipment and unplug AC power cord before replacing fuses.

At the rear of the UniDrive6000, depress the fuseholder tabs with a small, thin-bladed screwdriver (See **Figure B.2.1**) and ease the fuseholder out of the AC plug receptacle.

Remove and inspect the fuses. Replace as needed with 4A, SLO-BLO, 250V fuse (Schurter part number 21406-24).

Re-insert into the AC plug receptacle by pushing in the fuseholder until the tabs snap in place.

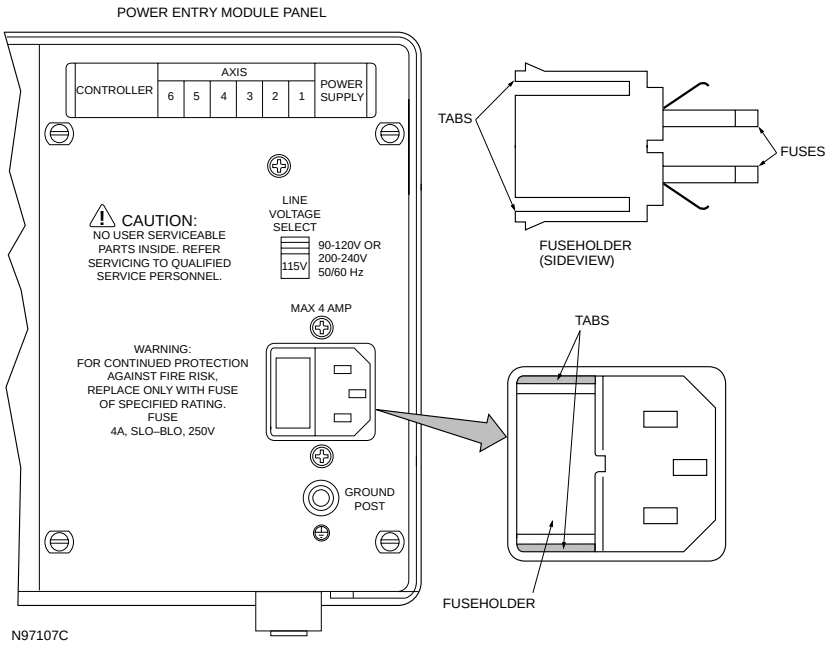


Figure B.2.1: Rear Power Line Panel Fuse Replacement

Re-connect and power-up the system to verify that the problem has been corrected.

B.2.2 Replacing Fuses On The UniDrive Motor Power Supply Board

A defective motor power supply fuse is indicated by the UniDrive Power LED illuminating red.

WARNING

Power-down equipment and unplug AC power cord before replacing fuses.

CAUTION

The UniDrive6000 rear power line panel and motor power supply board are sensitive to static electricity. Wear a properly grounded anti-static strap when handling equipment.

At the front of the UniDrive, remove the black plastic cap from the power switch.

At the rear of the UniDrive, loosen the thumbscrews on the power line panel, and slide the assembly forward on its card guides and out (See **Figure B.2.2**).

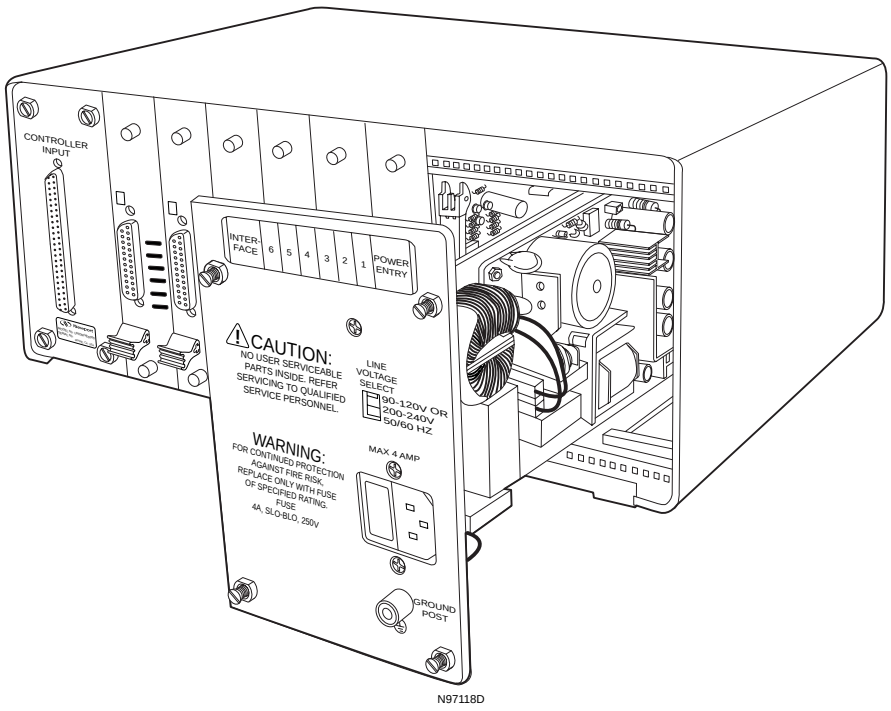


Figure B.2.2: Rear Power Line Board Removal

Slide the power supply board forward on its card guides and out (See **Figure B.2.3**).

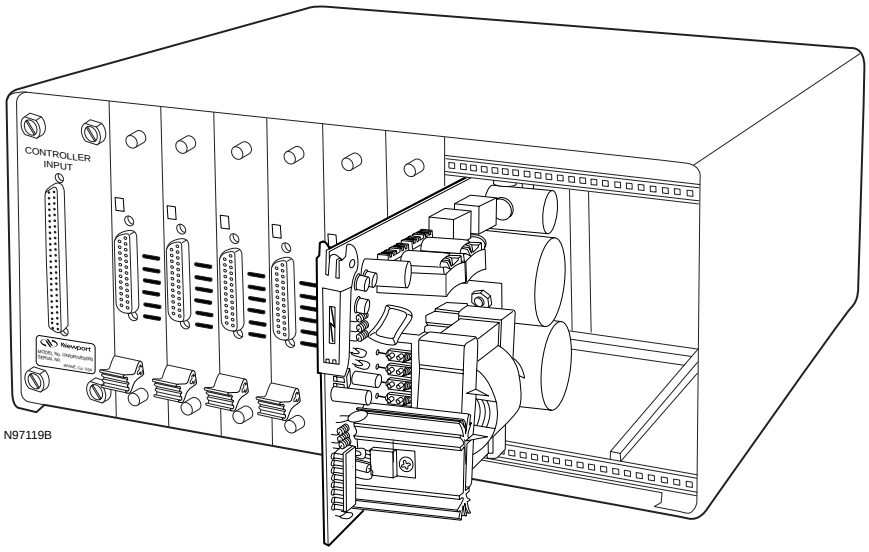


Figure B.2.3: Power Supply Board Removal

Place the power supply board on a flat surface, and rotate the notch in the fuseholder cover until it releases (See **Figure B.2.4**).

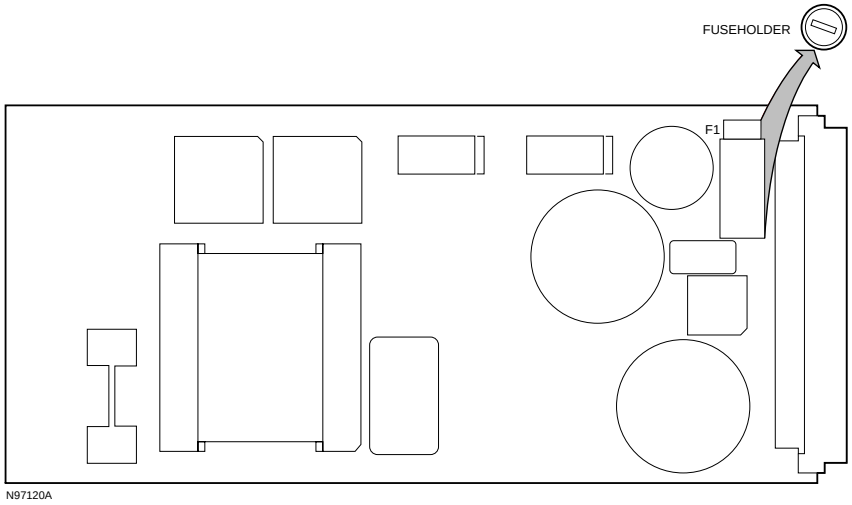


Figure B.2.4: Power Supply Board Fuse Replacement

Remove and inspect the fuse. Replace as needed with 3.15A, 250V fuse (Schurter part number 70353).

Re-install the power supply board.

Re-install the rear power line board, guiding the white plastic plunger back into the power switch cut-out at the front of the UniDrive.

Re-attach the black plastic cap to the end of the plunger.

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 UniDrive6000 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 ESP6000 Controller Card

The controller card utilizes four primary connectors. The card interfaces with the UniDrive6000 via a 100-to-100 pin cable or with the Universal Interface Box via a 100-to-68 pin cable. The connector functions are defined in the following paragraphs. Cabling for the connectors is described in Section 8, Optional Equipment, and connector orientations are shown in **Figures C.1.1, C.1.2,** and **C.1.3.**

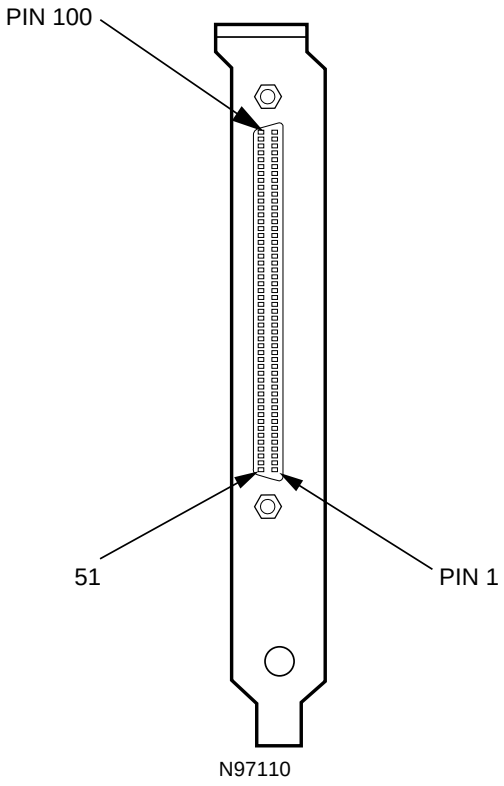


Figure C.1.1: Main I/O (100-Pin) Connector Orientation

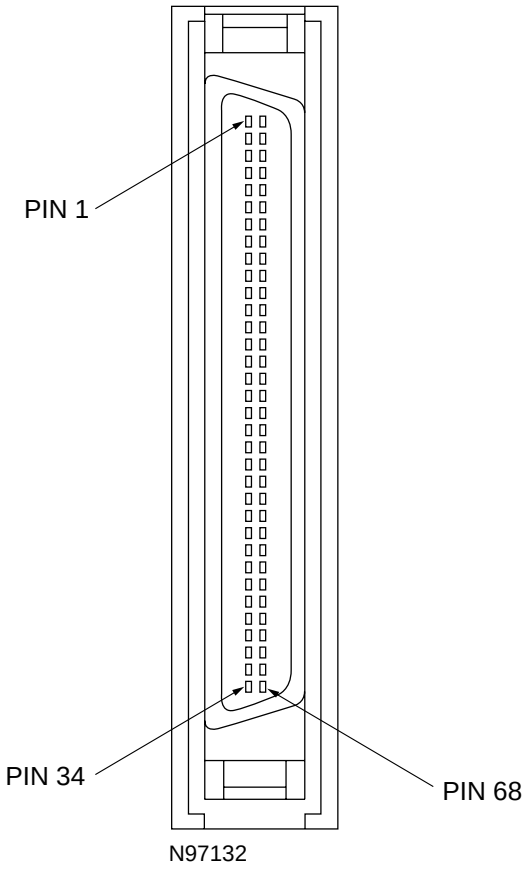


Figure C.1.2: One-Hundred to Sixty-Eight Pin Cable Connector Orientation

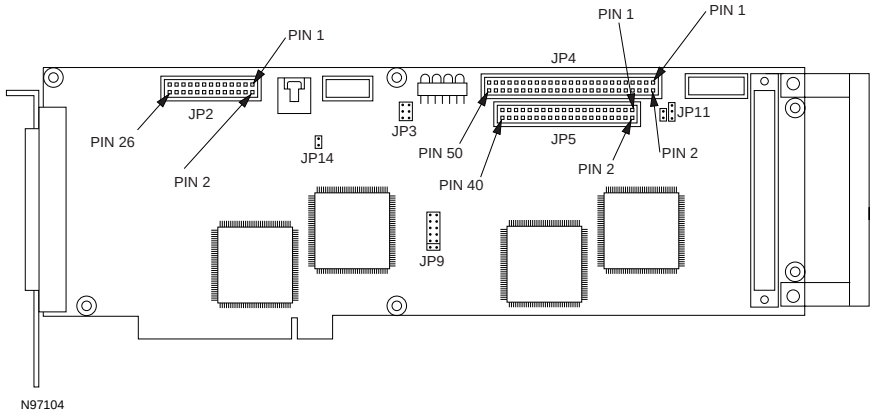


Figure C.1.3: JP2/JP4/JP5 Connector Orientation

C.1.1 Main I/O (100-Pin) Connector

This connector interfaces the ESP6000 controller card to the UniDrive6000 universal motor driver. Connector pin-outs are listed in **Table C.1.1,** and functionally described in the following paragraphs.

Pin	Function	Pin	Function
1	Cable Interlock Input	31	Travel Limit(-) Input, Axis-2
2	+5V, 250 mA (maximum)	32	Step/Direction Output, Axis-2
3	+5V, 250 mA (maximum)	33	Index(-) Input, Axis-2
4	Amplifier Fault Input, Axis-6	34	Encoder B(-) Input, Axis-2
5	Travel Limit(-) Input, Axis-6	35	Encoder A(-) Input, Axis-2
6	Step/Direction Output, Axis-6	36	Amplifier Enable Output, Axis-2
7	Index(-) Input, Axis-6	37	Amplifier Fault Input, Axis-1
8	Encoder B(-) Input, Axis-6	38	Travel Limit(-) Input, Axis-1
9	Encoder A(-) Input, Axis-6	39	Step/Direction Output, Axis-1
10	Amplifier Enable Output, Axis-6	40	Index(-) Input, Axis-1
11	Amplifier Fault Input, Axis-5	41	Encoder B(-) Input, Axis-1
12	Travel Limit(-) Input, Axis-5	42	Encoder A(-) Input, Axis-1
13	Step/Direction Output, Axis-5	43	Digital Ground
14	Index(-) Input, Axis-5	44	Digital Ground
15	Encoder B(-) Input, Axis-5	45	Digital Ground
16	Encoder A(-) Input, Axis-5	46	Digital Ground
17	Amplifier Fault Input, Axis-4	47	Analog Ground
18	Travel Limit(-) Input, Axis-4	48	Analog Ground
19	Step/Direction Output, Axis-4	49	-12V, 250mA (maximum)
20	Index(-) Input, Axis-4	50	+12V, 250mA (maximum)
21	Encoder B(-) Input, Axis-4	51	E-Stop Input
22	Encoder A(-) Input, Axis-4	52	Reserved
23	Amplifier Enable Output, Axis-4	53	Reserved
24	Amplifier Fault Input, Axis-3	54	Home Input, Axis-6
25	Travel Limit(-) Input, Axis-3	55	Travel Limit(+) Input, Axis-6
26	Step/Direction Output, Axis-3	56	Step Output, Axis-6
27	Index(-) Input, Axis-3	57	Index(+) Input, Axis-6
28	Encoder B(-) Input, Axis-3	58	Encoder B(+) Input. Axis-6
29	Encoder A(-) Input, Axis-3	59	Encoder A(+) Input, Axis-6
30	Amplifier Fault Input, Axis-2	60	Amplifier Enable Output, Axis-5

Table C.1.1: Main I/O Connector Pin-Outs

Pin	Function	Pin	Function
61	Home Input, Axis-5	81	Travel Limit(+) Input, Axis-2
62	Travel Limit(+) Input, Axis-5	82	Step Output, Axis-2
63	Step Output, Axis-5	83	Index(+) Input, Axis-2
64	Index(+) Input, Axis-5	84	Encoder B(+) Input, Axis-2
65	Encoder B(+) Input, Axis-5	85	Encoder A(+) Input, Axis-2
66	Encoder A(+) Input, Axis-5	86	Amplifier Enable Output 1
67	Home Input, Axis-4	87	Home Input, Axis-1
68	Travel Limit(+) Input, Axis-4	88	Travel Limit(+) Input, Axis-1
69	Step Output, Axis-4	89	Step Output, Axis-1
70	Index(+) Input, Axis-4	90	Index(+) Input, Axis-1
71	Encoder B(+) Input, Axis-4	91	Encoder B(+) Input, Axis-1
72	Encoder A(+) Input, Axis-4	92	Encoder A(+) Input, Axis-1
73	Amplifier Enable Output, Axis-3	93	Reset Output From Controller
74	Home Input, Axis-3	94	Servo DAC Output, Axis-6
75	Travel Limit(+) Input, Axis-3	95	Servo DAC Output, Axis-5
76	Step Output, Axis-3	96	Servo DAC Output, Axis-4
77	Index(+) Input, Axis-3	07	Servo DAC Output, Axis-3
78	Encoder B(+) Input, Axis-3	98	Servo DAC Output, Axis-2
79	Encoder A(+) Input, Axis-3	99	Servo DAC Output, Axis-1
80	Home Input, Axis-2	100	Cable Interlock Return

Table C.1.1: Main I/O Connector Pin-Outs (Continued)

+5V, 250 mA (maximum)

+5V supply available from the PC.

+12V, 250 mA (maximum)

+12V supply available from the PC.

-12V, 250 mA (maximum)

-12V supply available from the PC.

Amplifier Enable Output

Open-drain output with 1KΩ pull-up resistor to +5 volts. This output is asserted active True when the axis is in the motor ON state and False for the motor OFF. The actual TTL level is under-configurable.

Amplifier Fault Input

The Amplifier Fault input is pulled-up to +5 volts with a 1KΩ resistor. The active true state is user-configurable.

Analog Ground

Servo digital-to-analog (DAC) ground.

Cable Interlock Input

The Cable Interlock input is pulled-up to +5 volts with a 1KΩ resistor. If this input is a logical HIGH then is assumed that the 100-pin cable is not properly fastened and a Cable Interlock error will be generated.

Cable Interlock Return

This is the return for the Cable Interlock Input. This signal should be coupled to the Cable Interlock Input at the motor driver (amplifier) side to indicate that the 100-pin connector is properly fastened.

Digital Ground

Ground reference used for all digital signals.

E-Stop Input

The Emergency Stop input is pulled-up to –5 volts with a 1KΩ resistor. When this signal is asserted the controller will perform an Emergency Stop procedure as configured by the user. When used with the UniDriver6000 motor driver, this signal is coupled to the Stop All front panel pushbutton.

Home Input

This input is pulled-up to +5 volts with a 1KΩ resistor. 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 +5 volts with a 1KΩ resistor 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 +5 volts and pulled down to ground with 1KW resistors. 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 A(+) Input

The A(+) input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The A(-) encoder 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 +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(-) encoder 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 +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The B(+) encoder 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 +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(-) encoder encoded signal originates from the stage position feedback circuitry and is used for position tracking.

Reset Output From Controller

The Reset output is a TTL-buffered output that represents ESP6000 hardware reset status of the controller itself. When the controller is held in a reset state this output is a logical LOW. When connected to the UniDrive6000 this output resets all driver channels.

Servo DAC Output

The servo digital-to-analog converter (DAC) output is the ±10 volt control signal used to control DC servo motors. This signal is the output of the 18-bit servo DAC.

Step Output

The Step Output is an open collector (i.e., 7407) output pulled-up to +5 volts with a 1KΩ resistor. This output is used to control the commutation sequence of a stepper motor. The motor will increment one step for each 'pulse' output.

Step/Direction Output

The Step/Direction Output is an open collector (i.e., 7407) output pulled-up to +5 volts with a 1KΩ resistor. This output is used to control the commutation sequence of a stepper motor. In +Step/-Step mode, the motor will increment one step for each 'pulse' output. In Step/Direction mode, this signal will control the direction of motor rotation.

Travel (+)Limit Input

This input is pulled-up to +5 volts with a 4.7KΩ resistor and represents the stage positive 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 +5 volts with a 4.7KΩ resistor and represents the stage negative direction hardware travel limit. The active true state is user-configurable, the default is active HIGH.

C.1.2 Motor/Driver Interface (100-to-68 Pin) Cable

This cable interfaces the ESP6000 controller card to the Universal Interface Box (UIB). Connector pin-outs are listed in **Table C.1.2** and functional descriptions are provided in the following paragraphs.

68-Pin Connector	100-Pin Connector	Function
1	99	Servo DAC Output, Axis-1
2	47	Analog Ground
8	91	Encoder B(+) Input, Axis-1
9	92	Encoder A(+) Input, Axis-1
10	90	Index(+) Input, Axis-1
11	87	Home Input, Axis-1
12	88	Travel Limit(+) Input, Axis-1
13	02	+05V, 250 mA (maximum)
14	38	Travel Limit (-) Input, Axis-1
15	43	Digital Ground
16	97	Servo DAC Output, Axis-3
17	48	Analog Ground
23	78	Encoder B(+) Input, Axis-3
24	79	Encoder A(+) Input, Axis-3
25	77	Index(+) Input, Axis-3
26	74	Home Input, Axis-3
27	03	+05V, 250 mA (maximum)
28	75	Travel Limit(+) Input, Axis-3
29	46	Digital Ground
30	25	Travel Limit (-) Input, Axis-3
33	36	Amplifier Enable Output, Axis-2
34	23	Amplifier Enable Output, Axis-4
36	46	Digital Ground
37	98	Servo DAC Output, Axis-2
38	45	Digital Ground

Table C.1.2: Motor/Driver Interface (100-to-68 pin) Cable Connector Pin-Outs

68-Pin Connector	100-Pin Connector	Function
44	84	Encoder B(+) Input, Axis-2
45	85	Encoder A(+) Input, Axis-2
46	83	Index(+) Input, Axis-2
47	80	Home Input, Axis-2
48	81	Travel Limit(+) Input, Axis-2
49	31	Travel Limit(-) Input, Axis-2
50	96	Servo DAC Output, Axis-4
51	44	Digital Ground
57	71	Encoder B(+) Input, Axis-4
58	72	Encoder A(+) Input, Axis-4
59	70	Index(+) Input, Axis-4
60	67	Home Input, Axis-4
61	68	Travel Limit(+) Input, Axis-4
62	18	Travel Limit (-) Input, Axis-4
63	50	+12V, 250mA (maximum)
64	49	-12V, 250mA (maximum)
66	86	Amplifier Enable Output, Axis-1
67	73	Amplifier Enable Output, Axis-3
68	93	Reset Output From Controller

Table C.1.2: Motor/Driver Interface (100-to-68 pin) Cable Connector Pin-Outs (Continued)

+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.

Amplifier Enable Output, Axis 1, 2, 3, 4

Open-drain output with 1KΩ pull-up resistor to +5 volts. This output is asserted active True when the axis is in the motor ON state and False for the motor OFF. The actual TTL level is user-configurable.

Analog Ground

Servo digital-to-analog (DAC) ground.

Digital Ground

Ground reference used for all digital signals.

Encoder A(+) Input, Axis 1, 2, 3, 4

The A(+) input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The A(+) encoder encoded signal originates from the stage position feedback circuitry and is used for position tracking.

Home Input, Axis 1, 2, 3, 4

This input is pulled-up to +5 volts with a 1KΩ resistor. The Home signal originates from the stage and is used for homing the stage to a repeatable location.

Index(+) Input, Axis 1, 2, 3, 4

The (+)Index input is pulled-up to +5 volts with a 1KΩ resistor and is buffered with 26LS32 differential receiver. The (+)index signal originates from the stage and is used for homing the stage to a repeatable location.

Reset Output From Controller

The Reset output is a TTL buffered output that represents ESP6000 hardware reset status of the controller itself. When the controller is held in a reset state this output is a logical LOW. When connected to the UniDrive6000 this output resets all driver channels.

Travel Limit(-) Input, Axis 1, 2, 3, 4

This input is pulled-up to +5 volts with a 4.7KΩ resistor and represents the stage negative direction hardware travel limit. The active true state is user-configurable. The default is active HIGH.

Travel Limit(+) Input, Axis 1, 2, 3, 4

This input is pulled-up to +5 volts with a 4.7KΩ resistor and represents the stage positive direction hardware travel limit. The active true state is user-configurable. The default is active HIGH.

C.1.3 Digital I/O (50-Pin) JP4 Connector

This connector provides access to the ESP6000 Opto22™ compatible 24-bit digital I/O interfaces. Connector pin-outs are listed in **Table C.1.3,** and functionally described in the following paragraphs.

JP4 Pins	Function	DB 50 Pins
1	Port C, Bit-7	1
2	Ground	2
3	Port C, Bit-6	3
4	Ground	4
5	Port C, Bit-5	5
6	Ground	6
7	Port C, Bit-4	7
8	Ground	8
9	Port C, Bit-3	9
10	Ground	10
11	Port C, Bit-2	11
12	Ground	12
13	Port C, Bit-1	13
14	Ground	14
15	Port C, Bit-0	15
16	Ground	16
17	Port B, Bit-7	17

Table C.1.3: Digital Connector Pin-Outs

JP4 Pins	Function	DB 50 Pins
18	Ground	18
19	Port B, Bit-6	19
20	Ground	20
21	Port B, Bit-5	21
22	Ground	22
23	Port B, Bit-4	23
24	Ground	24
25	Port B, Bit-3	25
26	Ground	26
27	Port B, Bit-4	27
28	Ground	28
29	Port B, Bit-1	29
30	Ground	30
31	Port B, Bit-0	31
32	Ground	32
33	Port A, Bit-7	33
34	Ground	34
35	Port A, Bit-6	35
36	Ground	36
37	Port A, Bit-5	37
38	Ground	38
39	Port A, Bit-4	39
40	Ground	40
41	Port A, Bit-3	41
42	Ground	42
43	Port A, Bit-2	43
44	Ground	44
45	Port A, Bit-1	45
46	Ground	46

Table C.1.3: Digital Connector Pin-Outs (Continued)

JP4 Pins	Function	DB 50 Pins
47	Port A, Bit-0	47
48	Ground	48
49	+5V, 250mA (maximum)	49
50	Ground	50

Table C.1.3: Digital Connector Pin-Outs (Continued)

+5V, 250mA (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 1KΩ 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.1.4 Auxiliary I/O (40-Pin) JP5 Connector

This connector provides access to the ESP6000 auxiliary I/O signals. The auxiliary I/O connector provides access to: (a) additional quadrature encoder counters, (b) digital I/O, and (c) E-Stop input. Connector pin-outs are listed in **Table C.1.4,** and functionally described in the following paragraphs.

JP5 Pins	Function	DB 37 Pins
1	Auxiliary Ch. 7 Input A(+)	1
2	Auxiliary Ch. 7 Input A(-)	20
3	Auxiliary Ch. 7 Input B (+)	2
4	Auxiliary Ch. 7 Input B (-)	21
5	Reserved	3
6	Reserved	22
7	Auxiliary Ch. 8 Input A(+)	4

Table C.1.4: Auxiliary Connector Pin-Outs

JP5 Pins	Function	DB 37 Pins
8	Auxiliary Ch. 8 Input A(-)	23
9	Auxiliary Ch. 8 Input B (-)	5
10	Auxiliary Ch. 8 Input B(-)	24
11	Reserved	6
12	Reserved	25
13	Axis-6 Encoder Input A	7
14	Axis-6 Encoder Input B	26
15	Reserved	8
16	Reserved	27
17	Axis-1 position compare output	9
18	Axis-2 position compare output	28
19	Digital Ground	10
20	Global position latch input	29
21	Port A, Bit-0	11
22	Port A, Bit-1	30
23	Port A, Bit-2	12
24	Port A, Bit-3	31
25	Port B, Bit-0	13
26	Port B, Bit-1	32
27	Port B, Bit-2	14
28	Port B, Bit-3	33
29	E-Stop	15
30	Reserved	34
31	+5V, 250mA (maximum)	16
32	DSP Reset Output	35
33	+12V, 250mA (maximum)	17
34	Reserved	36
35	-12V, 250mA (maximum)	18
36	Reserved	37

Table C.1.4: Auxiliary Connector Pin-Outs (Continued)

JP5 Pins	Function	DB 37 Pins
37	Digital Ground	19
38	Unused	-
39	Unused	-
40	Unused	-

Table C.1.4: Auxiliary Connector Pin-Outs (Continued)

+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.

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 100-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 +5 volts 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 100-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(+)

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 different 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 position tracking.

Auxiliary Ch. 8 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.

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/O connector (JP4) 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 ESP6000 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 ESP6000 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 minimum 10 microsecond duration. This signal should be debounced so as not to generate multiple E-Stops 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 UniDriver6000 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 +5V 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.1.5 Analog I/O (26-Pin) JP2 Connector

This connector interfaces the ESP6000 controller card to customer-defined analog I/O devices. Connector pin-outs are listed in **Table C.1.5,** and functionally described in the following paragraphs.

JP2 Pins	Function	DB 25 Pins
1	Digital Ground	1
2	E-Stop Interrupt to DSP	14
3	-12V, 250mA (maximum)	2
4	Reserved	15
5	+12V, 250mA (maximum)	3
6	Reset Output from DSP	16
7	+5V, 250mA (maximum)	4
8	Reserved	17
9	Reserved	5
10	Reserved	18
11	Analog Ground	6
12	Axis-6 Servo DAC Output	19
13	Analog Ground	7
14	Axis-5 Servo DAC Output	20
15	Analog Ground	8
16	Reserved	21
17	Analog Ground	9
18	Analog Input 0	22
19	Analog Input 1	10
20	Analog Input 2	23
21	Analog Input 3	11
22	Analog Input 4	24
23	Analog Input 5	12
24	Analog Input 6	25
25	Analog Input 7	13
26	Unused	-

Table C.1.5: Analog Connector Pin-Outs

+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 9, Data Acquisition for information.

Axis-5, 6 Servo DAC Output

The servo Digital-to-Analog converter (DAC) output is the ±10 volt, 18-bit resolution control signal used 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 ESP6000 controller card.

Reset Output from DSP

This signal is a buffered active-low signal connected to the ESP6000 controller card reset signal.

C.2 UniDrive6000 Universal Motor Driver

C.2.1 Controller Input Connector

This connector interfaces the UniDrive6000 to the ESP6000 controller card via a one hundred-pin Newport-supplied cable. Refer to the ESP6000 controller card paragraph in this section for pin-out descriptions. Connector orientation is shown in **Figure C.2.1.**

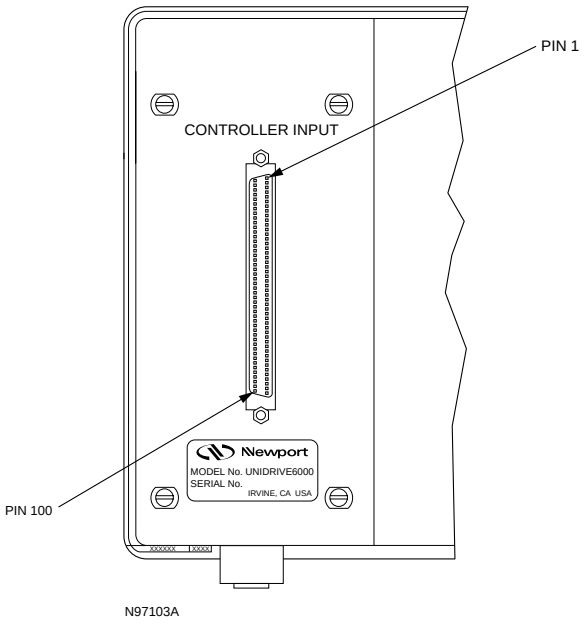


Figure C.2.1: UniDrive Controller Input Connector Orientation

C.2.2 Motor Driver Card 25-Pin I/O Connector

This connector interfaces a UniDrive6000 driver card to motorized stages. Cabling to the connector is provided with the applicable stage. Connector orientation is shown in **Figure C.2.2** and pin-outs are listed in **Table C.2.1.**

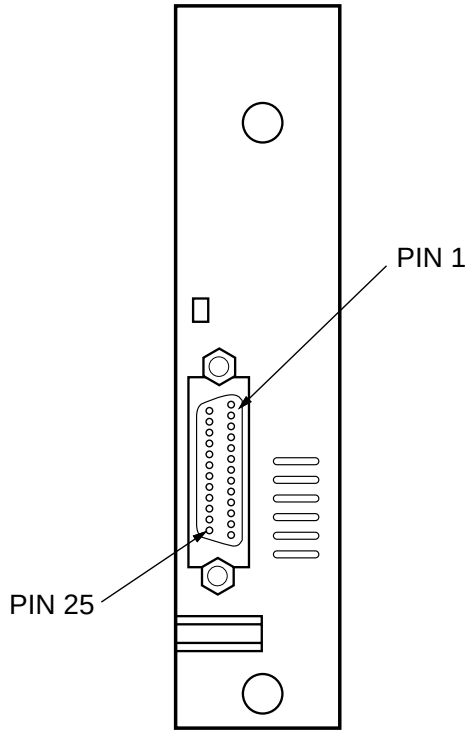


Figure C.2.2: Driver Card Connector Orientation

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

Table C.2.1: Driver Card Connector Pin-Outs

Pins	Stepper Motor	DC Motor
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: +5 / +12 (+5V Standard)	Encoder Supply: +5 / +12 (+5V Standard)
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.2.1: Driver Card Connector Pin-Outs (Continued)

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 a maximum amplitude of 60V DC.

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 a maximum amplitude of 60V 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 a maximum amplitude of 60V 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 a maximum amplitude of 60V 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 a maximum amplitude of 60V 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 a maximum amplitude of 60V DC.

Tacho Generator(+) Input

This input can be connected to the positive lead of a tachometer. The maximum input voltage range is +/-60V.

Tacho Generator(-) Input

This input can be connected to the positive lead of a tachometer. The maximum input voltage range is +/-60V.

Travel(+) Limit Input

This input is pulled-up to +5 volts with a 4.7KΩ resistor by the controller and represents the stage positive 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 +5 volts 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.

Encoder A (+) Input

The A(+) input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The A(+) encoder encoded signal originates from the stage position feedback circuitry and is used for position tracking.

Encoder A(-) Input

The A9-) 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(-) encoder 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 +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The B(-) encoder 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 +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(-) encoder 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 +5 volts 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 +5 volts 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 +5 volts 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: +5/ +12 (+5V Standard), 250mA (maximum)

+5V or +12V supply available from the IniDrive6000. The standard supply configuration is +5 volts (250mA maximum). This supply is provided for stage home, index, travel limit, and encoder feedback circuitry.

Limit Ground

Ground for stage travel limit signals. Limit ground is combined with digital ground at the controller side.

Shield Ground

Motor cable shield ground.

C.3 Terminal Block Board

The terminal block board provides access to I/O signals available on the 100-pin connector. This is useful for applications where a motor driver other than a UniDrive6000 is going to be incorporated. The various connector functions and signals are defined in the following paragraphs.

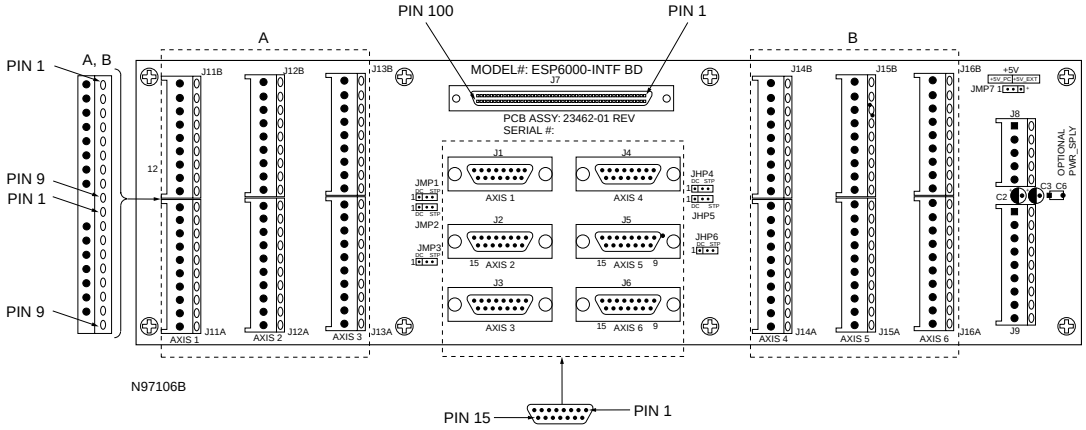


Figure C.3.1: Terminal Block Board Connector Orientation

C.3.1 MD4 15-Pin Connector

This connector is used for interfacing with the MD4 motor driver. Connector pin-outs are listed in **Table C.3.1,** and functionally described in the following paragraphs.

J1-J6 Pins	Function
1	Not Connected
2	+5V
3	Enable_1
4	A_1
5	+P_1
6	-P_1
7	Home_1
8	1_1
9	Not Connected
10	Not Connected
11	Not Connected
12	B_1
13	+Limit_1
14	-Limit_1
15	DGND

Table C.3.1: MD4 Connector Pin-Outs

+5V

+5 volt (250mA maximum) supply. This supply is provided for stage home, index, travel limit, and encoder feedback circuitry.

+Limit_1

This input is pulled-up to +5 volts with a 4.7KΩ resistor by the controller and represents the stage positive direction hardware travel limit. The active true state is user configurable. The default is active HIGH.

-Limit_1

This input is pulled-up to +5 volts with a 4.7KΩ resistor by the controller and represents the stage positive direction hardware travel limit. The active true state is user-configurable. The default is active HIGH.

+P_1

The Step Output is an open collector (i.e., 7407) output pulled-up to +5 volts with a 1KΩ resistor. This output is used to control the commutation sequence of a step motor. The motor will increment one step for each 'pulse' output.

-P_1

The Step/Direction Output is an open collector (i.e., 7407) output pulled-up to +5 volts with a 1KΩ resistor. This output is used to control the commutation sequence of a stepper motor. In +Step/ -Step mode, the motor will increment one step for each 'pulse' output. In Step/Direction mode, this signal will control the direction of motor rotation.

A_1

The A_1 input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The A_1 encoder encoded signal originates from the stage position feedback circuitry and is used for position tracking.

B_1

The B_1 input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The B_1 encoder encoded signal originates from the stage position feedback circuitry and is used for position tracking.

DGND

Digital ground.

Enable_1

Open-drain output with 1KΩ pull-up resistor to +5 volts. This output is asserted active True when the axis is in the motor ON state and False for the motor OFF state. The actual TTL level is user-configurable.

Home_1

This input is pulled-up to +5 volts 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.

I_1

The (+)Index input is pulled-up to +5 volts 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.

C.3.2 Eighteen-Lead Connector

This connector interfaces the ESP6000 controller card to customer-defined devices. The connector is physically comprised of two banks of leads, upper and lower. Connector pin-outs are listed in **Table C.3.2** and **C.3.3,** and functionally described in the following paragraphs.

J11B-J16B Leads	Function
1	A_1-_6
2	A_1-_6
3	B_1-_6
4	B_1-_6
5	I_1-_6
6	I_1-_6
7	+5V
8	DGND
9	DGND

Table C.3.2: Eighteen-Lead Upper Connector Pin-Outs

+5V

+5 volt (250mA maximum) supply. This supply is provided for stage home, index, travel limit, and encoder feedback circuitry.

A_1-_6

The A_1 input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The A_1 encoder encoded signal originates from the stage position feedback circuitry and is used for position tracking.

B_1-_6

The B_1 input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The B_1 encoder encoded signal originates from the stage position feedback circuitry and is used for position tracking.

DGND

Digital ground.

I_1-_6

The I_1 index input is pulled-up to +5 volts 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.

J11A-J16A Leads	Function
1	Home_1-_6
2	-Limit_1-_6
3	+Limit_1-_6
4	Fault_1-_6
5	Enable_1-_6
6	-P_1-_6
7	+P_1-_6
8	AGND
9	ESP_DAC_1 – 6

Table C.3.3: Eighteen-Lead Lower Connector Pin-Outs

+Limit_1-_6

This input is pulled-up to +5 volts with a 4.7KΩ resistor by the controller and represents the stage position direction hardware travel limit. The active true state is user-configurable. The default is active HIGH.

-Limit_1-_6

This input is pulled-up to +5 volts with a 4.7KΩ resistor and represents the stage negative direction hardware travel limit. The active true state is user-configurable. The default is active HIGH.

+P_1-_6

The Step Output is an open collector (i.e., 7407) output pulled-up to +5 volts with a 1KΩ resistor. This output is used to control the commutation sequence of a stepper motor. The motor will increment one step for each 'pulse' output.

-P_1-_6

The Step/Direction Output is an open collector (i.e., 7407) output pulled-up to +5 volts with a 1KΩ resistor. This output is used to control the commutation sequence of a step motor. In +Step/ -Step mode, the motor will increment one step for each 'pulse' output. In Step/Direction mode, this signal will control the direction of motor rotation.

Enable_1-_6

Open-drain output with a 1KΩ pull-up resister to +5 volts. This output is asserted active True when the axis is in the motor ON state and False for the motor OFF state. The actual TTL level is user-configurable.

Home_1-_6

This input is pulled-up to +5 volts with a 1KΩ resister by the controller. The Home signal originates from the stage and is used for homing the stage to a repeatable location.

Fault_1-_6

This Amplifier Fault input is pulled-up to +5 volts with a 1KΩ resistor by the controller. The state of this signal is controlled by the motor driver and monitored by the controller.

ESP_DAC_1 - 6

ESP_DAC is the controller's servo control, ±10 volt analog signal output.

AGND

Servo digital-to-analog converter (DAC) ground.

C.3.3 Optional Power Supply Connector

This connector is used to provide an external supply power for stage home, limit, and encoder circuits. Connector pin-outs are listed in **Table C.3.4,** and functionally described in the following paragraphs.

J8 Pins	Function
1	+5V
2	DGND
3	-12V
4	+12V
5	AGND

Table C.3.4: Optional Power Supply Connector Pin-Outs

+12V

+12 volt supply.

+5V

+5 volt supply.

-12V

-12 volt supply.

AGND

Analog ground.

DGND

Digital ground.

C.3.4 Nine-Lead Connector

This connector provides access to non-motion signals from the ESP6000 controller card. Connector pi-outs are listed in **Table C.3.5,** and functionally described in the following paragraphs.

+12V (250mA maximum)

+12V supply available from the PC.

-12V (250mA maximum)

-12V supply available from the PC.

J9 Leads	Function
1	+12V
2	-12V
3	+5V
4	DGND
5	DGND
6	E_Stop Input
7	Reset Output
8	RS_A
9	RS_B

Table C.3.5: Nine-Pin Connector Pin-Outs

+5V (250mA)

+5 volt (250mA maximum) supply from the PC. This supply is provided for stage home, index, travel limit, and encoder feedback circuitry.

DGND

Digital ground.

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 minimum 10 microsecond duration. This signal should be debounced so as not to generate multiple E-Stops within a 100 millisecond time period. When this signal is asserted the controller will perform an Emergency Stop procedure as configured by the user.

Reset Output

The Reset output is a TTL-buffered output that represents ESP6000 hardware reset status of the controller itself. When the controller is held in a reset state this output is a logical LOW. When connected to the UniDrive6000 this output resets all driver channels.

RS_A

The RS_A input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The RS_A encoder encoded signal originates from the stage position feedback circuitry and is used for position tracking.

RS_B

The RS_B input is pulled-up to +5 volts with a 1KΩ resistor. The signal is buffered with a 26LS32 differential receiver. The RS_B encoder encoded signal originates from the stage position feedback circuitry and is used for position tracking.

C.3.5 Jumpers

Terminal block board jumper functions are listed in **Table C.3.6.**

Reference Designation	Function
JMP1-JMP6	DC/Stepper Motor Selection
JMP7	External Power Supply Selection

Table C.3.6: Jumpers

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 and binary. To also help in working with the I/O port related commands, the **Table D.1** 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	*Dcl*	00010001
18	*Dc2*	00010010

Table D.1: Binary Conversion Table

Number (decimal)	ASCII Code	Binary Code
19	*Dc3*	00010011
20	*Dc4*	00010100
21	*Nak*	00010101
22	*Syn*	00010110
23	*Eth*	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
33	!	00100001
34	"	00100010
35	#	00100011
36	$	00100100
37	%	00100101
38	&	00100110
39	'	00100111
40	(	00101000
41	)	00101001
42	*	00101010
43	+	00101011
44	,	00101100
45	-	00101101
46	.	00101110

Table D.1: Binary Conversion Table (Continued)

Number (decimal)	ASCII Code	Binary Code
47	/	00101111
48	**0**	0011000
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**	01000010
68	**D**	01000100
69	**E**	01000101
70	**F**	01000110
71	**G**	01000111
72	**H**	01001000
73	**I**	01001001
74	**J**	01001010
75	**K**	01001011

Table D.1: Binary Conversion Table (Continued)

Number (decimal)	ASCII Code	Binary Code
76	**L**	01001100
77	**M**	01001101
78	**N**	01001110
79	**O**	01001111
80	**P**	01010000
81	**Q**	01010001
82	**R**	01010010
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

Table D.1: Binary Conversion Table (Continued)

Number (decimal)	ASCII Code	Binary Code
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 (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

Table D.1: Binary Conversion Table (Continued)

Number (decimal)	ASCII Code	Binary Code
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
183		10110111
184		10111000
185		10111001
186		10111010
187		10111011
188		10111100
189		10111101
190		10111110

Table D.1: Binary Conversion Table (Continued)

Number (decimal)	ASCII Code	Binary Code
191		10111111
192		11000000
193		11000001
194		11000010
195		11000010
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

Table D.1: Binary Conversion Table (Continued)

Number (decimal)	ASCII Code	Binary Code
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
233		11101001
234		11101010
235		11101011
236		11101100
237		11101101
238		11101110
239		11101111
240		11110000
241		11110001
242		11110010
243		11110011
244		11110100
245		11110101
246		11110110
247		11110111
248		11111000

Table D.1: Binary Conversion Table (Continued)

Number (decimal)	ASCII Code	Binary Code
249		11111001
250		11111010
251		11111011
252		11111100
253		11111101
254		11111110
255		11111111

Table D.1: Binary Conversion Table (Continued)

Appendix E – System Upgrades

E.1 ESP6000 Controller Card

E.1.1 Installing New Software

Refer to Appendix G, Factory Service to control Newport Corporation for software upgrades.

A new version of the Windows software can be installed any time after the ESP6000 controller card and driver software have been installed. New-version software installations are performed in the same manner as first-time installations. Before installing new software, however, users need to uninstall existing software. Procedures for uninstalling software are provided in the following paragraphs.

From a Windows 95 desk-top, select **START, SETTINGS, CONTROL PANEL,** and **ADD/REMOVE PROGRAMS** (refer to *Installing The Motion Utility in the System Startup* section). The Add/Remove Programs Properties screen appears (See **Figure E.1.1**).

Figure E.1.1: Add/Remove Programs Properties Screen

Select the Install/Uninstall tab from the Add/remove Programs Properties screen, then select ESP-Util and **ADD/REMOVE.** The Select Uninstall Method screen appears (See **Figure E.1.2**).

Select **NEXT** from the Select Uninstall Method screen. The Perform Uninstall screen appears (See **Figure E.1.3**).

Select **FINISH.** A message screen will appear with status information, and the user will be returned to the Add/Remove Programs Properties screen when the uninstall is complete. Refer to *Installing The Windows Software* in the System Startup section for installing the new-version software.

Figure E.1.2: Select Uninstall Method Screen

Figure E.1.3: Perform Uninstall Screen

E.1.2 Installing New Firmware

New firmware installation may be required due to upgrading. Refer to Appendix G, Factory Service to contact Newport Corporation for ordering information.

To begin an installation, select **FIRMWARE** from the ESP6000 Setup Menu (See **Figure E.1.4**).

Figure E.1.4: ESP6000 Setup Menu

The Update Firmware message screen appears (See **Figure E.1.5**).

Figure E.1.5: Update Firmware Message Screen

Select **OK.** The Open screen appears (See **Figure E.1.6**).

NOTE

This is the user's last chance to cancel the download. The user should make sure that they want to burn in the new firmware before selecting Open.

Figure E.1.6: Open Screen

Select the .fwr file in the Firmware directory and select **OPEN** (this file is supplied with initial delivery/installation). The ESP6000 Firmware Update screen appears (See **Figure E.1.7**).

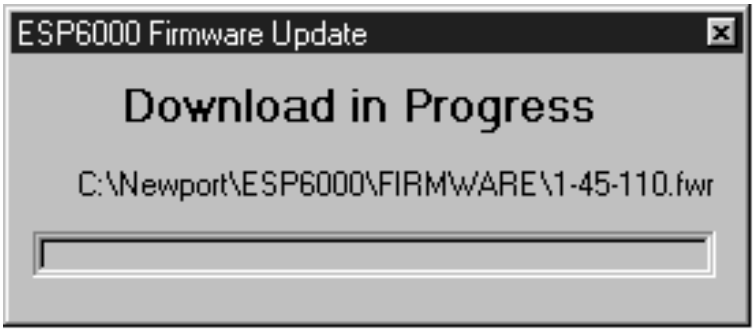


Figure E.1.7: Firmware Update Screen

The Firmware Update screen will display status until downloading is complete, then the user will be returned to the ESP6000 Initialization screen (See **Figure E.1.8**).

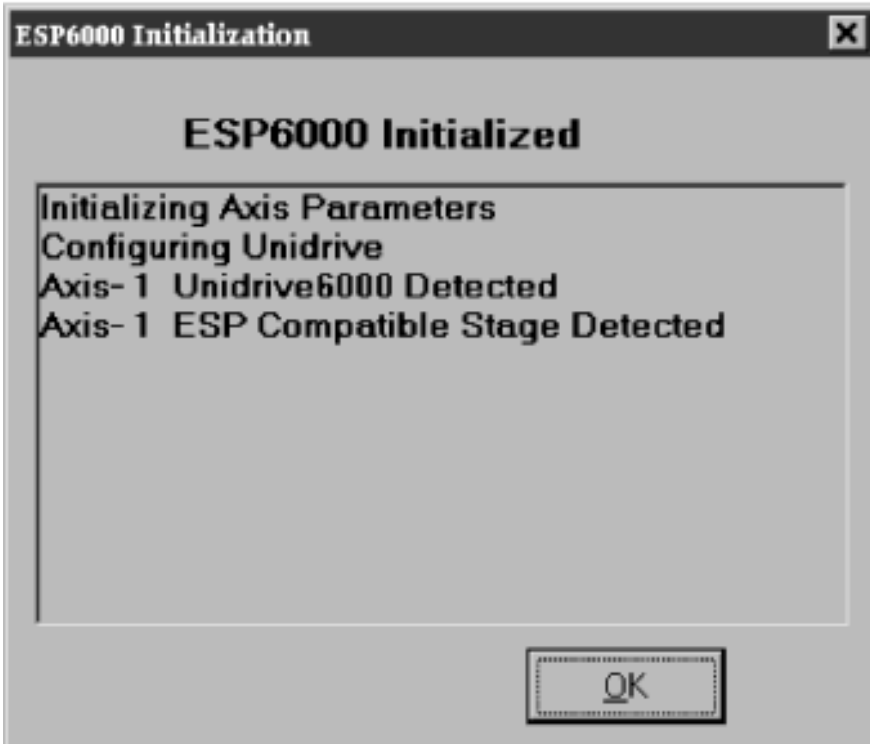


Figure E.1.8: ESP6000 Initialization Screen

When the screen message indicates that the ESP6000 is initialized, the system is ready for use again.

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 ESP6000 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 ESP6000 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,** as shown on the following page.

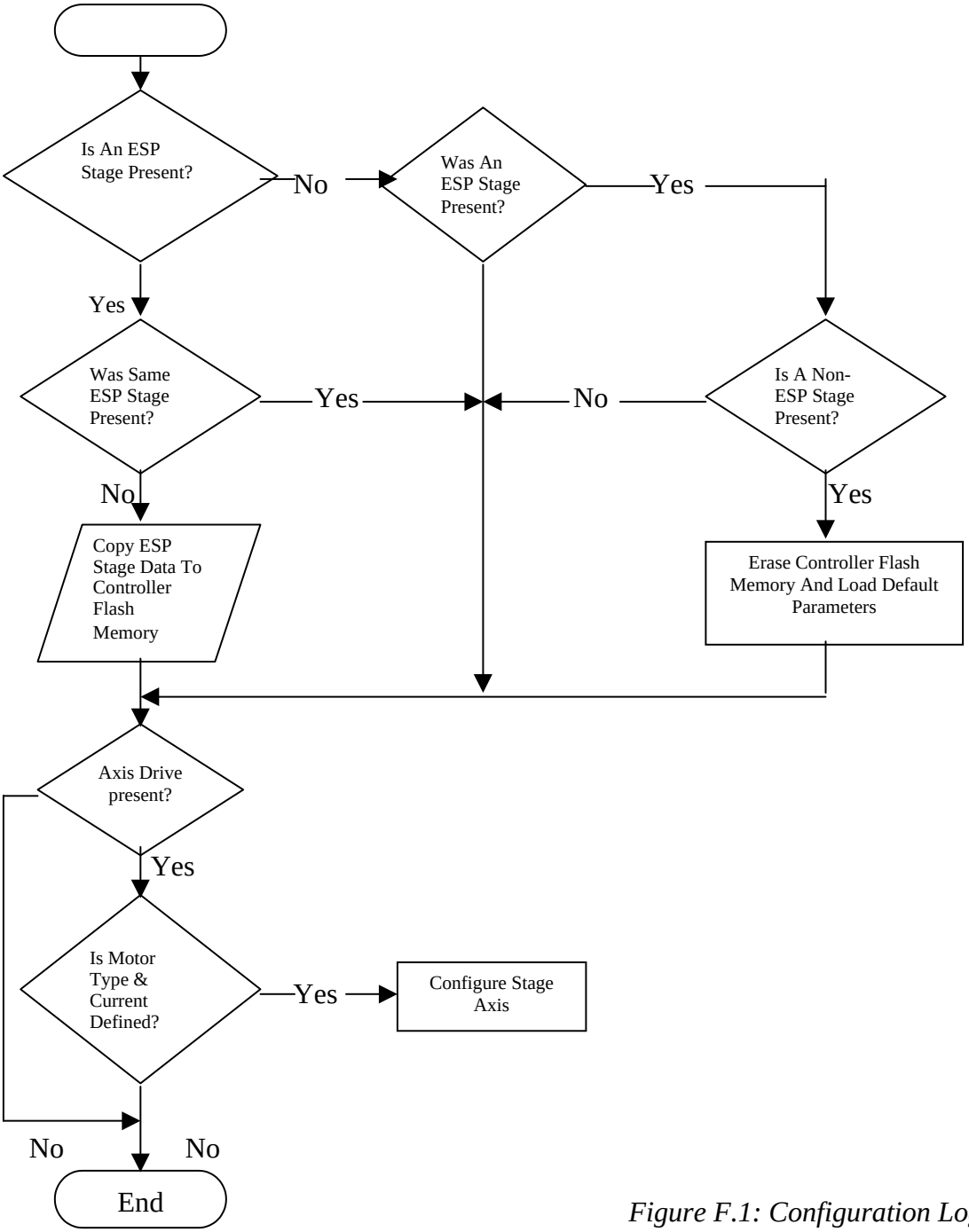


Figure F.1: Configuration Logic

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

1 qm 1	//set motor type to DC servo
1 qi0.15	//set motor maximum current to 0.15 amps
1 qv30	//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 25 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 command stepper (ESP6000)
1qu1	//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
1fr 0.01	//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
1ac30	//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 ESP6000 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	newport.tech@newport.com
Web Page URL	www.newport.com/support/technical_help

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 (ESP6000)
- Equipment serial number (for either ESP6000 controller card or UniiDrive6000 universal motor driver)
- Distribution revision number from a floppy disk (See **Figure 4.1.34** in the Windows Utilities section)
- Problem description (document using the Service Form, see the following page)

If the instrument is to be returned for repair, the user will be given a Return Authorization Number that should be referenced in their shipping documentation. Complete a copy of the Service Form on the next page and include it with your shipment.

H.1 Service Form

Newport Corporation

U.S.A. Office: 949/863-3144

FAX: 949/253-1800

Name _____ RETURN AUTHORIZATION # _____

Company _____ (Please obtain RA # prior to return of item)

Address _____ Date _____

Country _____ Phone Number _____

P.O. Number _____ FAX Number _____

Item(s) Being Returned:

Model # _____ Serial # _____

Description: _____

Reason for return of goods (please list any specific problems) _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____