

PACSystems™ RX3i & Series 90-30

DSM314 MOTION CONTROLLER

USER MANUAL

Contents

Chapter 1: Product Overview	1
1.1 Features of the Motion Mate DSM314.....	2
1.1.1 High Performance	2
1.1.2 Easy to Use	2
1.1.3 Versatile I/O	3
1.2 Section 1: Motion System Overview.....	4
1.2.1 DSM314 Operation with a Host Controller.....	4
1.3 Section 2: Overview of DSM314 Operation	9
1.3.1 Standard Mode Operation	10
1.4 Section 3: α Series Servos (Digital Mode)	12
1.4.1 α Series Integrated Digital Amplifier (SVU)	13
1.5 Section 4: β Series Servos (Digital Mode)	14
1.5.1 β Series Digital Amplifiers	15
1.6 Section 5: SL Series Servos (Analog Velocity Mode)	16
Chapter 2: System Overview	17
2.1 Unpacking the System.....	18
2.1.1 Unpacking the DSM314.....	18
2.1.2 Unpacking the Digital Servo Amplifier	18
2.1.3 Unpacking the Motor	18
2.2 Assembling the Motion Mate DSM314 System.....	19
2.2.1 General Guidelines	19
2.2.2 Motion Mate DSM314 Connections.....	19
2.2.3 Connecting the α Series SVU Digital Servo Amplifier	20
2.2.4 Connecting the β Series SVU Digital Servo Amplifier	28
2.2.5 Installing and Wiring the DSM314 for Analog Mode.....	36
2.2.6 Grounding the Motion Mate DSM314 Motion System.....	37
2.3 Turning on the Motion Mate DSM314.....	38
2.4 Connecting the Programmer to the Host Controller	39
2.5 Machine Edition Configuration	40
2.6 Storing Your Configuration to the Host Controller	44
2.7 Alarms.....	46
2.8 Configuration Settings	46
2.9 Getting Help.....	46

Chapter 3: Installing and Wiring the DSM314.....	47
3.1 Hardware Description.....	47
3.1.1 LED Indicators	48
3.1.2 The DSM COMM (Serial Communications) Connector.....	49
3.1.3 I/O Connectors.....	49
3.1.4 Shield Ground Connection	50
3.2 Installing the DSM314 Module.....	51
3.3 I/O Wiring and Connections.....	54
3.3.1 I/O Circuit Types.....	54
3.3.2 Terminal Boards	55
3.3.3 Digital Servo Axis Terminal Board - IC693ACC335.....	56
3.3.4 Auxiliary Terminal Board - IC693ACC336	61
3.3.5 Cables.....	64
Chapter 4: Configuration	93
4.1 Connecting the Programmer to the Host Controller	93
4.2 Rack/Slot Configuration.....	94
4.3 Module Configuration	97
4.3.1 Setting the Configuration Parameters	98
4.3.2 Settings.....	99
4.3.3 Serial Communications Port Configuration Data.....	104
4.3.4 Control (CTL) Bits	105
4.3.5 Output Bits.....	106
4.3.6 Axis Configuration Data.....	107
4.3.7 Tuning Data	122
4.3.8 Computing Data Limit Variables	128
4.3.9 Advanced Tab Data	129
4.3.10 Power Consumption Data	130
Chapter 5: DSM314 to Host Controller Interface.....	131
5.1 Section 1: %I Status Bits.....	131
5.2 Section 2: %AI Status Words	136
5.3 Section 3: %Q Discrete Commands.....	139
5.4 Section 4: %AQ Immediate Commands	145
Chapter 6: Non-Programmed Motion.....	163
6.1 DSM314 Home Cycle.....	163

6.1.1	Home Switch Mode	163
6.1.2	Move+ and Move- Modes	166
6.2	Jogging with the DSM314	167
6.3	Move at Velocity Command.....	168
6.4	Force Servo Velocity Command (DIGITAL Servos; Analog Torque Mode)	169
6.5	Force Analog Output Command (ANALOG Velocity Interface Servos)	169
6.6	Position Increment Commands	170
6.7	Other Considerations	170

Chapter 7: Programmed Motion 171

7.1	Single-Axis Motion Programs and Subroutines	171
7.2	Multi-Axis Motion Programs and Subroutines	172
7.3	Motion Program Command Types	172
7.4	Program Blocks and Motion Command Processing	174
7.5	Prerequisites for Programmed Motion	174
7.6	Conditions That Stop a Motion Program	175
7.7	Motion Program Basics	175
7.7.1	Motion Language Syntax and Commands	176
7.7.2	Motion Program Commands	178
7.7.3	Program and Subroutine Structure	187
7.7.4	Command Usage Examples	190
7.7.5	Types of Programmed Move Commands	192
7.7.6	Other Programmed Motion Considerations	208
7.7.7	Feedhold with the DSM314	210
7.7.8	Feedrate Override	211
7.7.9	Multi-axis Programming	212
7.7.10	Parameters (P0-P255) in the DSM314.....	213
7.7.11	Calculating Acceleration, Velocity and Position Values	215
7.7.12	Motion Editor Error and Warning Messages	218

Chapter 8: Follower Motion 223

8.1	Master Sources	223
8.2	External Master Inputs	224
8.2.1	Example 1: Following Axis 3 Actual Position Master Input	224
8.3	Internal Master Axis Command Generators	224
8.3.1	Example 2: Following an Internal Master command	225
8.4	A:B Ratio.....	225

8.4.1	Example 3: Sample A:B Ratios	226
8.5	Velocity Clamping	227
8.5.1	Example 5: Velocity Clamping	227
8.6	Unidirectional Operation	228
8.6.1	Example 9: Unidirectional Operation	228
8.7	Enabling the Follower with External Input	228
8.8	Disabling the Follower with External Input	229
8.9	Follower Disable Action Configured for Incremental Position	229
8.10	Follower Axis Acceleration Ramp Control.....	229
8.10.1	Follower Mode Command Source and Connection Options	233

Chapter 9: Combined Follower and Commanded Motion 239

9.1	Example 1: Follower Motion Combined with Jog	239
9.2	Follower Motion Combined with Motion Programs	240
9.3	Example 2: Follower Motion Combined with Motion Program	244

Chapter 10: Introduction to Local Logic Programming 246

10.1	Local Logic Programming	246
10.2	When to Use Local Logic Versus Ladder Logic.....	249
10.3	Getting Started with Local Logic and Motion Programming	249
10.3.1	Requirements	249
10.3.2	Creating a Local Logic Program	250
10.4	Local Logic Variable Table	251
10.5	Connecting the Local Logic Editor to the DSM.....	253
10.6	Building a Local Logic Program	254
10.6.1	Creating a Local Logic Program	254
10.6.2	Checking Local Logic Syntax	257
10.6.3	Setting up Hardware Configuration for Local Logic	258
10.7	Downloading a Local Logic Program	262
10.8	Executing Your Local Logic Program	264
10.9	Using the Motion Program Editor.....	265
10.9.1	Creating a Motion Program	265
10.9.2	Setting Motion Program Parameters in Hardware Configuration	271
10.10	Executing Your Motion Program	274

Chapter 11: Local Logic Tutorial 275

11.1	Statements.....	275
11.2	Comments	276

11.3	Variables	276
11.4	Operators	277
11.4.1	Arithmetic Operators	277
11.4.2	Relational Operators	278
11.4.3	Bitwise Logical Operators	279
11.5	Local Logic / Host Controller / Motion Program Communication	280
11.6	Local Logic Programming Examples	280
11.6.1	Torque Limiting Program Example	280
11.6.2	Gain Scheduler Program Example	282
11.6.3	Programmable Limit Switch Program Example	282
11.6.4	Trigger Output Based Upon Position Program Example	283
11.6.5	Windowing Strokes Program Example	285

Chapter 12: Local Logic Language Syntax..... 286

12.1	Syntactic Elements	286
12.1.1	Numeric Constants	286
12.1.2	Local Logic Variables	287
12.1.3	Local Logic Statements	288
12.1.4	Whitespace	289
12.1.5	Comments	290
12.1.6	PRAGMA Directive	291
12.1.7	Local Logic Keywords and Operators	291
12.2	Enabling and Disabling Local Logic	292
12.3	Local Logic Outputs/Commands	292
12.4	Local Logic Arithmetic Operators	293
12.4.1	Operator +	294
12.4.2	Operator -	294
12.4.3	Operator *	295
12.4.4	Operator MOD	295
12.4.5	Function ABS	296
12.5	Local Logic Bitwise Logical Operators	296
12.5.1	Operator BWAND	297
12.5.2	Operator BWOR	297
12.5.3	Operator BWXOR	298
12.5.4	Operator BWNOT	298
12.6	Comparison Operators	299

12.7	Local Logic Runtime Errors.....	300
12.7.1	Overflow Status.....	300
12.8	Local Logic Error Messages	301
12.8.1	Local Logic Build Error Messages.....	301
12.8.2	Local Logic Syntax Errors	302
12.8.3	Local Logic Parse Errors	302
12.8.4	Local Logic Parse Warnings.....	305
12.8.5	Local Logic Download Error Messages	305
12.8.6	Local Logic Runtime Errors	307
Chapter 13: Local Logic Variables.....		308
13.1	Local Logic Variable Types	308
13.2	Local Logic System Variables	309
13.2.1	First_Local_Logic_Sweep Variable	309
13.2.2	Overflow Variable.....	309
13.2.3	System_Halt Variable	310
13.3	Double Precision 64 Bit Registers.....	310
13.4	Local Logic User Data Table	311
13.5	Digital Outputs / CTL Variables	312
Chapter 14: Local Logic Configuration.....		319
14.1	CTL Bit Configuration.....	319
14.2	CTL bits CTL01-CTL32	320
14.3	CTL01-CTL24 Bit Configuration Selections	321
14.4	FBSA Function and CTL Bit Assignments.....	322
14.5	Faceplate Output Bit Configuration	322
Chapter 15: Using the Electronic CAM Feature		324
15.1	Electronic CAM Overview.....	324
15.2	Basic Cam Shapes/Definition	326
15.3	CAM Syntax	327
15.3.1	CAM Types	327
15.3.2	Interpolation and Smoothing	330
15.3.3	Interaction of Motion Programs with CAM	332
15.3.4	CAM Command.....	333
15.3.5	CAM-LOAD Command.....	334
15.3.6	CAM-PHASE Command	335

15.3.7	CAM and MOVE Instructions.....	335
15.3.8	Time-Based CAM Motion	336
15.3.9	CAM Scaling Editor and Hardware Configuration	336
15.3.10	Synchronization of CAM Motion with External Events	340
15.3.11	CAM-Specific DSM Error Codes	341
15.4	Electronic Cam Programming Basics	343
15.4.1	Requirements	343
15.4.2	Introduction to Electronic Cam Programming	343

Appendix A: Error Reporting..... 364

A-1	DSM314 Error Codes	364
A-1.1	Module Status Code Word.....	364
A-1.2	Axis Error Code Words.....	364
A-1.3	Error Code Format.....	365
A-1.4	Response Methods.....	365
A-1.5	System Error Codes	384
A-2	DSM Digital Servo Alarms (B0–BE)	384
A-3	Troubleshooting Digital Servo Alarms.....	386
A-4	LED Indicators	389

Appendix B: DSM314 Communications Request Instructions .. 391

B-1	Communications Request Overview	391
B-1.1	Structure of the Communications Request	392
B-1.2	Monitoring the Status Word	394
B-1.3	Operation of the Communications Request.....	395
B-2	The COMM REQ Ladder Instruction	396
B-3	The User Data Table (UDT) COMM REQ.....	398
B-3.1	User Data Table COMM REQ Features and Usage Information	398
B-3.2	The UDT COMM REQ Command Block.....	399
B-3.3	User Data Table COMM REQ Example	401
B-3.4	User Data Table COMM REQ Example	403
B-4	The Parameter Load COMM REQ.....	404
B-4.1	The Command Block	404
B-4.2	DSM Parameter Load COMM REQ Example	407
B-5	COMM REQ Ladder Logic Example	409

Appendix C: Position Feedback Devices 414

C-1	Digital Serial Encoder Modes	414
C-2	Incremental Encoder Mode Considerations.....	414
C-3	Absolute Encoder Mode Considerations.....	415
C-3.1	Absolute Encoder - First Time Use or Use After Loss of Encoder Battery Power	415
C-3.2	Absolute Encoder Mode - Position Initialization	415
C-3.3	Absolute Encoder Mode - DSM314 Power-Up	416
C-3.4	Incremental Quadrature Encoder	417
Appendix D: Tuning Digital and Analog Servo Systems		418
D-1	Start-Up and Tuning Information for Digital Servo Systems.....	418
D-1.1	Validating Home Switch, Over Travel Inputs and Motor direction	418
D-1.2	Tuning a Digital Servo Drive.....	421
D-2	Start-Up and Tuning Information for Analog Servo Systems	431
D-2.1	Analog Mode Velocity Interface System Startup Procedures	431
D-2.2	Analog Mode Torque Interface System Startup Procedures	433
D-3	System Troubleshooting Hints (Analog Mode).....	448
Appendix E: Local Logic Execution Time		450
E-1	Local Logic Execution Timing Data.....	450
E-2	Example 1.....	450
E-3	Example 2.....	451
Appendix F: Updating Firmware in the DSM314		458
F-1	Windows Update (for Windows 95/NT/98/2000).....	459
F-2	DOS Update	459
F-3	Restarting an Interrupted Firmware Upgrade.....	460
Appendix G: Strobe Accuracy Calculations		461
G-1	Analog Mode	461
G-2	Digital Mode.....	461
Appendix H: Using VersaPro with the DSM314		466
H-1	Getting Started	466
H-1.1	Starting VersaPro	466
H-2	Starting the Configuration Process	469
H-3	Configuring the DSM314	471
H-4	Connecting to and Storing Your Configuration to the PLC.....	474
H-5	Creating a Motion Program	476

H-5.1	Accessing the Motion Editor Screen.....	476
H-5.2	Saving your Motion Program	478
H-5.3	Storing your Motion Programs and Subroutines to the PLC	478
H-5.4	Printing a Hardcopy of your Motion Programs and Subroutines	478
H-6	Creating a Local Logic Program.....	480
H-6.1	Checking Local Logic Syntax	484
H-6.2	Viewing the Local Logic Variable Table	485
H-7	Creating a Cam Block.....	487

Warnings And Caution Notes as Used in this Publication



Warning

Warning notices are used in this publication to emphasize that hazardous voltages, currents, temperatures, or other conditions that could cause personal injury to exist in this equipment or may be associated with its use.

In situations where inattention could cause either personal injury or damage to equipment, a Warning notice is used.



Caution

Caution notices are used where equipment might be damaged if care is not taken.

Notes: Notes merely call attention to information that is especially significant to understanding and operating the equipment.

These instructions do not purport to cover all details or variations in equipment, nor to provide for every possible contingency to be met during installation, operation, and maintenance. The information is supplied for informational purposes only, and Emerson makes no warranty as to the accuracy of the information included herein. Changes, modifications, and/or improvements to equipment and specifications are made periodically and these changes may or may not be reflected herein. It is understood that Emerson may make changes, modifications, or improvements to the equipment referenced herein or to the document itself at any time. This document is intended for trained personnel familiar with the Emerson products referenced herein.

Emerson may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not provide any license whatsoever to any of these patents.

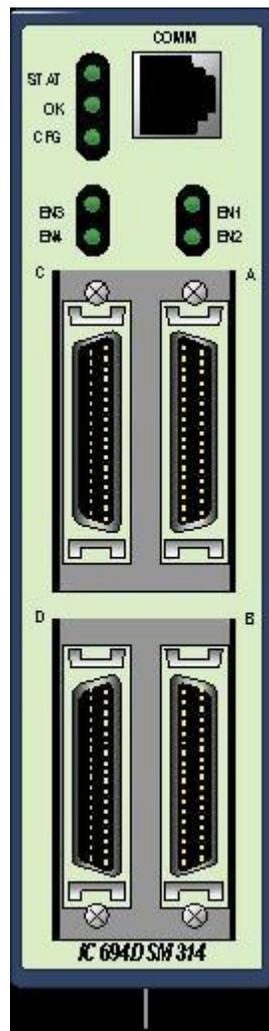
Emerson provides the following document and the information included therein as-is and without warranty of any kind, expressed or implied, including but not limited to any implied statutory warranty of merchantability or fitness for particular purpose.

Chapter 1: Product Overview

The Motion Mate DSM314 is a high performance, easy-to-use, multi-axis motion control module that is highly integrated with the PACSystems RX3i and Series 90-30 host controller logic solving and communications functions.

The two versions of the DSM314, IC693DSM314 and IC694DSM314 are functionally identical.

Figure 1



The DSM314 supports two primary control loop configurations:

- Standard Mode (Follower Control Loop Disabled)
- Follower Mode (Follower Control Loop Enabled)

Servo Types Supported

- Digital – α Series and β Series digital servo amplifiers and motors. These products are documented in Servo Product Specifications Guide, GFH-001.
- Analog – SL Series analog servos and third-party analog velocity command interface and analog torque command interface servos are supported. The SL Series servos are documented in the SL Series Servo User's Manual, GFK-1581.

1.1 Features of the Motion Mate DSM314

1.1.1 High Performance

- Digital Signal Processor (DSP) control of servos
- Block Processing time under 5 milliseconds
- Velocity Feed forward and Position Error Integrator to enhance tracking accuracy
- High resolution of programming units
 - Position: -536,870,912...+536,870,911 User Units
 - Velocity: 1 ... 8,388,607 User Units/sec
 - Acceleration: 1 ... 1,073,741,823 User Units/sec/sec

1.1.2 Easy to Use

- Simple and powerful motion program instruction set
- Simple 1 to 4-axis motion programs. Multi-axis programs using Axes 1 and 2 may utilize a synchronized block start.
- Non-volatile storage for 10 programs and 40 subroutines created with the programming software.
- Compatible with Series 90-30 CPUs equipped with firmware version 10.0 or later (does not work with CPUs 311 – 341 and 351) and PACSystems RX3i CPUs (version 2.8 or later).
- Single point of connection for all programming and configuration tasks, including motion program creation (Motion Programs 1 – 10) and Local Logic programming. All programming and configuration are loaded through the host controller's programming communications port. In turn, the CPU loads all configuration, motion programs, and Local Logic programs to the DSM314 across the host controller backplane.
- User scaling of programming units (User Units) in both Standard and Follower modes.
- DSM314 firmware, stored in flash memory, is updated via the front panel COMM port. Firmware update kits provide firmware and Loader software on floppy disk. Firmware is also available for download on the Emerson web site (<https://www.emerson.com/Industrial-Automation-Controls/support>).

- Recipe programming using command parameters as operands for Acceleration, Velocity, Move, and Dwell Commands
- Automatic Data Transfer between host controller tables and DSM314 without user programming
- Ease of I/O connection with factory cables and terminal blocks
- Electronic CAM capability, starting with Firmware Release (Version) 2.0

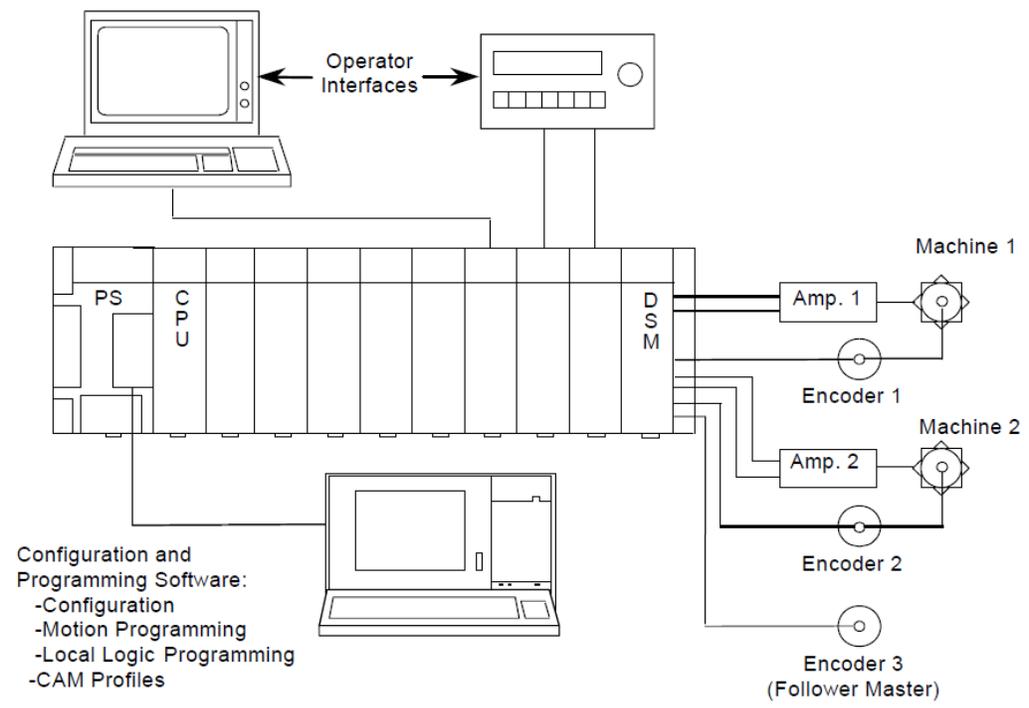
1.1.3 Versatile I/O

- Control of α Series and β Series Digital servos, SL-Series servos, or third-party servos with analog velocity command or analog torque command interface.
- Home and overtravel switch inputs for each Servo Axis
- Two Position Capture Strobe Inputs for each axis can capture axis and/or master position with an accuracy of +/-2 counts plus 10 microseconds of variance.
- 5v, 24v and analog I/O for use by the host controller
- Incremental Quadrature Encoder input on each axis for Encoder/Analog mode
- Quadrature Encoder input for Follower Master axis
- 13-bit Analog Output can be controlled by the host controller or used as Digital Servo Tuning monitor
- High speed digital output (four each 24V and four each 5V) via on-board Local Logic control

1.2 Section 1: Motion System Overview

The DSM314 is an intelligent, fully programmable, motion control option module for the Series 90-30 and PACSystems RX3i control systems. The DSM314 allows a user to combine high performance motion control and Local Logic capabilities with logic solving functions in one integrated system. The figure below illustrates the hardware and software used to set up and operate a servo system. This section briefly discusses each system element to provide an overall understanding of system operation.

Figure 2: Hardware and Software Used to Configure, Program, and Operate a DSM314 Servo System



1.2.1 DSM314 Operation with a Host Controller

The DSM314 and host controller (either PACSystems RX3i or Series 90-30 PLC) operate together as one integrated motion control package. The DSM314 communicates with the host controller through the backplane interface. Every host controller sweep, data such as Commanded Velocity and Actual Position within the DSM314 is transferred to the host controller in %I and %AI data. Also, every host controller sweep, %Q and %AQ data is transferred from the host controller to the DSM314. The %Q and %AQ data is used to control the DSM314. %Q bits perform functions such as initiating motion, aborting motion, and clearing strobe flags. %AQ commands perform functions such as initializing position and loading parameter registers.

Besides the use of %I, %AI, %Q, and %AQ addresses, an additional way to send parameters from the host controller to the DSM314 is with the COMM_REQ ladder program instruction. Details about using the COMM_REQ instruction with the DSM can be found in Appendix B, DSM314 COMM_REQ Instructions

Host Controller Data Latency and DSM314 Latencies

The DSM314 is an intelligent module operating asynchronously to the CPU module. Data is exchanged between the CPU and the DSM314 automatically. For information about the operation of the CPU sweep refer to the following:

- Series 90-30 PLC CPU Instruction Set Reference Manual, GFK-0467M or later
- PACSystems CPU Reference Manual, GFK-2222

Host Controller to DSM Data Transfers

- Host controller-based functions may retrieve DSM status (%I and %AI) information from the DSM data memory asynchronously. The DSM internally refreshes all status data except Actual Velocity at the position loop rate (once every 0.5 to 2ms). Actual Velocity is updated in the DSM data memory every 128 milliseconds. The DSM performs averaging to generate an accurate Actual Velocity reading; therefore, the Actual Velocity reading is not intended for high-speed control purposes.
- The host controller requires approximately 2-4 milliseconds back-plane overhead when reading data (%I and %AI) from and writing data (%Q and %AQ) to DSM internal memory if the DSM is in the CPU rack. The host controller normally reads input data from and writes output data to the DSM once per host controller sweep. In the worst case, the DSM internal data update (which takes 0.5 to 2ms to occur) occurs just after the host controller scan's input update. In this case, the host controller does not read DSM data again until its next scan and any changes in DSM data will be available in the host controller either 4-6ms later or approximately one host controller sweep later, whichever is larger.
- The configuration software automatically selects the lengths of %AI and %AQ data based upon the number of axes configured. A host controller CPU requires time to read and write the data across the backplane with the DSM314. The following manuals document the host controller sweep impact:
 - Series 90-30 PLC Instruction Set Reference Manual, GFK-0467M or later.
 - PACSystems CPU Reference Manual, GFK-2222B or later.

Also refer to the Important Product Information sheet that comes packaged with the DSM module.

- Host controller commands to the DSM (%Q, %AQ) are output to the DSM at the end of the logic solving sweep. The DSM processes the commands within 4 milliseconds after receipt.

Motion Program/CTL Faceplate Inputs

- Delays associated with motion program control or branching via faceplate CTL inputs are equal to a position loop update time interval (0.5 to 2ms) plus the input filter delay (5ms typical for 24 volt CTL inputs or 10 μ s for 5 volt CTL inputs). See tables 1, 2, and 3 for position loop update times.

Local Logic

- Delays associated with Local Logic data updates are based upon the position loop update time interval (see “DSM314 Servo Loop Update Times”) and are not related to the host controller scan. Therefore, Local Logic programs can utilize rapidly changing DSM internal data that cannot be utilized by the host controller CPU due to the host controller to DSM data transfer time and the host controller’s longer scan time.

DSM314 Servo Loop Update Times

When controlling a digital AC servo, the DSM314 uses the loop update times shown in Table 1.

Table 1: Digital Servo Loop Update Times

Motor Current / Torque Loop:	250 microseconds
Motor Velocity Loop:	1 millisecond
Motor Position Loop:	2 milliseconds

When controlling an Analog servo, the DSM314 without Local Logic uses the loop update times shown in Table 2.

Table 2: Analog Servo Loop Update Times without Local Logic

1-Axis Position Loop without Local Logic:	0.5 milliseconds
2-Axes Position Loop without Local Logic:	1 millisecond
3-4 Axes Position Loop without Local Logic:	2 milliseconds

When controlling an Analog servo, the DSM314 with Local Logic uses the following loop update times shown in Table 3. The loop update rates with Local Logic are longer since Axis #4 time slot is used to calculate the Local Logic function.

Table 3: Analog Servo Loop Update Times with Local Logic

1 Axis Position Loop with Local Logic:	1 millisecond
2 –3 Axes Position Loop with Local Logic:	2 milliseconds

Analog Torque mode includes a velocity regulator in addition to the position regulators. For an axis in Analog Torque mode, the velocity regulator is run every 0.5 milliseconds.

DSM314 Position Strobes

Each axis connector on the DSM314 faceplate has two Position Strobe inputs. A rising edge pulse on a Strobe input causes the axis Actual Position to be captured. The position capture resolution is +/- 2 counts with an additional 10 microseconds of variance for the strobe input filter delay. The actual error seen is dependent upon servo acceleration and strobe input filtering/sampling. Consult Appendix G for the exact formulas used to calculate strobe accuracy.

The strobe data is updated within one position loop update interval (0.5 - 2 ms) in the associated Strobe Position %AI data register. The Strobe Position data is also stored in a DSM Parameter Register that can be used as an operand for Motion Program PMOVE and CMOVE commands and in Local Logic. The Strobe Position data update to the host controller is dependent on the host controller sweep time and may take longer than 2 ms.

In Digital mode, these strobes are 5V single-ended/differential inputs (IN1-IN2).

In Analog mode, these strobes are only 5V single-ended (IO5-IO6). In Analog mode only, these strobe inputs are pulled high (as seen in the host controller %I Strobe status bits) if not physically connected to a device.

DSM314 Scan Time Contribution

The tables below list the time that the DSM314 adds to host controller scan time. The scan time contribution is related to (1) the number of DSM314 axes configured, and (2) the type of rack (main, expansion, or remote) the DSM314 is mounted in.

DSM314 Scan Time Contribution (in Milliseconds)

No. of Axes Configured	90-30 CPU364 Rack			90-30CPU374 Rack			RX3i CPU310 Rack		
	Main	Expansion	Remote	Main	Expansion	Remote	Main	Expansion	Remote
1	1.9	2.9	7.9	1.3	2.3	6.9	1.8	2.3	6.9
2	2.5	3.8	11.0	1.9	3.1	10.0	2.5	3.2	10.0
3	3.1	4.7	14.2	2.4	3.9	13.0	3.1	4.2	13.1
4	3.6	5.6	17.3	2.9	4.8	16.0	3.8	5.1	16.2

Note:

1. Be aware that the DSM314's internal Local Logic engine has a maximum scan time of 2ms that is independent of the host controller scan. This allows the user the flexibility to control time critical motion tasks within the Local Logic program. See the applicable chapters in this manual for details on Local Logic programming.
2. (90-30 Feature only) For applications where the above additions to scan rates will affect machine operation, you may need to use the "suspend I/O," "DOIO," and "SNAP" features to transfer necessary data to and from the DSM314 selectively. These features let you avoid transferring all the %I, %Q, %AI, %AQ data every scan, if you do not require it that frequently, which reduces the scan time contribution amount.

Software

The DSM314 requires one of the following configuration/programming software packages:

- Machine Edition Logic Developer – PLC, version 4.5 or later for RX3i
- Machine Edition Logic Developer – PLC, version 2.1 or later, or VersaPro, version 1.1 or later for Series 90-30

The programming/configuration software package is used for the following tasks. The information created by these tasks is sent to the DSM314 over the host controller backplane each time the host controller is powered up.

- Configuration. Allows user to select module settings and default operational parameters.
- Motion program creation. Up to 10 motion programs and 40 subroutines are allowed.
- Local Logic program creation. A Local Logic program runs synchronously with the motion program but is independent of the host controller's CPU scan. This allows the DSM314 to interact quickly with motion I/O signals on its faceplate connectors. This internal response time to motion I/O signals is much faster than would be possible if the logic for these signals was handled in the main ladder program running in the host controller. This is due to (1) the delay in communicating the signals across the backplane and (2) the longer host controller sweep time.
- CAM profile creation. A CAM profile specifies the response of a follower servo to a master position index. CAM profiles are referenced by name in the associated motion program.

Note: *The CAM editor is fully integrated with Logic Developer – PLC.*

Operator Interfaces

Operator interfaces provide a way for the operator to control and monitor the servo system through a control panel or CRT display. These interfaces communicate with the host controller through discrete I/O modules or an intelligent serial communications or network communications module.

Operator data is automatically transferred between the host controller and the DSM314 through %I, %AI, %Q, and %AQ references that are specified when the module is configured. This automatic transfer of data provides a flexible and simple interface to a variety of operator interfaces that can interface to the host controller.

Servo Drive and Machine Interfaces

The servo drive and machine interface are made through a 36-pin connector for each axis. This interface carries the signals that control axis position such as the Pulse Width Modulated (PWM) signals to the amplifier, Digital Serial Encoder Feedback signals or Analog Servo Command and Quadrature Encoder Feedback. Also provided are Home Switch and Axis Overtravel inputs as well as general-purpose host controller inputs and outputs.

Standard cables that connect directly to custom DIN rail or Panel mounted terminal blocks simplify user wiring and are available from Emerson. The terminal blocks provide screw terminal connection points for field wiring to the DSM314 module. For more information concerning the cables and terminal blocks used with the DSM314 module, refer to chapter 3.

1.3 Section 2: Overview of DSM314 Operation

Each DSM314 axis may be operated with the Follower Control Loop enabled or disabled:

Standard Mode (Follower Control Loop Axis Configuration = Disabled)

- In Digital Standard mode, the module provides closed loop position, velocity, and torque control for up to two α or β Series servomotors on Axis 1 and Axis 2. Axis 3 can be used as an Analog Velocity command interface servo axis or an Aux master axis.
- In Analog Standard mode, the module provides closed loop position control for up to four servomotors. Also, based upon the axis configuration, the DSM provides velocity loop control for Analog Torque mode. When the DSM is used with analog velocity interface servos, velocity and torque control loops are closed in the servo amplifier, while the DSM closes the position loop. When the DSM is used with analog torque interface servos, the torque control loop is closed in the servo amplifier, while the DSM closes the velocity and position loops.
- For both digital and analog applications, user programming units can be adjusted by configuring the ratio of User Units and Counts configuration parameters. Jog, Move at Velocity and Execute Motion Program commands allow Standard mode to be used in a wide variety of positioning applications.

Follower Mode (Follower Control Loop Axis Configuration = Enabled)

- In Digital Follower mode, the module provides closed loop position, velocity, and torque control for up to two α or β Series servomotors on Axis 1 and Axis 2. Axis 3 can be used as an Analog Velocity Command Interface servo axis or an Aux master axis.
- In Analog Follower mode, the module provides closed loop position control for up to four servomotors (one or two of the four available axes may instead be used as an Aux master axis). Additionally, based on the axis configuration, the module provides velocity loop control for Analog Torque mode. When the DSM is used with analog velocity interface servos, velocity and torque control loops are closed in the servo amplifier, while the DSM closes the position loop. When the DSM is used with analog

torque interface servos, the torque control loop is closed in the servo amplifier, while the DSM closes the position and velocity loops.

- In both digital and analog applications, the module provides the same features as Standard mode including configurable User Units to Counts ratio.
- In addition, a Master Axis position input can be configured. Each Follower axis tracks the Master Axis input at a programmable (A:B) ratio. Motion caused by Jog, Move at Velocity and Execute Motion Program commands can be combined with follower motion generated by the master axis.
- Follower options include:
 - Master Axis source configurable as Actual or Commanded Position from any other axis
 - Master Source Select %Q bit switches between two Master Axis sources
 - Acceleration Ramp to smoothly accelerate a slave axis until its position and velocity synchronize to the master
 - Separate enable and disable follower trigger sources

Note that Winder mode is not supported in the DSM314. It is supported in the DSM302.

1.3.1 Standard Mode Operation

Figure 3 is a simplified diagram of the Standard mode Position Loop. An internal motion Command Generator provides Commanded Position and Commanded Velocity to the Position Loop. The Position Loop subtracts Actual Position (Position Feedback) from Commanded Position to produce a Position Error. The Position Error value is multiplied by a Position Loop Gain constant to produce the Servo Velocity Command. To reduce Position Error while the servo is moving, Commanded Velocity from the Command Generator is summed as a Velocity Feedforward term into the Servo Velocity Command output.

The following items are included in the data reported by the DSM314 to the host controller:

Commanded Velocity- the instantaneous velocity generated by the DSM314's internal path generator.

Commanded Position- the instantaneous position generated by the DSM314's internal path generator.

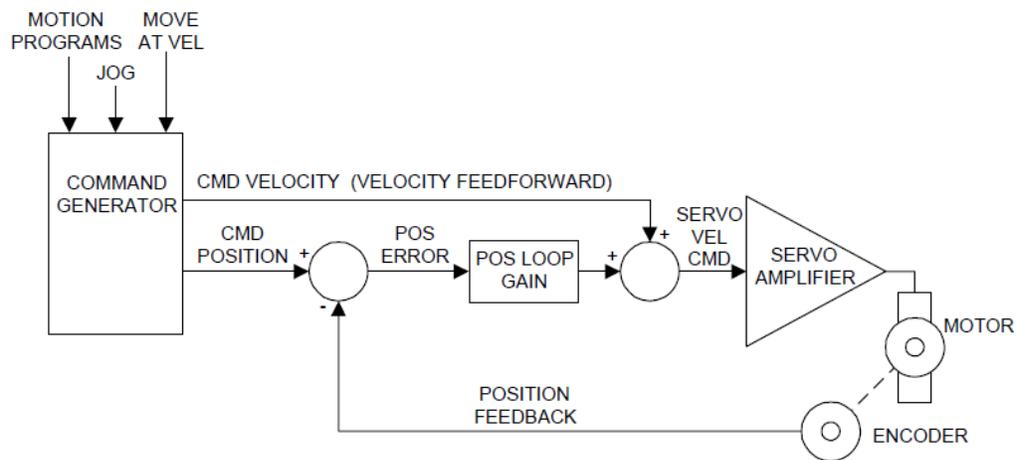
Actual Velocity- the velocity of the axis indicated by the feedback.

Actual Position- the position of the axis indicated by the feedback.

Position Error- the difference between the Commanded Position and the Actual Position.

The DSM314 allows a **Position Loop Time Constant** (in units of 0.1 millisecond) and a **Velocity Feedforward** (in units of 0.01 percent) to be programmed. The Position Loop Time Constant sets the Position Loop Gain and determines the response speed of the closed Position Loop. The **Velocity Feedforward** percentage determines the amount of Commanded Velocity that is summed into the Servo Velocity Command.

Figure 3: Simplified Standard Mode Position Loop with Velocity Feedforward (Analog Velocity Interface)



Follower Mode Operation

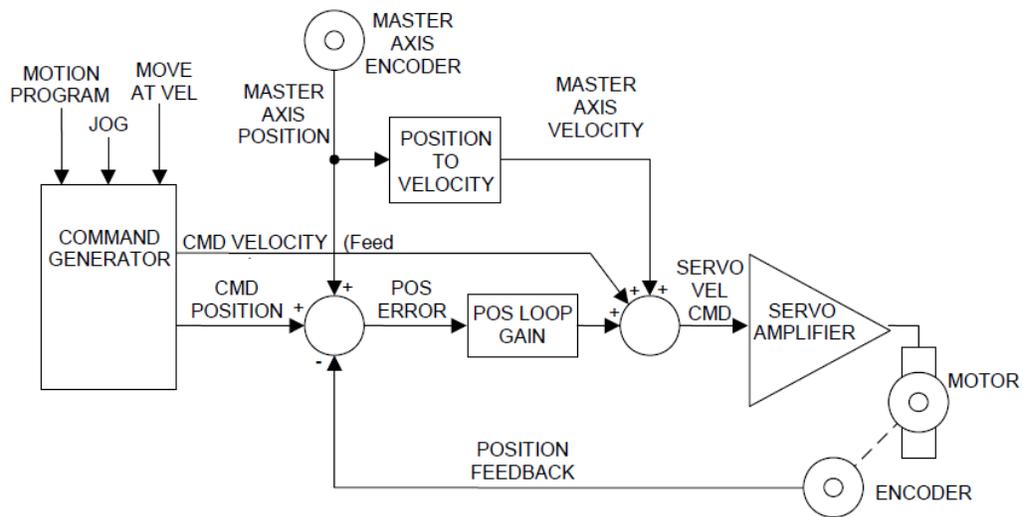
Figure 4 is a simplified diagram of the **Follower** mode Position Loop. It is like the **Standard** mode Position Loop (see previous page) with the addition of a Master Axis input. The Master Axis input is an additional command source producing a Master Axis Position and Master Axis Velocity. Master Axis Position is summed with Commanded Position from the axis Command Generator. Master Axis Velocity is summed with the Commanded Velocity (Velocity Feedforward) output of the axis Command Generator. **Therefore, the servomotor's position and velocity are determined by the sum of the Command Generator output and Master Axis input.** The Command Generator and Master Axis input can operate simultaneously or independently to create Servo Axis motion.

The DSM314 allows several sources for the Master Axis input:

- Axis 1 Commanded Position
- Axis 1 Actual Position (Axis 1 Encoder)
- Axis 2 Commanded Position
- Axis 2 Actual Position (Axis 2 Encoder)
- Axis 3 Commanded Position
- Axis 3 Actual Position (Axis 3 Encoder)
- Axis 4 Commanded Position
- Axis 4 Actual Position (Axis 4 Encoder)

The ratio at which a Servo Axis follows the Master Axis is programmable as the ratio of two integer numbers. For example, a Servo Axis can be programmed to move 125 Position Feedback units for every 25 Master Axis Position units. Each time the Master Axis Position changed by 1 position unit; the Servo Axis would move $(125 / 25) = 5$ Position Feedback units.

Figure 4: Simplified Follower Mode Position Loop with Master Axis Input (Analog Velocity Interface)



1.4 Section 3: α Series Servos (Digital Mode)

The Digital α Series Servo features include:

- World-leading reliability
- Low maintenance, no component drift, no commutator brushes
- All parameters digitally set; no re-tuning required
- Absolute encoder eliminates re-homing (requires optional battery kit)
- An optional motor brake is available
- Optional IP67 environmental rating is also available for most motors
- High resolution 64K count per revolution encoder feedback (incremental or absolute)

The Servo motors, proven on over three million axes installed worldwide, offer the highest reliability and performance. The latest technologies such as high-speed serial encoders and high efficiency Integrated Power Modules (IPMs), further enhance customer benefits.

The servo system is unique in that all the control loops - current, velocity and position - are closed in the motion controller. This approach reduces setup time and delivers significant throughput advantages even in the most challenging applications.

The servo drives are less costly to integrate and maintain. Control circuits are unaffected by temperature changes. There are no personality modules. The servos have a broad application range, that is, a wide load inertia range, flexible acc/dec and position feedback configurations, etc.

Extensive customization features are available to optimize performance and overcome machine limitations. IPM based servo amplifiers require 60% less panel space than conventionally switched amplifiers and produce 30% less heat.

1.4.1 α Series Integrated Digital Amplifier (SVU)

The α Series Integrated Servo Amplifiers (SVU) packages the amplifier with an integral power supply in a stand-alone unit. This unit is the same physical size and footprint as the previous “C” Series of Servo Amplifiers.

The Integrated α Series SVU Amplifiers use the same connections as the “C” Series Amps except that the Emergency Stop circuit uses the internal 24v supply, thus there is no longer a requirement for a 100v power supply.

The heat sinks on the SVU design mount through the panel to keep heat outside the enclosure.

Since the α SVU Amplifiers do not provide regeneration to line capability, discharge resistors may be required. These are available in several sizes.

SVU style α Series Servo Amplifiers are available in five sizes, with peak current limit ratings from 12A to 130A. (Note: Only the 80A and 130A models are currently offered by NA.)

Cables to connect the SVU Amps to the DSM314 and to the motors are available in various lengths.

Refer to publication GFH-001, Servo Product Specification Guide for more information about the α Series servo products.

α Series Servo Motors

The α Series of servomotors incorporate design improvements to provide the best performance possible. Ratings up to 56 Nm are offered. These motors are up to 15% shorter and lighter than the previous S Series of servomotors. New insulation on the windings and an overall sealant coating help protect the motor from the environment.

The standard encoder supplied with the motor is a 64K absolute unit. Holding brakes (90 Vdc) and IP67 sealants are options. The α Series servomotors are approved to conform to international standards for CE (EMC and Low Voltage), IEC and UL/CUL. The following table indicates a sample of the α Series motors available (some α L, α C, α HV, and α M also available).

For more information refer to Chapter 4 of this manual, “Configuring the DSM314,” under the section labeled “Motor Type.” See also, the following publications:

- GFH-001, Servo Products Specification Guide
- GFZ-65142E, α Series AC Motor Descriptions Manual

Table 4: Selected α Series Servo Motor Models

α Model Number	Torque Nm	Output KW	Max. Speed (RPM)
α 1	1	0.3	3000
α 2	2	0.4	2000
α 2	2	0.5	3000
α 3	3	0.9	3000
α 6	6	1.0	2000
α 6	6	1.4	3000
α 12	12	2.1	2000
α 12	12	2.8	3000
α 22	22	3.8	2000
α 22	22	4.4	3000
α 30	30	3.3	1200
α 30	30	4.5	2000
α 30	30	4.8	3000
α 40	38	5.9	2000
α 40/Fan	56	7.3	2000

1.5 Section 4: β Series Servos (Digital Mode)

The Digital β Series Servo features include:

- World leading reliability
- Low maintenance, no component drift, no commutator brushes
- All parameters digitally set; no re-tuning required
- Absolute encoder eliminates re-homing (optional battery kit required)
- Optional motor brake
- High resolution (32K – Beta) (64K – Beta M) count per revolution encoder

The β Series Servos offer the highest reliability and performance. The latest technologies, such as high-speed serial encoders and high efficiency Integrated Power Modules, further enhance the performance of the servo system. Designed with the motion control market in mind, the β Series Servo Drives is ideally suited for the packaging, material handling, converting, and metal fabrication industries.

The servo system is unique in that all the control loops - current, velocity, and position - are closed in the motion controller. This approach reduces setup time and delivers significant throughput advantages even in the most challenging applications.

The servo drives are less costly to integrate and maintain. Control circuits are unaffected by temperature changes. There are no personality modules. The servos have a broad application range including a wide load inertia range, flexible acceleration/deceleration and position feedback configurations. Extensive software customization features are available to optimize performance and overcome machine limitations.

1.5.1 β Series Digital Amplifiers

The β Series servo amplifier integrates a power supply with the switching circuitry. Therefore, can provide a compact amplifier that is 60% smaller than conventional models. In fact, the β Series amplifier has the same height and depth as the RX3i and Series 90-30 modules. This allows efficient panel layout when using the DSM314 motion controller.

The amplifier is designed to conform to international standards.

Emerson offers three communication interfaces for the β Series amplifiers: pulse width modulated (PWM), Servo Serial Bus (FSSB), and I/O Link Interface. Only the pulse width modulated (PWM) interface may be used with the DSM314 module. The PWM interface utilizes the standard servo communication protocol. Position feedback is communicated serially between the DSM controller and the motor mounted serial encoder.

β Series Servo Motors

The β Series Servomotors are built on the superior technology of the α Series servos. They incorporate several design innovations that provide the best possible combination of high performance, low cost, and compact size. Ratings of 0.5 to 12 Nm are offered.

These motors are up to 15% shorter and lighter than comparable servos. New insulation on the windings and an overall sealant coating help protect the motor from the environment.

The β Series motors conform to international standards (IEC). The motor protection level is IP65 (IP67 may be made available through special order).

A (32K – Beta) (64 K – Beta M) absolute encoder is standard with each β Series servo. An optional 90 Vdc holding brake is also available with each model.

For more information, refer to Chapter 4 of this manual, “Configuring the DSM314,” under the section labeled “Motor Type.” See also, the following publications:

- GFH-001, Servo Products Specification Guide
- GFZ-65232E, β Series AC Motor Descriptions Manual

Table 5: Selected β Series Servo Motor Models

β Model Number	Torque ¹ Nm	Output KW	Max. Speed RPM
β 0.5	0.5	0.2	3000
β M0.5	0.65	0.2	5000
β 1	1	0.3	3000
β M1	1.2	0.4	5000
β 2	2	0.5	3000
β 3	3	0.5	3000
β 6	6	0.9	2000
α C12	12	1.4	2000

¹ Indicates continuous, 100% duty cycle

Note: The α C12 motor is listed with the β motors due to similar attributes and amplifier series.

1.6 Section 5: SL Series Servos (Analog Velocity Mode)

The DSM314 supports all models of the SL Series Servos. For details on the SL Series Servo amplifiers, motors, and accessories, please see the SL Series Servo User's Manual, GFK-1581.

Chapter 2: System Overview

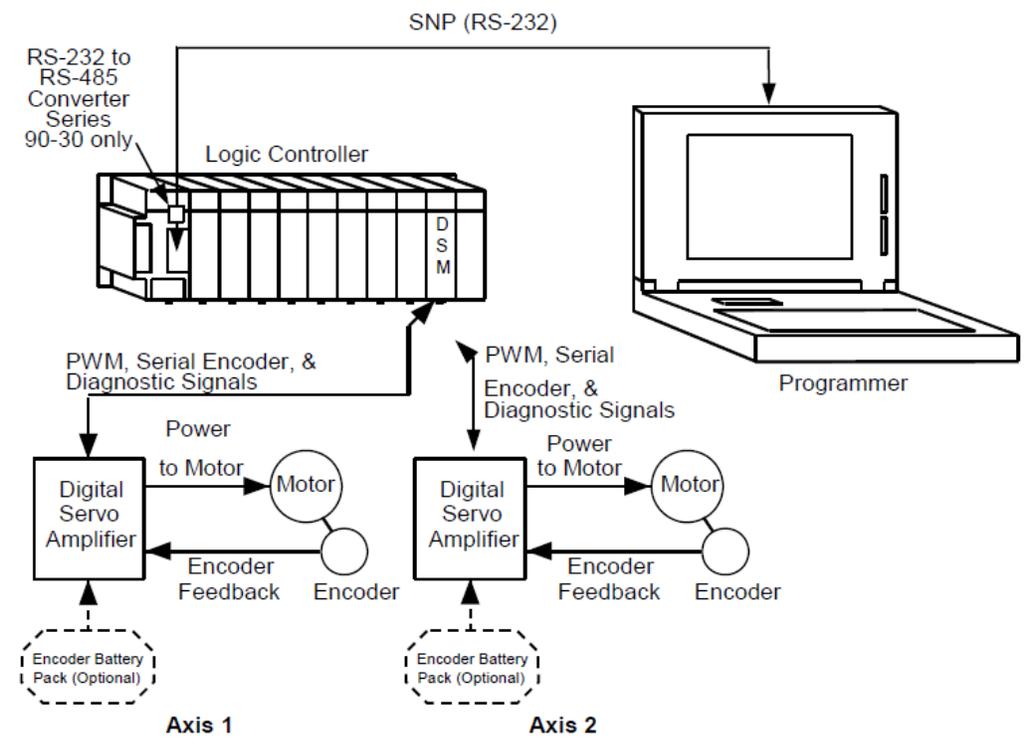
A typical DSM314 motion system includes the DSM314 motion controller, a logic controller (Series 90-30 PLC or PACSystems RX3i), motor(s), servo amplifier(s), I/O, and the Human Machine Interface (HMI).

The DSM314 control system consists of two parts: the servo control and the machine control.

The servo control translates motion commands into signals that are sent to the servo amplifier. It also runs the Local Logic and Motion programs. The servo amplifier receives the control signals from the servo control and amplifies them to the required power level of the motor. The DSM314 provides the servo control.

The machine control/host control (PACSystems RX3i or Series 90-30 PLC) houses the DSM314 module and I/O modules. The machine control executes user defined control logic (but not Local Logic). The machine control and the servo control (DSM314) exchange data over the backplane.

Figure 5: Typical Two-Axis Motion Mate DSM314 Digital Motion Control System



2.1 Unpacking the System

The DSM314, Digital Servo Amplifiers, and Motors are packed separately. This section describes how to unpack the hardware and perform a preliminary check on the components.

2.1.1 Unpacking the DSM314

Carefully unpack the DSM314 and host controller system components. Verify that you have received all the items listed on the bill of material. Keep all documentation and shipping papers that accompanied the DSM314 motion system.

2.1.2 Unpacking the Digital Servo Amplifier

There are two digital amplifier and servo subsystem packages shipped for use with the DSM314, the α Series or the β Series.

The digital servo amplifier is shipped in a double-layered box. Remove the top layer of packing material to uncover the amplifier. Next, carefully remove the inner box from the outer layer. Then lift the amplifier out of the inner box. Retain any loose parts or gasket materials packed with the amplifier. Visually inspect the amplifier for damage during shipment.

Note: Do not change any pre-configured jumpers or switches on the amplifier at this time.

2.1.3 Unpacking the Motor

Motors are packed two different ways, depending on their size. The largest motors are shipped on wooden pallets and are covered with cardboard. Most motors, however, are packed in cardboard boxes.

1. Unpacking Instructions:
 - For those motors packed in boxes, open the box from the top. The motors are packed in two pieces of form-fitted material. Carefully lift the top piece from the box. This should allow sufficient clearance for removing the motor.
 - If the motor is attached to a pallet, remove the cardboard covering. This allows access to the bolts holding the motor to the pallet. Remove the bolts to free the motor from the pallet.
2. Inspect the motor for damage.
3. Confirm that the motor shaft turns by hand.

Note: If the motor was ordered with the optional holding brake, the shaft will not turn until the brake is energized.

Next step. . . . **Assembling the Motion Mate DSM314 System**

2.2 Assembling the Motion Mate DSM314 System

2.2.1 General Guidelines

- Always make sure that the connectors lock into the sockets. The connectors are designed to fit only one way. Do not force them.
- Do not overlook the importance of properly grounding the DSM314 system components, including the DSM314 faceplate shield ground wire. Grounding information is included in this section.

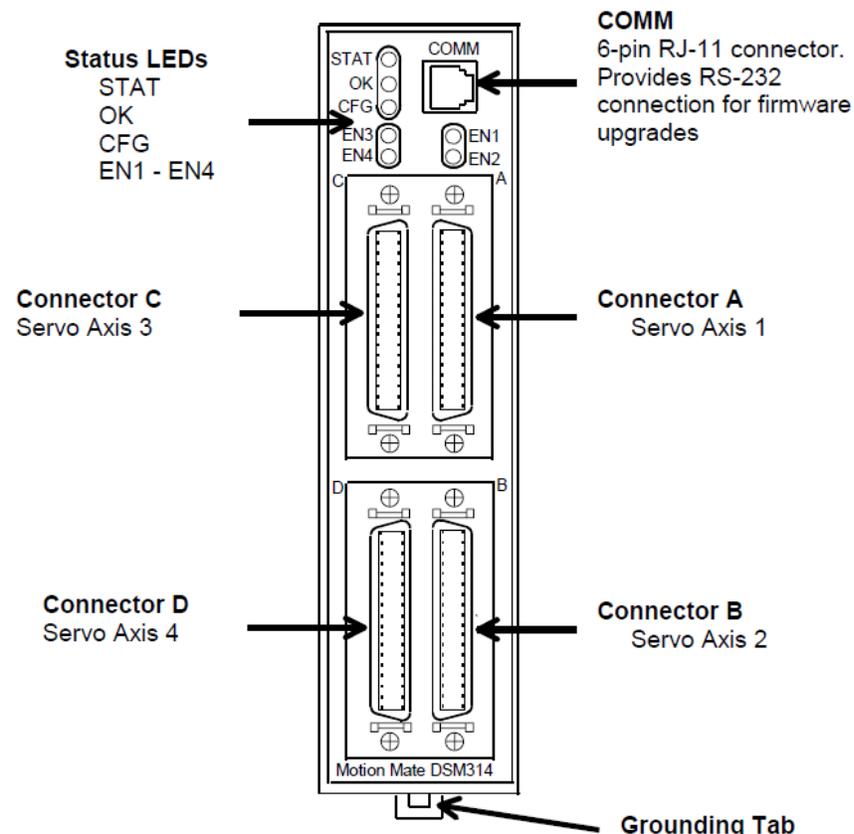
All user connections, except for the grounding tab, are located on the front of the DSM314 module. The grounding tab is located on the bottom of the module. Refer to the figure below.

For instructions about installation of the DSM314 when IEC and other standards must be observed, see Installation Requirements for Conformance to Standards, GFK-1179.

2.2.2 Motion Mate DSM314 Connections

Figure 6 provides an overview of the faceplate and labels on the DSM314 module. For additional information and a complete connection diagram, please refer to chapter 3, Installing and Wiring the Motion Mate DSM314.

Figure 6: Face Plate Connections on the Motion Mate DSM314 Motion Control System



2.2.3 Connecting the α Series SVU Digital Servo Amplifier

Skip to the next section if you are connecting a β Series amplifier.

The α Series Digital Servo Amplifier does not require tuning adjustment during initial startup or when a component is replaced. It also does not need adjustment when environmental conditions change.

To connect the α Series Digital Servo Amplifier, follow the steps outlined below.

1. **Connect the α Series Servo Amplifier to the DSM314.**
 - A. Before connecting the servo command cable, make sure the DSM314 faceplate shield ground wire is connected. This wire is shipped with the DSM314 module and must be connected from the $\frac{1}{4}$ inch blade terminal on the bottom of the module to a suitable panel earth ground.
 - B. The servo command cable contains the pulse width modulated (PWM) output signal from the DSM, the serial data from the motor encoder, and diagnostic signals from the amplifier. The signals carried in this cable are at data communications voltage levels and should be routed away from other conductors, especially high current conductors.
 - C. Locate the servo command cable IC800CBL001 (1 meter) or IC800CBL002 (3 meter). Insert the mating end of this cable into the connector JS1B, located on the Servo Amplifier bottom (see Figure 2-4).
 - D. If you are not using the IC693ACC335 axis terminal board to break out user I/O such as overtravel or home limit inputs, insert the other end of the cable into the connector labeled A, for axis 1, or B for axis 2, on the front of the DSM314. If you are using the terminal board, insert the other end of the cable into the terminal block connector marked SERVO. Next locate the terminal board connection cable IC693CBL324 (1 meter) or IC693CBL325 (3 meter). Insert one end of this cable into the terminal board connector marked DSM. Insert the other end of the cable into the connector labeled A, for servo axis 1, or B for servo axis 2, on the front of the DSM314 module.

Note: Refer to “I/O Connections” in chapter 3 for information concerning the user I/O available for IC693ACC335 terminal block connections.

2. Check SVU Amplifier Channel Switch Settings

Confirm that the Channel Switches (DIP switches), located behind the SVU amplifier door, are set as shown in the following tables. Note that the OFF position is to the left, and the ON position is to the right. Note also, that the switches are numbered from bottom to top (Switch 1 is the bottom switch). For example, in Figure 7, Switches 1, 3, and 4 are shown ON, and switch 2 is shown OFF.

Figure 7: SVU Amplifier Channel Switches

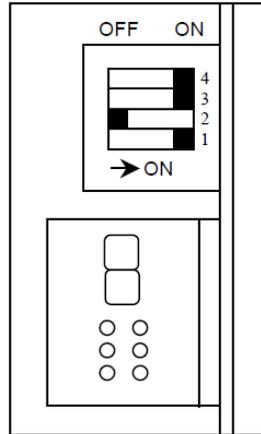


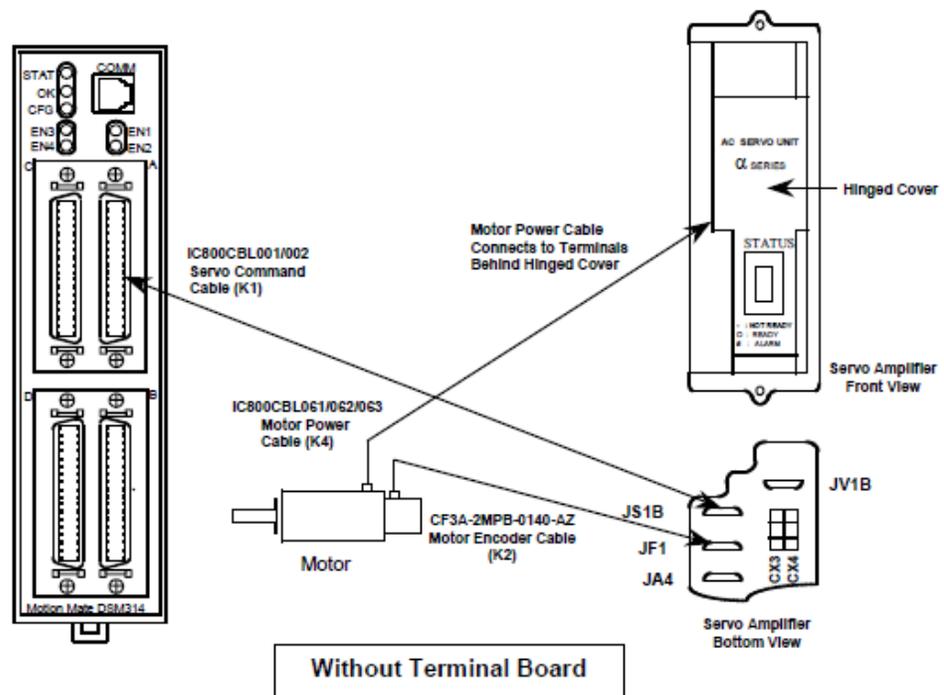
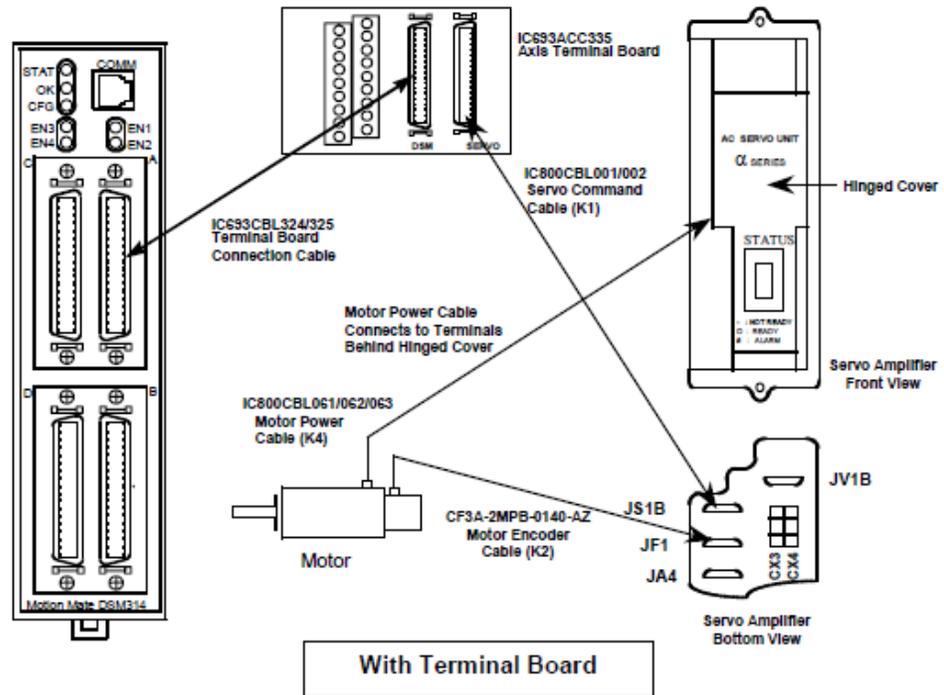
Table 6: SVU Amplifier Channel Switch Settings

Amplifier SVU1-80				
Regenerative Discharge Unit	SW1	SW2	SW3	SW4
Built-in (100 W)	ON	OFF	ON	ON
Separate ZA06B-6089-H500 (200 W)	ON	OFF	ON	OFF
Separate ZA06B-6089-H713 (800 W)	ON	OFF	OFF	OFF

Amplifier SVU1-130				
Regenerative Discharge Unit	SW1	SW2	SW3	SW4
Built-in (400 W)	ON	OFF	ON	ON
Separate ZA06B-6089-H711 (800 W)	ON	OFF	ON	OFF

(To connect additional amplifiers, repeat steps B, C and D above for each additional amplifier.)

Figure 8: Connecting the α Series Digital Servo Amplifier to the Motion Mate DSM314



3. **Connect the Motor Power Cable to the α Series Digital Servo Amplifier.**
 - A. The motor size ordered for your system determines the K4 motor power cable you will use if you ordered prefabricated cables with your system. The following table lists the prefabricated cables commonly specified for each group of motors. A complete listing of α Series servomotor power cables available through Emerson can be found in the Servo Product Specifications Guide, GFH-001.

Table 7: Prefabricated α Servo Motor Power Cable (K4) Part Number Examples

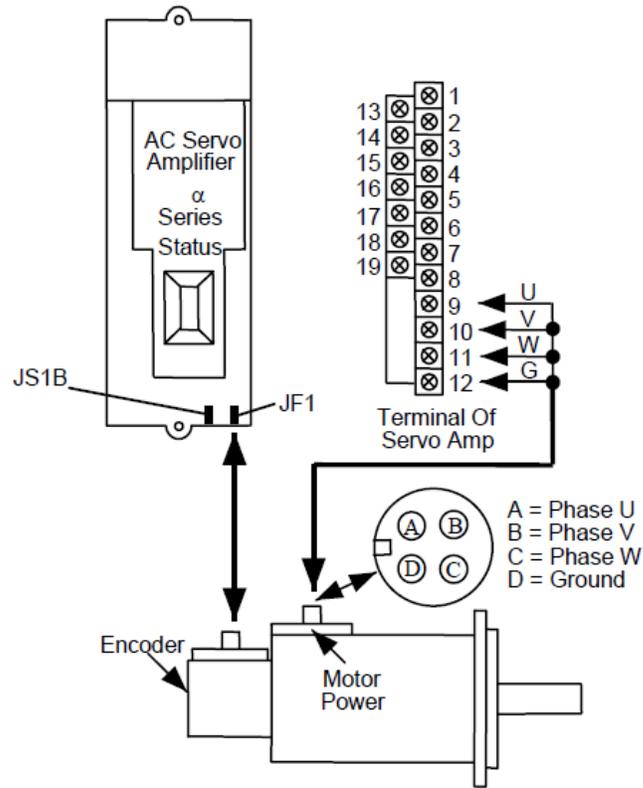
Motor Type	Severe Duty Cable Catalog Number	Cable Description	Cable Length
α 3/3000 α 6/3000	IC800CBL061	Elbow MS Connector	14 Meters
α 12/3000 α 22/2000 α 30/1200	IC800CBL062	Elbow MS Connector	14 Meters
α 30/3000 α 40/2000	IC800CBL063	Elbow MS Connector	14 Meters

- B. One end of this cable has four wires labeled U, V, W, and GND that connect to screw terminals 9–12 on the servo amplifier. Connect these four wires to the terminal strip as shown in Figure 9.
- C. Attach the other end of the cable to the motor after first removing the plastic caps protecting the motor’s connector. Note that this cable is keyed and can only be properly attached to one of the motor’s connection points.

(Repeat this procedure as needed for any other axes in the system.)

For the most current information on the motor power cables or wiring custom motor power cables please refer to the latest version of the α Series Servo Motor Description Manual, GFZ-65142E.

Figure 9: Connecting the Motor to the α Series Servo Amplifier Terminal Strip



4. **Connect the Motor Encoder to the α Series Digital Servo Amplifier.**
 - A. Remove the protective plastic cap from the encoder connector on the motor, and locate the K2 feedback cable CF3A-2MPB-0140-AZ. The cable is configured so that it can only be attached to one connection on the motor.
 - B. Plug the opposite end into the connection labeled JF1 on the bottom of the α Series servo amplifier (see Figure 10).

Repeat this procedure for all axes in the system.

Figure 10: Connecting the α Series Motor Encoder

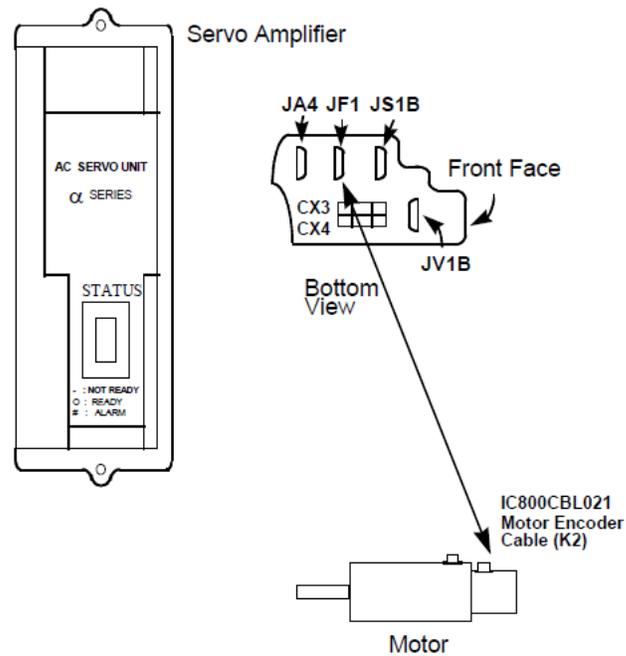


Table 8: Prefabricated α Servo Motor Encoder Cable (K2) for α 3 to α 40 Models

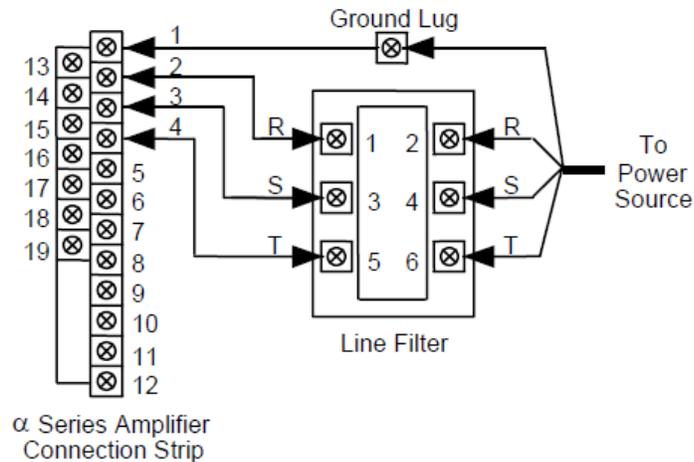
Motor Models	Severe Duty Cable	Cable Length
α 3 to α 40	CF3A-2MPB-0140-AZ	14 meters

Note: Details on α cables can be found in the α Series AC Servo Motor Descriptions Manual, GFZ-65142E, and in the α and β Series Product Specifications Guide, GFH-001.

5. Connect 220-Volt AC 3 Phase Power to the α Series Digital Amplifier

An AC line filter will reduce the effect of harmonic noise to the power supply; its use is recommended. Two or more amplifiers may be connected to one AC line filter if its power capacity has not been exceeded. Figure 11 shows how to connect the amplifier to the line filter.

Figure 11: Connecting the Servo Amplifier to the Line Filter and Power Source



Note: → 220-Volt-AC-three-phase-power-is-required.¶

Note: You must supply the cable for both the connections between the line filter and the servo amplifier, and the connection between the line filter and the power source. Use four-conductor, 600V, 60°C (140°F), UL or CSA approved cable between the line filter and the servo amplifier.

The gauge of wire used for connecting the line filter to the power source must be sized, based on the circuit breaker between the power source and the line filter and the number of servos connected to the line filter.

If a separate isolation transformer is used to supply AC power to the amplifiers, a line filter is not required.

6. Connect the Machine Emergency Stop to the α Series Digital Servo Amplifier

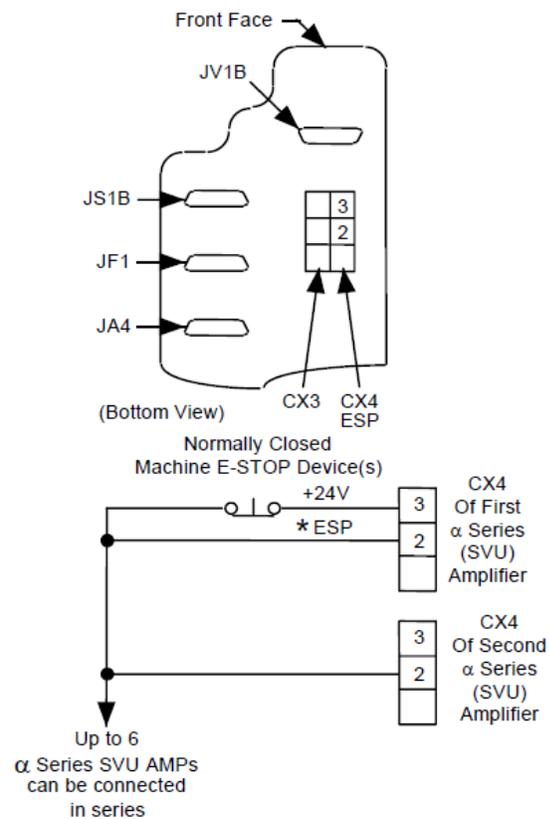
Pin 3 of connector CX4, located on the bottom of the α Series (SVU) amplifier, supplies +24 volts DC for the E-STOP circuit. Route this through the machine E-STOP circuit so that there is +24 volts DC to pin 2 when not in E-STOP. If no E-STOP switch is used this connection must be made with a wire jumper.

Note: You must supply the cable for this connection. Keyed connector plugs marked as connector X and terminal connector pins are included with the amplifier package. You must install this connection as a switch or jumper for the amplifier to operate.

CAUTION

Do not apply any external voltage to this connector.

Figure 12: Connecting Emergency Stop to the α Series Servo Amplifier



For more information, refer to the α Series Servo Amplifier (SVU) Descriptions Manual, GFZ-65192EN.

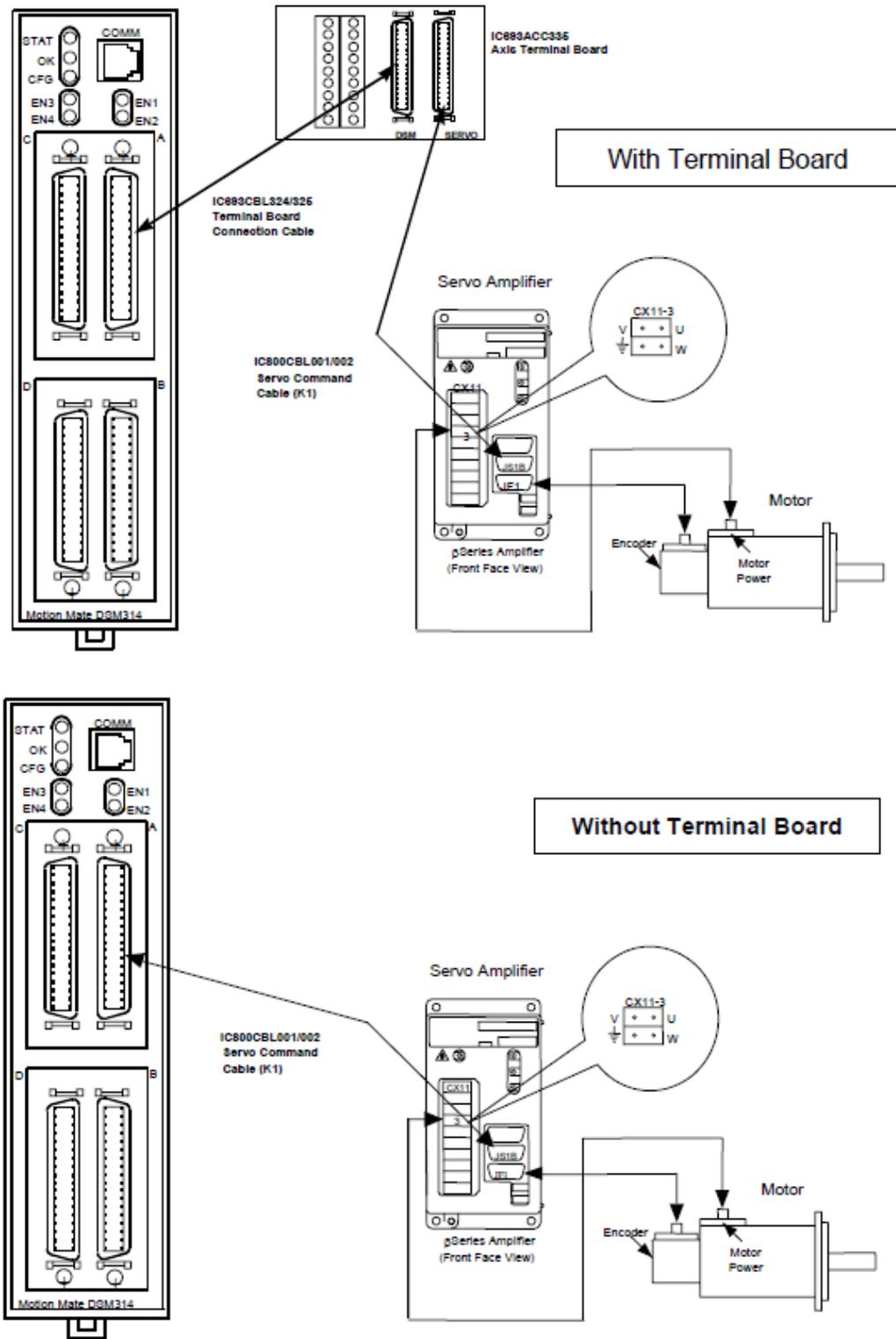
2.2.4 Connecting the β Series SVU Digital Servo Amplifier

The β Series Digital Servo Amplifier does not contain any user adjustments. To connect the β Series Servo Amplifier, follow the steps outlined below. Refer to the previous section for α Series Amplifiers.

1. **Connect the β Series Digital Servo Amplifier to the DSM314**
 - A. Before connecting the servo command cable, make sure the DSM314 faceplate shield ground wire is connected. This wire is shipped with the DSM314 module and must be connected from the ¼ inch blade terminal on the bottom of the module to a suitable panel earth ground.
 - B. The servo command cable contains the pulse width modulated (PWM) output signal of the motion controller, the serial data from the motor encoder, and diagnostic signals from the amplifier. The signals carried in this cable are at data communications voltage levels and should be routed away from other high current conductors.
 - C. Locate the servo command cable IC800CBL001 (1 meter) or IC800CBL002 (3 meter). Insert the mating end of this cable into the connector JS1B, located on the front of the Servo Amplifier (see Figure 13).
 - D. This step depends on whether you are using a terminal board:
 - If you are not using the IC693ACC335 axis terminal board to break out user I/O, such as overtravel or home limit inputs, insert the other end of the cable into the connector labeled A, for servo axis 1, or B for servo axis 2, on the front of the DSM314.
 - If you are using the IC693ACC335 axis terminal board, insert the other end of the cable into the terminal board connector marked SERVO. Next locate the servo command cable IC693CBL324 (1 meter) or IC693CBL325 (3 meter). Insert one end of this cable into the terminal block connector marked DSM. Insert the other end of the cable into the connector labeled A, for servo axis 1, or B for servo axis 2, on the front of the DSM314.

To connect additional amplifiers, repeat steps B - D above for each additional amplifier.

Figure 13: β Series Servo Amplifier Connections



For more information, refer to the connection section of the Servo Product Specification Guide, GFH-001.

2. Connect the Motor Power Cable (K4) to the β Series Digital Servo Amplifier

CAUTION

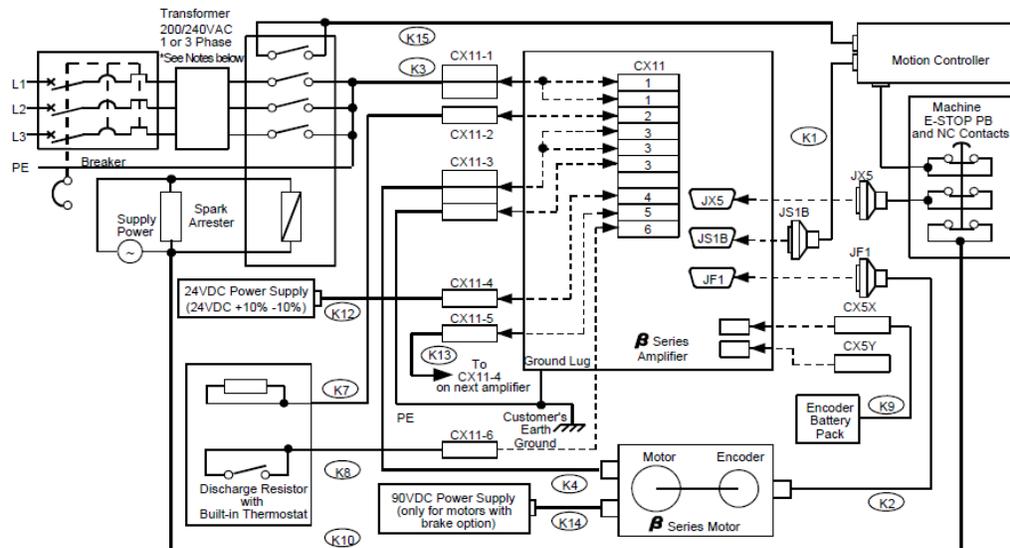
Make connections to the CX-11 connector carefully. This connector is not keyed. Double-check your connections before applying power. Incorrect connections could result in equipment malfunction or damage. Amplifier versions later than revision G have keyed connectors.

- A. The size of the motor ordered for your system determines the motor power cable (K4) you must use. You can choose to purchase prefabricated cables or to build custom cables. Refer to the β Series Control Motor Descriptions Manual, GFZ-65232EN, for information about custom cables or installation for conformance to CE mark. The amplifier end of the prefabricated motor power cable is constructed to connect to terminal block CX11-3 on the amplifier.

Table 9: K4 Cable – β Series Motor Cable Examples

Servo Motor Type	K-4 Motor Cable Part Number	Cable Description
β 0.5/3000	IC800CBL067	14 Meter
β 1/3000, β 2/3000, β 3/3000, and β 6/2000	IC800CBL068	14 Meter
α C12/2000	CF3A-2MPB-0140-AZ	14 Meter
β M 0.5/5000	CP8B-1WPB-0140-AZ	14 Meter
β M 1/5000	CP8B-1WPB-0140-AZ	14 Meter

Figure 14: Connecting the β Series Digital Servo Amplifier Terminal Strip



*Notes:

1. Line filter and lightning surge absorber can be used in place of a transformer when 200-240 volts AC is available to the cabinet.
2. For single-phase operation, AC line phase L3 is not connected. Refer to the Servo System Specifications the Servo Product Specification Guide, GFH-001 for output current de-rating.

- B. Attach the other end of the motor power cable to the motor, after first removing the plastic cap protecting the motor’s connector. Note that this cable is keyed and can only be properly attached to one of the motor’s connection points.
- C. Motor power cables purchased from Emerson include a 1-meter, single conductor wire with a CX11-3 connector on one end and a ring terminal on the other. This cable provides grounding connections for the frame of the motor and should always be connected. Custom cable builders should always include this cable. See the previous connection diagram for proper connection to the amplifier.

(Repeat this procedure as needed for the other axis in the system.)

For more information, please refer to the Servo Product Specification Guide, GFH-001.

3. Connect the Motor Encoder Cable (K2) to the β Series Digital Servo Amplifier

The motor size ordered for your system determines the K4 motor power cable you will use if you ordered prefabricated cables with your system. Please refer to the table below to determine the correct encoder cable catalog number.

- D. Remove the protective plastic cap from the connector on the motor, and locate the encoder cable K2, (see table 10). This cable has two distinct connectors.
- E. Plug the end of the cable with the D-shell style connector into the connection labeled JF1 on the servo amplifier (see Figure 13).
- F. The other end of the cable is configured so that it can only be attached to one connection on the motor encoder (red end cap).
- G. (Repeat this procedure for all axes in the system.)

Table 10: K2 Cable – β Series Encoder Cable Examples

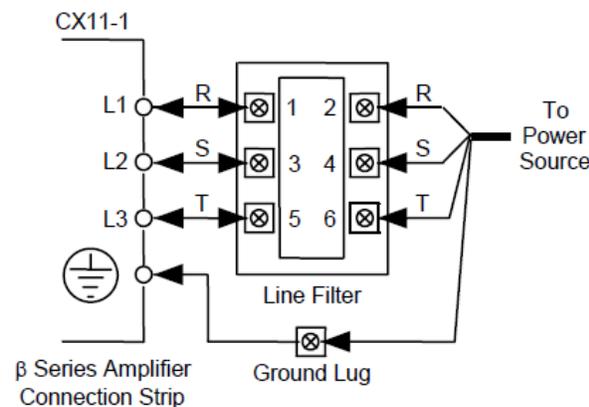
Motor Type	K2 Encoder Cable Part Number	Cable Description
β 0.5/3000	IC800CBL022	14 Meter
β 1/3000, β 2/3000, β 3/3000, and β 6/2000	IC800CBL023	14 Meter
α C12/2000	CF3A-2MPB-0140-AZ	14 Meter
β M 0.5/5000	CFBA-0WPB-0140-AZ	14 Meter
β M 1/5000	CFBA-0WPB-0140-AZ	14 Meter

4. Connect the 220 VAC Power Cable (K3) to the β Series Digital Amplifier

The AC power cable is a user-supplied cable that connects to CX11-1 on the face of the β Series amplifier. The connector for the amplifier end of this cable is part of kit ZA06B-6093-K305 supplied with each amplifier package. See the Servo Product Specification Guide, GFH-001, for more detailed information.

An AC line filter will reduce the harmonic noise effect to the power supply; its use is recommended. A line filter is not needed if an isolation transformer or separate power transformer is used. Two or more amplifiers may be connected to one AC line filter or transformer if its power capacity is not exceeded. Figure 15 shows how to connect the amplifier to the line filter.

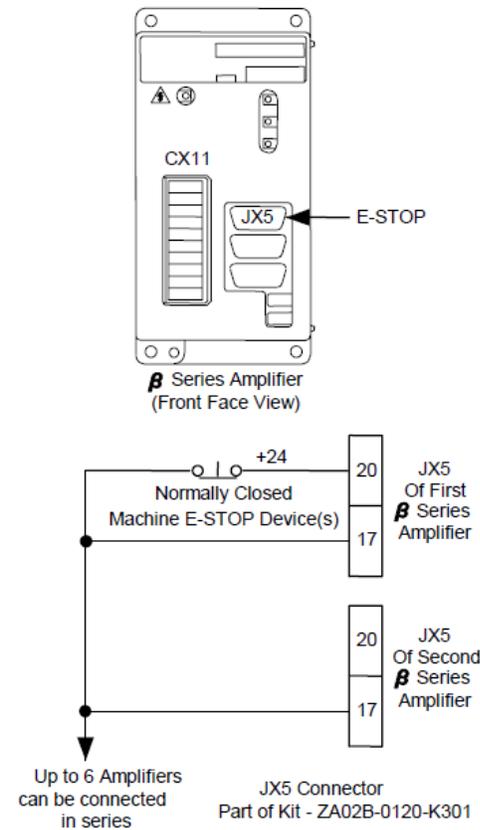
Figure 15: Connecting the β Series Servo Amplifier to the Line Filter and Power Source



Note: You must supply the cable for the connection between the line filter and the power source. Use 4-conductor, 600V, 60°C (140°F), UL or CSA approved cable between the line filter and the servo amplifier. The gauge of wire used for connecting the line filter to the power source must be sized, based on the size of the circuit breaker between the power source and the line filter and the number of servos connected to the line filter. The power connectors and terminals are supplied as part of the amplifier package.

5. Connect the Machine Emergency Stop to the β Series Digital Servo Amplifier

Figure 16: Connecting the E-STOP to the β Series Servo Amplifier



Note: You must supply the cable for this connection package. The JX5 connector and connector cover is included with the amplifier as part number ZA02B-0120-K301. If no E-STOP circuit is required, this connection must be made with a wire jumper or the amplifier will not enable.

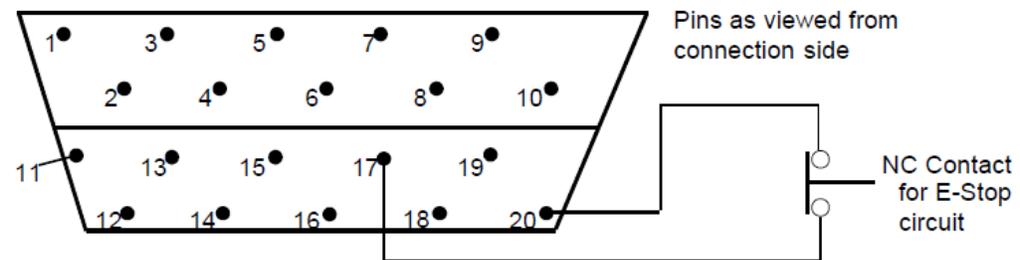
Connector JX5 Pin 20 supplies +24V DC for the E-STOP circuit. Wire Pin 20 through a normally closed contact or switch so there is +24V DC to JX5 Pin 17 when not in E- STOP. Emerson uses two brands of connectors for the JX5 connector. See figure 2-13 for proper connection to each type.

CAUTION

Do not apply any external voltage to this connection.

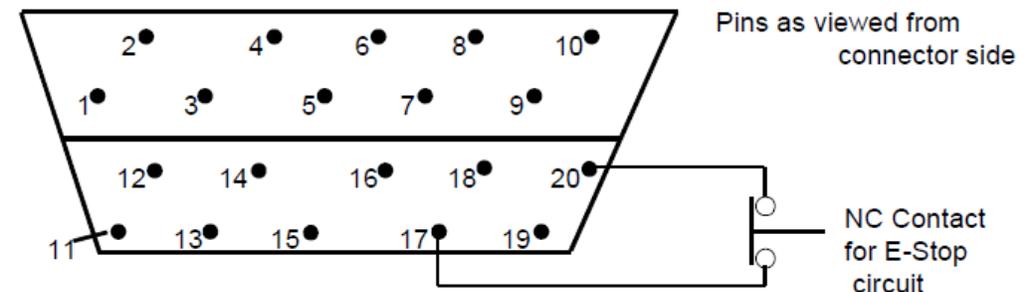
HIROSE 20 Pin PCR Type Connector Pin Configuration

Figure 17



HONDA 20 Pin PCR Type Connector Pin Configuration

Figure 18: 20-Pin PCR Connector Pin-Out



6. Connect 24V DC Cable (K12) to the β Series Digital Servo Amplifier

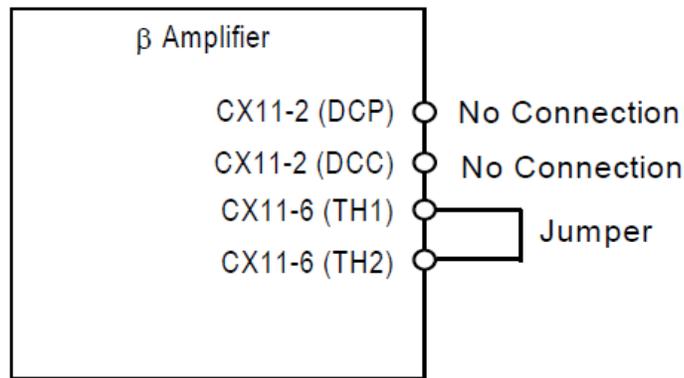
A connector for the external 24 VDC supply is included with the amplifier package as a part of kit ZA06B-6093-K305 and should be connected to CX11-4. The other end of the cable must be connected to a 24VDC source capable of supplying at least 450 milliamps of current for each β Series amplifier. The Emerson IC690PWR024 power supply is recommended. Do not apply power at this time.

7. Connect Cable K8 – Jumper or External Regeneration Resistor to the β Series Digital Servo Amplifier

Without External Regeneration Resistor (Using a Jumper)

If you do not have an external regeneration resistor, you must leave the connections on CX11-2 (DCP and DCC) open. However, you must jumper the CX11-6 (TH1 and TH2) terminals, shown in the figure below. (This jumper completes the circuit that would otherwise be completed by the normally closed thermal over-temperature switch in the external regeneration resistor unit.) If you do not have this jumper installed, the amplifier will not function. The jumper and its connector are included as a part of the connector kit ZA06B-6093-K305 that is shipped with each amplifier package.

Figure 19: Installing a Jumper when an External Regeneration Resistor is not Used



With External Regeneration Resistor

If you have an external regeneration resistor, observe that it has four wires. The two smaller wires (K8) connect to the resistor's internal, normally closed, over-temperature switch. This switch will open and shut down the amplifier if the resistor gets too hot. The two larger wires (K7) connect to the resistor. All connectors needed to connect this resistor unit to the amplifier are provided in the amplifier package.

Connect the two over-temperature switch wires (K8) to CX11-6 terminals TH1 and TH2. (These connections are not polarity sensitive.)

Connect the two resistor wires (K7) to CX11-2, terminals DCP and DCC. (These connections are not polarity sensitive.)

Figure 20: Connecting the External Regeneration Resistor

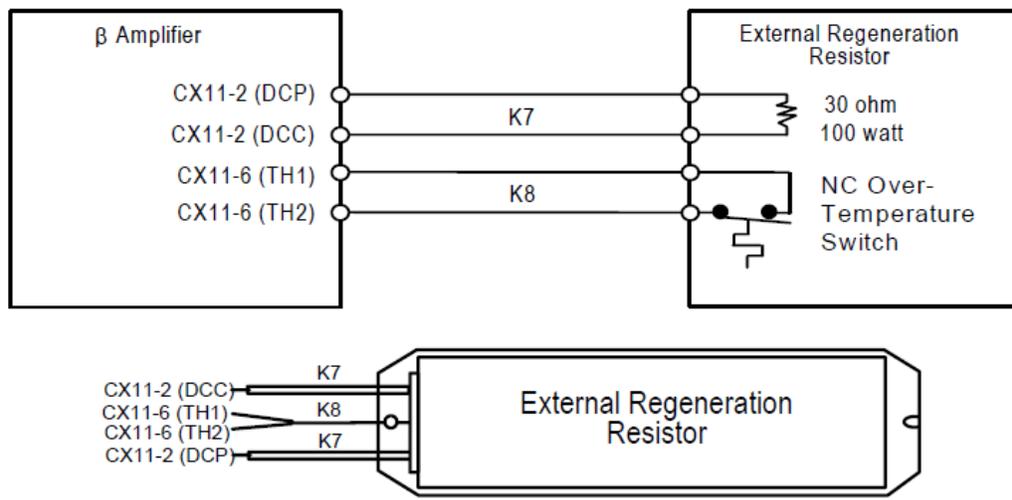
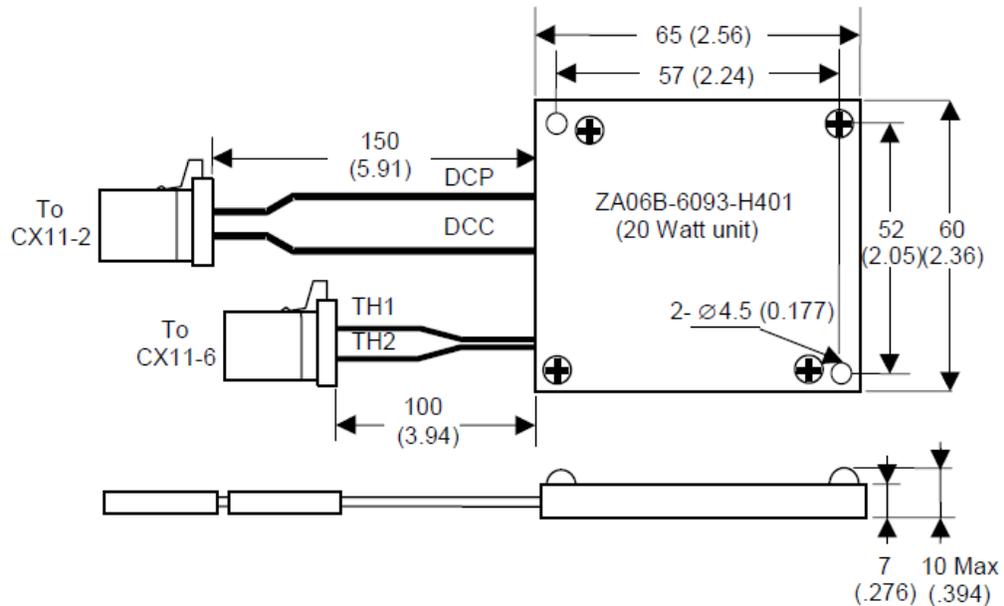


Figure 21: ZA06B-6093-H401 20-Watt Regenerative Resistor



2.2.5

Installing and Wiring the DSM314 for Analog Mode

Important Analog Servo Considerations:

- The Analog Servo Velocity Command or Analog Torque mode output is a single-ended signal on pin 6 of the Auxiliary Terminal Board. This signal is referenced to 0v of the DSM module and host controller. This signal should be connected to the velocity command or torque command input of the servo amplifier.

Note: It is important to correctly configure the DSM for either Analog Torque mode or Analog Velocity mode. Which mode you select depends on the type of servo amplifier in use.

- The DSM314 provides a low current (30 ma) solid state relay output on pin 15 of the Auxiliary Terminal Board for connection to a servo amplifier enable input.
- In analog mode, the DSM314 requires a Drive Ready input (IN_4 signal) on pin 5 of the Auxiliary Terminal Board. **This signal must be switched to 0v when the amplifier is ready to control the servo. The DSM starts checking the Drive Ready input one second after the Drive Enable relay turns on in response to the Enable Drive %Q bit. If the servo amplifier does not provide a suitable output, the IN_4 input to the DSM314 can be connected to 0v or the function can be disabled in the module configuration. For details, refer to chapter 4.**
- Quadrature encoder feedback is used in analog mode. Encoder wiring connections are detailed in figures 49 through 53.
- Figures 49 through 53 are generic analog wiring diagrams for the DSM.
- For details about interfacing the DSM314 to the SL Servo products, refer to the manual, SL Series Servo User's Manual, GFK-1581.

2.2.6 Grounding the Motion Mate DSM314 Motion System

The DSM314 System must be properly grounded. Many problems occur simply because this practice is not followed. To properly ground your Motion Mate DSM314 system, you should follow these guidelines:

- The grounding resistance of the system ground should be 100 ohms or less (class 3 grounding).
- The DSM314 faceplate shield ground wire (shipped with the module) must be connected from the ¼ inch blade terminal at the bottom of the module to a panel frame ground.
- If an axis terminal board is used, two shield (“S”) connections are provided and one of these must be connected to a panel frame ground.
- The system ground cable must have sufficient cross-sectional area to safely carry the accidental current flow into the system ground when an accident such as a short circuit occurs. Typically, it must have at minimum the cross-sectional area of the AC power cable. Figure 22 illustrates the grounding systems.
- The amplifier ground connections, power earth (PE) connections, and motor frame ground connections should always be wired to conform to local electrical wiring regulations. When installing in conformance to CE Mark directives, a grounding bar and clamp(s) (ordered separately) is required for the terminal block to amplifier cable.

Refer to Chapter 3, Installing and Wiring the DSM314, I/O Cable Grounding section, for more details.

Figure 22: Motion Mate DSM314 System Grounding Connections

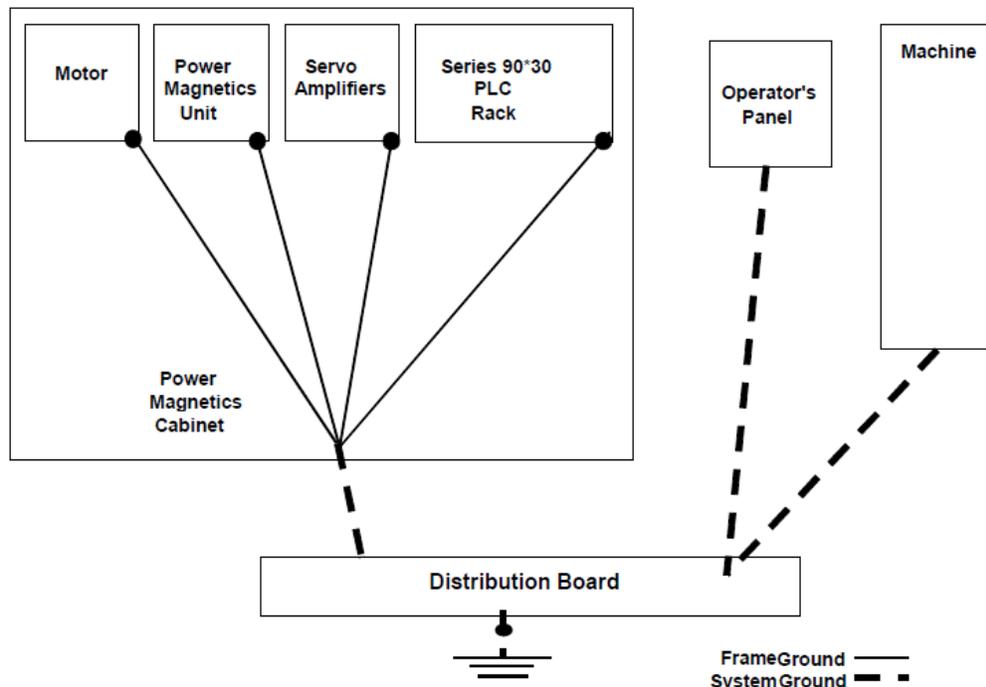


Table 11: Grounding Systems

Grounding System	Description
Frame Ground System	The frame ground system is used for safety and to suppress external and internal noises. In a frame ground system, the frames, cases of the units, panels, and shields for the interface cables between the units are connected.
System Ground System	The system ground system is used to connect the frame ground systems connected between devices or units with the ground.

This completes the steps required to assemble the Motion Mate DSM314 system.

2.3 Turning on the Motion Mate DSM314

Before turning on the power, you should:

- Confirm that the supplied cables are properly attached to the appropriate connectors.
- Confirm that all wiring to the power sources is correct.
- Make sure that the motors are properly secured.
- Check that all components are properly grounded, including the DSM314 faceplate shield.
- If you are using more than one motor, confirm that the servo amplifier connections and the feedback cables are not crossed between motors.

There is a specific sequence for turning on power to the DSM314 Control System. **In the order listed**, perform these steps:

1. Turn on the 220V AC power to the Digital servos. Verify that the charged LED indicator on the amplifier is on.
2. For β Series Digital amplifiers turn on the 24V DC source. Verify that the amplifier Power indicator is on.
3. Switch on the power to the host controller. Check that the PWR LED on the Power Supply is illuminated.
4. Using the correct communication cable, connect a personal computer with the configuration/programming software to the host controller. (For Series 90-30, you can also use a Handheld Programmer – HHP.) For more information, please refer to the appropriate hardware installation manual.
5. Place the host controller in the STOP/Disabled mode.
6. If using an optional motor mounted holding brake, apply applicable power (90 VDC for α and β Series motors, and 24 VDC for S-Series and MTR Series motors) to the brake leads to disengage the holding brake.

2.4 Connecting the Programmer to the Host Controller

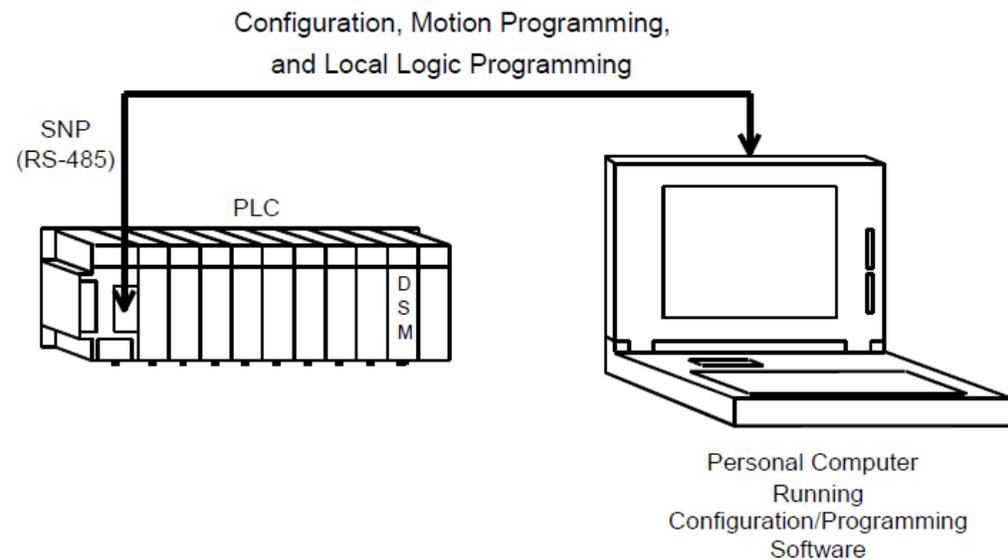
All DSM314 programming is done through the configuration/programming software interface, yielding a single point of programming for the module. For more information, please refer to the Series 90-30 PLC Installation and Hardware Manual (GFK-0356 or later) or the PACSystems RX3i System Manual (GFK-2314 or later). The programming environment has several communications options. One communications option is to connect the programmer directly to the host controller SNP port, as shown in the following figure. Consult the software documentation for additional communications methods.

The DSM314 Controller is configured using the following programming software:

RX3i	Machine Edition version 4.5 or later
Series 90-30	Machine Edition version 2.1 or later VersaPro version 2.1 or later

Note: The DSM314 also has a serial port on the module faceplate. This serial port is used only for updating the DSM314 firmware.

Figure 23: DSM Programmer Connection Diagram

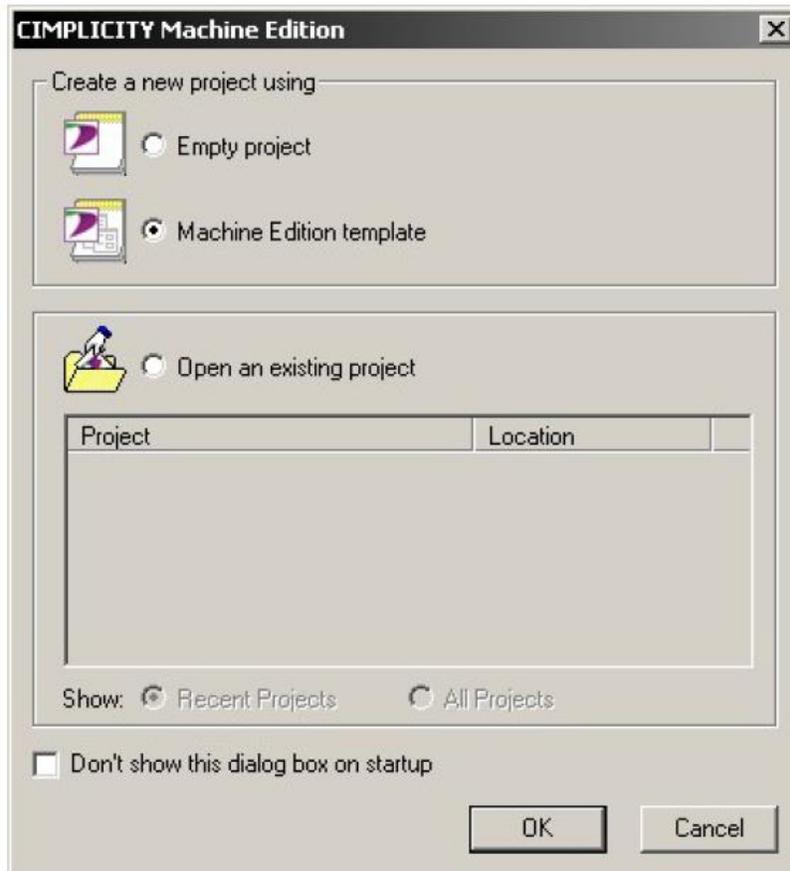


2.5 Machine Edition Configuration

This section describes configuration using Machine Edition software. For VersaPro software, refer to Appendix H.

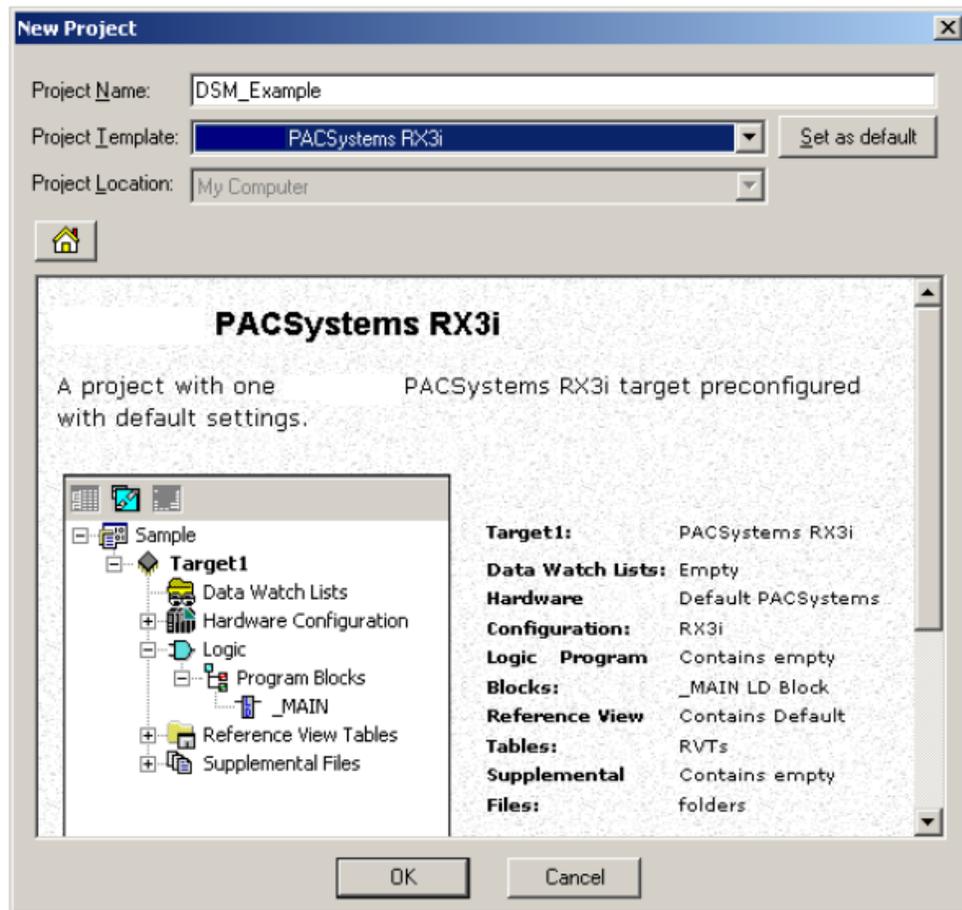
1. Start the Machine Edition Logic Developer – PLC software. The Machine Edition dialog box appears.

Figure 24



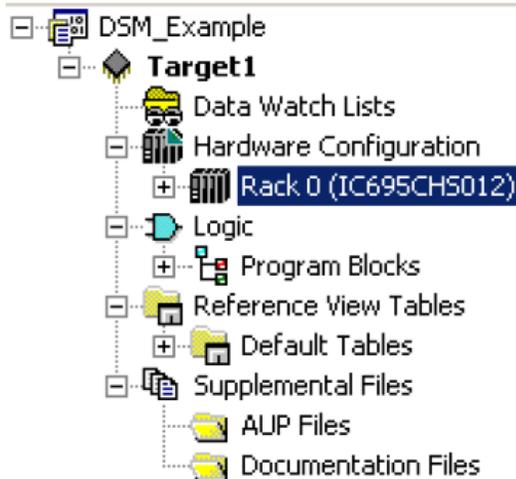
2. Under Create a New Project, choose Machine Edition Template and click OK. The New Project dialog box appears.
3. Type a name for Project Name. In the Project Template dropdown list, select Series 90-30 PLC or PACSystems RX3i. Click OK.

Figure 25



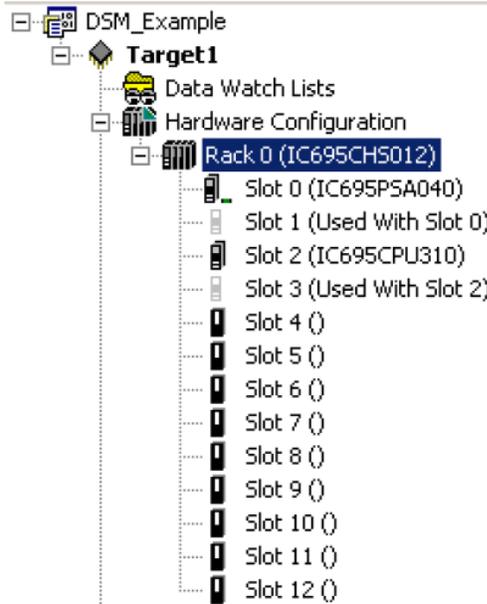
Your project appears in the Navigator window as shown in the following figure.

Figure 26



4. Expand  the Main Rack node, which contains the default power supply and CPU.

Figure 27

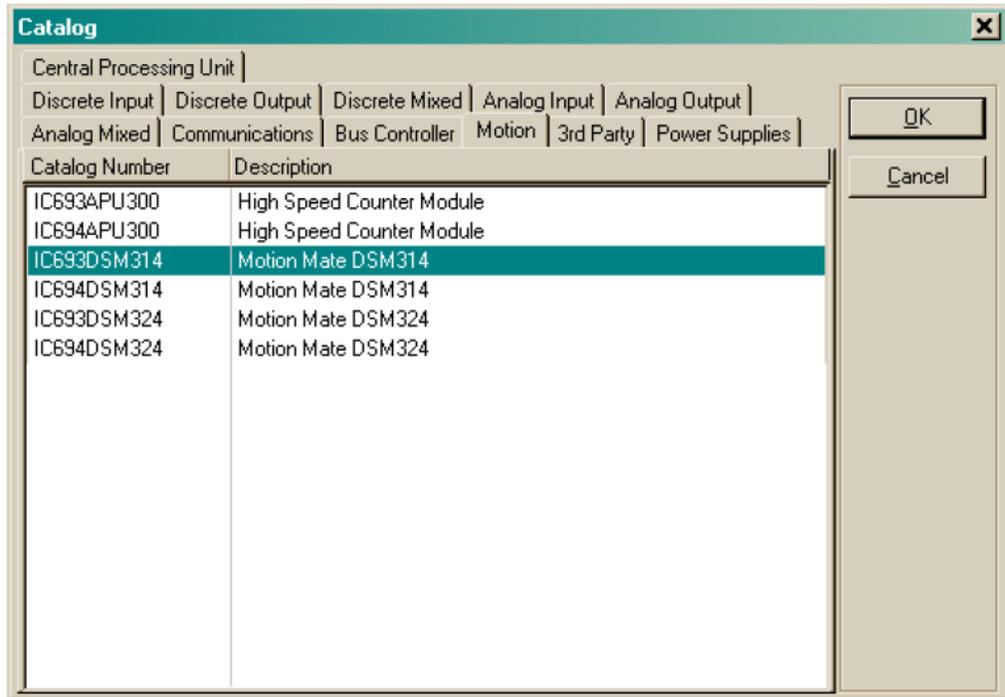


5. If necessary, replace the power supply and/or CPU with the models that will be used in your application. To replace a module, right click and choose Replace Module.
6. Add a DSM314 to the rack configuration.

Note: Because an IC694DSM314 module and an IC693DSM314 module have the same functionality, a Series 90-30 PLC supports them in the same way. If you install an IC694DSM314 in a Series 90-30 PLC, however, you cannot select it in Logic Developer - PLC. You must select an IC693DSM314 module and configure it as if it were an IC694DSM314.

- A. Right click an empty slot and choose Add Module. The Module Catalog dialog box appears.
- B. Select the Motion tab, choose the DSM314 and click OK.

Figure 28



This operation adds the DSM314 to the rack and displays the DSM314 configuration screens that allow you to customize the DSM314 to your particular application. Refer to chapter 4 for details concerning the DSM314 configuration settings.

You should complete the configuration of your host controller to include the Power Supply, Rack, CPU and additional modules to match the target system. Consult the software user's manual, and on-line help as needed.

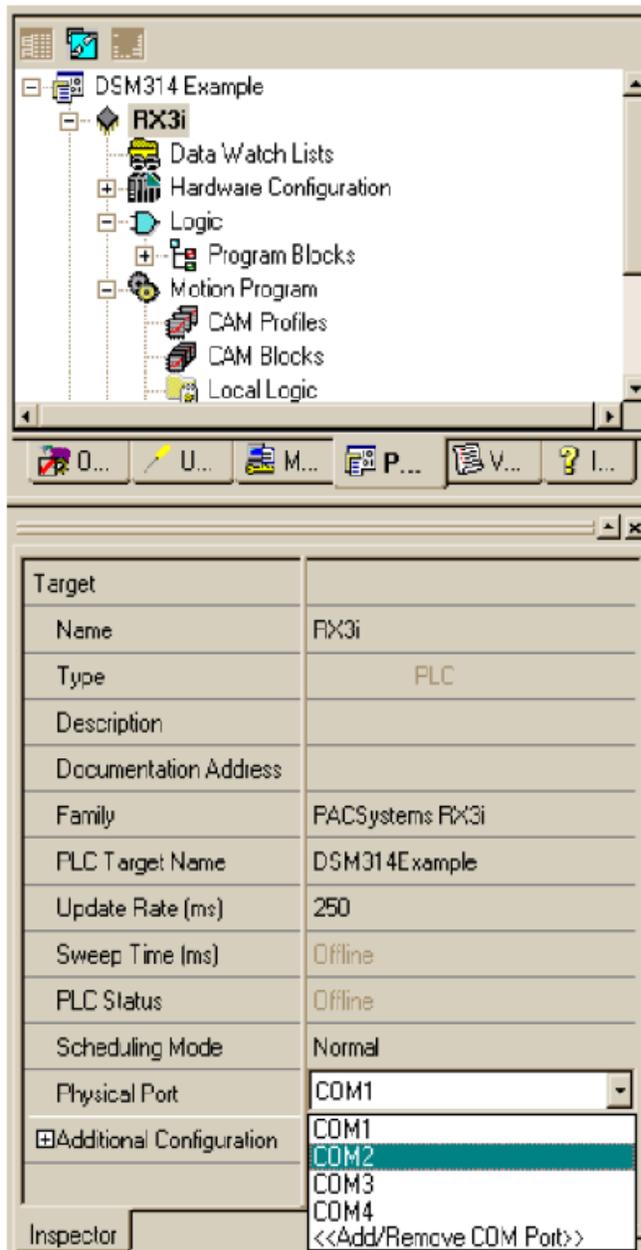
Important

The completed configuration must be stored to the host controller. See "Storing Your Configuration to the Host Controller" on page 44 for instructions on how to do this. For additional details, consult the software user's manual, and on-line help.

2.6 Storing Your Configuration to the Host Controller

To perform the download operation, first make sure that the communications port is properly configured. To access communications setup in Machine Edition software, right click the target you want to connect to in the Navigator window and choose Properties. In the Inspector window, select the Physical Port through which you want to connect. (For information on downloading using VersaPro, see Appendix H.)

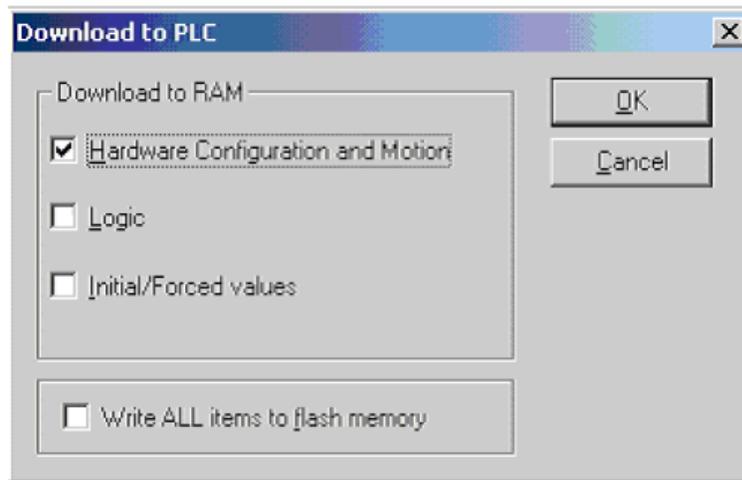
Figure 29: Communications Setup



After configuring the communications port, the local logic program can be downloaded (stored) to the Host Controller CPU. To store the current folder to the Host Controller, choose Target from the Menu Bar and Go Online with “<Target>” from the submenu. Once connected, choose Target from the Menu Bar and Download “<Target>” to PLC from the submenu. The store operation begins the folder transfer process from the programmer to the Host Controller CPU. When you initiate the store operation, a dialog box is presented that allows you to choose what to store to the Host Controller. To store the hardware configuration, select Hardware configuration and Motion.

Note: Local Logic and Motion programs are transferred with the Hardware Configuration.

Figure 30: Machine Edition Download Dialog Box



Machine Edition will indicate any errors or that it has successfully downloaded the program in the Feedback Zone window.

Note:

A host controller status error of “System Configuration Mismatch” with the same rack/slot location as a DSM314 indicates that there is a parameter configured and sent to the DSM314 that has been rejected by the DSM314. Carefully check each parameter of your DSM314 configuration with the configuration settings in this manual for the discrepancy. Correct the discrepancy, clear the host controller Fault, and re-Store the configuration. Check that the error has been corrected. See the next section, Enabling Run Mode on the PLC, for instructions on viewing and clearing PLC faults.

The DSM314 can detect many typical configuration errors. These are returned as error codes of the form Dxxx (hex) in the Module Status Code %AI word or Axis Error Code %AI words. These errors do not cause a host controller status of “System Configuration Mismatch”. Refer to Appendix A for a description of these errors. Correct any configuration errors and restore the configuration with the host controller in Stop mode.

2.7 Alarms

The first step in correcting a problem is to determine if any alarms have occurred. Host controller alarms or errors may be viewed in the PLC fault table. Servo and motion subsystem alarms may be viewed in the DSM314 Module Status Code %AI word or one of the Axis Error Code %AI words. Consult Chapter 5 for additional information on error reporting through the %AI data.

For more information on DSM314 alarms, please refer Appendix A, “Error Codes,” which contains a list of alarm codes and descriptions

For more information on diagnostics, see Appendix D, “Tuning a Digital or Analog Servo System.”

2.8 Configuration Settings

If your system powers up with alarms, it may be due to an incorrect configuration setting.

The configuration must be stored to the host controller CPU and the host controller must be in Run/Output Enabled mode.

If you cannot move an axis or execute a jog, check to see that all conditions necessary to perform these operations are met. Refer to the appropriate sections in this manual.

2.9 Getting Help

For additional information, see <https://www.emerson.com/Industrial-Automation-Controls/support>

- Save the paperwork that came with your system.
The Important Product Information sheet will contain the latest information on this product, some of which may not be included in this manual.
- Back up your ladder logic folder.

Important

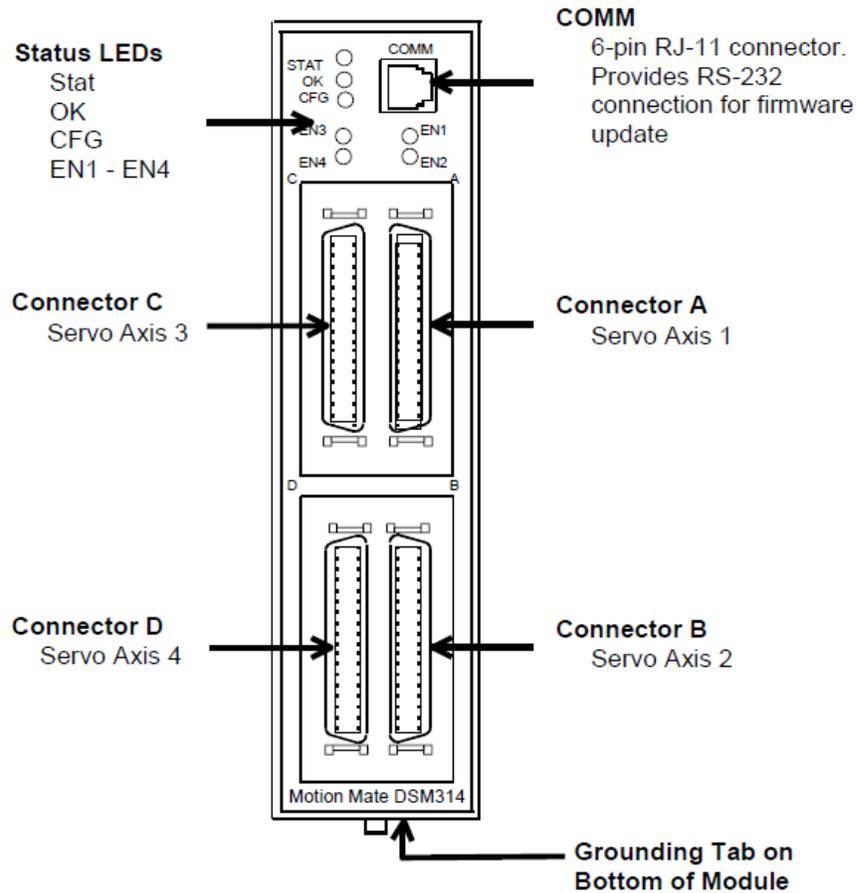
Do this frequently while developing your application.

Chapter 3: Installing and Wiring the DSM314

3.1 Hardware Description

This section identifies the module's major hardware features. The module's faceplate provides seven status LEDs, one communications port RJ-11 connector and four user I/O connectors (36 pin). A grounding tab on the bottom of the module provides a convenient way to connect the module's faceplate shield to a panel ground.

Figure 31: DSM314 Module



3.1.1 LED Indicators

There are seven LED status indicators on the DSM314 module, described below:

STAT Normally ON. FLASHES to provide an indication of operational errors. Flashes slow (four times/second) for Status-Only errors. Flashes fast (eight times/second) for errors that cause the servo to stop.

ON: When the LED is steady ON, the DSM314 is functioning properly. Normally, this LED should always be ON.

OFF: When the LED is OFF, the DSM314 is not functioning. This is the result of a hardware or software malfunction that prevents the module from powering up.

Flashing: When the LED is FLASHING, an error condition is being signaled.

Constant Rate, CFG LED ON:

The LED flashes slow (four times/second) for Status Only errors and fast (eight times/second) for errors that cause the servo to stop. The Module Error Present %I status bit will be ON. An error code (hex format) will be placed in the Module Status Code %AI word or one of the Axis Error Code %AI words.

Constant Rate, CFG LED Flashing:

If the STAT and CFG LEDs both flash **together** at a constant rate, the DSM314 module is in boot mode waiting for a new firmware download. If the STAT and CFG LEDs both flash **alternately** at a constant rate, the DSM314 firmware has detected a software watchdog timeout due to a hardware or software malfunction.

Irregular Rate, CFG LED OFF:

If this occurs immediately at power-up, then a hardware or software malfunction has been detected. The module will blink the STAT LED to display two error numbers separated by a brief delay. The numbers are determined by counting the blinks in both sequences. Record the numbers and contact Emerson for information on

correcting the problem.

OK The OK LED indicates the current status of the DSM314 module.

ON: When the LED is steady ON, the DSM314 is functioning properly. Normally, this LED should always be ON.

OFF: When the LED is OFF, the DSM314 is not functioning. This is the result of a hardware or software malfunction that prevents the module from powering up.

CFG This LED is ON when a module configuration has been received from the host controller.

EN1 When this LED is ON, the Axis 1 Drive Enable relay output is active.

EN2 When this LED is ON, the Axis 2 Drive Enable relay output is active.

- EN3 When this LED is ON, the Axis 3 Drive Enable relay output is active.
- EN4 When this LED is ON, the Axis 4 Drive Enable relay output is active.

3.1.2 The DSM COMM (Serial Communications) Connector

The module’s front panel contains a single RJ-11 connector for serial communications, labeled “COMM”. It is used to download firmware updates to the DSM module from a personal computer running the PC Loader or Win Loader utility software. (See Appendix F for details.)

This serial COMM port connects to the personal computer’s serial port and uses the SNP protocol and the RS-232 serial communications standard. The baud rate is configurable from 300 to 19,200 baud. The COMM port is configured using the configuration software.

A 1-meter cable, IC693CBL316, is available from Emerson to connect the COMM port to a personal computer. This cable uses a 9-pin female D-shell connector for the computer side and an RJ-11 connector for the DSM314. **If a longer cable is used, the maximum recommended length is 50 feet.**

Table 12: DSM314 COMM Port Pin Assignments

RJ-11 Pin Number	9-Pin (female) Number	Signal Name	Description
1	7	CTS	Clear to Send
2	2	TXD	Transmit Data
3	5	0V	Signal Ground
4	5	0V	Signal Ground
5	3	RXD	Receive Data
6	8	RTS	Request to Send

Note: Pin 1 is at the bottom of the connector when viewed from the front of the module.

3.1.3 I/O Connectors

The DSM314 is a two-axis digital servo/one axis analog velocity interface or four axis analog servo (Torque Mode and/or Velocity Mode) controller with four 36-pin I/O connectors labeled A, B, C, and D. The connectors are assigned as follows:

Table 13: Axis I/O Connector Assignments

Connector	Axis Number	Axis Type	I/O Usage
A	1	Servo Axis	Closed Loop Digital or Analog Servo Control
B	2	Servo Axis Aux Axis	Closed Loop Digital or Analog Servo Control or Position Feedback and auxiliary analog / digital I/O
C	3	Servo Axis Aux Axis	Closed Loop Analog Servo Control or Position Feedback and auxiliary analog / digital I/O
D	4	Servo Axis Aux Axis	Closed Loop Analog Servo Control or Position Feedback and auxiliary analog / digital I/O

All four connectors provide similar analog and digital I/O circuits. Only Axis 1 and Axis 2 can be configured to control digital servos. **If digital servos are used, both Axis 1 and Axis 2 must be configured for Digital Servo mode.** When Axis 1 and Axis 2 are configured for digital servos, Axis 3 can be used for Analog Velocity Interface Servo or Aux Axis control. **Axis 4 is not available for Analog Velocity Interface Servo, Torque Interface Servo or Aux Axis control when Axis 1 and 2 are configured for digital servos.**

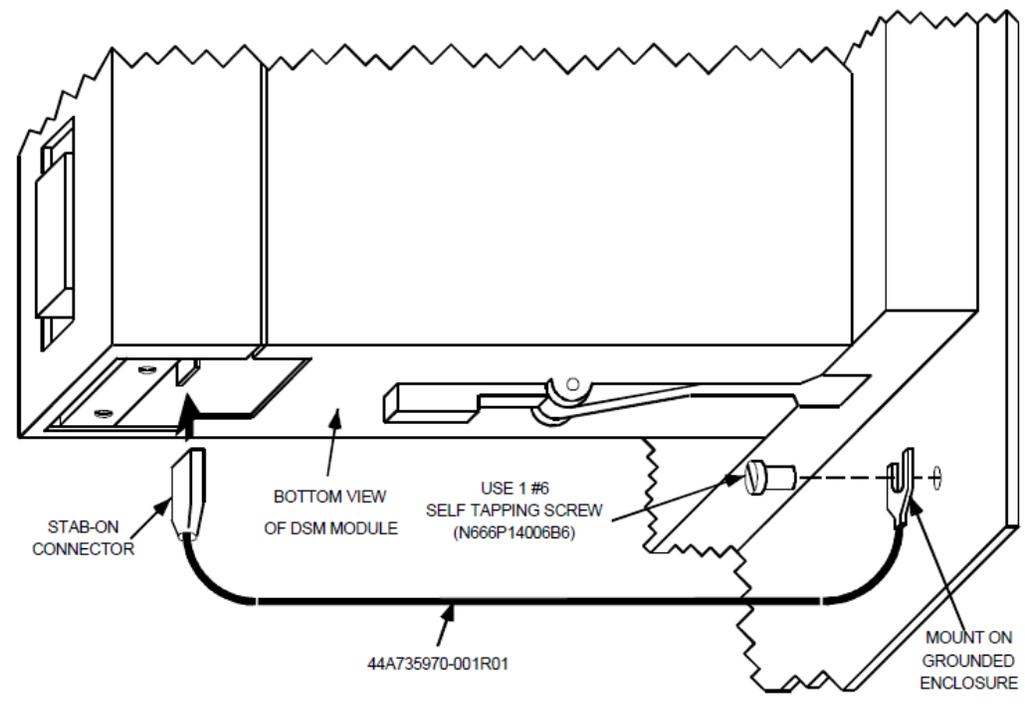
When Axis 1 is configured for Analog Servo control (Torque Interface or Velocity Interface), Axis 2 - Axis 4 are also available for Analog Servo (Torque Interface or Velocity Interface) or Aux Axis control. Aux Axis functions include position input for Follower Master axes and internal (virtual master) command generation.

Any of these four connectors used in a system typically is cabled to an appropriate Terminal Board with cable IC693CBL324 (1 meter) or IC693CBL325 (3 meters). Three different terminal boards provide screw terminals for connecting to external devices. The terminal boards are described later in the “Terminal Board” section of this chapter.

3.1.4 Shield Ground Connection

The DSM314 faceplate shield must be connected to frame ground. This connection from the DSM314 to frame ground can be made using the green ground wire (part number 44A735970-001R01) provided with the module. This wire has a stab-on connector on one end for connection to a ¼ inch terminal on the bottom of the DSM314 module and a terminal on the other end for connection to a grounded enclosure.

Figure 32: Connecting the Shield Ground



3.2 Installing the DSM314 Module

The DSM324i can operate in the main rack, expansion rack, or remote baseplate of any supported Host Controller (PACSystems Rx3i, firmware release 2.8 or later or Series 90-30 PLC, firmware release 10.0 or later). The configuration files created by the configuration software must match the physical configuration of the modules.

For general Series 90-30 installation and environment considerations, refer to the Series 90-30 PLC Installation and Hardware Manual, GFK-0356.

For general PACSystems RX3i installation and environment considerations, refer to the PACSystems RX3i System Manual, GFK-2314 or later.

To install the DSM314 on the baseplate, follow these steps:

1. Use the configuration software or the Hand-Held Programmer (Series 90-30 only) to stop the host controller. This prevents the local application program, if any, from initiating any command that may affect the module operation on subsequent power-up.
2. Power down the host controller system.
3. Align the module with the desired base slot and connector. Tilt the module upward so that the top rear hook of the module engages the slot on the baseplate top edge.
4. Swing the module down until the connectors mate and the lock-lever on the bottom of the module snaps into place engaging the baseplate notch.
5. Connect the faceplate shield wire from the ¼ inch blade terminal on the bottom of the module to a suitable panel earth ground.
6. Refer to Figures 3-10 through 3-23 and Tables 3-7 through 3-14 for I/O wiring requirements.
7. Power up the host controller rack. The Status LED of the Motion Mate DSM314 will turn ON when the controller has passed its power-up diagnostics.
8. Repeat this procedure for each DSM314 module in your host controller system.
9. Configure the DSM314 module(s) as described in Chapter 4.

The following table lists the DSM314 module current draw and defines the number of modules that can be installed in a particular host controller system.

The number of modules in a system may be restricted by:

- Host controller rack power supply capacity
- Host controller I/O Table space. The DSM module requires the use of %I, %Q, %AI, and %AQ memory in the host controller's I/O Table, with the %I and %Q type usually being the most restrictive of the four. %AI is also restrictive on CPUs that do not support configurable %AQ memory (such as the 350 CPU). The amount of available memory varies with the model of host controller CPU to be used.
- Host controller Configuration data storage capacity
- Available CPU memory

The absolute limits for each host controller type must not be exceeded because in some cases they are based on I/O Table and Configuration data capacity.

The practical number of axes must consider I/O use and sweep time of the entire system.

Table 14: Maximum Number of DSM314 Modules per Host Controller System by Rack and Power Supply Types

Power Supply Voltage: Power Supply Current Draw by DSM: Available +5V Current/Module to supply external encoder, if used:	5 VDC from host controller backplane 800 mA plus encoder supply current (see next item). 500 mA (if used, must be added to module +5V current draw)
PACSystems RX3i Main Rack Model 310 CPUs:	5 DSM314 modules in CPU baseplate per PWR 040 Up to 12 DSM314s with two multifunctional DC power supplies (PSD140) in a 16-slot rack
PACSystems RX3i Expansion/Remote Racks (5 and 10-slot expansion or remote baseplates - 8 total baseplates per system)	2 DSM314 modules in remote baseplate with PWR321/322/328 3 DSM314 modules in expansion baseplate with PWR321/322/328 6 DSM314 modules in expansion/remote baseplate with PWR330/331/332
PACSystems RX3i Maximum	61 total DSM314 modules per PACSystems RX3i system. (60 maximums if an Ethernet module, IC695ETM001, is required.)
Series 90-30 Model 350, 352, 360, 363, 364, 366, 367, 374 CPUs: (5 and 10-slot CPU (main) baseplates, 5 and 10-slot expansion or remote baseplates - 8 total baseplates per system)	2 DSM314 modules in CPU baseplate with PWR321/322/328 5 DSM314 modules in CPU baseplate with PW330/331/332 3 DSM314 modules in expansion/remote baseplate with PWR321/322/328 6 DSM314 modules in remote baseplate with PWR330/331/332 7 DSM314 modules in expansion baseplate with PWR330/331/332 20 total DSM314 modules per Series 90-30 system with PWR321/322/328* 20 total DSM314 modules per Series 90-30 system with PWR330/331/332*

* The maximum number of modules supported in a system may be reduced by other modules in the system, such as APM and GBC modules. It may also be further reduced by having datagrams set up that read the reference or fault tables. If the configuration and user program is stored at the same time, the presence of either C blocks within

the LD program, or a C logic program may also affect the number of DSM314 modules that can be included in a system. If the store fails, it may be possible to store the configuration to the system by first storing the logic program, and then storing the configuration on a separate store request.

The numbers listed in the above table are the theoretical maximums. However, an important factor in determining the module mix in any baseplate is that the total power consumption of all modules must not exceed the total load capacity of the power supply. It is possible that a module mix would not allow the maximum number of DSM314s to be installed in a baseplate due to power supply limitations. The configuration software has a power supply usage display that can be used to check this.

This calculation can also be done manually as explained in:

- PACSystems RX3i System Manual, GFK-2314, which also lists load requirement specifications for PACSystems RX3i modules.
- Series 90-30 PLC Installation Manual, GFK-0356P or later, which also lists load requirement specifications for Series 90-30 modules.

The available power supplies are:

PACSystems RX3i Power Supplies

- IC695PSA040 - AC/DC Power Supply - allows 30 watts (6000 ma) for +5 VDC
- IC695PSD040 - 24 VDC input Power Supply - allows 30 watts (6000 ma) for +5 VDC
- IC695PSD140 – AC/DC Multifunctional Power Supply - allows 30 watts (6000 mA) for +5 VDC
- IC695PSA140 - 24VDC input Multifunctional Power Supply - allows 30 watts (6000 mA) for +5 VDC
- IC694PWR321 – AC/DC Serial Expansion Power Supply - allows 15 watts (3000 mA) for +5 VDC
- IC694PWR330 – High Capacity AC/DC Serial Expansion Power Supply - allows 30 watts (6000 mA) for +5 VDC
- IC694PWR331 - High Capacity 24 VDC Serial Expansion Power Supply - allows 30 watts (6000 mA) for +5 VDC

Standard Series 90-30 Power Supplies

- IC693PWR321 - Standard AC/DC Power Supply - allows 15 watts (3000 mA) for +5 VDC
- IC693PWR322 - 24/48 VDC input Power Supply - allows 15 watts (3000 mA) for +5 VDC
- IC693PWR328 - 48 VDC input Power Supply - allows 15 watts (3000 mA) for +5 VDC

High Capacity Series 90-30 Power Supplies

- IC693PWR330 - High Capacity AC/DC Power Supply - allows 30 watts (6000 mA) for +5 VDC
- IC693PWR331 - High Capacity 24 VDC input Power Supply - allows 30 watts (6000 mA) for +5 VDC
- IC693PWR332 – High Capacity 12 VDC input Power Supply - allows 30 watts (6000 mA) for +5 VDC

Note: *If you are installing the ground plate on a painted surface, the paint must be removed where the ground plate is to be mounted to ensure a good ground connection between the plate and mounting surface.*

Note: *Refer to GFK-0867B, (Emerson Product Agency Approvals, Standards, General Specifications), or later version for product standards and general specifications.*

Installation instructions in this manual are provided for installations that do not require special procedures for noisy or hazardous environments. For installations that must conform to more stringent requirements (such as CE Mark), see GFK-1179, Installation Requirements for Conformance to Standards.

3.3 I/O Wiring and Connections

3.3.1 I/O Circuit Types

Each of the module's four connectors (Connector A, B, C, and D) provide the following types of I/O circuits:

- Three differential / single ended 5v inputs (IN1-IN3)
- 5 VDC Encoder Power (P5V)
- One single ended 5v input (IN4)
- Four single ended 5v input / output circuits (IO5-IO8)
- Three 24v inputs (IN9-IN11)
- One 24v, 125 mA solid state relay output (OUT1)
- Two differential 5v line driver outputs (OUT2-OUT3)
- One 24v, 30 mA solid state relay output (OUT4)
- Two differential +/- 10v Analog Inputs (AIN1-AIN2)
- One single ended +/- 10v Analog Output (AOUT1)

Not all of these I/O circuits are available for user connections. Some of the circuits are used to control the digital servo amplifier. Refer to Tables 3-11 through 3-14 for additional information.

3.3.2 Terminal Boards

- Axis Terminal Board, Catalog No. IC693ACC335 – Used in digital mode only. It connects DSM connector A or B to a α or β Digital Servo amplifier. It also provides screw terminal connections for I/O devices. This terminal board contains two 36 pin connectors. One connects to the DSM via cable IC693CBL324/325, and the other connects to the Digital Servo amplifier via the servo command cables IC800CBL001 / 002. See Figures 40, 46, 47, and 48.

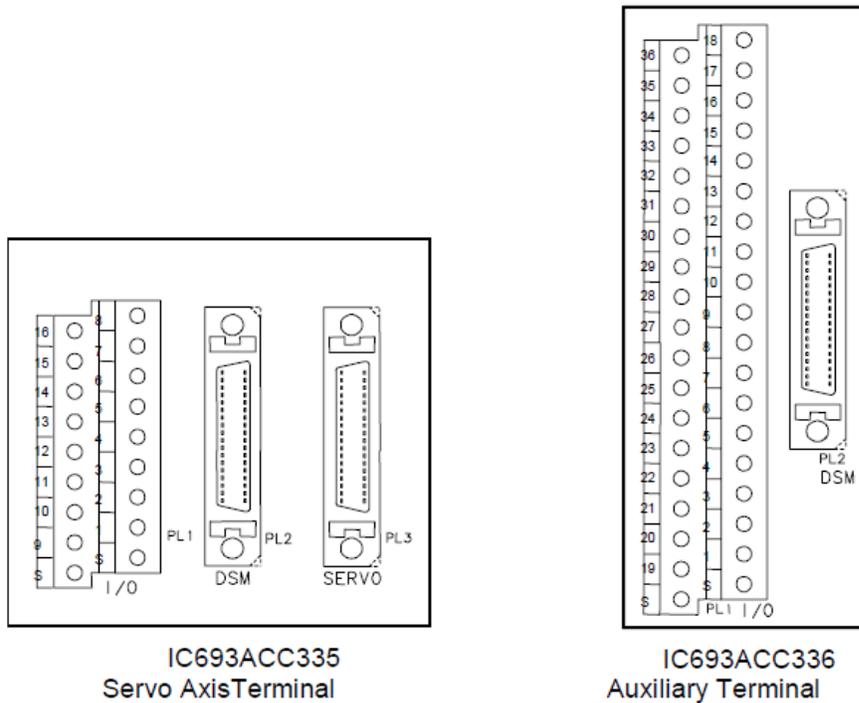
Note: For Digital Servo applications that do not require use of the DSM's A or B connector I/O signals, the DSM connector can be cabled directly to the digital servo amplifier. Refer to Section 3, "I/O Wiring and Connections," later in this chapter for additional information.

- Auxiliary Terminal Board, Catalog No. IC693ACC336 – This terminal board contains a single 36 pin connector that connects to the DSM314 module. This board has two basic applications (see Figures 40 and 41):
 1. For Analog servos, it connects to DSM Connector A, B, C or D to provide screw terminals for wiring to a third-party Analog servo amplifier and I/O devices. See Figures 49 through 53.
 2. For Auxiliary axes, it connects to DSM Connector B, C, or D to provide screw terminals for wiring to external devices such as Strobe sensors, Home switches, and Overtravel Limit switches. Note: See Figure 53.
- SL-Series Servo to APM/DSM Terminal Board, Catalog IC800SLT001 – Used to connect DSM connector A, B, C or D to a SL-Series analog velocity interface servo amplifier, as well as provide screw terminals for wiring to I/O devices. It contains two connectors. One connects to the DSM module, and the other to the SL-Series Servo amplifier. For additional information, please see the SL-Series Servo User's Manual, GFK-1581.

Table 15: DSM Terminal Board Quick Selection Table

DSM Application	DSM Connector	DSM Axis Mode	Terminal Board Required
Connect to α -Series or β -Series digital servo and I/O.	A or B	Digital	IC693ACC335
Connect directly to α -Series or β -Series digital servo. No I/O connections needed.	A or B	Digital	None
Connect to third party analog servo and I/O.	A, B, C or D	Analog	IC693ACC336
Connect to SL-Series analog servo and I/O.	A, B, C or D	Analog Velocity Interface	IC800SLT001
Connect to Auxiliary Axis I/O on DSM connector B, C or D or S2K Series analog servo and I/O.	B, C or D	Analog or Aux	IC693ACC336

Figure 33: Axis and Auxiliary Terminal Board Assemblies



Note: Each terminal board is shipped with DIN Rail mounting feet. Instructions for converting a terminal board to panel-mount are included in this chapter.

3.3.3 Digital Servo Axis Terminal Board - IC693ACC335

Description

The IC693ACC335 Digital Servo Axis Terminal Board is used to connect the DSM314 to Digital Servo Amplifiers. The board contains two 36-pin connectors, labeled **DSM** and **SERVO**. A cable IC693CBL324 (1 meter) or IC693CBL325 (3 meters) connects from **DSM** connector (PL2) to the DSM314 faceplate connector A or B. A Servo Command Cable IC800CBL001 (1 meter) or IC800CBL002 (3 meters) connects from the **SERVO** connector (PL3) to the JS1B connector on a α Series or β Series Digital Servo Amplifier.

Eighteen screw terminals are provided on the Digital Servo Axis Terminal Board for connections to user devices. These terminals have the following assignments:

Table 16: IC693ACC335 Digital Axis Terminal Board Pin Assignments

Axis Terminal Board I/O Screw Terminal	DSM314 Faceplate Pin	Circuit Identifier	Circuit Type	Servo Axis 1, 2 Circuit Function	Signal Name (Axis 1 listed) *	Maximum Voltage
1 9	1 19	IN1	Single ended /differential 5v inputs	Strobe Input 1 (+) Strobe Input 1 (-)	IN1P_A IN1M_A	5 VDC
2 10	2 20	IN2		Strobe Input 2 (+) Strobe Input 2 (-)	IN2P_A IN2M_A	5 VDC
3	4	P5V	5v Power	5v Power	P5V_A	5 VDC
11	22	0V	0v	0v	0V_A	5 VDC
6	16	IN9	24v optically isolated inputs	Overtravel (+)	IN9_A	30 VDC
14	34	IN10		Overtravel (-)	IN10_A	30 VDC
7	17	IN11		Home Switch	IN11_A	30 VDC
15	35	INCOM	24v Input Common	24v Input Common	INCOM_A	30 VDC
8 16	18 36	OUT1	24 v, 125 mA DC SSR output	Host controller 24v Output (+) Host controller 24v Output (-)	OUT1P_A OUT1M_A	30 VDC
5 13	14 32	OUT3	Differential 5v output	Host controller 5v Output (+) Host controller 5v Output (-)	OUT3P_A OUT3M_A	5 VDC
4	6	AOUT	+/- 10v Analog Out	Host controller Analog Out	AOUT_A	5 VDC
12	24	ACOM	Analog Out Com	Analog Out Com	ACOM_A	5 VDC
S (2 pins)		SHIELD	Cable Shield	Cable Shield	SHIELD_A	5 VDC

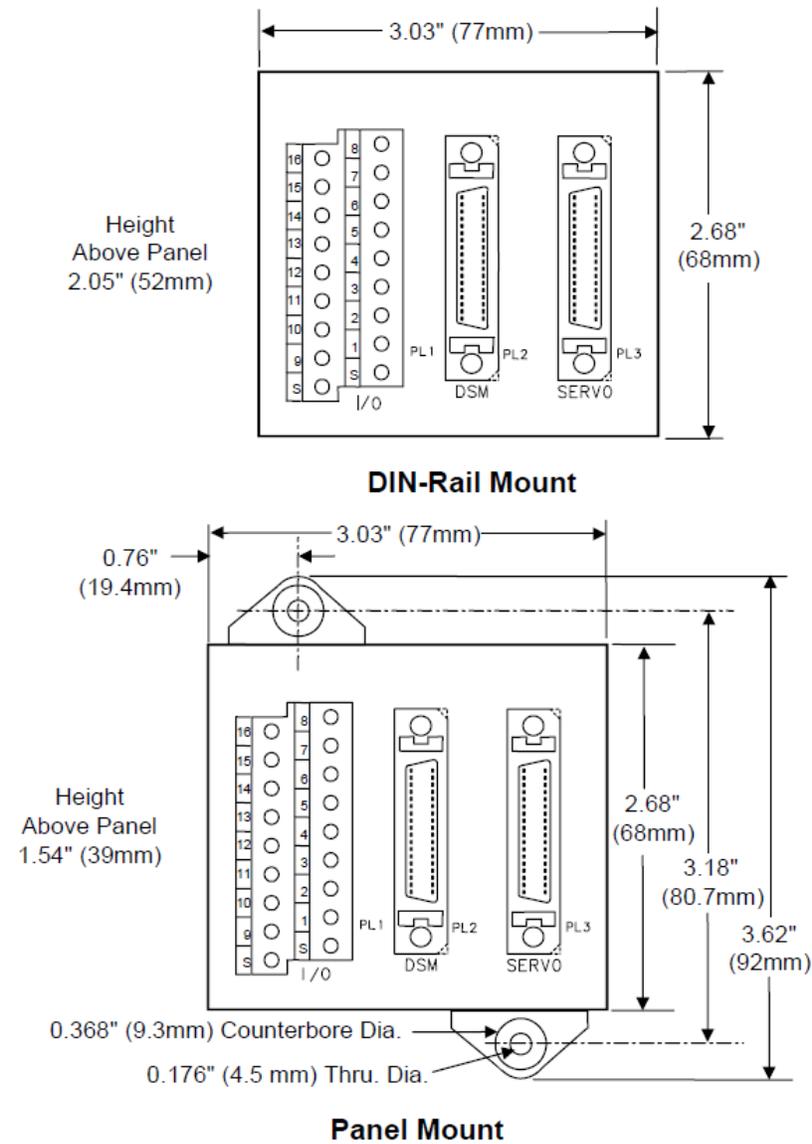
* For signal names pertaining to servo axis 2, change all “_A” to “_B”.

Six 130V MOVs are installed between selected I/O points and the shield (frame ground) for noise suppression. The I/O terminal points so connected are 6, 7, 8, 14, 15, and 16. The I/O terminals support a wire gauge of 14-28 AWG. Maximum screw torque that may be applied is 5 inch-pounds.

Note: Two of the screw terminals are labeled S for Shield. A short earth ground wire should be connected from one of the S terminals directly to a panel earth ground. The cable shields for any shielded cables from user devices should connect to either of the S terminals.

Mounting Dimensions

Figure 34: IC693ACC335 Digital Axis Terminal Board Mounting Dimensions



Converting From DIN-Rail Mounting to Panel Mounting

The following parts are used in either the DIN-rail or Panel mount assembly options. The axis terminal board is shipped configured for DIN-rail mounting. The instructions in this section guide you in converting the board to its panel mounting optional configuration.

The following table and drawings describe the various plastic parts that make up the axis terminal board assembly and shows a side view of the board configured for DIN-rail mounting

Table 17: Axis Terminal Board Assembly Components

Plastic Component Part Number	Description	Quantity	Mounting Styles Used With
UMK-BE 45	Base Element	1	DIN, Panel
UMK-SE 11.25-1	Side Element	2	DIN, Panel
UMK-FE	Foot Element	2	DIN
UMK-BF*	Mounting Ear	2	Panel

* Parts shipped with axis terminal board for optional panel mounting

Figure 35: Digital Servo Axis Terminal Board Assembly Drawings

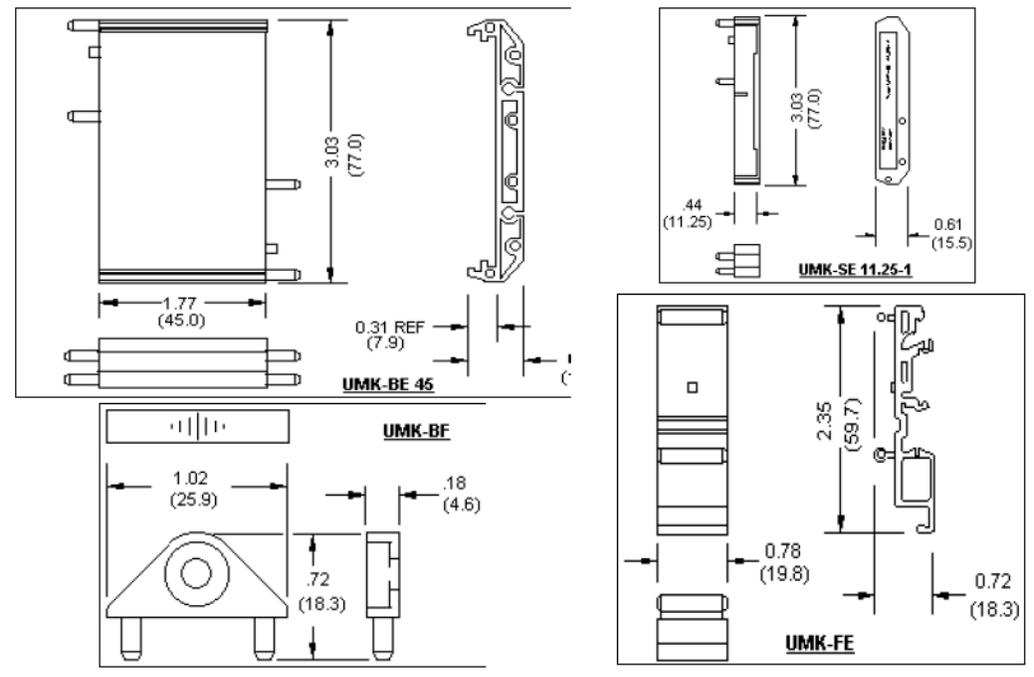
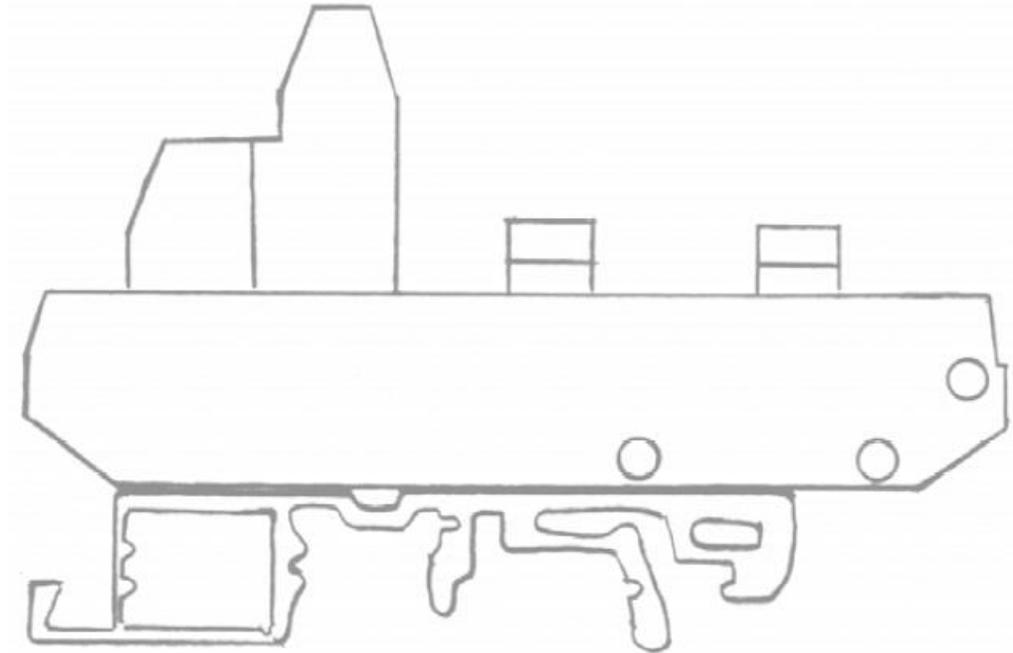


Figure 36: Digital Servo Axis Terminal Board Assembly Side View



The following procedure should be used to convert the Digital Servo axis terminal board to its panel mounting form. Remember to save all removed parts for possible later conversion back to DIN-rail mounting.

1. Carefully remove one UMK-SE 11.25-1 side element from the UMK-BE 45 base element. If a screwdriver or other device is used, exercise extreme caution to avoid damaging either the plastic parts or the circuit board.
2. Slide the UMK-FE foot element off the base element. Save this part for possible future use in converting the terminal board back to its DIN-rail mounting configuration.
3. Snap the side element, removed in step 1 above, back into the base element.
4. Insert one UMK-BF mounting ear into the appropriate two holes in the side element. Note that the mounting ear has a recessed hole for later inserting a (user supplied) mounting screw. The recessed hole should face upwards to accommodate the mounting screw.
5. Repeat steps 1-4 above for the other side of the terminal board.

3.3.4 Auxiliary Terminal Board - IC693ACC336

Description and Mounting Dimensions

The IC693ACC336 Auxiliary Terminal Board is used to connect the DSM314 to Analog Servo Axes and auxiliary devices such as Incremental Quadrature Encoders, Strobe detectors and external switches. The board contains one 36 pin connector, labeled DSM. A cable IC693CBL324 (1 meter) or IC693CBL325 (3 meters) connects from the DSM connector (PL2) to the DSM314 faceplate.

Thirty-eight screw terminals are provided on the Auxiliary Terminal Board for connections to user devices. These screw terminals have the same pin labels as the 36-pin DSM314 faceplate connector. For detailed connection information, refer to “Analog Servo Axis 1-4 Circuit and Pin Assignments “ on page 71.

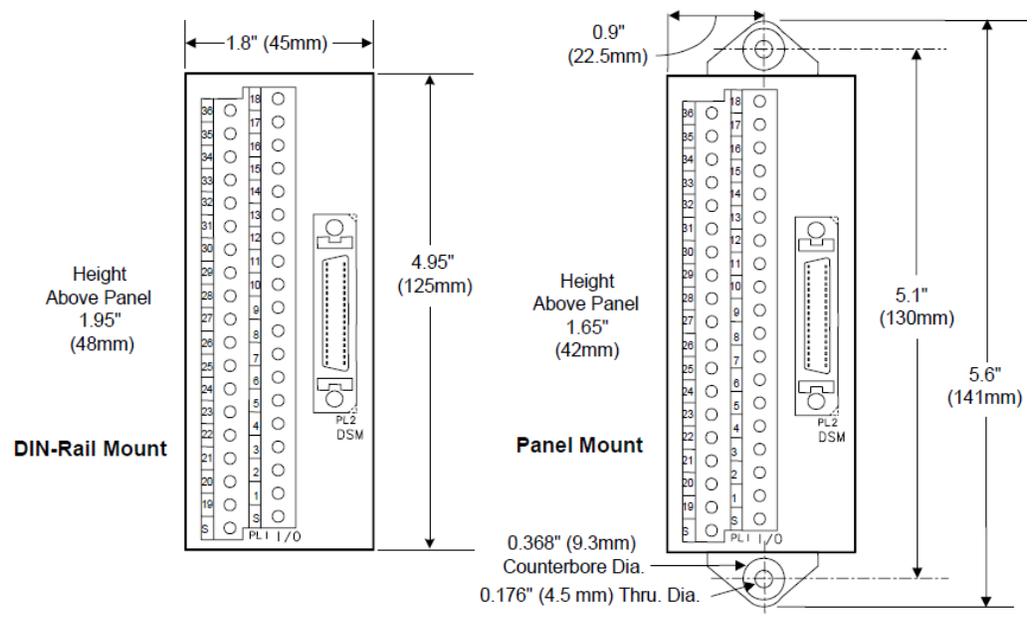
The maximum voltage that should be applied to I/O terminals 16-18 and 34-36 is 30 VDC. The maximum voltage for any other input terminal is 5 VDC.

Six 130V MOVs are installed between selected I/O points and the shield (frame ground) for noise suppression. The I/O terminal points so connected are 16, 17, 18, 34, 35, and 36.

The I/O terminals support a wire gauge of 14-28 AWG. Maximum screw torque that may be applied is 5 inch-pounds.

Note: Two of the screw terminals are labeled S for Shield. A short earth ground wire should be connected from one of the S terminals directly to a panel earth ground. The cable shields for any shielded cables from user devices should connect to either of the S terminals.

Figure 37: IC693ACC336 Terminal Board Mounting Dimensions



Converting From DIN-Rail Mounting to Panel Mounting

The following parts are used in either the DIN-rail or Panel mount assembly options. The auxiliary terminal board is shipped configured for DIN-rail mounting. The instructions in this section guide you in converting the board to its panel mounting optional configuration.

The following table and drawings describe the various plastic parts that make up the auxiliary terminal board assembly and shows a side view of the board configured for DIN-rail mounting.

Table 18: Auxiliary Terminal Board Components

Phoenix Contact Part Number	Description	Quantity
UM45 Profil 105.25	PCB Carrier	1
UM 45-SEFE with 2 screws	Side element with Foot	2
UMK 45-SES with 2 screws*	Side Element	2
UMK-BF*	Mounting Ear	2
* Parts shipped with auxiliary terminal board for optional panel mounting		

Figure 38: Auxiliary Terminal Board Assembly Drawings

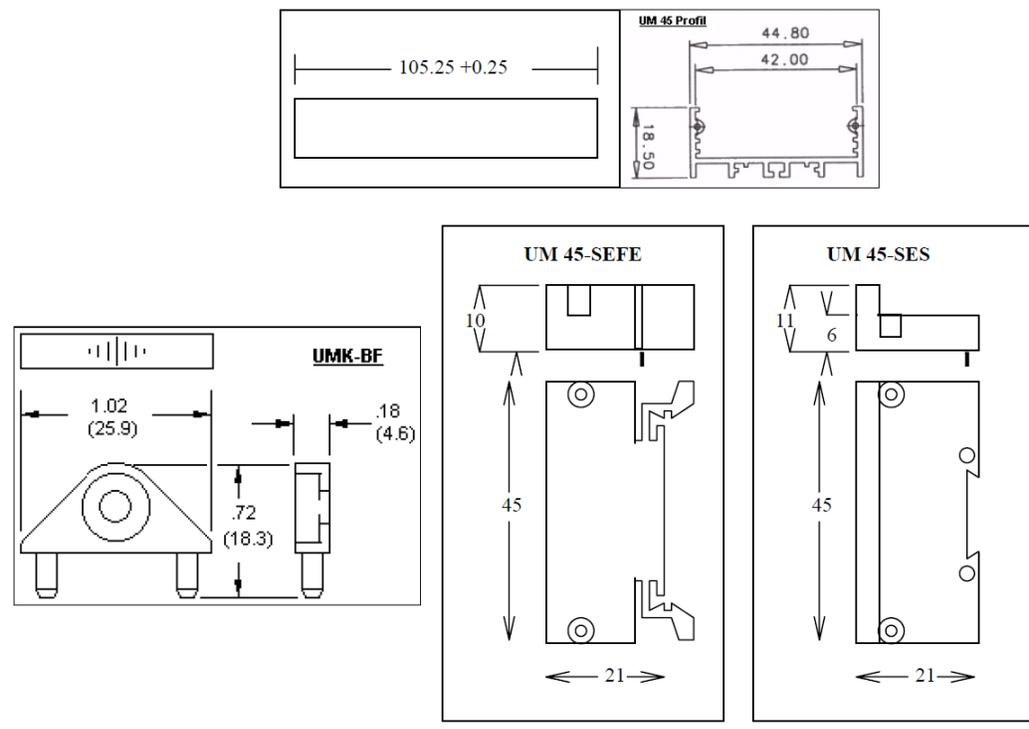
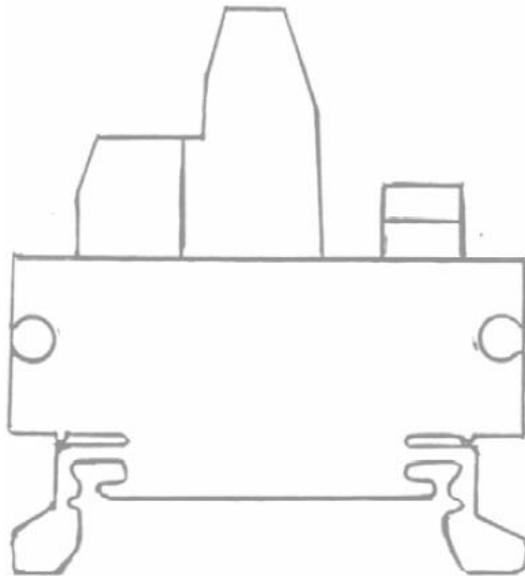


Figure 39: Auxiliary Terminal Board Assembly Side View



The following procedure should be used if you wish to mount the auxiliary terminal board directly to a panel instead of on a DIN-rail. Remember to save all removed parts for possible later conversion back to DIN-rail mounting.

1. Using a small bladed Phillips screwdriver, carefully remove the two screws holding one UM-45 SEFE side element with foot to the UM 45 profile PCB carrier. Save this part for possible future use in converting the terminal board back to its DIN-rail mounting configuration.
2. Attach one UMK 45-SES side element to the PCB carrier in place of the side removed in step 1 above, again using the two screws. Be careful to not over tighten the screws.
3. Insert one UMK-BF mounting ear into the appropriate two holes in the side element. Note that the mounting ear has a recessed hole for later inserting a (user supplied) mounting screw. The recessed hole should face upwards to accommodate the mounting screw.
4. Repeat steps 1-3 above for the other side of the terminal board.

3.3.5 Cables

Five cables are available for the DSM314:

Table 19: Cables for the DSM314

Cable	Description	Length	Application
IC693CBL316	Station Manager Cable	1 meter	DSM314 Comm for firmware upgrade
IC693CBL324	Terminal Board Connection Cable	1 meter	DSM314 to Servo Axis Terminal Board or Aux Terminal Board
IC693CBL325	Terminal Board Connection Cable	3 meters	DSM314 to Servo Axis Terminal Board or Aux Terminal Board
IC800CBL001	Digital Servo Command Cable	1 meter	Digital Servo Axis Terminal Board or DSM to Digital Servo Amp
IC800CBL002	Digital Servo Command Cable	3 meters	Digital Servo Axis Terminal Board or DSM to Digital Servo Amp

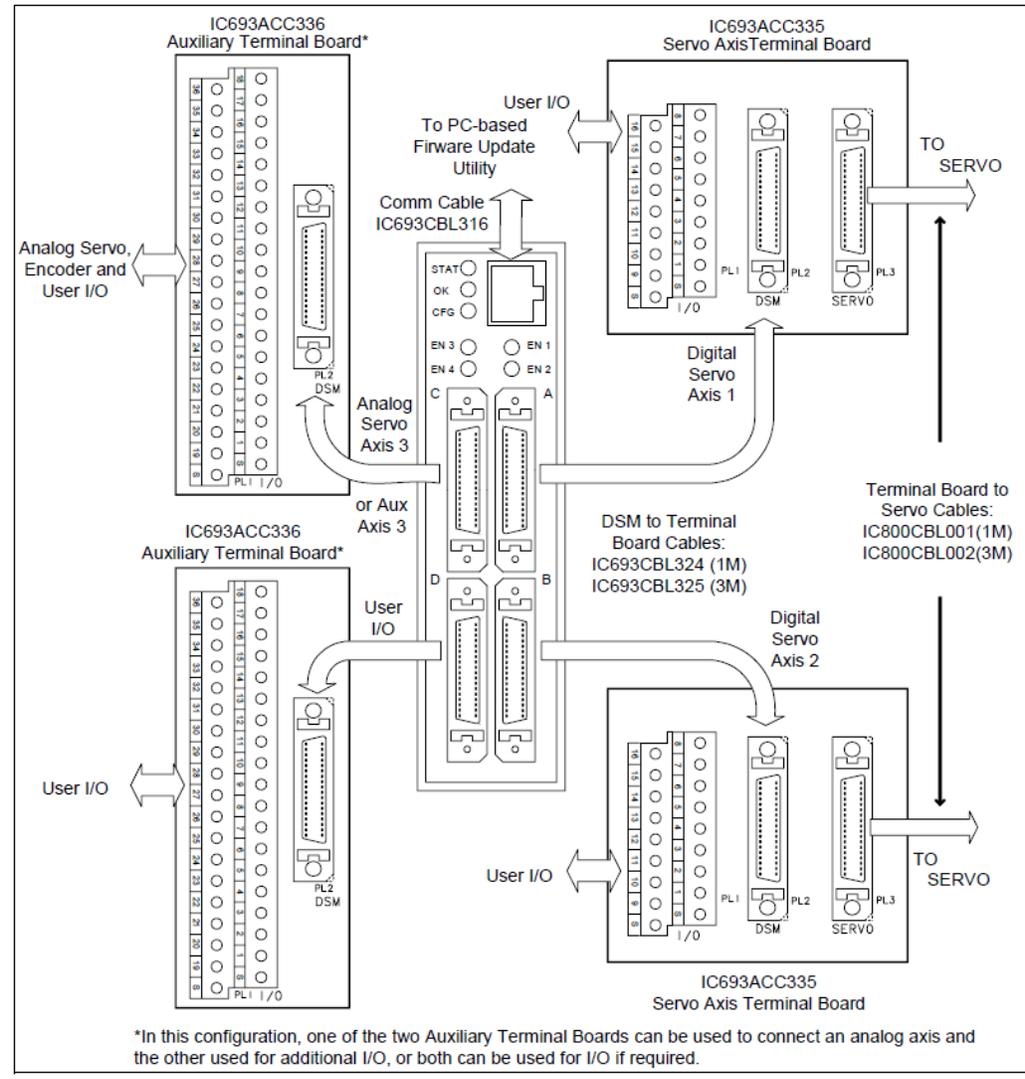
Custom Terminal Board and Servo cables are available in longer lengths by contacting your Emerson distributor. The maximum recommended cable length for the DSM connector to the α and β Series servo amplifier is 50 meters.

The cables use special shielding and construction to ensure reliable servo operation. We recommend that users do not attempt any field modifications of the cables or connectors.

Note: *If a Digital Servo Axis does not use any of the devices that normally connect to the IC693ACC335 Digital Servo Terminal Board screw terminals, the Terminal Board and Terminal Board Cable IC693CBL324/325 are not needed. Instead, the Digital Servo Command Cable IC800CBL001/002 can be connected directly from the Digital Servo Amplifier to the DSM314 faceplate A or B connector. When this is done, the OT Limit Sw configuration parameter must be set to Disabled in the configuration software or the DSM will not operate.*

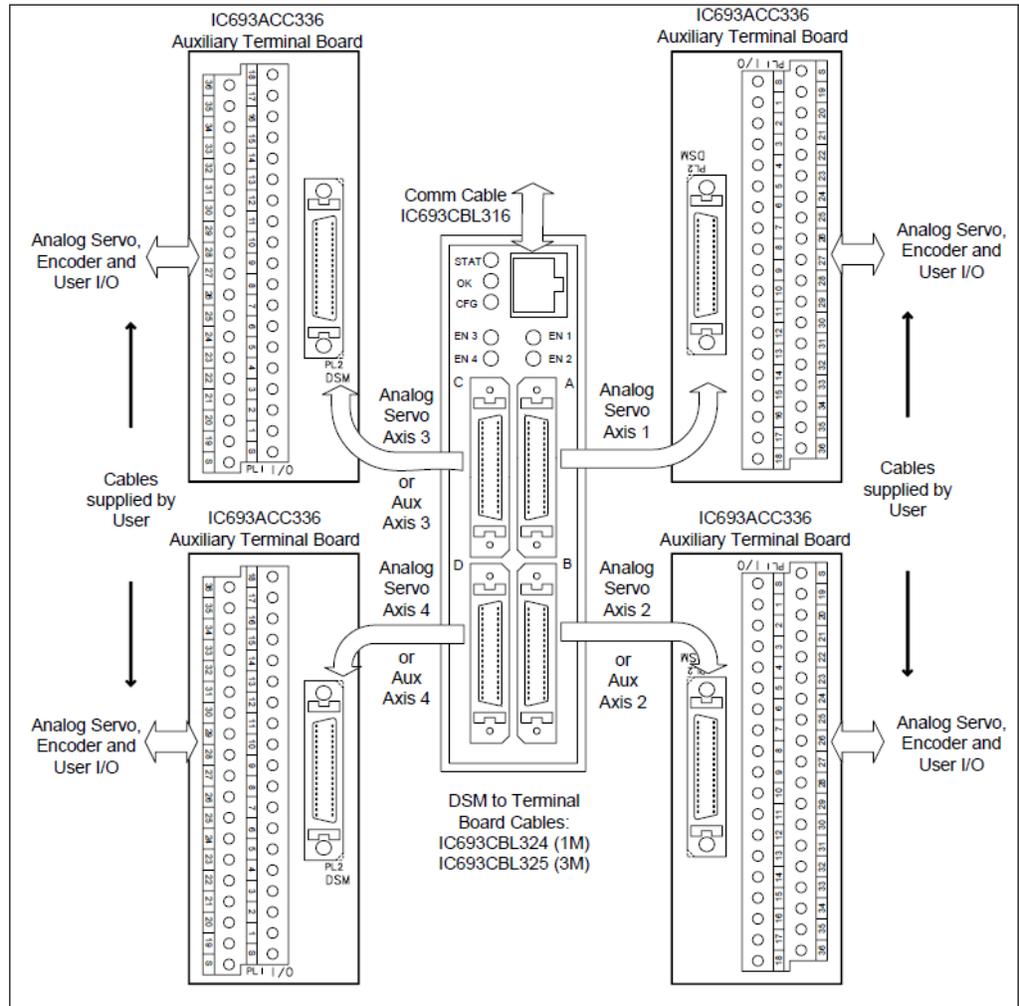
The figure below illustrates the Digital Servo Axis terminal board and cables associated with the DSM314.

Figure 40: DSM314 Digital Servo Terminal Boards and Connectors



The figure below illustrates the Analog Servo terminal boards and cables associated with the DSM314.

Figure 41: DSM314 Terminal Boards and Connectors for S2K or Third-Party Analog Servos



Note: See GFK-1581 for SL Servos and GFK-1866 for S2K servos.

I/O Cable Grounding

Properly routing signal cables, amplifier power cables and motor power cables along with installation of proper Class 3 grounding will insure reliable operation. Typically, Class 3 grounding specifies a ground conductor of a minimum wire diameter larger than the power input wire diameter, connected via a maximum 100-ohm resistance to an earth ground. Consult local electrical codes and install in conformance to local regulations.

The specifications for completing the α and β Series Digital Servo amplifier installation and wiring, including amplifier grounding are completely described in the manual GFH- 001, Servo Product Specification Guide.

When routing signal lines, amplifier input power line and motor power line, the signal lines must be separated from the power lines. The following table indicates how to separate the cables.

Table 20: Separation of signal lines

Group	Signal	Action
A	Amplifier input power Motor Power Master Control Contactor (MCC) drive coil. The MCC switches amplifier input power.	Separate a minimum 10cm from group “B” signals by bundling separately or use electromagnetic shielding (grounded steel plate). Use noise protector for MCC.
B	DSM to Axis Terminal cable Axis terminal cable to amplifier DSM to Aux Terminal cable Encoder feedback cable	Separate a minimum 10cm from group “A” signals by bundling separately or use electromagnetic shielding (grounded steel plate). Use all required individual cable shield grounds and grounding bar connections.

DSM to α or β Series Digital Servo Amplifier – Signal Cable Grounding

The signal cables used with the DSM314 contain shields that must be properly grounded to ensure reliable operation. The illustration below shows cable grounding recommendations for typical installations. The following points should be considered:

1. The DSM314 faceplate ground wire must be connected to a reliable panel ground.
2. The Digital Servo Axis Terminal Block and Auxiliary Terminal Block each provide two screw terminals labeled S. A short ground wire must be connected from one of the S terminals to a reliable panel ground.

The α and β Series Digital Servo amplifier encoder feedback cable always requires an ZA99L-0035-0001 Cable Shield Grounding Clamp and one of the 11 available slots on a Z44B295864-001 Grounding Bar at the amplifier end of the cable. This clamp arrangement serves as a mechanical strain relief and as cable shield ground. The outer insulation of the Digital servo amplifier cable must be removed to expose the cable shield in the contact area of the clamp.

Figure 42: Detail of Cable Grounding Clamp ZA99L-0035-0001

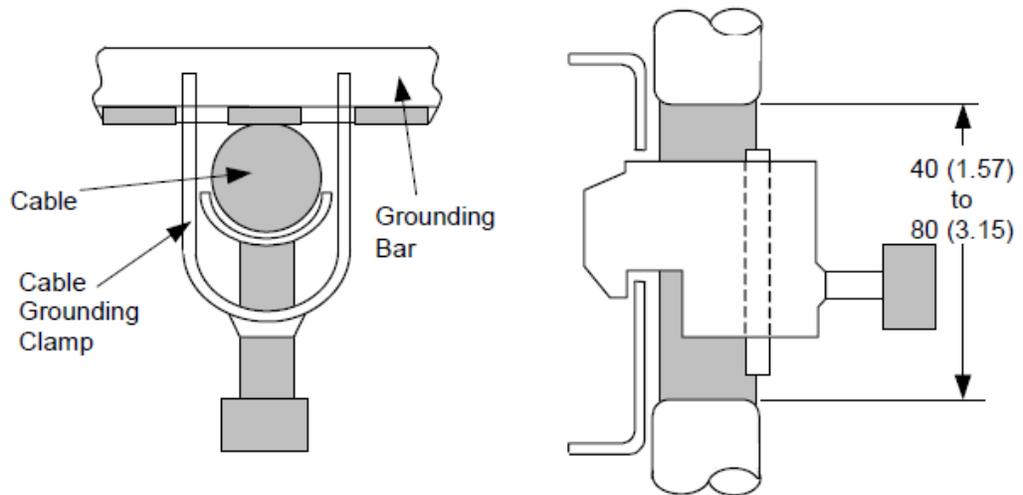


Figure 43: Z44B295864-001 Grounding Bar, Side View Dimensions

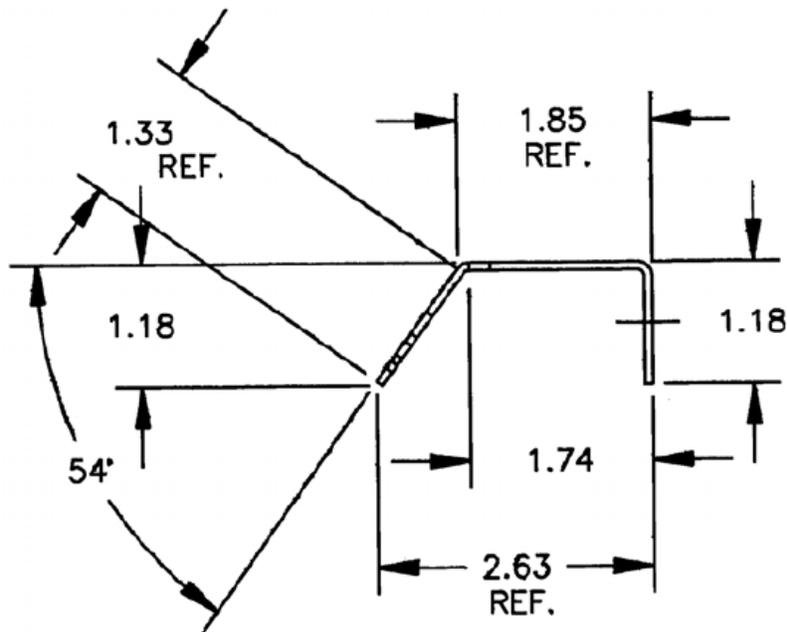
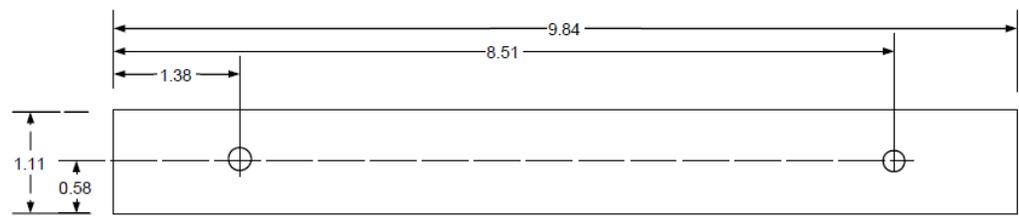


Figure 44: Z44B295864-001 Grounding Bar Dimensions, Rear View Showing Mounting Holes



- For installations that must meet IEC electrical noise immunity standards, a Cable Shield Grounding Clamp ZA99L-0035-0001 and one of the 11 available slots on the Grounding Bar Z44B295864-001 must also be used at the Digital Servo Axis Terminal Block end of the servo amplifier cable IC800CBL001/002. If the Digital servo amplifier cable is connected directly to the DSM314 faceplate (no Digital Servo Axis Terminal Block used) the Grounding Clamp and Bar are not required at the faceplate end of the cable.

For additional information, refer to Installation Requirements for Conformance to Standards, GFK-1179.

Figure 45: DSM314 I/O Cable Grounding

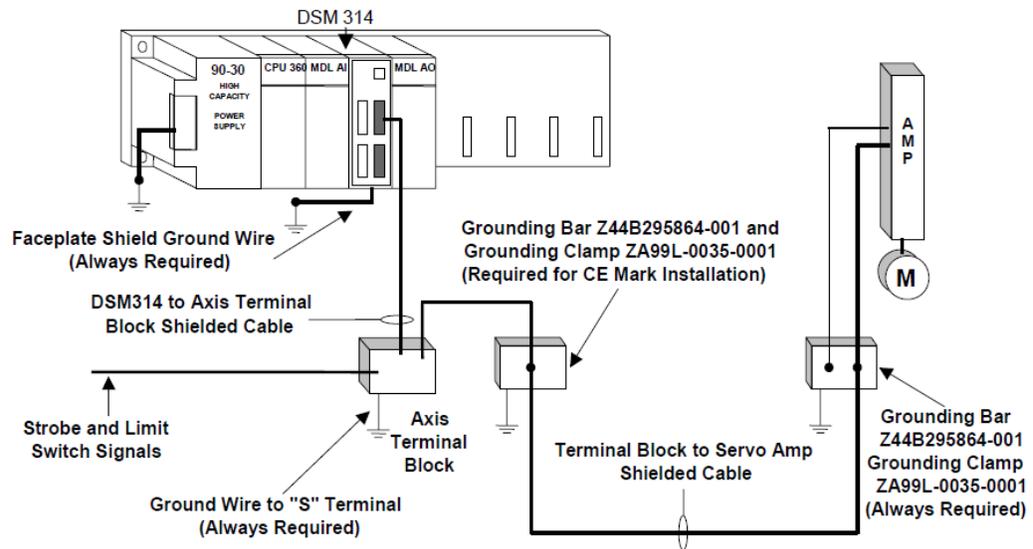


Figure 3-15. DSM314 I/O Cable Grounding

I/O Circuit Identifiers and Signal Names

I/O circuit identifiers provide a consistent method of naming the I/O circuits. For example, IN1 refers to the first of three differential / single ended 5v inputs for each axis.

Signal names are assigned to the circuit identifiers for each axis. The signal name consists of the circuit identifier followed by a suffix A-D to identify the axis connector. Differential circuits also have suffixes P (positive) and M (minus) to identify the (+) and (-) signal for each differential pair.

Example: OUT2 is the circuit identifier for the first differential 5v output on each connector. The signal names associated with circuit OUT2 are:

Table 21: Signal Names Associated with OUT2

Axis:	Axis 1	Axis 2	Axis 3	Axis 4
Connector:	A	B	C	D
(+) Output Signal:	OUT2P_A	OUT2P_B	OUT2P_C	OUT2P_D
(-) Output Signal:	OUT2M_A	OUT2M_B	OUT2M_C	OUT2M_D

I/O Circuit Function and Pin Assignments

The next three tables list the I/O circuit functional assignments as well as the connector and terminal board pin assignments for each axis connector. Although each connector has the same I/O circuits, the functional assignment of the I/O circuits is axis dependent:

Table 22: Connector Axis Assignment and Function

Connector	Axis Number	Axis Type	I/O Usage
A	1	Servo Axis	Closed Loop Digital / Analog Servo Control and user I/O
B	2	Servo Axis	Closed Loop Digital / Analog Servo Control or Auxiliary analog and digital I/O
C	3	Servo Axis	Closed Loop Analog Servo Control or Auxiliary analog and digital I/O
D	4	Servo Axis	Closed Loop Analog Servo Control or Auxiliary analog and digital I/O

Digital Servo Axis 1, 2 Circuit and Pin Assignments

This table identifies all circuits and pin assignments for Digital Servo Axis 1 and Digital Servo Axis 2. The shaded areas indicate signals that are cabled to the servo amplifier and are not available for user connections.

Table 23: Circuit and Pin Assignments for Digital Servo Axis 1 and Digital Servo Axis 2

Circuit Identifier	Circuit Type	Analog Servo Axis 1, 2 Circuit Function	Axis 1 Signal Name	Axis 2 Signal Name	Faceplate Pin	Axis Term Board Terminal
IN1	Single ended / differential 5v inputs	Strobe Input 1 (+)	IN1P_A	IN1P_B	1	1
		Strobe Input 1 (-)	IN1M_A	IN1M_B	19	9
IN2		Strobe Input 2 (+)	IN2P_A	IN2P_B	2	2
		Strobe Input 2 (-)	IN2M_A	IN2M_B	20	10
IN3		Ser Encoder Data (+)	IN3P_A	IN3P_B	3	
		Ser Encoder Data (-)	IN3M_A	IN3M_B	21	
P5V	5v Power	5v Power	P5V_A	P5V_B	4	3
0V	0v	0v	0V_A	0V_B	22,23	11
IN4	Single ended 5v in	Servo Ready Input	IN4_A	IN4_B	5	
IO5	Single ended 5v inputs / outputs	Servo PWM / Alarm	IO5_A	IO5_B	9	
IO6		Servo PWM / Alarm	IO6_A	IO6_B	10	
IO7		Servo PWM / Alarm	IO7_A	IO7_B	11	
IO8		Servo ENBL / Alarm	IO8_A	IO8_B	12	
0V	0v	0v	0V_A	0V_B	27-30	

Circuit Identifier	Circuit Type	Analog Servo Axis 1, 2 Circuit Function	Axis 1 Signal Name	Axis 2 Signal Name	Faceplate Pin	Axis Term Board Terminal
IN9	24v optically isolated inputs	Overtravel (+)	IN9_A	IN9_B	16	6
IN10		Overtravel (-)	IN10_A	IN10_B	34	14
IN11		Home Switch	IN11_A	IN11_B	17	7
INCOM	24v Input Common	24v Input Common	INCOM_A	INCOM_B	35	15
OUT1	24 v, 125 mA DC SSR output	Host controller 24v Output (+)	OUT1P_A	OUT1P_B	18	8 16
		Host controller 24v Output (-)	OUT1M_A	OUT1M_B	36	
OUT2	Differential 5v outputs	Ser Encoder Req (+)	OUT2P_A	OUT2P_B	13	
		Ser Encoder Req (-)	OUT2M_A	OUT2M_B	31	
OUT3		Host controller 5v Output (+)	OUT3P_A	OUT3P_B	14	5 13
		Host controller 5v Output (-)	OUT3M_A	OUT3M_B	32	
ENBL	24v, 30 mA SSR output	Servo MCON (+)	ENBL1_A	ENBL1_B	15	
		Servo MCON 0v	ENBL2_A	ENBL2_B	33	
AIN1	Differential +/- 10v Analog Inputs	IR Phase Current (+)	AIN1P_A	AIN1P_B	7	
		IR Phase Current (-)	AIN1M_A	AIN1M_B	25	
AIN2		IS Phase Current (+)	AIN2P_A	AIN2P_B	8	
		IS Phase Current (-)	AIN2M_A	AIN2M_B	26	
AOUT1	+/- 10v Analog Out	Host controller Analog Out	AOUT_A	AOUT_B	6	4
ACOM	Analog Out com	Analog Out Com	ACOM_A	ACOM_B	24	12
SHIELD	Cable Shield	Cable Shield	SHIELD_A	SHIELD_B		S

Analog Servo Axis 1-4 Circuit and Pin Assignments

This table identifies all circuits and pin assignments for Analog Servo Axis 1 - Analog Servo Axis 4. The shaded areas indicate signals that are unused and not available for user connections.

Table 24: Circuit and Pin Assignments for Analog Servo Axis 1 - Analog Servo Axis 4

Circuit Identifier	Circuit Type	Analog Servo Axis 1-4 Circuit Function	Axis 1 Signal Name	Axis 2 Signal Name	Axis 3 Signal Name	Axis 4 Signal Name	Faceplate Pin	Aux Term Board Terminal
IN1	Single ended /	Encoder Chan A (+) Encoder Chan A (-)	IN1P_A IN1M_A	IN1P_B IN1M_B	IN1P_C IN1M_C	IN1P_D IN1M_D	1 19	1 19

Circuit Identifier	Circuit Type	Analog Servo Axis 1-4 Circuit Function	Axis 1 Signal Name	Axis 2 Signal Name	Axis 3 Signal Name	Axis 4 Signal Name	Faceplate Pin	Aux Term Board Terminal
IN2	differential 5v inputs	Encoder Chan B (+) Encoder Chan B (-)	IN2P_A IN2M_A	IN2P_B IN2M_B	IN2P_C IN2M_C	IN2P_D IN2M_D	2 20	2 20
IN3		Encoder Marker (+) Encoder Marker (-)	IN3P_A IN3M_A	IN3P_B IN3M_B	IN3P_C IN3M_C	IN3P_D IN3M_D	3 21	3 21
P5V	5v Power	5v Encoder Power	P5V_A	P5V_B	P5V_C	P5V_D	4	4
0V	0v	0v	0V_A	0V_B	0V_C	0V_D	22,23	22,23
IN4	Single ended 5v in	Servo Ready Input	IN4_A	IN4_B	IN4_C	IN4_D	5	5
IO5	Single ended 5v inputs / outputs	Strobe 1 Input	IO5_A	IO5_B	IO5_C	IO5_D	9	9
IO6		Strobe 2 Input	IO6_A	IO6_B	IO6_C	IO6_D	10	10
IO7		Not Used	IO7_A	IO7_B	IO7_C	IO7_D	11	11
IO8		Not Used	IO8_A	IO8_B	IO8_C	IO8_D	12	12
0V	0v	0v	0V_A	0V_B	0V_C	0V_D	27-30	27-30
IN9	24v optically isolated inputs	Overtravel (+)	IN9_A	IN9_B	IN9_C	IN9_D	16	16
IN10		Overtravel (-)	IN10_A	IN10_B	IN10_C	IN10_D	34	34
IN11		Home Switch	IN11_A	IN11_B	IN11_C	IN11_D	17	17
INCOM	24v Input Common	24v Input Common	INCOM_A	INCOM_B	INCOM_C	INCOM_D	35	35
OUT1	24 v, 125 mA DC SSR output	PLC 24v Output (+) PLC 24v Output (-)	OUT1P_A OUT1M_A	OUT1P_B OUT1M_B	OUT1P_C OUT1M_C	OUT1P_D OUT1M_D	18 36	18 36
OUT2	Differential 15v outputs	Not Used Not Used	OUT2P_A OUT2M_A	OUT2P_B OUT2M_B	OUT2P_C OUT2M_C	OUT2P_D OUT2M_D	13 31	13 31
OUT3		PLC 5v Output (+) PLC 5v Output (-)	OUT3P_A OUT3M_A	OUT3P_B OUT3M_B	OUT3P_C OUT3M_C	OUT3P_D OUT3M_D	14 32	14 32
ENBL	24v, 30 mA SSR output	Servo Enable (+) Servo Enable (-)	ENBL1_A ENBL2_A	ENBL1_B ENBL2_B	ENBL1_C ENBL2_C	ENBL1_D ENBL2_D	15 33	15 33
AIN1	Differential 1 +/- 10v Analog Inputs	PLC Analog In (+) PLC Analog In (-)	AIN1P_A AIN1M_A	AIN1P_B AIN1M_B	AIN1P_C AIN1M_C	AIN1P_D AIN1M_D	7 25	7 25
AIN2		PLC Analog In (+) PLC Analog In (-)	AIN2P_A AIN2M_A	AIN2P_B AIN2M_B	AIN2P_C AIN2M_C	AIN2P_D AIN2M_D	8 26	8 26

Circuit Identifier	Circuit Type	Analog Servo Axis 1-4 Circuit Function	Axis 1 Signal Name	Axis 2 Signal Name	Axis 3 Signal Name	Axis 4 Signal Name	Faceplate Pin	Aux Term Board Terminal
AOUT1	+/- 10v Analog Out	Servo Vel Cmd (+) or Servo Torque Cmd (+)	AOUT_A	AOUT_B	AOUT_C	AOUT_D	6	6
ACOM	Analog Out Com	Servo Vel Cmd Com or Servo Torque Com	ACOM_A	ACOM_B	ACOM_C	ACOM_D	24	24
SHIELD	Cable Shield	Cable Shield	SHIELD_A	SHIELD_B	SHIELD_C	SHIELD_D		S

Aux Axis 2-4 Circuit and Pin Assignments

This table identifies all circuits and pin assignments for Aux Axis 2 - Aux Axis 4. The shaded areas indicate signals that are unused and not available for user connections.

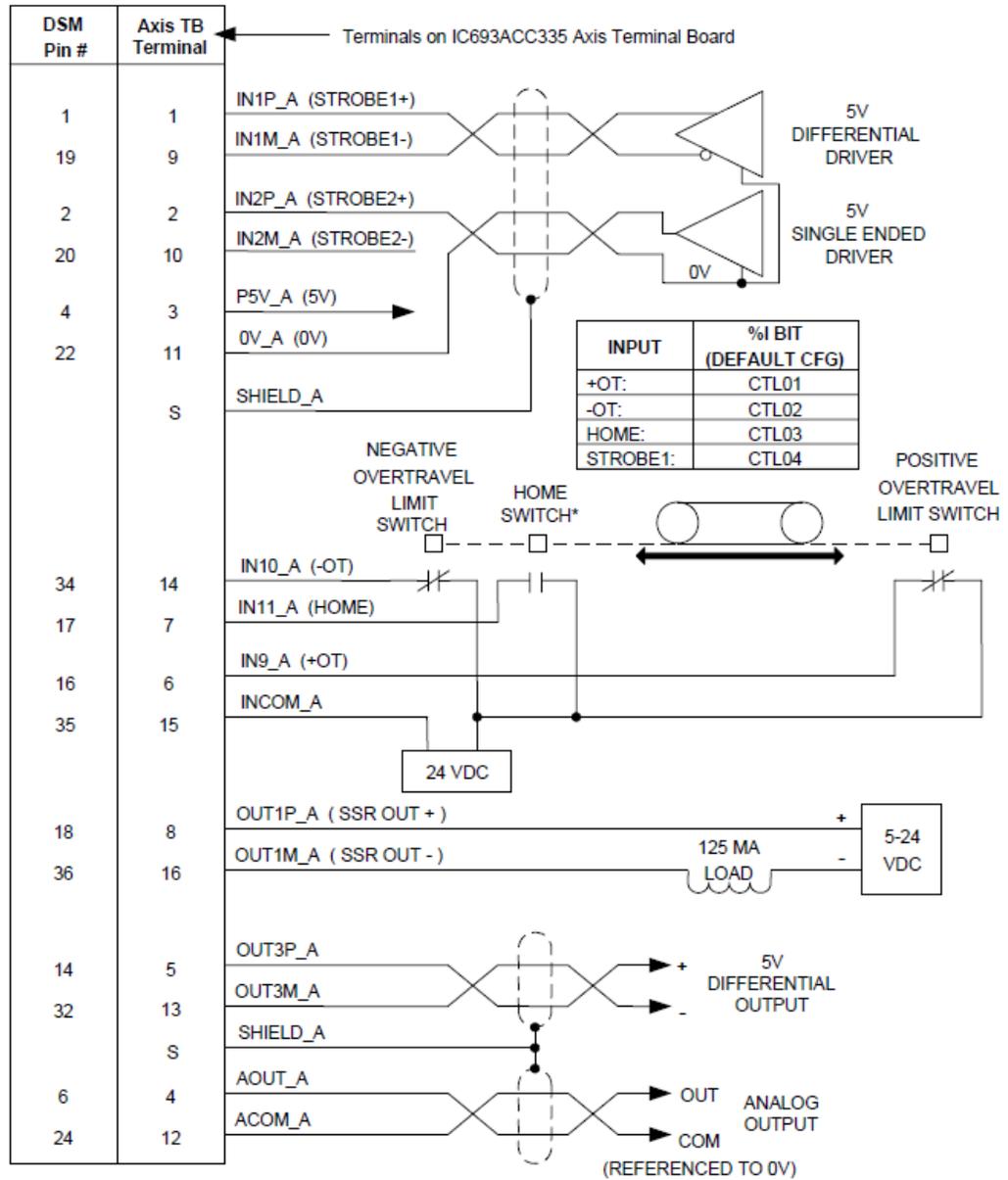
Table 25: Circuit and Pin Assignments for Aux Axis 3 (Connector C)

Circuit Identifier	Circuit Type	Aux Axis 2-4 Circuit Function	Axis 2 Signal Name	Axis 3 Signal Name	Axis 4 Signal Name	Faceplate Pin	Aux Term Board Terminal
IN1	Single ended / differential 5v inputs	Encoder Chan A (+) Encoder Chan A (-)	IN1P_B IN1M_B	IN1P_C IN1M_C	IN1P_D IN1M_D	1 19	1 19
IN2		Encoder Chan B (+) Encoder Chan B (-)	IN2P_B IN2M_B	IN2P_C IN2M_C	IN2P_D IN2M_D	2 20	2 20
IN3		Encoder Marker (+) Encoder Marker (-)	IN3P_B IN3M_B	IN3P_C IN3M_C	IN3P_D IN3M_D	3 21	3 21
P5V	5v from PLC	5v Encoder Power	P5V_B	P5V_C	P5V_D	4	4
0V	0v	0v	0V_B	0V_C	0V_D	22,23	22,23
IN4	Single ended 5v in	PLC 5v Input	IN4_B	IN4_C	IN4_D	5	5
IO5	Single ended 5v inputs / outputs	Strobe 1 Input	IO5_B	IO5_C	IO5_D	9	9
IO6		Strobe 2 Input	IO6_B	IO6_C	IO6_D	10	10
IO7		Not Used	IO7_B	IO7_C	IO7_D	11	11
IO8		Not Used	IO8_B	IO8_C	IO8_D	12	12
0V	0v	0v	0V_B	0V_C	0V_D	27-30	27-30
IN9	24v optically isolated inputs	PLC 24v Input	IN9_B	IN9_C	IN9_D	16	16
IN10		PLC 24v Input	IN10_B	IN10_C	IN10_D	34	34
IN11		Home Switch	IN11_B	IN11_C	IN11_D	17	17
INCOM	24v Input Common	24v Input Common	INCOM_B	INCOM_C	INCOM_D	35	35
OUT1	24v, 125 mA DC SSR output	PLC 24v Output (+) PLC 24v Output (-)	OUT1P_B OUT1M_B	OUT1P_C OUT1M_C	OUT1P_D OUT1M_D	18 36	18 36
OUT2		Not Used Not Used	OUT2P_B OUT2M_B	OUT2P_C OUT2M_C	OUT2P_D OUT2M_D	13 31	13 31
OUT3		PLC 5v Output (+) PLC 5v Output (-)	OUT3P_B OUT3M_B	OUT3P_C OUT3M_C	OUT3P_D OUT3M_D	14 32	14 32
ENBL	24v, 30 mA SSR output	ON when Force	ENBL1_B	ENBL1_C	ENBL1_D	15	15
		Analog Output %AQ Cmd is active	ENBL2_B	ENBL2_C	ENBL2_D	33	33
AIN1	Differential +/- 10v Analog Inputs	PLC Analog In (+) PLC Analog In (-)	AIN1P_B AIN1M_B	AIN1P_C AIN1M_C	AIN1P_D AIN1M_D	7 25	7 25
AIN2		PLC Analog In (+) PLC Analog In (-)	AIN2P_B AIN2M_B	AIN2P_C AIN2M_C	AIN2P_D AIN2M_D	8 26	8 26
AOUT1	+/- 10v Analog Out	PLC Analog Out	AOUT_B	AOUT_C	AOUT_D	6	6
ACOM	Analog Out com	Analog Out Com	ACOM_B	ACOM_C	ACOM_D	24	24
SHIELD	Cable Shield	Cable Shield	SHIELD_B	SHIELD_C	SHIELD_D		S

I/O Connection Diagrams

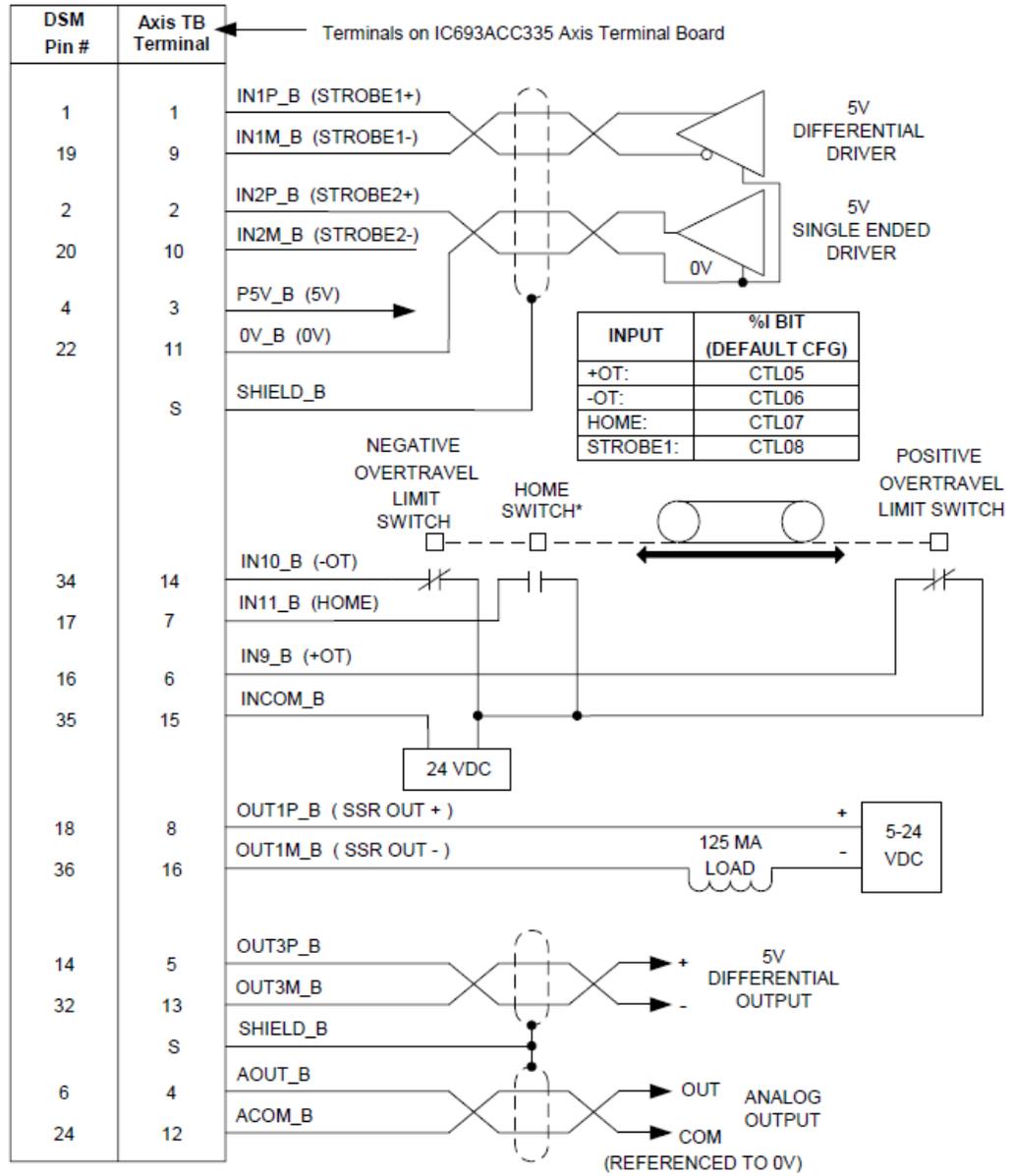
The following diagrams illustrate typical user connections to the DSM314.

Figure 46: Digital Servo Axis-1 Connections



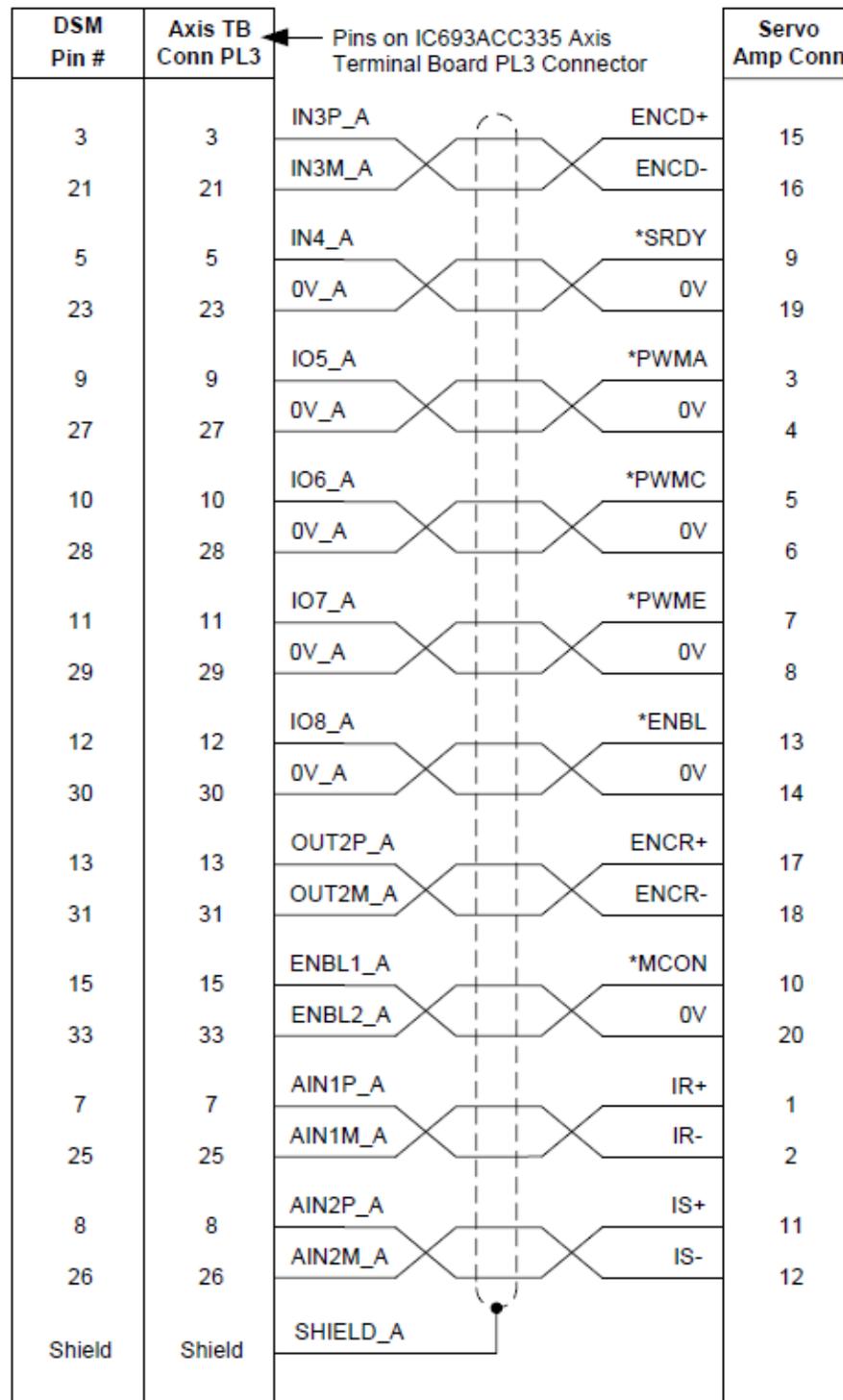
* Note: See Chapter 6 for home switch information

Figure 47: Digital Servo Axis-2 Connections



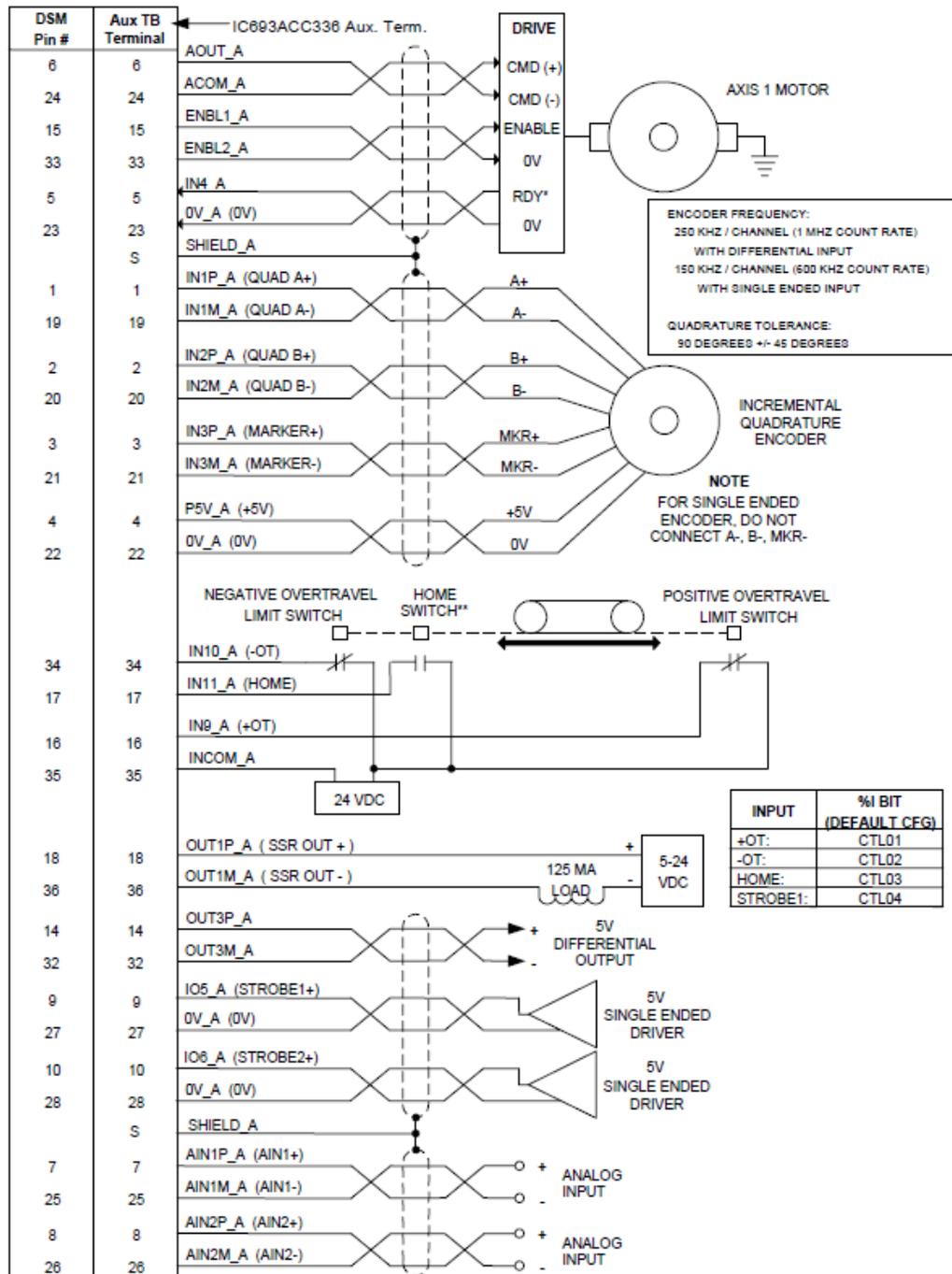
*Note: See Chapter 6 for home switch information

Figure 48: α and β Series Digital Servo Command Cable (IC800CBL001/002) Connections



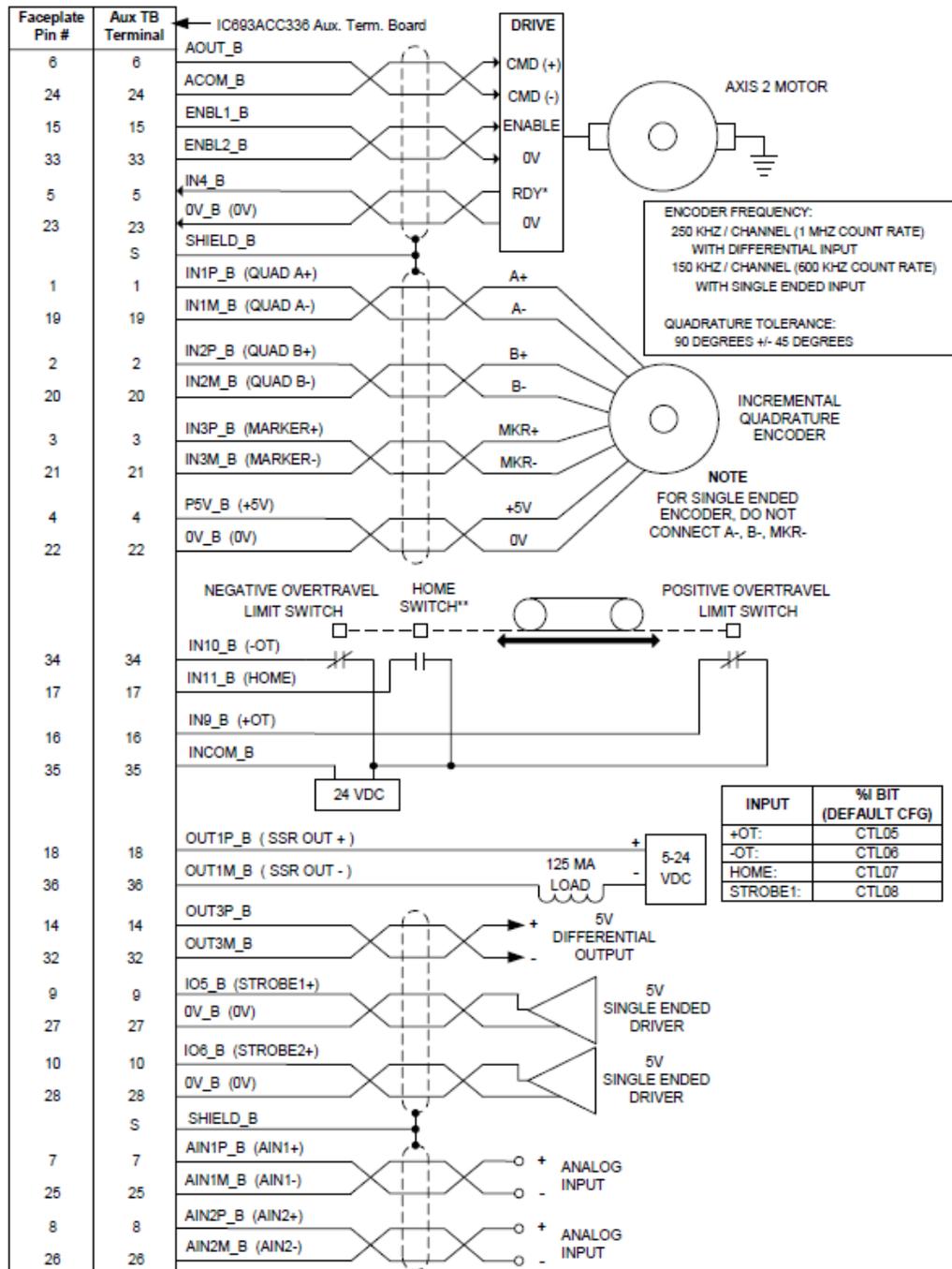
* Denotes a negated signal

Figure 49: Analog Servo Axis-1 Connections



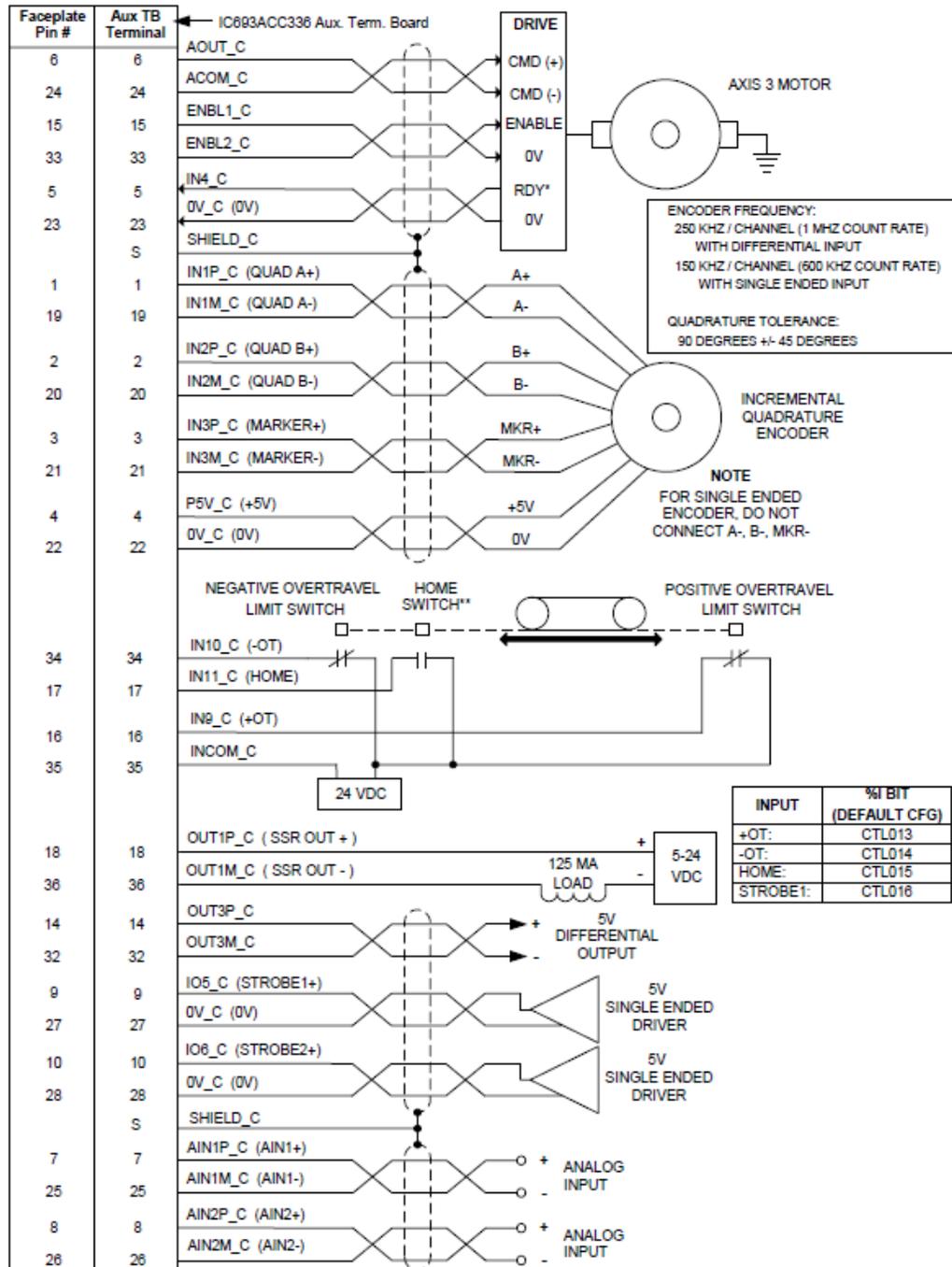
NOTES: * Denotes a negated signal
** See Chapter 6 for home switch information

Figure 50: Analog Servo Axis-2 Connections



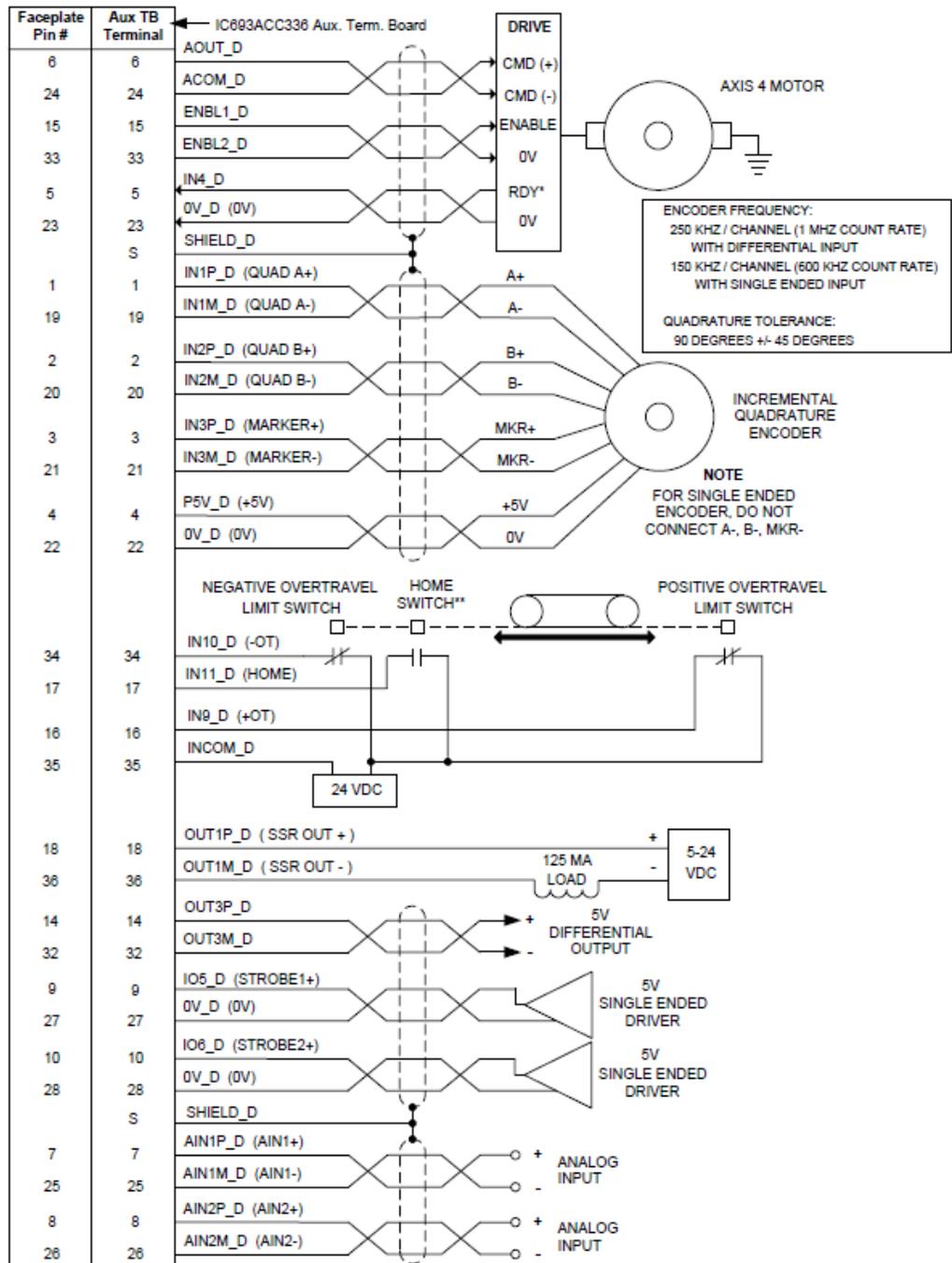
NOTES: * Denotes a negated signal.
 ** See Chapter 8 for home switch information

Figure 51: Analog Servo Axis-3 Connections



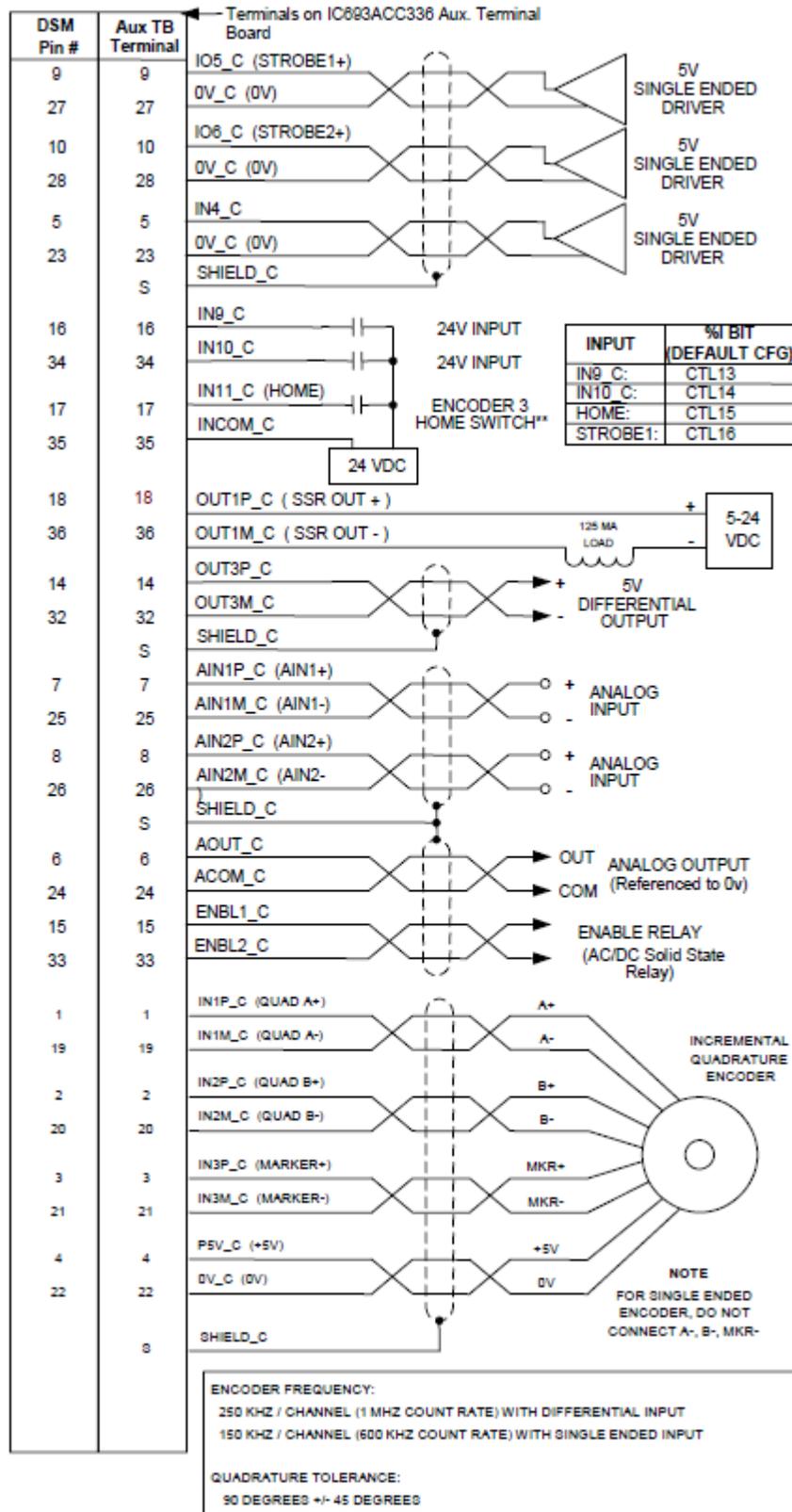
NOTES: * Denotes a negated signal.
** See Chapter 6 for home switch information

Figure 52: Analog Servo Axis-4 Connections



NOTES: * Denotes a negated signal.
** See Chapter 6 for home switch information

Figure 53: Aux Axis Connections (Axis 3 Shown)



** Note: See Chapter 6 for home switch information

I/O Specifications

The specifications and simplified schematics for the module's I/O circuits are provided on the following pages. The I/O circuits described are as follows:

- Differential/Single Ended 5v Inputs (IN1, IN2, IN3)
- Single Ended 5v Sink Input (IN4)
- Optically Isolated 24v Source/Sink Inputs (IN9, IN10, IN11, INCOM)
- Single Ended 5v Inputs/Outputs (IO5, IO6, IO7, IO8)
- 5v Differential Outputs (OUT2, OUT3)
- 24v DC Optically Isolated Output (OUT1)
- Optically Isolated Enable Relay Output (OUT4)
- Differential +/- 10v Analog Inputs (AIN1, AIN2)
- Single Ended +/- 10v Analog Outputs (AOUT1, ACOM)
- +5v Power (P5V, 0V)

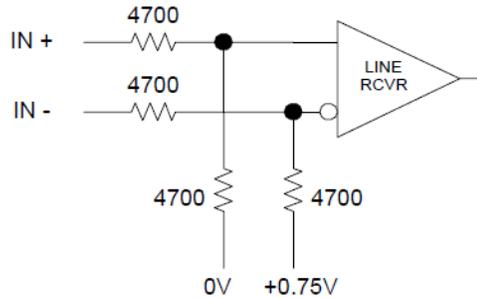
Differential / Single Ended 5v Inputs

Circuit Identifier	Digital Servo Axis 1, 2 Circuit Function	Analog Servo Axis 1- 4 and Aux Axis 2-4 Circuit Function	Signal Name (X = A, B, C, or D Connector)	Faceplate Pin	Auxiliary Terminal Board	Servo Terminal Board
IN1	Strobe Input 1 (+) Strobe Input 1 (-)	Encoder Chan. A (+) Encoder Chan. A (-)	IN1P_X IN1M_X	1 19	1 19	1 9
IN2	Strobe Input 2 (+) Strobe Input 2 (-)	Encoder Chan. B (+) Encoder Chan. B (-)	IN2P_X IN2M_X	2 20	2 20	2 10
IN3	Ser Encoder Data (+) Ser Encoder Data (-)	Encoder Marker (+) Encoder Marker (-)	IN3P_X IN3M_X	3 21	3 21	N/C N/C

I/O Type:	Differential / Single Ended 5v Inputs
Circuit Type:	Source Input (9.4K ohm pull-down to 0v)
Input Impedance (+) or (-) Input:	9.4K ohms common mode to 0v 18.8K ohms differential
Maximum Input Voltage:	+/- 15 v common mode +/- 20 v differential
Logic 0 Threshold:	+0.8 v max single ended +0.4 v max differential
Logic 1 Threshold:	+2.0 v min single ended +1.5 v min differential
Input Filtering:	0.5 microsecond typical
Quadrature Encoder Frequency:	250 KHz/channel (1 MHz count rate) max with differential inputs 150 KHz/channel (600 KHz count rate) max with single ended inputs Quadrature Tolerance: 90 degrees +/- 45 degrees

Strobe Response: Minimum Pulse Width: 3 microseconds Position
Capture Accuracy: +/- 2 counts with an additional 10
microseconds of variance

Note: Use (+) Input for single ended mode and leave (-) input floating. Use faceplate 0v pins for common mode reference or single ended signal return. Inputs can be driven by 5v TTL or CMOS logic.

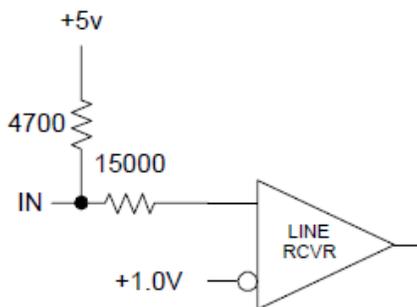


Single Ended 5v Sink Input

Circuit Identifier	Servo Axis 1-4 Circuit Function	Aux Axis 2-4 Circuit Function	Signal Name (X = A, B, C, or D Connector)	Faceplate Pin	Auxiliary Terminal Board	Servo Terminal Board
IN4	Servo Ready Input	Faceplate 5v Input	IN4_X	5	5	N/C

I/O Type: Single Ended 5v Sink Input
 Circuit Type: Sink Input (4.7K ohm pull-up to internal +5v) Input
 Impedance: 4.7K ohms to +5v
 Maximum Input Voltage: +/- 10.0 v
 Logic 0 Threshold: +0.8 v max
 Logic 1 Threshold: +2.0 v min
 Input Filtering: 1.0 microseconds (typical) hardware filter + position loop sampling rate (0.5, 1.0 or 2.0 milliseconds).

Note: This input must be pulled to 0v to turn on.

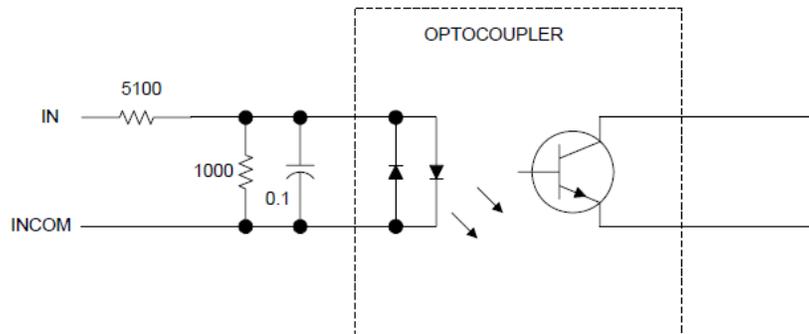


Optically Isolated 24v Source / Sink Inputs

Circuit Identifier	Servo Axis 1-4 Circuit Function	Aux Axis 2-4 Circuit Function	Signal Name (X = A, B, C, or D Connector)	Faceplate Pin	Auxiliary Terminal Board	Servo Terminal Board
IN9	Overtravel (+)	Faceplate 24v Input	IN9_X	16	16	6
IN10	Overtravel (-)	Faceplate 24v Input	IN10_X	34	34	14
IN11	Home Switch	Home Switch	IN11_X	17	17	7
INCOM	24v Input Common	24v Input Common	INCOM_X	35	35	15

I/O Type: Optically Isolated 24v Source / Sink Inputs
 Circuit Type: Source / Sink (5K resistance to INCOM)
 Input Impedance: 5.4K ohms to INCOM (@ 24 VDC)
 Maximum Input Voltage: +/- 30.0v (referenced to INCOM)
 Logic 0 Threshold: +/- 6.0 v max (referenced to INCOM)
 Logic 1 Threshold: +/- 18.0 v min (referenced to INCOM)
 Input Filtering: 5 milliseconds typical

Note: These inputs use bi-directional optocouplers and can be turned on with either a positive or negative input with respect to INCOM.

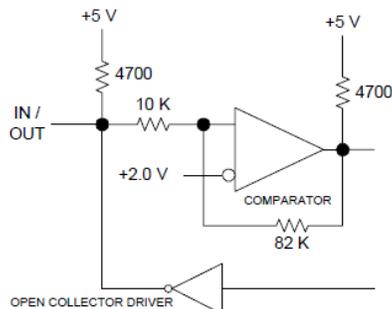


Single Ended 5v Inputs/Outputs

Circuit Identifier	Digital Servo Axis 1, 2 Circuit Function	Analog Servo Axis 1-4 and Aux Axis 2-4 Circuit Function	Signal Name (X = A, B, C, or D Connector)	Faceplate Pin	Auxiliary Terminal Board	Servo Terminal Board
IO5 0v	Servo PWM / Alarm 0v	Strobe 1 Input 0v	IO5_X / IN5_X 0V_X	9 27	9 27	N/C N/C
IO6 0v	Servo PWM / Alarm 0v	Strobe 2 Input 0v	IO6_X / IN6_X 0V_X	10 28	10 28	N/C N/C
IO7 0v	Servo PWM / Alarm 0v	Not Used 0v	IO7_X / IN7_X 0V_X	11 29	11 29	N/C N/C
IO8 0v	Servo ENBL / Alarm 0v	Not Used 0v	IO8_X / IN8_X 0V_X	12 30	12 30	N/C N/C

I/O Type:	Single Ended 5v Inputs / Outputs
Circuit Type:	Sink (4.7K ohm pull-up to internal +5v)
Input Impedance:	4.7K ohms to internal +5v
Maximum Input Voltage:	-1.0 v, +7.0v
Logic 0 Input Threshold:	+0.8 v max
Logic 1 Input Threshold:	+2.4 v min
Input Filtering:	10 microseconds typical
Output Sink Current	10 mA max
On State Output Voltage	+0.5v at 10 mA
Strobe Response:	Minimum Pulse Width: 10 microseconds. Position Capture Accuracy: +/- 2 counts with an additional 10 microseconds of variance

Note: For digital servos, these points act as the PWM / ENBL outputs and Alarm inputs. For Analog Servos and Aux axes, these points are input only. The listed 0v pins should be normally used for the signal return.

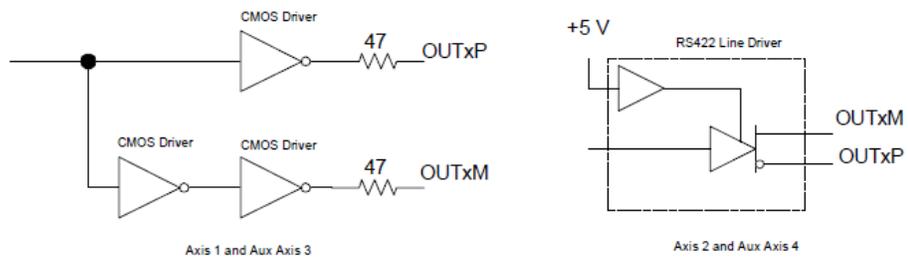


5v Differential Outputs

Circuit Identifier	Digital Servo Axis 1, 2 Circuit Function	Analog Servo Axis 1-4 and Aux Axis 2-4 Circuit Function	Signal Name (X = A, B, C, or D Connector)	Faceplate Pin	Auxiliary Terminal Board	Servo Terminal Board
OUT2	Serial Encoder Req (+)	Not Used	OUT2P_X	13	13	N/C
	Serial Encoder Req (-)	Not Used	OUT2M_X	31	31	N/C
OUT3	Faceplate 5v Output (+)	Faceplate 5v Output (+)	OUT3P_X	14	14	5
	Faceplate 5v Output (-)	Faceplate 5v Output (-)	OUT3M_X	32	32	13

I/O Type: 5v Differential Outputs
 Circuit Type: Differential Totem Pole (Source / Sink)
 Output Source/Sink Current: 20 mA max
 Output Voltage: +/- 1.5 v min across 120-ohm differential load

Note: Axis 1 and Axis 3 use CMOS Drivers with 47-ohm series resistors. Axis 2 and Axis 4 use RS-422 Line Drivers.



24v DC Optically Isolated Output

Circuit Identifier	Servo Axis 1-4 Circuit Function	Aux 2-4 Axis Circuit Function	Signal Name (X = A, B, C, or D Connector)	Faceplate Pin	Auxiliary Terminal Board	Servo Terminal Board
OUT1	Faceplate 24v Output (+)	Faceplate 24v Output (+)	OUT1P_X	18	18	8
	Faceplate 24v Output (-)	Faceplate 24v Output (-)	OUT1M_X	36	36	16

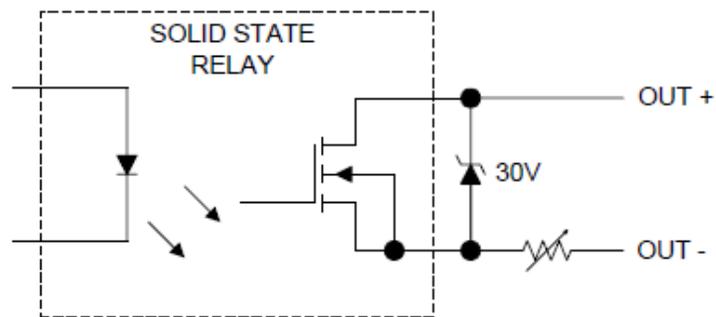
I/O Type: 24v DC Optically Isolated Output

Circuit Type: Isolated Solid-State Relay (SSR)

Output Current: 125 mA continuous, 500 mA for 10 ms (resistive or inductive)

Output Voltage Drop: 1.0 v max at 0.125 amps

Note: Output is protected by a 30v transzorb and a 0.2-amp Polyswitch. If a short circuit occurs, the output will automatically switch to a high impedance state until the load is removed. The load should not be reapplied for 60 seconds. This is a dc output and it will appear to be always ON if connections to it are reversed.



Optically Isolated Enable Relay Output

Circuit Identifier	Digital Servo Axis 1, 2 Circuit Function	Analog Servo Axis 1-4 Circuit Function	Aux Axis 2-4 Circuit Function	Signal Name (X = A, B, C, or D Connector)	Faceplate Pin	Auxiliary Terminal Board	Servo Terminal Board
ENBL	Servo MCON (+)	Drive Enable (+)	Drive Enable (+)	ENBL1_X	15	15	N/C
	Servo MCON 0v	Drive Enable (-)	Drive Enable (-)	ENBL2_X	33	33	N/C

I/O Type: Optically Isolated Enable Relay Output

Circuit Type: Isolated AC Solid State Relay (SSR)

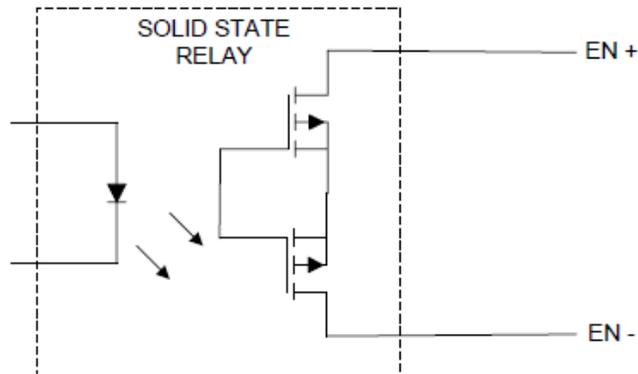
Output Current: 30 mA continuous, 50 mA for 10 ms

Output Voltage Drop: 1.0 v max at 10 mA

Note: This is a low current SSR output. The output is ON when the associated faceplate Axis Enabled LED is illuminated. This occurs when:

The servo is enabled

A Force Digital Servo Velocity %AQ Cmd is used (Axis 1, 2) A Force Analog Output %AQ Cmd is used



Differential +/- 10v Analog Inputs

Circuit Identifier	Digital Servo Axis 1, 2 Circuit Function	Analog Servo Axis 1-4 and Aux Axis 2-4 Circuit Function	Signal Name (X = A, B, C, or D Connector)	Faceplate Pin	Auxiliary Terminal Board	Servo Terminal Board
AIN1	IR Phase Current (+)	Faceplate Analog In (+)	AIN1P_X	7	7	N/C
	IR Phase Current (-)	Faceplate Analog In (-)	AIN1M_X	25	25	N/C
AIN2	IS Phase Current (+)	Faceplate Analog In (+)	AIN2P_X	8	8	N/C
	IS Phase Current (-)	Faceplate Analog In (-)	AIN2M_X	26	26	N/C

I/O Type: Differential +/- 10v Analog Inputs

Circuit Type: Differential Input

Input Impedance: 102K ohms common mode with respect to faceplate connector 0v 204K ohms differential

Maximum Input Voltage: +/- 15 v common mode with respect to faceplate connector 0v +/- 20 v differential

Resolution: 15 bits

Linearity: 13 bits

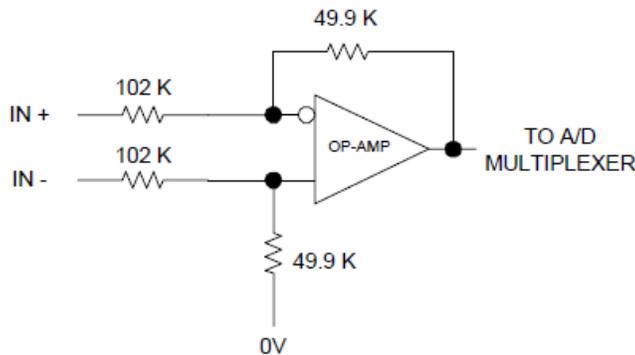
Input Offset: +/- 1.0 millivolt

Gain Factor: +/- 10.0v = +/- 32,000 counts

Gain Accuracy: +/- 0.5 %

Update Rate: 2 milliseconds + host controller sweep time when data is reported to the host controller's %AI table.

Note: Use faceplate 0v pins for common mode reference.



Single Ended +/- 10v Analog Output

Circuit Identifier	Analog Servo Axis 1-4 Circuit Function	Digital Servo Axis 1,2 and Aux Axis 2-4 Circuit Function	Signal Name (X = A, B, C, or D Connector)	Faceplate Pin	Auxiliary Terminal Board	Servo Terminal Board
AOUT1	Analog Servo Velocity Command or Analog Servo Torque Command	Faceplate Analog Out	AOUT_X	6	6	4
ACOM	Analog Out Com	Analog Out Com	ACOM_X	24	24	12

I/O Type: Single Ended Analog

Output Circuit Type: Op Amp Voltage Follower Output

Load Impedance: 2K ohms minimum

Output Current: 5 mA max

Resolution: 13 bits

Linearity: 13 bits

Output Offset Voltage: +/- 500 microvolts max

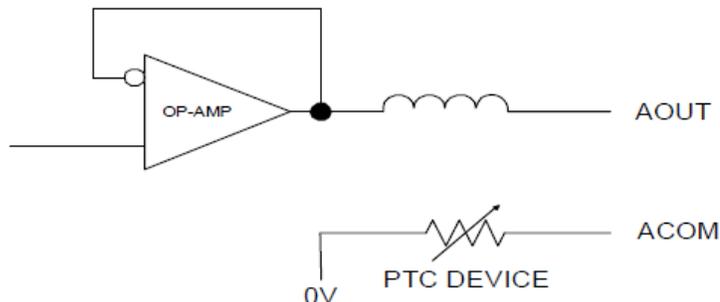
Force D/A Gain Factor: +/- 10.0v = +/- 32000 counts

Gain Accuracy: +/- 1.0 %

Force Analog Output Update Rate: Host controller sweep rate when used by Force Analog Output %AQ command.
250 microseconds when used as Digital Servo tuning output.

Note: Since this is a single ended output, it should normally drive a user device with a differential input to prevent common mode noise problems. The positive differential input should be connected to AOUT and the negative differential input to ACOM.

The Select Analog Output Mode %AQ command can be used to select the source for the analog output. Refer to Chapter 5 for more information.



+5v Power

Circuit Identifier	Servo Axis Circuit Function	Aux Axis Circuit Function	Signal Name (X = A, B, C, or D Connector)	Faceplate Pin	Auxiliary Terminal Board	Servo Terminal Board
P5V	5v Power	5v Power	P5V_X	4	4	3
0V	0v	0v	0V_X	22	22	11

I/O Type: +5V Encoder Power
 Circuit Type: +5V Power with Electronic Short Circuit Protection
 Output Voltage: 4.70 v to 5.20 v at 0.5 amp
 Output Current: 0.5 amp max (total for all connectors)

Notes:

Note: This output is intended to power external devices such as Incremental Quadrature Encoders requiring less than 0.5 amps total from all four axis connectors. The output current is provided by the host controller backplane +5v supply and is protected by an electronic short circuit protector in the DSM314 module.

The total external device current drawn from this +5V circuit must be added to the power supply consumption value in the DSM314 configuration screen in the configuration software and must be added in if performing a manual power supply loading calculation.

The listed 0v pin should normally be used as the power return signal.

Chapter 4: Configuration

This chapter describes the configuration steps necessary to set up the DSM314 for a specific application. Refer to Chapter 2 for instructions on how to configure the system to send a Jog command to the DSM to test that the system components are operable. Refer to Chapter 15 for Electronic CAM configuration information.

The DSM314 Controller is configured using the following programming software:

RX3i	Machine Edition version 4.5 or later
Series 90-30	Machine Edition version 2.1 or later VersaPro version 2.1 or later (refer to Appendix H)

Configuration is a two-part procedure consisting of:

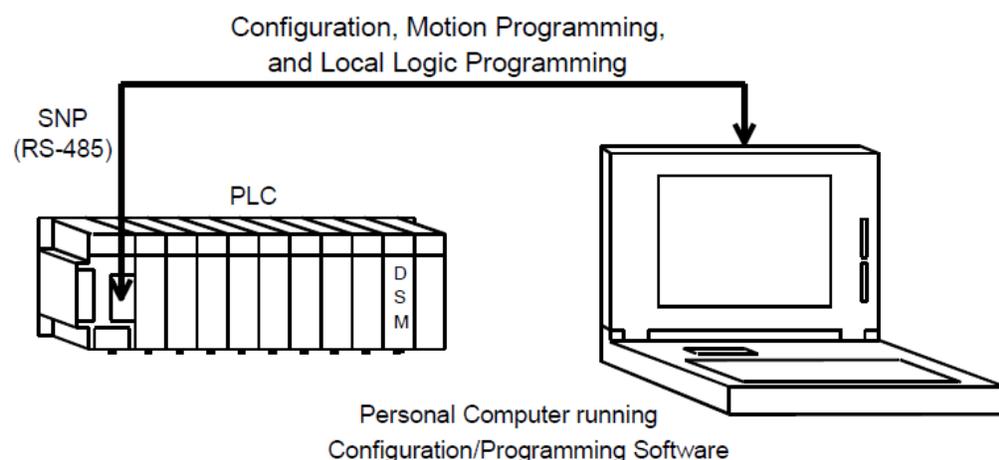
1. Rack/Slot Configuration
2. Module Configuration

4.1 Connecting the Programmer to the Host Controller

All DSM314 programming is done through the configuration/programming software interface, yielding a single point of programming for the module. For more information, please refer to the Series 90-30 PLC Installation and Hardware Manual, GFK-0356 or the PACSystems RX3i System Manual, GFK-2314. The programming environment has several communications options. One communications option is to connect the programmer directly to the host controller SNP port, as shown in the following figure. Consult the software documentation for additional communications methods.

Note: *The DSM314 also has a serial port on the module faceplate. This serial port is used only for updating the DSM314 firmware.*

Figure 54: DSM Programmer Connection Diagram

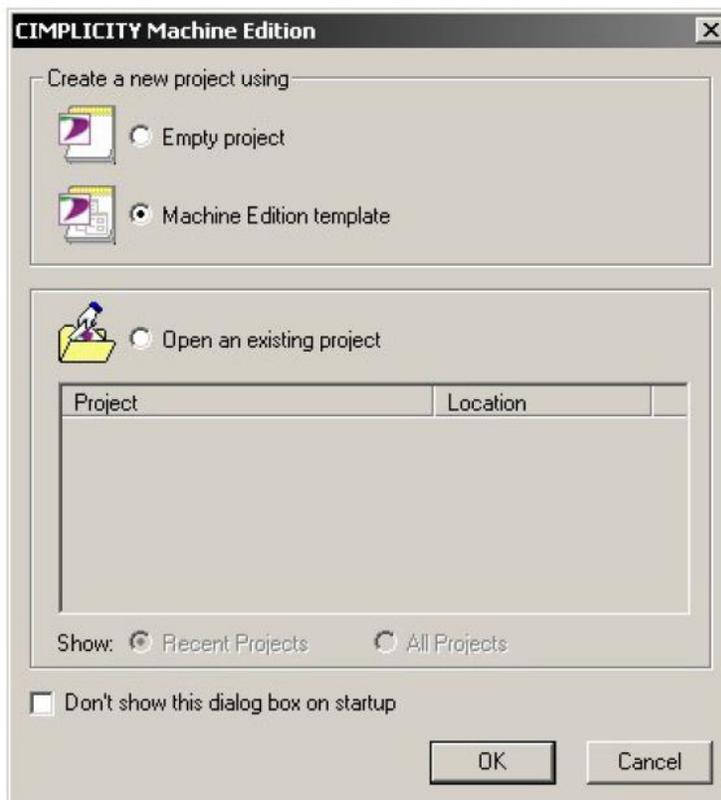


4.2 Rack/Slot Configuration

The hardware configuration defines the type and location of all modules present in the PLC racks. This is done by first completing setup screens that represent the modules in a baseplate, and saving the information to a configuration file, which is then downloaded to the PLC CPU.

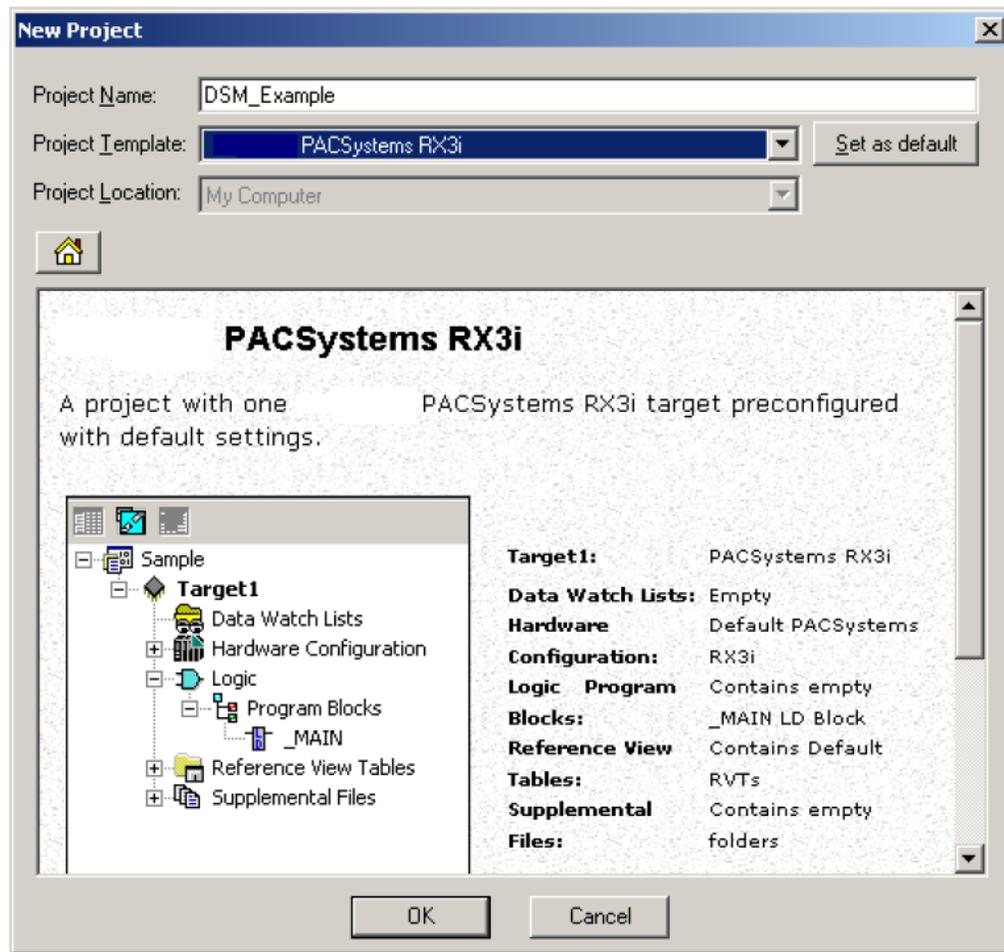
1. Start the Machine Edition Logic Developer – PLC software. The Machine Edition dialog box appears.

Figure 55



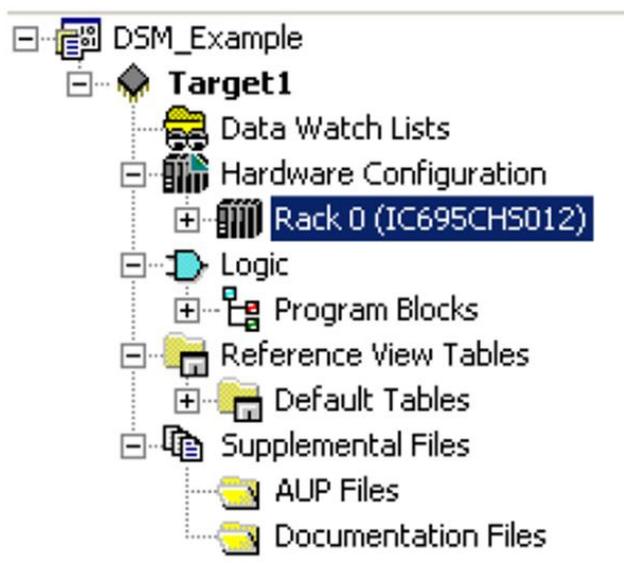
2. Under Create a New Project, choose Machine Edition Template and click OK. The New Project dialog box appears.
3. Type a name for Project Name. In the Project Template dropdown list, select Series 90-30 PLC or PACSystems RX3i. Click OK.

Figure 56



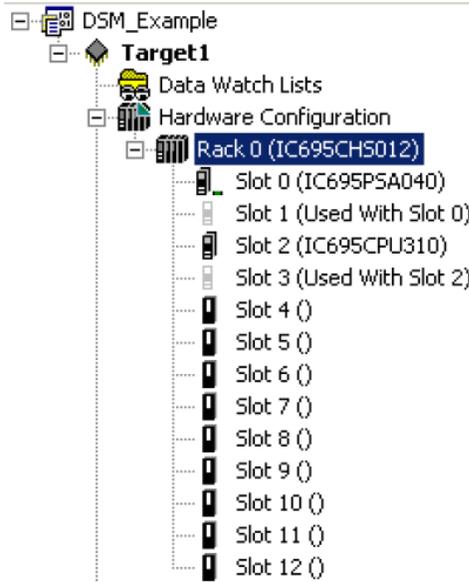
Your project appears in the Navigator window as shown in the following figure.

Figure 57



- Expand  the Main Rack node, which contains the default power supply and CPU.

Figure 58

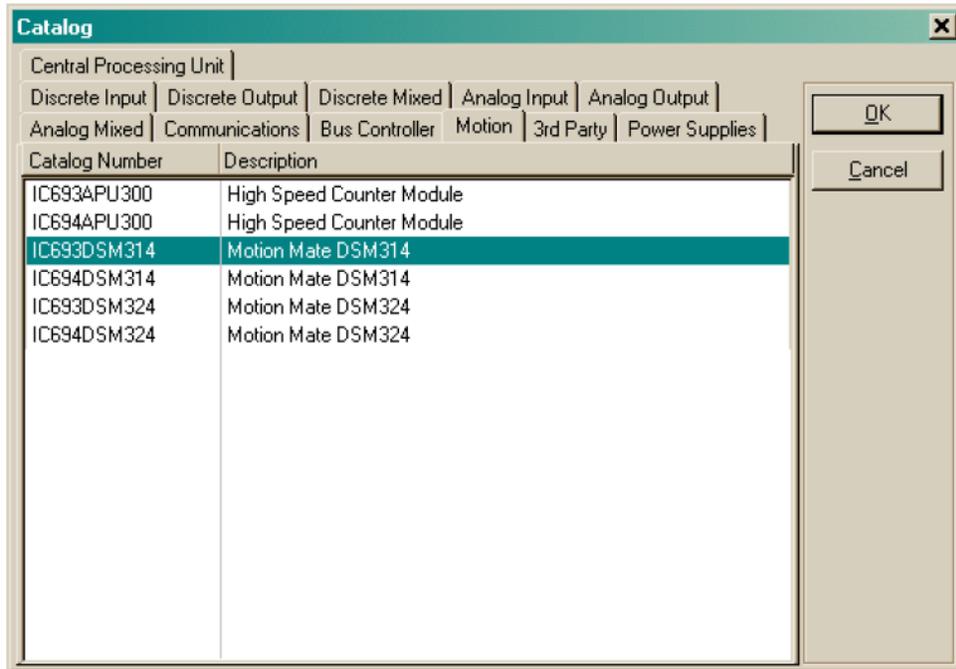


- If necessary, replace the power supply and/or CPU with the models that will be used in your application. To replace a module, right click and choose Replace Module.
- Add a DSM314 to the rack configuration.

Note: Because an IC694DSM314 module and an IC693DSM314 module have the same functionality, a Series 90-30 PLC supports them in the same way. If you install an IC694DSM314 in a Series 90-30 PLC, however, you cannot select it in Logic Developer - PLC. You must select an IC693DSM314 module and configure it as if it were an IC694DSM314.

- Right click an empty slot and choose Add Module. The Module Catalog dialog box appears.
- Select the Motion tab, choose the DSM314 and click OK.

Figure 59



This operation adds the DSM314 to the rack and displays the DSM314 configuration screens that allow you to customize the DSM314 to your application. Refer to chapter 4 for details concerning the DSM314 configuration settings.

You should complete the configuration of your host controller to include the Power Supply, Rack, CPU and additional modules to match the target system. Consult the software user’s manual and on-line help as needed.

Important

The completed configuration must be stored to the host controller. See “Storing Your Configuration to the PLC” in Chapter 2 for instructions on how to do this. For additional details, consult the software user’s manual, and on-line help.

Note: A host controller status error of “System Configuration Mismatch” with the same rack/slot location as a DSM314 indicates that there is a parameter configured and sent to the DSM314 that has been rejected by the DSM314. Carefully check each parameter of your DSM314 configuration with the configuration settings in this manual for the discrepancy. Correct the discrepancy, clear the host controller Fault, and re-Store the configuration. Check that the error has been corrected. See the next section, *Enabling Run Mode on the PLC*, for instructions on viewing and clearing PLC faults.

The DSM314 can detect many typical configuration errors. These are returned as error codes of the form Dxxx (hex) in the Module Status Code %AI word or Axis Error Code %AI words. These errors do not cause a host controller status of “System Configuration Mismatch”. Refer to Appendix A for a description of these errors. Correct any configuration errors and restore the configuration with the host controller in Stop mode.

4.3 Module Configuration

4.3.1 Setting the Configuration Parameters

The hardware configuration data is presented in a tabular format. The tabs correspond to the groupings shown below. The tab and/or tabs that correspond to the groups are shown in parenthesis after the group name. Note that tabs appear and disappear based upon the configuration selections made on the Settings tab. For example, if Axis 4 is disabled, the Axis #4 and Tuning #4 tabs are not shown.

- Module Configuration Data (Settings, CTL Bits, Output Bits)
- Serial Communication (SNP Port)
- Axis Configuration Data
- Axis Tuning
- Advanced Settings
- Power Consumption

The content of each tab is as shown below:

Tab Name	Function or Description
Settings	Contains PLC Reference assignments and lengths, DSM Axis Setup and other global data
SNP Port	DSM front panel SNP port setup
CTL Bits	Configuring the DSM's 24 control bits
Output Bits	Configuring the DSM's 8 faceplate digital outputs
Axis #1 - Axis #4	Configuring axis parameters such as Position Limits, Find Home Velocity and Jog Acceleration
Tuning #1 – Tuning #4	Configuring servo loop tuning items such as Motor Type, Position Loop Time Constant and Velocity Feedforward.
Advanced	Allows user entry of custom tuning parameters for any axis
Power Consumption	Lists DSM power required from backplane supply (4.0 watts + encoder power)

For additional details concerning the operation of the configuration software, please consult the online help or PAC Machine Edition Logic Developer-PLC Getting Started, GFK-1918.

4.3.2 Settings

The Settings tab contains configuration information that allows you to define basic module operation. These settings specify the number of controlled axes, axis operating modes etc. The selections on these tabs can cause other tabs within the configuration to appear or disappear. For example, if you disable axis #4, the Axis and Tuning tabs relating to axis #4 are not displayed.

The Settings tab is where you define the Motion program, Local Logic, and Cam block names. These names determine which programs stored in the CPU will be transferred to each DSM on system power-up.

During each CPU sweep, data is automatically transferred between the DSM314 and the CPU. The Settings tab contains the CPU interface data references and the starting locations for the automatic transfers. The configuration parameters in the Settings tab are described in Table 26. All Reference Section designations shown in the tables pertain to this chapter.

Table 26: Settings Tab

Configuration Parameter	Description	Values	Default	Units	Reference Section
Number of Axes	Number of Controlled Axis	1 2 3 4	4	N/A	1.01
%I Reference	Start address for %I ref type (80 bits)	CPU Dependent	%I00001 or next higher reference	N/A	1.02
%I Length	%I reference address length	32 = 1 Axis 48 = 2 Axis 64 = 3 Axis 80 = 4 Axis	N/A Length automatically determined by Number of Axes setting	N/A	1.02
%Q Reference	Start address for %Q ref type (80 bits)	CPU Dependent	%Q00001 or next higher reference	N/A	1.02
%Q Length	%Q reference address length	32 = 1 Axis 48 = 2 Axis 64 = 3 Axis 80 = 4 Axis	N/A Length automatically determined by Number of Axes setting	N/A	1.02
%AI reference	Start address for %AI ref type (84 bits)	CPU Dependent	%AI00001 or next higher reference	N/A	1.02
%AI Length	%AI reference address length	24 = 1 Axis 44 = 2 Axis 64 = 3 Axis 84 = 4 Axis	N/A Length automatically determined by Number of Axes setting	N/A	1.02
%AQ reference	Start address for %AQ ref type (12 bits)	CPU Dependent	%AQ00001 or next higher reference	N/A	1.02
%AQ Length	%AQ reference address length	3 = 1 Axis 6 = 2 Axis 9 = 3 Axis	N/A Length automatically determined by	N/A	1.02

Configuration Parameter	Description	Values	Default	Units	Reference Section
		12 = 4 Axis	Number of Axes setting		
Axis 1 Mode	Axis 1 Control Mode	Analog Servo Digital Servo	Analog Servo	N/A	1.03
Axis 2 Mode	Axis 2 Control Mode	Analog Servo Digital Servo Auxiliary Axis	Analog Servo	N/A	1.03
Axis 3 Mode	Axis 3 Control Mode	Analog Servo Auxiliary Axis	Auxiliary Axis	N/A	1.03
Axis 4 Mode	Axis 4 Control Mode	Disabled Analog Servo Auxiliary Axis	Disabled	N/A	1.03
Local Logic Mode	The Local Logic Engine mode	Disabled Enabled	Disabled	N/A	1.04
Total Encoder Power	Encoder power requirements	RX3i: 0.000 through 0.500 90-30: 0 through 2.5	0	RX3i: Amps 90-30: Watts	1.05
Motion Program Block Name	The motion program name to execute on the module	Name must be 20 characters or less and begin with a letter or underscore (_). Only alphanumeric characters and non-consecutive underscores are allowed.	<blank>	N/A	1.06
Local Logic Block Name	The local logic program name to execute on the module		<blank>	N/A	1.07
CAM Block Name	The CAM block name to execute on the module		<blank>	N/A	1.08
I/O Scan Set (RX3i only)	The scan set (as defined in the CPU's Scan Sets tab) to be assigned to this module.		1 through 32	1	N/A

1.01 Number of Axes. This parameter selects the number of axes the DSM314 is going to control and the size of automatic data transfers between the PLC and DSM. (Default = 4.) The following two tables document the possible axis combinations for Analog and Digital modes. Axes identified as **Limited Aux Axis** provide position feedback but no internal motion command generation.

Table 27: Number of Axes

Item	# Axes	Axis 1	Axis 2	Axis 3	Axis 4	Local Logic	Sample Rate (ms)
1.	4	Analog Servo	Analog Servo	Analog Servo	Analog Servo	Disabled	2.0
2.	4	Analog Servo	Analog Servo	Analog Servo	Auxiliary Axis	Disabled	2.0
3.	4	Analog Servo	Analog Servo	Analog Servo	Disabled	Disabled	2.0
4.	4	Analog Servo	Analog Servo	Analog Servo	Disabled	Enabled	2.0
5.	4	Analog Servo	Analog Servo	Analog Servo	Limited Aux Axis	Enabled	2.0
6.	4	Analog Servo	Analog Servo	Auxiliary Axis	Analog Servo	Disabled	2.0
7.	4	Analog Servo	Analog Servo	Auxiliary Axis	Auxiliary Axis	Disabled	2.0
8.	4	Analog Servo	Analog Servo	Auxiliary Axis	Disabled	Disabled	2.0
9.	4	Analog Servo	Analog Servo	Auxiliary Axis	Disabled	Enabled	2.0
10.	4	Analog Servo	Analog Servo	Auxiliary Axis	Limited Aux Axis	Enabled	2.0
11.	4	Analog Servo	Auxiliary Axis	Analog Servo	Analog Servo	Disabled	2.0
12.	4	Analog Servo	Auxiliary Axis	Analog Servo	Auxiliary Axis	Disabled	2.0
13.	4	Analog Servo	Auxiliary Axis	Analog Servo	Disabled	Disabled	2.0
14.	4	Analog Servo	Auxiliary Axis	Analog Servo	Disabled	Enabled	2.0
15.	4	Analog Servo	Auxiliary Axis	Analog Servo	Limited Aux Axis	Enabled	2.0
16.	4	Analog Servo	Auxiliary Axis	Auxiliary Axis	Analog Servo	Disabled	2.0
17.	4	Analog Servo	Auxiliary Axis	Auxiliary Axis	Auxiliary Axis	Disabled	2.0
18.	4	Analog Servo	Auxiliary Axis	Auxiliary Axis	Disabled	Disabled	2.0
19.	4	Analog Servo	Auxiliary Axis	Auxiliary Axis	Disabled	Enabled	2.0
20.	4	Analog Servo	Auxiliary Axis	Auxiliary Axis	Limited Aux Axis	Enabled	2.0
21.	3	Analog Servo	Analog Servo	Analog Servo	NA	Disabled	2.0
22.	3	Analog Servo	Analog Servo	Analog Servo	NA	Enabled	2.0
23.	3	Analog Servo	Analog Servo	Auxiliary Axis	NA	Disabled	2.0
24.	3	Analog Servo	Analog Servo	Auxiliary Axis	NA	Enabled	2.0
25.	3	Analog Servo	Auxiliary Axis	Analog Servo	NA	Disabled	2.0
26.	3	Analog Servo	Auxiliary Axis	Analog Servo	NA	Enabled	2.0
27.	3	Analog Servo	Auxiliary Axis	Auxiliary Axis	NA	Disabled	2.0
28.	3	Analog Servo	Auxiliary Axis	Auxiliary Axis	NA	Enabled	2.0
30.	2	Analog Servo	Analog Servo	NA	NA	Disabled	1.0
31.	2	Analog Servo	Analog Servo	NA	NA	Enabled	2.0
32.	2	Analog Servo	Auxiliary Axis	NA	NA	Disabled	1.0
33.	2	Analog Servo	Limited Aux Axis	NA	NA	Enabled	1.0
34.	1	Analog Servo	NA	NA	NA	Disabled	0.5
35.	1	Analog Servo	NA	NA	NA	Enabled	1.0

Table 28: Digital Axis Configurations

Item	# Axes	Axis 1	Axis 2	Axis 3	Axis 4	Local Logic	Sample Rate (ms)
1.	4	Digital Servo	Digital Servo	Analog Servo	Disabled	Disabled	2.0
2.	4	Digital Servo	Digital Servo	Analog Servo	Disabled	Enabled	2.0
3.	4	Digital Servo	Digital Servo	Auxiliary Axis	Disabled	Disabled	2.0
4.	4	Digital Servo	Digital Servo	Auxiliary Axis	Disabled	Enabled	2.0
5.	3	Digital Servo	Digital Servo	Analog Servo	NA	Disabled	2.0
6.	3	Digital Servo	Digital Servo	Analog Servo	NA	Enabled	2.0
7.	3	Digital Servo	Digital Servo	Auxiliary Axis	NA	Disabled	2.0
8.	3	Digital Servo	Digital Servo	Auxiliary Axis	NA	Enabled	2.0
9.	2	Digital Servo	Digital Servo	NA	NA	Disabled	2.0
10.	2	Digital Servo	Digital Servo	NA	NA	Enabled	2.0
11.	1	Digital Servo	NA	NA	NA	Disabled	2.0
12.	1	Digital Servo	NA	NA	NA	Enabled	2.0

- 1.02 I/Q/AI/AQ Len.** Displays the beginning addresses and number of %, %Q, %AI, and %AQ references assigned to the DSM314. The reference sizes are set when the user configures the number of axes.
- 1.03 Axis n Mode.** These parameters define the command output types provided to the servo sub-systems. Digital Servo selects a special digital output for Digital servo drives. If **Digital Servo** is selected, Axes 1 and 2 must be digital. **Analog Servo** selects a +/-10-volt velocity command or +/- 10-volt torque command for standard analog servo drives. The torque or velocity interface is configured by the Analog Servo Command setting in the module configuration. **Auxiliary Axis** disables the position loop so that the internal command generator and encoder position input can be used for follower or cam functions. If any axis connector is used as a master source input for follower mode, it should be configured as **Auxiliary Axis**. An **Auxiliary Axis** will output an analog voltage proportional to Commanded Velocity if Velocity Feedforward is set to a non-zero value. When an axis is configured as Auxiliary Axis and identified as Limited Aux Axis in Table 27, position feedback is available but internal motion command generation is not available. An axis configured as **Disabled** (applies to Axis 4 only) provides analog and digital i/o but no position feedback or internal motion command generation. (Default = Analog Servo (Axis 1-2), Aux Axis (Axis 3), Disabled (Axis 4)).

- 1.04 Local Logic Mode.** This parameter defines the Local Logic engine status. To enable Local Logic this parameter must be set to Enabled. If Local Logic is enabled, the maximum number of Servo axes available is 3. A **Local Logic Block Name** must also be entered when **Local Logic Mode** = Enabled. (Default = Disabled)
- 1.05 Total Encoder Power.** This parameter defines the total power consumption for all encoders attached to the DSM module. (Default = 0). This parameter should account for all analog axis and master encoders and is used to update the Power Consumption display in the configuration software.
- 1.06 Motion Program Block Name.** This parameter defines the optional Motion Program block name to execute on the DSM module. If no name is entered, the DSM will assume that Motion Program blocks are not used. If a name is entered, a Motion Program block of the same name must exist within the active folder. Entering an invalid name will cause an error to be generated when storing the hardware configuration to the PLC. The name may consist of up to 31 characters, but cannot have any blank spaces, although you are allowed to use the underline character. Both upper- and lower-case characters are permitted. (Default = <blank>).
- 1.07 Local Logic Block Name.** This parameter defines the optional Local Logic block name to execute on the DSM module. If no name is entered, the DSM will assume that Local Logic blocks are not used. If a name is entered, a Local Logic block of the same name must exist within the active folder. Entering an invalid name will cause an error to be generated when storing the hardware configuration to the PLC. The name may consist of up to 31 characters, but cannot have any blank spaces, although you are allowed to use the underline character. Both upper- and lower-case characters are permitted. For Local Logic to operate, the **Local Logic Mode** must also be set to Enabled. (Default = <blank>).
- 1.08 CAM Block Name.** Defines the optional CAM block name to execute on the DSM module. If no name is entered, the DSM will assume that CAM blocks are not used. If a name is entered, a CAM block of the same name must exist within the active folder. Entering an invalid name will cause an error to be generated when storing the hardware configuration to the PLC. The rules for CAM Block names are:
- Only the characters A-Z, a-z, 0-9, and _ (underscore symbol) are allowed. Consecutive underscores and blank spaces are not allowed.
 - The CAM block name must begin with a letter or underscore symbol.
 - A block cannot have the same name as another block that exists in an open folder.
 - A CAM block name may contain up to a maximum of seven characters.
 - This feature was first supported in DSM314 firmware release 2.00.

See Chapter 15 for CAM feature details. (Default = <blank>).

1.09 I/O Scan Set.

The scan set (as defined in the CPU's Scan Sets tab) to be assigned to this module. For details on scan set operation refer to the PACSystems CPU Reference Manual, GFK-2222.

4.3.3 Serial Communications Port Configuration Data

The DSM314's Serial Communications Port uses an RJ-11 connector labeled COMM on the module's faceplate and supports the RS-232 protocol. It is used for firmware upgrades to flash memory and must be configured properly to communicate with the upgrade software running on your programmer. Make sure the programmer's configuration parameters and the DSM314's Serial Communications Port configuration parameters match. These configuration parameters are described in Table 29.

Table 29: SNP Port Tab

Configuration Parameter	Description	Values	Defaults	Units	Ref.
Data Rate	Baud rate of SNP Port	300, 600, 1200, 2400, 4800, 9600, 19200	19200	N/A	2.01
Stop Bits	Number of stop bits	1 or 2	1	N/A	2.02
Parity	Parity	ODD, EVEN, NONE	ODD	N/A	2.03
Idle Time	Maximum link idle time	1...255	10	sec	2.04
Modem Turnaround Time	Modem turnaround time	0...255	0	.01 sec/count	2.05
SNP ID	SNP ID	Seven characters consisting of A-F and 0-9. First character must be A-F.	A000001	N/A	2.06

- 2.01 Baud Rate.** The baud rate parameter specifies the transmission rate, in bits per second, of data through the serial port.
- 2.02 Stop Bits.** All serial communications devices use at least one (1) stop bit. For slower devices, set this parameter to two (2) stop bits.
- 2.03 Parity.** Specifies whether or not a parity bit is to be used (**NONE** if not), and if so, whether it should be **ODD** or **EVEN**.
- 2.04 Idle Time.** Specifies the time, in seconds, that the DSM314 will wait for a new message to be received from the master device before assuming that communications have been lost or terminated. In such a case, the DSM314 will reinitialize to wait for the start of a new SNP connection sequence.
- 2.05 Modem Turnaround Time.** When utilizing a modem, a Modem Turnaround Time must be specified. This is the time required for the modem to start data transmission after receiving the transmit request. If no modem is used, 0 should be specified. If a modem is used, a value greater than 0 must be specified.
- 2.06 SNP ID.** An identifier consisting of from 0 to 7 characters consisting of A-F and 0-9. The first character specified must be in the set A-F. The identifier must be utilized for a multi-drop network. The DSM314 will support multi-drop connections only if the RS232 connection is converted to RS422/485.

Note: Since this Serial Communications Port is used only for upgrading the DSM314's firmware, it is recommended you leave this port's communications settings at their default values. Use cable IC693CBL316 to connect this port to the serial port of a personal computer running the firmware upgrade software.

4.3.4 Control (CTL) Bits

The CTL Bits configuration tab allows the user to configure the input source for Control Bits (CTL01-CTL24). The configuration screen allows the user to select a CTL bit configuration that corresponds with Motion Program and Local Logic program requirements. CTL Bits configuration parameters are described in Table 30. For additional information concerning CTL bit configuration, consult chapter 14.

Table 30: CTL Bits Tab

Configuration Parameter	Description	Default	Ref
CTL01 Config	CTL01 Bit Configuration	IN9_A (Axis 1 +OT)	Chapter 14
CTL02 Config	CTL02 Bit Configuration	IN10_A (Axis 1 -OT)	Chapter 14
CTL03 Config	CTL03 Bit Configuration	IN11_A (Axis 1 Home Sw)	Chapter 14
CTL04 Config	CTL04 Bit Configuration	Strobe1 Level (Axis 1)	Chapter 14
CTL05 Config	CTL05 Bit Configuration	IN9_B (Axis 2 +OT)	Chapter 14
CTL06 Config	CTL06 Bit Configuration	IN10_B (Axis 2 -OT)	Chapter 14
CTL07 Config	CTL07 Bit Configuration	IN11_B (Axis 2 Home Sw)	Chapter 14
CTL08 Config	CTL08 Bit Configuration	Strobe1 Level (Axis 2)	Chapter 14
CTL09 Config	CTL09 Bit Configuration	%Q bit Offset 12	Chapter 14
CTL10 Config	CTL10 Bit Configuration	%Q bit Offset 13	Chapter 14
CTL11 Config	CTL11 Bit Configuration	%Q bit Offset 14	Chapter 14
CTL12 Config	CTL12 Bit Configuration	%Q bit Offset 15	Chapter 14
CTL13 Config	CTL13 Bit Configuration	IN9_C (Axis 3 +OT)	Chapter 14
CTL14 Config	CTL14 Bit Configuration	IN10_C (Axis 3 -OT)	Chapter 14
CTL15 Config	CTL15 Bit Configuration	IN11_C (Axis 3 Home Sw)	Chapter 14
CTL16 Config	CTL16 Bit Configuration	Strobe1 Level (Axis 3)	Chapter 14
CTL17 Config	CTL17 Bit Configuration	%Q bit Offset 24	Chapter 14
CTL18 Config	CTL18 Bit Configuration	%Q bit Offset 25	Chapter 14
CTL19 Config	CTL19 Bit Configuration	%Q bit Offset 40	Chapter 14
CTL20 Config	CTL20 Bit Configuration	%Q bit Offset 41	Chapter 14
CTL21 Config	CTL21 Bit Configuration	%Q bit Offset 56	Chapter 14
CTL22 Config	CTL22 Bit Configuration	%Q bit Offset 57	Chapter 14
CTL23 Config	CTL23 Bit Configuration	%Q bit Offset 72	Chapter 14
CTL24 Config	CTL24 Bit Configuration	%Q bit Offset 73	Chapter 14

Each CTL bit shown in the previous table can be configured to one of the values in the following table

Table 31: Allowed Values for CTL Bits Tab

Local Logic Controlled	IN9_D (Axis 4 +OT)	Strobe2 Level (Axis4)	%Q bit Offset 57
IN9_A (Axis 1 +OT)	IN10_D (Axis 4 -OT)	%Q bit Offset 12	%Q bit Offset 72
IN10_A (Axis 1 -OT)	IN11_D (Axis 4 Home Sw)	%Q bit Offset 13	%Q bit Offset 73
IN11_A (Axis 1 Home Sw)	Strobe1 Level (Axis1)	%Q bit Offset 14	FBSA * Write Bit 1
IN9_B (Axis 2 +OT)	Strobe2 Level (Axis1)	%Q bit Offset 15	FBSA * Write Bit 2
IN10_B (Axis 2 -OT)	Strobe1 Level (Axis2)	%Q bit Offset 24	FBSA * Write Bit 3
IN11_B (Axis 2 Home Sw)	Strobe2 Level (Axis2)	%Q bit Offset 25	FBSA * Write Bit 4
IN9_C (Axis 3 +OT)	Strobe1 Level (Axis3)	%Q bit Offset 40	Local Logic Active Flag
IN10_C (Axis 3 -OT)	Strobe2 Level (Axis3)	%Q bit Offset 41	
IN11_C (Axis 3 Home Sw)	Strobe1 Level (Axis4)	%Q bit Offset 56	

* FBSA is an acronym for “Fast Backplane Status Access” (Service Request #46). See GFK-0467L or later for details.

4.3.5 Output Bits

The Output bits configuration tab allows the user to configure the DSM314 faceplate digital outputs for either Local Logic program control or PLC program control. Output Bit parameters are described in Table 32. Refer to Chapter 14 for additional information concerning Output bit configuration.

Table 32: Output Bits Tab

Configuration Parameter	Description	Values	Defaults	Ref
Out1_A Config	Out1_A Control Source	PLC Control (%Q bit Offset 24) DSM Control (Digital Output1_1)	PLC Control	Chapter 14
Out3_A Config	Out3_A Control Source	PLC Control (%Q bit Offset 25) DSM Control (Digital Output3_1)	PLC Control	Chapter 14
Out1_B Config	Out1_B Control Source	PLC Control (%Q bit Offset 40) DSM Control (Digital Output1_2)	PLC Control	Chapter 14
Out3_B Config	Out3_B Control Source	PLC Control (%Q bit Offset 41) DSM Control (Digital Output3_2)	PLC Control	Chapter 14
Out1_C Config	Out1_C Control Source	PLC Control (%Q bit Offset 56) DSM Control (Digital Output1_3)	PLC Control	Chapter 14
Out3_C Config	Out3_C Control Source	PLC Control (%Q bit Offset 57) DSM Control (Digital Output3_3)	PLC Control	Chapter 14
Out1_D Config	Out1_D Control Source	PLC Control (%Q bit Offset 72) DSM Control (Digital Output1_4)	PLC Control	Chapter 14
Out3_D Config	Out3_D Control Source	PLC Control (%Q bit Offset 73) DSM Control (Digital Output3_4)	PLC Control	Chapter 14

4.3.6 Axis Configuration Data

The DSM314 Axis configuration parameters define items such as User Units to Counts ratio, Jog Velocity, Jog Acceleration, End of Travel, and Velocity limits. The configuration parameters for each control loop mode are defined and briefly described here. The numbers in the “Ref” column refer to section reference numbers in this chapter. Values for MaxPosnUu, MaxVelUu, and MaxAccUu in the following table can be calculated using the formulas in Table 37 (“Computing Data Limit Variables”).

Table 33: Axis Configuration Data

Parameter	Description	Values	Defaults	Units	Ref
User Units	User Units Value	1...65,535	1	N/A	5.01
Counts	Feedback Counts	1...65,535	1	N/A	5.01
Overtravel Limit Sw	Over travel Limit Switch Enable / Disable	Enabled Disabled	Enabled	N/A	5.02
Drive Ready Input	Drive Ready Input Control	Enabled Disabled	Enabled	N/A	5.03
High Position Limit	High Position Limit	-MaxPosnUu ...+MaxPosnUu-1*	+8388607	User units	5.04
Low Position Limit	Low Position Limit	-MaxPosnUu ...+MaxPosnUu-1*	-8388608	User units	5.05
High Software EOT Limit	High Software End of Travel Limit	-MaxPosnUu ...+MaxPosnUu-1*	+8388607	User units	5.06
Low Software EOT Limit	Low Software End of Travel Limit	-MaxPosnUu ...+MaxPosnUu-1*	-8388608	User units	5.07
Software End of Travel	Software End of Travel Control	Disabled Enabled	Disabled	N/A	5.08
Velocity Limit	Axis Velocity Limit	1...MaxvelUu	1,000,000	User units/sec	5.09
Command Direction	Allowable Commanded Direction	Bi-directional Positive Only Negative Only	Bi- directional	N/A	5.10
Axis Direction	Axis Direction	Normal Reverse	Normal	N/A	5.11
Feedback Source	Feedback type	Default Ext Quadrature Encoder Ext Serial Encoder	Default	N/A	5.12
Feedback Mode (Digital Mode only)	Feedback Mode	Incremental Absolute	Incremental	N/A	5.13
Reversal Compensation	Reversal Compensation	0...255	0	user units	5.14
Drive Disable Delay	Drive Disable Delay	0...60,000	100	ms	5.15
Jog Velocity	Jog Velocity	1...MaxVelUu	+1000	$\frac{\text{User Units}}{\text{sec}}$	5.16
Jog Acceleration	Jog Acceleration	1...MaxAccUu*	+10,000	$\frac{\text{User Units}}{\text{sec}^2}$	5.17
Jog Acceleration Mode	Jog Acceleration Mode	Linear Scurve	Linear	N/A	5.18

Parameter	Description	Values	Defaults	Units	Ref
Home Position	Home Position	Low Position Limit ... High Position Limit	0	user units	5.19
Final Home Velocity	Final Home Velocity	1...MaxVelUu*	+500	$\frac{\text{User Units}}{\text{sec}}$	5.21
Home Offset	Home Offset Value	-32,768...+32,767	0	user units	5.20
Find Home Velocity	Find Home Velocity	1...MaxVelUu*	+2000	$\frac{\text{User Units}}{\text{sec}}$	5.22
Home Mode	Find Home Mode	Home Switch Move + Move -	Home Switch	N/A	5.23
Return Data 1 Mode	Return Data 1 Mode	0...FF	0		5.24
Return Data 1 Offset	Return Data 1 Offset	-2,147,483,648 to 2,147,483,647	0		5.24
Return Data 2 Mode	Return Data 2 Mode	0...FF	0		5.24
Return Data 2 Offset	Return Data 2 Offset	-2,147,483,648 to 2,147,483,647	0		5.24
Cam Master Source	Cam Master Source	Cmd Position 1 Actual Position 1 Cmd Position 2 Actual Position 2 Cmd Position 3 Actual Position 3 Cmd Position 4 Actual Position 4	Actual Position 3	N/A	5.25
Follower Control Loop	Follower Control Loop Enable	Disabled Enabled	Disabled	N/A	5.26
Ratio A Value	Follower A/B Ratio A	-32768...+32767	1	N/A	5.27
Ratio B Value	Follower A/B Ratio B	1...32767	1	N/A	5.27
Follower Master Source 1	Follower Master Source 1	None Cmd Position 1 Actual Position 1 Cmd Position 2 Actual Position 2 Cmd Position 3 Actual Position 3 Cmd Position 4 Actual Position 4	None	N/A	5.28
Follower Master Source 2	Follower Master Source 2	None Cmd Position 1 Actual Position 1 Cmd Position 2 Actual Position 2 Cmd Position 3 Actual Position 3 Cmd Position 4 Actual Position 4	None	N/A	5.29

Parameter	Description	Values	Defaults	Units	Ref
Follower Enable Trigger	Follower Enable Input Trigger	None CTL01-CTL32	None	N/A	5.30
Follower Disable Trigger	Follower Enable Input Trigger	None CTL01-CTL32	None	N/A	5.31
Follower Disable Action	Follower Disable Action	Stop Inc Position Abs Position	Stop	N/A	5.32
Ramp Makeup Acceleration	Follower Ramp Makeup Acceleration	1...MaxAccUu*	10,000	$\frac{\text{User Units}}{\text{sec}^2}$	5.33
Ramp Makeup Mode	Follower Ramp Makeup Mode	Makeup Time Makeup Velocity	Makeup Time	N/A	5.34
Ramp Makeup Time	Follower Ramp Acceleration Makeup Time	0...32000	0	mSec	5.35
Ramp Makeup Velocity	Follower Ramp Makeup Velocity	1...MaxVelUu*	+100,000	$\frac{\text{User Units}}{\text{sec}}$	5.36

* See Table 37 for calculating MaxAccUu, MaxPosnUu, and MaxVelUu.

- 5.01 User Units, Counts.** The User Units to Counts ratio sets the number of programming units for each position feedback count. This allows the user to program the DSM314 in application-specific units. The User Units and Counts values must be within the range of 1 to 65,535. The User Units to Counts ratio must be within the range of 8:1 to 1:32. For example, if there is 1.000 inch of travel for 8192 feedback counts, a 1000:8192 User Units: Counts ratio sets 1 User Unit equal to 0.001 inch. Default is 1:1.

The User Units to Counts ratio sets the number of position programming units for each feedback count. It is a requirement to set this value correctly for the mechanical systems coupled to the axis, otherwise movement to unsafe and inaccurate positions may occur.

Note: *It is important to set this relationship at the beginning of the configuration session; most other configuration fields are specified in user units.*

For example, Velocity will be specified in user units per second and Acceleration will be specified in user units per second per second.

This ratio is a very powerful scaling feature. A User Unit to Counts ratio can be configured to allow programming in other than default counts. In a simplified example, suppose an encoder feedback application has an encoder that produces 1,000 quadrature counts per revolution (250 lines) and is geared to a machine that produces one inch per revolution. The default unit would be one thousandth of an inch per count. However, you may want to write programs and use the DSM300 Series module with metric units. A ratio of 2540 User Units to 1000 Counts can be configured to allow this. With this ratio, one user unit would represent .01 millimeters. 2540 user units would represent 25.40 millimeters (one inch) of travel.

The example below illustrates how to meet the requirements that the User Units and Counts values be within the range of 1 to 65,535, and the User Units to Counts ratio be within the range of 8:1 to 1:32.

The basic equation to satisfy is:

$$\frac{\text{User Units}}{\text{Counts}} = \frac{(\text{Load Movement per Motor Rotation}) \div (\text{Desired User Units Resolution})}{\text{Encoder Counts per Motor Rotation}}$$

The numerator and denominator must each fit within the RANGE limits. The reduced fraction must fit between the RATIO limits. The decimal point is always implied, not used. The User Units to Counts ratio is always expressed as an integer ratio.

Sample Application

Use the User Units to Counts ratio to configure the DSM314 so you can program in engineering units rather than encoder counts. As an example, assume a machine has a motor with a motor-mounted quadrature encoder connected through a gear reducer to a spur gear. The spur gear is mounted to the end of a pinch-roller shaft. The pinch roller feeds sheet material for a cut-to-length application. The motion program will specify the length of cut sheets. The programmer wishes to program in 0.01-inch resolution.

The following data is given:

- 2000-line encoder (x4 = 8000 counts per encoder revolution)
- 20:1 gear reduction
- 14.336-inch pitch diameter spur gear
- Inch desired programming unit (.01)

Although several approaches are possible, the most straightforward is to base the calculations on a single spur gear revolution.

1. First determine the number of User Units per spur gear revolution:

$$14.336\text{-inch pitch diameter} * \pi (\text{pi}) = 45.0378 \text{ inches circumference}$$

$$45.0378 \text{ inches} / 0.01\text{-inch desired programming units} = 4503.78 \text{ User Units per revolution of spur gear}$$

2. Then determine the number of encoders counts per spur gear revolution:

$$2000 \text{ lines} * \frac{4 \text{ counts}}{\text{line}} * \frac{20 \text{ motor revs.}}{1 \text{ gear rev.}} = 160,000 \text{ encoder counts per spur gear revolution}$$

3. Then check the value of the User Units to Counts ratio. The ratio must be in the 8:1 to 1:32 (8 to 0.03125) range and the two numbers must be in the 1 to 65535 range.

$$4503.78 \text{ User Units} / 160,000 \text{ encoder counts} = 0.02815 \text{ or } 1:35.5$$

This ratio is too small, so something must be changed. Any of the following system components could be changed to solve the problem:

- Change the spur gear diameter to 15.92 inch or larger
- Change the encoder lines per revolution to 1800 or less
- Change the gear reduction to 18:1 or less
- Change the desired programming unit to 0.001 inch

By far, the easiest component to change is the desired programming unit to 0.001 inch.

4. Recalculate to determine the revised User Units per revolution using 0.001-inch programming unit.

$$14.336 \text{ inches diameter} * \pi = 45.0378 \text{ inches circumference}$$

$$45.0378 \text{ inches} / 0.001\text{-inch programming unit} = 45,037.8 \text{ User Units per revolution of spur gear}$$

Thus, the User Units to Counts ratio is $45,038 / 160,000 = 0.2815$ or about 1:3.6, which is within the valid ratio range.

So, a $45,038 / 160,000$ ratio would be used except that 160,000 is larger than the maximum 65,535 range value. Dividing both numbers by 10 solves this to make the ratio $4,504 / 16,000$. Note that in the

above example, we simply reduced the fraction and ignored the slight rounding error

One method of avoiding “rounding” is to express the numeric ratio as a fraction. From the previous example, any number set that produced a 0.2815 ratio could be used. An example is 2815 / 10000.

Another approach is to rationalize the fraction (reduce it to its lowest terms). This is done by evenly dividing both the numerator and denominator by successively smaller prime numbers, beginning with the largest prime that will evenly divide into both the numerator and the denominator, until no more division without remainders is possible.

Always maintain an exact integer fraction, a decimal ratio expressed as a fraction, or a rationalized fraction when configuring the User Units to Counts ratio for the best accuracy. The user must determine if the rounding error, if present, is of significance. A rotary mode application that always operates in one direction will accumulate rounding errors over time and “drift”. A linear application will only accumulate error for the length of travel then “rewind” as the axis reverses.

5.02 Overtravel Limit Switch. Selects whether the DSM300 Series module uses the hardware over travel limit switch inputs.

DISABLED, the faceplate overtravel inputs (IN09 and IN10) may be used as general-purpose motion program flow control and program branching inputs (assigned to CTL01- CTL24).

ENABLED, indicates that the DSM300 will check the axis over travel inputs continuously, every 10 milliseconds whenever the %I Drive Enabled input is true. If either limit switch opens (the input goes to logic zero, Off) all motion is immediately commanded to stop. No deceleration control is active; the servo velocity command is set to zero. The solid-state axis enable relay will not open until after the %Q Enable Drive command is set to zero. An error code indicating which limit is tripped is reported to the %AI Axis Error Code. At this point, only one DSM314 action is allowed: the appropriate %Q Jog and %Q Clear Error bits may be used simultaneously to back away from the Limit Switch. The %Q Clear Error bit must be maintained ON to Jog off the limit switch. The user may also manually move the disabled axis off the limit switch. After the alarm is cleared, normal operation may resume.

CAUTION

Force D/A commands ignore the limit switches and should be used with caution.

- 5.03 Drive Ready Input.** Enables or disables the Drive Ready input for Analog Servos. This configuration item is ignored for a Digital Servo or Auxiliary axis. If the Drive Ready input is enabled, the Drive Ready faceplate input signal (IN4) must be turned on (set to 0v) **within 1 second** after the Enable Drive %Q bit is turned on. If the Drive Ready faceplate input is turned off while the Drive Enabled %I bit is on, error code C0h will be reported and the axis will stop. The Drive Ready Input configuration should be set to Disabled for Analog Servos that do not provide a compatible Drive Ready output signal.
- 5.04 High Position Limit.** (User Units). When moving in the positive direction, the Actual Position will roll over to the low limit when this value is reached. **The Position Limits can be used for continuous rotary applications when the Software End of Travel configuration is set to Disabled.** The High Position Limit should always be set one User Unit smaller than the desired cycle. For example, a 360° machine would have a High Position Limit setting of 359. At the next count past 359, the count would roll over to the value set in the Low Position Limit parameter (0 in this example). **For proper operation, the rollover modulus (High Position Limit - Low Position Limit +1) must always be greater than the distance traveled by the axis in one position loop sample time (normally 2 ms).** See Appendix C for considerations when using an absolute mode encoder. Default: 8,388,607.
- 5.05 Low Position Limit.** Low Pos Limit (User Units). When moving in the negative direction, the Actual Position will roll over to the high limit when this value is reached. **The Position Limits can be used for continuous rotary applications when the Software End of Travel configuration is set to Disabled. For proper operation, the rollover modulus (High Position Limit - Low Position Limit +1) must always be greater than the distance traveled by the axis in one position loop sample time (normally 2 ms).** See Appendix C for considerations when using an absolute mode encoder. Default: -8,388,608.
- 5.06 High Software EOT Limit.** High Software End of Travel Limit (User Units). If the limit is enabled and the DSM314 is programmed to go to a position greater than the High Software EOT value, an error will result and the DSM314 will not allow axis motion. If the Follower control loop is enabled, the High Software EOT Limit is ignored for slave axis motion resulting from master axis commands. The limit only applies to slave axis motion resulting from internally generated jog and motion program commands. **The limit is always ignored for Move at Velocity %AQ commands.** Default: +8,388,607.

In Analog or Digital Servo modes, the High Software EOT limit is used only when the **Software End of Travel** configuration is set to Enabled. If the High Software EOT Limit is enabled and its value is more positive than the High Position Limit, the High Software EOT Limit will internally be set equal to the High Position Limit. Axis error code 17h will also be reported, indicating that the limit has been adjusted. **The High Software EOT Limit is ignored for Jog commands if the Position Valid %I bit is off.**

In Auxiliary Axis mode, the High Software EOT limit has separate purposes depending on the setting for Software End of Travel:

Software End of Travel set to Enabled - Motion Programs and Jog commands are restricted to the High Software EOT Limit value. A Move at Velocity %AQ Command can cause Commanded Position to exceed the EOT limit. Commanded Position will roll over at the maximum positive and negative position values (-2,147,483,648 ...+2,147,483,647 at 1:1 scaling).

Software End of Travel set to Disabled - The High Software EOT Limit is used as the rollover value for Commanded Position. Motion Program, Jog and Move at Velocity commands will all cause Commanded Position to roll over at the High Software EOT Limit.

- 5.07 Low Software EOT Limit.** Low Software End of Travel Limit (User Units). If the limit is enabled and the DSM314 is programmed to go to a position less than the Low Software EOT, an error will result and the DSM314 will not allow axis motion. If the Follower control loop is enabled, the High Software EOT Limit is ignored for slave axis motion resulting from master axis commands. The limit only applies to slave axis motion resulting from internally generated jog and motion program commands. **The limit is always ignored for Move at Velocity %AQ commands.** Default: -8,388,608

In Analog or Digital Servo modes, the Low Software EOT limit is used only when the **Software End of Travel** configuration is set to Enabled. If the Low Software EOT Limit is enabled and its value is more negative than the Low Position Limit, the Low Software EOT Limit will internally be set equal to the Low Position Limit. Axis error code 17h will also be reported, indicating that the limit has been adjusted. **The Low Software EOT limit is ignored for Jog commands if the Position Valid %I bit is off.**

In Auxiliary Axis mode, the Low Software EOT limit has separate purposes depending on the setting for **Software End of Travel**:

Software End of Travel set to Enabled - Motion Programs and Jog commands are restricted to the Low Software EOT Limit value. A Move at Velocity %AQ Command can cause Commanded Position to exceed the EOT limit. Commanded Position will roll over at the maximum positive and negative position values (-2,147,483,648 ...+2,147,483,647 at 1:1 scaling).

Software End of Travel set to Disabled - The Low Software EOT Limit is used as the rollover value for Commanded Position. Motion Program, Jog and Move at Velocity commands will all cause Commanded Position to roll over at the Low Software EOT Limit.

- 5.08 Software End of Travel.** Enables or disables the High Software EOT Limit and Low Software EOT Limit. Default: Disabled
- 5.09 Velocity Limit.** Axis Velocity Limit (User Units/sec). The Velocity Limit applies to the sum of all velocity command sources for an axis, including the internal path generator and external follower master axis commands. If a servo velocity command exceeds the limit, **error code F2h will be reported** and the servo command will internally be set to the limit value. Default: 1,000,000

- 5.10 Command Direction.** Allows an axis to be configured for unidirectional or bi-directional operation. If unidirectional operation is selected (Positive Only or Negative Only), servo commands in the opposite direction will not be sent to the servo position loop. Default: Bi-directional
- 5.11 Axis Direction** For all digital servos, a configured axis direction of Normal defines the positive axis direction as counterclockwise (CCW) motor shaft rotation when viewed looking into the motor shaft. A configured axis direction of Reverse defines the positive axis direction as clockwise (CW) shaft rotation.
- For analog servos, a configured axis direction of Normal defines the positive axis direction as encoder channel A leading channel B. A configured axis direction of reverse defines the positive axis direction as encoder channel B leading channel A. In practice, the axis direction configuration allows the user to easily reverse the motion caused by all commands without having to change the motion program. Default: Normal
- 5.12 Feedback Source.** This configuration item is unused in the present DSM314 firmware. It must be set to Default.
- 5.13 Feedback Mode.** Only used when the Axis Mode is set to Digital Servo. This item configures Incremental or Absolute feedback type for the serial encoder. Incremental means the serial encoder is being used as an incremental encoder and encoder battery alarms will not be reported. Absolute means the serial encoder is being used as an absolute encoder (encoder backup battery installed), which maintains position if system power is cycled. In Absolute mode, encoder battery alarms will be reported. See appendix C, Position Feedback Devices, for more information. Default: Incremental
- 5.14 Reversal Compensation.** A compensation factor that allows the servo to reverse direction and still provide accurate positioning in systems exhibiting backlash. Backlash is exhibited by a servomotor that must move a small amount (lost motion) before the load begins moving when direction is reversed. For example, consider a dead bolt door lock. Imagine the servo controls the key in the lock and the feedback reports bolt movement. When the servo turns the key counterclockwise, the bolt moves left. However, as the servo turns the key clockwise, the bolt does not move until the key turns to a certain point. The Reversal Compensation feature adds in the necessary lost motion to quickly move the servo to where motion will begin on the feedback device. The DSM314 removes the compensation distance when a move in the negative direction is commanded and adds the compensation distance before a move in the positive direction. Default: 0.

Note: *Reversal compensation is not available if the **Follower Control Loop** configuration is set to Enabled.*

- 5.15 Drive Disable Delay.** Servo Drive Disable Delay (milliseconds). The time delay from the time the zero-velocity command is received until the drive enable (digital servo MCON) signal switches off. Disable Delay is effective when the Enable Drive %Q bit is turned off or certain error conditions (Stop Mode) occur. Disable Delay should be longer than the worst-case deceleration time of the servo from maximum speed. Because turning OFF the Enable Drive %Q bit stops the DSM314 from commanding the servo, there are times when the drive enable signal should stay ON. For example, if the servo runs into an End of Travel Limit and the drive enable signal was immediately turned OFF due to the error, the servo may continue moving until it coasted to a stop. Thus, to allow the DSM314 to command and control a fast stop, the Drive Disable Delay should be longer than the deceleration time of the servo from maximum speed.

The disable delay may be used to control when torque is removed from the motor shaft. Applications using an electro-mechanical brake generally need time for the brake to engage prior to releasing servo torque. The delay should be set to a value longer than the engagement time for the brake. Default: 100.

- 5.16 Jog Velocity.** Jog Velocity (User Units/second). The velocity at which the servo moves during a Jog operation. Jog Velocity is used by motion programs when no Velocity command is included in the program. Jog Velocity is always used by the %AQ Move Command (27h). Default: 1000.
- 5.17 Jog Acceleration.** Jog Acceleration Rate (User Units/second/second). The acceleration and deceleration rate used during Jog, Find Home, Move at Velocity, Abort All Moves and **Normal Stop** operations. A **Normal Stop** occurs when the PLC switches from Run to Stop or after certain programming errors (refer to Appendix A). Jog Acceleration is used by motion programs when no Acceleration command is included in the program. Jog Acceleration is always used by the %AQ Move Command (27h). The value of Jog Acceleration should be set high enough to perform satisfactorily during Abort all Moves and Normal Stop operations. Default: 10000.

Note: A minimum value after scaling is used in the DSM314. This value is determined by the rule: $\text{Jog Acc} * (\text{user units/counts}) \geq 32 \text{ counts/sec/sec}$.

- 5.18 Jog Acceleration Mode.** Jog Acceleration Mode (LINEAR or S-CURVE). The acceleration mode for Jog, Find Home, Move at Velocity, Abort All Moves and **Normal Stop** operations. A **Normal Stop** occurs when the PLC switches from Run to Stop or after certain programming errors (refer to Appendix A). LINEAR (constant acceleration) causes commanded velocity to change linearly with time. S-CURVE (jerk limited acceleration) causes commanded velocity to change more slowly than the linear mode at the beginning and end of acceleration intervals. **Motions using S-Curve acceleration require twice the time and distance to change velocity compared to motions using the same acceleration value with Linear acceleration.** In order to maintain equal machine cycle times, an S-Curve motion profile requires an acceleration value (and peak motor torque) twice as large as the equivalent Linear acceleration motion profile. Therefore, a tradeoff between motor cost and machine cycle time may be necessary. Default: LINEAR.

- 5.19 Home Position.** Home Position (User Units). The value assigned to Commanded Position when a Find Home cycle completes.
- 5.20 Home Offset.** Home Position Offset (User Units). A value added to or subtracted from the servo's final stopping point when a Find Home cycle completes. **Home Offset** adjusts the final servo stopping point relative to the encoder marker. See chapter 6 for details on the home cycle. Default: 0.
- 5.21 Final Home Velocity.** Final Home Velocity (User Units/second). The velocity at which the servo seeks the final Home Switch transition and Encoder Marker pulse at the end of a Find Home cycle. This velocity is also used for the home cycle MOVE+ and MOVE- modes. See chapter 6 for details on the home cycle. Final Home Velocity must be slow enough to allow a 10 millisecond (filter time) delay between the final Home Switch transition and the Encoder Marker pulse. Default: 500
- 5.22** servo seeks the initial Home Switch transitions during the Find Home cycle when the Home Mode is configured for HOMESW. If desired, Find Home Velocity can be set to a high value to allow the servo to quickly locate the Home Switch. Default: 2000
- 5.23 Home Mode.** Find Home Mode. The method used to find home during a Find Home cycle. HOME SWITCH indicates that a Home Switch is to be monitored to Find Home. MOVE+ and MOVE- specify direct positive and negative movement to the next encoder marker at the Final Home Velocity. See chapter 6, "Non-Programmed Motion," for details on the Home Cycle, Home Switch, move+, and Move- Modes. Default: HOMESW.
- 5.24 Return Data 1 Mode and Offset, Return Data 2 Mode and Offset.** These configuration parameters allow alternate data to be reported in the User Selected Data 1 and User Selected Data 2 %AI location for each axis. The alternate data includes information such as Parameter memory contents and the DSM314 Firmware Revision.

There are two Return Data configuration parameters, a mode selection and an offset selection. The mode parameter selects the Return Data type. The offset parameter is only used when the Parameter Data mode (18h) is selected. Mode default = 0 (Torque Command). Offset default = 0. The following Return Data selections are allowed:

Table 34: User Selected Return Data

Digital	Analog Torque	Analog Velocity	Selected Return Data	Data Mode	Data Offset
Y	Y	N	Torque Command	00h	not used
Y	Y	Y	DSM Firmware Revision	10h	not used
Y	Y	Y	DSM Firmware Build ID No. (hex)	11h	not used
Y	N	N	Absolute Feedback Offset (cts)	17h	not used
Y	Y	Y	Parameter Data	18h	Parameter Number (0–255)
Y	Y	Y	CTL bits 1-32	19h	not used
Y	Y	Y	Analog Inputs - Axis 1	1Ch	not used
Y	Y	Y	Analog Inputs - Axis 2	1Dh	not used
Y	Y	Y	Analog Inputs - Aux 3	1Eh	not used
Y	Y	Y	Analog Inputs - Aux 4	1Fh	not used
Y	Y	Y	Commanded Position (user units)	20h	not used
Y	Y	Y	Follower Program Command Position (cts)	21h	not used
Y	Y	Y	Unadjusted Actual Position (cts)	28h	not used
Y	Y	Y	Unadjusted Strobe 1 Position (cts)	29h	not used
Y	Y	Y	Unadjusted Strobe 2 Position (cts)	2Ah	not used

Torque Command is scaled so that +/- 10000 = +/- 100% torque.

DSM Firmware Revision is interpreted as two separate words for major-minor revision codes.

DSM Firmware Build ID is interpreted as a single hex word.

Absolute Feedback Offset is the position offset (in counts) that is used to initialize Actual Position when a digital Absolute Encoder is used. Actual Position = Absolute Encoder Data + Absolute Feedback Offset.

Analog Inputs provides two words of data for each axis: low word = AIN1 and high word = AIN2. The data is scaled so that +/- 32000 = +/- 10.0v.

Commanded Position (user units) is a copy of the Commanded Position %AI data reported for each axis. Refer to Chapter 5.

Follower Program Command Position (cts) is the active commanded position (in feedback counts) updated and used by the internal motion command generator. Refer to Chapter 9 - Combined Follower and Commanded Motion.

Unadjusted Actual Position is the accumulated actual position (in counts, not user units) with a 32-bit binary rollover value of -2,147,483,648 ... +2,147,483,647. A Find Home or Set Position command sets the Unadjusted Actual Position to a value equal to the %AI Actual Position data scaled to counts. For details on the operation of Unadjusted Actual Position, refer to “Return Data” in Chapter 5.

Unadjusted Strobe 1 Position is the value of Unadjusted Actual Position captured when a Strobe 1 input occurs.

Unadjusted Strobe 2 Position is the value of Unadjusted Actual Position captured when a Strobe 2 input occurs.

At least three PLC sweeps or 10 milliseconds (whichever represents more time) must elapse before the new Selected Return Data is available in the PLC.

- 5.25 **Cam Master Source.** This configuration item is unused in the present DSM314 firmware.
- 5.26 **Follower Control Loop.** When this configuration item is set to Enabled, the servo axis will follow a master axis input in addition to the standard internally generated motion functions. Default: Disabled
- 5.27 **Ratio A Value and Ratio B Value.** (Follower Control Loop must be Enabled) The A over B ratio sets the follower slave/master gear ratio.

$$\text{Follower Axis Motion (counts)} = \frac{\text{A * Master Reference Counts}}{\text{B}}$$

The range for A is -32,768 to +32,767 and B is 1 to +32,767. When A is negative, the slave axis will move in the opposite direction from the master. The DSM firmware supports A/B slave/master follower ratios in the ranges of 32:1 to 1:10,000. Default: 1:1.

- 5.28 **Follower Master Source 1.** (Follower Control Loop must be Enabled.) Configures follower Master Axis Source 1. Allowed choices are Commanded or Actual Position for any of the 4 axes (as long as it's a configured axis). Follower Master Source 1 is active when the Follower Master Source Select %Q bit is OFF.

Cmd Position or Actual Position of a slave axis should not be selected as a master source for that axis. If an unconfigured axis is selected for Follower Master Source 1, it will be ignored. Default: None. Refer to Chapter 8 for information on follower mode.

- 5.29 **Follower Master Source 2.** (Follower Control Loop must be Enabled.) Configures follower Master Axis Source 2. Allowed choices are Commanded or Actual Position for any of the 4 axes (as long as it's a configured axis). Follower Source 2 is active when the Follower Master Source Select %Q bit is ON.

Cmd Position or Actual Position of a slave axis should not be selected as a master source for that axis. If an unconfigured axis is selected for Follower Master Source 2, it will be ignored. Default: None. Refer to Chapter 8 for information on follower mode.

5.30 Follower Enable Trigger. Follower Enable Trigger Input. Selects the control bit, CTL01-CTL32, to be used as the Follower Enable trigger input. The follower axis is enabled when the selected trigger input transitions ON **and** the Enable Follower %Q bit is also ON. After Follower is enabled, the PLC Enable Follower %Q bit and an optional Follower Disable trigger bit controls the active state of the following function. None means the follower axis is enabled only by the Enable Follower %Q bit. Default: None.

5.31 Follower Disable Trigger. Follower Disable Trigger Input. Selects the control bit, CTL01-CTL32, to be used as the Follower Disable trigger input. The trigger input is tested only when the Enable Follower %Q bit is ON. When the Enable Follower %Q bit is ON, an OFF to ON transition of the trigger bit will disable the follower. Turning OFF the Enable Follower %Q bit immediately disables the follower, regardless of the disable trigger configuration. Default: None.

5.32 Follower Disable Action. Stop means the follower will immediately decelerate to zero velocity at the configured Follower Ramp Acceleration rate. Inc Position means the follower will continue at its present velocity, then decelerate and stop after a specified distance has elapsed. The incremental distance is specified in a parameter register for each axis:

P227 = Axis 1 Incremental distance

P235 = Axis 2 Incremental distance

P242 = Axis 3 Incremental distance

P250 = Axis 4 Incremental distance

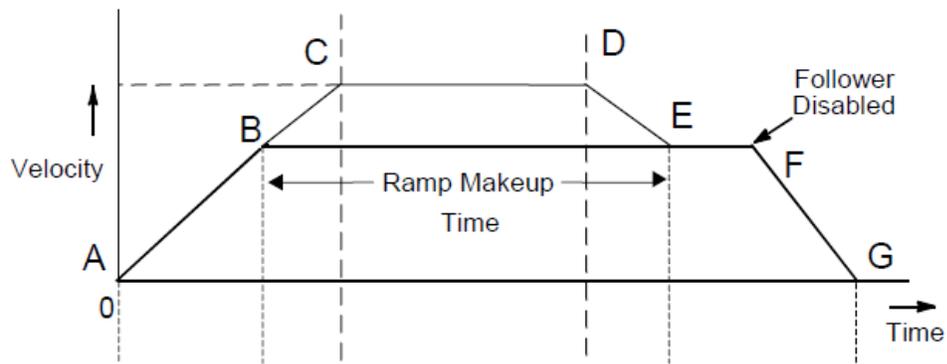
The incremental distance represents the total actual position change that will occur from the point where the follower is disabled until it stops.

A configuration of Abs Position is not supported in the present DSM314 firmware.
Default: Stop

5.33 Ramp Makeup Acceleration. Follower Ramp Makeup Acceleration (uu/sec^2). Specifies the acceleration used to:

- Accelerate the follower axis to match master velocity after the follower is enabled (sector AB in Figure 60),
- Make up the master command counts lost during follower acceleration (sector BC and DE in Figure 60),
- Decelerate to a stop after the follower is disabled (sector FG in Figure 60).

Figure 60: Velocity profile during the follower ramp cycle



5.34 Ramp Makeup Mode. Choices are Makeup Time or Makeup Velocity, explained below.

- **Makeup Time Mode** – in this mode the makeup process takes the amount of time specified by Ramp Makeup Time parameter (refer to Figure 60). This is the default mode.
- **Makeup Velocity Mode** – This mode is reserved for future use.

5.35 Ramp Makeup Time. Follower Acceleration Ramp Makeup Time (milliseconds). Specifies the time in milliseconds used to make up the master command counts lost during a follower acceleration ramp. If the distance correction is not possible in the configured makeup time (because the value is too small) then the correction time is longer, and a warning error is reported. This setting only has an effect when the Ramp Makeup Mode is set to Makeup Time.

If an acceleration ramp without any correction for lost counts is desired, Makeup Time should be set to 0. In this case, the motor will synchronize velocity relative to the master, but will not attempt to correct for any positional deviation that occurs while the follower axis is accelerating.

Makeup time has a minimum value of 10, so for values entered in the range of 1...10 use 10 instead.

Default: 0.

Refer to Chapter 8, Follower Motion, Follower Axis Acceleration Ramp Control section, for a much more detailed discussion of this feature

5.36 Ramp Makeup Velocity. This field is reserved.

4.3.7 Tuning Data

The DSM314 Tuning tabs are used to configure Servo axis tuning data. Parameters such as Motor Type, Velocity at Max Cmd, Velocity Feed Forward Percentage, and Position Loop Time Constant are configured in these tabs. From one to four Tuning tabs may appear in the DSM314 configuration window, one tab for each Servo axis configured in the Settings tab.

The numbers in the “Ref” column of the table below refer to item numbers in this chapter.

Table 35: Tuning Tab Items

Configuration Parameter	Description	Values	Defaults	Units	Ref
Motor Type	Motor Type	0...65535	0	N/A	6.01
Analog Servo Command	Analog Servo Command Type	Velocity Torque ^(Note 1)	Velocity	N/A	6.02
Position Error Limit	Position Error Limit	100...60,000	60,000	User Units	6.03
In Position Zone	In Position Zone	1...60,000	10	User Units	6.04
Pos Loop Time Constant	Position Loop Time Constant	0...65535	1000	0.1 mSec	6.05
Velocity at MaxCmd	Velocity at Maximum Command	256.. MaxVelUu ^(Note 2)	100,000	User Units	6.06
Velocity Feed Forward Percentage	Velocity Feed Forward Percentage	0...12000	0	.01%	6.07
Acceleration Feed Forward Percentage	Acceleration Feed Forward Percentage	0...12000	0	.01%	6.08
Integrator Mode	Position Loop Integrator Mode	Off Continuous Servo Null	Off	N/A	6.09
Integrator Time Constant	Position Loop Integrator Time Constant	0...10000	0	mSec	6.10
Velocity Loop Gain	Velocity Loop Gain	0...65535	16	N/A	6.11

Note:

- Torque Mode is supported in DSM firmware version 3.0 or later
- See Table 37 for calculating MaxVelUu.

6.01 Motor Type. Selects the type of AC servomotor to be used with the DSM314 in Digital Mode ONLY. The DSM314 internally stores setup motor parameter tables for each of the motors supported. A motor type of 0 disables digital servo control by the DSM314 for the digital servo axis. **Motor type must be set to 0 when no digital servo is attached if any %Q bit commands or %AQ data commands will be sent to the axis.** Supported Motor types are listed in the tables below.

The Motor Type must be 0 for ANALOG Mode or if no motor is attached to the axis. Default: 0.

Motor part numbers are used to determine the proper Motor type code and are in the form ZA06B-xxxx-yyyy, where xxxx represents the motor specification field. For example: When reading a motor number from a motor label of ZA06B-0032-B078, the motor specification digits 0032 indicate the motor model of β 2/3000. The β

Series table references the Motor Type Code (36) needed for the configuration field. Supported Motor types are listed in the tables below. The list of supported motors may be expanded in future releases.

α Series Servo Motor

Motor Type Code	Motor Model	Motor Specification
61	α 1/3000	0371
46	α 2/2000	0372
62	α 2/3000	0373
15	α 3/3000	0123
16	α 6/2000	0127
17	α 6/3000	0128
18	α 12/2000	0142
19	α 12/3000	0143
27	α 22/1500	0146
20	α 22/2000	0147
21	α 22/3000	0148
28	α 30/1200	0151
22	α 30/2000	0152
23	α 30/3000	0153
30	α 40/2000	0157
29	α 40/FAN	0158

α L Series Servo Motor

Motor Type Code	Motor Model	Motor Specification
56	α L3/3000	0561
57	α L6/3000	0562
58	α L9/3000	0564
59	α L25/3000	0571
60	α L50/2000	0572

α C Series Servo Motor

Motor Type Code	Motor Model	Motor Specification
7	α C3/2000	0121
8	α C6/2000	0126
9	α C12/2000	0141
10	α C22/1500	0145

α HV Series Servo Motor

Motor Type Code	Motor Model	Motor Specification
3	α 12HV/3000	0176
4	α 22HV/3000	0177
5	α 30HV/3000	0178

α M Series Servo Motor

Motor Type Code	Motor Model	Motor Specification
24	α M3/3000	0161
25	α M6/3000	0162
26	α M9/3000	0163

β Series Servo Motor

Motor Type Code	Motor Model	Motor Specification
13	β 0.5/3000	0013
35	β 1/3000	0031
36	β 2/3000	0032
33	β 3/3000	0033
34	β 6/2000	0034

β M Series Servo Motor

Motor Type Code	Motor Model	Motor Specification
115	β M 0.5/5000	0115
116	β M 1/5000	0116

6.02 Analog Servo Command. The Analog Servo Command determines whether the analog command issued by the DSM300 series module is a velocity or torque command. The torque command selection is supported in the DSM314 firmware 3.0 or later. Default: Velocity

6.03 Position Error Limit. Position Error Limit (User Units). The Position Error Limit is the maximum Position Error (Commanded Position - Actual Position) allowed when the DSM314 is controlling a servo. Position Error Limit should normally be set to a value 10% to 20% higher than the highest Position Error encountered under normal servo operation. Default: 60000.

The Position Error Limit range formula is:

$$256 \times (\text{user units/counts}) \text{ Position Error Limit } 60,000 \times (\text{user units/counts})$$

If Velocity Feedforward is not used, Position Error Limit can be set to a value approximately 20% higher than the Position Error required to produce a 4000-rpm command. The Position Error (User Units) required to produce a 4000-rpm command with 0% Velocity Feed forward is:

$$\text{Position Error (user units)} = \frac{\text{Position Loop Time Constant (ms)} \times \text{Servo Velocity @ 4000 rpm}}{1000} \quad (\text{user units/sec})$$

Example

The user units counts ratio is 2:1 and the Position Loop Time Constant is 50 ms.

Step 1:

$$\begin{aligned} \text{Calculate servo velocity at 4000 rpm} &= \frac{(2 \text{ user units/count}) \times (8192 \text{ counts/rev}) \times (4000 \text{ revs/minute})}{(60 \text{ seconds/minute})} \\ &= 1,092,266 \text{ user units/second} \end{aligned}$$

Step 2:

$$\begin{aligned} \text{Calculate Position Error at 4000 rpm} &= \frac{(50 \text{ milliseconds}) \times (1,092,266 \text{ user units/second})}{1000 \text{ milliseconds/second}} \\ &= 54613 \text{ user units} \end{aligned}$$

If Velocity Feedforward is used to reduce the following error, a smaller error limit value can be used, but in general, the error limit value should be 10% - 20% higher than the largest expected following error.

Note: *An Out of Sync error will occur and cause a fast stop if the Position Error Limit Value is exceeded by more than 1000 counts. The DSM314 attempts to prevent an Out of Sync error by temporarily halting the internal command generator whenever position error exceeds the Position Error Limit. Halting the command generator allows the position feedback to catch up and reduce position error below the error limit value.*

If the feedback does not catch up and the position error continues to grow, the Out of Sync condition will occur. Possible causes are:

-
1. Erroneous feedback wiring
 2. Feedback device coupling slippage
 3. Servo drives failure.
 4. Mechanically forcing the motor/encoder shaft past the servo torque capability.

5. Commanded motor acceleration or motor deceleration that is greater than system capability.

6.04 In Position Zone. In Position Zone (User Units). When the Position Error is less than or equal to the active **In Position Zone** value, the In Zone %I bit will be ON. Default: 10.

6.05 Pos Loop Time Constant (0.1ms). Position Loop Time Constant (units = 0.1 milliseconds). The desired servo position loop time constant. This value configures the amount of time required for the servo velocity output to reach 63% of its final value when a step change occurs in the Velocity command. The lower the value, the faster the system response. Values that are too low will cause system instability and oscillation. Default: 1000 = 100 ms.

Note: For accurate commanded velocity profile tracking, Pos Loop Time Constant should be 1/4 to 1/2 of the MINIMUM system acceleration or deceleration time. For example, if the fastest acceleration that must occur occupies 100msec of time the Pos Loop Time Constant should be between 25 to 50msec. To maintain system stability, use the largest value possible.

For users familiar with servo bandwidth expressed in rad/sec:

$$\text{Bandwidth (rad/sec)} = 1000 / \text{Position Loop Time Constant (ms)}$$

For users familiar with servo gain expressed in ipm/mil:

$$\text{Gain (ipm/mil)} = 60 / \text{Position Loop Time Constant (ms)}$$

Table 36: Gain / Bandwidth / Position Loop Time Constant

Gain (ipm/mil)	Bandwidth (rad/sec)	Position Loop Time Constant (ms)
0.5	8.5	120
0.75	12.5	80
1.0	16.6	60
1.5	25.1	40
2.0	33.4	30
2.5	41.8	24
3.0	50	20

For applications that do not require feedback control or employ very crude positioning systems, an **Open Loop Mode** exists. Setting a zero Position Loop Time Constant, which indicates that the positioning loop is disabled, selects this mode. Note that in Open Loop Mode, the only way to generate motion is to program a non-zero Velocity Feedforward. The Position Error is no longer used to generate motion because Position Error is based on position feedback and Open Loop Mode ignores all feedback.

CAUTION

For Analog Axes, the Position Loop Time Constant will not be accurate unless the Velocity at Max Cmd value is set correctly.

- 6.06 Velocity at MaxCmd** (User Units/Second.) All DSM314 analog servo functions depend on this value being correct for proper operation.

For Digital Servo Mode, the Velocity at Max Cmd configuration field is not used.

For Analog Servo Mode in Velocity Mode, the Velocity at Max Cmd configuration field is the Actual Servo Velocity (User Units/second) desired for a 10 Volt DSM314 analog velocity command output to the servo. The Force D/A Output %AQ Immediate Command and the Actual Velocity %AI status word can be used for a command voltage to empirically determine the proper configuration value if necessary.

For Analog Servo Mode in Torque Mode, the Velocity at Max Cmd configuration field is the maximum velocity that the user desires the servo to be able to run. The value is determined by the capabilities of the servo system being controlled and the capabilities of the driven load.

In Digital Mode only, if the user sends the DSM314 a velocity command that exceeds the servo system capability, the DSM314 will clamp that command value at the appropriate maximum motor velocity boundary. **Note that no error will be reported back to the DSM314.**

See Appendix D, “Start-up and Tuning Digital and Analog Servo Systems,” for more information on determining the correct value.

Default: 100000.

CAUTION

The Velocity at 10V must be configured correctly in order for the analog servo Pos Loop Time Constant and Velocity Feedforward factors to be accurate.

- 6.07 Velocity Feed Forward (0.01%).** Velocity Feed forward gain (units = 0.01 percent). The Commanded Velocity percentage that is added to the DSM314's position loop velocity command output. Increasing Velocity Feedforward causes the servo to operate with faster response and reduced position error. The optimum value for each system has to be determined individually. For Digital Servos, 95% **Velocity Feed Forward Percentage** value is a good starting point. For analog servos, 70% is a good starting point. The servo system capabilities will determine the optimum value. If **Velocity Feed Forward** is changed, **Pos Err Limit** may require adjustment. Default: 0.

CAUTION

For Analog Axes, the Velocity Feed Forward Percentage will not be accurate unless the Velocity at MaxCmd value is first set correctly.

- 6.08 Acceleration Feed Forward Percentage.** This configuration item is not used in the current DSM314 firmware.
- 6.09 Integrator Mode** Integrator Mode. Position loop position error integrator operating mode. Off means the integrator is not used. Continuous means the integrator runs continuously even during servo motion. Servo Null means the integrator only runs when the Moving %I status bit is OFF. **Integrator Mode** should normally be set to Off. Continuous mode may be used for Follower operation only when a constant or slowly changing master velocity is expected. This parameter should not be used to dampen disturbances in the position loop feedback. Never select Continuous for point to point positioning applications. Default: OFF.
- 6.10 Integrator Time Constant** Integrator Time Constant (milliseconds). This is the position loop position error integrator time constant. This value indicates the time required to reduce the position error by 63%. For example, if the Integrator Time Constant is 1000 (1 second), the Position Error would be reduced to 37% of its initial value after 1 second. A value of zero turns off the integrator. **If used, the Integrator Time Constant should be 5 to 10 times greater than the Position Loop Time Constant to prevent instability and oscillation.** Default: 0.
- 6.11 Velocity Loop Gain** Used to set velocity loop gain. This applies to Digital Servos and Analog Torque Mode Servos only. **This parameter is not used for Analog Servos in Velocity Mode.** The formula

$$\text{Velocity Loop Gain} = \frac{\text{Load Inertia (J}_L\text{)}}{\text{Motor Inertia (J}_M\text{)}} \times 16$$

can be used to select an initial velocity loop gain value. The allowable value range is 0 to 255. The value of 0 should be used if the motor shaft is not attached to a load. Default: 16 (load inertia equals motor inertia).

4.3.8 Computing Data Limit Variables

The data limit values for parameters MaxPosnUu, MaxVelUu, and MaxAccUu, referred to in some of the tables in this chapter, can be calculated using the following formulas:

Table 37: Computing Data Limit Variables

Formulas for Computing Data Limit Variables		
Position Limit MaxPosnUu	Velocity Limit MaxVelUu	Acceleration Limit MaxAccUu
If uu:cts >= 1:1 MaxPosnUu = 536,870,912 Else (uu:cts < 1:1) MaxPosnUu = 536,870,912 * uu/cts	MaxVelUu = 1,000,000 * uu/cts	If uu:cts >= 1:1 MaxAccUu = 1,073,741,823 Else (uu:cts < 1:1) MaxAccUu = 1,073,741,823 * uu/cts

4.3.9 Advanced Tab Data

The Advanced Tab allows up to 16 custom tuning parameters and associated data to be entered for each axis. Although the Advanced Tab has 16 rows for entering axis tuning parameter data, the DSM314 Release 1.0 firmware only allows Entry rows 1 and 2 to be used. The figure below shows data in the cells for Axis 1 on Entry rows 1 and 2. DSM firmware version 3.0 or later removes this restriction.

Figure 61: Advanced Tab

Settings	SNP Port	CTL Bits	Output Bits	Axis #1	Axis #2	Axis #3	Tuning #1	Tuning #2	Advanced	Power Consumption
Entry				Axis 1 Par #	Axis 1 Data	Axis 2 Par #	Axis 2 Data	Axis 3 Par #	Axis 3 Data	Axis 3 Data
1				1	2	0	0	0	0	0
2				3	10	0	0	0	0	0
3				0	0	0	0	0	0	0
4				0	0	0	0	0	0	0
5				0	0	0	0	0	0	0
6				0	0	0	0	0	0	0
7				0	0	0	0	0	0	0
8				0	0	0	0	0	0	0
9				0	0	0	0	0	0	0
10				0	0	0	0	0	0	0
11				0	0	0	0	0	0	0
12				0	0	0	0	0	0	0
13				0	0	0	0	0	0	0
14				0	0	0	0	0	0	0
15				0	0	0	0	0	0	0
16				0	0	0	0	0	0	0

Tuning Parameter 1: Sets Digital Encoder Resolution (for digital servos only). Settings other than 0 result in a derating of the maximum supported motor speed. Note that, for settings 0 and 1, some motors’ maximum speed ratings are below the maximum supported speed shown in the table. Range of allowable settings: 0 – 3. In Figure 61 above, Tuning Parameter 1 is set to a value of 2 for Axis 1.

Table 38: Tuning Parameter 1 Values

Tuning Parameter 1 Values Counts/Revolution Maximum Supported Motor Speed

0	8192	44001,2
1	16384	36622
2	32768	1831
3	65536	915

Note:

- Default setting
- Some motors’ maximum speed rating is lower than the value in the table

Tuning Parameter 3: Sets minimum velocity output (millivolts) for analog servos. Allowed data range is 0 -1000 millivolts. The recommended setting is 5 - 10mv, or just enough to make the servo pull in to +/- 1 count of position error. In Figure 4.2 above, Tuning Parameter 3 is set to a value of 10 for Axis 1.

Tuning Parameter 6: Sets the encoder resolution. The parameter is only used in torque mode. For correct torque mode operation, this value must be set to the number of quadrature encoder counts (4X encoder lines) generated by the motor feedback device per revolution. The user can determine the value from the feedback device specification. As a double check, the user may wish to connect the feedback device to the DSM and manual rotate the motor shaft one revolution. The reading on the DSM %AI data for actual position should closely match (variations are caused by the accuracy of manual turning shaft one revolution) the value placed in this parameter. The allowed range is 100-32767 counts/revolution. The default value is 4096 counts per revolution

Tuning Parameter 7: Sets the velocity regulator proportional gain. The parameter is only used in torque mode. The proportional gain is multiplied by velocity error (velocity command - velocity feedback) to generate the portion of the torque command due to the proportional term. Correctly setting this value will determine how well the velocity regulator performs in the control system. Appendix D describes a method to correctly tune this parameter. The allowable range for the velocity loop proportional gain term is 0-32767. The default value is 1500.

Tuning Parameter 8: Sets the velocity regulator integral gain. The parameter is only used in torque mode. The integral gain is the term multiplied by the area of the velocity error (velocity command - velocity feedback) to generate the portion of the torque command due to the integral term. Correctly setting this value will determine how well the velocity regulator performs in the control system. Appendix D describes a method to correctly tune this parameter. The allowable range for the velocity loop proportional gain term is 0-32767. The default value is 0.

Tuning Parameter 10: Sets the Torque Command Filter setting. The torque command filter allows the user to activate a low pass filter for the velocity regulator output. . The filter is typically used to keep the controller from exciting a machine resonance. The allowable range for torque filter settings is 0 – 3. The default value is 0.

Table 39: Tuning Parameter 10 Values

Tuning Parameter 10 Values	Torque Command Low Pass Filter Setting
0	OFF1
1	Low Bandwidth Filter (150 Hz 3db point)
2	Medium Bandwidth Filter (250 Hz 3db point)
3	High Bandwidth Filter (350 Hz. 3db point)

4.3.10 Power Consumption Data

This is a display-only tab that indicates the power required by the DSM314 module.

Chapter 5: DSM314 to Host Controller Interface

This chapter defines the data that is transferred between the CPU and the Motion Mate DSM314 automatically each host controller sweep, without user programming. This data is categorized as follows:

- Input Status Data (Transferred from Motion Mate DSM314 to CPU)
 - Status Bits: 32 (1 Axis), 48 (2 Axes), 64 (3 Axes), 80 (4 Axes) bits of %I data
 - Status Words: 24 (1 Axis), 44 (2 Axes), 64 (3 Axes), 84 (4 Axes) words of %AI data
- Output Command Data (Transferred from CPU to Motion Mate DSM314)
 - Discrete Commands: 32 (1 Axis), 48 (2 Axes), 64 (3 Axes), 80 (4 Axes) bits of %Q data
 - Immediate Commands: 3 (1 Axis), 6 (2 Axes), 9 (3 Axes), 12 (4 Axes) words of %AQ data

Note: Throughout this chapter words shown in italics refer to actual host controller machine data references (%I, %A, %AI, %AQ).

5.1 Section 1: %I Status Bits

The following %I Status Bits are transferred automatically from the DSM314 to the CPU each sweep. The actual addresses of the Status Bits depend on the starting address configured for the %I reference (see Table 40, “Settings Tab”). The bit offsets listed in the following table are offsets to this starting address. All reference section designations pertain to this chapter.

Table 40: %I Status Bits

Bit Offset	Description	Axis	Ref.	Bit Offset	Description	Axis	Ref.
00	Module Error Present	N/A	1.01	40	Position Error Limit	Servo 2	1.12
01	Local Logic Active	N/A	1.02	41	Torque Limit	Servo 2	1.13
02	New Configuration Received	N/A	1.03	42	Servo Ready / IN4_B (5v)	Servo 2	1.14
03	Reserved			43	Reserved		
04	CTL01 (function selected by config)	N/A	1.04	44	Follower Enabled	Servo 2	1.15
05	CTL02 (function selected by config)	N/A	1.04	45	Velocity Limit	Servo 2	1.16
06	CTL03 (function selected by config)	N/A	1.04	46	Follower Ramp Active	Servo 2	1.17
07	CTL04 (function selected by config)	N/A	1.04	47	Reserved		
08	CTL05 (function selected by config)	N/A	1.04	48	Axis OK	Servo 3	1.05
09	CTL06 (function selected by config)	N/A	1.04	49	Position Valid	Servo 3	1.06
10	CTL07 (function selected by config)	N/A	1.04	50	Drive Enabled	Servo 3	1.07

Bit Offset	Description	Axis	Ref.	Bit Offset	Description	Axis	Ref.
11	CTL08 (function selected by config)	N/A	1.04	51	Program Active	Servo 3	1.08
12	CTL13 (function selected by config)	N/A	1.04	52	Moving	Servo 3	1.09
13	CTL14 (function selected by config)	N/A	1.04	53	In Zone	Servo 3	1.10
14	CTL15 (function selected by config)	N/A	1.04	54	Strobe 1 Flag (5v)	Servo 3	1.11
15	CTL16 (function selected by config)	N/A	1.04	55	Strobe 2 Flag (5v)	Servo 3	1.11
16	Axis OK	Servo 1	1.05	56	Position Error Limit	Servo 3	1.12
17	Position Valid	Servo 1	1.06	57	Reserved		
18	Drive Enabled	Servo 1	1.07	58	Servo Ready/IN4_C Input	Servo 3	1.14
19	Program Active	Servo 1	1.08	59	Reserved		
20	Moving	Servo 1	1.09	60	Follower Enabled	Servo 3	1.15
21	In Zone	Servo 1	1.10	61	Velocity Limit	Servo 3	1.16
22	Strobe 1 Flag (5v)	Servo 1	1.11	62	Follower Ramp Active	Servo 3	1.17
23	Strobe 2 Flag (5v)	Servo 1	1.11	63	Reserved	Servo 3	
24	Position Error Limit	Servo 1	1.12	64	Axis OK	Servo 4	1.05
25	Torque Limit	Servo 1	1.13	65	Position Valid	Servo 4	1.06
26	Servo Ready / IN4_A (5v)	Servo 1	1.14	66	Drive Enabled	Servo 4	1.07
27	Reserved			67	Program Active	Servo 4	1.08
28	Follower Enabled	Servo 1	1.15	68	Moving	Servo 4	1.09
29	Velocity Limit	Servo 1	1.16	69	In Zone	Servo 4	1.10
30	Follower Ramp Active	Servo 1	1.17	70	Strobe 1 Flag (5v)	Servo 4	1.11
31	Reserved			71	Strobe 2 Flag (5v)	Servo 4	1.11
32	Axis OK	Servo 2	1.05	72	Position Error Limit	Servo 4	1.12
33	Position Valid	Servo 2	1.06	73	Reserved	Servo 4	
34	Drive Enabled	Servo 2	1.07	74	Servo Ready / IN4_D (5v)	Servo 4	1.14
35	Program Active	Servo 2	1.08	75	Reserved	Servo 4	
36	Moving	Servo 2	1.09	76	Follower Enabled	Servo 4	1.15
37	In Zone	Servo 2	1.10	77	Velocity Limit	Servo 4	1.16
38	Strobe 1 Flag (5v)	Servo 2	1.11	78	Follower Ramp Active	Servo 4	1.17
39	Strobe 2 Flag (5v)	Servo 2	1.11	79	Reserved	Servo 4	

1.01 Module Error Present. This status bit is set when the DSM314 detects any error. Errors related to a specific Servo or Auxiliary Axis will be identified in the associated Axis n Error Code %AI word. Module errors not related to a specific axis will be identified in the Module Status Code %AI word. See section 2, “%AI Status Words”, for more details. The Clear Error %Q bit is the only command that will clear the Module Error Present %I status bit and the associated Module Status Code and Axis n Error Code %AI word(s). If the condition causing the error is still present, the Module Error Present %I status bit will not be cleared.

- 1.02 Local Logic Active.** When this status bit is ON, it indicates that a Local Logic program is executing.
- 1.03 New Configuration Received.** The New Configuration Received %I status bit is set whenever the host controller sends a reset command or new configuration to the DSM314. New Configuration Received should be cleared by a host controller program before any %AQ Immediate commands such as In Position Zone or Position Loop Time Constant have been sent to the DSM314. The status bit can then be monitored by the host controller. If the bit is set, then the DSM314 has been reset or reconfigured. The host controller should clear the bit and then re-send all necessary %AQ commands. The bit is cleared by %AQ Immediate command 49h. Refer to section 4, “%AQ Immediate Commands,” later in this chapter, for more details about the %AQ immediate command interfaces.
- 1.04 Configurable %I Status Bits.** These inputs indicate the state of configurable CTL bits CTL01-CTL08 and CTL13-CTL16. The default CTL bit assignments report the level of external input devices connected to faceplate signals. All CTL bits may be tested during the execution of motion program Wait and Conditional Jump commands. CTL bits can also be used to trigger the follower ramp enable / disable functions. The CTL bit assignments are selected through configuration. Consult Chapters 4 and 14 for additional information. Default CTL01-CTL08 and CTL13-CTL16 assignments are shown in Table 41.

Table 41: Defaults for Configurable %I Status Bits

Bit Name	Signal Name	Signal Use	Input Type	Faceplate Connector Pin	Digital Servo TB Pin	Analog Servo / Aux Axis TB Pin
CTL01	IN9_A	Servo Axis 1 (+) Overtravel	24v	A-16	6	16
CTL02	IN10_A	Servo Axis 1 (-) Overtravel	24v	A-34	14	34
CTL03	IN11_A	Servo Axis 1 Home Switch	24v	A-17	7	17
CTL04	IN1_A	Servo Axis 1 Strobe 1 Level	5v	A-1, 19	1,9	9
CTL05	IN9_B	Servo Axis 2 (+) Overtravel	24v	B-16	6	16
CTL06	IN10_B	Servo Axis 2 (-) Overtravel	24v	B-34	14	34
CTL07	IN11_B	Servo Axis 2 Home Switch	24v	B-17	7	17
CTL08	IN1_B	Servo Axis 2 Strobe 1 Level	5v	B-1,19	1,9	9
CTL13	IN9_C	Servo Axis 3 (+) Overtravel	24v	C-16	NA	16
CTL14	IN10_C	Servo Axis 3 (-) Overtravel	24v	C-34	NA	34
CTL15	IN11_C	Servo Axis 3 Home Switch	24v	C-17	NA	17
CTL16	IN5_C	Servo Axis 3 Strobe 1 Level	5v	C-9	NA	9

- 1.05 Axis OK.** The Axis OK status bit is ON when the DSM314 is ready to receive commands and control a servo. An error condition that stops the servo will turn Axis OK OFF. **When Axis OK is OFF, no commands other than the Clear Error %Q bit will be accepted by the axis.**

- 1.06 Position Valid.** For a Servo Axis, the Position Valid status bit indicates that a Set Position command or successful completion of a Find Home cycle has initialized the position value in the Actual Position % AI status word. For a Servo Axis, Position Valid must be ON in order to execute a motion program.

For an Auxiliary Axis, the Position Valid status bit indicates that an Aux Encoder Set Position command or successful completion of a Find Home cycle has initialized the position value in the Actual Position % AI status word. For an Aux Axis, Position Valid is **not** required to be ON in order to execute a motion program.

If the DSM314 is configured to use an absolute feedback digital encoder (α or β Series servo with optional encoder battery), Position Valid is automatically set whenever the digital encoder reports a valid absolute position. See Appendix C for details of operation when absolute mode digital encoders are used.

- 1.07 Drive Enabled.** The Drive Enabled status bit indicates the state of the Enable Drive %Q bit and the solid-state relay output supplied by the DSM314. The ON state of the Drive Enabled %I bit corresponds to the CLOSED state of the relay output and the ON state of the associated faceplate EN LED. In Digital mode, the solid-state relay provides the MCON signal to the Digital Servo through the servo command cable. Drive Enabled is cleared following power-up or an error condition that stops the servo.

- 1.08 Program Active.** The Program Active status bit for each axis indicates that a Motion Program (1-10) or a Move %AQ command (27h) is executing on that axis. Executing a multi-axis program will set the Program Active %I bits for both Axis 1 and Axis 2.

- 1.09 Moving.** The Moving status bit is ON when Commanded Velocity is non-zero, otherwise it is OFF. All Move, Jog, and Move at Velocity commands will cause the Moving bit to be set to ON. The Force Servo Velocity %AQ command and Follower acceleration ramp will not set the Moving bit.

In Follower mode, Moving is ON for the conditions described above and is not affected by the enabled or disabled state of the follower master input. When the Follower acceleration / deceleration ramp is active, a separate %I bit, Follower Ramp Active, is ON. Refer to Chapter 8, Follower Motion, for additional information on the Follower Acceleration Ramp.

- 1.10 In Zone.** Operation of the In Zone bit depends only on the Position Error value and is not related to the state of the Moving bit. In Zone will be ON whenever Position Error is less than or equal to the configured **In Position Zone** value. In Zone (ON) can be used in combination with the Moving bit (OFF) to determine when the axis has arrived at its destination.

Table 42: In Zone Bit Operation

Cmd Generator Active (Moving %I bit ON)	Position Error \leq In Position Zone	In Zone bit	Axis at Destination
No	No	OFF	No
No	Yes	ON	Yes
Yes	No	OFF	No
Yes	Yes	ON	No

1.11 Strobe 1 Flag, Strobe 2 Flag. The Strobe 1 Flag and Strobe 2 Flag status bits indicate that an OFF to ON transition has occurred at the associated faceplate Strobe Input. When this occurs, an axis position is captured and reported in the Strobe n Position %AI status word, where “n” is Axis 1 - Axis 4. The Strobe n Flag %I bit is cleared by the associated Reset Strobe n %Q bit. **A maximum of 2 host controller sweeps is required for the Strobe n Flag %I bit to be cleared in the host controller after a Reset Strobe n %Q bit is turned ON.** Once the Strobe n Flag bit is cleared, new data may be captured by another Strobe Input. The position capture resolution is +/- 2 counts with an additional 10 microseconds of variance for the strobe input filter delay.

Note: *The Strobe n Flag bits do not indicate the logic level of the faceplate input, they only indicate that an OFF to ON transition has occurred on the input.*

1.12 Position Error Limit. The Position Error Limit status bit is set when the absolute value of the position error exceeds the configured Position Error Limit value. When the Position Error Limit status bit is set, Commanded Velocity and Commanded Position are frozen to allow the axis to “catch up” to the Commanded Position.

1.13 Torque Limit. The Torque Limit status bit is set when the commanded torque exceeds the torque limit setting for the configured motor type.

1.14 Servo Ready. This status bit is set when faceplate signal IN4 of the associated connector (A, B, C or D) is ON (active low: ON = 0v, OFF = +5v). For each Servo Axis, this input reports the Servo Ready state of the servo amplifier.

1.15 Follower Enabled. This status bit indicates when the Follower is enabled for the axis. The Enable Follower % Q bit and an optional CTL01-CTL32 faceplate trigger input enable the Follower function. If follower ramp acceleration control is active when Follower Enabled turns on, the axis will accelerate to the master velocity command, and when it turns off, the axis will decelerate to zero master velocity command. Both acceleration and deceleration during the ramp process will utilize the configured Follower Ramp Acceleration.

1.16 Velocity Limit. The Velocity Limit status bit is set if the velocity requested by any axis command (internal path generator or internal/external follower source) exceeds the configured velocity limit. Therefore, Velocity Limit is an indication that the axis is no longer locked to its position command. If Follower is enabled, an error code is reported in the associated axis Error Code variable when Velocity Limit is set.

An exception exists when unidirectional motion is configured by setting Command Direction to Positive Only or Negative Only. Positive Only means that the velocity

limit is zero for negative motion. Negative Only means that the velocity limit is zero for positive motion. No error is generated for the limit that is set to zero. For example, if Command Direction is set to Negative Only and + Counts are commanded, the Velocity Limit Status bit is set, but no Status Error code is reported.

- 1.17 Follower Ramp Active.** When the follower is enabled, Follower Ramp Active is ON during initial acceleration and distance makeup. When the follower is disabled, Follower Ramp Active is ON until the Follower Disable Action incremental distance (if selected) has been traveled and the follower has decelerated to zero velocity.

5.2 Section 2: %AI Status Words

The following %AI Status Words are transferred automatically from the DSM314 to the CPU each sweep. The total number of the %AI Status Words is configured with the Configuration Software to be a length of 24, 44, 64 or 84. The actual addresses of the Status Words depend on the starting address configured for the %AI references. See Table 40, “Settings Tab.” The word numbers listed in the following table are offsets to this starting address. All reference section designations pertain to this chapter. All %AI data except Actual Velocity is updated within the DSM314 at the position loop sampling rate (2 ms for digital servos, 0.5 ms or 1.0 ms for some analog servo configurations). Actual Velocity is updated once every 128 milliseconds.

Table 43: %AI Status Words

Word Offset	Description	Axis	Ref	Word Offset	Description	Axis	Ref
00	Module Status Code	N/A	2.01				
01-03	Reserved						
04	Axis 1 Error Code	Servo 1	2.02	44	Axis 3 Error Code	Servo 3	2.02
05	Command Block Number	Servo 1	2.03	45	Command Block Number	Servo 3	2.03
06-07	Commanded Position	Servo 1	2.04	46-47	Commanded Position	Servo 3	2.04
08-09	Actual Position	Servo 1	2.05	48-49	Actual Position	Servo 3	2.05
10-11	Strobe 1 Position	Servo 1	2.06	50-51	Strobe 1 Position	Servo 3	2.06
12-13	Strobe 2 Position	Servo 1	2.06	52-53	Strobe 2 Position	Servo 3	2.06
14-15	Position Error	Servo 1	2.07	54-55	Position Error	Servo 3	2.07
16-17	Commanded Velocity	Servo 1	2.08	56-57	Commanded Velocity	Servo 3	2.08
18-19	Actual Velocity	Servo 1	2.09	58-59	Actual Velocity	Servo 3	2.09
20-21	User Selected Data 1	Servo 1	2.10	60-61	User Selected Data 1	Servo 3	2.10
22-23	User Selected Data 2	Servo 1	2.11	62-63	User Selected Data 2	Servo 3	2.11
24	Axis 2 Error Code	Servo 2	2.02	64	Axis 4 Error Code	Servo 4	2.02
25	Commanded Block Number	Servo 2	2.03	65	Command Block Number	Servo 4	2.03

26-27	Commanded Position	Servo 2	2.04	66-67	Commanded Position	Servo 4	2.04
28-29	Actual Position	Servo 2	2.05	68-69	Actual Position	Servo 4	2.05
30-31	Strobe 1 Position	Servo 2	2.06	70-71	Strobe 1 Position	Servo 4	2.06
32-33	Strobe 2 Position	Servo 2	2.06	72-73	Strobe 2 Position	Servo 4	2.06
34-35	Position Error	Servo 2	2.07	74-75	Position Error	Servo 4	2.07
36-37	Commanded Velocity	Servo 2	2.08	76-77	Commanded Velocity	Servo 4	2.08
38-39	Actual Velocity	Servo 2	2.09	78-79	Actual Velocity	Servo 4	2.09
40-41	User Selected Data 1	Servo 2	2.10	80-81	User Selected Data 1	Servo 4	2.10
42-43	User Selected Data 2	Servo 2	2.11	82-83	User Selected Data 2	Servo 4	2.11

2.01 Module Status Code. Module Status Code indicates the current DSM314 operational status. When the Module Error Present %I flag is set, and the error is not related to a specific axis, an error code number is reported in the Module Status Code that describes the condition causing the error. A new Module Status Code **will not replace** a previous Module Status Code unless the new Module Status Code has Fast Stop or System Error priority.

The Module Status Code word is also used to report System Status Errors. These are of the format Dxxx, Exxx, and Fxxx. For details on System Status Error codes, refer to Appendix A.

For a list of Motion Mate DSM314 error codes refer to Appendix A.

2.02 Axis 1 - Axis 4 Error Code. The Servo Axis n Error Code, where n = Axis 1 - Axis 4, indicates the current operating status of each axis. When the Module Error Present %I flag is set, and the error is related to a particular axis, an error code number is reported, which describes the condition causing the error. A new Axis Error Code will replace a previous Axis Error Code if it has equal or higher priority (Warning, Normal Stop, Fast Stop) compared to the previous Axis Error Code.

For a list of Motion Mate DSM314 error codes refer to Appendix A.

2.03 Command Block Number. Command Block Number indicates the block number of the command that is presently being executed in the active Program or Subroutine. It changes at the start of each new block as the program commands are executed, and thus identifies the present operating location within the program. Block numbers are displayed only if the motion program uses them. Additionally, the most recently used block number will be displayed until superseded by a new value. The Command Block Number is set to zero on power cycle or reset.

2.04 Commanded Position. Commanded Position (user units) is where the axis is commanded to be at any instant in time. For a Servo Axis, the difference between Commanded Position and Actual Position is the Position Error value that produces the Velocity Command to drive the axis. The rate at which the Commanded Position is changed determines the velocity of axis motion.

If Commanded Position moves past either of the count limits, it will roll over to the other limit and continue in the direction of the axis motion.

2.05 Actual Position. Actual Position (user units) is a value maintained by the DSM314 to represent the physical position of the axis. It is set to an initial value by the Set Position %AQ Immediate command or to Home Position by the Find Home cycle. When digital absolute encoders are used, Actual Position is automatically set whenever the encoder reports a valid position. The motion of the axis feedback device continuously updates the axis Actual Position.

If Actual Position moves past either of the count limits, it will roll over to the other limit and continue in the direction of the axis motion.

2.06 Strobe 1, 2 Position. Strobe 1 Position and Strobe 2 Position (user units) contain the axis actual position when a Strobe 1 Input or Strobe 2 Input occurs. When a Strobe Input occurs, the Strobe 1 Flag or Strobe 2 Flag %I bit is set to indicate to the host controller that new Strobe data is available in the related Strobe 1 Position or Strobe 2 Position status word. The host controller must set the proper Reset Strobe 1 or Reset Strobe 2 Flag %Q bit to clear the associated Strobe 1,2 Flag %I bit.

Strobe 1, 2 Position will be maintained and will not be overwritten by additional Strobe Inputs until the Strobe 1, 2 Flag %I bit has been cleared. If the Reset Strobe Flag %Q bit is left in the

ON state (thus holding the Strobe 1, 2 Flag %I bit in the cleared state), then each Strobe Input that occurs will cause the axis position to be captured in Strobe 1, 2 Position.

The Strobe 1, 2 Position actual position values are also placed in data parameter registers for use with motion programs commands. The data parameter register assignments are as follows:

	Servo Axis 1	Servo Axis 2	Servo Axis 3	Servo Axis 4
Strobe 1 Position	P224	P232	P240	P248
Strobe 2 Position	P225	P233	P241	P249

This feature allows the strobe input to trigger a Conditional JUMP in a program block using the Strobe 1 Position or Strobe 2 Position as the destination of a CMOVE or PMOVE command.

See Chapter 1, “Product Overview, DSM314 Position Strobes,” for information on strobe latency and processing times.

- 2.07 Position Error.** Position Error (user units) is the difference between Commanded Position and Actual Position. In the servo control loop, Position Error is multiplied by a gain constant to provide the servo velocity command.
- 2.08 Commanded Velocity.** Commanded Velocity (user units/sec) is a value generated by the DSM314 axis command generator. Commanded Velocity indicates the instantaneous velocity command that is producing axis motion. At the beginning of a move it will increase at the acceleration rate, and once the programmed velocity has been reached, it will stabilize at the programmed velocity value.
- In Follower mode, Commanded Velocity only represents the output of the axis command generator. The Follower Master Axis input or the Follower Acceleration Ramp controller does not affect Commanded Velocity.
- 2.09 Actual Velocity.** Actual Velocity (user units/sec) represents the axis velocity derived from the Feedback device and is updated by the DSM314 once every 128 milliseconds.
- 2.10 User Selected Data 1.** There is one of these words for each of the four axes. The information reported in User Selected Data 1 is determined by module configuration (see Chapter 4) or the Select Return Data 1 %AQ command (see Section 4, “%AQ Immediate Commands,” in this chapter).
- 2.11 User Selected Data 2.** There is one of these words for each of the four axes. The information reported in User Selected Data 2 is determined by module configuration (see Chapter 4) or the Select Return Data 2 %AQ command (see Section 4, “%AQ Immediate Commands,” in this chapter). Refer to Section 4 “%AQ Immediate Commands” for additional information.

5.3 Section 3: %Q Discrete Commands

The %Q Outputs listed in Table 44 represent Discrete Commands that are sent automatically to the DSM314 from the CPU each host controller sweep. A command is executed by turning on its corresponding Output Bit. The actual addresses of the Discrete Command bits depend on the starting address configured for the %Q references. See Table 40, “Settings Tab.” The Bit Offsets listed in the following table are offsets to this starting address. Numbers in the “Ref” columns pertain to sections in this chapter.

Table 44: %Q Discrete Commands

Bit Offset	Description	Axis	Ref	Bit Offset	Description	Axis	Ref
00	Clear Error	N/A	3.01	40	OUT1_B / Config. CTL bit src.	Servo 2	3.12
01	Enable Local Logic	N/A	3.02	41	OUT3_B / Config. CTL bit src.	Servo 2	3.13
02	Execute Motion Program 1	N/A	3.03	42	Reserved		
03	Execute Motion Program 2	N/A	3.03	43	Reserved		
04	Execute Motion Program 3	N/A	3.03	44	Enable Follower	Servo 2	3.14
05	Execute Motion Program 4	N/A	3.03	45	Select Follower Master Source	Servo 2	3.15
06	Execute Motion Program 5	N/A	3.03	46	Reserved		
07	Execute Motion Program 6	N/A	3.03	47	Reserved		
08	Execute Motion Program 7	N/A	3.03	48	Abort All Moves	Servo 3	3.05
09	Execute Motion Program 8	N/A	3.03	49	Feed Hold (Pause Program)	Servo 3	3.06
10	Execute Motion Program 9	N/A	3.03	50	Enable Drive / MCON	Servo 3	3.07
11	Execute Motion Program 10	N/A	3.03	51	Find Home	Servo 3	3.08
12	Configurable CTL bit source	N/A	3.04	52	Jog Plus	Servo 3	3.09
13	Configurable CTL bit source	N/A	3.04	53	Jog Minus	Servo 3	3.10
14	Configurable CTL bit source	N/A	3.04	54	Reset Strobe 1	Servo 3	3.11
15	Configurable CTL bit source	N/A	3.04	55	Reset Strobe 2	Servo 3	3.11
16	Abort All Moves	Servo 1	3.05	56	OUT1_C / Config. CTL bit src.	Servo 3	3.12
17	Feed Hold (Pause Prgm)	Servo 1	3.06	57	OUT3_C / Config. CTL bit src.	Servo 3	3.13
18	Enable Drive / MCON	Servo 1	3.07	58	Reserved		
19	Find Home	Servo 1	3.08	59	Reserved		
20	Jog Plus	Servo 1	3.09	60	Enable Follower	Servo 3	3.14
21	Jog Minus	Servo 1	3.10	61	Select Follower Master Source	Servo 3	3.15
22	Reset Strobe 1	Servo 1	3.11	62	Reserved		
23	Reset Strobe 2	Servo 1	3.11	63	Reserved		
24	OUT1_A / Config. CTL bit src.	Servo 1	3.12	64	Abort All Moves	Servo 4	3.05
25	OUT3_A / Config. CTL bit src.	Servo 1	3.13	65	Feed Hold (Pause Program)	Servo 4	3.06
26	Reserved			66	Enable Drive / MCON	Servo 4	3.07
27	Reserved			67	Find Home	Servo 4	3.08
28	Enable Follower	Servo 1	3.14	68	Jog Plus	Servo 4	3.09
29	Select Follower Master Source	Servo 1	3.15	69	Jog Minus	Servo 4	3.10
30	Reserved			70	Reset Strobe 1	Servo 4	3.11
31	Reserved			71	Reset Strobe 2	Servo 4	3.11
32	Abort All Moves	Servo 2	3.05	72	OUT1_B / Config. CTL bit src.	Servo 4	3.12
33	Feed Hold (Pause Program)	Servo 2	3.06	73	OUT3_B / Config. CTL bit src.	Servo 4	3.13
34	Enable Drive / MCON	Servo 2	3.07	74	Reserved	Servo 4	

35	Find Home	Servo 2	3.08	75	Reserved	Servo 4	
36	Jog Plus	Servo 2	3.09	76	Enable Follower	Servo 4	3.14
37	Jog Minus	Servo 2	3.10	77	Select Follower Master Source	Servo 4	3.15
38	Reset Strobe 1	Servo 2	3.11	78	Reserved	Servo 4	
39	Reset Strobe 2	Servo 2	3.11	79	Reserved	Servo 4	

3.01 Clear Error. When an error condition is reported, this command is used to clear the Module Error Present %I status bit as well as the associated Module Status Code and Axis 1-Axis 4 Error Code %AI status words. Error conditions that are still present (such as an End of Travel limit switch error) will not be cleared and must be cleared by some other corrective action. If the Clear Error bit is maintained ON, a Jog command can be used to move away from an open hardware overtravel limit switch.

3.02 Enable Local Logic. This command enables the current Local Logic program within the DSM to execute. Refer to Chapter 4 for information on configuring the Local Logic program name.

3.03 Execute Motion Program 1 - 10. These commands are used to select stored motion programs for immediate execution. Each command uses a one-shot action; thus a command bit must transition from OFF to ON each time a program is to be executed. Programs may be temporarily paused by a Feed Hold command.

When a program begins execution, Rate Override is always set to 100%. A Rate Override %AQ command can be sent on the same sweep as the Execute Motion Program n %Q bit and will be effective as the program starts.

Only one Motion Program can be executed at a time per axis. The Program Active %I status bit must be OFF, or Motion Program execution will not be allowed to start. A multi-axis Motion Program uses both axis 1 and axis 2, so both Program Active bits must be OFF to start a multi-axis Motion Program.

3.04 Configurable CTL Bit Sources. %Q bit offsets 12-15 are configurable as sources for CTL bits CTL01-CTL24. Refer to Chapter 4 for additional information. The default configuration is:

%Q bit offset 12: CTL09

%Q bit offset 13: CTL10

%Q bit offset 14: CTL11

%Q bit offset 15: CTL12

3.05 Abort All Moves. This command causes any motion in progress to halt at the current Jog Acceleration rate and configured Jog Acceleration Mode. **Therefore it is important to use a Jog Acceleration that will provide deceleration in a satisfactory distance.** Any pending programmed or immediate command is canceled and therefore not allowed to become effective. The abort condition is in effect as long as this command is on. If motion was in progress when the command was received, the Moving status bit will remain set until the commanded velocity reaches zero.

3.06 Feed Hold (On Transition). This command causes any motion programs in progress to stop at the active program acceleration rate. The Feed Hold command does not stop motion commanded by a master source in Follower Enabled Mode. Once the motion is stopped, the Moving status bit is cleared, and the In-Zone status bit is set when the In Zone condition is attained. Jog commands are allowed when in the Feed hold condition. After an ON transition, program motion will stop, even if the command bit transitions back OFF before motion stops.

Feed Hold (Off Transition). This command causes any motion programs interrupted by Feed Hold to resume at the programmed acceleration and velocity rate. Additional program moves will then be processed, and normal program execution will continue. Feed Hold OFF behaves in a similar fashion to an Execute Program command except the path generation software uses only the remaining distance in the program.

If jogging occurred while Feed Hold was ON, the interrupted Move command will resume from where the axis was left after the Jog. The Move finishes at the correct programmed velocity and continues to the original programmed position as if no jog displacement occurred.

3.07 Enable Drive / MCON. If the Module Error Present and Drive Enabled %I status bits are cleared, this command will cause the Drive Enable relay contact to close and the Drive Enabled %I bit to be set. When the Drive Enabled %I bit is set, the path generation and position control functions are enabled, and servo motion can be commanded. A signal will be sent (MCON) to the digital servo enabling the drive. Enable Drive must be maintained ON to allow normal servo motion (except when using Jog commands). If using the Force Analog Output immediate command (see Section 4.06, “Force Analog Output”), the applicable Enable Drive signal must be on to produce an analog output with this command.

3.08 Find Home. This command causes the DSM314 to establish the Home Position. A Home Limit Switch Input from the I/O connector roughly indicates the reference position for Home, and the next encoder marker encountered indicates the exact home position. When the **Home Mode** axis configuration is set to MOVE+ or MOVE-, the Home Limit Switch input will be ignored. For a Servo Axis, the configured **Home Offset** defines the location of Home Position as the offset distance from the Home Marker. The Position Valid %I bit indication is set at the conclusion of the Home Cycle. **See Chapter 6 for additional Home Cycle information. See Appendix C for absolute encoder information.**

3.09 Jog Plus. When this command bit is ON, the axis moves in the positive direction at the configured Jog Acceleration and Jog Velocity rates. Turning Jog Plus OFF causes the axis to decelerate and stop. If Jog Plus is momentarily turned off, even for one host controller sweep, the axis will decelerate to a stop then accelerate and continue jogging. The axis will move as long as the Jog Plus command is maintained and the configured **Positive End Of Travel** software limit or Positive Overtravel switch is not encountered. The Overtravel switch inputs can be disabled using the **OT Limit** configuration parameter. **Jog Plus may be used to jog off of the Negative Overtravel**

switch if the Clear Error %Q bit is also maintained on. See Chapter 6, Non-Programmed Motion, for more information on Jogging with the DSM314.

- 3.10 **Jog Minus.** When this command bit is ON, the axis moves in the negative direction at the configured Jog Acceleration and Jog Velocity rates. Turning Jog Minus OFF causes the axis to decelerate and stop. If Jog Minus is momentarily turned off, even for one host controller sweep, the axis will decelerate to a stop then accelerate and continue jogging. The axis will move as long as the Jog Minus command is maintained and the configured **Negative End Of Travel** software limit or Negative Overtravel switch is not encountered. The Overtravel switch inputs can be disabled using the **OT Limit** configuration parameter. **Jog Minus may be used to jog off of the Positive Overtravel switch if the Clear Error %Q bit is also maintained on. See Chapter 6, “Non-Programmed Motion,” for more information on Jogging with the DSM314.**
- 3.11 **Reset Strobe 1, 2 Flag.** The Strobe n Flag %I status bit flag informs the host controller that a Strobe Input has captured an axis position that is now stored in the associated Strobe n Position %AI status word. When the host controller acknowledges this data, it may use the Reset Strobe n Flag %Q command bit to clear the Strobe n Flag %I status bit flag. Once the Strobe n Flag %I bit is set, additional Strobe Inputs will not cause new data to be captured. The flag must be cleared before another Strobe Position will be captured. As long as the Reset Strobe n Flag %Q command bit is set, the Strobe n Flag bit will be held in the cleared state. In this condition, the latest Strobe Input position is reflected in the Strobe n Position status word, although the flag cannot be used by the host controller to indicate when new data is present.
- 3.12 **OUT1_A, B, C, D Output Control / Configurable CTL Bit Source.** Each axis connector has a 24-vdc solid state relay (SSR) output rated at 125 ma. The OUT1_A, OUT1_B, OUT1_C and OUT1_D Output Control %Q bits can control the state of the associated output, but only if the associated Output Bits configuration is set for host controller Control. Refer to Chapter 4 for configuration information.

For each axis, the following connector terminals are assigned:

	Faceplate Connector Pin	Auxiliary TB IC693ACC336 Terminal	Servo TB IC693ACC335 Terminal
OUT1 SSR (+) terminal	18	18	18
OUT1 SSR (-) terminal	36	36	16

These %Q bits are also available as sources for configurable CTL bits, independent of the Output Bits configuration. Refer to Chapter 4 for information on configuring the CTL01-CTL24 bit sources.

Note: The OUT_1A, B, C, D bits will not control the faceplate outputs unless the associated Output Bits configuration is set for host controller Control. Refer to Chapter 4 for configuration information.

- 3.13 OUT3_A, B, C, D Output Control / Configurable CTL Bit Source.** Each axis connector has a differential 5-vdc output that is suitable for driving 5v TTL or CMOS loads. The OUT3_A, OUT3_B, OUT3_C and OUT3_D Output Control %Q bits control the state of the associated output, but only if the associated Output Bits configuration is set for PLC Control. Refer to Chapter 4 for configuration information.

For each axis the following connector terminals are assigned:

	Faceplate Connector Pin	Auxiliary IC693ACC336 Terminal	TB Servo TB IC693ACC335 Terminal
OUT3 (+) terminal	14	14	5
OUT3 (-) terminal	32	32	13

Note: The OUT_3A, B, C, D bits will not control the faceplate outputs unless the associated Output Bits configuration is set for PLC Control. Refer to Chapter 4 for configuration information.

These %Q bits are also available as sources for configurable CTL bits, independent of the Output Bits configuration. Refer to Chapter 4 for information on configuring the CTL01-CTL24 bit sources.

- 3.14 Enable Follower.** When this bit is set and the Follower Enabled %I status bit indicates the Follower is enabled, motion commanded by the external or internal master will act as an input to the follower loop. An optional Follower Trigger bit may be configured to initiate follower motion. When a Follower Trigger is used, Enable Follower must be ON for the trigger condition to be tested. Clearing Enable Follower disconnects the follower loop from the master source. Jog, Move at Velocity, and Execute Program n commands will be allowed regardless of the state of Enable Follower. When the Follower is enabled, Jog, Move at Velocity, or Execute Program n commands will be superimposed on the master velocity or position command. Find Home is not allowed unless Enable Follower is cleared. Refer to Chapter 8 for additional information. This bit is only used by follower mode.
- 3.15 Select Follower Master Source.** This bit switches the follower master axis source from Follower Master Source 1 (bit OFF) to Follower Master Source 2 (bit ON). The Follower Master sources are configurable as Commanded Position or Actual Position from any of the 4 axes.

5.4 Section 4: %AQ Immediate Commands

The following %AQ Immediate Command words are transferred each host controller sweep from the CPU %AQ data to the DSM314. The number of %AQ words configured (6, 9, or 12) depends upon the number of controlled axes configured. The actual addresses of the Immediate Command words depend on the starting address configured for the %AQ words. See Table 40, “Settings Tab.” The word offset numbers listed in the following table are offsets to this starting address. The words are assigned as follows:

Table 45: %AQ Word Assignments

Word Offset	Description	Axis
00	Immediate Command Word	Servo 1
01-02	Command Data	Servo 1
03	Immediate Command Word	Servo 2
04-05	Command Data	Servo 2
06	Immediate Command Word	Servo 3
07-08	Command Data	Servo 3
09	Immediate Command Word	Servo 4
10-11	Command Data	Servo 4

Only one %AQ Immediate command may be sent to each axis of the DSM314 every host controller sweep, the only exception being the Load Parameter Immediate command, which is axis independent. The number of Load Parameter Immediate commands that can be sent in one sweep depends upon the number of %AQ words configured (see Table 47 for details).

Even though the commands are sent each sweep, the DSM314 will act on a command ONLY if it changed since the last sweep. When any of the 3 words change, the DSM314 will accept the data as a new command and respond accordingly.

The Axis OK %I bit must be ON for an axis to accept a new %AQ Immediate Command. Under some conditions such as a disconnected digital encoder, un-powered servo amplifier, or un-cleared error, Axis OK will be OFF and the %AQ command processing for that axis will be disabled. If Digital Servo Axis 1 or 2 is not used for motor control, the configured **Motor Type** must be set to 0 or an error will be reported, and Axis OK will stay OFF.

The 6-byte format for the Immediate Commands is defined in Table 46. The actual addresses of the Immediate Command Words depend on the starting address configured for the %AQ references. **The word numbers listed in the following table are offsets to this starting address.**

The word offsets are shown in reverse order and in hexadecimal to simplify the data entry. The following example sends the Set Position command to axis 1. The first word, word 0, contains the actual command number. For the Set Position command, the command number is 0023h. The second and third words contain the data for the Set Position command that is a position. The second word, word 1, is the least significant word of the position and the third word, word 2, is the most significant word.

Example:

To set a position of 3,400,250, first convert the value to hexadecimal. 3,400,250 decimal equals 0033E23A hexadecimal. For this value, 0033 is the most significant word and E23A is the least significant word. The data to be sent to the DSM314 would be:

Word 2	Word 1	Word 0	Command
0033	E23A	0023	Set Position 3,400,250

Setting up word 0 as a hexadecimal word and words 1 and 2 as a double integer in a Reference View Table display will simplify immediate command entry.

The data limit values **MaxPosnUu**, **MaxVelUu** and **MaxAccUu** are computed as shown below:

Formulas for Computing Data Limit Variables		
Position Limit MaxPosnUu	Velocity Limit MaxVelUu	Acceleration Limit MaxAccUu
If uu:cts >= 1:1 MaxPosnUu = 536,870,912 Else (uu:cts < 1:1) MaxPosnUu = 536,870,912 * uu/cts	MaxVelUu = 1,000,000* uu/cts	If uu:cts >= 1:1 MaxAccUu = 1,073,741,823 Else (uu:cts < 1:1) MaxAccUu = 1,073,741,823* uu/cts

In the following %AQ command table, only the word offsets for Servo Axis 1 are listed. Word offsets for the other axes are computed by adding 3 (Servo Axis 2), 6 (Servo Axis 3), or 9 (Servo Axis 4) to the listed word offsets. The Ref column numbers refer to sections in this chapter.

Table 46: %AQ Immediate Commands Using the 6-Byte Format

Word 2		Word 1		Word 0		Immediate Command Definition	Ref
Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0		
xx	xx	xx	xx	00	00h	Null	4.01
xx	xx	xx	RO%	00	20h	Rate Override RO% = 0 ... 120%	4.02
xx	xx	*	Incr	00	21h	Position Increment Without Position Update Incr. = -128 ... +127 User Units	4.03
Velocity				00	22h	Move At Velocity Vel. = -MaxVelUu ... +MaxVelUu	4.04
Position				00	23h	Set Position Pos. = -MaxPosnUu ... + MaxPosnUu-1	4.05
xx	xx	Analog Output		00	24h	Force Analog Output Analog Output = -32,000 ... + 32,000	4.06
xx	xx	*	Incr.	00	25h	Position Increment With Position Update Incr. = -128 ... +127 User Units	4.07
xx	xx	xx	In Posn Zone	00	26h	In Position Zone Range = 0 ... 255	4.08
Position or Parameter #				Move Type	27h	Move Command Pos. = -MaxPosnUu ... + MaxPosnUu-1 Par # = 0 ... 255	4.09
Velocity				00	28h	Jog Velocity Vel. = +1 ... +MaxVelUu	4.10
Acceleration				00	29h	Jog Acceleration Acc. = +1 ... + MaxAccUu	4.11
xx	xx	Time Constant (0.1 ms units)		00	2Ah	Position Loop Time Constant Time Constant = 0 - 65535 (0.1 ms units)	4.12
xx	xx	VFF (0.01% units)		00	2Bh	Velocity Feedforward VFF = 0 ... 12000 (0.01% units)	4.13
xx	xx	Integr. TC		00	2Ch	Integrator Time Constant Time Constant = 0, 10 ... 10,000 ms	4.14
Ratio B		Ratio A		00	2Dh	Follower A/B Ratio Ratio A = -32,768 ... +32,767 Ratio B = +1 ... +32,767	4.15
xx	xx	xx	VLGN	00	2Eh	Velocity Loop Gain (Digital mode only) VLGN = 0 ... 255	4.16
xx	xx	Torque Limit (0.01% units)		00	2Fh	Torque Limit (Digital mode and Analog Torque Mode only) Range = 0-10000 (0.01% units)	4.17
Position				00	31h	Set Aux Encoder Position	4.18

Word 2		Word 1		Word 0		Immediate Command Definition	Ref
Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0		
						Pos. = -MaxPosnUu ... + MaxPosnUu-1	
xx	xx	Servo Velocity Cmd		00	34h	Force Servo Velocity Servo Velocity Cmd = -4,095 ... +4,095 RPM	4.19
		<i>Note: Not used in Analog Velocity Mode - See Force D/A Output command</i>					
xx	xx	Offset		Mode	40h	Select Return Data 1	4.20
xx	xx	Offset		Mode	41h	Select Return Data 2	4.21
xx	xx	Make-Up Time		00	42h	Follower Ramp Distance Make-Up Time Active Range = 0, 10 ... 32000 ms	4.22
xx	xx	KpVel		07	46h	Velocity Regulator Proportional Gain (Analog Torque Mode Only) KpVel = 0 - 32767	4.23
xx	xx	KiVel		08	46h	Velocity Regulator Integral Gain (Analog Torque Mode Only) KiVel = 0 - 32767	4.24
xx	xx	TqFilt mode		0A	46h	Torque Command Filter (Analog Torque Mode Only) TqFilt = 0 - 3	4.25
xx	xx	Mode		Axis	47h	Select Analog Output Mode (Digital mode only)	4.26
xx	xx	xx	xx	00	49h	Clear New Configuration Received	4.27
Parameter Data				Par #h	50h	Load Parameter Immediate Par # = 0 ... 255 Parameter Data = Range depends on parameter usage.	4.28

* = Only 00 or FFh are acceptable.

xx = don't care

4.01 Null. This is the default %AQ Immediate command. Since the %AQ words are automatically transferred each CPU sweep, the Null command should always be used to avoid inadvertent execution of another %AQ Immediate command.

4.02 Rate Override. This command immediately changes the % feedrate override value, which will modify the commanded velocity for all subsequent programmed moves. This new value will become effective immediately when received by the DSM314. It is stored and will remain effective until overwritten by a different value. A rate override has no effect on non- programmed motion or acceleration. Rate Override is set to 100% whenever a program is initiated. The Rate Override command can be sent on the same CPU sweep as an Execute Program %Q bit and the Override value

will immediately take effect. Rate Override can be used to adjust the programmed velocity (not acceleration) of a particular move or a set of moves on any given axis.

- 4.03 Position Increment Without Position Update.** (User units) This command offsets the axis position from -128 to +127 user units without updating the Actual Position, Unadjusted Actual Position (UAP), or Commanded Position. The DSM314 will immediately move the axis by the increment commanded if the servo is enabled. Position Increments can be used to make minor machine position corrections to compensate for changing actual conditions. See Chapter 6, “Non-Programmed Motion,” for more information on using Position Increment Commands with the DSM314.

Note: *The %AQ Position Increment without Position Update command (21h) does not change the UAP. If an application uses this command, the UAP will no longer match Actual Position. For details on the operation of UAP, see page 156.*

- 4.04 Move At Velocity.** (User units/sec) This command is executed from the CPU to move the axis at a constant velocity. The active **Jog Acceleration** rate and configured **Jog Acceleration Mode** are used for Move at Velocity commands. Axis actual position data will roll over at the configured Hi or Lo Limit when reached during these moves. **See Chapter 6, “Non-Programmed Motion, for more information on the Move at Velocity Command.”**

- 4.05 Set Position.** (User units) This command changes the axis position register values without moving the axis. Operation of the command depends on the axis configuration:

Servo Axis - The Commanded Position and Actual Position values will both be changed so that no motion command will be generated. The Actual Position will be set to the value designated and the Commanded Position will be set to the value + Position Error. Set Position cannot be performed when the Moving %I bit or the Program Active %I bit is ON. Set Position is allowed if the In Zone %I bit is OFF as long as Actual Velocity is ≤ 100 cts/sec. The position value must be within the End of Travel Limits and Count Limits or a status error will be reported. The Position Valid %I bit is set after a successful Set Position command. See Appendix C for considerations when using absolute mode encoders. The Set Position command is commonly used to set the starting position reference point to zero (or another value) without homing the axis.

Aux Axis - Commanded Position is set to the command data. For an Aux Axis, Actual Position is independent of Commanded Position and is not affected by Set Position. **Refer to paragraph 4.18 Set Aux Encoder Position to set Actual Position for an Aux axis encoder.** Set Position cannot be performed when the Moving %I bit or the Program Active %I bit is ON. The position value must be within the End of Travel Limits or a status error will be reported.

Note: *When a digital servo system with absolute encoder (Feedback Mode = Absolute) is first powered up after removal or replacement of the encoder battery, the encoder must be rotated past its internal reference point. If this is not done the Set Position command will be ignored and Error Code 53h (Set Position before encoder passes reference point) will be reported.*

4.06 Force Analog Output. Each axis connector supports one analog output signal. The Force Analog Output immediate command may be used in the CPU application program to set the value of this DC voltage output. The Force Analog Output command operates one of the analog outputs on DSM faceplate connector C or D in Digital mode, or in Analog Velocity mode, on connector A, B, C, or D. Multiple Force Analog Output commands can be used to operate outputs on different connectors by using the appropriate %AQ word offsets (see the paragraph before Table 46). A Force Analog Output command has a range of +32000 (+10.00 Vdc) to -32000 (-10.00 Vdc). When the axis is configured for Analog Torque mode the Force Analog output command is NOT available.

Note: *It is necessary to enable the applicable %Q “Enable Drive” bit (there is one for each axis) to activate the analog output value set by this command. This differs from IC693DSM302 functionality.*

There are two requirements to sustain the forced analog output voltage: (1) the Force Analog Output command and value must remain continuously in the %AQ data, and (2) the associated %Q “Enable Drive” bit must be on. The %Q “Enable Drive” bit can be used to switch the analog output voltage on and off.

When a Force Analog Output command is active for a given axis, any other %AQ immediate command for that axis will remove the Force Analog Output command and turn off the associated analog output.

There are some differences between the Digital and Analog Axis Modes when using this command, which are detailed below:

Digital Mode

- The Force Analog Output command can only be used on connectors C and D in Digital mode (in Digital mode, both Axis 1 and Axis 2, on connectors A and B respectively, must be digital). In fact, Force Analog Output is the default signal on connectors C and D in Digital mode.
- If Axes 1 and 2 (connectors A and B) are configured for digital servo, their analog outputs are used only for servo tuning, and this function cannot be overridden by the Force Analog Output command. Issuing a Force Analog Output command to a digital axis (connector A or B) will have no effect, and no error will be reported.
- In Digital mode, a Force Analog Output signal can be overridden if another signal is routed to connector C or D by the Select Analog Output Mode command. If the default Force Analog Output command has been overridden on connectors C or D, it can be reinstated by either (1) issuing the immediate command Select Analog Output (Signal Code 00) to each affected axis or (2) power cycling the DSM314. See Section 4.25, “Select Analog Output Mode.”

Force Analog Output (Digital Mode) Example

In this example, Axes 1 and 2 are configured as Digital, the beginning DSM314 %Q address is configured as %Q1, and the beginning %AQ address is configured as %AQ1. Connectors C and D are set at their default analog output condition (Force Analog Output).

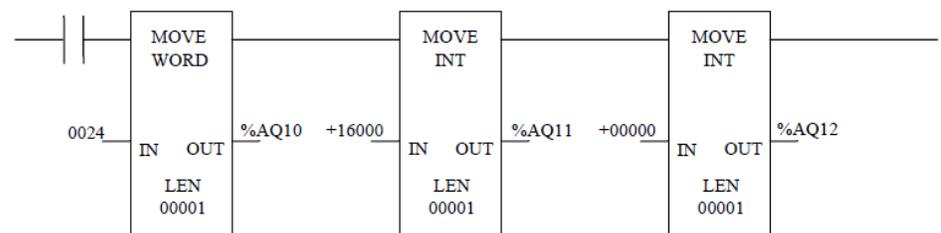
To force an analog output of +5VDC on connector D, the Force Analog Output immediate command will be issued in the ladder logic program. Since the first %AQ word was configured as %AQ1, the three words that apply to Connector D (“Axis 4”), are %AQ10, %AQ11, and %AQ12 (see the paragraph above Table 46 for details). Since %Q1 was configured as the first %Q bit, the Enable Drive (Servo 4) bit for Axis 4 is %Q67 (see Table 44, “%Q Discrete Commands”).

So the following values must be moved into the applicable words, using Move instructions in ladder logic (using a WORD type Move instruction makes it easier to move a hex number):

- %AQ10 Set to 24h (which specifies the Force Analog Output command)
- %AQ11 Set to +16000 (which equals +5VDC)
- %AQ12 Set to 0 (this word is not used to convey significant data)

Additionally, the %Q67 bit (Enable Drive) must be set to logic 1.

Figure 62



Analog Velocity Mode

- In Analog Velocity mode, the Force Analog Output command can be used on all four connectors to force a voltage output.
- The Select Analog Output command, discussed in the “Digital Mode” section above, does not work in Analog mode.

Analog Torque Mode

- In Analog Torque mode, the Force Analog Output command is NOT available.

4.07 Position Increment with Position Update. (User units) This command is similar to the Position Increment Without Position Update command (#21h) except that Actual Position and Commanded Position (returned in %AI data) are both updated by the increment value. If the servo is enabled, the DSM314 will immediately move the axis by the increment value. Position Increments can be used to make minor machine position corrections to compensate for changing actual conditions. **See Chapter 6, “Non-Programmed Motion, for more information on Position Increment Commands with the DSM314.”**

4.08 In Position Zone. (User Units) This command can be used to set the active **In Position Zone** to a value different than the configured value.

The DSM314 compares **In Position Zone** to the Position Error in order to control the In Zone %I bit. When the Position Error is \leq **In Position Zone**, the In Zone %I bit is ON.

If the DSM314 is power cycled or the host controller CPU is reset for any reason, the value set by this command will be lost and the In-Position zone value set by configuration software will be reinstated.

4.09 Move Command. This command will produce a single move profile that will move the axis to the position commanded each time it is sent. The current **Jog Acceleration** and **Jog Velocity** (which can also be changed by %AQ commands) will be used for the move. A PMOVE command does not complete (Program Active %I bit turns OFF) until Commanded Position has reached the destination and the In Zone %I bit is on. A CMOVE command completes (Program Active %I bit turns off) whenever Commanded Position reaches the destination even if In Zone is OFF. Therefore, a CMOVE will complete even if Actual Position has not yet reached the CMOVE destination. The Program Active %I bit can be monitored to determine when an AQ Move command is active.

The data field for this command may contain the move position or distance in bytes 2-5 with the command type (in hexadecimal format) as defined below:

Move Type (byte 1):

- 00h = Abs, Pmove, Linear
- 01h = Abs, Cmove, Linear
- 10h = Abs, Pmove, Scurve
- 11h = Abs, Cmove, Scurve
- 40h = Inc, Pmove, Linear
- 41h = Inc, Cmove, Linear
- 50h = Inc, Pmove, Scurve
- 51h = Inc, Cmove, Scurve

The data field for this command may contain a parameter number in byte 2 (bytes 3-5 unused) with the command type as defined below:

Move Type (byte 1):

- 80h = Abs, Pmove, Linear
- 81h = Abs, Cmove, Linear
- 90h = Abs, Pmove, Scurve
- 91h = Abs, Cmove, Scurve
- C0h = Inc, Pmove, Linear
- C1h = Inc, Cmove, Linear
- D0h = Inc, Pmove, Scurve
- D1h = Inc, Cmove, Scurve

The Move Command is executed as a single move motion program. Therefore, all the restrictions that apply to motion program execution also apply to the Move Command. For example, if a program is already active for axis 1, then an attempt to send this command for axis 1 will result in an error condition being reported.

- 4.10 Jog Velocity.** (User units/sec) This command sets the velocity used when a Jog %Q bit is used to jog in the positive or negative direction. Jog Velocity is used by motion programs when no Velocity command is included in the program. Jog Velocity is always used by the %AQ Move Command (27h). A host controller reset, or power cycle returns this value to the configured data.
- 4.11 Jog Acceleration.** (User units/sec/sec) This command sets the acceleration value used by Jog, Find Home, Move at Velocity, Abort All Moves and **Normal Stop** operations. A **Normal Stop** occurs when the host controller switches from Run to Stop or after certain programming errors (refer to Appendix A). Jog Acceleration is used by motion programs when no Acceleration command is included in the program. Jog Acceleration is always used by the %AQ Move Command (27h). A host controller reset, or power cycle returns this value to the configured data.

Note: A minimum value after scaling is used in the DSM314. This value is determined by the rule:

$$\text{Jog Acc} * (\text{user units/counts}) \geq 32 \text{ counts/sec/sec.}$$

- 4.12 Position Loop Time Constant.** (0.1 Milliseconds) This command allows the servo position loop time constant to be changed from the configured value. The lower the Position Loop Time Constant value, the faster the system response. Values that are too low will cause system instability and oscillation. For accurate tracking of the commanded velocity profile, the **Position Loop Time Constant** should be 1/4 to 1/2 of the MINIMUM system acceleration or deceleration time. For Analog mode, the “Vel at Max Cmd” configuration value must be set correctly for proper operation of the **Position Loop Time Constant**. A host controller reset, or power cycle returns this value to the configured data.

- 4.13 Velocity Feedforward.** This command sets the **Velocity Feedforward** gain (0.01 percent). It is the percentage of Commanded Velocity that is added to the DSM314 velocity command output. Increasing **Velocity Feedforward** causes the servo to operate with faster response and reduced position error. Optimum **Velocity Feedforward** values are 90-100 %. For analog servos, the “Vel at Max Cmd” configuration value must be set correctly for proper operation of the **Velocity Feedforward** gain factor. A host controller reset or power cycle returns this value to the configured data.
- 4.14 Integrator Time Constant.** (Milliseconds) This command sets the **Integrator Time Constant** for the position error integrator. The value specifies the amount of time in which 63% of the Position Error will be removed. The **Integrator Time Constant** should be 5 to 10 times greater than the **Position Loop Time Constant** to prevent instability and oscillation. **It is recommended that the position error integrator only be used in continuous follower applications. Use of the integrator in point to point positioning applications may result in position overshoot when stopping.**
- 4.15 Follower A/B Ratio.** This command allows the host controller to update the slave: master A/B ratio used in each follower loop. “A” is a 16-bit signed integer with a minimum value of - 32,768 and a maximum value of +32,767. “B” is a 16-bit integer with a minimum value of 1 and a maximum value of 32,767. The magnitude of the A/B ratio must be in the range 32:1 to 1:10,000 or a status error will be generated. Refer to Chapter 8 for additional information about the A/B ratio.
- 4.16 Velocity Loop Gain. (VLGN)** Digital Mode and Analog Torque Mode only. The velocity control loop gain for a digital servo axis and Analog Torque mode servo may be set with the Velocity Loop Gain command. The VLGN value is used to match the load inertia (J_L) to the motor inertia (J_M). VLGN is defined with a default value of 16 representing an inertia ratio of 1 to 1. The VLGN value is calculated assuming that the load is rigidly applied to the motor. Therefore, in actual machine adjustment the required value may significantly differ from the calculated value due to rigidity, friction, backlash, and other factors. A host controller reset or power cycle returns VLGN to the value set in the configuration software. A suggested starting point for Velocity Loop Gain is:

$$\text{Velocity Loop Gain} = \frac{\text{Load Inertia } (J_L)}{\text{Motor Inertia } (J_M)} \times 16$$

The allowed range of Velocity Loop Gain is 0 to 255.

For example: The motor inertia (J_M) of a particular servo is 0.10 lb-in-s². The load inertia (J_L) in this application is 0.05 lb-in-s². VLGN = (0.05 / 0.10) * 16 = 8

The default Velocity Loop Gain is set using the Velocity Loop Gain setting in the configuration software.

CAUTION

An incorrect VLGN value may cause an axis to be unstable. Care should be used when making any change to the VLGN value.

- 4.17 Torque Limit.** (0.01 percent) Digital Mode and Analog Torque Mode only. The Torque Limit Command provides a method of limiting the torque produced by the servomotor. In Analog Torque Mode, the Torque limit value limits the torque command to a percentage of the full-scale torque command value. Specifically, it limits the full scale of the analog output where full scale equals 10 volts. The DSM314 will set the Torque Limit at the default 10000 (100 %) whenever a power cycle or reset occurs. The host controller application logic must set any other value for desired Torque Limit. The valid range for Torque Limit is 0 to 10000 in units of 0.01%. This represents 0 - 100 % of peak torque at commanded velocity. If an over-range value of 10001 - 65535 is sent, the torque limit will be set to 10000. Torque Limit can be changed during axis motion and takes effect immediately. Refer to the appropriate servo motor manual for the motor torque curve to determine the actual value of torque output available at a given velocity. A simple example would be the use of Torque Limit to prevent over-tightening on a machine.
- 4.18 Set Aux Encoder Position.** (User Units) This command sets the Actual Position value for an Aux Axis Encoder without using a Find Home operation. The Position Valid % bit for the Aux Axis will be set when the command is received.
- 4.19 Force Servo Velocity.** (RPM) Digital Mode and Analog Torque Mode only. This command bypasses the position loop and forces a velocity command to the digital servo for tuning purposes. In Analog Torque Mode it bypasses the position loop and forces a velocity command to the velocity regulator. Acceleration control is not used and changes in velocity take effect immediately. A Force Servo Velocity command value of +4095 will produce a motor velocity of + 4,095 RPM and -4095 will produce a motor velocity of -4,095 RPM (depending on individual motor maximum velocities). The digital servo control loops may limit actual motor speed to a lower value. Care should be taken not to operate a servomotor past the rated duty cycle.
- The Enable Drive %Q bit must be active with no other motion commanded for the Force Servo Velocity command to operate. The command must remain continuously in the %AQ data for proper operation. When a Force Servo Velocity command is active for a given axis, any other %AQ immediate command for that axis will remove the Force Servo Velocity data and halt the servo. Chapter 6, Non-Programmed Motion, also contains information on Force Servo Velocity.
- 4.20 Select Return Data 1.** This command allows alternate data to be reported in the User Selected Data 1 %AI location for each axis. The alternate data includes information such as Parameter memory contents and the DSM314 Firmware Revision.

The Select Return Data 1 command uses a mode selection and an offset selection. The mode selection (byte offset +1 of the six-byte command) determines the Return Data type. The offset selection (byte offsets +2, +3 of six-byte command) selects an individual data item for some modes. Setting the mode to 00h causes the default

Torque Command to be reported. **The default mode and offset for User Selected Data 1 can be set in the module configuration software.**

- 4.21 Select Return Data 2.** This command allows alternate data to be reported in the User Selected Data 2 %AI location for each axis. The alternate data includes information such as Parameter memory contents and the DSM314 Firmware Revision.

The Select Return Data 2 command uses a mode selection and an offset selection. The mode selection (byte offset +1 of the six-byte command) determines the Return Data type. The offset selection (byte offsets +2, +3 of six-byte command) selects an individual data item for some modes. Setting the mode to 00h causes the default Torque Command to be reported. **The default mode and offset for User Selected Data 2 can be set in the module configuration software.**

The following selections are allowed for Select Return Data 1 and Select Return Data 2.

Return Data

Digital	Analog Torque	Analog Velocity	Selected Return Data	Data Mode	Data Offset
Y	Y	N	Torque Command	00h	not used
Y	Y	Y	DSM Firmware Revision	10h	not used
Y	Y	Y	DSM Firmware Build ID No. (hex)	11h	not used
Y	N	N	Absolute Feedback Offset (cts)	17h	not used
Y	Y	Y	Parameter Data	18h	Parameter Number (0-255)
Y	Y	Y	CTL bits 1-32	19h	not used
Y	Y	Y	Analog Inputs - Axis 1	1Ch	not used
Y	Y	Y	Analog Inputs - Axis 2	1Dh	not used
Y	Y	Y	Analog Inputs - Aux 3	1Eh	not used
Y	Y	Y	Analog Inputs - Aux 4	1Fh	not used
Y	Y	Y	Commanded Position (user units)	20h	not used
Y	Y	Y	Follower Program Command Position (cts)	21h	not used
Y	Y	Y	Unadjusted Actual Position (cts)	28h	not used
Y	Y	Y	Unadjusted Strobe 1 Position (cts)	29h	not used
Y	Y	Y	Unadjusted Strobe 2 Position (cts)	2Ah	not used

Torque Command is scaled so that +/- 10000 = +/- 100% torque.

DSM Firmware Revision is interpreted as two separate words for major-minor revision codes.

DSM Firmware Build ID is interpreted as a single hex word.

Absolute Feedback Offset is the position offset (in counts) that is used to initialize Actual Position when a digital Absolute Encoder is used. Actual Position = Absolute Encoder Data + Absolute Feedback Offset.

Analog Inputs provides two words of data for each axis: low word = AIN1 and high word = AIN2. The data is scaled so that +/- 32000 = +/- 10.0v.

Commanded Position (user units) is a copy of the Commanded Position %AI data reported for each axis. Refer to paragraph 2.04 in Chapter 5.

Follower Program Command Position (cts) is the active commanded position (in feedback counts) updated and used by the internal motion command generator. Refer to Chapter 9 - Combined Follower and Commanded Motion.

Unadjusted Actual Position (UAP) is the accumulated actual position (in counts, not user units) with a 32 bit binary rollover value of -2,147,483,648 ... +2,147,483,647. A Find Home or Set Position command sets the UAP to a value equal to the Actual Position data scaled to counts.

UAP is initialized or reset when a Set Position or Home operation is completed. It tracks actual motor rotation after these operations within the 32 bit raw encoder count range with rollover, regardless of how the rotation is commanded (except for the Position Increment without Position Update command).

If a Set Position command is executed or a Find Home cycle is completed, the UAP is set to the raw counts equivalent to the Set Position data or the configured Home Position after scaling by the Counts to User Units ratio on the individual axis tab of the DSM configuration. That is, the raw counts are calculated as:

$$\text{UAP} = (\text{Set Position data or Home Position value}) \times (\text{Counts}) / (\text{User Units}),$$

rounded to the nearest integer value.

The Home Offset configuration parameter, which is provided to allow the actual stopping position on a homing cycle to be offset from the encoder marker location, does not affect the UAP, since this added move takes place before the home position is set.

The UAP value is maintained through power cycles as long as the encoder backup battery power is maintained, even if the axis is moved while power is off. (The DSM reads the encoder absolute data as part of the power on sequence.) It is also maintained during axis E- Stop and emergency stop fault conditions including out of sync.

Note: *The %AQ Position Increment without Position Update command (21h) does not change the UAP. If an application uses this command, the UAP will no longer match Actual Position.*

Unadjusted Strobe 1 Position is the value of Unadjusted Actual Position captured when a Strobe 1 input occurs.

Unadjusted Strobe 2 Position is the value of Unadjusted Actual Position captured when a Strobe 2 input occurs.

Note: *At least three host controller sweeps or 10 milliseconds (whichever represents more time) must elapse before the new Selected Return Data is available in the host controller.*

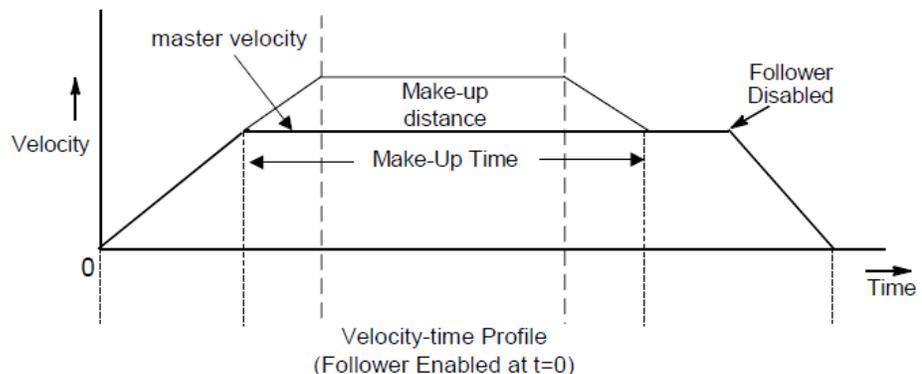
4.22 Follower Ramp Distance Make-Up Time. When the Follower Ramp feature has been selected and the follower is enabled, the following axis is ramped up to the Master velocity at the configured **Follower Ramp Acceleration** rate when the Master Velocity is non-zero at the time the Follower is enabled. The master counts that accumulate during acceleration of the follower axis are stored. In this mode, the follower axis will accelerate to a velocity that exceeds the Master Velocity in order to make up the position error that accumulated while the Follower axis was accelerating to the Master Velocity. This make-up distance correction has a trapezoidal velocity profile determined by the **Follower Ramp Distance Make-Up Time** and **Ramp Makeup Acceleration** at the beginning of the correction. This mode is used when the Follower axis must be position-and-velocity-synchronized to the Master position at the instant the Follower mode was enabled.

If the **Follower Ramp Distance Make-Up Time** is too short, then the velocity profile is a triangular profile. If during the distance correction, velocity exceeds 80% of the velocity limit, then the automatically calculated velocity will be clamped at 80% of the configured velocity limit. In both cases a warning message is reported, and the real distance make-up time is longer than programmed, but the distance is still corrected properly.

Setting a **Follower Ramp Distance Make-Up Time** of 0 allows the Ramp feature to accelerate the axis without making up any of the accumulated counts. In this instance, the Follower axis velocity will not exceed the master velocity. For applications where the Follower axis only needs to be synchronized to the master velocity and lost counts do not matter, set the distance make-up time = 0.

Typical velocity profile during the follower ramp cycle is shown below.

Figure 63



See Chapter 8, “Follower Motion, Follower Axis Acceleration Ramp Control” section, for a much more detailed discussion of this feature.

- 4.23 Velocity Loop Proportional Gain. Analog Torque Mode only.** The Velocity Loop Proportional Gain AQ command allows the user to set the velocity regulator proportional gain in Analog Torque mode. The proportional gain is multiplied by velocity error (velocity command - velocity feedback) to generate the portion of the torque command due to the proportional term. Correctly setting this value will determine how well the velocity regulator performs in the control system. Appendix D describes a method to correctly tune this parameter. The allowable range for the velocity loop proportional gain term is 0-32767. The default value is 1500.
- 4.24 Velocity Loop Integral Gain. Analog Torque Mode only** The velocity loop integral gain AQ command allows the user to set the velocity regulator integral gain in Analog Torque mode. The integral gain is the term multiplied by the area of the velocity error (velocity command - velocity feedback) to generate the portion of the torque command due to the integral term. Correctly setting this value will determine how well the velocity regulator performs in the control system. Appendix D describes a method to correctly tune this parameter. The allowable range for the velocity loop proportional gain term is 0-32767. The default value is 0
- 4.25 Torque Command Filter. Analog Torque Mode only.** The torque command filter AQ command allows the user to activate a low pass filter for the velocity regulator output (Torque Command). The filter is typically used to keep the controller from exciting a machine resonance. The allowable setting for the Torque Command filter are shown in Table 48.

Table 47: Torque Filter Commands

TqFilt Mode	Torque Command Low Pass Filter Setting
0	OFF1
1	Low Bandwidth Filter (150 hz 3db point)
2	Medium Bandwidth Filter (250 hz 3db point)
3	High Bandwidth Filter (350 hz. 3db point)

1 Default setting

- 4.26 Select Analog Output Mode. Digital Mode only.** For digital servos, this command lets you choose what analog signals will be sent to the Analog Output pins (pins 6 and 24) on the four DSM faceplate connectors. The Select Analog Output Mode command uses a Signal Code to specify the signal to be sent, and a Connector Code to specify the DSM connector to receive the signal. This command is particularly useful for servo tuning. This command can be sent from the Command registers for any axis (1-4).

Use the following structure to set up the 6-byte %AQ Immediate Command (described in Table 46):

- Byte 0 contains the Select Analog Output Mode command code (47h).
- Byte 1 contains the Connector Code, a hex number.
- Bytes 2-3 contain the Signal Code, a decimal number.
- Bytes 4-5 are not used and should contain 0.

Connector Codes

Connector Code	Connector Selected	Connector Pins
01h	Connector A	Pin 6 = OUT Pin 24 = COM (Ref. to 0V) Refer to the I/O Connection Diagrams in Chapter 3 for Terminal Board connections.
02h	Connector B	
03h	Connector C	
04h	Connector D	

Signal Codes

Note in the following Signal Code table that only some of the signals have a default output.

Signal Code	Signal Description	Default Output to:
00 decimal*	%AQ Force Analog Output data*	Connector C or D
10 decimal	Servo Axis 1 Torque Command	None
15 decimal	Servo Axis 1 Actual Velocity	Connector A
20 decimal	Servo Axis 2 Torque Command	None
25 decimal	Servo Axis 2 Actual Velocity	Connector B

- * Cannot be re-routed. This signal code can only be used to restore this signal back to its default output.

Note: *The analog output is not available for user control on digitally controlled axes. Issuing the Force Analog Output or the Select Analog Output commands for digital axes will have no effect on these analog outputs.*

The Select Analog Output Mode has three basic uses:

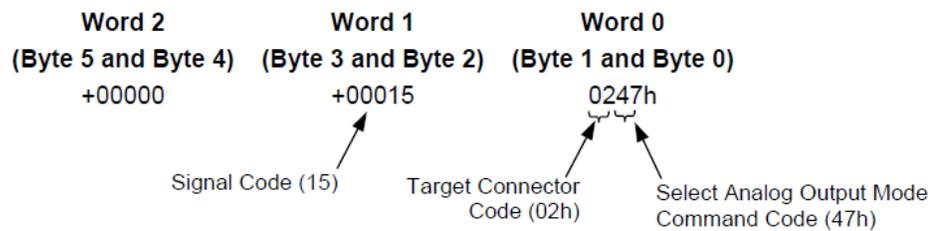
1. Re-route either Servo Axis 1 Actual Velocity or Servo Axis 2 Actual Velocity from its default output to a different output. The %AQ Force Analog Output data signal cannot be re-routed to a different connector; however, it can be replaced on its default output connector (C or D) by another signal that is routed there by the Select Analog Output Mode command.
2. Route one of the two signals lacking a default output, Servo Axis 1 Torque Command and Servo Axis 2 Torque Command, to one of the outputs, thus replacing the previous signal on that output. This is shown in Example 2, below.
3. Restore signals with default outputs that were replaced by a re-routed signal. In Example 2, the %AQ Force Analog Output signal, which is normally found

on Connector D by default, is replaced by the Servo Axis 1 Torque Command signal that was routed to connector D by the Select Analog Output Mode command. In Example 3, the %AQ Force Analog Output signal is restored to Connector D by using the Select Analog Output Mode command.

Example 1:

In this example, the Servo Axis 1 Actual Velocity signal (Signal Code=15) is re-routed from its default output on Connector A to Connector B (Connector Code=02h), replacing any previous signal on Connector B. This is accomplished by placing the following data in the %AQ immediate command words:

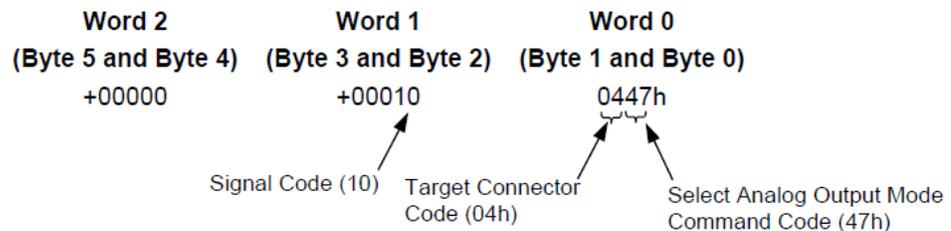
Figure 64



Example 2:

In this example, the Servo Axis 1 Torque Command signal (Signal Code=10) is selected as the Analog Output on Connector D (Connector Code=04h), replacing any previous signal on Connector D. To accomplish this, place the following data in the %AQ immediate command words:

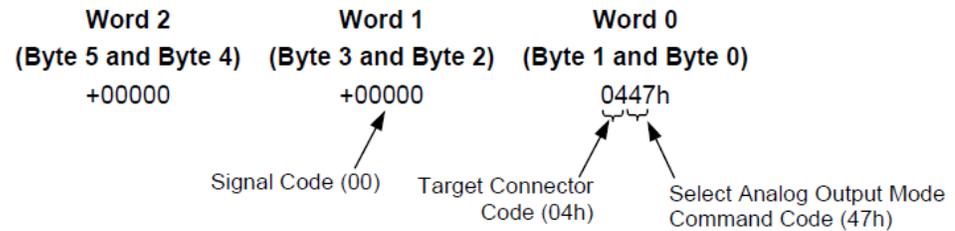
Figure 65



Example 3:

In Example 2, the %AQ Force Analog Output default signal was replaced as the Analog Output on Connector D by the Servo Axis 1 Torque Command signal. To restore the %AQ Force Analog Output signal (Signal Code=00) to Connector D (Connector Code=04h), place the following data in the %AQ immediate command words:

Figure 66



- 4.27 Clear New Configuration Received.** This command clears the New Configuration Received %I bit. Once cleared, the Configuration Complete bit is only set when the host controller resets or reconfigures the module. The host controller can monitor the bit to determine if it must re-send other %AQ commands, such as In Position Zone or Jog Acceleration. This would only be necessary if the %AQ commands were used to override DSM314 configuration data programmed with the host controller configuration software. This command can be sent from the Command registers for any axis (1-4).
- 4.28 Load Parameter Immediate.** This command is executed from the host controller to immediately change a DSM314 data parameter value. It can be sent from the Command registers for any axis (1-4). Data parameters are only used by motion programs. Each parameter change requires a command. Byte 1 of Word 0 contains the Parameter Number (in hexadecimal format) to be changed. The DSM314 contains 256 double word parameters, numbered 0-255 (decimal). For details, see “Parameters (P0-P255) in the DSM314” in chapter 7.

Table 48: Number of Load Parameter Immediate Commands Permitted per Sweep

Number of Axes Configured	Number of %AQ Words	Number of Load Parameter Immediate Commands Permitted per Sweep
2	6	2
3	9	3
4	12	4

Chapter 6: Non-Programmed Motion

The DSM314 can generate motion in an axis in one of several ways without using a motion program.

- Find Home and Jog Plus/Minus use the %Q bits to command motion.
- Move at Velocity, Move, Force Servo Velocity, Force Analog Output, and Position Increment use %AQ immediate commands.

During Jog, Find Home, Move at Velocity, Move and Force Servo Velocity, any other commanded motion, programmed or non-programmed, will generate an error. The only exception is the Position Increment %AQ command, which can be commanded any time. See the description of Position Increment motion below for more details.

Non-programmed motions (Abort All Moves, Jog Plus/Minus, Move at Velocity, AQ Move Cmd and Normal Stop) use the **Jog Acceleration** and **Jog Acceleration Mode**. The Feed Hold %Q command uses the programmed acceleration and acceleration mode.

6.1 DSM314 Home Cycle

A home cycle can be used to establish a correct Actual Position relative to a machine reference point. The configured **Home Offset** defines the location of Home Position as the offset distance from the Home Marker.

The Enable Drive %Q bit must be ON during an entire home cycle. However, the Find Home %Q bit does not need to be held ON during the cycle; it may be turned on momentarily with a one-shot. Note that turning ON the Find Home %Q bit immediately turns OFF the Position Valid %I bit until the end of the home cycle. The Abort All Moves %Q bit halts a home cycle, but the Position Valid bit does not turn back ON. No motion programs can be executed unless the Position Valid bit is ON.

6.1.1 Home Switch Mode

If the **Find Home Mode** is configured as HOMESW (HOME Switch), the Home Switch input from the axis I/O connector is used first to roughly indicate the reference position for home. Then, the next encoder marker encountered when traveling in the negative direction indicates the exact location. An open Home Switch input indicates the servo is on the positive side of the home switch and a closed Home Switch input indicates the axis is on the negative side of the home switch. An OFF to ON transition of the Find Home %Q command yields the following home cycle. Unless otherwise specified, acceleration is at the current **Jog Acceleration** and configured **Jog Acceleration Mode**.

Find Home Routine for Home Switch

If initiated from a position on the positive side of the home switch, in which case the home switch must be OPEN (Logic 0), the Find Home routine starts with step 1 below. (All of the first several steps of the following routine are necessary to allow for a variety of possible home switch designs and starting positions.) If the Find Home routine is initiated from a position on the negative side of the home switch, in which case the home switch must be CLOSED (Logic 1), the routine starts with step 3 below.

1. The axis is moved in the negative direction at the configured **Find Home Velocity** until the Home Switch input closes.
2. The axis decelerates and stops.
3. The axis is accelerated in the positive direction and moved at the configured **Find Home Velocity** until the Home Switch input opens.
4. The axis decelerates and stops.
5. The axis is accelerated in the negative direction and moved at the configured **Final Home Velocity** until the Home Switch input closes.
6. The axis continues negative motion at the configured **Final Home Velocity** until a marker pulse is sensed. The marker establishes the home reference position.
7. The axis decelerates and stops (at a position past the marker pulse).
8. The axis is moved, at the current **Jog Velocity**, the number of user units specified by the **Home Offset** value from the home reference position. If **Home Offset** = 0, the axis moves back to the position of the marker pulse.
9. The axis decelerates and stops.
10. The DSM314 sets the Commanded Position and Actual Position %AI status words to the configured **Home Position** value. Finally, the DSM314 sets the Position Valid %I bit to indicate the home cycle is complete.

Home Switch Example

Many different home switch designs are possible. The switch may be normally open or normally closed and may be mounted in one of several possible locations. The example given in this section illustrates a fairly common arrangement used for linear axes. In the following picture, the home switch is a normally open proximity switch, mounted near the end of the machine slide's travel range (in the negative direction). The imaginary line that divides the home switch's positive and negative sides is the home switch's operating point, located approximately on the switch's centerline. If the machine slide travels in the negative direction far enough so that the right-hand edge of the home switch cam causes the home switch to close, we consider the machine slide as having crossed over to the "negative side" of the home switch. **The home switch cam is long enough so that while the machine slide is on the negative side of the home switch, it will keep the normally open home switch closed.**

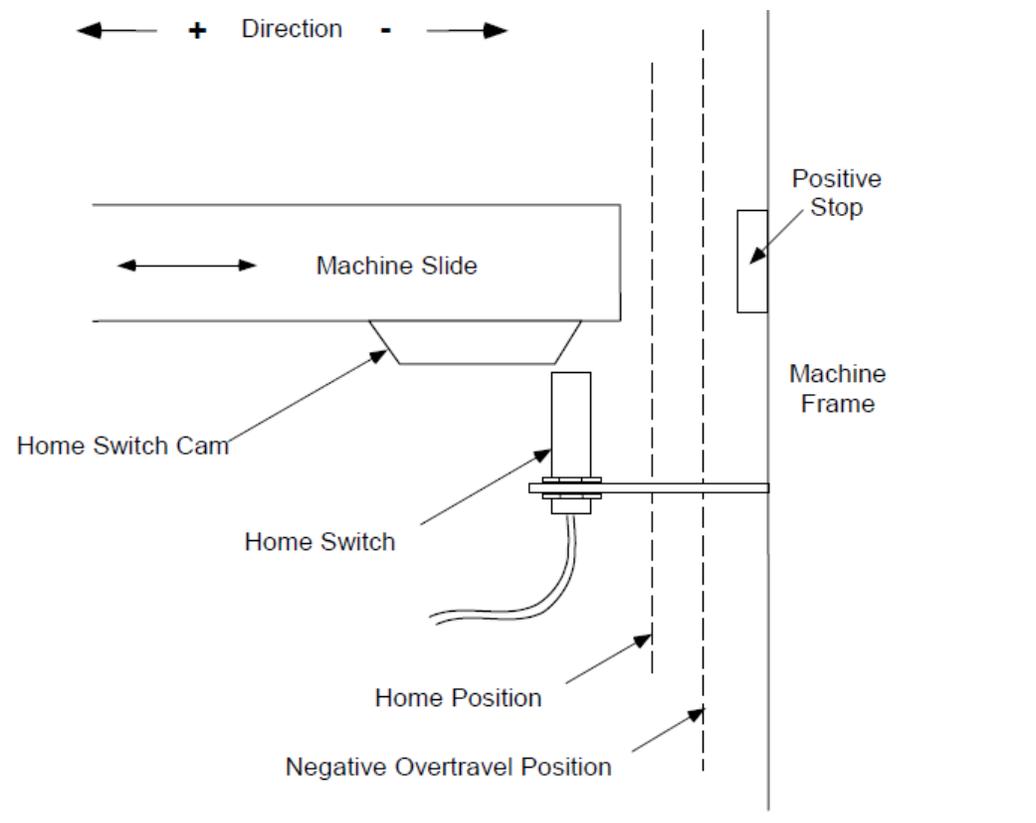
Note the relationships of the home position, the negative overtravel position, and the positive stop position. A small amount of distance is provided in the negative direction between the home position and the negative overtravel position. This is to allow some “working room” for adjustment and setup of these positions and for the “find home” routine, which requires that its final move be in the negative direction.

Distance is also provided between the overtravel limit position and the positive stop. Enough distance should be allowed here to prevent the machine slide from hitting the positive stop. The correct distance needs to be greater than the worst-case stopping distance required by the machine slide after it reaches the overtravel limit position.

In this example, the machine slide’s working range is on the positive side of the home switch. If the DSM’s Home Position parameter was set to 0, this would simplify programming absolute positioning commands since only positive numbers would be used.

Often, the home position needs to be set to an exact distance from a reference point on the machine. To facilitate this adjustment, the home switch cam could be made with slotted mounting holes that would allow a coarse adjustment of the cam to bring the calibration to within one turn of the encoder. Then, the small remaining distance would be accurately measured, and the value obtained would be entered into the DSM’s Home Offset parameter.

Figure 67: Home Switch Example



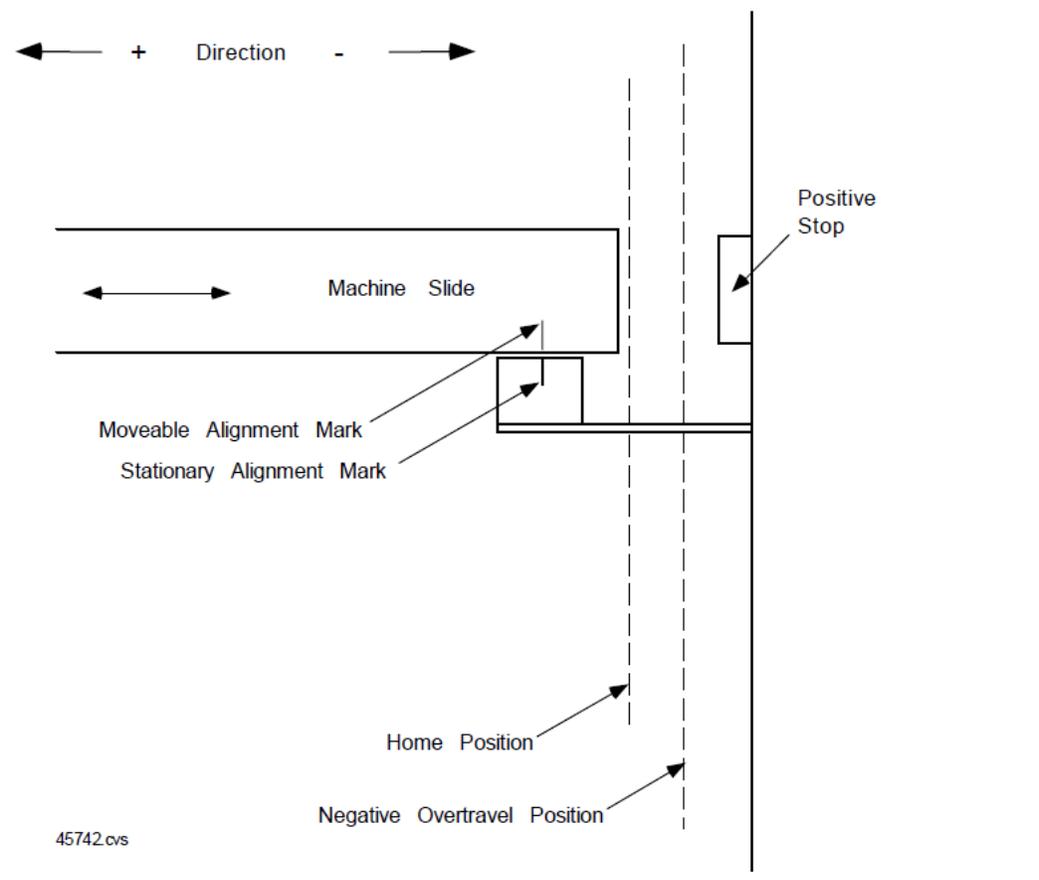
6.1.2 Move+ and Move- Modes

If **Find Home Mode** is configured as MOVE+ or MOVE-, the first encoder marker pulse encountered when moving in the appropriate direction (positive for MOVE+, negative for MOVE-) after the find home command is given is used to establish the exact location. In this mode, the operator usually jogs the axis to a position close (within one revolution of the encoder) to the home position first, then initiates the find home command. To assist the operator in jogging to the correct position, a set of alignment marks indicating a close proximity to the home position is sometimes placed on the machine and machine axis.

Move - (Minus) Home Cycle Example

The next picture shows an example of the Home Position parameter set to Move - (minus). In this example, the operator jogs the axis until the moveable mark on the machine slide lines up with the stationary mark on the alignment plate mounted to the machine frame. (Note that the marks align on the positive side of home position since the Home Position parameter is set to Move -). Then the operator initiates the find home routine, which causes the axis to move in the negative direction until the marker pulse occurs.

Figure 68: Move - (Minus) Home Position Example



Find Home Routine for Move + or Move –

When the find home command (an OFF to ON transition of the Find Home %Q bit) is initiated, the following sequence of events occurs:

1. The axis is accelerated at the **Jog Acceleration** rate and moved at the configured **Final Home Velocity** (positive direction for MOVE+, negative direction for MOVE–) until a marker pulse is sensed. This marker pulse establishes the home reference position.
2. The axis is stopped (at a position past the marker pulse) using the configured **Jog Acceleration** rate and with the configured Jog Acceleration Mode.
3. The axis is moved, at the configured **Jog Velocity** and with the configured **Jog Acceleration** rate and **Jog Acceleration Mode**, the number of user units specified by the **Home Offset** value from the home reference position. If **Home Offset** = 0, the axis moves back to the position of the marker pulse.
4. The axis is stopped at the configured **Jog Acceleration** rate and with the configured **Jog Acceleration Mode**.
5. The DSM314 sets the Commanded Position and Actual Position %AI status words to the configured **Home Position** value; the DSM314 sets the Position Valid %I bit to indicate the home cycle is complete.

6.2 Jogging with the DSM314

The **Jog Velocity**, **Jog Acceleration**, and **Jog Acceleration Mode** are configuration parameters in the DSM314. These values are used whenever a Jog Plus or Jog Minus %Q bit is turned ON. Note that if both bits are ON simultaneously, no motion is generated. The **Jog Acceleration** and **Jog Acceleration Mode** are also used during Find Home, Move at Velocity, Abort All Moves and Normal Stop. Programmed motions use the **Jog Velocity** and **Jog Acceleration** as defaults.

A Jog Plus/Minus %Q command can be performed when no other motion is commanded, or while programmed motion is temporarily halted due to a Feed Hold %Q command. The Enable Drive %Q bit does not need to be ON to jog, but it can be ON. Turning on a Jog Plus/Minus %Q bit will automatically close the Enable Relay and turn on the Drive Enabled %I bit. When an overtravel limit switch is OFF, Jog Plus/Minus and Clear Error %Q bits may be turned on simultaneously to move away from the open limit switch. Thus, a Jog Plus %Q command will not work while the positive end of travel switch is open and Jog Minus will not work while the negative end of travel switch is open. Turning a Jog %Q bit OFF causes the axis to decelerate and stop. If a Jog %Q bit is momentarily turned off, even for one CPU sweep, the axis will decelerate to a stop then accelerate and continue jogging.

6.3 Move at Velocity Command

A Move at Velocity %AQ command is generated by placing the value 22h in the first word of %AQ data assigned to an axis. The second and third words together represent a signed 32-bit velocity. Note that the third word is the most significant word of the velocity. Once the command is given, the %AQ data can be cleared by sending a NULL command or changed as desired. Move at Velocity will not function unless the servo drive is enabled (Enable Drive %Q command and Drive Enabled %I status bit are set).

The listing of %AQ immediate commands shows the words in reverse order to make understanding easier. For example, to command a velocity of 512 user units per second in a DSM314 configured with %AQ data starting at %AQ1, the following values should be used: 0022h (34 decimal) in %AQ1, 0200h (512 decimal) in %AQ2, and 0 in %AQ3. When the DSM314 receives these values, if Drive Enabled %I is ON, Abort All Moves %Q is OFF, and no other motion is commanded it will begin moving the axis at 512 user units per second in the positive direction using the current **Jog Acceleration** and **Acceleration Mode**.

The Drive Enabled %I bit must be ON before the DSM314 receives the immediate command or an error will occur. Also, if a Move at Velocity command is already in the %AQ data, the velocity value must change while the Drive Enabled bit is ON for the DSM314 to accept it. The DSM314 detects a Move at Velocity command when the %AQ values change.

When the DSM314 is performing a Move at Velocity command, it ignores the software end of travel limits (**Pos EOT** and **Neg EOT**). Hardware overtravel limits must be ON if they are enabled.

A Move at Velocity command can be stopped without causing an error in two ways: a Move at Velocity command with a velocity of zero, or turning the Abort All Moves %Q bit ON for at least one CPU sweep.

6.4 Force Servo Velocity Command (DIGITAL Servos; Analog Torque Mode)

This command bypasses the position loop and forces a velocity RPM command to the digital servo or Analog Torque Interface for tuning purposes. Acceleration control is not used and changes in velocity take effect immediately. A Force Servo Velocity command value of +4095 will produce a motor velocity of + 4,095 RPM and -4095 will produce a motor velocity of -4,095 RPM (depending on individual motor maximum velocities). The digital servo control loops may limit actual motor speed to a lower value.

⚠ CAUTION

Care should be taken not to operate a servomotor beyond its rated duty cycle.

The Enable Drive %Q bit must be active with no other motion commanded for the Force Servo Velocity command to operate. The command must remain continuously in the %AQ data for proper operation. When a Force Servo Velocity command is active for a given axis, any other %AQ immediate command for that axis will remove the Force Servo Velocity data and halt the servo. A one-shot Force Servo Velocity command will therefore only operate during the sweep in which it appears.

Refer to Chapter 5, Motion Mate DSM314 to Host Controller Interface, for more information on this command.

Note: *The Force Analog Output command, described below, is used for analog servos with a Velocity command interface.*

6.5 Force Analog Output Command (ANALOG Velocity Interface Servos)

In Analog Velocity Interface mode, the Force Analog Output %AQ immediate command operates the analog output on the DSM faceplate connectors A, B, C, or D. A Force Analog Output value of +32000 will produce +10.00 Vdc and a Force Analog Output value of -32000 will produce -10.00 Vdc.

Force Analog Output operates only while the %AQ data is active. When a Force Analog Output command is active for a given axis, any other %AQ immediate command for that axis will remove the Force Analog Output command and turn off the associated analog output.

Refer to Chapter 5, “Motion Mate DSM314 to Host Controller Interface”, for more information on this command.

6.6 Position Increment Commands

To generate small corrections between the axis position and the DSM314 tracking, the Position Increment %AQ commands can be used to offset Actual Position by a specific number of user units. If the Drive Enabled %I bit is ON, the axis will immediately move the increment amount. If the position increment without position update is used (%AQ command 21h), the Actual Position %AI status word reported by the DSM314 will remain unchanged. If the Position Increment with Position Update is used (%AQ command 25h), the Actual Position and Commanded Position %AI status words reported by the DSM314 will be changed by the increment value. Position Increment can be used at any time, though simultaneous use with the Force Servo Velocity command is impossible because the Force Servo Velocity command must remain in the %AQ command data area or the servo will be stopped.

6.7 Other Considerations

Other considerations when using non-programmed motion are as follows:

- The Abort All Moves %Q bit, when ON, will prevent any non-programmed motion from starting.
- Turning ON the Abort All Moves %Q bit will immediately stop any current non-programmed motion at the current **Jog Acceleration**.
- A Set Position %AQ command during non-programmed motion will cause a status error.
- Turning OFF the Enable Drive %Q bit while performing a home cycle or executing a Move at Velocity %AQ command will cause a stop error.
- The Feed Hold %Q bit has no effect on non-programmed motion.
- The Rate Override %AQ command has no effect on non-programmed motion.
- Changing the **Jog Velocity** or **Jog Acceleration** will not affect moves in progress.

Chapter 7: Programmed Motion

A motion program consists of a group of user-programmed motion command statements that are stored to and executed in the DSM314. The DSM314 executes motion program commands sequentially in a block-by-block fashion once a program is selected to run.

The motion program is executed autonomously from the host controller, although the host controller starts the DSM314 motion program and can interface with it (with parameters and certain commands) during execution. In addition, external inputs (CTL bits) connected directly to the DSM314 faceplate or controlled by Local Logic can be used in motion programs to delay or alter program execution flow. The host controller receives status information (such as position, velocity, and Command Block Number) from the DSM314 during program execution. Motion programs 1–10 and subroutines 1–40 are created using the host controller programming software and are stored along with the module's configuration settings to the DSM314 via the host controller backplane.

For further information, please refer to the online help for your software, or the software user manual, PAC Machine Edition Logic Developer-PLC Getting Started, GFK-1918.

7.1 Single-Axis Motion Programs and Subroutines

A single-axis program contains program statements for one axis only. The programmed axis is specified in the first line of the program, for example: PROGRAM 1 AXIS1. The DSM314 may operate up to four single-axis programs. These programs may run independently or simultaneously. For example, motion Program 1 may be written for Axis 1 and motion Program 2 written for Axis 2. Each axis may be home referenced and the motion program for each axis may execute independently without regard to the state of the other axis. Alternately, Program 1 and Program 2 may start simultaneously (via the run program %Q bits) during the same CPU sweep.

DSM314 motion programs support the subroutine feature, which may include all the available motion program commands including the CALL command. The SYNC Block command is reserved for multi-axis (Axis 1 and 2) programs and subroutines. Subroutines can be nested, using CALL statements, to a maximum of eight levels. Single-axis subroutines, similar to motion programs, contain commands for only one axis. The difference is that the axis number is not specified in a single-axis subroutine. A single-axis motion program may CALL any single-axis subroutine stored in module memory. For example, single-axis motion Program 1, operating Axis 1, may include a CALL statement to single-axis Subroutine 1. Additionally, single-axis motion Program 2, operating Axis 2, may include a CALL statement to single-axis Subroutine 1. Single-axis motion programs cannot CALL multi-axis subroutines.

The motion program and subroutine structure allow flexibility in execution and axis control in the DSM314 module. The practical limitation is that each axis may only execute one program at a time. For example, if Program 1 is enabled to run in Axis 1, it must either complete or abort prior to enabling Program 2 to run in Axis 1.

7.2 Multi-Axis Motion Programs and Subroutines

The term multi-axis is specified in the definition statement (on the first line) of a program or subroutine, for example: PROGRAM 2 MULTI-AXIS, or SUBROUTINE 7 MULTI-AXIS. Axis 1 and Axis 2 are the only two axis numbers permitted in a multi-axis program or subroutine. Both axes must be home referenced and meet the remaining prerequisites (see the section “Prerequisites for Programmed Motion” on page 173) before a program can be executed. A multi-axis motion program may CALL only multi-axis subroutines. One motion program instruction, SYNC Block, is available only in a multi-axis motion program or subroutine. Subroutine “nesting” limitations are the same as for a single-axis motion program. In a multi-axis program, there are two categories of moves: 1-Axis moves, and 2-Axis moves.

1-Axis moves: When two consecutive 1-Axis moves are programmed, the second move will begin execution within 2 milliseconds after the first move finishes.

2-Axis moves: A 2-Axis move is programmed with three consecutive blocks. The first of the three blocks must contain the SYNC Block command. The next two blocks contain the move commands, one for Axis 1, and one for Axis 2. When the SYNC Block command is executed, the two moves will be started “together” (within 2 milliseconds). Note that only the start of the moves is synchronized.

More information about multi-axis programming, program block structure, flow control (JUMP), and the SYNC Block command, is provided later in this chapter.

7.3 Motion Program Command Types

The motion program commands are grouped into four categories:

Type 1 Commands

CALL (Subroutine) JUMP

Type 2 Commands

Block number

SYNC (Block Synchronization)

LOAD (Parameter)

ACCEL (Acceleration)

VELOC (Velocity)

Type 3 Commands

PMOVE (Positioning Move)

CMOVE (Continuous Move)

DWELL

WAIT

Program/Subroutine Definition Commands

PROGRAM

ENDPROG

SUBROUTINE

ENDSUB

Type 1 commands can redirect the program path execution, but do not directly affect positioning.

- Call (Subroutine) executes a subroutine before returning execution to the next command.
- Jumps may be conditional or unconditional. An unconditional jump always redirects execution to a specified program location. A conditional jump is assigned a CTL bit to check. If the CTL bit is ON, the jump redirects execution to a specified program location. If the CTL bit is OFF, the jump is ignored.

Type 2 commands also do not affect position.

- Block numbers provide an identification or label for the Type 3 command that follows. Block numbers are required with JUMP commands; otherwise, they are optional. If a program block does not contain a block number, the previous block number, if any, remains in effect.
- The SYNC (synchronize block) command is a two-axis synchronization command (this may or may not delay motion on one axis).
- The Load Parameter command allows the user to load a value into a parameter register.
- The Velocity (VELOC) and Acceleration (ACCEL) commands specify velocity and acceleration rates for the Type 3 MOVE command or commands that follow. Velocity and Acceleration commands remain in effect until changed.

Type 3 commands start or stop motion and thus affect positioning control.

- Positioning (PMOVE) and Continuous (CMOVE) moves command motion.
- The Dwell, Wait, and End of Program commands stop motion.

7.4 Program Blocks and Motion Command Processing

A program block consists of and is defined as one (and only one) Type 3 command with any number and combination of preceding Type 1 and 2 commands.

A block number has two primary uses: (1) it provides a Jump-To identification (label), and (2) it identifies the section of the program that is currently executing via the Block Number %AI Status words for each axis. Type 2 commands are optional; a program block can contain a single Type 3 command. Type 2 commands and Conditional Jumps do not take effect until the DSM executes the next Type 3 command.

While the DSM314 is executing a program block, the following program block is processed into a buffer command area. This buffering feature minimizes block transition time. Thus, parameters used in a move must be loaded before the move command that was programmed two blocks earlier completes execution. In other words, in order to minimize the block-to-block transition time, a new block is pre-processed during previous block execution. Program block parameters must be loaded before the preceding block begins execution.

When a DSM314 is executing a multi-axis program, the program commands are scanned independently by each axis and only the data designated for that axis is executed. Note that some multi-axis program commands do not specify an axis (Block number, Jump, Call, and End) and therefore apply to both axes.

A multi-axis program can contain SYNC commands to synchronize the axes at designated points. When the first axis reaches a SYNC block (a block containing a SYNC command), it will not execute the next block until the other axis has also reached the SYNC Block. Refer to Example 18, “Multi-axis Programming”, later in this chapter, for an example of this.

7.5 Prerequisites for Programmed Motion

The following conditions must be satisfied before a motion program can be initiated (for a multi-axis program, the conditions must be met for both axes):

- The Enable Drive %Q bit must be ON
- The Drive Enabled %I bit must be ON
- The Position Valid %I bit must be ON
- The Moving %I bit must be OFF
- The Program Active %I bit must be OFF
- The Abort All Moves %Q bit must be OFF
- The axis position must be within the configured end of travel limits (**High Software EOT and Low Software EOT**), unless the **Software End of Travel** mode is configured as Disabled
- The Overtravel Limit Switch inputs must be ON (24V input is high) if enabled
- A Force Digital Servo Velocity %AQ command must not be active

- The program to be executed must be a valid program stored in the DSM314

7.6 Conditions That Stop a Motion Program

A motion program will immediately cease when one of the following conditions occurs:

- The Abort All Moves %Q bit turns ON
- The Enable Drive %Q bit turns OFF
- An Overtravel Limit Switch turns OFF when **OT Limit Switch** is ENABLED via configuration.
- The next programmed move, either PMOVE or CMOVE, will pass a **Software EOT Limit** (unless the **Software End of Travel** mode is configured as Disabled)
- A Stop Normal or Stop Fast Response Method Error occurs. See Appendix A, “Error Reporting.”

7.7 Motion Program Basics

Number of Programs, Subroutines, and Statements

The DSM314 supports 10 motion programs, 40 subroutines, and a maximum total of 1000 motion program statements.

Format

- Motion programs and subroutines are written using ASCII text.
- Only one motion language statement is permitted per line, and a motion language statement may not span more than one line. Normal comments may span multiple lines.
- White space and blank lines may be used to improve readability and to separate certain items.
- The Motion Editor is not case sensitive.
- All motion programs and subroutines must be contained in a single file.

Single-axis and multi-axis programs and subroutines

A given single-axis program must have the capability to be run on any one axis specified in the Program definition statement. Therefore, motion language commands in single-axis programs and subroutines will not specify an axis. Rather, the axis specified in the PROGRAM statement is used for all motion commands in the program. Multi-axis programs and subroutines can only call multi-axis subroutines. Likewise, single-axis programs and subroutines can only call single-axis subroutines.

Program and subroutine definition statements

The Motion Editor requires “Program” and “Subroutine” definition statements that specify program/subroutine number and axis configuration (PROGRAM 1 AXIS2 or SUBROUTINE 2 MULTI-AXIS). These statements are placed on the first line of the program or subroutine. Programs are terminated with an ENDPORG statement, subroutines are terminated with and ENDSUB statement. These statements serve as separators between programs and

subroutines, identify the program and subroutine numbers, and indicate the type of program (single-axis or multi-axis).

Block numbers and sync blocks

Block numbers will be suffixed with a colon (1: for example). Sync blocks are identified by a line with a block number followed by the SYNC command (2: SYNC for example). Block numbers may appear alone on a line or preceding a motion command on the same line.

7.7.1 Motion Language Syntax and Commands

White space

White space has no significance and is ignored, except where necessary to use as a separator. For example, in “CMOVE AXIS1 50000, ABS,S-CURVE” a space is required as a separator between CMOVE and AXIS1, but is not required in the phrase 50000,ABS because the comma separates the parameters. Blanks, blank lines, and tabs are considered white space.

Numeric Constants

Numeric constants are limited to 32-bit integer values, which may be signed or unsigned depending on the context in which they are used. All motion commands further limit this range. Numeric constants may be entered as decimal, hexadecimal, or binary values. Hexadecimal and binary constants are identified by the prefixes, 16# and 2#, respectively (do not use a space between the prefix and the number). Hexadecimal and binary constants cannot be prefixed with a negative sign. Therefore, negative values must be entered in two's complement form. Numeric constants may contain single underline characters (e.g. 5_000_000) between digits to improve the readability of large numbers or to represent implied decimal points in fixed point numbers.

Comments

The (* character pair introduce a normal comment, which terminates with the *) character pair. These comments may appear anywhere white space can, for example within or following a motion program statement, alone on a line, or spanning several lines. These comments do not nest. The // character pair introduces a single line comment. All text following the // to the end of the line is ignored by the Motion Editor. However, if using the //, do not force a break to the next line (by using a Return) or an error will result. If you wish to make long comments readable on the Motion Editor screen without the need for scrolling to the right, you can use the (* and *) symbols (required for multi-line comments) along with Returns (created by pressing the Enter key), which force the text to break to the next line.

Motion Program Key Words

The following words have special significance in the motion programming language.

ABS	AXIS3	ENDSUB	MULTI-AXIS	SUB
ABSOLUTE	AXIS4	ENDS	PMOVE	SYNC
ACCEL	CALL	INCR	PROGRAM	VELOC
ACC	CMOVE	INCREMENTAL	PROG	VEL
ACCELERATION	DWELL	JUMP	S-CURVE	VELOCITY
AXIS1	ENDP	LINEAR	SINGLE-AXIS	WAIT
AXIS2	ENDPROG	LOAD	SUBROUTINE	

Variables

Motion Programs support a limited set of predefined variables: the parameter data registers and the CTL bits. In the following table, x represents a decimal value in the specified range. The value x is interpreted based on its numeric value. Therefore, a given variable may be referenced several ways. For example, P1 and P001 both refer to Parameter Data Register 1 and will be accepted by the Motion Editor.

Variable	Constraints
Px	$0 \leq x \leq 255$
CTLx	$01 \leq x \leq 32$

Separators

Separators are used to separate elements or are added to elements to indicate that they serve a unique function.

Separator	Function
,	Separate command parameters
:	Identifies a constant as a block number

7.7.2 Motion Program Commands

This section describes the motion commands. Most motion commands have two forms, multi-axis and single-axis. The multi-axis form is used in multi-axis programs and subroutines and requires the axis to be specified as a parameter in certain commands (for example: VELOC AXIS1 5000). In single-axis programs the axis number is specified in the program header (for example: PROGRAM 2 AXIS1) and must not be specified within the program.

Some of the command keywords have aliases. The alias command keywords are functionally equivalent to the actual keywords. Alias usage is optional and largely a matter of personal preference.

Items that appear within angle brackets (“<”, “>”) represent classes of items, and are described in more detail. Items that appear in square brackets (“[”, “]”) are optional. Items that appear in curly brackets (“{”, “}”) are required for multi-axis programs and subroutines but are illegal when used in single-axis programs or subroutines.

The general format of motion language commands is KEYWORD {axis} <parameter [, parameter]>. If the axis is specified, it immediately follows the command keyword. Command parameter(s) follow the axis, if specified. If there are multiple parameters, they are separated by commas.

Note: *The DSM314 does not support the NULL command or Program Zero.*

ACCEL

The ACCEL statement sets the axis acceleration for subsequent moves and remains in effect in a given program unless changed. If an ACCEL statement is not used in a program, the moves will accelerate at the current Jog Acceleration value. Moves programmed before the first ACCEL statement will accelerate at the current Jog Acceleration. Moves programmed after an ACCEL statement will use the value in the ACCEL statement.

Note: *ACCEL commands for a given axis in a program or subroutine must be separated by a PMOVE statement, CMOVE statement, or an unconditional jump.*

Syntax:

ACCEL {<axis>} <acceleration>

Parameter	Description
<axis>	The axis number can only be specified in a multi-axis program or subroutine. The axis may be specified using the keywords or constants.
<acceleration>	The acceleration is specified by using either an unsigned constant in the range of 1 - 1,073,741,823 or by using a parameter data register.

Aliases:

ACC, ACCELERATION

Errors:

1. ACCEL commands must be separated by at least one move command.
2. Specified acceleration constant is not in the range of 1 - 1,073,741,823
3. Parameter data register is not in the range of 0 - 255.
4. Axis specified in single-axis program.
5. No axis specified in multi-axis program.
6. Specified axis does not support programmed motion.

Block Number

Block numbers may be used as the destination of JUMP commands. They may appear alone on a line, or preceding a command.

Syntax:

<block num>: [<command>]

Parameter	Description
<block num>	Block number must be in the range of 1 – 65535
< command>	Any command except PROGRAM, SUBROUTINE, ENDPROG, ENDSUB, or another block number may follow a block number on the same line

Aliases:

None

Errors:

1. All block numbers and synch block numbers must be unique within a program or subroutine.
2. Block number must be in the range of 1 – 65535.

CALL

The CALL command calls a subroutine from a program or subroutine.

Syntax:

CALL <subroutine destination>

Parameter	Description
<subroutine destination>	A subroutine destination specified as a constant, 1 – 40, or a parameter data register.

Aliases:

None

Errors:

1. Subroutine number must be in the range of 1 – 40, or parameter data register 0 – 255.

2. If caller is a subroutine, it cannot call itself (no recursive calls) or call another subroutine that directly or indirectly references it.
3. Call destination subroutine must be defined in the same file.
4. Single-axis programs and subroutines can only call single-axis subroutines. Multi-axis programs and subroutines can only call multi-axis subroutines.

CMOVE

The CMOVE command programs a continuous move using the specified position and acceleration mode.

Syntax:

CMOVE [<axis>] <position>, <positioning mode>, <acceleration mode>

Parameter	Description
<axis>	The axis can only be specified in a multi-axis program or subroutine. The axis may be specified using the AXISx keywords or constants.
<position>	The destination positions. May be a constant or a parameter data register.
<positioning mode>	Specifies incremental (INCR) or absolute (ABS) positioning.
<acceleration mode>	Specifies linear (LINEAR) or s-curve (S-CURVE) acceleration for the move.

Aliases:

None

Errors:

1. Axis specified in single-axis program.
2. No axis specified in multi-axis program.
3. Position must be in the range of -536,870,912 – 536,870,911, or parameter data register 0 - 255.
4. Positioning mode must be either INCR or ABS.
5. Acceleration mode must be either LINEAR or S-CURVE.
6. Specified axis does not support programmed motion.

DWELL

DWELL causes motion to cease for a specified time period before processing the next command. Specifying a dwell of zero (either as a constant or the value in a parameter data register) causes no dwell to occur (this is a change from APM and DSM302 functionality).

A single DWELL command only applies to one axis. Therefore, in a multi-axis program, you must designate an axis number for each DWELL command. For example: DWELL AXIS1 2000. If you wish to pause both axes in a multi-axis program, you must use a DWELL command for each axis.

Syntax:

DWELL {<axis>} <delay>

Parameter	Description
<axis>	The axis can only be specified in a multi-axis program or subroutine. The axis may be specified using the AXISx keywords or constants.
<delay>	Delay in milliseconds specified as a constant or a parameter data register. Range is 0-60,000 ms. A value of 0 is interpreted as a null command.

Aliases:

None

Errors:

1. Axis specified in single-axis program.
2. No axis specified in multi-axis program.
3. Delay must be in the range of 0 – 60,000 or parameter data register 0 - 255.
4. Specified axis does not support programmed motion.

ENDPROG

The ENDPROG statement terminates a motion program definition.

Syntax:

ENDPROG

Aliases:

ENDP

ENDSUB

The ENDSUB statement terminates a motion subroutine definition.

Syntax:

ENDSUB

Aliases:

ENDS

JUMP

Jump to a block number or sync block within the current program or subroutine. The jump may be conditional, based on the state of a CTL bit, or unconditional.

Syntax:

JUMP <condition>, <destination>

Parameter	Description
<condition>	Jump condition, must specify CTL01 – CTL32 or UNCOND
<destination>	Destination block or synch block number

Aliases:

None

Errors:

1. Jump condition must be CTL in the range of 1 – 32, or keyword UNCOND.
2. Destination block must be in the range of 1 - 65535 and must be defined within the same program or subroutine as the JUMP statement.

LOAD

Initializes or changes a parameter data register with 32-bit twos-complement integer value.

Syntax:

LOAD <parameter data register>, <load value>

Parameter	Description
<parameter data register>	Parameter data register to be initialized. Restricted to registers P000 – P255.
<load value>	32-bit numeric constant.

Aliases:

None

Errors:

1. Parameter data register must be in the range of P000 – P255.
2. Load value must be in the range of a 32-bit twos-complement value.

PMOVE

The PMOVE command programs a positioning move using the specified position and acceleration mode.

Syntax:

PMOVE {<axis>} <position>, <positioning mode>, <acceleration mode>

Parameter	Description
<axis>	The axis can only be specified in a multi-axis program or subroutine. The axis may be specified using the AXISx keywords or constants.
<position>	The destination positions. May be a constant or a parameter data register.
<positioning mode>	Specifies incremental (INCR) or absolute (ABS) positioning.
<acceleration mode>	Specifies linear (LINEAR) or s-curve (S-CURVE) acceleration for the move.

Aliases:

None

Errors:

1. Axis specified in single-axis program.
2. No axis specified in multi-axis program.
3. Position must be in the range of -536,870,912 – 536,870,911, or parameter data register 0 - 255.
4. Positioning mode must be either INCR or ABS.
5. Acceleration mode must be either LINEAR or S-CURVE.
6. Specified axis does not support programmed motion.

PROGRAM

The PROGRAM statement is the first statement in a motion program. The program statement identifies the program number (1-10) and the axis configuration. Program definitions cannot nest.

There are two types of motion programs, single axis in which all commands are directed to the same axis, and multi-axis, which may contain commands for axis 1 and axis 2. The program type is specified by the PROGRAM statement. A single-axis program is identified by specifying the target axis following the program number (for example, PROGRAM 3

AXIS1). A multi-axis program is identified by the word MULTI-AXIS following the program number (for example, PROGRAM 4 MULTI-AXIS).

The program axis configuration is used to enforce whether or not the axis parameter must be supplied in the program's motion commands. It also restricts multi-axis programs to calling multi-axis subroutines, and single-axis programs to calling single-axis subroutines. The axis specified in a single-axis program is used by any subroutine it calls; therefore, an axis number should not be specified anywhere within a single-axis subroutine.

Syntax:

PROGRAM <program number> <axis configuration>

Parameter	Description
<program number>	The program number must be a decimal value in the range of 1 – 10. Within a source file, each PROGRAM defined must have a unique number.
<axis configuration>	The axis configuration must have a value of MULTI-AXIS for multi-axis programs, or axis designation (for example, AXIS1) for single-axis programs. Axes may be specified using the AXISx keywords or constants, where x = 1-4.

Aliases:

PROG

SUBROUTINE

The SUBROUTINE statement is the first statement in a motion subroutine. The subroutine statement identifies the subroutine number (1-40) and the axis configuration. Subroutine definitions cannot nest.

There are two types of motion subroutines, single axis in which all commands are directed to the same axis, and multi-axis, which may contain commands for axis 1 and axis 2. The subroutine type is specified by the SUBROUTINE statement. A single-axis subroutine is identified by the word SINGLE-AXIS following the subroutine number. A multi-axis subroutine is identified by the word MULTI-AXIS following the subroutine number.

The subroutine axis configuration is used to enforce whether or not the axis parameter must be supplied in the subroutine's motion commands. It also restricts multi-axis subroutines to calling multi-axis subroutines, and single-axis subroutines to calling single-axis subroutines. A single-axis subroutine uses the axis number specified in the calling program.

Syntax:

SUBROUTINE <subroutine number> <axis configuration>

Parameter	Description
<subroutine number>	The subroutine number must be a decimal value in the range of 1 – 40. Within a source file, each subroutine defined must have a unique number.
<axis configuration>	The axis configuration must have a value of MULTI-AXIS or SINGLE-AXIS.

Aliases:

SUB

Sync Block

A sync block is a special case of a block number. A sync block may only be used in a multi-axis program.

A sync block is identified by a block number followed by the command SYNC. The SYNC command must appear on the same line as the block number.

Syntax:

<block num>: SYNC

Parameter	Description
<block num>	Block number must be in the range of 1 – 65535

Aliases:

none

Errors:

1. Sync blocks can only appear in multi-axis programs.
2. All block numbers and synch block numbers must be unique within a program or subroutine.
3. Sync blocks and block numbers cannot appear in consecutive statements without an intervening command.
4. Sync block numbers must be in the range of 1 – 65535.

VELOC

Sets the axis velocity used by subsequent motion program move commands and remains in effect until changed by another VELOC statement. If a VELOC statement is not used in a program, moves will use the current Jog Velocity value. Also, moves programmed before the first VELOC statement will use the current Jog Velocity.

Note: VELOC commands for a given axis in a program or subroutine must be separated by a PMOVE statement, CMOVE statement, or an unconditional jump.

Syntax:

VELOC [<axis>] <velocity>

Parameter	Description
<axis>	The axis can only be specified in a multi-axis program or subroutine. The axis may be specified using the AXISx keywords or constants.
<velocity>	The desired velocity. May be a constant or a parameter data register.

Aliases:

VEL, VELOCITY

Errors:

1. Axis specified in single-axis program.
2. No axis specified in multi-axis program.
3. Velocity must be a constant in the range of 1 – 8388607.
4. VELOC commands must be separated by at least one move command.
5. Specified axis does not support programmed motion.

WAIT

Permits synchronization with some external event through the CTL bits. Execution of the next command is suspended until the specified CTL is set.

A single WAIT command only applies to one axis. Therefore, in a multi-axis program, you must designate the axis number that a WAIT applies to. For example: WAIT AXIS1 CTL01. If you wish to make both axes wait in a multi-axis program, you must use a separate WAIT command for each axis.

Syntax:

WAIT [<axis>] <ctl>

Parameter	Description
<axis>	The axis can only be specified in a multi-axis program or subroutine. The axis may be specified using the AXISx keywords or constants.
<ctl>	Specifies CTL01 – CTL32.

Aliases:

none

Errors:

1. Axis specified in single-axis program.
2. No axis specified in multi-axis program.
3. CTL must be in the range of 1 – 32.
4. Specified axis does not support programmed motion.

7.7.3 Program and Subroutine Structure

Single-axis Program Structure

- **PROGRAM definition statement.** It must be the first line of the program. It must identify the program number and axis number. The program number has a space between the PROGRAM keyword and the number. In contrast, the axis number must not have a space within it. For example:

```
PROGRAM 1 AXIS3
```

- **Body.** The program body contains the actual program commands. Note that in a single-axis program, you must not specify an axis number in any of the commands. Doing so will generate an error. An example of correct syntax for a single-axis program is:

```
ACCEL 50000
```

- **End of Program.** Uses the ENDPROG statement. This statement clearly identifies the end of the program and helps separate one program or subroutine from another. The ENDPROG should be the only thing on the last line of any program:

```
ENDPROG
```

Single-Axis Program Example

Note that the axis number is specified in the first line and is not specified in the program body. Note also, that there is no space in the term AXIS1.

```
PROGRAM 2 AXIS1
```

```
ACCEL 50000
```

```
VELOC 5000
```

```
PMOVE 10000, ABS, LINEAR
```

```
DWELL 6000
```

```
PMOVE 5000, ABS, LINEAR
```

```
ENDPROG
```

Multi-Axis Program Structure

- **PROGRAM definition statement.** Must be the first line of the program. It must identify the program number and the fact that this is a multi-axis program by using the MULTI-AXIS term. For example:

```
PROGRAM 3 MULTI-AXIS
```

- **Body.** Contains the actual program commands. Note that in a multi-axis program, you must specify an axis number in many of the commands. Failure to do so will generate an error. The axis number term, such as AXIS1, must not have a space within it. An example of correct syntax for a multi-axis program command is:

```
ACCEL AXIS1 50000
```

- **End of Program.** Uses the ENDPROG statement. This statement clearly identifies the end of the program and helps separate one program or subroutine from another. The ENDPROG should be the only thing on the last line of any program:

ENDPROG

Multi-Axis Program Example

Note that the term MULTI-AXIS must be used in the PROGRAM statement on the first line, and that axis numbers are specified in the applicable commands in the program body.

PROGRAM 1 MULTI-AXIS

ACCEL AXIS1 500000

VELOC AXIS1 5000

1: CMOVE AXIS2 -100000, ABS, LINEAR DWELL AXIS2 6000

JUMP CTL31, 1

CALL P255

LOAD P215, 2000

PMOVE AXIS1 8388607, INCR, S-CURVE

ENDPROG

Single-axis Subroutine Structure

- **SUBROUTINE definition statement.** It must be the first line of the subroutine. It must identify the subroutine number and contain the SINGLE-AXIS statement. For example:

SUBROUTINE 3 SINGLE-AXIS

- **Body.** The subroutine body contains the actual programmed commands. Note that in a single-axis subroutine, you must not specify an axis number in any of the commands. Doing so will generate an error. An example of correct syntax for a single-axis subroutine command is:

ACCEL 50000

- **End of Subroutine.** Uses the ENDSUB statement. This statement clearly identifies the end of the subroutine and helps separate one subroutine or program from another. The ENDSUB should be the only thing on the last line of any subroutine:

ENDSUB

Single-Axis Subroutine Example

An axis number should not be specified in a single-axis subroutine. That is because a single-axis subroutine will apply to the axis specified in the single-axis program that calls it. This allows a subroutine to be used by different single-axis programs, regardless of the particular axis number they specify.

```
SUBROUTINE 15 SINGLE-AXIS
    ACCEL 50000
    VELOC 10000
    PMOVE 200000, ABS, LINEAR
    DWELL 3000
    PMOVE 50000, ABS, LINEAR
ENDSUB
```

Multi-Axis Subroutine Structure

- **SUBROUTINE definition statement.** It must be the first line of the subroutine. It must identify the subroutine number and the fact that this is a multi-axis program by using the MULTI-AXIS term. For example:

```
SUBROUTINE 7 MULTI-AXIS
```

- **Body.** The subroutine body contains the actual programmed commands. Note that in a multi-axis subroutine, you must specify an axis number in many of the commands. Failure to do so will generate an error. An example of correct syntax for a multi-axis subroutine command is:

```
ACCEL AXIS2 50000
```

- **End of Subroutine.** Uses the ENDSUB statement. This statement clearly identifies the end of the subroutine and helps separate one subroutine or program from another. The ENDSUB should be the only thing on the last line of any subroutine:

```
ENDSUB
```

Multi-Axis Subroutine Example

```
SUBROUTINE 2 MULTI-AXIS
    ACCEL AXIS2 P100
    VELOC AXIS2 P105
2:   SYNC
    CMOVE AXIS2 P001, INCR, S-CURVE
    DWELL AXIS2 P001
    JUMP CTL01, 2
    PMOVE AXIS2 P214, ABS, LINEAR
ENDSUB
```

7.7.4 Command Usage Examples

The following examples are not complete programs. For example, in many cases the PROGRAM and ENDPROG statements are not shown. These statements (in correct context) would need to be added to make the program compile successfully. Programmed moves have three parameters:

1. The distance (data) to move or position to move to,
2. The type of positioning reference (command modifier) to use for the move, and
3. The type of acceleration (command modifier) to use while performing the move.

Note: *Motion programs can contain statements that use constants as data associated with commands or variables that are also referred to as parameters (P0-P255).*

Absolute or Incremental Positioning

Absolute Positioning

In an absolute positioning move, the first parameter is the position to move to. The following is an absolute positioning move example.

```
PMOVE 5000, ABS, LINEAR
```

In this example, the axis will move from its current position, whatever it may be, to the position 5000. Thus, the actual distance moved depends upon the axis' current position when the move is encountered. If the initial position is 0, the axis will move 5000 user units in the positive direction. If the initial position is 8000, the axis will move 3000 user units in the negative direction. If the initial position is 5000, the axis will not move.

Incremental Positioning

In an incremental move, the first parameter specifies the distance to move from the current position. The DSM314 translates incremental move distances into absolute move positions. This eliminates error accumulation. The following is an incremental positioning move example.

```
PMOVE 5000, INCR, LINEAR
```

In this example, the axis will move from its current position to a position 5000 user units greater. With an incremental move, the first parameter specifies the actual number of user units the axis moves.

Types of Acceleration

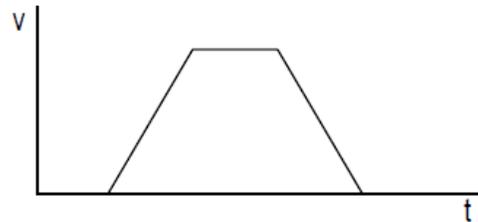
Linear Acceleration

A sample linear move profile that plots velocity versus time is shown in Figure 69. As illustrated, a linear move uses constant (linear) acceleration. The area under the graph represents the distance moved.

Figure 69: Sample Linear Motion

```
ACCEL 1000  
VELOC 2000  
PMOVE  
LINEAR
```

6000, INCR,

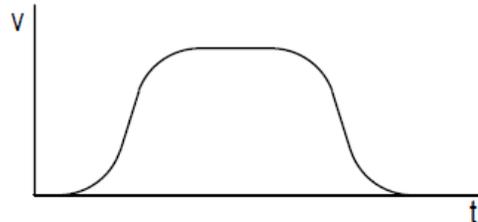


S-Curve Acceleration

An S-Curve motion sample, plotting velocity versus time, is shown below. As illustrated, S-Curve acceleration is non-linear. When the move begins, the acceleration starts slowly and builds until it reaches the programmed acceleration. This should be the midpoint of the acceleration. Then, the acceleration begins decreasing until it is zero, at which time the programmed velocity has been reached. An S-Curve move requires twice the time and distance to accelerate and decelerate that a comparable linear move need. The area under the graph represents the distance moved.

Figure 70: Sample S-Curve Motion

```
ACCEL          2000  
VELOC          2000  
PMOVE          8000, INCR, S-  
CURVE
```



7.7.5 Types of Programmed Move Commands

The following examples are not complete programs. For example, in many cases the PROGRAM and ENDPORG statements are not shown. These statements (in correct context) would need to be added to make the program compile successfully.

Positioning Move (PMOVE)

A PMOVE must always come to a complete stop. The stop must long enough to allow the In Zone %I bit to turn ON before the next move can begin.

A PMOVE uses the most recently programmed velocity and acceleration. If a VELOC command has not been encountered in the motion program, the **Jog Velocity** is used as default. If an ACCEL command has not been encountered in the motion program, the **Jog Acceleration** is used as default.

Continuous Move (CMOVE)

A CMOVE does not stop when completed unless it is followed by a DWELL or a WAIT, the next programmed velocity is zero, or it is the last program command. It does not wait for In Zone %I bit to turn ON before going to the next move. A normal CMOVE is a command that reaches its programmed position at the same time that it reaches the velocity of the following Move command.

A CMOVE uses the most recently programmed velocity and acceleration. If a VELOC command has not been encountered in the motion program, the **Jog Velocity** is used as default. If an ACCEL command has not been encountered in the motion program, the **Jog Acceleration** is used as default.

A special form of the CMOVE command can be used to force the DSM314 to reach the programmed CMOVE position before starting the velocity change associated with the next move command (that is, execute the entire CMOVE command at a constant velocity). **Programming an incremental CMOVE command with an operand of 0 (for example: CMOVE 0, INCR, LINEAR) will delay the servo velocity change until the next move command in sequence.** The following sequence of commands illustrates this effect (assume ACCELS are chosen to allow motions to complete normally):

Command	Data	Comments
VELOC	10000	//Set velocity of first move = 10000
CMOVE	15000, ABS, LINEAR	//Reach velocity of second move (20000) at position = 15000
VELOC	20000	//Set velocity of second move = 20000
CMOVE	30000, ABS, LINEAR	(*Stay at velocity = 20000 until position = 30000, then change to velocity = 5000*)
CMOVE	0, INCR, LINEAR	(*Flag to signal the DSM314 to wait for next move before changing to the next velocity*)
VELOC	5000	//Set velocity of third move = 5000
PMOVE	40000, ABS, LINEAR	//Final stop position = 40000

Note: White space characters (blank spaces, tabs, etc.) were used in the program above to improve readability.

Figure 71: Example 1, Before Inserting CMOVE (0)

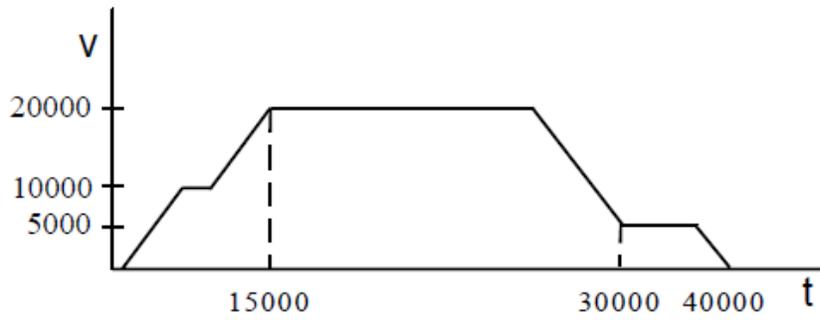
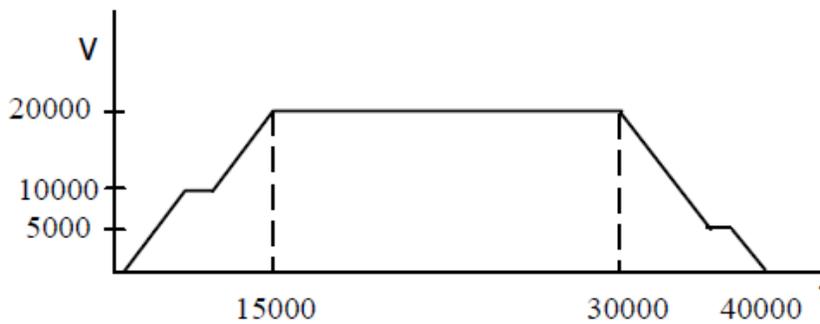


Figure 72: Example 2, After Inserting CMOVE (0)



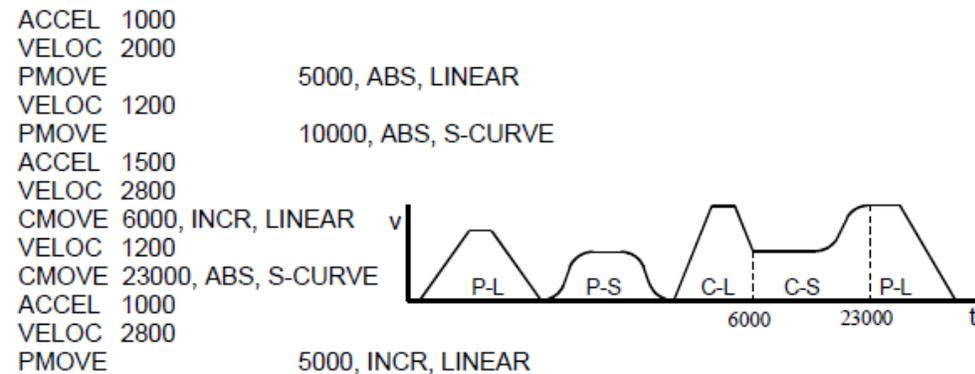
Programmed Moves

By combining CMOVEs and PMOVEs, absolute and incremental moves, and linear and s-curve motion, virtually any motion profile can be generated. The following examples show some simple motion profiles, as well as some common motion programming errors.

Example 1: Combining PMOVEs and CMOVEs

This example shows how simple PMOVEs and CMOVEs combine to form motion profiles.

Figure 73: Combining PMOVEs and CMOVEs



The first PMOVE accelerates to program velocity, moves for a distance, and decelerates to a stop. This is because motion stops after all PMOVEs. When the first move stops, it is at the programmed distance.

The second move is an s-curve PMOVE. It, like the first, accelerates to the programmed velocity, moves for a time, and decelerates to zero velocity because it is a PMOVE.

The next move is a linear CMOVE. It accelerates to program velocity, moves for a time, and then decelerates to a lower velocity using linear acceleration. When a CMOVE ends, it will be at the programmed position of the move just completed, and at the velocity of the next move. Thus when the fourth move begins, it is already at its programmed velocity.

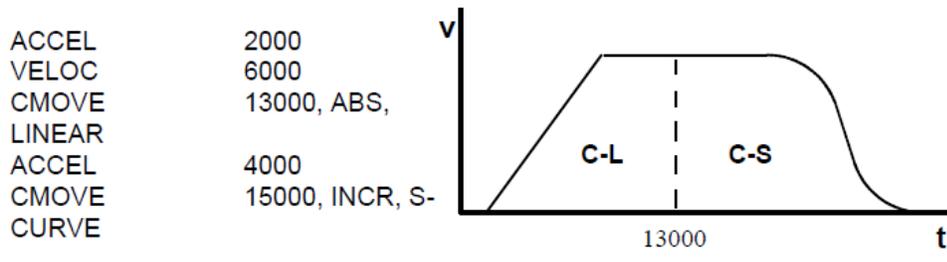
The fourth move is a CMOVE, so as it approaches its final position, it accelerates to be at the velocity of the fifth move when it completes. The graph shows the acceleration of the fourth move is s-curve.

Finally, the fifth move begins and moves at its programmed velocity for a time until it decelerates to zero. Any subsequent moves after the fifth would begin at zero velocity because the fifth move is a PMOVE.

Example 2: Changing the Acceleration Mode During a Profile

The following example shows how a different acceleration, and an even acceleration mode, can be used during a profile using CMOVEs. The first CMOVE accelerates linearly to the programmed velocity. Because the second CMOVE's velocity is identical to the first, the first CMOVE finishes its move without changing velocity. The acceleration of the second move is S-curve as it decelerates to zero velocity.

Figure 74: Changing the Acceleration Mode During a Profile



Example 3: Not Enough Distance to Reach Programmed Velocity

CMOVES and PMOVES can be programmed that do not have enough distance to reach the programmed velocity. The following graph shows a CMOVE that could not reach the programmed velocity. The DSM314 accelerates to the point where it must start decelerating to reach the programmed position of C1 at the velocity of the second CMOVE.

Figure 75: Not Enough Distance to Reach Programmed Velocity

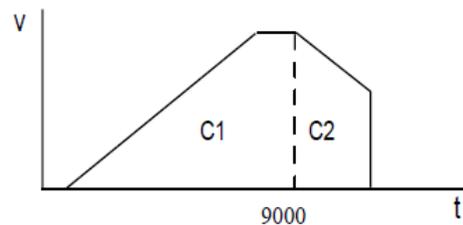


Example 4: Hanging the Move When the Distance Runs Out

A serious programming error involves “hanging” (i.e. leaving no desirable options for the command generator) the move at a high velocity when the distance runs out. In the following example, the first CMOVE accelerates to a high velocity. The second CMOVE has an identical velocity. However, the distance specified for the second CMOVE is very short. Thus, the axis is running at a very high velocity and must stop in a short distance. If the programmed acceleration is not large enough, the following profile could occur. The DSM314 attempts to avoid overshooting the final position by commanding a zero velocity. This rapid velocity change is undesirable and can cause machine damage.

Figure 76: Hanging the DSM314 When the Distance Runs Out

```
ACCEL 500
VELOC 3000
CMOVE 9000, ABS, LINEAR
ACCEL 600
CMOVE 4800, INCR, LINEAR
```



DWELL Command

A DWELL command is used to generate no motion for a specified number of milliseconds. The DWELL command may use a value stored in a designated parameter.

A DWELL after a CMOVE will make the CMOVE stop before the next move, unless the specified dwell duration is zero milliseconds. A DWELL is treated as a “null” command and skipped (CMOVE continues to the next Move following the DWELL) if the DWELL command has a value of zero, or references a parameter register that has a value of zero.

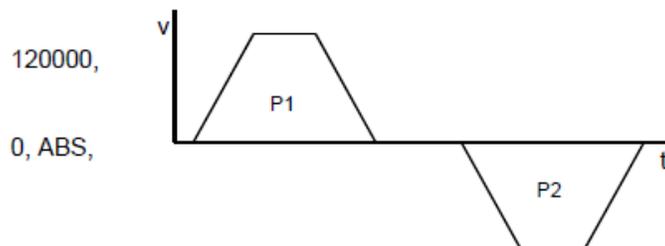
A single DWELL command only applies to one axis. Therefore, in a multi-axis program, you must designate an axis number with each DWELL command. For example: DWELL AXIS1 2000. If you wish to pause both axes in a multi-axis program, you must use a DWELL command for each axis.

Example 5: DWELL

A simple motion profile, which moves to a specific point, waits, and returns to the original point is shown below.

Figure 77: Dwell Command Example

```
ACCEL 30000
VELOC 15000
PMOVE
120000,
ABS, LINEAR
DWELL 4000
PMOVE
0, ABS,
LINEAR
```



Wait Command

The WAIT command is similar to the DWELL command. Instead of generating no motion for a specified period of time, a WAIT stops program motion until a specified CTL bit turns ON. Thus motion stops any time a WAIT is encountered, even if the CTL bit is ON before the WAIT is reached in the program. The trigger to continue the program can be any of the twelve CTL bits.

If, in the previous example, WAIT were substituted for DWELL, the motion profile would be the same except the second PMOVE would not start until the CTL bit turned ON. If the CTL bit was ON when the program reached the WAIT, the second PMOVE would begin immediately after the first PMOVE finished.

Also, if WAIT were used instead of DWELL in the previous example, CMOVEs and PMOVEs would generate similar velocity profiles. The WAIT will stop motion whether the previous move is a CMOVE or PMOVE.

A single WAIT command only applies to one axis. Therefore, in a multi-axis program, you must designate an axis number with each WAIT command. For example: WAIT AXIS1 CTL001. If you wish to have both axes wait in a multi-axis program, you must use a separate WAIT command for each axis.

Subroutines

The DSM314 can store up to ten separate programs and forty subroutines. Subroutines can be defined as two types: single-axis and multi-axis. Subroutines are available for all motion programs created with the Motion Editor. Commands within single-axis subroutines do not contain an axis number; this allows single-axis subroutines to be called from any single-axis program (the commands in the subroutine use the axis number specified by the calling program). Commands within multi-axis subroutines contain axis numbers just like commands within multi-axis programs. Multi-axis subroutines can only be called from multi-axis programs or subroutines. Single-axis subroutines can only be called from single-axis programs or subroutines. For example, a single-axis program for axis 1 and a single-axis program for axis 2 can call the same single-axis subroutine simultaneously. Each subroutine must be assigned a unique number between 1 and 40.

Subroutines are programmed using the CALL command, which specifies the subroutine number to be called. When a CALL is encountered during program execution, program execution is redirected to the subroutine. When the subroutine completes, program execution resumes at the command after the CALL command. Subroutines can be called from another subroutine, but once a subroutine has been called, it must complete before it can be called again for the same axis. Thus, recursion is not allowed.

Block Numbers and Jumps

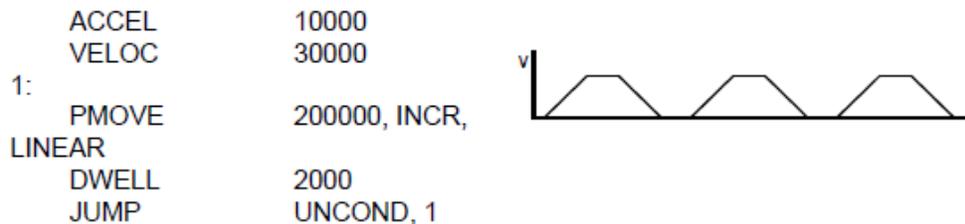
Block numbers are used as reference points within a motion program and to control jump testing. A %AI data word displays the current block number which can be monitored to ensure correct program execution or to determine when events should occur. A block number can also serve as a JUMP command destination. Jumps may be unconditional or conditional. An unconditional jump command simply tells the DSM314 to continue program execution at the destination block number. A conditional jump only executes if the specified condition occurs. Examples of both types of jumps follow.

Unconditional Jumps

Example 6: Unconditional Jump

In the example below, the program executes a PMOVE, dwells for 2 seconds, then unconditionally jumps back to the beginning of the program at block 1. Thus, the PMOVE repeats until an end of travel limit (High Software EOT or Low Software EOT) or Overtravel Limit Switch is reached. An Abort All Moves %Q bit command could also be used to halt the program.

Figure 78: Unconditional Jump



Conditional Jumps

A conditional jump is a JUMP command with a CTL bit specified in the command. Conditional jumps are Type 1 commands in that they affect program path execution, but they are also similar to Type 2 commands because they do not take effect until a Type 3 command following the JUMP command is executed. When a conditional JUMP command is executed, the DSM314 examines the specified CTL bit. If the bit is ON, program execution jumps to the destination block number; if the bit is OFF, the program continues executing the command after the JUMP. Note that the Type 3 command after the conditional jump and at the jump destination will affect jump behavior.

Conditional Jump commands should not be used with multi-axis programs containing SYNC blocks unless the JUMP is triggered while both axes are testing the same JUMP command. Failure to follow this recommendation can result in unpredictable operation.

Conditional Jump testing starts when the next PMOVE, CMOVE, DWELL, or WAIT command following a Conditional JUMP becomes active.

When Conditional Jump testing is active, the designated CTL bit is tested at the position loop update rate (0.5, 1.0 or 2.0 milliseconds depending on configuration).

Conditional Jump testing ends when the designated CTL bit turns ON (Jump Trigger occurs) or when a new Block Number becomes active.

If more than one Conditional Jump is programmed without an intervening PMOVE, CMOVE, DWELL, or WAIT command, only the last Conditional Jump will be recognized.

A Conditional Jump cannot be used as the last line of a Subroutine (or on the line before an Unconditional Jump to the end of a subroutine) because jump testing terminates when the End Subroutine command is processed.

In summary, a Conditional Jump transfers control to a new program block on the basis of one of the external CTL input bits turning ON. Tests for CTL bit status can be carried out once or continuously during the following Type 3 command if it is in the same program block. Multiple Conditional Jumps are not supported within the same program block (the following example illustrates this incorrect usage of the Conditional Jump command).

Conditional Jump Example 1:

```
PROGRAM 1 MULTI-AXIS
    VELOC  AXIS1 10000
    ACCEL  AXIS1 10000
1:    JUMP  CTL01, 2          //This JUMP command will be ignored
    JUMP  CTL02, 3          //This JUMP command will be recognized
    CMOVE AXIS1 +40000, INCR, LINEAR
2:    CMOVE AXIS2 +20000, INCR, LINEAR
3:    PMOVE AXIS2 +100000, ABS, LINEAR
4:    DWELL  AXIS2 100
ENDPROG
```

The first JUMP is not programmed correctly because (1) it is not followed by an intervening Type 3 command, and (2) it is in the same block as another JUMP command. When a new Block Number becomes active AFTER a Conditional JUMP command, Jump testing will occur one final time.

Conditional Jump Example 2:

```
PROGRAM 2 AXIS1
    VELOC  10000
    ACCEL  10000
1:    CMOVE 20000, ABS, LINEAR
    JUMP  CTL01, 3
2:    PMOVE 40000, ABS, LINEAR //CTL01 tested only once
3:    DWELL  100
```

ENDPROG

In the example above, The CTL01 bit test occurs just once because the PMOVE following the JUMP contains a new Block Number (2). However, changing the location of Block Number 2 causes CTL bit testing throughout the PMOVE following the JUMP, as seen in the following example:

Conditional Jump Example 3:

PROGRAM 3 AXIS1

```
          VELOC  10000
          ACCEL  10000
1:        CMOVE  20000, ABS, LINEAR
2:        JUMP   CTL01, 3
          PMOVE  40000, ABS, LINEAR //CTL01 tested throughout PMOVE
3:        DWELL  100
ENDPROG
```

In this example, the CTL01 bit is tested throughout the PMOVE because the PMOVE and JUMP commands are in the same Block.

The DSM314 can perform a Conditional JUMP from an active CMOVE to a program block containing a CMOVE or PMOVE without stopping. **For the axis to jump without stopping, the distance represented by the CMOVE or PMOVE in the jump block must be greater than the servo stopping distance.** The servo stopping distance is computed using the present commanded velocity and the acceleration parameters that would be in effect when the jump block became active.

The axis will STOP before jumping if a Conditional Jump trigger occurs under any of the following conditions:

- When a PMOVE is active
- When a CMOVE is active and the Jump destination block contains a CMOVE or PMOVE representing motion in the opposite direction.
- When a CMOVE is active and the Jump destination block contains a CMOVE or PMOVE representing motion in the same direction with insufficient distance for the axis to stop.
- When a CMOVE is active and the Jump destination block contains a DWELL, WAIT or END (program) command.

If the axis does STOP before a Conditional Jump, the current programmed acceleration and acceleration mode will be used.

Unconditional Jumps do not force the axis to stop before jumping to a new program block. For example, a CMOVE followed by a JUMP Unconditional to another CMOVE will behave just as if the two CMOVEs occurred without an intervening Unconditional JUMP.

If Conditional Jump testing is active when the DSM314 command processor encounters a CALL SUBROUTINE command, the axis will stop and terminate jump testing before the CALL is executed.

If Conditional Jump testing is active when the DSM314 command processor encounters an END SUBROUTINE command, the axis will stop and terminate jump testing before the END SUBROUTINE is executed.

Jump Testing

Conditional jumps perform jump testing. If the CTL bit is ON, the jump is immediately performed. If the CTL bit is OFF, the DSM314 watches the CTL bit and keeps track of the JUMP destination. This monitoring of the CTL bit is called jump testing. If during jump testing the CTL bit turns ON before a BLOCK command, another JUMP command, or a CALL command is encountered, the jump is performed. These commands will end jump testing.

Example 7: Jump Testing

Consider the following two single-axis program section examples. In Example 1, the move to position 2000 is completed before jump testing begins. The block number occurring immediately after the JUMP command ends jumps testing. Thus, the duration for which the CTL bit is monitored is very short. However, in Example 2, the JUMP command is encountered before the CMOVE command. This starts the jump testing before motion begins, and jump testing continues as long as the move lasts. If the CTL bit turns ON while the move is being performed, the jump will be performed. After the move completes, the next block number is encountered, which ends jump testing, and program execution continues normally. If additional moves were programmed ahead of the next block number, jump testing would continue during those moves until the next block number was encountered.

Example 1

```
ACCEL 5000  
VELOC 1000  
1:  
    CMOVE 2000, ABS, LINEAR  
    JUMP CTL01, 3  
2:
```

Example 2

```
ACCEL 5000  
VELOC 1000  
1:  
    JUMP CTL01, 3  
    CMOVE 2000, ABS, LINEAR  
2:
```

Normal Stop Before JUMP

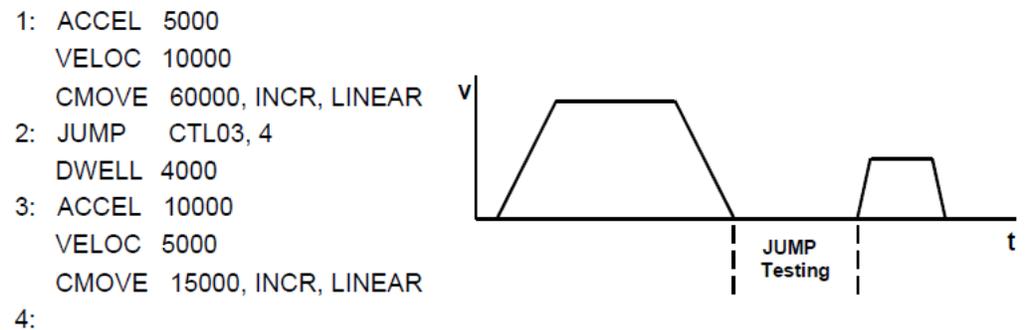
A conditional jump command is similar to Type 2 commands in that jump testing does not start until the Type 3 command immediately after the JUMP is executed. If this Type 3 command would normally stop motion, then motion will stop before jump testing begins. Type 3 commands that will stop motion are: DWELL, WAIT, ENDPROG, and moves in the opposite direction.

Thus, even though the CTL bit may be ON before the block with the conditional JUMP and Type 3 command is executed, axis motion will stop before program execution continues at the jump destination. This stopping is NOT a Jump Stop, which is described in Example 10.

Example 8: Normal Stop Before JUMP

The following example contains a jump followed by a DWELL command. The DSM314, because it processes ahead, knows it must stop after the CMOVE command. Thus, it comes to a stop before the DWELL is executed. Since jump testing does not begin until the DWELL is executed, testing begins after motion stops. Jump testing ends when the following CMOVE begins due to the associated BLOCK command. The dashed lines in the velocity profile indicate when jump testing takes place. In this example, the CTL03 bit does not turn ON during the program execution.

Figure 79: Normal Stop Before JUMP



Jumping Without Stopping

If the Type 3 command following a conditional jump is a CMOVE and the Type 3 command at the destination is a move command with sufficient distance to fully decelerate to zero when completed, the jump will be executed without stopping. This is the only way to sustain motion when a jump is performed.

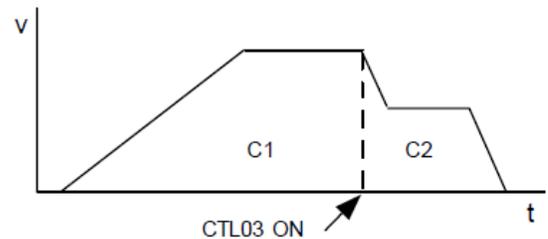
Example 9: JUMP Without Stopping

This is a simple example of a conditional jump from one CMOVE to another. While jump testing the CTL03 bit, the first CMOVE accelerates to the programmed velocity. Before the dashed line, the CTL03 bit is OFF, but at the dashed line the CTL03 bit turns ON. Program execution is immediately transferred to block 3 and the CMOVE there begins. Because the velocity at the jump destination is different, the velocity changes at the acceleration programmed of the jump destination block. Finally, as the second CMOVE completes, velocity is reduced to zero and the program ends.

Figure 80: JUMP Without Stopping

```

1: ACCEL 2000
   VELOC 10000
   JUMP CTL03, 3
   CMOVE 120000, INCR, LINEAR
3: ACCEL 20000
   VELOC 5000
   CMOVE 15000, INCR, LINEAR
  
```



Jump Stop

A jump stop is a stop that is caused by a jump. **When a jump stop occurs, the current programmed acceleration and acceleration mode are used.** Note that s-curve motion will achieve constant velocity before beginning to decelerate. See the s-curve jump examples for more details. There are two ways of generating a jump stop each described below.

A conditional JUMP triggered during a PMOVE will always generate a jump stop. Because a PMOVE always stops before continuing to a subsequent motion, a jump stop always occurs when a jump takes place during a PMOVE.

When a conditional jump trigger occurs during a CMOVE, however, a jump stop will not occur if the motion programmed at the jump destination is a PMOVE or CMOVE representing sufficient distance in the same direction. A jump stop will occur if the PMOVE or CMOVE at the jump destination does not represent sufficient distance or represents motion in the opposite direction.

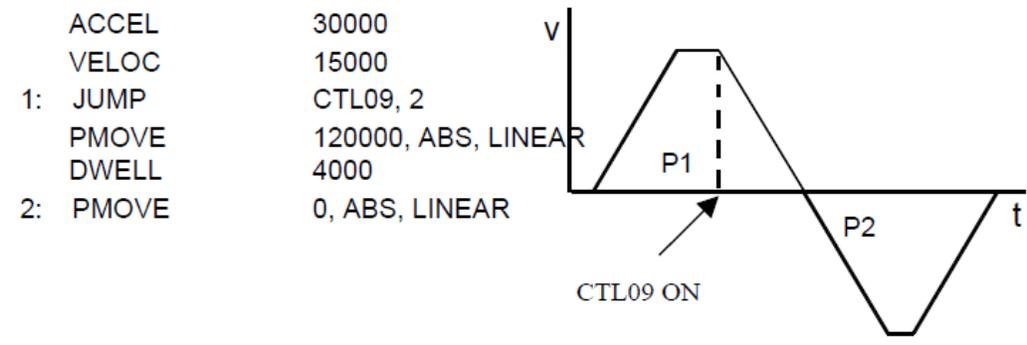
In an s-curve move, a jump stop will do one of two things. If the jump takes place after the midpoint of the acceleration or deceleration, the acceleration or deceleration is completed before the jump stop is initiated. If the jump occurs before the midpoint of the acceleration or deceleration, the profile will immediately begin leveling off. Once acceleration or deceleration is zero, the jump stop begins. See the s-curve jump examples.

Example 10: Jump Stop

The following is an example conditional jump with a jump stop. An enhancement on Example 5, DWELL, would be to watch an external CTL bit that would indicate a problem with the positive motion. If the CTL bit never turns on, the profile for the following program will be identical to the profile shown in the DWELL example. If the CTL bit turned on during the first PMOVE or the DWELL, the reverse movement would immediately commence.

The following profile would appear if the CTL bit turned on during the first PMOVE, at the dashed line. Because the first move completed early due to the CTL bit turning on, the second move would not have to move as far to get back to 0 position as it did in the DWELL example. Note that because the motion programmed at the jump destination is in the opposite direction as the initial motion, the profile would be identical if the moves were CMOVEs instead of PMOVEs.

Figure 81: Jump Stop

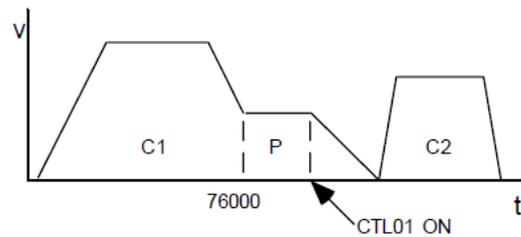


Example 11: Jump Followed by PMOVE

In this JUMP example, the command after the JUMP is a PMOVE in the same direction. The velocity profile below shows the acceleration and movement for the first CMOVE and the deceleration to the PMOVE's velocity. The CTL01 bit, OFF when the PMOVE begins, turns ON at the second dashed line. Motion stops after a PMOVE, even if a conditional jump goes to another block. Thus the CTL01 bit triggers a deceleration to zero before the final CMOVE begins.

Figure 82: Jump Followed by PMOVE

1:	ACCEL	2000
	VELOC	8000
	CMOVE	76000, INCR,
	LINEAR	
2:	ACCEL	1000
	VELOC	4000
	JUMP	CTL01, 3
	PMOVE	50000, INCR,
	LINEAR	
3:	ACCEL	6000
	VELOC	6000
	CMOVE	6000, INCR, LINEAR



S-CURVE Jumps

Jumps during linear motion and jumps during s-curve motion at constant velocities immediately begin accelerating or decelerating to a new velocity. Jumps during a s-curve acceleration or deceleration, however, require different rules in order to maintain a s-curve profile. What happens when a jump occurs during an s-curve move while changing velocity depends on whether the jump occurs before or after the midpoint (the point where the acceleration magnitude is greatest) and whether the velocity at the jump destination is higher or lower than the current velocity.

S-CURVE Jumps after the Midpoint of Acceleration or Deceleration

If the jump occurs after the midpoint of the change in velocity, the change will continue normally until constant velocity is reached; then the velocity will be changed to the new velocity using the acceleration mode of the move at the jump destination.

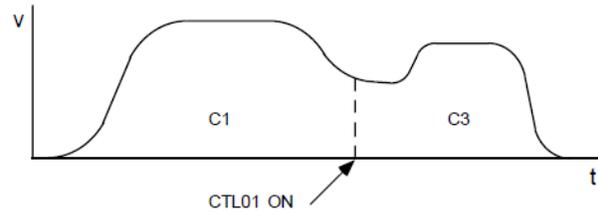
Example 12: S-CURVE - Jumping After the Midpoint of Acceleration or Deceleration

In the following example, a jump occurs during the final phase of deceleration, at the dashed line. The deceleration continues until constant velocity is reached and then the acceleration to the higher velocity begins.

Figure 83: Jumping After the Midpoint of Acceleration or Deceleration

```

ACCEL 50000
VELOC 100000
1: JUMP CTL01, 3
   CMOVE 500000, ABS, S-
   CURVE
2: VELOC 60000
   CMOVE 500000, INCR, S-
   CURVE
3: VELOC 85000
   ACCEL 100000
   CMOVE 250000, INCR, S-CURVE
    
```



S-CURVE Jumps before the Midpoint of Acceleration or Deceleration

If a jump takes place before the midpoint of acceleration or deceleration, the result depends on whether the velocity at the jump destination is higher or lower than the velocity before the jump took place. In the first case, when accelerating but the new velocity is lower, or decelerating and the new velocity is greater, the DSM314 will immediately begin reducing the acceleration or deceleration to zero. Once at zero velocity, the DSM314 will use the jump destination acceleration and velocity and change to the new velocity.

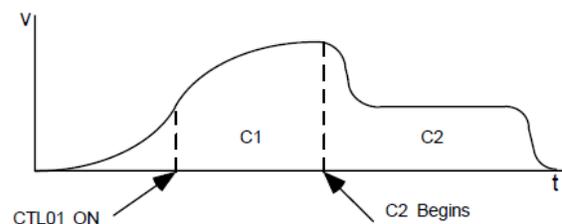
Example 13: S-CURVE - Jumping Before the Midpoint of Acceleration or Deceleration

In the following example, during the acceleration of the first CMOVE, a jump takes place at the first dashed line. Because the velocity at the jump destination is lower than the velocity of the first CMOVE the DSM314 slows the acceleration to zero. Constant velocity, zero acceleration, occurs at the second dashed line. There, the DSM314 begins decelerating to the new velocity using the acceleration at the jump destination. Finally, the second CMOVE finishes.

Figure 84: Jumping before the Midpoint of Acceleration or Deceleration

```

ACCEL 1000
VELOC 50000
1: JUMP CTL01, 3
   CMOVE 50000, INCR, S-CURVE
3: VELOC 5000
   ACCEL 10000
   CMOVE 15000, INCR, S-CURVE
    
```



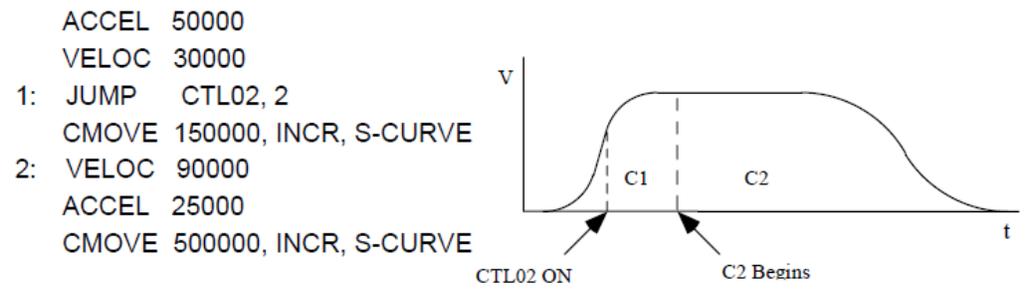
S-CURVE Jumps to a Higher Acceleration while Accelerating or a lower Deceleration while Decelerating

The second case involves jumping to a higher velocity while accelerating or a lower velocity while decelerating. When this occurs, the DSM314 continues to the first move's acceleration or deceleration. This acceleration or deceleration is maintained, similar to be a linear acceleration, until the axis approaches the new velocity. Then the normal S-curve is used to reduce acceleration or deceleration to zero.

Example 14: S-CURVE - Jumping to a Higher Velocity While Accelerating or Jumping to a Lower Velocity While Decelerating

In this example, a JUMP command is triggered during the initial phase of acceleration (at the first dashed line) and the velocity at the jump destination is higher than that of the current move. The first dashed line indicates the maximum acceleration of the first CMOVE. This value is held as the axis continues to accelerate until it s-curves back to constant velocity. Constant velocity, the second dashed line, indicates the beginning of the second CMOVE. This move continues until it decelerates to zero at the end of the program.

Figure 85: Jumping to a Higher Velocity While Accelerating, or Jumping to a Lower Velocity While Decelerating



7.7.6 Other Programmed Motion Considerations

The following examples are not complete programs. For example, in many cases the PROGRAM and ENDPORG statements are not shown. These statements (in correct context) would need to be added to make the program compile successfully.

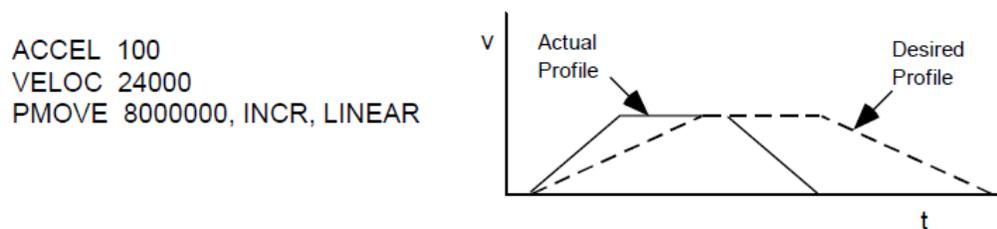
Maximum Acceleration Time

The maximum time for a programmed acceleration or deceleration is 131 seconds. If the time to accelerate or decelerate is computed to be longer than this time, the DSM314 will compute an acceleration to be used based on 131 seconds. To obtain longer acceleration times, multiple CMOVEs with increasing or decreasing velocities must be used.

Example 15: Maximum Acceleration Time

The following two program examples are only valid for a DSM314 using a 2ms position loop update time. They show a hypothetical problem with a very long acceleration time in Example 1, and a possible solution in Example 2. In Example 1 below, 240 seconds is required to reach the programmed velocity of 24,000 at an acceleration rate of 100 ($24000 \div 100 = 240$). Since this is greater than the DSM's limit of 131 seconds per acceleration or deceleration, the DSM will calculate a value within its limit. In this case, the DSM calculates that to reach a velocity of 24,000 in 131 seconds, an acceleration of 183 would be required. The Example 1 solid line velocity profile shows the higher (183) acceleration rate used by the DSM. The dashed line profile in that drawing indicates the desired (programmed) acceleration rate and velocity profile that could not be attained.

Figure 86: Maximum Acceleration Time Example 1



One solution (which requires some extra calculations) for obtaining a low acceleration for a long period of time breaks a move up into separate continuous moves (using CMOVE commands), with each move's acceleration time being less than 131 seconds. In the problem introduced in Example 1, the programmed move would require 240 seconds each for acceleration and deceleration. Dividing this time in half by using two moves with acceleration or deceleration times of 120 seconds each, places the moves within the DSM's limit of 131 seconds. This scheme is used in the following example.

Example 2 shows how the result desired in Example 1 could be obtained by replacing Example 1's single move with four moves. Four moves are required since both the acceleration and deceleration portions of the profile must each be divided into two moves. To divide the total acceleration (or deceleration) time in half, calculate the distance at the midpoint of either slope, when velocity is 12000, to be 720,000 user units.

The distance traveled during acceleration or deceleration is calculated using the formula:

$$\text{Distance traveled} = \frac{\text{Change in velocity} \times \text{Required time}}{2}$$

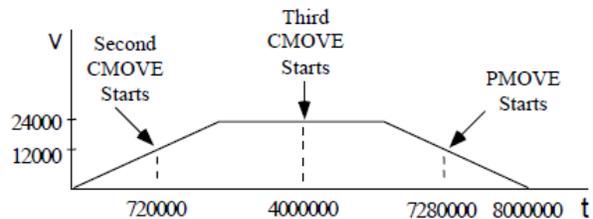
$$720,000 = \frac{12,000 \times 120}{2}$$

(For 240 seconds is needed to reach a velocity of 24,000, a velocity of 12,000 can be reached in 120 seconds.) The initial CMOVE and the final PMOVE both use this distance. A second CMOVE “takes over” at the midpoint of the acceleration slope from the first CMOVE and accelerates to the target velocity of 24,000. A third CMOVE is required for dividing up the deceleration portion of the profile. The final move, a PMOVE, “takes over” from the third CMOVE at the deceleration midpoint distance (720,000 user units from the final position). The third CMOVE, as it approaches its final position, will automatically decelerate to the PMOVE’s velocity of 12,000. The dashed lines in the Example 2 drawing separate the four moves. To calculate the distances of the second and third CMOVES, subtract the distances calculated for the first CMOVE and final PMOVE (720,000 each for a total of 1,440,000) from the final distance of 8,000,000. This gives a remaining distance of 6,560,000, which is divided equally between the second and third CMOVES (3,280,000 each).

Figure 87: Maximum Acceleration Time Example 2

```

ACCEL 100
VELOC 12000
CMOVE 720000, INCR, LINEAR
VELOC 12000
CMOVE 3280000, INCR, LINEAR
VELOC 24000
CMOVE 3280000, INCR, LINEAR
VELOC 12000
PMOVE 720000, INCR, LINEAR
    
```



7.7.7 Feedhold with the DSM314

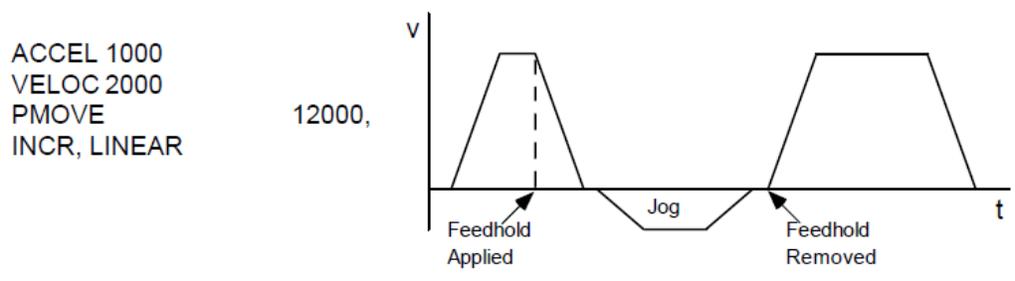
Feedhold is used to temporarily pause program execution without ending the program, often to examine some aspect of a system. It causes all axis motion to end at the programmed acceleration. When Feedhold is ended, program execution resumes. Interrupted motion will resume at the programmed acceleration and velocity.

Feedhold is asserted by turning ON the Feed Hold %Q bit and lasts until the %Q bit is turned OFF. The Abort All Moves %Q bit turning ON or an error that would normally cause a stop error will end feedhold as well as terminate the program. During Feedhold, jogging positive and negative is allowed, but no other motion. When Feedhold is terminated and program execution resumes, the DSM314 remembers and will move to its previous destination.

Example 16: Feedhold

The following example illustrates a motion profile when Feedhold is applied. The linear move accelerates to the programmed velocity at the programmed rate. Feedhold is applied at the dashed line, so velocity decreases at the programmed acceleration to zero. Then, a Jog is performed using the Jog Minus %Q bit. This is evident because the jog velocity is negative. Note that the acceleration used during the Jog is the current **Jog Acceleration**, which is different than the programmed acceleration. Note also, the Feed Hold %Q command must be applied during the entire duration of the Jog. After the jog motion has ceased, the Feedhold is ended and the program continues to completion.

Figure 88: Feedhold Example



7.7.8 Feedrate Override

Some applications require small modifications to a programmed velocity to handle outside changes. A Rate Override %AQ immediate command, which is sent to the DSM through ladder logic, allows changes to a programmed feedrate (velocity) during program execution. (Details about the Rate Override command are found in Chapter 5.) When a program begins executing, the override rate is initially set to 100%. Thus, changes to feedrate before the execute program bit is turned ON will be ignored. However, a rate override commanded on the same sweep as an execute program bit will be effective.

A percentage can be assigned to the feedrate override of from 0% to 120%. When a Rate Override is commanded, the DSM314 internally multiplies the feedrate percentage by programmed velocity to obtain a new velocity. If the axis is moving, the current move's Jog Acceleration Mode is used to change velocity to the new velocity. All future move velocities will be affected by the feedrate change. Note that when a feedrate of 0% is applied, no motion will be generated until a new feedrate is commanded. Also note the Moving %I bit stays ON when the feedrate is 0%.

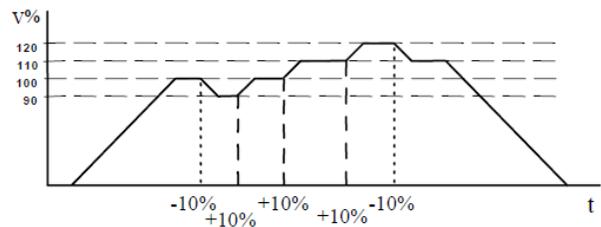
Rate Override has no effect on non-programmed motion such as Jog, Find Home, or Move at Velocity.

Example 17: Feedrate Override

During execution of this program, feedrate changes of + or -10% are commanded. Dotted lines indicate -10%, dashed lines indicate +10%.

Figure 89: Feedrate Override Example

ACCEL	1000
VELOC	6000
PMOVE	110000,
INCR, LINEAR	



7.7.9 Multi-axis Programming

Sync Blocks can be used in a multi-axis program to synchronize the axis motion commands at positions where timing is critical.

Example 18: Multi-axis Programming

This example assumes that axis 1 controls vertical motion and axis 2 controls horizontal motion. The objective is to move a piece of material from point A to point E as quickly as possible while avoiding the obstacle that prevents a direct move between those points.

A simple way would be to move straight up from point A to point C, and then from point C to point E. This sequence, however, wastes time. A better way would begin the horizontal movement before reaching point C. It has been determined that after axis 1 has moved to a position of 30,000, user units (to point B), axis 2 could then start and still clear the obstacle. The program segment could be programmed as follows:

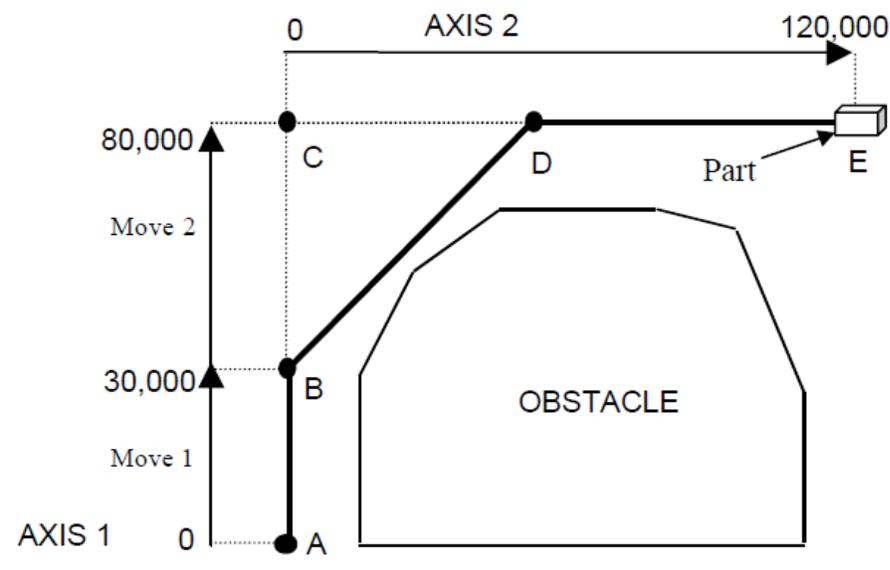
```

10:  CMOVE  AXIS1 30000, INCR, LINEAR
20:  SYNC
    PMOVE   AXIS1 50000, INCR, LINEAR
    PMOVE   AXIS2 120000, INCR, LINEAR

```

When Block 10 is executed, axis 1 begins its 30,000-unit move while axis 2 pauses. When the axis 1 move completes, two things occur: axis 1 begins the 50,000-unit PMOVE commanded in Block 20 (SYNC) without stopping (because the first move was a CMOVE), and axis 2 begins its 120,000-unit move. In the figure below, the axis 1 first move transfers the part from point A to point B. At point B, axis 1 continues moving (performing its second move) and axis 2 begins its move, bringing the part to point D. Axis 1 completes its second move at point D and stops; however, axis 2 continues, and moves the part to point E.

Figure 90: Feedrate Override Example



If this program segment is not at the beginning of a program, and for some reason axis 2 has not yet reached Block 20 when axis 1 has moved 30,000 counts, an error would occur. Axis 1 would continue to 80,000 counts, and the DSM314 would report a “Block Sync Error during a CMOVE” in the Status Code.

If it is imperative that the axes synchronize at Block 20, changing Block 10 to a PMOVE would guarantee synchronization, but then axis 1 would stop at 30,000 counts.

7.7.10 Parameters (P0-P255) in the DSM314

The DSM314 maintains 256 double word parameters (0 through 255) in memory. These parameters can be used as variables in ACCEL, VELOC, DWELL, PMOVE, and CMOVE motion commands. Be aware that range limits still apply, and errors may occur if a parameter contains a value out of range. Parameters 216-255 are special purpose parameters. Some of the special purpose parameters are automatically written by the DSM314. For example, P224 is automatically updated when Position Strobe 1 on Axis 1 occurs. The following table describes the function of the special purpose parameters.

Table 49: Special Purpose Parameters

Parameter Number	Special Purpose Function	Axis	Units
216-223	Reserved		
224	Position Strobe 1	Axis 1	user units
225	Position Strobe 2	Axis 1	user units
226	Commanded Position at Follower Enable Trigger	Axis 1	user units
227	Follower Incremental Stop Distance	Axis 1	user units
228-231	Reserved		
232	Position Strobe 1	Axis 2	user units
233	Position Strobe 2	Axis 2	user units
234	Commanded Position at Follower Enable Trigger	Axis 2	user units
235	Follower Incremental Stop Distance	Axis 2	user units
236-239	Reserved		
240	Position Strobe 1	Axis 3	user units
241	Position Strobe 2	Axis 3	user units
242	Commanded Position at Follower Enable Trigger	Axis 3	user units
243	Follower Incremental Stop Distance	Axis 3	user units
244-247	Reserved		
248	Position Strobe 1	Axis 4	user units
249	Position Strobe 2	Axis 4	user units

Parameter Number	Special Purpose Function	Axis	Units
250	Commanded Position at Follower Enable Trigger	Axis 4	user units
251	Follower Incremental Stop Distance	Axis 4	user units
252-255	Reserved		

Parameters are all reset to zero after a power cycle or after a DSM314 configuration is stored by the host controller. Parameters can be assigned in three ways:

- The motion program LOAD command.
- The Load Parameter Immediate %AQ command.
- The COMM_REQ function block. This is the preferred way if you need to send multiple parameters per scan. The COMM_REQ function block is described in Appendix B.

Assigning a value to a parameter overwrites any previous value. Parameter values can be changed during program execution, but the change must occur before the DSM314 begins executing the Type 3 command (Move, Wait or Dwell) previous to the Type 3 command that uses the parameter. This is due to the pre-processing of Type 3 commands that occurs within the DSM314. Note that a JUMP command clears preprocessing and forces the program commands at the jump target to be processed.

Below is an example of a motion program using Parameters. The values of Parameters 1-5 are pre-loaded with a COMREQ command from the host controller at least two program blocks before usage.

(Remember that “program blocks” are not the same as sections of the motion program that are labeled with the BLOCK # command.)

Block/Command/Data Comments

```

1:  VELOC P001 // Set velocity of first move = value in Parameter 1
    ACCEL P002 // Set acceleration of first move = value in Parameter 2
    CMOVE P003, ABS, LINEAR // Reach velocity of 2nd move (20000) at position = Par. 3
2:  VELOC 20000 // Set velocity of second move = 20000
    PMOVE 20000, INCR, LINEAR // Normal PMOVE
    DWELL P004 // Dwell for Parameter 4 time
    PMOVE P005, INCR, LINEAR // PMOVE to value in Parameter 5
(* Strobe #1 occurs on Axis-1 during move to Param. 5 position *)
    DWELL 1000 // Dwell for one second
    LOAD P006,2000 // Load Parameter 6 parameter
3:  MOVE P224, INCR, LINEAR // Move to strobed position for Strobe #1 on axis-1
    DWELL 2000 // Dwell for two seconds
    PMOVE P006, ABS, S-CURVE // Final stop position = value in Parameter 6

```

7.7.11 Calculating Acceleration, Velocity and Position Values

One method of determining the value for APM or DSM motion program variables such as Acceleration, Velocity or Position is to plot the desired move or move segment as a velocity profile. A velocity profile plots time on the horizontal axis of a graph and velocity on the vertical axis. The key to understanding profile generation is to break the complete move into smaller segments that may be analyzed geometrically. Most applications will use the economical trapezoidal move, velocity profile as illustrated below. To move as quickly as possible, use a triangular velocity profile if the servo has sufficient speed range. A triangular move would accelerate half the distance then decelerate the remaining half. Another alternative is to use a trapezoidal profile with a shorter slew segment.

Kinematic Equations

Kinematics is the branch of mechanics that studies the motion of a body or a system of bodies without consideration given to its mass or the forces acting on it. The following table includes transformations of the basic linear equations as applied to the acceleration portion of motion profiles. Use these formulae to calculate the velocity and acceleration for the acceleration portions of the move.

Table 50: Linear Equation Transformations

Given \ Solve For	A, X	A, V	A, t	V, t	V, X	X, t
Acceleration				V/t	V ² /2X	2X/t ²
Velocity	$\sqrt{2AX}$		At			2X/t
X (Distance)		V ² /2A	At ² /2	Vt/2		
time	$\sqrt{2X/A}$	V/A			2X/V	

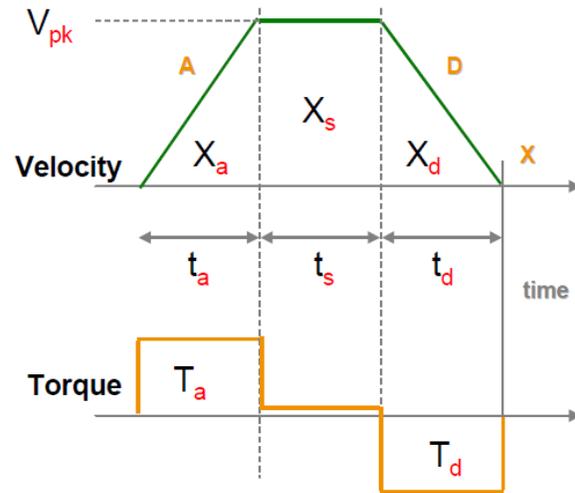
Figure 91 provides an example of a trapezoidal move. Beginning at zero velocity the axis accelerates in a positive direction (ta), run (slew) at velocity for some time (ts), then decelerate back to zero velocity (td). That's a complete move or move segment. By looking at the figure, you can easily separate the different portions of the move. A common rule of thumb is to divide the trapezoidal move into three-time portions, one-third for acceleration, one-third at slew velocity and the remaining third to decelerate. The slew (Xs) section of an equally divided trapezoidal velocity profile represents ½ of the distance moved and the acceleration and deceleration portions each represent ¼ of the total distance. The rule of thirds minimizes the RMS torque current in the motor and is the most economical use of energy.

Figure 91: Trapezoidal Move

Trapezoidal Move

- Limits max motor speed
- Higher accel torque than triangle move
- Symmetrical profile (1/3, 1/3, 1/3 time) maximizes power transfer to load
- Most common for long moves

- A = acceleration
- D = deceleration
- X = distance
- V_{pk} = velocity peak
- t_a = time acceleration
- t_s = time at slow velocity
- t_d = time deceleration
- T_a = acceleration Torque
- T_d = deceleration Torque
- X_a = acceleration distance
- X_s = slow distance
- X_d = deceleration distance



Equations

$$x = V_{pk} (0.5t_a + t_s + 0.5t_d)$$

$$V_{pk} = \frac{x}{(0.5t_a + t_s + 0.5t_d)}$$

$$a = \frac{V_{pk}}{t_a} \quad \text{or} \quad D = \frac{V_{pk}}{t_d}$$

Once the move segment outline is drawn, you will need to examine specifications or physical restrictions applicable to the move. For instance, the move may have to complete in a certain time interval ($t_a + t_s + t_d$) or move a fixed distance (X). The maximum velocity (V_{pk}) of the servomotor is one example of a physical limitation. Given any two known values of the acceleration portion of the move segment, a remaining variable can be found using the kinematic equations as illustrated in the example below.

Trapezoidal Velocity Profile Application Example

For this example, assume that a complete move of 16 inches must be made in three seconds and the maximum motor velocity, translated through the gearing is 15 inches per second. Using the rule of thumb for trapezoidal moves, divide the move's time into thirds: $t_a = 1$ sec, $t_s = 1$ sec and $t_d = 1$ sec. You can also subdivide the 16-inch move into three distances. The slew (X_s) section of an equally divided trapezoidal velocity profile represents $\frac{1}{2}$ of the distance moved and the acceleration (X_a) and deceleration (X_d) portions each represent $\frac{1}{4}$ of the total distance: $X_a = 4$ in, $X_s = 8$ in and $X_d = 4$ in.

To calculate peak Velocity (V_{pk}), the first acceleration portion of the move must travel a given distance (X_a) in a given time (t_a). From the above Kinematic Velocity formula ($2X/t$) using the given, $X_a = 4$ inches and $t_a = 1$ second, $(2 * 4 \text{ inches}) / 1 \text{ second} = 8 \text{ inches/second}$.

To calculate Acceleration the simplest formula is $(V/T) = (8 \text{ inches/second} / 1 \text{ second}) = 8 \text{ inches/second/second}$.

The Position (Distance = X) is the entire distance moved ($X_a + X_s + X_d$) or 16 inches.

Triangular Velocity Profiles

The triangular velocity profile minimizes servo acceleration rate and requires a higher servomotor velocity when compared to a trapezoidal profile of the same distance and time. Use a triangular profile for fast short moves.

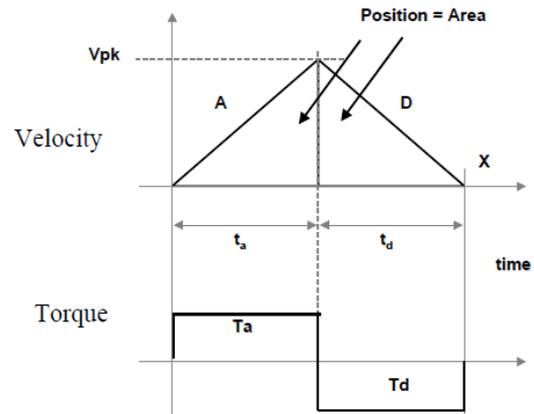
Figure 92: Triangular Velocity Profile

Equations:

$$x = \frac{1}{2} V_{pk} (t_a + t_d)$$

$$V_{pk} = \frac{2(x)}{(t_a + t_d)}$$

$$a = \frac{V_{pk}}{t_a}$$



Non-Linear or S-Curve Acceleration

S-Curve or jerk limited acceleration calculation is simple to do if the linear calculation is accomplished first. The APM and DSM motion controllers use 100% jerk limiting. To convert a linear acceleration to 100% jerk limited acceleration you either double the Acceleration value (2*A) or double the time used for acceleration (2ta). Using S-Curve acceleration at the same acceleration rate (A) as linear acceleration will require twice the time (ta) reaching velocity. If the time duration of the move must remain the same and the servo has sufficient peak torque, use twice the acceleration (2*A) to reach velocity in the same amount of time.

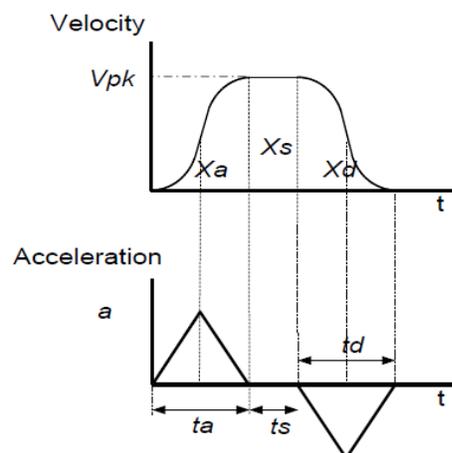
Figure 93: S-Curve Acceleration

Equations:

$$X_a = X_d = \frac{V_{pk}^2}{a}$$

$$t_a = t_d = \frac{2V_{pk}}{a}$$

$$t_s = \frac{X_s}{V_{pk}}$$



7.7.12 Motion Editor Error and Warning Messages

The editor will generate three types of error messages; syntax errors, semantic errors, and warnings. These are explained below.

The editor will only generate program code if your source motion program contains no syntactic or semantic errors. If the editor detects unrecognized syntax or semantic errors, it will generate an error message that can be used to troubleshoot the program. The last page of this chapter discusses this subject (“Using Error Messages to Troubleshoot Motion Programs”).

Error messages displayed in the Status window contain a numeric error code. The following listing matches error code, error description, and common cause information.

The Motion Editor enforces maximum limits for position, velocity, and acceleration based 8:1 uu/cts scaling.

Syntax Errors

The programming software’s motion editor translates programs into the code used by the DSM314. If the source code violates the syntactic rules, the editor cannot recognize the code and generates syntax errors. Syntax errors will attempt to describe the error source.

Semantic Errors

This section describes parse errors reported by the motion parser and their typical causes.

(M200) Undefined identifier

Text string is not a recognized motion program variable or keyword.

(M201) Parameter register must be in range of P000 - P255

Motion program referenced a parameter register outside the range of P000 - P255.

(M203) CTL variable must be in range CTL01 - CTL32 (DSM314)

Motion program referenced a CTL bit outside the valid range.

(M204) Invalid motion program input

Motion program file contains an invalid character. Motion program files must contain only ASCII text or white space.

(M210) Hexadecimal constants must be in range of 16#0 - 16#FFFFFFFF

Motion program contains a hexadecimal number outside the valid range.

(M211) Binary constants must be in range of 0 to (2³²)-1

Motion program contains a binary number outside the valid range. A binary number cannot contain more than 32 binary digits.

(M212) Integer constants must be in range of 0 to 4294967294

Motion program contains an unsigned integer value that cannot be represented in 32 bits.

(M213) Signed integer constants must be in range of -2147483648 to 2147483647

Motion program contains a signed integer value that cannot be represented in 32 bits.

(M214) SYNC Statement is only valid in multi-axis programs and subroutines

A single-axis motion program or subroutine attempted to define a sync block.

(M215) Multi-axis programs do not support Axis 3 or 4

Commands in multi-axis programs can only reference axis 1 or 2.

(M220) Specified axis is out of range

A single-axis motion program can only reference axis 1, 2, 3, or 4.

(M221) Acceleration must be in range 1 – 1073741823

An ACCEL command has specified an acceleration outside the valid range.

(M222) Velocity must be in range 1 – 8388607

A VELOC command has specified a velocity outside the valid range.

(M223) Position must be in range -536870912 – 536870911

A CMOVE or PMOVE command has specified a position outside the valid range.

(M224) Dwell must be in range 0 – 60000

A DWELL command has specified a dwell outside the valid range.

(M225) Block number must be in range 1 – 65535

A motion program or subroutine has attempted to define a block number outside the valid range.

(M230) Must specify an axis in a multi axis program

ACCEL, VELOC, CMOVE, PMOVE, DWELL, and WAIT commands in a multi-axis program or subroutine must specify an axis.

(M231) Cannot specify axis in a single-axis program

ACCEL, VELOC, CMOVE, PMOVE, DWELL, and WAIT commands in a single-axis program or subroutine must not specify an axis.

(M233) Acceleration reassignment without intervening move command

It is illegal to change the acceleration for a given axis if there is not an intervening PMOVE or CMOVE command.

(M234) Velocity reassignment without intervening move command

It is illegal to change the velocity for a given axis if there is not an intervening PMOVE or CMOVE command.

(M235) Block number already defined in this program unit

The motion program or subroutine has attempted to define a block number that has already been defined.

(M236) Jump destination block not defined

The motion program or subroutine has a JUMP statement to a block number that has not been defined.

(M237) Call destination subroutine not defined

The motion program or subroutine contains a call to a subroutine that has not been defined.

(M238) Program must be in range 1 – 10

A PROGRAM statement is attempting to define a program number that is outside the valid range.

(M239) Attempt to redefine program. Program already defined

A PROGRAM statement is attempting to define a program using a program number that is already defined.

(M240) End program definition with ENDPROG statement

A PROGRAM has been terminated with an ENDSUB statement, or an ENDSUB statement was encountered within a program.

(M242) Missing ENDPROG statement

A PROGRAM had not been terminated with an ENDPROG statement when the end of file was encountered.

(M243) Subroutine must be in range 1 – 40

A SUBROUTINE statement is attempting to define a subroutine number that is outside the valid range.

(M244) Attempt to redefine subroutine. Subroutine already defined

A SUBROUTINE statement is attempting to define a program using a subroutine number that is already defined.

(M245) End subroutine definition with ENDSUB statement

A SUBROUTINE has been terminated with an ENDPROG statement, or an ENDPROG statement was encountered within a subroutine.

(M246) No subroutine is being defined

The program block contains an ENDSUB command, but there is no open SUBROUTINE.

(M247) Subroutine cannot call itself

The DSM does not support any kind of recursion. Once invoked a subroutine cannot call itself or be called by a subroutine that it has invoked.

(M248) Axis definition of subroutine must match caller

An attempt has been made to call a single-axis subroutine from a multi-axis program or subroutine, or call a multi-axis subroutine from a single-axis program or subroutine.

(M249) Already defining program or subroutine

A PROGRAM or SUBROUTINE statement has been encountered within an unterminated PROGRAM or SUBROUTINE.

(M280) Instruction limit exceeded, max 1000

A motion program block can contain no more than 1000 program statements. This error is issued if the number of statements exceeds that limit.

(M281) File must contain at least one program

A motion program block must contain at least one PROGRAM; otherwise, there is no way to invoke it. This error is issued if a motion program block does not contain any PROGRAMs.

(M282) Statement must be within a program or subroutine

This error is issued if motion program commands occur outside a PROGRAM or SUBROUTINE.

(M283) This instruction is invalid for the specified module type

A motion program block contains an instruction that is invalid for the destination module.

(M293) Maximum error count exceeded.

The motion program parser reports up to 30 errors when parsing a motion program block. When that limit is reached, this error is issued and no more errors are reported.

(M300) Parse directives must precede any executable statements

A #pragma directive must be issued at the beginning of the motion program block, i.e. preceding any motion program statements.

(M301) Invalid directive option

An invalid #pragma directive has been specified.

(M302) Invalid directive parameter

An invalid option has been specified as #pragma directive parameter.

Warnings

Warnings are generated for code that seems questionable but does not specifically cause an error. This section describes parse warnings reported by the motion parser and their typical causes.

(M482) Unexpected end of program: unclosed comment

A comment was not terminated when an end of file was encountered.

(M483) Nested comments.

The motion parser does not support nesting comments. A warning is issued if a comment is defined within a comment.

(M490) Program contains no executable statements

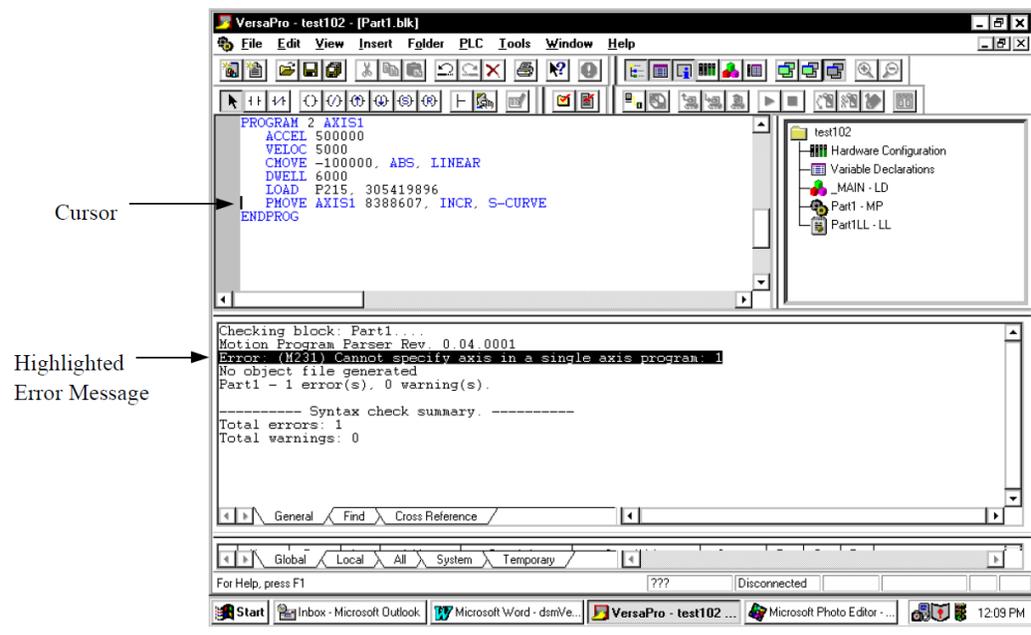
A warning is issued if a program block contains no executable statements.

Using Error Messages to Troubleshoot Motion Programs

After creating motion programs or subroutines in the Motion Editor window, you can check for basic errors by clicking the Block Check icon  on the toolbar.

The editor checks the motion program block and report any errors it detects in the Information window. The next figure shows an example of an error detected during the check.

Figure 94: Using Error Messages to Troubleshoot Motion Programs



The error shown in the above figure, “Error: (M231) Cannot specify axis in a single-axis program: 1,” refers to the last line of the program, just before the ENDPORG statement. Notice that AXIS1 is found on this line. Since PROGRAM 2 is a single-axis program, the use of axis numbers within the program is not allowed so an error was generated.

In the above example, the error message was double clicked, as indicated by the fact that it is highlighted in reverse video. When this is done, the cursor jumps to the place in the program that produced the error. You will note the presence of the cursor at the start of the line containing the AXIS1 statement.

For further help in troubleshooting errors, the “Motion Editor Error and Warning Messages” section of this chapter lists common causes for the various error codes. For example, the listing for error (M231), seen in the example above, states:

(M231) Cannot specify axis in a single-axis program

ACCEL, VELOC, CMOVE, PMOVE, DWELL, and WAIT commands in a single-axis program or subroutine must not specify an axis.

Chapter 8: Follower Motion

Configuring the DSM314 for Follower Control Loop = Enabled (in the configuration software Axis Configuration tab) allows each Servo Axis (slave) to respond to a Master Axis input using a programmable slave: master ratio. The DSM314 defines the slave: master ratio as the ratio of two integer numbers A and B. The basic formula for computing Follower motion is:

$$\text{Follower Servo Axis motion (slave axis)} = \text{Master Axis motion} \times (A/B)$$

or

$$\text{slave : master ratio} = A : B \text{ ratio}$$

If a Jog, Move at Velocity or Execute Motion Program command is also initiated, the axis motion will represent the combination of the Master Axis motion and the internally commanded motion. This Chapter provides details of servo motion related to the Master Axis input. Refer to Chapter 9 for additional information about combined Follower and commanded motion.

When the Enable Follower %Q bit is turned ON, an axis will immediately begin following the selected Master Source unless a Follower Enable Trigger input has been selected. If a Follower Enable Trigger input has been selected, then the Enable Follower %Q bit must be ON and an OFF to ON transition of the trigger input must occur. The external trigger input CTL01 - CTL032 is selected in the configuration software.

The DSM314 always operates the follower axis in “ramp makeup” mode. If the master axis has a nonzero velocity when the follower is enabled, the slave axis will accelerate at the configured Ramp Makeup Acceleration to a speed that allows it to catch up to the master axis.

8.1 Master Sources

A DSM314 Servo Axis can be configured to follow any two of eight possible master input sources. The two sources are identified as Source 1 and Source 2. A Follower Master Source Select %Q bit determines whether Source 1 or Source 2 is the active source. The available selections for Source 1 and Source 2 are:

- Axis 1 Commanded Position
- Axis 1 Actual Position
- Axis 2 Commanded Position
- Axis 2 Actual Position
- Axis 3 Commanded Position
- Axis 3 Actual Position
- Axis 4 Commanded Position
- Axis 4 Actual Position

Note that follower motion is summed with Jog, Move at Velocity, or Motion Programs. If a slave axis is following a master input at velocity $V1$, and a Jog is commanded at velocity $V2$, the axis will move at velocity $V1 + V2$.

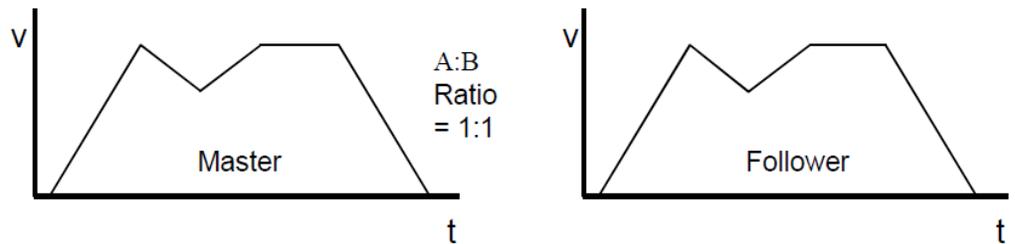
8.2 External Master Inputs

Actual Position for Axis 1 - Axis 4 represent external master axis sources. An encoder connected to the axis or the feedback of a servo system may be used as an actual position source. The DSM314 follower loop allows a slave axis to follow a selected external source as shown in this example:

8.2.1 Example 1: Following Axis 3 Actual Position Master Input

In this example, a graph of velocity (v) versus time (t) shows the velocities of the master input (Actual Position 3), and the slave axis that is following the master. The DSM314 is configured with Follower Master Source 1 = Actual Position 3 and the Select Master Source %Q bit is OFF. The A:B ratio is 1:1. The velocity profile of the following (slave) axis is identical to the master input.

Figure 95: Following Encoder 3 Master Input



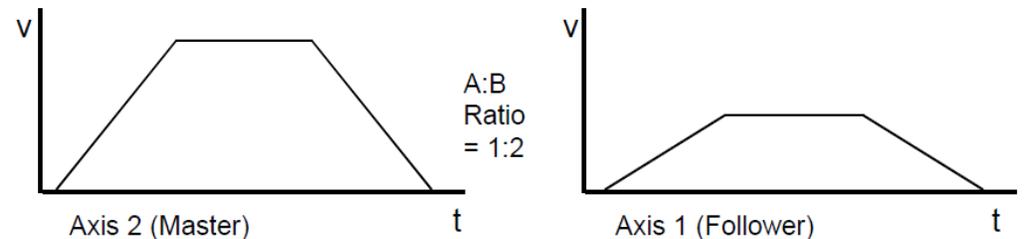
8.3 Internal Master Axis Command Generators

Commanded Position for Axis 1 - Axis 4 represent internal master axis sources. The DSM314 follower loop will allow a slave axis to follow a selected internal command source as shown in this example:

8.3.1 Example 2: Following an Internal Master command

In this example, Axis 1 of the DSM314 is configured with Follower Master Source 2 = Commanded Position 2 and the Select Internal Master %Q bit is ON. The A:B ratio is 1:2. Axis 2 is commanded to Move at Velocity 12000 and then 0. Axis 1 follows axis 2 at half of the axis 2 velocity and acceleration and moves only half the distance that axis 2 has moved.

Figure 96: Following Servo Axis 2 Encoder



8.4 A:B Ratio

A DSM314 axis following a master input can do so at a wide range of slave : master (A:B) ratios. The “A” value can be any number from -32768 to 32767. The “B” value can be anywhere between 1 and 32767. The magnitude of the A:B ratio can be from 1:10,000 to 32:1. Thus very precise ratios such as 12,356:12,354 or 32,000:1024 can be used.

The Follower A/B Ratio %AQ command can be used to change the A:B ratio at any time, even while following. However, an invalid ratio will generate a status error and be ignored. An invalid ratio is a ratio with B equal to or less than zero or A:B magnitude greater than 32:1 or less than 1:10,000.

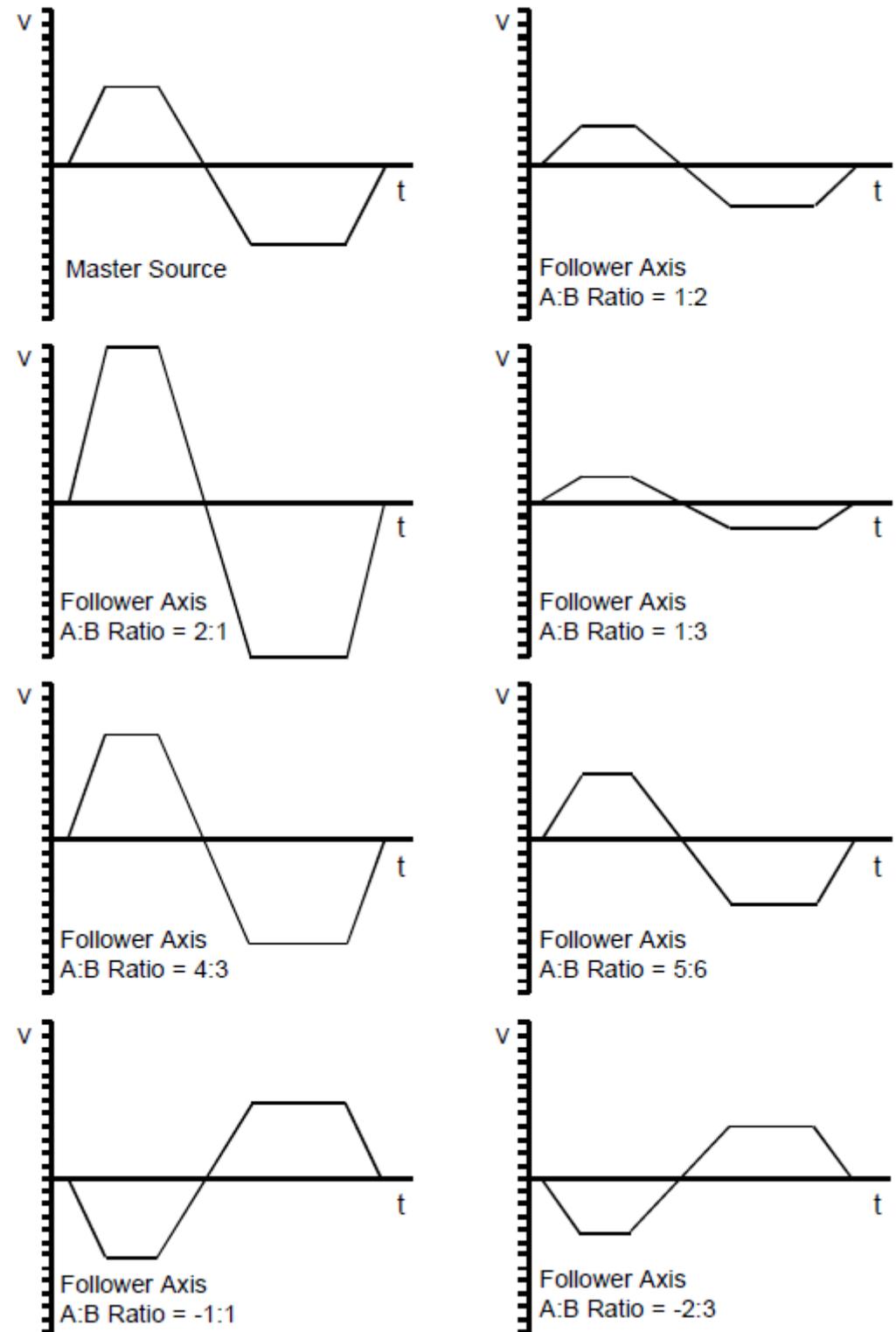
When following with a non 1:1 ratio, the velocity profile of the master and follower will look somewhat different. Horizontal lines, indicating constant velocity, and slanted lines, indicating acceleration and deceleration, will be different. **If the A:B ratio is less than 1:1, the follower velocity and acceleration will be less than the master. Likewise, if the A:B ratio is greater than 1:1, the follower velocity and acceleration will be greater than the master.** The duration of motion, and time that the slave axis will accelerate, decelerate, or stay at constant velocity are the same for the master and follower.

The distance moved, which in a velocity profile is the area between the graph and the time axis, will be that of the master multiplied by the A:B ratio. If A is zero, no following motion will be generated. If A is negative, the following axis will move with the direction of motion reversed.

8.4.1 Example 3: Sample A:B Ratios

All of the following samples are following the master source input at various A:B ratios.

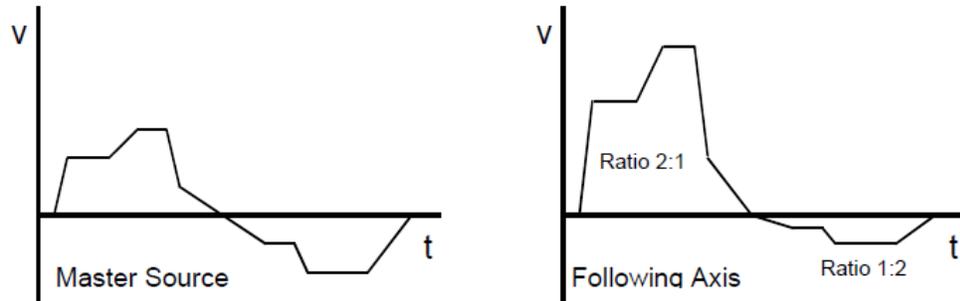
Figure 97: Sample A:B Ratios



Example 4: Changing the A: B Ratio

One example of variable A:B ratios is to use one ratio while moving positive, and another when moving negative. Note that determination of positive and negative velocity and update of the A:B ratio must be done in the host controller or the DSM314 Local Logic program. In the profile below, the following axis uses a 2:1 ratio when moving positive and a 1:2 ratio when moving negative.

Figure 98: Changing the A:B Ratio



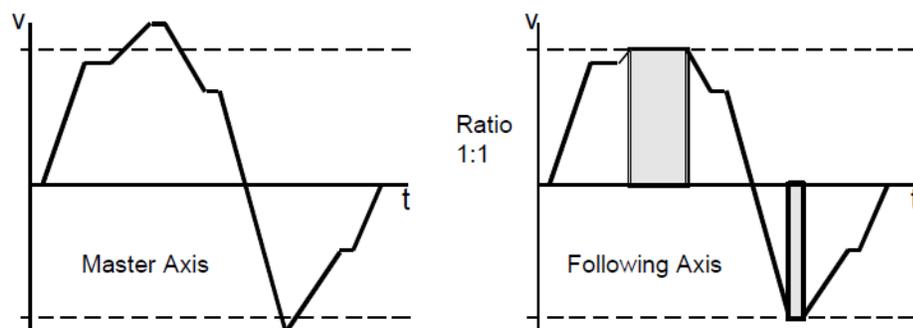
8.5 Velocity Clamping

Velocity clamping is available using the Velocity Limit set in the Configuration software. When the master velocity exceeds the configured limit, the following axis will continue to move at the limit velocity multiplied by the A:B ratio. The Velocity Limit %I bit is set and a status error is generated to indicate that the slave axis is no longer locked to the master input positioning. The slave axis has essentially fallen behind the master input.

8.5.1 Example 5: Velocity Clamping

The Velocity Limit is set to 100,000 in this example. Thus, the slave axis velocity is clamped at 100,000 user units/sec in either direction. When the master axis peaks greater than the limits, the following axis stays at the limit. After the master slows to under the limit, the following axis continues tracking the master axis velocity. Counts generated in excess of the Velocity Limit are lost to the follower. The horizontal dashed lines indicate the velocity limits. The shaded area indicates the times when the In-Velocity Limit bit is ON and the following axis is falling behind the master.

Figure 99: Velocity Clamping



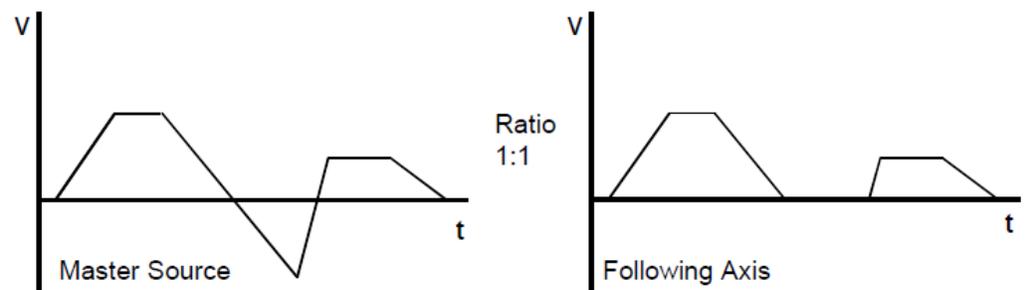
8.6 Unidirectional Operation

Setting the axis Command Direction configuration to Positive Only or Negative Only results in unidirectional follower motion. Any master axis counts in the zero limited direction are ignored. No error is generated by counts in the zero limited direction. The In Velocity Limit %I bit, however, does reflect the presence of a master command in the zero limited direction.

8.6.1 Example 9: Unidirectional Operation

In this example, the Command Direction configuration is set to Positive Only. As shown in the velocity profile below, the slave axis follows the positive counts, but ignores the negative counts. Note that when the master is moving negative, the In Velocity Limit %I bit is ON, but no status error is generated.

Figure 100: Unidirectional Operation



8.7 Enabling the Follower with External Input

Any CTL bit CTL01- CTL32 can be configured as an enable trigger for the follower axis. If a CTL bit source is configured as an external faceplate input, that input can be used to start the follower. When no input is selected, the follower is enabled and disabled directly by the Enable Follower %Q bit. When an input is selected for the Enable Trigger and the Enable Follower %Q bit is set, the next positive transition of the defined input will instantly enable the follower. The follower will remain enabled until the Enable Follower %Q bit is cleared. The faceplate 24v inputs have 5 ms filters that result in a Follower Enable Trigger response time of 5-7 milliseconds. The faceplate 5v inputs do not have these filters and will provide an Enable Trigger response time of 2 millisecond or less.

When the Enable Follower trigger occurs, the Commanded Position at that point is captured in a parameter register so that it can be used in a Programmed Move command. The position is captured in parameter 226 (for Servo Axis 1), parameter 234 (for Servo Axis 2), parameter 242 (for Servo Axis 3) or parameter 250 (for Servo Axis 4).

Follower Enabled status is returned in a %I bit for each axis.

8.8 Disabling the Follower with External Input

Any CTL bit CTL01- CTL32 can be configured as a Disable Trigger for the follower axis. The trigger input is tested only when the Enable Follower %Q bit is ON. When the Enable Follower %Q bit is ON, an OFF to ON transition of the trigger bit will disable the follower. Turning OFF the Enable Follower %Q bit immediately disables the follower, regardless of the disable trigger configuration.

8.9 Follower Disable Action Configured for Incremental Position

Configuring the **Follower Disable Action** for Inc Position allows the follower axis to perform an **Incremental Registration Move**. Disabling the follower with the Enable Follower %Q bit or optional **Disable Trigger** will cause the axis to continue at its present velocity, then decelerate and stop after a specified distance has elapsed. The incremental distance is specified in a parameter register for each axis:

P227 = Axis 1 Incremental distance

P235 = Axis 2 Incremental distance

P242 = Axis 3 Incremental distance

P250 = Axis 4 Incremental distance

The incremental distance represents the total actual position change that will occur from the point where the follower is disabled until it stops. Superimposed motion commands (Jog, Move at Velocity or Motion Programs) should not be active during a Follower Registration Move.

8.10 Follower Axis Acceleration Ramp Control

For applications where the Follower is enabled after the Master command is already up to speed, the Follower Ramp feature can be used to apply a controlled acceleration rate to bring the follower axis up to speed. This may be done without losing any Master command counts from the point at which the Follower was enabled. During the automatically generated Follower Ramp Control make-up move, the acceleration/deceleration does not exceed the configured **Follower Ramp Acceleration** value and provides a smooth motion. When the follower is enabled, the slave axis is ramped up to the master velocity at the active configured **Follower Ramp Acceleration** rate. This function is most useful when the master source is in motion before the follower mode is enabled. In addition to the host controller Enable Follower %Q bit, a CTL bit (CTL01-CTL32) may be configured as the enable follower signal for position registration functions. When the Enable Follower %Q bit is ON, then the CTL bit chosen acts as a rising edge trigger to enable follower mode. After Follower is enabled, only the host controller Enable Follower %Q bit controls the active state of the following function. When the follower axis is enabled to a moving master source, some master source counts cannot be used immediately. The master counts that accumulate during acceleration of the follower axis are stored. When the follower axis reaches the

master velocity, they will be inserted during make-up distance correction motion. This motion has an automatically calculated trapezoidal velocity profile determined by the **Follower Ramp Distance Makeup Time**, the amount of accumulated counts, and the configured **Follower Ramp Acceleration**. Set the **Follower Ramp Distance Make-up Time** to the desired time in the configuration software or it can be changed with the host controller %AQ Command 42h.

If the **Follower Ramp Distance Makeup Time** is too short, then the automatically generated velocity profile is triangular in profile. If during the distance correction velocity exceeds 80% of the configured Velocity Limit, then the automatically calculated move velocity will be clamped at 80% of the limit value. Clamping the makeup move velocity at 80% of the velocity limit allows the system some reserve velocity capacity for continued tracking of the master source velocity. In both cases a warning message is reported, and the real distance make-up time is longer than programmed, but the distance is still corrected properly.

Setting a **Follower Ramp Distance Make-Up Time** of 0 allows the Ramp feature to accelerate the axis without making up any of the accumulated counts. In this instance velocity will not exceed the master velocity. For applications where lost counts do not matter, set the distance make-up time = 0.

By default, the superimposed motion profile that is automatically generated by the follower ramp function (with non-zero makeup time) is trapezoidal using the **Follower Ramp Acceleration** and a distance derived from the active **Ramp Makeup Time**.

The value of the **Velocity Limit** may affect functionality differently depending on the relationships of the master source velocity. The following case examples illustrate these points.

Case 1: The master source velocity is less than 80% of the configured **Velocity Limit** and the makeup time (Mkup Time) is a long enough interval so that the resultant velocity remains less than 80% of the Vlim. This is the preferred operation; no errors are reported, and the over speed move of the ramp function occurs within the specified makeup time. The follower axis velocity will not exceed 80% of the Vlim unless the master source velocity increases.

Case 2: The master source velocity is below 80% of the configured **Velocity Limit** but the makeup time interval is too short to allow operation as in case 1. A status only error (ECh) will be returned when the follower velocity matches the master command velocity. The makeup move will accelerate using the active **Follower Ramp Acceleration** to 80% of the velocity limit (Vlim). The makeup move will occur, and all accumulated counts stored during initial acceleration will be used.

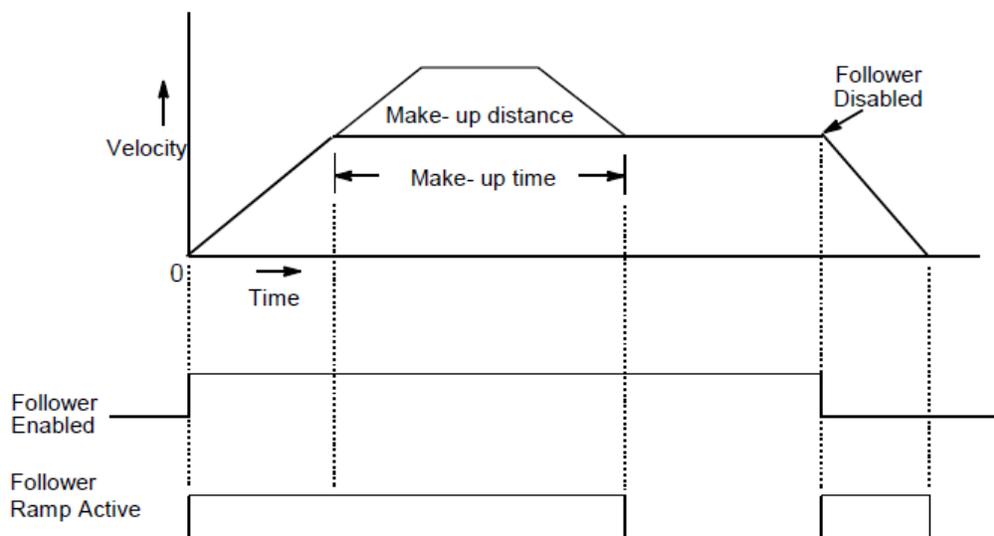
Case 3: The master source velocity is greater than 80% of the configured **Velocity Limit** when the follower velocity matches the master command velocity. A status only error (EAh) is returned and no makeup correction move is attempted.

Case 4: At the time when the follower velocity matches the master command velocity and the makeup move is to occur and conditions are the same as in Case 1 or Case 2 and the makeup move has initiated, the master source increases to >80% of the **Velocity Limit**. The amount of accumulated counts and the active makeup time value will determine if the makeup move will complete in the specified makeup time. A status only error (F2h) will occur if the combined master command velocity and the makeup move velocity reach 100% of the velocity limit. The master command velocity will not exceed 100% of the **Velocity Limit** value. Accumulated counts may be lost and the makeup move will not complete.

The Follower Ramp Active %I bit indication is turned on while the ramp control is in effect for both the ramp up/make-up and ramp down.

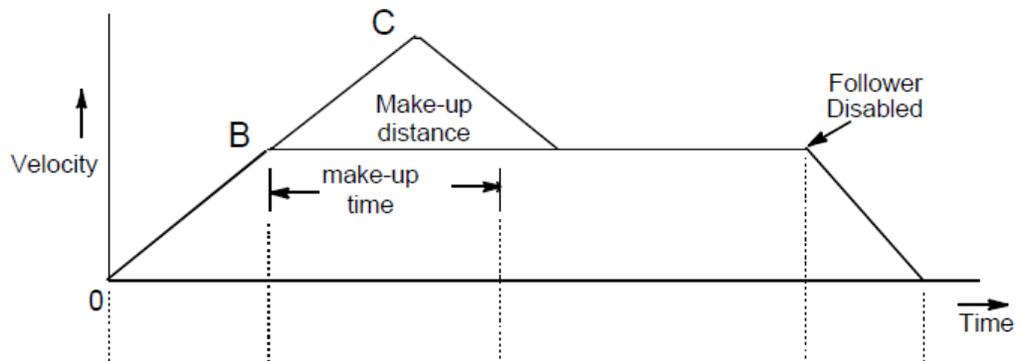
The Follower Enabled and Follower Ramp Active %I bits can be monitored by the host controller to determine which part of the follower ramp up/ramp down cycle is active. The following figure shows the state of Follower Enabled and Follower Ramp Active during a follower cycle.

Figure 101: Follower Ramp Up/Ramp Down Cycle (Case 2)



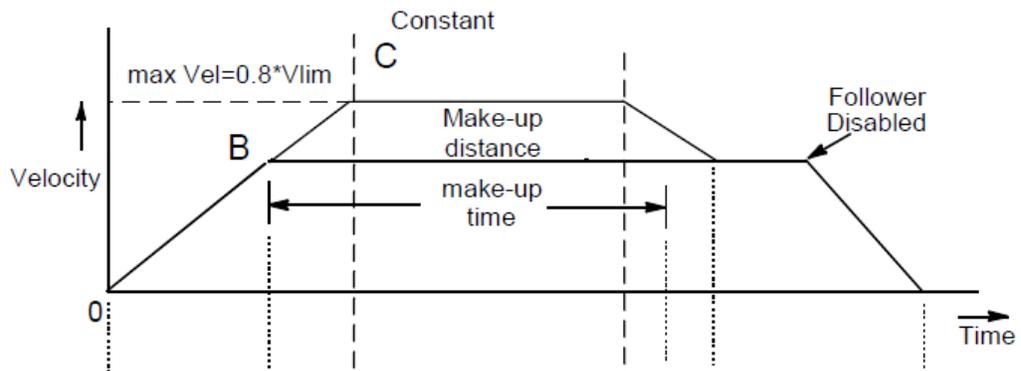
The programmed make-up time can be too short for the required distance correction. In this case a warning error is reported (in the point B of the trajectory), but system continues acceleration up to the speed, insuring the minimum possible distance correction time. The velocity profile for such case is shown on the figure 106.

Figure 102: Follower Ramp Up/Ramp Down Cycle – Case 2 with make-up time too small.



During the ramp phase of the distance correction, the velocity limit is controlled. If calculated velocity is too high, then the velocity is clamped, and warning error code is set (in the point C of the trajectory). Figure 107 shows the velocity profile during the follower ramp cycle for this case.

Figure 103: Follower Ramp Up/Ramp Down Cycle - case with active velocity limit.



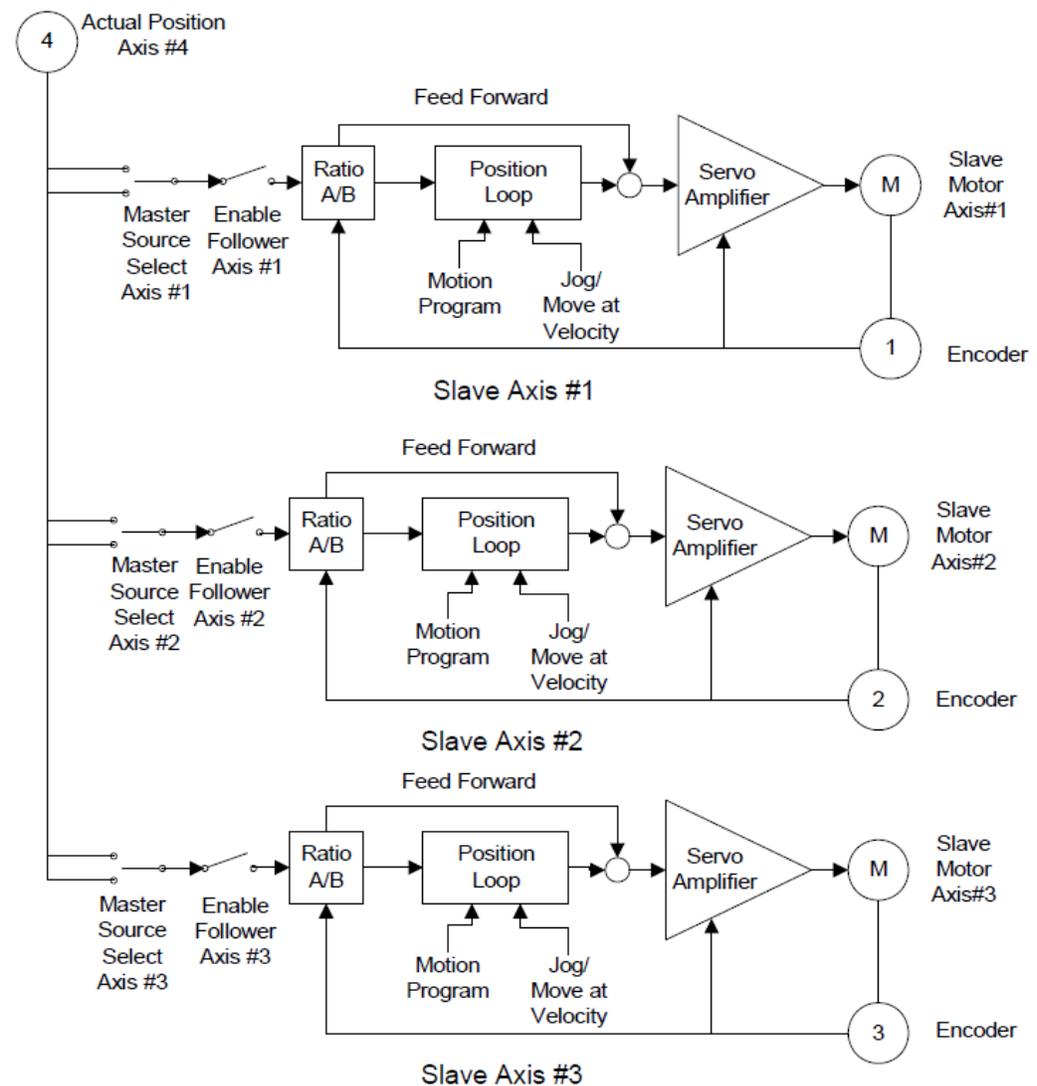
If the acceleration time (sector BC of the trajectory in figure 107) exceeds 128 seconds, then another warning error will be reported. In this case the distance also will be corrected accurately.

8.10.1 Follower Mode Command Source and Connection Options

The diagrams on the following pages illustrate a variety of Master axis and Follower slave axis loop connection options.

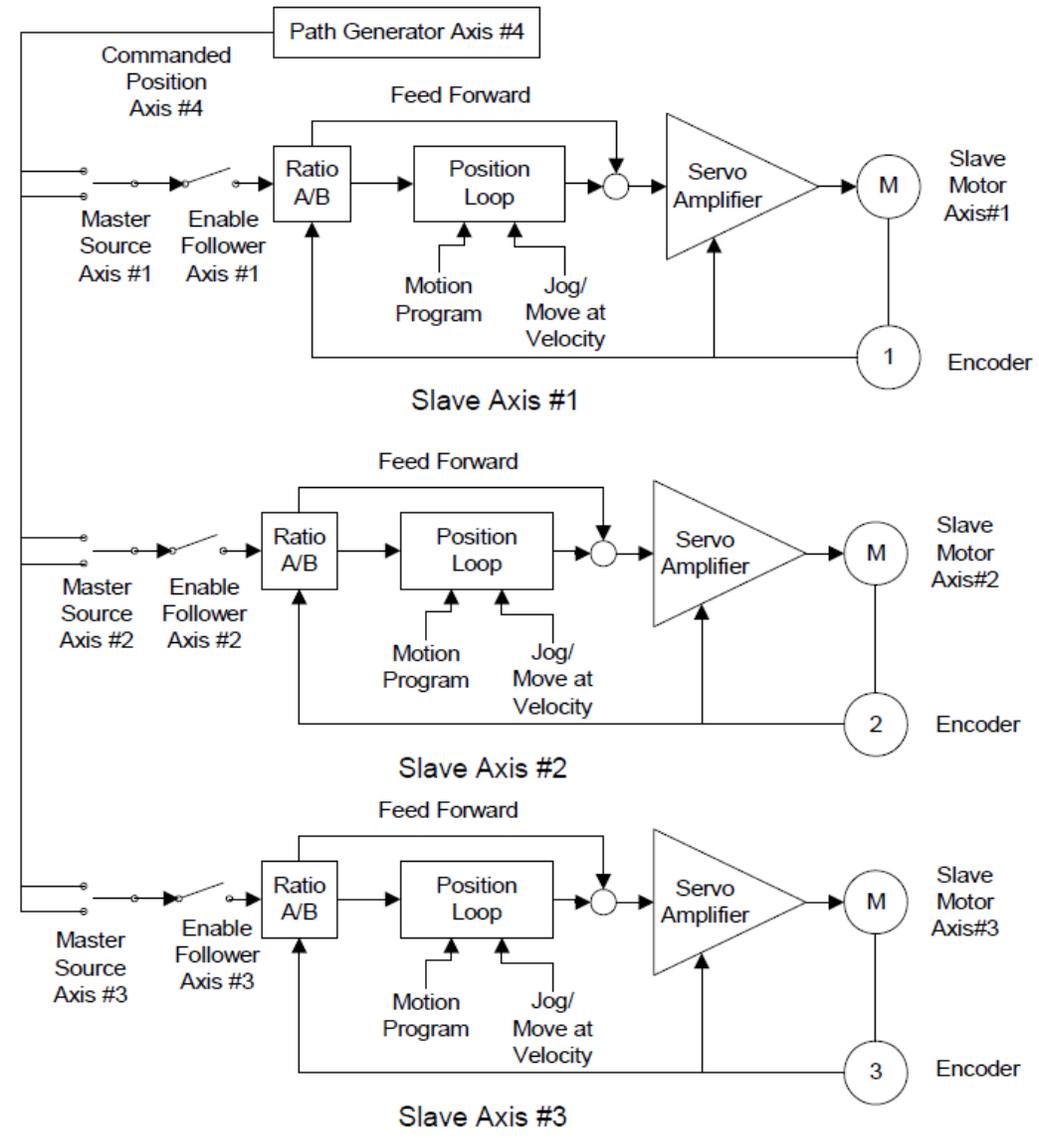
The diagram below illustrates the three DSM314 analog axes connected in parallel with Actual Position for Axis #4. The reader should note that with this configuration, the Local Logic function can be run. This is because the command generator for axis #4 is not required for this configuration. The Master Source Configuration items are all set to Actual Position Axis #4. This is not a requirement. However, it does eliminate a source of error due to the master source select bit being set incorrectly.

Figure 104: 3-Axis Analog Follower / Parallel Structure / Follower Source = Actual Position 4



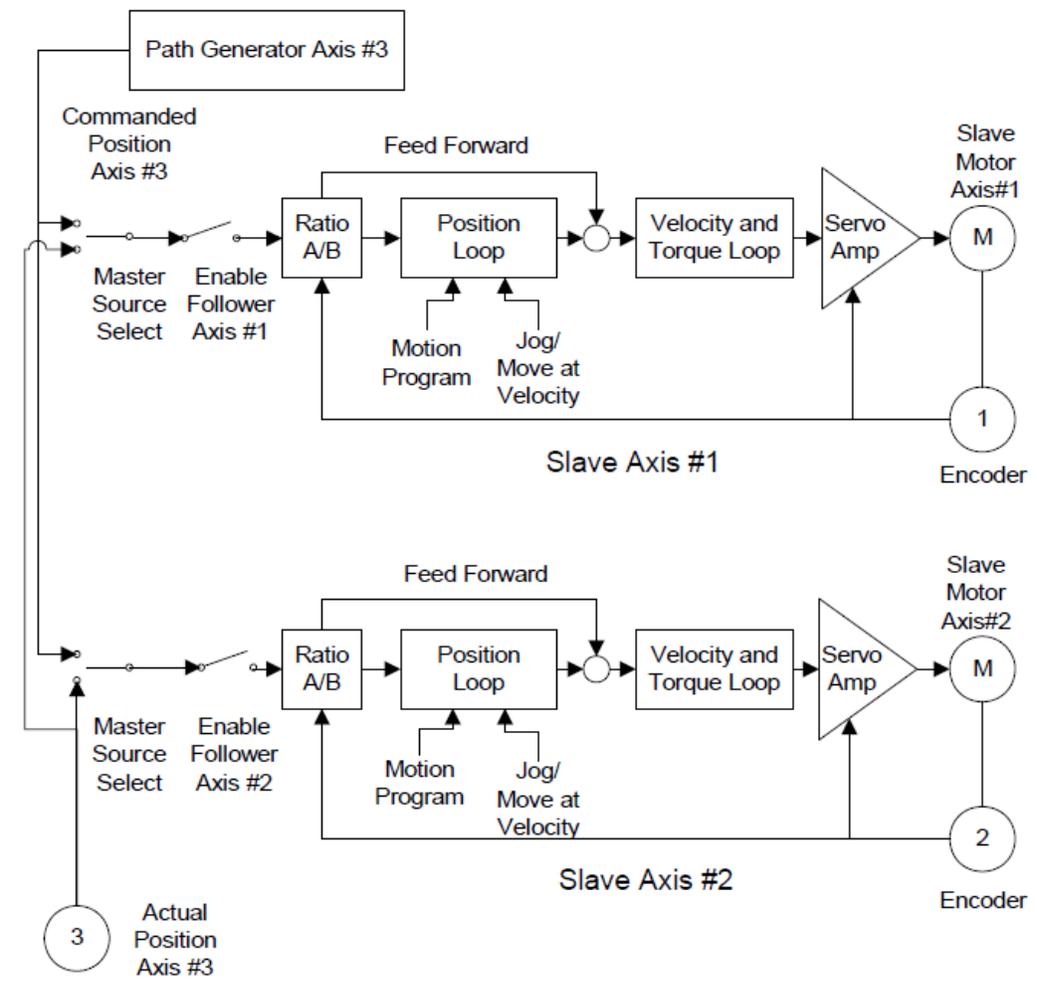
The diagram below illustrates the three DSM314 analog axes connected in parallel with Commanded Position for Axis #4. The reader should note that with this configuration, the Local Logic function cannot be run. This is because the command generator for axis #4 is required for this configuration. The Master Source Configuration items are all set to Commanded Position Axis #4. This is not a requirement. However, it does eliminate a source of error due to the master source select bit being set incorrectly.

Figure 105: 3-Axis Analog Follower / Parallel Structure / Source = Commanded Position 4



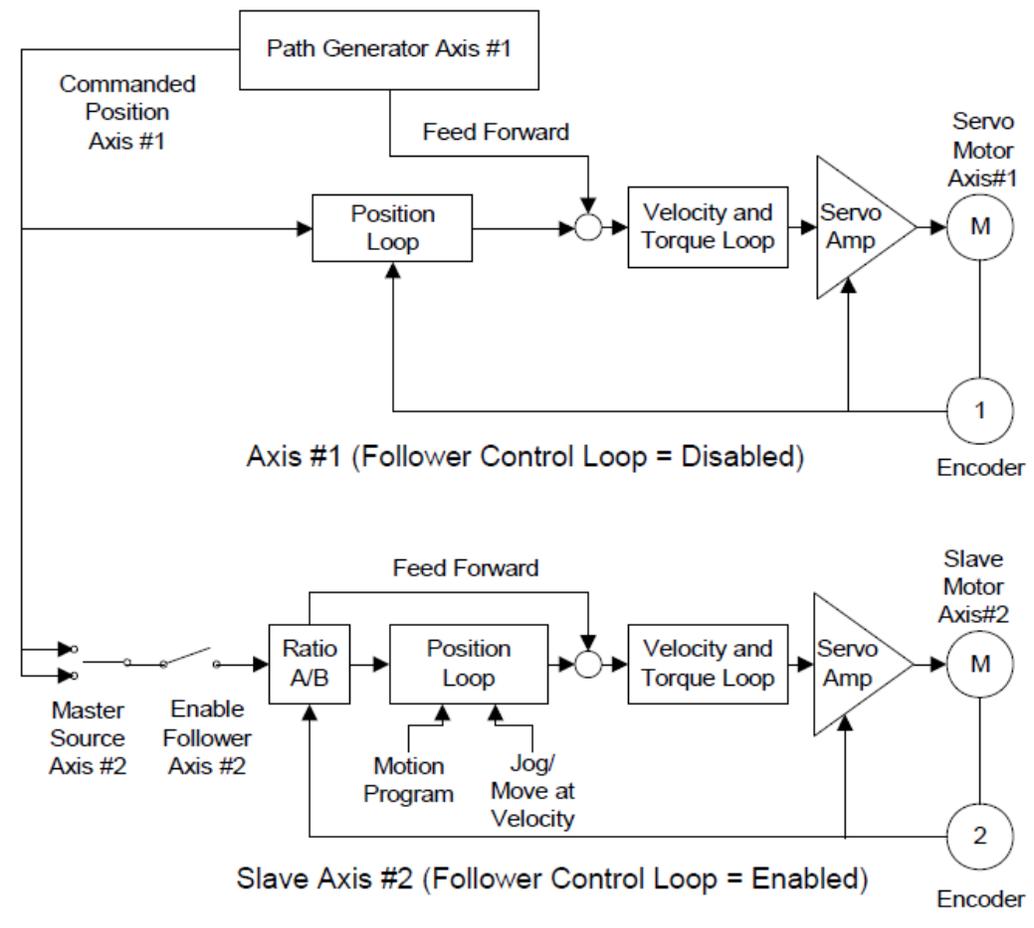
The diagram below illustrates the two DSM314 digital axes connected in parallel with Commanded Position or Actual Position for Axis #3. The reader should note that with this configuration the Local Logic function can be run. This is because the command generator for axis #4 is not required for this configuration.

Figure 106: 2-Axis Digital Follower / Parallel Structure / Source = Commanded or Actual Position 3



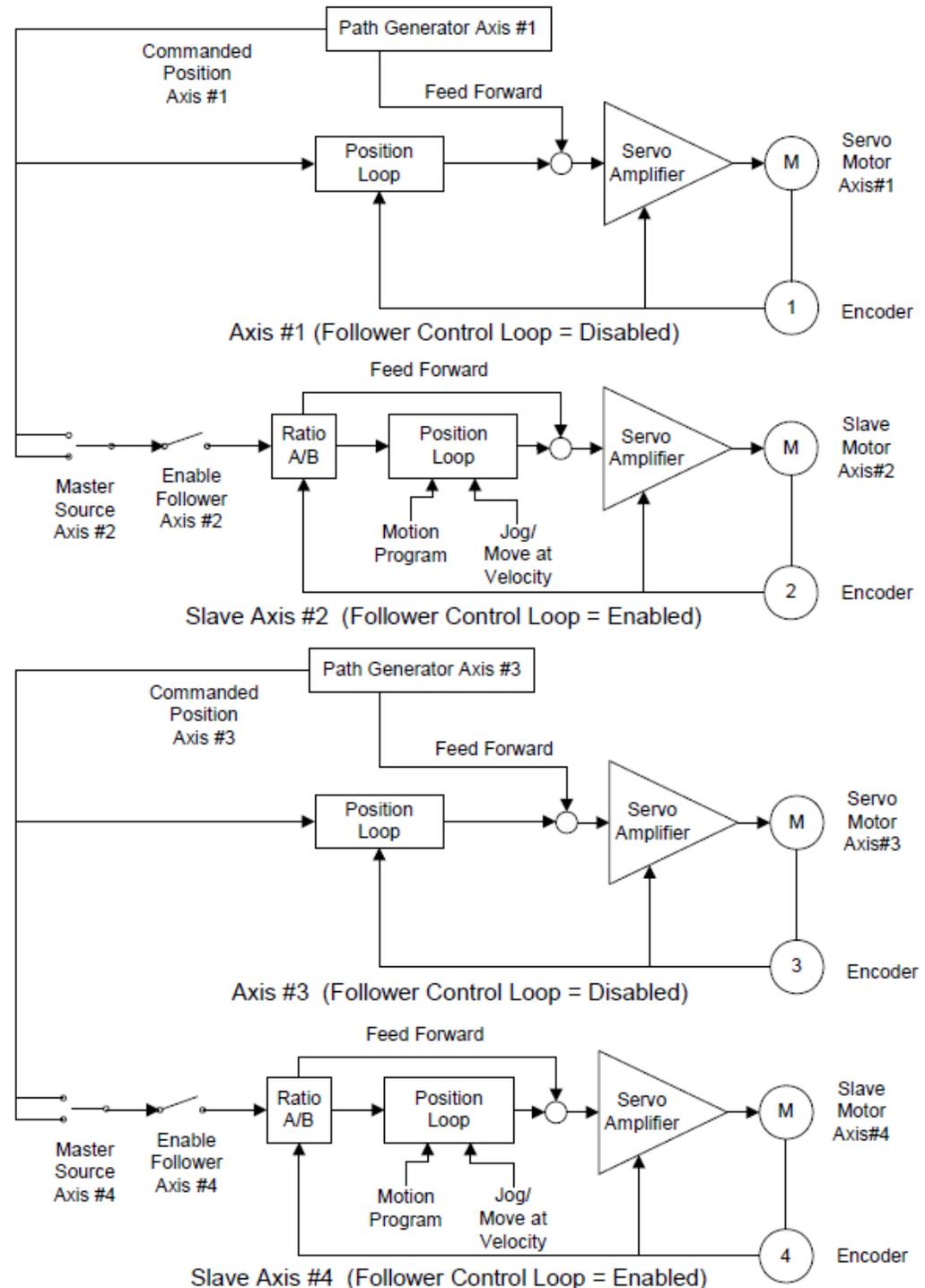
The diagram below illustrates two DSM314 digital axes connected in parallel with Commanded Position from Axis 1 driving servo loops for Axis 1 and Axis 2. This will allow both axes to run from the same commanded path. Note that Axis 1 is configured with Follower Control Loop = Disabled. This configuration does not allow for load sharing between axes that are tightly coupled. The reader should note that with this configuration the Local Logic function can be run. This is because the command generator for axis #4 is not required for this configuration.

Figure 107: 2-Axis Digital Follower / Parallel Structure / Source = Commanded Position 1



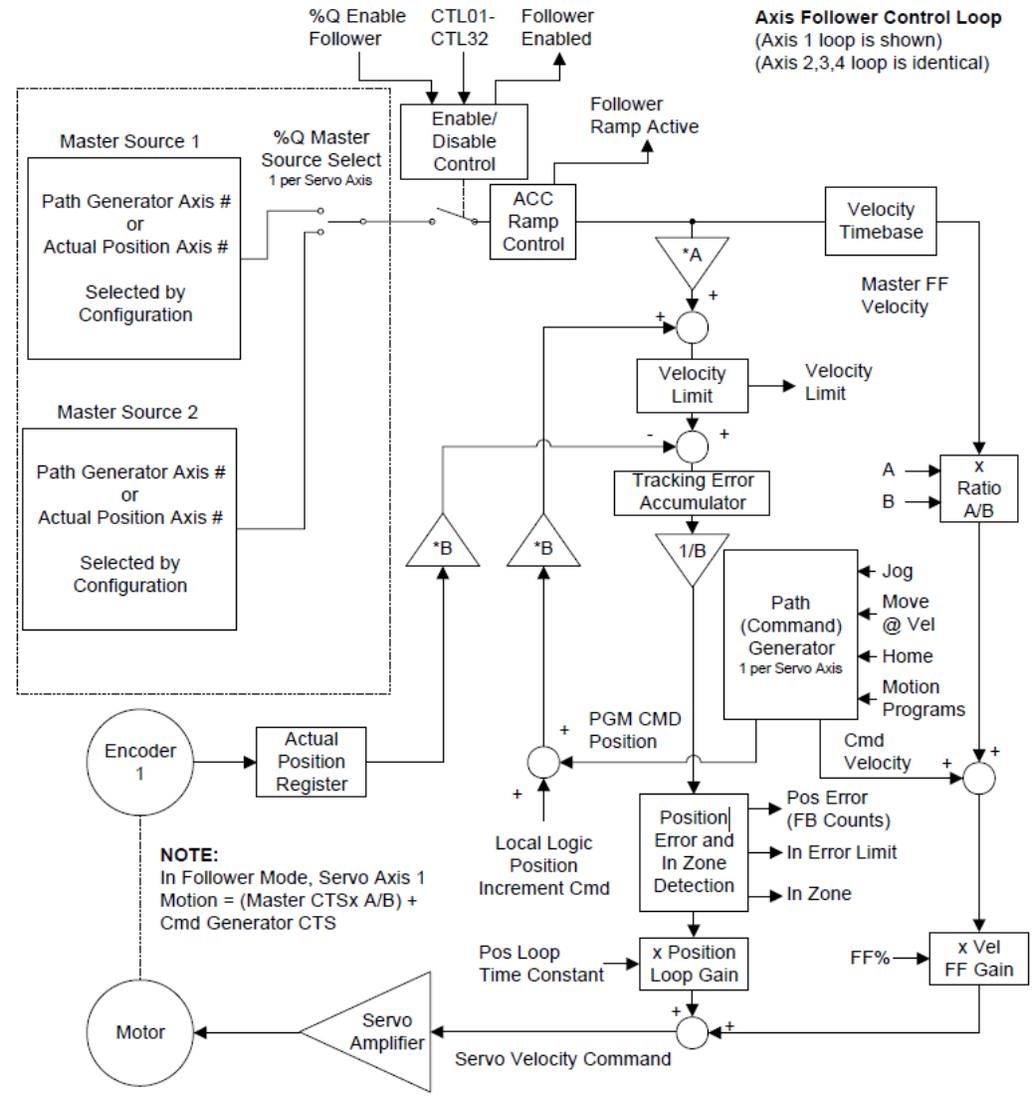
The diagram below illustrates the four DSM314 analog axes connected in two parallel pairs. The reader should note that with this configuration the Local Logic function cannot be run. This is because the servo position loop for axis #4 is required for this configuration.

Figure 108: Four-Axis Analog Follower / Parallel Structure / Src = Cmd Pos 1 & Cmd Pos 3



Follower Control Loop Block Diagram

Figure 109: Follower Axis Control Loop Block Diagram



Chapter 9: Combined Follower and Commanded Motion

Combined motion consists of Follower motion commanded from a master axis combined with one of these internally commanded motions:

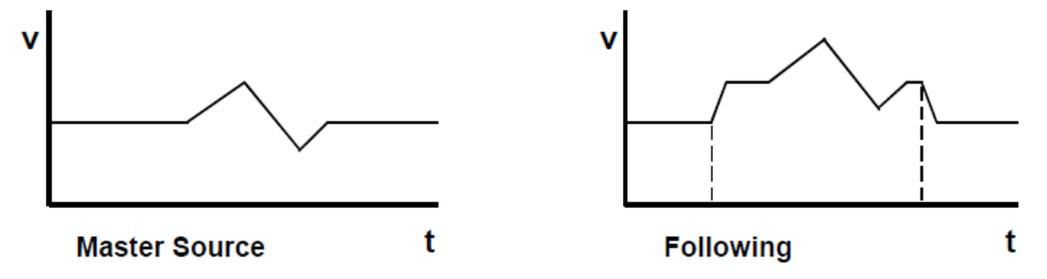
- Jog Plus/Minus %Q Command
- Move at Velocity %AQ Command
- Move %AQ Command
- Stored Motion Program

Combined motions are additive. The slave axis motion is equal to the sum of the motion commanded by the master axis and the internally commanded motion.

9.1 Example 1: Follower Motion Combined with Jog

In this example, the Enable Follower %Q bit is set, causing the slave axis to follow the master input. While the slave axis is following, the Jog Plus %Q bit is set. The following axis accelerates from its master's velocity to its master's velocity added to the current Jog Velocity. This acceleration will be just as if the axis was not following a master source at the time. When the Jog Plus %Q bit is cleared, the following axis decelerates to its master's velocity. In the velocity profiles below, the dotted lines indicate when the Jog Plus %Q bit is turned ON and then OFF.

Figure 110: Combined Motion (Follower + Jog)



9.2 Follower Motion Combined with Motion Programs

Motion commands from stored programs or the Move %AQ command can also be combined with the master command to drive the follower axis. These point-to-point move commands can come from one of the stored motion programs 1 through 10 and any stored subroutines they may call. The Move %AQ command is treated as a single line motion program, which uses the present **Jog Velocity** and **Jog Acceleration**. Program execution is started by the host controller setting an Execute Program n %Q bit or sending a Move %AQ command.

If there is no master command, the axis can be commanded solely from the stored motion program data. Thus, with no master input to Servo Axis 2 and Commanded Position 2 selected as the master source for Servo Axis 1, a stored program can be used to control Servo Axis 2 with Servo Axis 1 following per the designated ratio.

When PMOVES are executed with Follower not enabled, the In Zone %I bit must be set at the end of the move before programmed motion will continue. When Follower is enabled, since In Zone may not turn on while also following a master command, the In Zone indication will not be required to continue. The next Move will take place when the commanded distance for the previous move has completed. The In Zone %I bit will always indicate the true in zone condition.

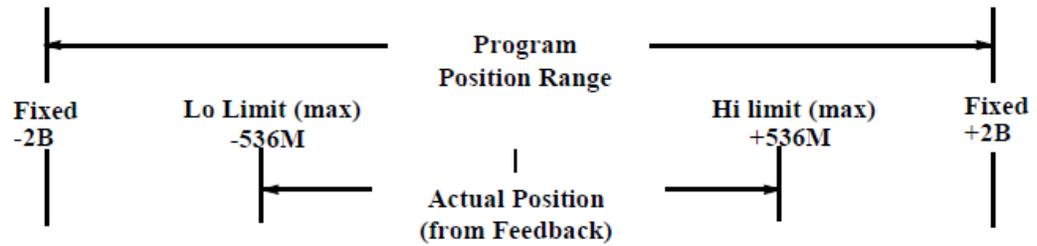
The active commanded position updated and used by the stored motion program is referred to as Program Command Position. Each time a program is selected for execution, this position register is initialized in one of the two ways listed below.

1. If the follower is **not enabled**, the Program Command Position is set to the current Commanded Position = Actual Position + Position Error.
2. If the follower is **enabled**, the Program Command Position is set to the Program Reference Position (0). Since the Program Command Position is only updated by internally generated commands (and not by the master command), it will then indicate the position commanded by the stored program. Absolute move commands from the stored program will be referenced to the Program Reference Position.

Therefore when an absolute move is the first move in a program, it will behave like an incremental move when the follower is enabled. Additional absolute moves within a program will be referenced to the current Program Command Position, which is updated by each move. Once a motion program finishes, executing another program with follower enabled will again cause the Program Command Position to be initialized to zero.

Position ranges (in counts) for the Actual and Program Command Position registers are indicated in the figure below.

Figure 111: Combined Motion (Follower + Jog)



With sustained commanded motion in the same direction, the Program Command Position will roll over at +2,147,483,647 or -2,147,483,648 counts.

The Actual Position, however, will be confined by the configured High Position Limit and Low Position Limit.

Table 51 below indicates which source commands affect these position registers and the actual and commanded velocities. Program Command Position is updated only by internally generated move commands (program commands, Jog Plus Minus, Find Home, and Move at Velocity). The Commanded Velocity (returned in %AI data) also only indicates velocity commanded by these internally generated move commands. Actual Position and Actual Velocity %AI return data reflect the combination of the master input and the move commands. In other words, counts coming from the master source affect only the Actual Position and Actual Velocity. If there are no internally generated move commands, the Commanded Velocity will be 0 and the Program Command Position will not change.

Table 51: Command Input Effect on Position Registers

COMMAND Input	Follower Enabled	Follower Registers Affected by input
Master Commands (from selected Master source)	No	None affected
	Yes	Actual Position %AI status word is updated Commanded Position %AI status word is updated (Actual Position + Position Error) Program Command Position is Not affected Actual Velocity %AI status word is updated Commanded Velocity %AI status word is Not affected
Program Commands	No	Actual Position %AI status word is updated Commanded Position %AI status word is updated (Actual Position + Position Error) Program Command Position is updated Actual Velocity %AI status word is updated Commanded Velocity %AI status word is updated (by Program commanded velocity only)
	Yes	Actual Position %AI status word is updated (by Program command + Master command) Commanded Position %AI status word is updated

COMMAND Input	Follower Enabled	Follower Registers Affected by input
		<p>(Actual Position + Position Error)</p> <p>Program Command Position is updated (by Program command only)</p> <p>Actual Velocity %AI status word is updated (by Program command velocity + Master command velocity)</p> <p>Commanded Velocity %AI status word is Updated (by Program command velocity only)</p>
Other Internally Generated Move Commands (Home, Jog, and Move at Velocity)	No	<p>Actual Position %AI status word is updated</p> <p>Commanded Position %AI status word is updated (Actual Position + Position Error)</p> <p>Program Command Position is updated but not used</p> <p>Actual Velocity %AI status word is updated.</p> <p>Commanded Velocity %AI status word is updated (by Internal command velocity only)</p>
	Yes (Find Home is not allowed)	<p>Actual Position %AI status word is updated (by Internal command + Master command)</p> <p>Commanded Position %AI status word is updated (Actual Position + Position Error)</p> <p>Program Command Position is updated but not used</p> <p>Actual Velocity %A status word is updated (by Internal command velocity + Master command velocity)</p> <p>Commanded Velocity %AI status word is updated (by Internal command velocity only)</p>

The Program Command Position can be synchronized to the Actual Position %AI value in three ways:

- Find Home %Q command execution
- Set Position %AQ command
- Execute Motion Program n %Q command (if the follower is not enabled)

The effect of these commands is indicated in Table 52 below.

Table 52: Actions Affecting Program Command Position

ACTION	Follower Enabled	Resulting Updates to Follower Position Registers
Home Found	No	Actual Position %AI status word is set to Home Value Program Command Position is set to Actual Position + Position Error
	Yes	Find Home %Q command is Not allowed Status Error is returned
Set Position %AQ Command	Not applicable	Actual Position %AI status word is set to %AQ Value Program Command Position is set to Actual Position + Position Error
		<i>Note: Set Position is not allowed if the Moving %I bit is ON.</i>
Execute Program	No	Actual Position %AI status word is NOT affected Program Command Position is set to Actual Position + Position Error
	Yes	Actual Position %AI status word is NOT affected Program Command Position is set to Reference Position (0)

Program moves will execute in a continuous fashion such that incremental PMOVE or CMOVE commands past the limits will roll over at the limit and continue. Absolute PMOVE or CMOVE commands can also be used for applications that do not require going beyond the high/low count limits.

Any internally generated move command can be immediately terminated by the Abort All Moves %Q command.

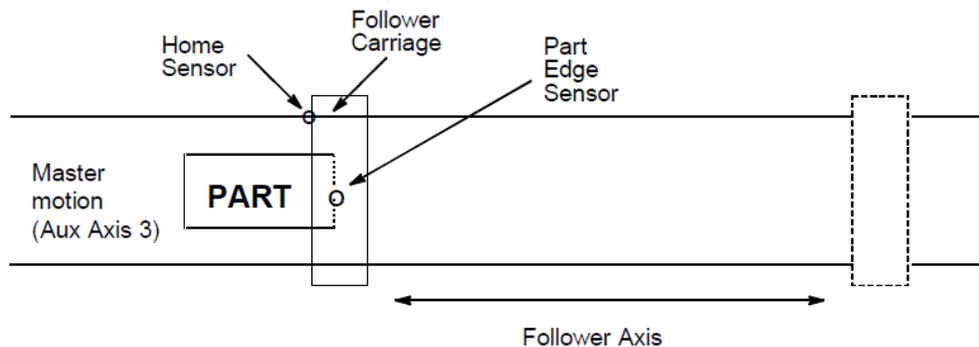
The User Selected Data %AI status word can be changed to report the Program Command Position by using the Select Return Data %AQ command. Refer to Chapter 5 for details.

The following application example illustrates how a stored program can be used to control positioning operations relative to the detected edge of a moving object as it moves at a rate detected by the master axis (Aux Axis 3) encoder input.

9.3 Example 2: Follower Motion Combined with Motion Program

Applications that require modifying parts on the fly (such as notching, marking, riveting, spot welding, spot gluing, and so forth) would make use of the point-to-point moves superimposed on follower motion and enable follower at input features. A typical configuration and control sequence required for these applications is shown below.

Figure 112



Control Sequence

1. With Enable Follower %Q bit OFF, the host controller commands Follower axis to home position where Actual Position & Program Command Position are synchronized and set to Home Position value. Position Valid %I bit indicates when this step is complete.
2. The host controller sets the Enable Follower %Q bit command.

Note: The CTL01- CTL24 bit to which the part edge sensor is connected would already have been configured in the Follower Enable Trigger configuration parameter.

3. When the Part edge sensor trips, the DSM314 enables the Follower axis to start following the master (Aux Axis 3) encoder inputs. The Follower Enabled %I bit indicates when the axis is following the master command. Note that the Accel Ramp and Make-Up Time feature could be used to allow the follower axis to catch up to the master axis if required.
4. Once the follower is enabled, the host controller sends the Execute Motion Program n %Q bit to start execution of the selected program for the follower axis. At the time the program is selected, Program Command Position will be set to program reference position (0) because the follower is enabled. Program execution is then relative to the moving part edge as the follower axis tracks the part. Program Command Position now contains the position of the follower axis relative to the part edge and Actual Position indicates the total distance the follower axis has moved from the Home point (master +/- program commands).
5. At the end of program, the host controller turns Enable Follower %Q bit OFF and loops back to step 1 to repeat for next part.

Note:

Since the DSM314 saved the Follower enable input trigger Commanded Position in a parameter register (#226 for axis 1, #234 for axis 2), step 1 this time could be used to execute another program with an absolute move command back to the parameter value position and continuing with step 2. In this case, the Moving and In Zone %I bit indications could be used to indicate when step 1 is complete.

This method is possible because the Program Command Position is set to the Actual Position + Position Error when Execute Motion Program is commanded with the follower disabled.

Chapter 10: Introduction to Local Logic Programming

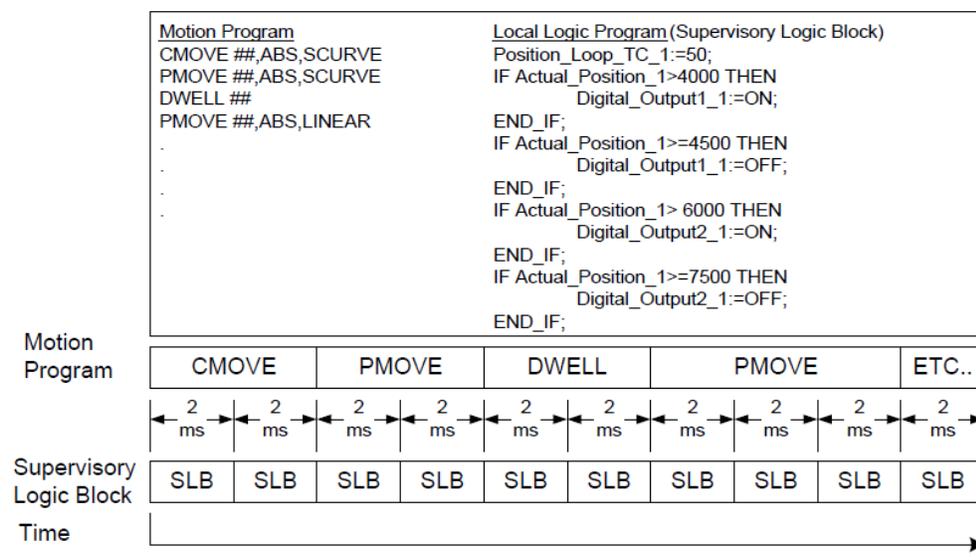
This chapter contains an introduction to the basic local logic programming concepts. The DSM and the DSM motion programming language are not discussed in detail in this chapter. These concepts are discussed in other chapters within this manual.

10.1 Local Logic Programming

The local logic program works in conjunction with the host controller logic program and motion program to yield a flexible programming environment. Specifically, local logic programs provide the user with the ability to perform math and logic that is deterministic and synchronized with the DSM Position Loop execution rate. This ability is critical to many applications where the accuracy and/or speed require this tight synchronization.

The DSM local logic function provides the user the ability to execute basic logic and mathematical functions within the DSM module. Additionally, local logic permits fast read/write access to local DSM digital and analog I/O. Consult Chapters 13 and 14 for a complete listing of available I/O. The local logic program execution method guarantees the local logic program runs at the position loop sample rate and completes each sample period. Note: If the module is unable to complete the local logic program execution within the allotted time the module generates an error message. Chapter 13 and Appendix E contain more information concerning program execution times. Additionally, the local logic program runs in parallel with normal DSM motion programs. The parallel program execution allows the local logic program to supervise the motion program. Thus, local logic programs are also called supervisory logic blocks (SLB). The local logic program execution versus motion program execution is shown in Figure 113.

Figure 113: Local Logic Versus Motion Program Execution



It is important to understand the concept shown in Figure 113. before writing local logic programs. The local logic program runs to completion each position loop sample period. The program then re-executes the complete local logic program the next position loop sample period. This execution method differs from the motion program execution method. The motion programs execute each command to completion in a sequential fashion, without any time guarantees. This concept is illustrated in Table 53. , which lists the first four local logic execution periods for the local logic and motion programs shown in Figure 113. In the example, note that the local logic program executes to completion each position loop sample period. The motion program statements execute until the controlled motion achieves the desired result. For additional details concerning motion program statement execution, consult chapter 7.

Table 53: Local Logic – Motion Program Execution Example

Position Loop Sample Number	Active Motion Program Statement	Local Logic Program Statements
n	CMOVE ##,ABS,S-CURVE	Position_Loop_TC_1:=50;
		IF Actual_Position_1>4000 THEN
		Digital_Output1_1:=ON;
		END_IF;
		IF Actual_Position_1>=4500 THEN
		Digital_Output1_1:=OFF;
		END_IF;
		IF Actual_Position_1> 6000 THEN
		Digital_Output3_1:=ON;
		END_IF;
		IF Actual_Position_1>=7500 THEN
		Digital_Output3_1:=OFF;
		END_IF;
n+1	CMOVE ##,ABS,SCURVE	Position_Loop_TC_1:=50;
		IF Actual_Position_1>4000 THEN
		Digital_Output1_1:=ON;
		END_IF;
		IF Actual_Position_1>=4500 THEN
		Digital_Output1_1:=OFF;
		END_IF;
		IF Actual_Position_1> 6000 THEN
		Digital_Output3_1:=ON;
		END_IF;
		IF Actual_Position_1>=7500 THEN
		Digital_Output3_1:=OFF;

Position Loop Sample Number	Active Motion Program Statement	Local Logic Program Statements
		END_IF;
n+2	CMOVE ##,ABS,SCURVE	Position_Loop_TC_1:=50;
		IF Actual_Position_1>4000 THEN
		Digital_Output1_1:=ON;
		END_IF;
		IF Actual_Position_1>=4500 THEN
		Digital_Output1_1:=OFF;
		END_IF;
		IF Actual_Position_1> 6000 THEN
		Digital_Output3_1:=ON;
		END_IF;
		IF Actual_Position_1>=7500 THEN
		Digital_Output3_1:=OFF;
		END_IF;
n+3	CMOVE ##,ABS,SCURVE	Position_Loop_TC_1:=50;
		IF Actual_Position_1>4000 THEN
		Digital_Output1_1:=ON;
		END_IF;
		IF Actual_Position_1>=4500 THEN
		Digital_Output1_1:=OFF;
		END_IF;
		IF Actual_Position_1> 6000 THEN
		Digital_Output3_1:=ON;
		END_IF;
		IF Actual_Position_1>=7500 THEN
		Digital_Output3_1:=OFF;
		END_IF;

10.2 When to Use Local Logic Versus Ladder Logic

The local logic programming language contains basic mathematical and logical constructs. The capabilities are not designed to replace the host controller's logic capabilities. Instead, local logic is designed to complement the host controller's logic and mathematical abilities. Specifically, local logic is designed to solve a small logic and mathematical set that requires tight synchronization with the controlled motion. The local logic program must run to completion each sample period. Thus, local logic programs are limited in size. The default local logic program size limit is 150 lines. The Local Logic build process will generate an error message when the 150-line limit is exceeded. A warning message is generated when 100 lines are exceeded. If the program is very large and computationally intensive it may exceed the allowed execution time and result in a watchdog timer warning/error (refer to Appendix E). In contrast, the host controller's program size is limited only by available memory. However, as host controller program sizes increase, the host controller sweep times increase. (For additional information concerning sweep times, please consult the PACSystems CPU Reference Manual, GFK-2222 or the Series 90-30/20/Micro PLC CPU Instruction Set Reference Manual, GFK-0467.) This is not true with local logic programs. Local Logic programs always execute to completion every position loop sample period. When using host controller logic, the added latency associated with the host controller sweep times for time-critical logic operations that are tightly coupled to motion can be unacceptable or limit process performance. These tightly coupled and time-critical processes are potential Local Logic applications. Each process will have to be evaluated on an individual basis to determine which sections to write in host controller logic and which sections to write in Local Logic.

10.3 Getting Started with Local Logic and Motion Programming

The sections that follow provide information on getting started with the Local Logic Editor and Motion program editors. The sections concentrate on program usage with an emphasis on program creation, syntax check, and program download.

10.3.1 Requirements

The Local Logic and Motion Program editors are integrated within the programming software environment. You need one of the following software packages. Please refer to the software documentation for installation instructions.

- Machine Edition Logic Developer – PLC version 2.1 or later
- VersaPro version 1.1 or later (Series 90-30 only. For details, refer to Appendix H.)

The DSM314 feature set also requires:

- PACSystems firmware release 2.8 or later, or
- 90-30 CPU firmware release 10.0 or later.

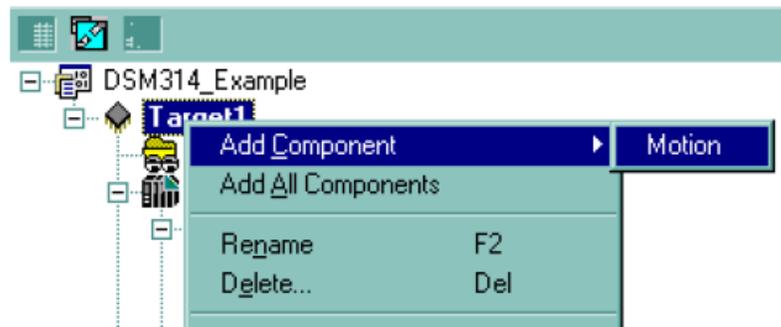
10.3.2 Creating a Local Logic Program

The Local Logic editor is integrated into the programming software environment. The editor allows you to easily create, edit, store, and download a Local Logic program. You create a Local Logic program in a VersaPro folder or a Machine Edition project. Refer to the software documentation for details on how to create or open a project.

For details on getting started with Machine Edition, refer to “Machine Edition Configuration” in chapter 2. For details on using VersaPro, refer to Appendix H.

1. To create a local logic program, open your project in Machine Edition.
2. In the Project tab of the Navigator window, right click the Target containing the DSM314, choose Add Component, and then choose Motion.

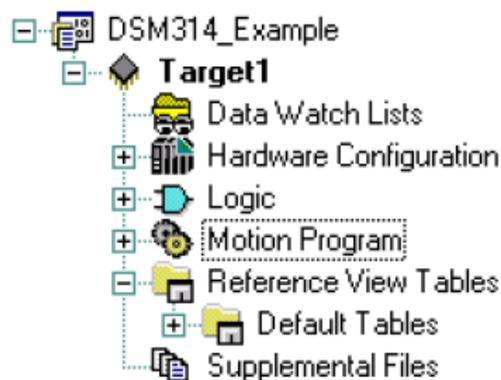
Figure 114



The Motion Program folder appears in the Navigator.

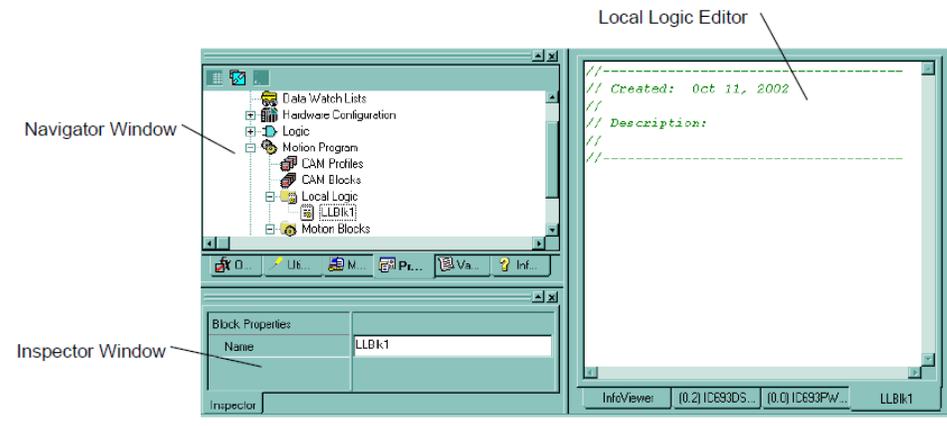
3. Expand the Motion Program folder. Select Local Logic and choose New. A local logic block is created in the Local Logic folder and the local logic editor opens.

Figure 115



- To change the name of your local logic block, edit the name in the Block Properties, which is displayed in the Inspector window.

Figure 116: Local Logic Editor Main Screen Layout, Machine Edition

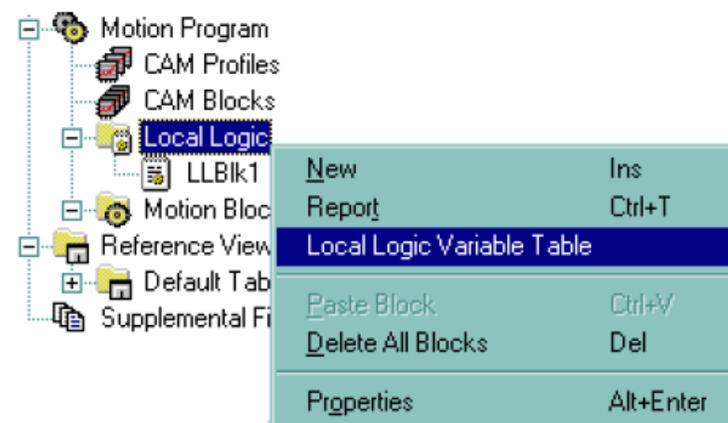


10.4 Local Logic Variable Table

The programming environment includes a window that contains the Local Logic variables. The Local Logic Variable table (LLVT) allows you to drag and drop or cut and paste the text from the table into a program. (Reference Figure 123.).

- To open the LLVT in Machine Edition, right click the Local Logic folder in the Navigator and choose Local Logic Variable Table.

Figure 117



- To open the LLVT in VersaPro, select Local Logic Variable Table from the View menu, press Alt + 6, or click the Toggle Local Logic Variable Table button  on the toolbar.

The table has several tabs that group the variables by category. The categories are:

- Axis 1 – Variables specific to axis number one
- Axis 2 – Variables specific to axis number two
- Axis 3 – Variables specific to axis number three
- Axis 4 – Variables specific to axis number four
- Global – Global data such as Module Status Code
- CTL bits – DSM general purpose control/status bits
- Parameter Registers - DSM Parameter data

Figure 118: Local Logic Variable View Table

	Name	Type	Group	Description	R	W
DSM 3 1 4	Actual_Position_1	32 Bits	Status Variables	Actual_Position (user units) is a value maintained by the DSM to represent the phy	✓	
	Actual_Velocity_1	32 Bits	Status Variables	Actual_Velocity (user units/sec) represents the axis velocity derived from the posit	✓	
	Analog_Input1_1	Signed 16 Bits	FacePlate I/O	The Analog_Input1 variable reports the input value for the first analog input of the a	✓	
	Analog_Input2_1	Signed 16 Bits	FacePlate I/O	The Analog_Input2 variable reports the input value for the second analog input of t	✓	
	Axis_Enabled_1	Bit	Status Variables	The Axis_Enabled status bit is ON when the DSM is ready to receive commands ar	✓	
	Block_1	Unsigned 16 Bits	Status Variables	Block is the present command block number reported by the motion program.	✓	
	Commanded_Position_1	32 Bits	Status Variables	Commanded_Position (user units) is the instantaneous axis position command. The	✓	
	Commanded_Torque_1	32 Bits	Status Variables	The Commanded_Torque variable reports the present digital servo torque comman	✓	
	Commanded_Velocity_1	32 Bits	Status Variables	Commanded_Velocity (user units/sec) is generated by the DSM axis command gen	✓	
	Digital_Output1_1	Bit	FacePlate I/O	The Digital_Output1 bit controls the axis faceplate digital OUT_1 signal. This bit can		✓
	Digital_Output3_1	Bit	FacePlate I/O	The Digital_Output3 bit controls the axis faceplate digital OUT_3 signal. This bit can		✓
	Drive_Enabled_1	Bit	Status Variables	The Drive_Enabled status bit indicates the state of the Enable Drive %Q bit and the	✓	
	Enable_Follower_1	Bit	Control Variables	When the Enable_Follower bit is set, motion commanded by the follower master will		✓

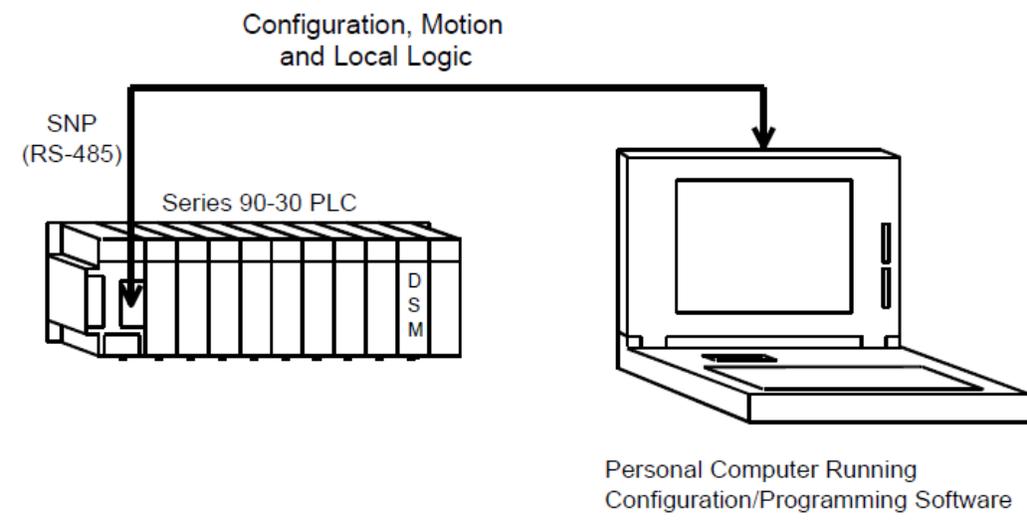
The table has six columns. The columns are as follows:

- **Name** – This column contains the variable name that is valid to be used within a local logic program
- **Type** – This is the data type for this variable. For example 32 Bits means that this variable is a 32 bit variable.
- **Group** – This is the group this variable is placed in. For example, FacePlate I/O means that this variable refers to a point on the module faceplate.
- **Description** – This column contains a textual description of the variable. If the user hovers the mouse pointer over the description a tool tip will be generated that allows the user too easily read the description.
- **R** – This column indicates if the variable can be Read by a Local Logic program
- **W** - This column indicates if the variable can be Written by a Local Logic program

10.5 Connecting the Local Logic Editor to the DSM

The configuration/programming software has several communications options. One communications option is to connect directly to the host controller SNP port, shown in Figure 119 below. Ethernet options are also available. All DSM314 programming is done through the software interface, yielding single point of programming for the module. (The DSM314 also has a serial port on the module faceplate, which is used only for updating the DSM314's firmware.) Local Logic and Motion programs are stored to a dedicated memory space inside the host controller CPU. The DSM314 then requests these programs by name from the CPU during configuration. The link to the programs the DSM314 requests from the CPU is contained in the Hardware Configuration for the host controller rack. The benefit is that programs are not module-specific but are rack/slot specific. Thus, if there is a need to swap DSM314s within a host controller, or to replace a DSM314, you need to perform the following three steps: (1) turn off power to the host controller, (2) change out the DSM314 modules, and (3) reapply power to the host controller. Upon powering up, the host controller will send the correct programs and configuration settings to the DSM314s.

Figure 119: Programmer Connection Diagram



10.6 Building a Local Logic Program

The programming software provides a self-contained environment that allows the user to perform all the actions necessary to create, edit, and download a local logic program to a DSM314 module.

10.6.1 Creating a Local Logic Program

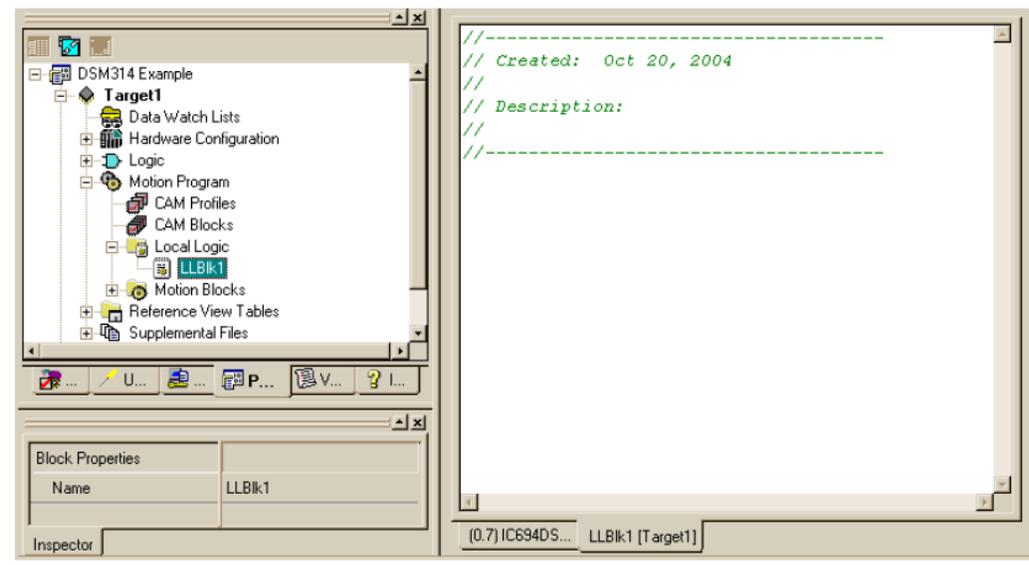
Create a Local Logic program named Example. For details on how to do this, see: Section

“Machine Edition Configuration”.

“Starting VersaPro”.

The resulting display is like the figure below.

Figure 120: Machine Edition New Local Logic Program



The Local Logic editor is a free-form text editor that allows you to enter programs in the style that you prefer. This example is a very simple Local Logic program that does not represent a fully functional application because it is intended for instructional purposes only. The example program is a simple timer application that relies on the digital servos position loop sample period (2 mSec) as a time base. See Chapter 1 for position loop sample periods for other configurations.

Sample Local Logic Program

```
(*****)
(* Program Name: LLExample *)
(* Description: The following example local logic program is *)
(* a program that creates a simple timer. The timer begins *)
(* after the first Local_Logic_Sweep has occurred. The *)
(* program makes uses the fact that Local Logic is run by *)
(* the DSM every 2 mSec. Thus, it counts local logic sweeps. *)
(* The program has three counters. The first counter is the *)
(* milliseconds counter while the other two counters are for *)
(* seconds and minutes. The seconds counter and minutes *)
(* counters roll over at 59. The program also sets CTL01 every *)
(* second. *)
(*****)
(* Variables *)
(* P001 = Milliseconds Counter *)
(* P003 = Seconds Counter *)
(* P004 = Minutes Counter *)
(* CTL01 = Seconds Signal *)
(* P100 = Used to check if 1 Sec has passed *)
(* P102 = Used to check if 1 Min has passed *)
(*****)
IF First_Local_Logic_Sweep THEN (* First execution sweep *)
    P001 := 0; (* Initialize P001 to 0 *)
    P003 := 0; (* Initialize P003 to 0 *)
    P004 := 0; (* Initialize P004 to 0 *)
END_IF;

P001:=P001+2; (* Time in Milliseconds *)

P100:= P001 MOD 1000; (* Check to see if 1 Sec (1000 mSec) Passed *)
IF P100 = 0 THEN (* Remainder of MOD Operation=0 1 Sec Passed *)
    P003:=P003+1; (* Time in Seconds *)
    CTL01 := 1;
    IF P003 = 60 THEN (* If Seconds = 60 then start over at 0 *)
        P003:=0;
    END_IF;
END_IF;
IF P100 <> 0 THEN
    CTL01 :=0; (* CTL01=0 When not incrementing sec counter *)
END_IF;

P101:=P001 MOD 60000; (* Check to see if 1 Min (60000mSec) Passed *)
IF P101 = 0 THEN (* Remainder of MOD Operation=0 1 Min Passed *)
    P004:=P004+1; (* Time in Minutes *)
    IF P004 = 60 THEN (* If Minutes = 60 then start over at 0 *)
        P004:=0;
    END_IF;
END_IF;
```

Once you type the above program into the text editor, the editor screen will look similar to Figure 121..

Figure 121: Local Logic (LLExample)

The screenshot shows the VersaPro software interface for a project named 'MotionTest'. The main window displays the ladder logic program for 'LLExample - LL'. The code is enclosed in comments and includes a description of the program's function: it creates a simple timer that begins after the first Local_Logic_Sweep. The program uses three counters: a milliseconds counter (P001), a seconds counter (P003), and a minutes counter (P004). The seconds and minutes counters roll over every 59 units. The program also sets a signal CTL01 every second.

Below the code, a table lists the variables used in the program:

Name	Type	Len	Address	Description	Stored Value	Scope	Ret	Ovr
Test	Bit	1	%G00001			Global	✓	
DSM_Time_Sec	Word	2	%A10061	DSM Time in Seconds		Global	✓	
DSM_ILLEnable	Bit	1	%Q00002	DSM #1LocalLogic Enab		Global	✓	
DSM_Time_Min	Word	2	%A10063	DSM Time in Minutes		Global	✓	
DSM_Time_MS	Word	2	%A10021	DSM Time in Milliseconds		Global	✓	
DSM_Time_Hrs	Word	2	%A10081	DSM Time in Hours		Global	✓	
DSM_Time_HalfSec	Word	2	%A10041	DSM Time Half Seconds		Global	✓	

The interface also shows a menu bar (File, Edit, View, Insert, Folder, PLC, Tools, Window, Help), a toolbar, and a status bar at the bottom indicating 'Run Enabled', 'Connected', and a cycle time of '4.6 msec Equal'.

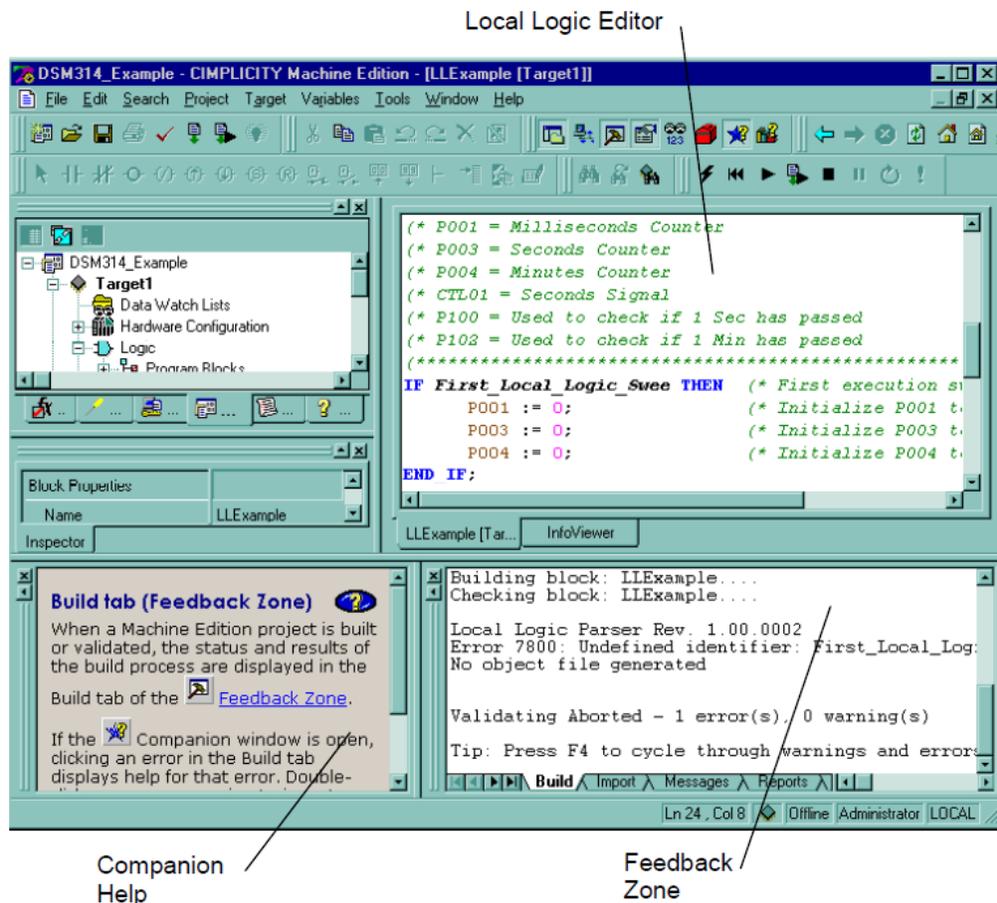
10.6.2 Checking Local Logic Syntax

At this point, you should validate the program to verify correct language syntax.

To check the language syntax, select Target, then Validate <Target Name>. You can also press F7 anywhere in the Machine Edition window. All logic blocks in the active target are checked. Results of syntax checking are displayed in the Feedback Zone. (If the Feedback Zone is not already open, starting the Validate process opens it.)

In the following example, the line “First_Local_Logic_Sweep” is incorrectly typed as “First_Local_Logic_Swee.”

Figure 122



Tip

To cycle through the warning and error messages in the Feedback Zone, press F4.

To go to the line that caused the error in the local logic program, double click the error description in the Feedback Zone. The focus shifts to the Local Logic Editor window and the cursor moves to the beginning of the line that has the error.

Chapter 12 contains details and corrective actions for syntax errors and warnings.

10.6.3 Setting up Hardware Configuration for Local Logic

Once a successful syntax check has occurred, you need to set up the hardware configuration that allows the example program to be downloaded to the correct DSM314 module. Note that this is not the typical order in which these steps are done. Most users first set up their hardware configuration and then generate the programming statements. However, the order in this example is reversed to better illustrate the link between hardware configuration and the Local Logic program name in the DSM314 hardware configuration.

For details on how to perform steps 1 and 2, see the following:

- VersaPro Configuration: Appendix H
 - Machine Edition Configuration: Chapter 4
1. If you have not already done so, open the hardware configuration and configure a CPU that supports PACSystems RX3i Release 2.8 (or later) or Series 90-30 Release 10.0 (or later) firmware and an appropriate power supply for your application. Add a DSM314 to your rack configuration. This operation adds the DSM314 to the rack and opens the DSM314 configuration screens, which allow you to customize the DSM314 to your particular application.

Note: For details concerning the DSM314 configuration settings, refer to chapter 4.

2. On the “Settings” tab, set the “Local Logic Mode” parameter to Enabled and type the name of the example program, “LLEExample” in the “Local Logic Block Name” field. The resulting Hardware Configuration screens will be as shown in Figure 123.

Note: This method of linking the DSM314 to a Local Logic program allows you to easily specify multiple DSM314s that use the same Local Logic program. This example has only one DSM314. However, if you have multiple DSM314s that need to run the same Local Logic program, simply indicate that in the configuration for each DSM314 that needs to execute this program. This allows the programmer to have one Local Logic source file for multiple DSM314s. Also note that this does not preclude DSM314s from executing different programs.

Figure 123: Hardware Configuration DSM314 Settings Tab (RX3i version shown)

Settings SNP Port CTL Bits Output Bits Axis #1 Axis #2 Axis #3 Tuning #1 Tu ◀ ▶	
Parameters	Values
<i>Number of Axes</i>	4
%I Reference	%I00001
%I Length	80
%Q Reference	%Q00001
%Q Length	80
%AI Reference	%AI00085
%AI Length	84
%AQ Reference	%AQ00001
%AQ Length	12
<i>Axis 1 Mode</i>	Analog Servo
<i>Axis 2 Mode</i>	Analog Servo
<i>Axis 3 Mode</i>	Auxiliary Axis
<i>Axis 4 Mode</i>	Disabled
<i>Local Logic Mode</i>	Disabled
Total Encoder Power (Amps)	0.000
Motion Program Block Name	
Local Logic Block Name	
Cam Block Name	
I/O Scan Set	1

3. Configure return data.

The example Local Logic program shown on page 254 uses parameter registers P001, P003, and P004 as counters that contain values representing time. To view these parameter registers in the DSM return data registers, you need to configure return data. To configure return data:

- A. Select the Axis #1 tab and input 18 in Return Data 1 Mode. This tells the DSM that you want to return parameter registers. In Return Data 1 Offset, enter a 1. This tells the DSM to return parameter P001.

The LLEExample program returns P001, P003 and P004. However, the grouping is better if you return P003 and P004 in Axis #2. Therefore, you can either leave Return Data 2 Mode and Return Data 2 Offset at the default values or enter in 18 in Return Data 2 Mode and 2 in Return Data 2 Offset to tell the DSM to return P002. Note that Select Return Data 1 Axis1 is returned in %AI memory offset 21 while Return Data 2 for Axis 1 is returned in % AI offset 23.

Figure 124: Hardware Configuration DSM314 Axis#1 Tab

Parameters	Values
Drive Disable Delay (ms):	100
Jog Velocity:	1000
Jog Acceleration:	10000
Jog Acceleration Mode:	Linear
Home Position:	0
Home Offset:	0
Find Home Velocity:	2000
Final Home Velocity:	500
Home Mode:	Home Switch
Return Data 1 Mode:	18
Return Data 1 Offset:	1
Return Data 2 Mode:	18
Return Data 2 Offset:	2
Cam Master Source:	Actual Position 3
Follower Control Logic:	Disabled
Ratio A Value:	1

The above steps must be repeated for P003 and P004.

- B. Select the Axis #2 tab and input 18 in Return Data 1 Mode. This tells the DSM that you want to return parameter registers. In Return Data 1 Offset, enter a 3. This tells the DSM to return parameter P003.
- C. On the Axis #2 tab, enter in 18 in Return Data 2 Mode and 4 in Return Data 2 Offset to tell the DSM to return P004.

Note: Select Return Data 2 Axis 2 is returned in %AI memory offset 41 while Return Data 2 for Axis 2 is returned in %AI offset 43.

Figure 125: Hardware Configuration DSM314 Axis #2 Tab

Parameters	Values
Drive Disable Delay (ms):	100
Jog Velocity:	1000
Jog Acceleration:	10000
Jog Acceleration Mode:	Linear
Home Position:	0
Home Offset:	0
Find Home Velocity:	2000
Final Home Velocity:	500
Home Mode:	Home Switch
Return Data 1 Mode:	18
Return Data 1 Offset:	3
Return Data 2 Mode:	18
Return Data 2 Offset:	4
Cam Master Source:	Actual Position 3
Follower Control Logic:	Disabled
Ratio A Value:	1

4. Configure the CTL bit.

The sample Local Logic program shown on page 254 controls CTL01, which is used to signal the Motion Program that a second has passed. The CTL bit must be configured to be under Local Logic Control. To do this, access the CTL Bits tab in hardware configuration. Select “CTL01 Config” and choose Local_Logic_Controlled. The resulting CTL01 tab is shown in Figure 126.

Figure 126: Hardware Configuration DSM314 CTL Bits Tab

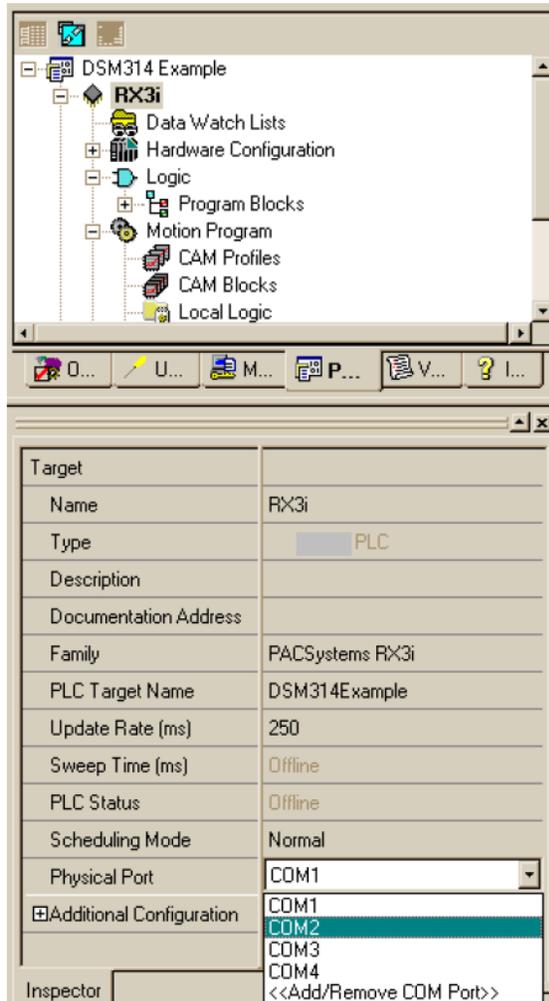
Parameters	Values
CTL01 Config:	Local Logic Controlled
CTL02 Config:	IN10_A (Axis 1 - OT)
CTL03 Config:	IN11_A (Axis 1 Home SW)
CTL04 Config:	Strobe 1 Level (Axis 1)
CTL05 Config:	IN9_B (Axis 2 + OT)
CTL06 Config:	IN10_B (Axis 2 - OT)
CTL07 Config:	IN11_B (Axis 2 Home SW)
CTL08 Config:	Strobe 1 Level (Axis 2)
CTL09 Config:	%Q Bit Offset 12
CTL10 Config:	%Q Bit Offset 13
CTL11 Config:	%Q Bit Offset 14
CTL12 Config:	%Q Bit Offset 15
CTL13 Config:	IN9_C (Axis 3 + OT)
CTL14 Config:	IN10_C (Axis 3 - OT)
CTL15 Config:	IN11_C (Axis 3 Home SW)
CTL16 Config:	Strobe 1 Level (Axis 3)
CTL17 Config:	%Q Bit Offset 24

5. This completes the configuration changes necessary for the example. Close the Hardware Configuration tool and save the folder. The link between the example Local Logic program and the DSM314 module is now complete. You can now create any required ladder logic and then perform a Check All on the programs.

10.7 Downloading a Local Logic Program

To perform the download operation, first make sure that the communications port is properly configured. To access communications setup, click on the target you want to connect to in the Navigator window. Using Machine Edition, in the Inspector window, select the Physical Port you want to connect through. (For information on downloading using VersaPro, see Appendix H.)

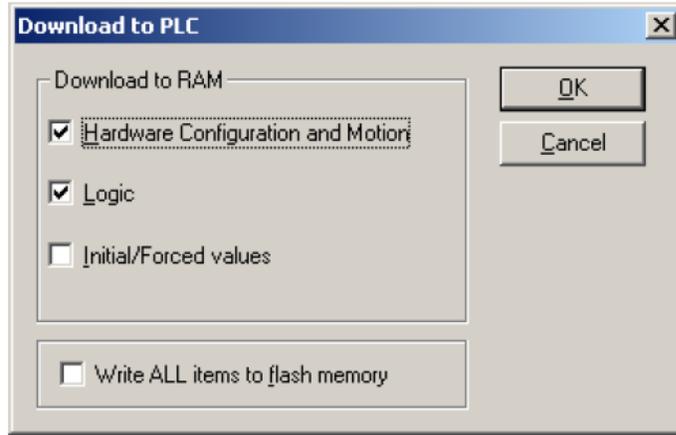
Figure 127: Communications Setup



After configuring the communications port, the local logic program can be downloaded (stored) to the Host Controller CPU. To store the current folder to the Host Controller, choose Target from the Menu Bar and Go Online with “<Target>” from the submenu. Once connected, choose Target from the Menu Bar and Download “<Target>” to PLC from the submenu. The store operation begins the folder transfer process from the programmer to the Host Controller CPU. When you initiate the store operation, a dialog box is presented that allows you to choose what to store to the Host Controller. In this case, you want to store the Local Logic program, Hardware configuration, and any Host Controller logic. To perform this operation, select, in the dialog box, Store hardware configuration and motion to the PLC and Store logic to PLC.

Note: The Local Logic and Motion programs are transferred as part of the Hardware configuration process. Thus to download an updated Local Logic program and/or Motion program, select the Hardware Configuration and Motion item in the Download to PLC dialog box.

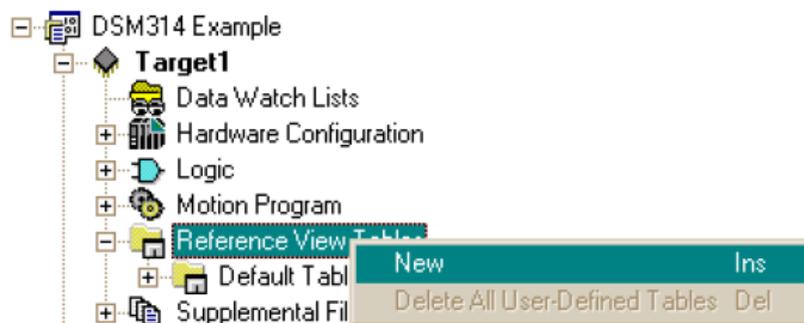
Figure 128: Machine Edition Download Dialog Box



Machine Edition will then check any blocks that have changed. If the build procedure is successful, it will download the files to the Host Controller. Machine Edition will indicate any errors or that it has successfully downloaded the program in the Feedback Zone window.

When the programs are downloaded to the host controller, you can interact with the DSM to verify that the Local Logic program is working correctly. The Reference View Table (RVT) display can be used for this operation. To create an RVT, right click on the Reference View Tables folder in the Navigator window and select New from the menu. The new RVT is added to the project.

Figure 129: Creating a New Reference View Table



You can insert variables, select variable display formats, toggle data points, and send AQ commands, among other actions. Consult the Machine Edition documentation for details on RVT construction. A sample RVT that is useful for this program is shown below.

Figure 130: Reference View Table

									Address
+0	+0	+0	+0	+0	+0	+0	+0	+0	%AI0001
+0	+0	+0	+0	+0	+0	+0	+0	+0	%AI0011
+0	+0	+0	+0	+0	+0	+0	+0	+0	%AI0021
+0	+0	+0	+0	+0	+0	+0	+0	+0	%AI0031
+0	+0	+0	+0	+0	+0	+0	+0	+0	%AI0041
+0	+0	+0	+0	+0	+0	+0	+0	+0	%AI0051
+0	+0	+0	+0	+0	+0	+0	+0	+0	%AI0061
+0	+0	+0	+0	+0	+0	+0	+0	+0	%AI0071
+0	+0	+0	+0	+0	+0	+0	+0	+0	%AI0081
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	%I00001
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	%I00065
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	%Q00001
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	%Q00065
+0	+0	+0	+0	+0	+0	+0	+0	+0	%AQ0001
+0	+0	+0	+0	+0	+0	+0	+0	+0	%AQ0011

10.8 Executing Your Local Logic Program

Once the download operation is complete, the module is ready to execute the local logic program. To cause the DSM module to execute the local logic program you must set the Q bit offset 1 from the host controller, while the host controller is in RUN mode. At this point, the local logic program is active and running within the DSM.

Note: *The LLEExample sample program is a simple counter application. The user can use the RVT to look at the passed parameters to verify that the program is active and functioning correctly. From the RVT, you can see that 1 Minute 8 Seconds have passed since Local Logic was started (see %AI0043 and %AI0041, respectively). Additionally, 68370 milliseconds have passed as shown in %AI0021. Additional details concerning the interface between the DSM and the host controller are contained in chapter 5. You should save the folder once the program has been verified to work correctly.*

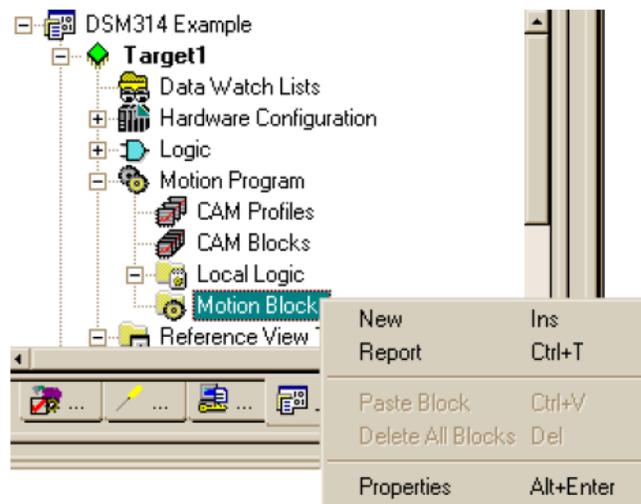
10.9 Using the Motion Program Editor

Now that you have successfully gotten the Local Logic program working, it would be useful to link in a Motion Program. The Motion Program editor is accessed in a manner very similar to the Local Logic editor. The editor allows you to easily create, edit, store, and download Motion programs.

10.9.1 Creating a Motion Program

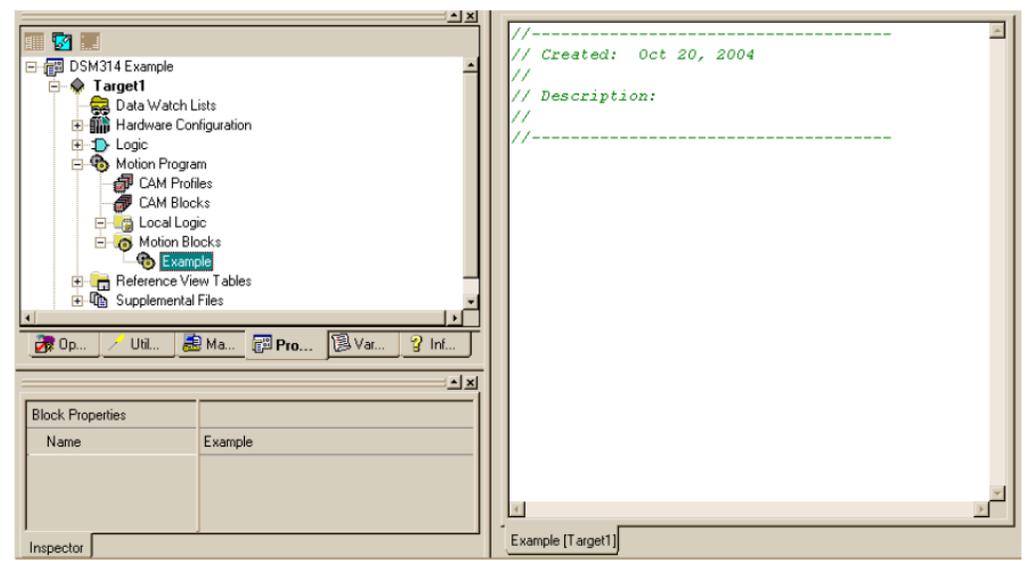
To create a Motion program in Machine Edition, expand the Motion Program folder in the Navigator, then right click the Motion Blocks folder and choose New. The new Motion block appears in the Navigator.

Figure 131: Creating a Motion Program in Machine Edition



To open the Motion editor, double click the Motion block.

Figure 132: Motion Program Editor



The Motion editor is a free-form text editor that allows you to enter a program in the style that you prefer. The example uses a very simple Motion program. The example does not represent a functional application and is for instructional purposes. The example is linked with the Local Logic program entered in “Creating a Local Logic Program.” The Local Logic program from page 254 is repeated for reference:

```
(*****)
(* Program Name: LLExample *)
(* Description: The following example local logic program is *)
(* a program that creates a simple timer. The timer begins *)
(* after the first Local_Logic_Sweep has occurred. The *)
(* program makes use of the fact that Local Logic is run by *)
(* the DSM every 2 mSec. Thus, it counts local logic sweeps. *)
(* The program has three counters. The first counter is the *)
(* milliseconds counter while the other two counters are for *)
(* seconds and minutes. The seconds counter and minutes *)
(* counters roll over at 59. The program also sets CTL01 every *)
(* second. *)
(*****)
(* Variables *)
(* P001 = Milliseconds Counter *)
(* P003 = Seconds Counter *)
(* P004 = Minutes Counter *)
(* CTL01 = Seconds Signal *)
(* P100 = Used to check if 1 Sec has passed *)
(* P102 = Used to check if 1 Min has passed *)
(*****)

IF First_Local_Logic_Sweep THEN (* First execution sweep *)
    P001 := 0; (* Initialize P001 to 0 *)
    P003 := 0; (* Initialize P003 to 0 *)
    P004 := 0; (* Initialize P004 to 0 *)
END_IF;

P001:=P001+2; (* Time in Milliseconds *)

P100:= P001 MOD 1000; (* Check to see if 1 Sec (1000 mSec) Passed *)
IF P100 = 0 THEN (* Remainder of MOD Operation=0 1 Sec Passed *)
    P003:=P003+1; (* Time in Seconds *)
    CTL01 := 1;
    IF P003 = 60 THEN (* If Seconds = 60 then start over at 0 *)
        P003:=0;
    END_IF;
END_IF;
IF P100 <> 0 THEN
    CTL01 :=0; (* CTL01=0 when not incrementing sec counter *)
END_IF;

P101:=P001 MOD 60000; (* Check to see if 1 Min (60000mSec) Passed *)
IF P101 = 0 THEN (* Remainder of MOD Operation=0 1 Min Passed *)
    P004:=P004+1; (* Time in Minutes *)
    IF P004 = 60 THEN (* If Minutes = 60 then start over at 0 *)
        P004:=0;
    END_IF;
END_IF;
```

The Local Logic program causes CTL01 to transition from logic 0 to logic 1 every second. For this simple Motion program example, the motor shaft rotates 1/60 of a revolution for each CTL01 transition. The motion program will therefore make the motor shaft act like the second hand on a quartz clock.

Before writing the Motion Program, you will need to determine axis scaling. The first variable you need to determine is the user units to counts ratio. The User Units to Counts ratio sets the number of programming units for each position feedback count. This allows the user to program the DSM314 in application-specific units. The User Units and Counts values must be within the range of 1 to 65,535. The User Units to Counts ratio must be within the range of 8:1 to 1:32. For example, if there is 1.000 inch of travel for 8192 feedback counts, a 1000:8192 User Units: Counts ratio sets 1 User Unit equal to 0.001 inch.

To set the User Units to Counts ratio the first piece of information required is the number of counts per revolution of the feedback device. This example uses a Beta 0.5 motor. The Beta 0.5 has a feedback resolution of 8192 counts per revolution. Now perform the calculation to determine the ratio. The basic equation is:

$$\frac{\text{User Units}}{\text{Counts}} = \left(\frac{\text{Load Movement Per Motor Rotation}}{\text{Desired Resolution}} \right) \cdot \frac{1}{\text{Encoder Counts Per Motor Rotation}}$$

For this example:

$$\frac{\text{User Units}}{\text{Counts}} = \left(\frac{1}{\frac{1}{60}} \right) \cdot \frac{1}{8192}$$

$$\frac{\text{User Units}}{\text{Counts}} = \frac{60}{8192}$$

This ratio is a problem since it violates the rule that the minimum User Units to Counts Ratio is $\frac{1}{32}$. The problem is easy to fix: change the programming units from 60th of a revolution to a 600th of a revolution. This will make 1 programming unit equal to $\frac{1}{600}$ revolution .

Repeat the above calculation:

$$\frac{\text{User Units}}{\text{Counts}} = \left(\frac{1}{\frac{1}{600}} \right) \cdot \frac{1}{8192}$$

$$\frac{\text{User Units}}{\text{Counts}} = \frac{600}{8192}$$

Thus, to have the motor travel $\frac{1}{60}$ of a revolution, you must enter 10 units in the motion program. Additional information on setting the User Units to Counts ratio is provided in Chapter 4.

The next item you need to determine is the motor top speed. This is a relatively simple calculation.

$$\begin{aligned} \text{TopSpeed} \cdot \left(\frac{\text{uu}}{\text{sec}}\right) &= \text{Motor Top Speed} \left(\frac{\text{rev}}{\text{sec}}\right) \cdot \text{Enc. Counts per Rev} \cdot \left(\frac{\text{cnts}}{\text{rev}}\right) \cdot \frac{\text{User Units}}{\text{Counts}} \cdot \left(\frac{\text{uu}}{\text{cnts}}\right) \\ \text{TopSpeed} \cdot \left(\frac{\text{uu}}{\text{sec}}\right) &= \frac{3000}{60} \cdot \left(\frac{\text{rev}}{\text{sec}}\right) \cdot 8192 \cdot \left(\frac{\text{cnts}}{\text{rev}}\right) \cdot \frac{600}{8192} \cdot \left(\frac{\text{uu}}{\text{cnts}}\right) \\ \text{TopSpeed} \cdot \left(\frac{\text{uu}}{\text{sec}}\right) &= 3000 \cdot 10 \cdot \left(\frac{\text{uu}}{\text{sec}}\right) \\ \text{TopSpeed} \cdot \left(\frac{\text{uu}}{\text{sec}}\right) &= 30000 \cdot \left(\frac{\text{uu}}{\text{sec}}\right) \end{aligned}$$

Next, you need to calculate the velocity and acceleration required for the move. In this example, a triangular velocity profile is chosen to minimize time. The equations to calculate the parameters are shown below.

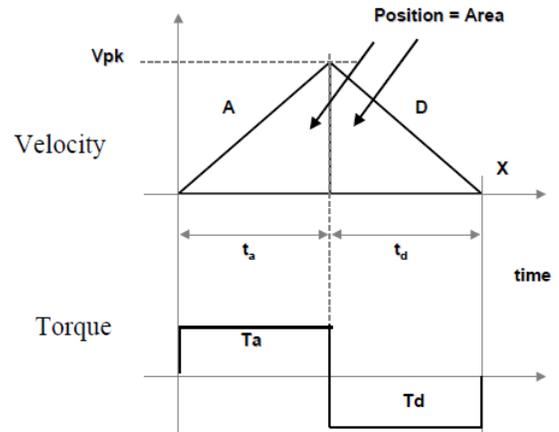
Figure 133: Motion Program Editor

Equations:

$$x = \frac{1}{2} V_{pk} (t_a + t_d)$$

$$V_{pk} = \frac{2(x)}{(t_a + t_d)}$$

$$a = \frac{V_{pk}}{t_a}$$



Applying the numbers from this example to the triangular velocity equations gives the following:

Given :

$$t_a = 0.01 \cdot \text{sec}$$

$$t_d = 0.01 \cdot \text{sec}$$

$$x = \frac{1}{60} \cdot \text{rev}$$

$$V_{pk} = \frac{2 \cdot \frac{1}{60} \cdot \text{rev}}{0.01 \cdot \text{sec} + 0.01 \cdot \text{sec}}$$

$$V_{pk} = 1.6667 \cdot \frac{\text{rev}}{\text{sec}}$$

$$V_{pk} = 1.6667 \cdot \frac{\text{rev}}{\text{sec}} \cdot 8192 \cdot \frac{\text{cnts}}{\text{rev}} \cdot \frac{600}{8192} \cdot \frac{\text{uu}}{\text{cnts}}$$

$$V_{pk} = 1000 \cdot \frac{\text{uu}}{\text{sec}}$$

$$a = \frac{V_{pk}}{t_a}$$

$$a = \frac{1000 \cdot \frac{\text{uu}}{\text{sec}}}{0.01 \cdot \text{sec}}$$

$$a = 10000 \cdot \frac{\text{uu}}{\text{sec}^2}$$

You are now ready to write a motion program. The code for the sample program is as follows.

```
(*****)
(* Program Name: MPEXAMPLE *)
(* Description: The following Motion program causes *)
(* the motor to rotate 10 Units (Distance based *)
(* upon scaling) every time CTL01 transitions from *)
(* 0 to 1. *)
(*****)
(* Variables *)
(* CTL01 = Program execution trigger *)
(*****)

PROGRAM 1 AXIS1 (* Program Number 1 for Axis 1 *)

    ACCEL 10000 (* 10000 uu/sec^2 *)
    VELOC 1000 (* 100 uu/sec *)

1:
    WAIT CTL01 (* Wait for CTL01 Signal *)
    PMOVE 10, INCR, LINEAR (* Position Move 10 Incr Linear *)
    JUMP UNCOND, 1 (* Jump back to start *)

ENDPROG (* End Program *)
```

When the above program has been typed into the text editor, the editor will look similar to Figure 134. .

Figure 134: Motion Editor MPEExample

```

(*****)
(* Program Name: MPEExample *)
(* Description: The following Motion program causes *)
(* the motor to rotate 10 Units (Distance based *)
(* upon scaling) every time CTL01 transitions from *)
(* 0 to 1. *)
(*****)
(* Variables *)
(* CTL01 = Program execution trigger *)
(*****)

PROGRAM 1 AXIS1 (* Program Number 1 for Axis 1 *)

ACCEL 10000 (* 10000 uu/sec^2 *)
VELOC 1000 (* 100 uu/sec *)

1:
WAIT CTL01 (* Wait for CTL01 Signal *)
PMOVE 10, INCR, LINEAR (* Position Move 10 Incr Linear *)
JUMP UNCOND, 1 (* Jump back to start *)

ENDPROG (* End Program *)

```

Note: When the cursor is in the motion editor window, the line and column numbers appear in the status bar at the bottom of the Logic Developer window.

At this point, you should check the program to verify correct language syntax. At this point, the user needs to check the program to verify correct language syntax. The language syntax verification is done by selecting Target from the main menu, and then selecting Validate '<Target>'. .

The information window displays the output of the syntax check operation. If the sample program has been entered correctly, you should receive a message indicating zero errors and zero warnings.

If the information window indicates a syntax error has occurred, press F4 to cycle through the warnings and errors. While the information window has focus, double click the error message. This causes the editor window to automatically go to the line in the program that caused the error.

Chapter 12 contains additional details that cover corrective actions for syntax errors and warnings. Once the program passes the syntax check, you need to set up the hardware configuration that will allow the program to be downloaded to the correct DSM314 module.

10.9.2 Setting Motion Program Parameters in Hardware Configuration

The section describes the parameters that must be set in the Hardware configuration to allow the motion program to function. For details concerning the DSM314 configuration settings, consult chapter 4.

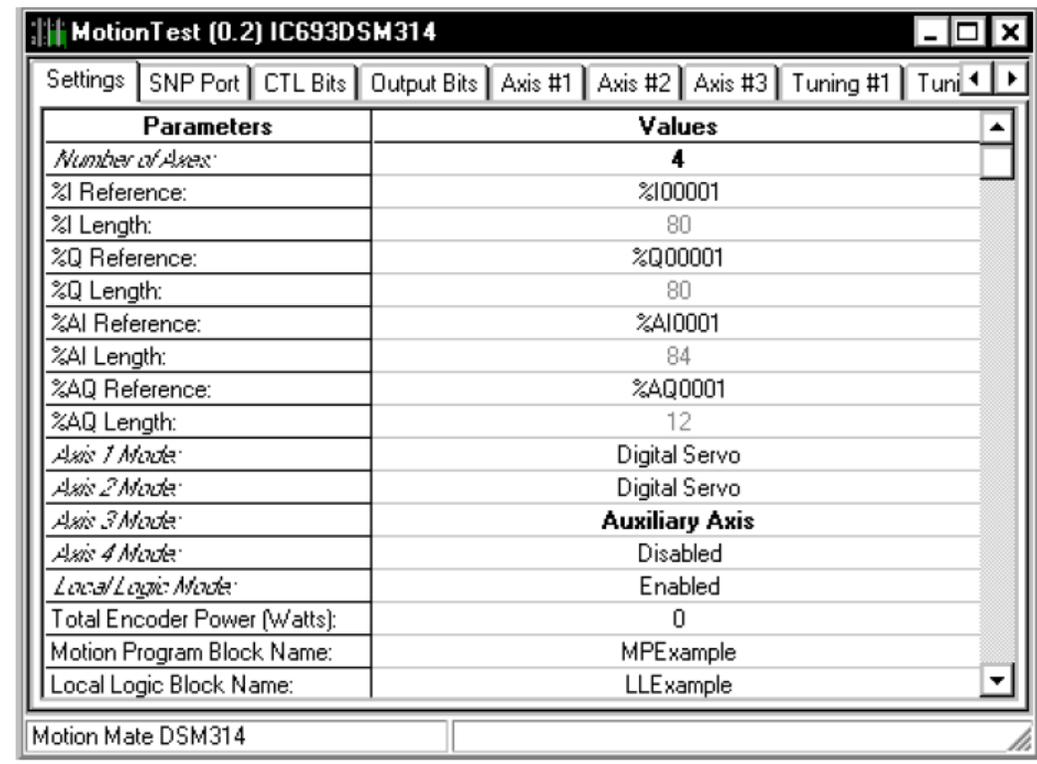
The order in which the example is done is not typical for most installations. Most users will first set up their hardware configuration and then generate the programming statements. However, this example is intended to illustrate the Motion programs and reverses the order to better illustrate the link between hardware configuration and the Motion program name in the DSM314 hardware configuration.

The first field you need to edit is the “Motion Program Block Name” on the Settings tab. This field identifies to the DSM314 the Motion program name to be downloaded to the module. Type the name of the example program, “MPEexample,” into this field.

Note: *This example has only one DSM314. However, if you have multiple DSM314s that need to run the same Motion program, you can indicate that in the configuration for the each DSM314. This allows the programmer to have one Motion program source file for multiple DSM314s. This does not prevent DSM314s from executing different programs.*

Since the example uses the Beta 0.5, set Axis1 Mode to Digital Servo.

Figure 135: Hardware Configuration DSM314 Settings Tab



You also need to configure the DSM with the values calculated above for User Units to Counts and top speed. The example also configures Axis direction and high position limit. These are optional. Consult chapter 4 for information on these configuration fields. To add these values, type the following into the fields on the Axis#1 tab.

UserUnits: 600

Counts: 8192

High Position Limit: 599 (Optional, causes position to roll over every revolution)

Velocity Limit: 30000

Axis Direction: Reverse (Optional causes servo to turn clockwise)

Figure 136: Hardware Configuration DSM314 Axis#1 Tab

Parameters	Values
User Units:	600
Counts:	8192
Over Travel Limit Switch:	Disabled
Drive Ready Input:	Enabled
High Position Limit:	599
Low Position Limit:	0
High Software EOT Limit:	8388607
Low Software EOT Limit:	-8388608
Software End of Travel:	Disabled
Velocity Limit:	30000
Command Direction:	Bidirectional
Axis Direction:	Reverse
Feedback Source:	Default
Feedback Mode:	Incremental
Reversal Compensation:	0
Drive Disable Delay (ms):	100
Jog Velocity:	1000
Jog Acceleration:	10000
Jog Acceleration Mode:	Linear
Home Position:	0
Home Offset:	0
Find Home Velocity:	2000
Final Home Velocity:	500

To finish the configuration, enter the following values in Tuning#1 tab.

Motor Type: 13

Position Error Limit: 200 (Optional. See Configuration information for additional information.)

In Position Zone: 5 (Optional. See Configuration information for additional information.)

Pos Loop Time Const: 200 (Note: Based upon application/mechanics. Refer to Chapter 4 and Appendix D)

Velocity FeedForward: 9000 (Note: Based upon application/mechanics. Refer to Chapter 4 and Appendix D)

The resulting display should be similar to Figure 10-20. .

Figure 137: Hardware Configuration Tuning#1 Tab

Parameters	Values
Motor Type:	13
Analog Servo Command:	Velocity
Position Error Limit:	200
In Position Zone:	5
Position Loop Time Const	200
Velocity at Max Cmd:	38400
Velocity Feed Forward (.0	9000
Acceleration Feed Forwar	0
Integrator Mode:	Off
Integrator Time Constant	0
Velocity Loop Gain:	32

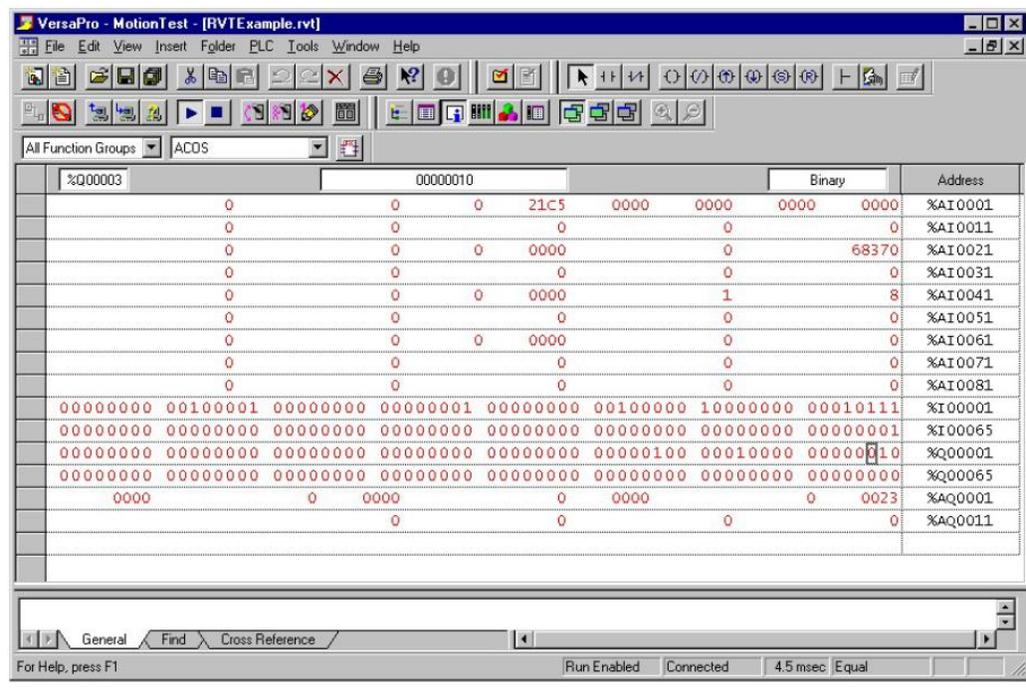
To save your work, select the File from the main menu and then select Save All from the file menu.

The link between the example Motion program, Local Logic program, and the DSM314 module is now complete. Create any required ladder logic, validate the programs and download them to the host controller.

10.10 Executing Your Motion Program

Once the download operation is complete, the module is ready to execute the Motion and Local Logic programs. To cause the DSM module to execute the local logic program, set the Q bit offset to 1 from the host controller, while the host controller is in RUN mode. This activates the Local Logic program within the DSM. The next thing you need to do is perform a Set Position command. This references the module and allows it to execute the desired motion program. To perform this function, open the RVT (RVTEExample) created in the Local Logic section and enter 0023 hex in AQ offset 1. This enters the Set Position command. Then enter 0 in AQ offset 2. Refer to Chapter 5 for additional information concerning entering AQ commands. The resulting display should be similar to the following figure.

Figure 138: RVTEExample Screen



At this point, if there are no errors, you can execute the motion program. Enter a 1 (or toggle) Q bit offset 2 (%Q00003). The motor should execute the motion program and advance 1/60 of a revolution each second.

Additional details concerning the interface between the DSM and the host controller are contained in Chapter 5.

Chapter 11: Local Logic Tutorial

The Local Logic programming language supports assignment, conditional statements, arithmetic, logical, and relational operations. The Local Logic program runs synchronously with the motion module position loop and therefore is deterministic. The language includes constructs that allow the Local Logic program to communicate information between the Logic program, the Motion Program, and the host controller. The tutorial focuses on the local logic language and its communication with motion programs. Chapter 7 provides additional information concerning the motion programmer language.

11.1 Statements

The Local Logic programming language supports assignment and conditional statements. Assignment statements permit arithmetic results and bitwise logical operations to be assigned to a variable. Conditional statements permit conditional local logic code execution. Conditional execution is based on the value of a constant or variable, or the result of a relational or bitwise logical expression.

Assignment statements use the “:=” operator. The following example multiplies two parameter registers and assigns the result to another parameter register.

```
P001:= P210 * P107;
```

Note: Assignment statements require a semi-colon terminator as shown above.

Conditional statements use the IF-THEN-END_IF keyword combination. The END_IF keyword concludes the conditional statement. The following example checks the Block_1 variables value and conditionally sets a value in a parameter register. Specifically, if the Block_1 variable’s value equals 5 then the parameter P010 value is set to 100.

```
IF Block_1 = 5 THEN  
    P010 := 100;  
END_IF;
```

The IF, THEN, and END_IF keywords are case sensitive, and the END_IF statement is terminated with a semi-colon. IF statements may be nested up to eight levels and the body of the IF statements may contain one or more statements. Refer to Chapter 12 for a detailed description of these statements.

11.2 Comments

Comments allow the programmer to describe program operation, or any information that the programmer wishes to embed in the program. Comment text begins with the (* character pair and terminates with the *) character pair and may appear anywhere within the program. For example:

```
(* Valid Comment Structure *)
```

The DSM during program execution ignores comments. Thus, comment lines do not count when determining local logic program length.

11.3 Variables

Local Logic provides the user access to motion controller data, control and status bits, and parameters using a fixed set of variables. The language also supports decimal, hexadecimal, and binary constants. Hexadecimal and binary value representations are unsigned constants in program statements, but are ALWAYS interpreted as signed two's complement in mathematical expressions. To assign a value to a variable the user would enter the following

```
Torque_Limit_1 :=5000; (* Sets Torque Limit Axis 1 to 50% *)
```

or in hexadecimal form

```
Torque_Limit_1:=16#1388; (* Set Torque Limit Axis 1 to 50% *)
```

When variables are assigned a numeric value they are automatically limit checked to a signed 32-bit value. For example the following values represent the largest positive and negative values that are acceptable.

```
P001:=16#7FFFFFFF; (* P001=2147483647 *)
```

or in decimal form

```
P001:=2147483647; (* P001=16#7FFFFFFF *)
```

To assign the maximum negative value the user would enter

```
P002:=16#80000000; (* P002=-2147483648 *)
```

or in decimal form

```
P002:=-2147483648; (* P002=16#80000000 *)
```

If the user enters a number that exceeds the above numerical limits an error will be generated indicating that the constant is out of range.

Local Logic variables have a read, write, or read/write “directional” attribute. (Additional information concerning the variables and their type are contained in chapter 13.) As an example, the variable Positive End of Travel for Axis 1 (Positive_EOT_1) is a read only variable. As such, the following is a valid construct:

```
P001:=Positive_EOT_1; (* P001 = Positive End of Travel Axis 1 *)
```

However, the following is an invalid construct:

```
Positive_EOT_1:=1;
```

The Local Logic Parser generates an error if the program attempts to write to a read only variable, or attempts to read a write only variable.

In addition, Local Logic variables have a size attribute ranging from Boolean (1-bit) to double integer (64-bits). The Local Logic Parser generates a warning message when a non-Boolean value is assigned to a Boolean variable. The warning indicates that data may be lost, due to truncation, when this assignment occurs. The user should note that double integer variables (64-bit) variables may only be used as the destination of a multiply operation, or the numerator of a divide or modulus operation.

Consult chapter 13 for additional information concerning Local Logic variables. Additionally, the Local Logic Variables Table (LLVT) within the programming software contains the information on the variables size, type and Read/Write properties.

11.4 Operators

Local logic provides three classes of operators. The operators are arithmetic, relational, and bitwise logical operators. An introduction to each operator follows. A more detailed discussion of the operators is contained in Chapter 12.

11.4.1 Arithmetic Operators

Local Logic provides the user with the ability to perform basic arithmetic operations. The language supports 32-bit integer operations and limited use of 64/32 bit operations where appropriate to maintain precision. All arithmetic functions, except the ABS function, require two operands.

Local Logic supports addition, subtraction, multiplication, division, absolute value, and modulus operations.

Example constructs are:

```
P010 := Commanded_Velocity_1 - P009; (* P010=Commanded Velocity Axis 1 - P009*)
```

The user should note that the following would be an invalid mathematical construct:

```
Commanded_Velocity_1 := P010 - P009; (* Commanded_Velocity_1=P010-P009*)
```

The reason this is invalid is that the mathematical expression is attempting to assign the result (P010-P009) to Commanded_Velocity_1 which is a read-only variable.

Storing intermediate results into parameter registers provides the flexibility necessary to solve complex mathematical expressions.

For example, the following construct is invalid since it contains more than one operation (Multiply and Subtraction):

```
P005: = Torque_Limit_1 *( P001 - P010);
```

To achieve the same result, the user can enter the following:

```
P004: = P001 - P010;
```

```
P005: = Torque_Limit_1 * P004;
```

11.4.2 Relational Operators

Relational operators compare two operands in a conditional statement. The < (less than), > (greater than), <= (less than or equal), >= (greater than or equal), = (equal), and <> (not equal) operators are valid relational operators. The IF statement body execution takes place when the conditional expression is a true. In the example, the variable Torque_Limit_1 is set to 10000 if the variable Block_1 equals 3. If the Block_1 value is not equal to 3 then the expression evaluates to false and program execution continues after the END_IF program statement.

Example:

```
IF Block_1 = 3 THEN  
    Torque_Limit_1 := 10000; (* Set Torque Limit = 100% @ Block 3 *)  
END_IF;
```

Complex relations may be solved by nesting IF statements. For example, to set Axis 1 torque limit (Torque_Limit_1) to 10000=100% (i.e. same scaling as in AQ command processing) when the motion program block 3 is active and axis 1 commanded velocity (Commanded_Velocity_1) is less than 1000, the following construct is valid:

```
IF Block_1 = 3 THEN  
    IF Commanded_Velocity_1 < 1000 THEN  
        Torque_Limit_1 := 10000; (* Set Torque Limit = 100% @ Block 3 *)  
    END_IF;  
END_IF;
```

11.4.3 Bitwise Logical Operators

Bitwise logical operators mask or invert an individual bit or groups of bits. The BWAND (and), BWOR (or), BWXOR (exclusive or), and BWNOT (ones-complement) operators are valid constructs. BWAND, BWOR, and BWXOR require two operands. The BWNOT operator requires one operand.

As an example, the following code segment isolates a copy of several bits in the CTL_1_to_32 word and assigns them to a parameter register.

Then, the least significant four bits of that value are tested and P002 is assigned a value 4985 if any are set.

```
P001 := CTL_1_to_32 BWAND 16#0000A005;
IF P001 BWAND 16#F THEN
    P002 := 4985;
END_IF;
```

Specifically, the statements perform the following operations. The first statement uses 16#0000A005 as a mask. The mask corresponds to a binary value as follows:

$$16\# 0000A005 = 2\#0000\ 0000\ 0000\ 0000\ 1010\ 0000\ 0000\ 0101$$

Thus, the statement

```
P001:=CTL_1_to_32 BWAND 16#0000A005
```

isolates bits 1,3,14, and 16 from CTL_1_to_32 and places the result in P001.

The next statement performs a bitwise test to see if any of the bits in the least significant byte are set. The test value corresponds to a binary value as follows:

$$16\#F = 2\#1111$$

Thus the statement

```
IF P001 BWAND 16#F THEN
```

performs a bitwise test with the least significant byte of P001 and if any of the bits in the least significant byte are set to a logical true (value = 1) then statements in the IF block are evaluated.

In this example, since CTL_1_to_32 is masked in the previous statement, the IF condition only tests bit 1 and bit 3 of CTL_1_to_32.

11.5 Local Logic / Host Controller / Motion Program Communication

The Local Logic program or host controller communicates with the motion program using parameters, CTL bits and Motion Program Block Numbers. These methods are used as follows:

- **Parameter Data** – The Parameter data (P000-P255) are accessible from Local Logic, host controller, and Motion Programs. The Parameter data are similar to variables in a program. For example, a motion program can DWELL a period of time that is determined by a parameter. The Local Logic program or the host controller can write the parameter that determines the DWELL time in motion program.
- **CTL Bits** – CTL Bits allow the Local Logic program or host controller to signal the Motion Program to start an event. For example, CTL bits are used to control Motion Program flow with the JUMP command.
- **Motion Program Block Numbers** – The Motion Program (when block numbers are used within the Motion Program) makes the current block number available to the Local Logic program or host controller. The current Block number can be used within the Local Logic program or host controller to make an action occur only during a specific Motion Program section.

The signaling constructs between programs (host controller, Motion, and Local Logic) allow them to interact and perform operation between programs. These signaling constructs are important for the programming examples that follow. For additional information on the host controller-to-motion program communications and program interactions the reader should consult chapter 5 and Chapter 7.

11.6 Local Logic Programming Examples

The preceding sections introduced the base local logic language constructs. To illustrate these concepts, the following sections contain program examples. These programs are for illustration only and do not necessarily represent functional applications. Additional details concerning the available local logic statements, variables and constructs are contained in chapters 12 and 13.

11.6.1 Torque Limiting Program Example

The following example illustrates a method to use local logic in concert with a motion program to perform torque limiting based upon a block number within a motion program. In the example, the servo axis 1 applies a nut on the threaded shaft. At the beginning the axis moves a little backward to improve the nut and shaft threads engagement. This motion has the torque limit set to the maximum value. Next the nut is twisted until tight with the torque limited to 30% of the maximum value. During this operation the motion command destination point usually is not reached and the axis stops when the load friction is greater than the torque limit. Subsequently, to release all tension in the mechanics, the torque is set to 0 and after 0.1 second the signal “screw operation done” is turned “on”. When the

“nut gripper released” signal is turned on by the host controller, the axis moves to the initial position with the full torque.

Torque Limiting Local logic program.

```
(*CONTROL BIT ASSIGNMENTS*)

(*CTL01 - start screw operation - signal from the host controller*)
(*CTL02 - loop the motion program 1*)
(*CTL03 - nut gripper released - signal from the host controller*)
(*CTL04 - screw operation done - signal to the host controller*)

(*PARAMETER DATA ASSIGNMENTS*)

(*P001 - acceleration of the reverse motion*)
(*P002 - velocity of the reverse motion*)
(*P003 - distance of the reverse motion*)
(*P004 - acceleration of the nut twist motion*)
(*P005 - velocity of the nut twist motion*)
(*P006 - distance of the nut twist motion*)
(*P007 - counter used for tension release timer*)

(*LOCAL LOGIC CODE*)

IF Program_Active_1 = 1 THEN (*axis 1 motion program active*)
  IF Block_1 = 10 THEN (*motion prog. waits for start command*)
    P007:=0; (*reset counter*)
    CTL04:=0; (*clear screw operation done*)
    Torque_Limit_1:= 10000; (*set torque limit to maximum value 100%*)
  END_IF;
  IF Block_1=30 THEN (*main nut twist motion*)
    Torque_Limit_1:= 3000; (*set torque limit to 30%*)
  END_IF;
  IF Block_1=40 THEN (*mot prog waits for released signal*)
    Torque_Limit_1:= 0; (*set torque limit =0*)
    P007:=P007 + 1; (*update timer counter*)
    IF P007=50 THEN (*100 ms elapsed*)
      CTL04:=1; (*set screw operation done*)
    END_IF;
  END_IF;
  IF Block_1=50 THEN
    Torque_Limit_1:= 10000; (*set torque limit to maximum value 100%*)
  END_IF;
END_IF;
```

Torque Limiting Motion Program

(* Torque Limiting Motion Program *)

Program 1 AXIS1

```
10:WAIT CTL01 (* Wait for start command *)
  ACCEL P001 (* Set acceleration of the motion *)
  VELOC P002 (* Set velocity of the motion *)
20:PMOVE P003,INCR,LINEAR (* Backward motion *)
  ACCEL P004 (* Set acceleration of the motion *)
  VELOC P005 (* Set velocity of the motion *)
30:CMOVE P006,INCR,LINEAR (* Main nut twist motion *)
40:WAIT CTL03 (* Wait for gripper released signal *)
50:PMOVE 0,ABS,LINEAR (* Move to the initial position *)
60:JUMP CTL02,10 (* Jump to begin if loop program *)
```

EndProg

11.6.2 Gain Scheduler Program Example

The following example illustrates a method to use local logic to implement a simple gainscheduling algorithm. Care should be taken whenever one implements an algorithm that dynamically changes the control characteristics. In many situations, dynamically changing the control characteristics can cause the controlled process to go unstable. Note that the Velocity_Loop_Gain control variable may be written multiple times in the same sweep in the following program. However, the final value written in a given sweep is the active value since variables are updated at the conclusion of Local Logic execution. Refer to Chapters 12 and 13 for a detailed description of the Local Logic control variables and outputs.

Gain Scheduler Local Logic Program

```
(* This example shows a way to implement a rudimentary gain
*)
(* scheduling algorithm. The base velocity loop gain is
*)
(* set to 12. However, based upon position and commanded
*)
(* velocity the velocity loop gain is adjusted.
*)

Velocity_Loop_Gain_1 := 12;      (* Base Velocity Loop Gain *)
IF Actual_Position_1 > 500 THEN (* If Actual Position Axis 1 > 500 *)
  IF Commanded_Velocity_1 < 10 (* If Commanded Velocity Axis 1 < 10 *)
    Velocity_Loop_Gain_1 := 16; (* Velocity Loop Gain Axis 1 = 16 *)
  END_IF;
END_IF;

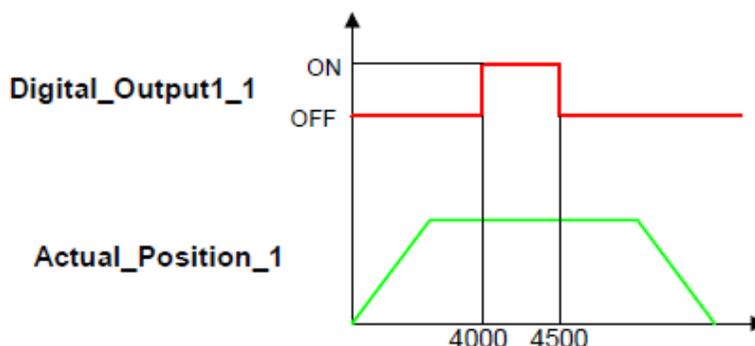
IF Actual_Position_1 > 1000 THEN (* If Actual Position Axis 1 > 1000 *)
  IF Commanded_Velocity_1 < 10 (* If Commanded Velocity Axis 1 < 10 *)
    Velocity_Loop_Gain_1 := 20; (* Velocity Loop Gain Axis 1 = 20 *)
  END_IF;
END_IF;

IF Actual_Position_1 > 1500 THEN (* If Actual Position Axis 1 > 1500 *)
  IF Commanded_Velocity_1 < 10 THEN (* If Cmd Velocity Axis 1 < 10 *)
    Velocity_Loop_Gain_1 := 24; (* Vel Loop Gain Axis 1 = 24 *)
  END_IF;
END_IF;
```

11.6.3 Programmable Limit Switch Program Example

The following example illustrates a method to use local logic to perform a programmable limit switch function. This particular programmable limit switch turns on/off an output based upon the current motor position and block within a motion program

Figure 139: Programmable Limit Switch Example



Programmable Limit Switch Local Logic Program

```
(* This example shows a way to cause Axis 1 Digital Output # 1 to *)
(* turn on when Axis 1 Actual Position is between 4000 and 4500, *)
(* but only while the program is in Block #4. This demonstrates *)
(* functionality that is often implemented using a high speed counter *)
(* (HSC) and PRESET's. This example will achieve a resolution of 2 ms.*)

Digital_Output1_1 := 0; (* Digital Output Axis 1 is Off *)
IF Block_1 = 4 THEN (* If current Block number for axis 1 equals 4 *)
  IF Actual_Position_1 > 4000 THEN (* If Act Pos Axis 1 > 4000 *)
    IF Actual_Position_1 < 4500 THEN (* If Act Pos Axis 1 < 4500 *)
      Digital_Output1_1 := 1; (* Digital Output Axis 1 is On *)
    END_IF;
  END_IF;
END_IF;
```

The motion program segment corresponding with the above local logic program is shown below.

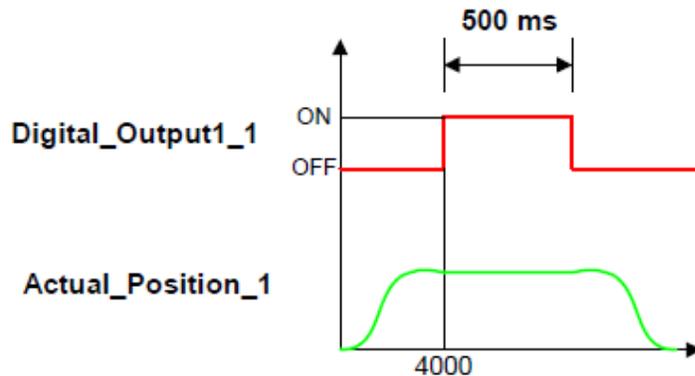
Programmable Limit Switch Example Motion Program Segment

```
Program 1 Axis1 (* Program Number 1 for Axis 1 *)
3: PMOVE 0,ABS,LINEAR (* Move to initial position *)
4: PMOVE 8000,ABS,LINEAR (* Move to position 8000 *)
5: PMOVE -8000,ABS,LINEAR (* Move to position -8000 *)
EndProg
```

11.6.4 Trigger Output Based Upon Position Program Example

The following example illustrates a method to use Local Logic to trigger a timed output based upon the current motor position. The reader should note that the timer implementation uses a counter within the program. The counter counts the number of times the program has been executed since the counter was last reset. Since local logic programs are executed every position loop sample period, the counter time period is based upon this period. This example uses digital servos, which have 2 mSec position loop sample periods. Therefore, the counter will count in 2 mSec increments. For other configurations, consult Chapter 1 for the position loop sample periods. Additionally, Local Logic allows the program to write a variable multiple time within a program. The last state that the variable is in at program completion is the one written to the output (refer to Chapter 12, section on Local Logic Outputs/Commands). This is important in the following program. The second IF-THEN-END_IF block turns the digital output for axis 1 (Digital_Output1_1) on when actual position for axis 1 (Actual_Position_1) is greater than 4000 regardless of the current timer value (P008). However, the last IF-THEN-END_IF block in the program checks the current timer value (P008) and turns the digital output 1 for axis 1 (Digital_Output_1) off if the timer exceeds 500. The application is shown pictorially in Figure 140.

Figure 140: Timer Output Based Upon Position Example



Timer Output Based Upon Position Local Logic Program

```
(* This example shows a way to trigger a timed output, based on *)
(* Axis 1 Actual Position *)
(* When Actual_Position_1 exceeds 4000, turn on Digital_Output1_1 *)
(* for 500 ms *)

IF Block_1 = 4 THEN (* Block 4 in the motion program resets timer *)
  P008 := 0; (* Reset Timer (P008) *)
  Digital_Output1_1 := 0; (* make sure the output is off before move *)
  P009 := 0; (* P009 tracks the digital output state *)
END_IF;

IF Block_1=5 THEN
  IF Actual_Position_1 > 4000 THEN (*Actual_Position_1 exceeds 4000 *)
    Digital_Output1_1 := 1; (* turn the Digital_Output1_1 ON *)
    P009 := 1;
  END_IF;
END_IF;

IF P009 = 1 THEN (* whenever Digital_Output1_1 is ON *)
  P008 := P008 + 2; (* increment the Timer by 2ms *)
  IF P008 > 500 THEN (* when the Timer exceeds 500 ms *)
    Digital_Output1_1 := 0; (* turn Digital_Output1_1 OFF *)
    P009 := 0; (* remember, only the last write in a Local Logic
    sweep actually occurs for the digital output*)
  END_IF;
END_IF;
```

The motion program segment corresponding with the above local logic program is shown below.

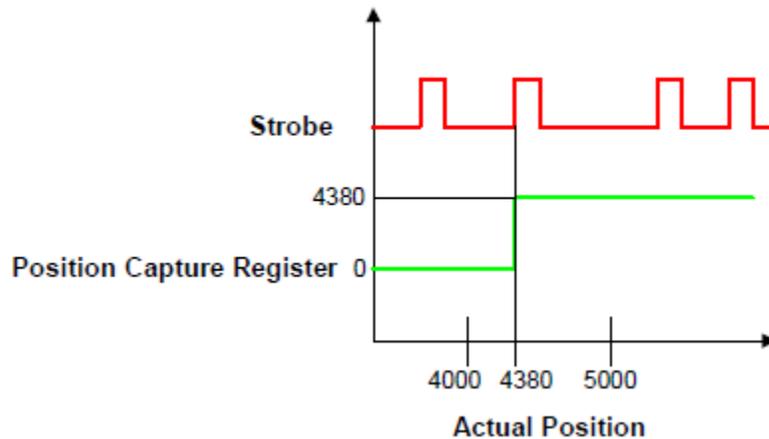
Timer Output Based Upon Position Example Motion Program Segment

```
Program 1 Axis1 (* Program Number 1 for Axis 1 *)
4: PMOVE0, ABS, LINEAR (* Move to initial position *)
5: PMOVE 12000, ABS, S-CURVE (* Move to position 12000 *)
EndProg
```

11.6.5 Windowing Strobes Program Example

The following example illustrates a method to use local logic to perform a windowing strobe function. The example ignores the strobe command unless the current motor position is inside the window (Actual Position > 4000 but less than 5000). If the motor position is inside the aforementioned window, the first strobe occurrence causes the current motor position to be captured within the strobe register. The application is shown pictorially in Figure 141.

Figure 141: Windowing Strobes Example



Windowing Strobes Local Logic Program

```
(* Windowing Strobes- Local Logic Example *)
(* P009 is the Strobe Position read from Strobe1_Position_1 *)
(* CTL13 is used as the Strobe Occurred Flag - must be configured
for Local Logic Control in Hardware Configuration*)

IF Block_1 = 6 THEN
  P009 := 0; (* Initialize P009 to 0 *)
  CTL13 := 0; (* reset strobe occurred flag *)
END_IF;
Reset_Strobe1_1 := 0; (* Keep Reset Strobe bit off by default *)
IF Strobe1_Flag_1 = 1 THEN (* If strobe occurs in "window" range, *)
  IF Actual_Position_1 > 4000 THEN (*set P009 = strobe position *)
    IF Actual_Position_1 < 5000 THEN
      P009 := Strobe1_Position_1;
      CTL13 := 1; (* Set the Strobe Occurred Flag ON. *)
    END_IF;
  END_IF;
  Reset_Strobe1_1 := 1; (* If a strobe occurs, set Reset Strobe bit. *)
END_IF;
```

The motion program segment corresponding with the above local logic program is shown below.

Windowing Strobes Example Motion Program Segment

```
Program 1 AXIS1 (* Program Number 1 for Axis 1 *)
  PMOVE 0,ABS,LINER (* Move to initial position *)
6: DWELL 10
7: PMOVE 10000,ABS,S-CURVE (* Move to position 10000 *)
EndProg
```

Chapter 12: Local Logic Language Syntax

This chapter describes the Local Logic programming language syntax, rules, and language elements. The language uses free-format text-based constructs derived from the IEC 1131 structured text standard. The sections that follow describe the available commands and the command syntax.

12.1 Syntactic Elements

The local logic language syntax is described in the following sections. The syntax is easy to learn and provides a rich feature set that allows the user to accomplish the programming task. Chapter 11 contains many examples that will further aid the reader in understanding the syntax and its application. The first-time user may also wish to consult the section on “Building Your First Local Logic Program” program contained in chapter 10 and the sample programs in the Chapter 11 tutorial as additional aids.

12.1.1 Numeric Constants

The local logic programming language supports decimal, hexadecimal, and binary constants. The DSM treats all constants as 32-bit signed twos-complement integer values. Single underline characters (i.e. 16#7fff_ffff) may be inserted between digits to improve the readability of large numbers.

Decimal constants must be in the range of -2147483648 to 2147483647. Only integer values are supported, therefore constants do not have a decimal point. Thus, as in all integer-based systems the decimal points are implied and the programmer must keep track of them if fractional math is needed.

Examples:

523	Positive decimal constant
-1048	Negative decimal constant
1_745_245	Positive decimal constant with embedded underscores

Hexadecimal (base 16) constants are identified by a 16# prefix and must have a value that can be represented in 32-bits (8 hexadecimal digits). Hexadecimal constants cannot have a sign (+/-) prefix. Hexadecimal digits A-F are not case sensitive, upper or lower case may be used.

Examples:

16#FFFF	Hexadecimal constant
16#7fff_ffff	Hexadecimal constant with embedded underscores

Binary (base 2) constants are identified by a 2# prefix and must have a value that can be represented in 32-bits (32 binary digits). Binary constants cannot have a sign (+/-) prefix.

Examples:

2#1010 Binary constant

2#11111110_11101101_10111110_11101111 Binary constant with embedded
underscores

A local logic program may have a maximum of 50 unique constants whose value is greater than 2047 or less than -2048. If a local logic program declares more than 50 unique constants, the build process generates an error. Most programs use much less than 50 constants, so this is generally not a constraint.

12.1.2 Local Logic Variables

The local logic language supports a number of predefined variables that allow access to the DSM I/O data, CTL bits, and other status and control information. A detailed description of the local logic variable set is contained in chapter 13 Each variable has two attributes, size and direction. Local Logic variables range in size from 1 Bit (Bit Operands) to 64 bits.

All Local Logic parameter registers are one of the following types.

- Double integer variables hold signed 32 bit values (-2147483648 to 2147483647). There are 256 Parameter registers (P000-P255).
- Long integer variables hold signed 64 bit values (+/-9.22 x 1018). The long integer variables are unique in that they may only be used for the result of a multiply or as the numerator in a divide or modulus operation. There are 8 long integer registers (D00-D07).

All Local Logic variables have one of the following directional attributes.

- Read-only variables may not be used as the destination of an assignment operation.
- Write-only variable may only be used as the destination of an assignment statement.
- Read-write variables may be used as a source or destination.

Refer to Chapter 13 for a list of all the Local Logic variable size and direction attributes.

12.1.3 Local Logic Statements

The Local Logic language supports two kinds of statements: Assignment and Conditional. A Local Logic program supports 150 statements. The Local Logic check block will generate an error message when the 150 line limit is exceeded. Warnings are issued when the Local Logic program exceeds 100 lines. The warning message can be turned off with the #pragma directive. Reference the #pragma sections for additional details. Semicolons separate program statements.

Local Logic Assignment Statements

Assignment statements permit simple arithmetic and bitwise operations to be performed with the result being assigned to a variable. An assignment statement has the following format.

```
<destination> := <expression>;
```

The <destination> operator may consist of any read-write or write-only variable. The <expression> may be a simple constant or variable, a mathematical or bitwise logical operation on two operands, an ABS function, or a bitwise NOT operation. Write-only variables can not be the expression for an assignment operation.

Examples:

P032 := Strobe1_Position_1 + 5000;	← This construct is okay.
P001 := ABS(Analog_Input1_1);	← This construct is okay.
Reset_Strobe1_1 := BWNOT Strobe1_Flag_1;	← This construct is okay.
P040 := 2#11111010_1011000;	← This construct is okay.
P011 := 3 * Strobe1_Position_1 + 20;	← This construct is ILLEGAL – too many operations.

If complex operations are required, perform the operation using a series of steps that use parameter registers to store intermediate results.

Examples:

To set Velocity_Loop_Gain_2 equal to $(1+75000/\text{Actual_Velocity_2})$, the programmer uses a series of statements similar to the following...

```
P012 := 75000 / Actual_Velocity_2;
Velocity_Loop_Gain_2 := 1 + P012;
```

The build process will issue a warning if a Boolean variable is used as the destination for an expression containing non-Boolean variables or a constant whose value is not zero or one. A warning is generated because the DSM will assign the Boolean variable the value of the least significant bit of the expression.

Local Logic Conditional Statements

Conditional statements permit conditional code execution based on simple relational and bitwise logical operations. A conditional statement has the following format.

```
IF <expression> THEN
    Local Logic Statements
END_IF;
```

The <expression> may consist of a constant, a variable, a relational or bitwise logical operation on two variables, or a bitwise complement of a constant or variable. Write-only variables are not allowed in the expression. If the relational expression is true, or if a bitwise operation, variable or constant has a non-zero value, the Local Logic statements in the body of the IF statement are executed. Any number of program statements may appear in the body of an IF statement (subject to the total limit). Each IF-THEN statement must have an accompanying END_IF.

Examples:

IF P226 THEN	← This construct is okay.
IF CTL_1_to_32 BWAND 2#1010 THEN	← This construct is okay.
IF Strobe1_Level_1 = TRUE THEN	← This construct is okay.
IF BWNOT P100 THEN	← This construct is okay.
IF BWNOT P001 <> P002 THEN	← This construct is ILLEGAL – too many operations.

If statements may nest up to 8 levels deep. When counting the number of program statements, the IF-THEN and END_IF statements count as two separate statements.

Table 54: Valid Operators

Statement Type	Valid Operators
Conditional	Relational <, >, <=, >=, <>, =
	Bitwise Logical BWAND, BWOR, BWXOR, BWNOT
Assignment	Arithmetic +, -, /, *, MOD
	Bitwise Logical BWAND, BWOR, BWXOR, BWNOT
	Abs Function ABS ()

12.1.4 Whitespace

Blanks, end-of-lines, and tabs are considered whitespace. Whitespace is ignored, except when used to separate adjacent syntactic elements, and may be used to improve program readability by the use of indention and blank lines.

12.1.5 Comments

Comments may be used to add information to the program that is ignored by the Local Logic program execution engine. Two types of comments are supported.

The (* character pair introduce a normal comment, which terminates with the *) character pair. These comments may appear anywhere whitespace can, for example within or following a local logic statement, alone on a line, or spanning several lines. These comments do not nest.

The // character pair introduces a single line comment. All text following the // to the end of the line is ignored by the Local Logic execution engine.

Note: *You should be aware that one can enter a local logic program and inadvertently comment out the code that one wants to execute. The common scenario that causes this to happen is as follows:*

```
(* This example shows a way to cause Axis 1 Digital Output # 1 to *)
(* turn on when Axis 1 Actual Position is between 4000 and 4500, *)
(* but only while the program is in Block #4. This demonstrates *)
(* functionality that is often implemented using a high speed counter *)
(* (HSC) and PRESET's. This example will achieve a resolution of 2 ms. )
Digital_Output1_1 := 0; (* Digital Output Axis 1 is Off *)
IF Block_1 = 4 THEN (* If current Block number for axis 1 equals 1 *)
```

In the above code segment, the end comment structure, line shown in bold/italic for illustrative purpose, is incorrect because the asterisk in the close comment structure is absent. The error causes the following line to be considered a comment as well. Thus, the statement Digital_Output_1:=0 is considered a comment and not executed. The color scheme within the Local Logic editor can be very useful to help find these types of problems. The coloring scheme by default will color the comments a different color than the programming statements. Thus, the user will have a visual method to help find these errors. Please consult chapter 2 for information on how to change the default color scheme for the editor.

12.1.6 PRAGMA Directive

The #pragma directive is used to configure the Local Logic parser. The directive is NOT required for the parser to operate. However, if the user wishes to turn off warning messages the #pragma directive allows this to occur. The #pragma directive MUST be the first line of the program. Additionally, no white space should be present prior to the directive.

To turn ALL Local Logic warnings off, issue the following command:

```
#pragma errorsonly 1  
  
or  
  
#pragma errorsonly ON
```

To turn warning messages back ON either delete the directive or change the directive as follows:

```
#pragma errorsonly 0  
  
or  
  
#pragma errorsonly OFF
```

12.1.7 Local Logic Keywords and Operators

The following keywords and operators have special significance in the Local Logic programming language. Keywords are case-sensitive and use only upper-case letters. These are discussed in further detail in the following sections.

Table 55: Local Logic Keywords

ABS	TRUE	+	>
BWAND	FALSE	-	<
BWOR	IF	/	>=
BWXOR	THEN	*	<=
BWNOT	END_IF	16#	=
ON	MOD	2#	<>
OFF	;	:=	

12.2 Enabling and Disabling Local Logic

Local Logic execution is enabled using a host controller Q bit. For example if a DSM is configured with a starting %Q reference of %Q0001 then the Local Logic enable bit is %Q0002 (beginning reference + offset of 1). The Local Logic program name must be specified in the hardware configuration software and the field for Local Logic Enabled/Disabled must be set to Enabled. Refer to Chapter 10 for a detailed description of configuring Local Logic in hardware configuration.

Local Logic executes only while the host controller is in RUN mode. If The host controller is switched to STOP mode or if the enable Local Logic Q bit is turned off, Local Logic execution is halted and all Digital Outputs, Control bits (Jog, Feedhold, Strobe Resets, Follower Enable) and CTL bits that are under the control of Local Logic are disabled.

Attempting to execute Local Logic in the First CPU Sweep will result in an error being reported. For example, switching from Stop Mode to Run Mode while the Local Logic Enable bit is on will generate an error and the Local Logic program will not execute. Toggle the Enable Q bit to run the Local Logic program.

Note: *The Local Logic Engine will not run if any custom Local Logic functions are enabled via the Advanced Parameters in Hardware Configuration. The custom function will normally not be available and is developed for application specific use only by Emerson.*

12.3 Local Logic Outputs/Commands

DSM command bit outputs (**Jog**, **Feedhold**, **Follower Enable** and **Strobe Resets**) are **OR**'ed between the host controller command and the Local Logic command. Therefore, either the host controller or Local Logic can control them i.e. the command bit output is active if either the host controller or Local Logic has turned it on.

AQ commands are accepted on a last-write basis. For example, if both the host controller (%AQ) and Local Logic issue a Follower Ratio command the last value written will be active.

DSM faceplate digital outputs (real outputs switched by the DSM) are individually configurable to be either under Local Logic control or host controller control, but not both simultaneously. Refer to Chapter 14 for a detailed description on configuring the Digital Outputs.

Local Logic digital outputs, immediate commands and command bits are updated at the end of each Local Logic Sweep (refer to Chapter 13 for a list of the command Variables and digital output variables). Therefore if the Local Logic program writes to the same command variable or digital output variable multiple times in the same sweep, the last value written will be the effective command.

For example, the sample code below shows the Jog_Plus variable, the Strobe_Reset variable and the Follower_Ratio being written multiple times within the same sweep. In all cases the final value written is the active value.

Example:

```
Jog_Plus_1 := TRUE; (* Turn on Jog Plus for Axis 1 *)
Strobe_Reset1_3 := 0; (* Turn off the Strobe 1 reset bit for Axis 3 *)
(* Some more code here *)
Follower_Ratio_A_1 := 10; (* Set the Follower Ratio A for Axis 1 to 10 *)
Jog_Plus_1 := FALSE; (* Turn off Jog Plus for Axis 1 *)
Strobe_Reset1_3 := 1; (* Turn on the Strobe 1 reset bit for Axis 3 *)
Follower_Ratio_A_1 := 20; (* Set the Follower Ratio A for Axis 1 to 20 *)
```

For each of the output commands shown above, the last value written is acted upon by the Logic Engine at the end of each sweep. Thus Jog_Plus_1 is turned OFF, Strobe_Reset1_3 is turned ON and Follower_Ratio_A_1 is set to 20.

12.4 Local Logic Arithmetic Operators

The Local Logic language contains familiar constructs to perform basic signed integer arithmetic computations. The language supports 32-bit arithmetic in the Local Logic program and limited use of 64/32-bit arithmetic. All operations require two operands except for the ABS function, which returns the absolute value of a variable or numeric constant.

Table 56: Arithmetic Operators

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Integer Division
MOD	Modulus
ABS	Absolute Value

Arithmetic expressions may only be used in assignment statements with one operation per statement.

The arithmetic operations do not require data type conversion functions since the motion module automatically does this operation.

12.4.1 Operator +

Adds source1 to source2 and stores the result in destination

Syntax

```
destination := source1 + source2;
```

The + operator syntax has these parts:

Part	Description
Destination	Any writeable local logic variable except Dxx registers.
source1	Any readable local logic variable/constant except Dxx registers.
source2	Any readable local logic variable/constant except Dxx registers.

Overflow – Set if the result of an addition is greater than 2,147,483,647 or less than -2,147,483,648. The Module_Status_Code is set to a value of 16#0095, which is a status-only error.

12.4.2 Operator -

Subtracts source2 from source1 and stores the result in destination

```
destination := source1 - source2;
```

The - operator syntax has these parts:

Part	Description
Destination	Any writeable local logic variable except Dxx registers.
source1	Any readable local logic variable/constant except Dxx registers.
source2	Any readable local logic variable/constant except Dxx registers.

Overflow – Set if the result of a subtraction is greater than 2,147,483,647 or less than -2,147,483,648. The Module_Status_Code is set to a value of 16#0095, which is a status-only error.

Remarks

The - operator may not be used as a unary operator except with a decimal (base 10) constant (e.g. P001 := -P003; is illegal). To negate a variable, subtract it from zero, e.g. P001 := 0 - P003;.

12.4.3 Operator *

Performs a signed multiply of source1 and source2 generating a signed 64-bit result. The result may be stored to a 32-bit or 64-bit destination.

Syntax 1

destination := source1 * source2;

Syntax 2

double destination := source1 * source2;

The * operator syntax has these parts:

Part	Description
Destination	Any writeable logic variable
double destination	Any of the 64-bit local logic parameter variables (Dxx registers).
source1	Any readable local logic variable/constant except Dxx registers.
source2	Any readable local logic variable/constant except Dxx registers.

Overflow – Never set.

Remarks

If the result is assigned to a 32-bit variable, the least significant 32-bits are stored. Any excess is truncated.

The second syntax may be used for multiplication operations where the result will fall outside the range of +/- 2 billion.

12.4.4 Operator MOD

The MOD operator returns the remainder resulting from the signed integer division of source1 by source2. A double precision (64-bit) parameter register may be used as the numerator.

Syntax 1

destination := source1 MOD source2;

Syntax 2

destination := double source1 MOD source2;

The MOD operator syntax has these parts:

Part	Description
Destination	Any writeable local logic variable except Dxx registers.
source1	Any readable local logic variable or numeric constant.
double source1	Any of the 64-bit local logic parameter variables (Dxx registers).
source2	Any readable local logic variable/constant except Dxx registers.

Overflow - See **remarks** below.

Remarks

In case of a divide by zero, the Module_Status_Code is set to 16#2093. In case of a divide overflow, the Module_Status_Code is set to 16#2094.

The modulus (remainder) is calculated by performing an integer division, therefore the MOD operator has the same error conditions as the divide operator.

A divide overflow occurs when the quotient of a divide operation cannot be correctly be represented as a signed 32-bit value. This can only occur when using a double operand as the numerator. A divide by zero occurs when the denominator of the divide has a value of zero.

A divide overflow or divide by zero are Stop Fast errors. Local Logic is immediately aborted, and motion is aborted by setting the servo velocity command to zero.

12.4.5 Function ABS

The ABS function returns the unsigned magnitude of the variable or constant parameter.

Syntax

destination := ABS(parameter);

The ABS operator syntax has these parts:

Part	Description
Destination	Any writeable local logic variable except Dxx registers.
Parameter	Any readable local logic variable/constant except Dxx registers.

Overflow – Set if the operand has a value of $-2,147,483,648$. The Module_Status_Code is set to a value of 16#0096, which is a status-only error.

12.5 Local Logic Bitwise Logical Operators

All logical operations are performed on a bit-by-bit basis, for example the result of a BWAND operation is composed of 32 and operations between each of the corresponding bits of the operands. The logic operators are prefixed with 'BW' to highlight the fact that they are not Boolean operators.

Table 57: Bitwise Logical Operators

Operator	Meaning
BWAND	Bitwise Logical AND
BWOR	Bitwise Logical OR
BWXOR	Bitwise Logical Exclusive OR
BWNOT	Bitwise Logical NOT (one's-complement)

Expressions using bitwise logical operators may be used in assignment or conditional statements. Only one bitwise logical operator may be used per expression.

12.5.1 Operator BWAND

Performs a bitwise and of source1 and source2.

Syntax 1

destination:= source1 BWAND source2;

Syntax 2

IF source1 BWAND source2 THEN

The BWAND operator syntax has these parts:

Part	Description
Destination	Any writeable local logic variable except Dxx registers.
source1	Any readable local logic variable/constant except Dxx registers.
source2	Any readable local logic variable/constant except Dxx registers.

Remarks

Syntax 1 is used for assignment; syntax 2 is used in a conditional evaluation.

12.5.2 Operator BWOR

The BWOR operator returns the bitwise or on source1 and source2.

Syntax 1

destination:= source1 BWOR source2;

Syntax 2

IF source1 BWOR source2 THEN

The BWOR operator syntax has these parts:

Part	Description
Destination	Any writeable local logic variable except Dxx registers.
source1	Any readable local logic variable/constant except Dxx registers.
source2	Any readable local logic variable/constant except Dxx registers.

Remarks

Syntax 1 is used for assignment, syntax 2 is used in a conditional evaluation.

12.5.3 Operator BWXOR

The BWXOR operator returns the bitwise exclusive or of source1 and source2.

Syntax 1

destination: = source1 BWXOR source2;

Syntax 2

IF source1 BWXOR source2 THEN

The BWXOR operator syntax has these parts:

Part	Description
Destination	Any writeable local logic variable except Dxx registers.
source1	Any readable local logic variable/constant except Dxx registers.
source2	Any readable local logic variable/constant except Dxx registers.

Remarks

Syntax 1 is used for assignment; syntax 2 is used in a conditional evaluation.

12.5.4 Operator BWNOT

The BWNOT operator returns the one's complement of the source parameter.

Syntax 1

destination:= BWNOT source;

Syntax 2

IF BWNOT source THEN

The BWNOT operator syntax has these parts:

Part	Description
Destination	Any writeable local logic variable except Dxx registers.
source1	Any readable local logic variable/constant except Dxx registers.

Remarks

Syntax 1 is used for assignment; syntax 2 is used in a conditional evaluation.

12.6 Comparison Operators

The comparison operators form a relational assertion between two operands. The comparison expression evaluates the conditional based on the operands signed integer value.

Table 58: Relational Operators

Operator	Meaning
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equal to
<>	Not Equal to

Comparison operators may only be used as expressions in conditional statements, and only one comparison operator may be used per expression.

IF source1 ComparisonOp source2 THEN

Comparison operators have these parts:

Part	Description
source1	Any readable Boolean or 32-bit local logic variable or numeric constant.
source2	Any readable Boolean or 32-bit local logic variable or numeric constant.
ComparisonOp	Any relational operator / Logical operator.

Remarks

The following table contains a list of the comparison operators and the conditions that determine whether the result evaluates to True or False:

Table 59: Local Logic Comparison Operators

Relational	Operator	True if	False if
Less than	<	source1 < source2	source1 >= source2
Less than or equal to	<=	source1 <= source2	source1 > source2
Greater than	>	source1 > source2	source1 <= source2
Greater than or equal to	>=	source1 >= source2	source1 < source2
Equality	=	source1 = source2	source1 <> source2
Inequality	<>	source1 <> source2	source1 = source2
Null Operator (IF Variable THEN...)	N/A	Variable is Non-Zero	Variable is Zero
Bitwise Logical Operators	BWAND, BWOR, BWXOR, BWNOR	Result is Non-Zero	Result is Zero

12.7 Local Logic Runtime Errors

12.7.1 Overflow Status

Some arithmetic operations may have results that cannot be correctly represented as a signed 32-bit value. An example is shown in the following code segment.

```
P001 := 16#7FFF_FFFF; // (1) P001 ← 2,147,483,647 (maximum positive 32 bit value)
P003 := P001 + 1; // (2) ! Overflow !
```

In the first line, P001 is loaded with 2,147,483,647, the largest value that can be represented as a 32 bit signed two's-complement value. In the second line, one is added to that value. The total, represented in hexadecimal, is 16#8000_0000. This value, interpreted as a 32-bit signed two's complement number represents the negative value - 2,147,483,648, not the positive value 2,147,483,648! In many situations, this result would be unexpected and have undesirable effects in subsequent program statements.

Three variables are available to the Local Logic program to detect overflows. The Overflow variable is a read-write Boolean variable available only to the local logic program (refer to Chapter 13, Section on "Local Logic System Variables"). When an overflow error occurs, and the Overflow variable is not cleared before the end of the Local Logic sweep, the DSM's Module Status Code %AI word (local logic variable Module_Status_Code) is set. The error code indicates the type of overflow and the Module Error Present %I bit (local logic variable Module_Error_Present) is set. The Module Status Code %AI word and the Module Error Present %I bit are not set until the current Local Logic sweep has finished executing. In contrast, the Overflow variable is set immediately following an instruction which causes an overflow. The Local Logic program may clear the Overflow variable by assigning it a value of zero. The Module Status Code must be cleared by the host controller by setting the module's Clear Error %Q bit.

Overflow and computation errors are Status Only errors with two exceptions. A divide by zero or divide overflow (the quotient cannot be represented in 32 bits) are Stop Fast errors. In the case of status only errors, Local Logic processing and path generation continue normally. A stop fast error will cause Local Logic processing to be aborted before proceeding to the next instruction, and any motion will be aborted by setting the servo velocity command to zero. Note: Clearing the Overflow variable has no effect on Stop Fast errors.

Refer to Table 63 for a listing of all Runtime Local Logic error codes.

Divide By Zero

The Logic Engine flags Divide By Zero operations as a Fast Stop Error, since the result of the operation is undefined. Local Logic execution and servo motion is halted. An error code 16#2093 is reported in the module status code and 16#2x9A in the Per-Axis error codes if the drives were enabled.

Watchdog Timeout Warning / Error

Local Logic programs are constrained to complete execution within 300 microseconds in the Logic Engine. This is to allow sufficient processing time in the module for Path Generation and other tasks. Refer to Appendix E for a detailed listing of the execution times for all valid Local Logic operations. The user can compute the execution time required for a given program using the data tables supplied in Appendix E. The Logic Engine reports a warning (Status Only) error code if the execution time takes more than 275 microseconds but less than 300 microseconds. A Fast Stop error is generated if the program execution time exceeds 300 microseconds. Refer to Table 63 for a list of the runtime warnings and error codes.

Note: *Local Logic execution is halted if there are any Local Logic Fast Stop errors (see Table 61) and all Digital Outputs, Control bits (Jog, Feedhold, Strobe Resets, Follower Enable) and CTL bits that are under the control of Local Logic are disabled. Local Logic execution resumes from the start when the user clears the error (via the error clear %Q bit).*

12.8 Local Logic Error Messages

12.8.1 Local Logic Build Error Messages

The local logic program build process communicates the build status through the local logic, editor error log window. In the event an error occurs, the build process reports the error and attempts to continue the build process.

Error messages generated by the local logic build process fall into three categories; syntax errors, parse errors, and parse warnings.

Parser error messages have several common elements.

Filename (Line): [Severity] [error message]

Filename is the filename of the current file being built.

Line is the line number in the file that the error was detected on.

Severity describes error severity. Errors prevent a binary creation. Warnings are informational.

Error Message is a short, general description of the error

12.8.2 Local Logic Syntax Errors

The build process enforces the local logic syntax. If the source program fails to meet this criterion, the build process reports a syntax error. The error message identifies the error as a syntax error. The syntactic element type found followed one or more of the syntactic elements the parser was expecting is contained within the error message. It is common for syntax errors to actually be reported on a line following the line with the actual error. Missing semi-colons are a typical example.

Example:

```
scratch.llp (3): Error :syntax error
```

```
actual: IF expecting: ;
```

In this case, line 2 is actually missing the semicolon. Since the semi-colon may actually follow on another line, the parser does not report the error until it sees a meaningful syntactic element that isn't a semi-colon.

Because of their nature, a single syntax error can cause “cascading errors.” Correcting one syntax error may eliminate several syntax error messages. To avoid confusion, when debugging Local Logic programs with syntax errors, correct the first error and rebuild the program to refresh the list of errors before proceeding.

12.8.3 Local Logic Parse Errors

Parse errors occur when the program syntax is correct, but there is a semantic problem. For example, it is invalid to assign a value to a double precision variable except as the result of a multiplication operation.

Examples:

Error (P203) Invalid assignment to Double precision var: D00

In this case the error message is followed by a string that identifies the token that caused the error. A list of Parse errors and typical causes follows:

Table 60: Local Logic Parse Errors

Error Number	Error Description
(P200)	Undefined identifier The program contains an unrecognized variable or keyword. Check spelling and command syntax.
(P201)	Parameter register must be in range of P000 - P255 This error is generated when the program specifies an undefined parameter register, for example P278.
(P202)	CTL variable must be in range CTL01 - CTL32 This error is generated when the program specifies an undefined CTL variable, for example CTL35.
(P203)	Invalid assignment to Double precision var

Error Number	Error Description
	The program has attempted an invalid usage of one of the double precision registers. Double precision registers may only be assigned values as the result of a multiply operation.
(P204)	Invalid use of Double precision var The program has attempted an invalid usage of one of the double precision registers. Double precision registers in expressions may only be used as the divisor in a divide or MOD operation.
(P205)	Assignment to read-only variable The program has attempted to assign a value to a read-only variable.
(P206)	Attempt to read write-only variable The program has attempted to use a write-only variable as one of operands in an arithmetic, logic, or relational expression.
(P207)	Subscripted variables are not supported Variables of the form Data_Table_Int[xx] are not supported. Data table operations require the use of the Data_Table_Ptr variable.
(P208)	Identifier name exceeds 50 chars The program has used attempted to reference a variable with an identifier length in excess of 50 characters.
(P209)	Double Precision register must be in range D00 - D07 This error is generated when the program specifies an undefined double precision register, for example D08.
(P220)	Hexadecimal constants must be in range of 16#0 - 16#FFFFFFF The program has defined a hexadecimal constant that cannot be represented in 32 bits
(P221)	Binary constants must be in range of 0 to (2 ³²)-1 The program has defined a binary constant that cannot be represented in 32 bits
(P222)	Integer constants must be in range of -2147483648 to 2147483647 The program has defined a decimal constant that cannot be represented in 32 bits
(P223)	Constant table overflow A program can contain a maximum of 50 unique constants greater than 2047 or less than -2048 (i.e. numbers that cannot be represented in less than 12 bits). A program may contain any number of constants in the range of -2048 to 2047.
(P230)	IF nesting limit of 8 levels exceeded. IF statements cannot nest more than 8 levels deep.
(P231)	Illegal term in IF statement The program has an arithmetic operator in an IF statement.
(P232)	Missing END_IF statement There is an IF statement that is missing a matching END_IF statement. This error is only detected at the end of the program.
(P233)	Unmatched END_IF encountered An END_IF statement has been found that doesn't have a corresponding IF statement.

Error Number	Error Description
(P240)	Assignment to constant The program has attempted to use a constant as the destination of an assignment statement.
(P241)	Invalid operator, assignment expected Another operator was encountered where the assignment operator (:=) was expected.
(P242)	Relational operator not allowed in assignment statement A comparison operation was attempted in an assignment statement. An assignment based on a relational may be performed by assigning a Boolean value in an IF statement.
(P260)	Invalid logic operator. Use BWAND, BWOR, BWXOR, or BWNOT - <operator> The program has used AND, OR, XOR, or NOT keywords, rather than BWAND, BWOR, BWXOR, or BWNOT, respectively.
(P280)	Instruction limit exceeded, max 150 A local logic program may be a maximum length of 150 statements. This error is reported if the program exceeds that length.
(P290)	Address out of range in direct memory access reference A direct memory variable has specified an invalid offset.
(P291)	Invalid direct memory address variable An invalid direct memory variable has been specified.
(P292)	Direct memory access var requires subscript The program has referenced a direct memory variable without specifying an offset
(P293)	Maximum error count exceeded. The Local Logic parser will report a maximum of 30 errors. When that limit has been exceeded this message is displayed and no further errors are reported.
298-(P299)	Internal Error. Contact Emerson Technical Support. If the parser reports error 298 or 299 for a user program, please notify Emerson technical support. Provide a copy of the program and error log.
(P300)	Parse directives must precede any executable statements. #pragma directives must appear before any executable statements in the Local Logic program block.
(P301)	Invalid directive option The specified #pragma directive is not recognized by the Local Logic Parser.
(P302)	Invalid directive parameter An invalid argument to the #pragma errors only directive was specified. The argument must be 1, ON, 0, or OFF.

12.8.4 Local Logic Parse Warnings

Parse warnings are generated for conditions that may have unexpected results or indicate a possible oversight in the Local Logic Program.

Table 61: Local Logic Parse Warnings

Error Number	Error Description
(P400)	Assignment to binary variable may result in loss of data This message is generated when a Boolean variable is assigned from a non-Boolean variable or constant, or an expression containing non-Boolean variables.
(P410)	Check instruction execution time This warning is generated for programs exceeding 100 statements. While there is a maximum instruction limit of 150 statements, it is possible to write a Local Logic program that takes too long to execute. For instruction times, refer to appendix A in the PACSystems CPU Reference Manual, GFK-2222 or Series 90-30 CPU Reference Manual, GFK-0467
(P481)	Obsolete syntax: function parameter requires parentheses The parameter of an ABS function call is not enclosed in parentheses.
(P482)	Unexpected end of program: unclosed comment A comment initiated with the “*” character pair was not closed when the end-of-program was encountered.
(P483)	Nested comments This warning is generated if a Local Logic program has defined comment text within another comment.
(P490)	Program contains no executable statements The program contains only white space and/or comments.

12.8.5 Local Logic Download Error Messages

The following errors may be reported in the Module Status Code when a Local Logic program is downloaded into the module.

Table 62: Local Logic Configuration Error Codes

Error Code (Hexadecimal)	Response	Description	Error Type
0A	System Error	Invalid Digital Output Configuration	Module
0B	System Error	Invalid CTL Bit Configuration	Module

Note: Refer to Chapter 14 for a detailed description on configuring CTL bits and Digital Outputs for Local Logic.

Table 63: Local Logic Preprocessing Error Codes

Error Code (Hexadecimal)	Response	Description	Error Type
F0A0	System Error	Local Logic Program Header Error	Module
F0A1	System Error	Local Logic Program Terminator Error	Module
F0A2	System Error	Local Logic Program Constant Header Error	Module
F0A3	System Error	Local Logic Program Constant Terminator Error	Module
F0A4	System Error	Local Logic Program Constant Pointer Error	Module
F0A5	System Error	Local Logic Program Compiled Code Limit Exceeded	Module
F0A6	System Error	Local Logic Program Unmatched IF_THEN Error	Module
F0A7	System Error	Local Logic Program Unmatched END_IF Error	Module
F0A8	System Error	Local Logic Program Nesting Limit Exceeded	Module
F0A9	System Error	Local Logic Program Scan Error	Module
F0AA	System Error	Local Logic Program Reserved Class Error	Module
F0AB	System Error	Local Logic Program Invalid Parameter Register	Module
F0AC	System Error	Local Logic Program Invalid Double Precision Register	Module
F0AD	System Error	Local Logic Program Digital Output Error	Module
F0AE	System Error	Local Logic Program CTL Bit Error	Module
F0B0	System Error	Local Logic Program Invalid Primary Operator	Module
F0B1	System Error	Local Logic Program Invalid Secondary Operator	Module
F0B2	System Error	Local Logic Program Invalid Secondary Source	Module
F0B3	System Error	Local Logic Program Invalid Primary Source	Module
F0B4	System Error	Local Logic Program Invalid Source	Module
F0B5	System Error	Local Logic Program Source Write Only Error	Module
F0B6	System Error	Local Logic Program Direct Memory Address Error	Module
F0B7	System Error	Local Logic Program Invalid Destination	Module
F0B8	System Error	Local Logic Program Destination Read Only	Module

12.8.6 Local Logic Runtime Errors

The following errors and warnings may be reported when a Local Logic program is executed in the module.

Table 64: Local Logic Runtime Error Codes

Error Code (Hexadecimal)	Response	Description	Error Type
91	Fast Stop	Local Logic Program System Halt Commanded	Module
92	Fast Stop	Local Logic Execution Time Limit Exceeded	Module
93	Fast Stop	Local Logic Divide By Zero	Module
94	Fast Stop	Local Logic Divide Overflow	Module
95	Status Only	Local Logic Add/Subtract Overflow	Module
96	Status Only	Local Logic Absolute value (ABS) overflow	Module
97	Status Only	Local Logic Execution Time Limit Warning	Module
98	Status Only	Local Logic Execute on First Sweep Error	Module
99	Status Only	Local Logic Invalid Program Name or Not Enabled in Hardware Configuration	Module
9A	Fast Stop	Local Logic Stop Error	Per-Axis

Chapter 13: Local Logic Variables

This chapter describes the local logic variable types, identifies the local logic system variables, double precision 64-bit registers, the local logic user data table, and digital outputs/CTL variables.

13.1 Local Logic Variable Types

Local Logic accesses the motion controller variables and parameter registers using pre-defined variable names. Refer to Table 15H13-1 through Table 20H13-6 for a complete listing of all Local Logic variables.

Examples:

```
IF Actual_Position_2 > 5000 THEN ...;
```

```
IF Strobe1_Level_2 = ON THEN ...;
```

Storing values to variables is performed by using the “:=” assignment operator:

Examples:

```
Torque_Limit_2 := 8500; (* Set Torque Limit to 85% *)
```

```
Position_Loop_TC_1 := 2500 / Actual_Velocity_1;
```

Local Logic variables are broken down into two categories: Global Variables and Per-Axis Variables. There are four sets of axis variables (Axis1 - Axis4). Each set of variables is subdivided into **Control variables**, **Status variables** and **Faceplate I/O** (refer to Table 65 through Table 70). A description of the terms used in the Variable Tables follows:

Variable Attribute

The attribute for each Local Logic variable is listed in Table 65 through Table 70. Variables can be **Read-Only**, **Write-Only** or **Read-Write**. The Parser reports an error if the user attempts to write to a Read-Only variable or read from a Write-Only variable.

Variable Size

Local Logic variables range in size from 1 bit (Bit Operands) to 64 Bits (for the Double Precision Dxx registers). Refer to Table 69 through Table 70 for a listing of the size of each Local Logic variable. Attempting to write a value larger than a given variable size will result in the value being truncated. For example, if the result of a math operation is 32 bits long and is assigned to a 16-bit variable only the low 16 bits will be stored. The Parser reports a warning if a Bit Operand is used as the destination variable in a non-Boolean Math operation (only the least significant bit of the result would be stored).

Note: *The AQ command variables (Torque Limit, Velocity Loop Gain, Follower Ratio, Position Increment and Position Loop Time Constant) may have an allowed range that is smaller than the Local Logic variable size. The module reports a warning error code and rejects any invalid values if the program attempts to write a value outside the valid range of an AQ command. Refer to Chapter 4 for a description of the allowed %AQ command ranges.*

Variable Sign

Local Logic variables that are less than 32 bits long are either Signed or Unsigned (except Bit Operands, which are always Unsigned). All Math/Logic operations in the Logic Engine are signed 32 bit operations (except the 64 bit signed Divide and Modulus operations). Signed variables that are less than 32 bits long are automatically sign extended to 32 bits when they are loaded by the Logic Engine. Unsigned variables are not sign extended. Thus the Logic Engine handles all data conversion and limit checking automatically.

13.2 Local Logic System Variables

The First_Local_Logic_Sweep, Overflow and System_Halt variables are used exclusively in the Logic engine and are described below.

13.2.1 First_Local_Logic_Sweep Variable

The First_Local_Logic_Sweep variable is a Read-Only Bit Operand (refer to Table 69). It is set by the Logic Engine during the first execution sweep when Local Logic is enabled and the host controller is in RUN mode. It is reset to zero for subsequent sweeps. Thus it can be used in the Local Logic program to initialize some variables. For example, the code below initializes some parameter registers and Control variables in the first sweep using the First_Local_Logic_Sweep variable.

```
IF First_Local_Logic_Sweep THEN (* If it's the First execution sweep then *)
    P001 := 0; (* Initialize P001 to 0 *)
    P015 := 1000; (* Initialize P015 to 1000 *)
    Velocity_Loop_Gain_1 := 20; (* Set the Velocity Loop Gain for Axis 1 to 20 *)
END_IF;
```

13.2.2 Overflow Variable

The Overflow variable is a Read-Write Bit Operand (refer to Table 69). It is set by the Logic Engine when an Addition, Subtraction or Absolute value (ABS) overflow occurs. A warning error code is also reported in the Module Status Code if the Overflow flag is set and an overflow error occurs (refer to Chapter 12). Note that the user can prevent Add/Subtract/ABS overflow warnings from being reported by setting the Overflow variable to zero at the end of the Local Logic program. Similarly the user can test for Overflow errors within the Local Logic program by reading the Overflow variable and performing some appropriate action. The Overflow variable is cleared under the following circumstances:

1. It is automatically cleared when Local Logic starts running, before the first execution sweep.
2. It can be cleared by the user in the Local Logic program (by setting Overflow := 0;).
3. It is cleared when the user toggles the error clear Q bit.

13.2.3 System_Halt Variable

The System_Halt variable is a **Write-Only** Bit Operand (refer to Table 69). If the Local Logic program writes a 1 to the System_Halt variable servo motion and Local Logic execution is halted. An error code is also reported in the Module Status Code (refer to Chapter 12). Thus the System_Halt variable can be used to trap for fatal error conditions and perform error recovery. The sample code below shows a possible scenario in which the System_Halt variable might be used:

```
IF Overflow THEN          (* Trap for an overflow *)
    System_Halt := TRUE;   (* Halt Local Logic Execution and Servo Motion *)
END_IF;
```

13.3 Double Precision 64 Bit Registers

Local Logic provides eight 64-bit registers (D00-D07) in addition to the 255 32-bit registers (refer to Table 69). This is to allow the user to store the result of multiplying two 32-bit numbers in a Dxx register and then perform a Divide/Modulus operation on the result.

Thus the 64 bit registers may be used under the following circumstances:

1. As the Destination register for a multiply operation.
2. As the Dividend (numerator) in a Divide/Modulus operation.

The Parser will flag an error if it is used in other operations. Example code for the use of the Dxx registers is shown below:

```
D01:= P001 * 2147483647; (* Perform a Multiply operation and store in a 64 bit register *)
P010:= D01 / 12500;     (* Divide the result and store in a 32 bit register *)
```

Note that the above scenario may result in a Divide Overflow, if the result does not fit in a 32-bit register. A Divide Overflow will halt Local Logic execution and servo motion, since the result of the operation is undefined (refer to Chapter 12). An error code will also be reported in the Module Status Code.

Note: *The contents of the 64-bit data registers (D00-D07) and 32-bit writeable data registers (P000-P255) are not automatically initialized by Local Logic when it starts running. The user should initialize any required variables using a separate Local Logic program or the First_Local_Logic_Sweep variable or host controller ladder program.*

13.4 Local Logic User Data Table

Local Logic provides an 8192 Byte Circular Buffer which can be used to store and retrieve data by the Local Logic program. Refer to Table 69 for a listing of the Data_Table variables. The data table is accessed using indirect memory addressing. The Data_Table_Ptr variable (the “Pointer”) is used to point to the correct Byte location in the 8192 Byte buffer. Therefore, the Data_Table_Ptr variable size is 13 bits (0-8191 allowed range). The Pointer is automatically incremented when a value is read from or written to the Circular Buffer. The amount by which the pointer is incremented depends on the size of the variable accessed. The Data_Table_Ptr is automatically initialized to 0 when Local Logic starts, before the first execution sweep. Thus, the Data Table variables can be used to access a large pre-loaded block of data in the Local Logic program. The following variables are used to access the Circular Buffer:

Data_Table_Ptr : Data Table Pointer- valid range 0-8191.

Data_Table_sint : Signed 8 Bits (Pointer auto-incremented by 1 for Read/Write)

Data_Table_usint : Unsigned 8 Bits (Pointer auto-incremented by 1 for Read/Write)

Data_Table_int : Signed 16 Bits (Pointer auto-incremented by 2 for Read/Write)

Data_Table_uint : Unsigned 16 Bits (Pointer auto-incremented by 2 for Read/Write)

Data_Table_dint : 32 Bits (Pointer auto-incremented by 4 for Read/Write)

The sample code below shows how a specific memory location can be accessed in the circular buffer:

```
Data_Table_Ptr := 100; (* Point to Byte offset 100 in the buffer *)
P001 := Data_Table_int; (* Read a signed 16 bit number from the buffer *)
(* The Data_Table_Ptr is auto-incremented to 102 *)
Data_Table_sint := -120; (* Write an 8 bit signed number to Byte 102 *)
Data_Table_Ptr := 0; (* Point to Byte offset 0 in the Buffer *)
```

13.5 Digital Outputs / CTL Variables

The eight Digital Outputs in the module (2 per axis) are individually configurable to be either under host controller control (PLC Control - default) or under Local Logic (DSM) control. If the Local Logic program writes to a particular Digital_Output variable (refer to Table 15H13-1 through Table 20H13-6) it must be configured for DSM control. The DSM module will reject any Local Logic programs that are downloaded with an incorrect Digital Output configuration. Refer to Chapter 14 for a detailed description on configuring the Digital Outputs.

CTL01-CTL24 are also individually configurable to have different input sources. Refer to Chapter 14 for a detailed description of the configuration options. CTL25 through CTL32 are not configurable and are always under Local Logic Control. The DSM module will reject any Local Logic programs that are downloaded with an incorrect CTL configuration. For example, if the Local Logic program has a statement that writes to CTL16 (e.g. CTL16 := 1;), then CTL16 must be configured as “Local Logic Controlled” in Hardware Configuration. CTL01 through CTL32 and the Motion program Block Numbers (variables Block_1, Block_2, Block_3, Block_4) can be used to synchronize the Motion Program and the Local Logic program.

Table 65: Axis 1 Variables

Local Logic Variable Name	Attribute	Size
FacePlate I/O		
Strobe1_Level_1	Read Only	Bit Operand
Strobe2_Level_1	Read Only	Bit Operand
Positive_EOT_1	Read Only	Bit Operand
Negative_EOT_1	Read Only	Bit Operand
Home_Switch_1	Read Only	Bit Operand
Digital_Output1_1 (1)	Write Only	Bit Operand
Digital_Output3_1 (1)	Write Only	Bit Operand
Analog_Input1_1	Read Only	Signed 16 Bits
Analog_Input2_1	Read Only	Signed 16 Bits
Control Variables		
Velocity_Loop_Gain_1	Read/Write	Unsigned 8 Bits
Position_Loop_TC_1	Write Only	Unsigned 16 Bits
Torque_Limit_1	Write Only	Unsigned 16 Bits
Follower_Ratio_A_1	Write Only	Signed 16 Bits
Follower_Ratio_B_1	Write Only	Signed 16 Bits
Position_Increment_Cts_1 (2)	Write Only	Signed 16 Bits
Reset_Strobe1_1	Write Only	Bit Operand
Reset_Strobe2_1	Write Only	Bit Operand
Enable_Follower_1	Write Only	Bit Operand
Jog_Plus_1	Write Only	Bit Operand

Local Logic Variable Name	Attribute	Size
Jog_Minus_1	Write Only	Bit Operand
FeedHold_1	Write Only	Bit Operand
Status Variables		
Error_Code_1	Read Only	Unsigned 16 Bits
Block_1	Read Only	Unsigned 16 Bits
Actual_Position_1	Read Only	32 Bits
Commanded_Position_1	Read Only	32 Bits
Position_Error_1	Read Only	32 Bits
Strobe1_Position_1	Read Only	32 Bits
Strobe2_Position_1	Read Only	32 Bits
Actual_Velocity_1	Read Only	32 Bits
Commanded_Velocity_1	Read Only	32 Bits
Commanded_Torque_1	Read Only	32 Bits
User_Selected_Data1_1	Read Only	32 Bits
User_Selected_Data2_1	Read Only	32 Bits
UnAdjusted_Actual_Position_Cts_1	Read Only	32 Bits
UnAdjusted_Strobe1_Position_Cts_1	Read Only	32 Bits
UnAdjusted_Strobe2_Position_Cts_1	Read Only	32 Bits
Axis_OK_1	Read Only	Bit Operand
Position_Valid_1	Read Only	Bit Operand
Strobe1_Flag_1	Read Only	Bit Operand
Strobe2_Flag_1	Read Only	Bit Operand
Drive_Enabled_1	Read Only	Bit Operand
Program_Active_1	Read Only	Bit Operand
Moving_1	Read Only	Bit Operand
In_Zone_1	Read Only	Bit Operand
Position_Error_Limit_1	Read Only	Bit Operand
Torque_Limited_1	Read Only	Bit Operand
Servo_Ready_1	Read Only	Bit Operand
Follower_Enabled_1	Read Only	Bit Operand
Follower_Velocity_Limit_1	Read Only	Bit Operand
Follower_Ramp_Active_1	Read Only	Bit Operand

Note:

1. These Digital Outputs must be configured for Local Logic control in Hardware Configuration in order to be write-able by Local Logic.
2. The Position_Increment_Cnts_n variable has a maximum range of ± 1023 counts.

Table 66: Axis 2 Variables

Local Logic Variable Name	Attribute	Size
FacePlate I/O		
Strobe1_Level_2	Read Only	Bit Operand
Strobe2_Level_2	Read Only	Bit Operand
Positive_EOT_2	Read Only	Bit Operand
Negative_EOT_2	Read Only	Bit Operand
Home_Switch_2	Read Only	Bit Operand
Digital_Output1_2 ⁽¹⁾	Write Only	Bit Operand
Digital_Output3_2 ⁽¹⁾	Write Only	Bit Operand
Analog_Input1_2	Read Only	Signed 16 Bits
Analog_Input2_2	Read Only	Signed 16 Bits
Control Variables		
Velocity_Loop_Gain_2	Read/Write	Unsigned 8 Bits
Position_Loop_TC_2	Write Only	Unsigned 16 Bits
Torque_Limit_2	Write Only	Unsigned 16 Bits
Follower_Ratio_A_2	Write Only	Signed 16 Bits
Follower_Ratio_B_2	Write Only	Signed 16 Bits
Position_Increment_Cts_2 ⁽²⁾	Write Only	Signed 16 Bits
Reset_Strobe1_2	Write Only	Bit Operand
Reset_Strobe2_2	Write Only	Bit Operand
Enable_Follower_2	Write Only	Bit Operand
Jog_Plus_2	Write Only	Bit Operand
Jog_Minus_2	Write Only	Bit Operand
FeedHold_2	Write Only	Bit Operand
Status Variables		
Error_Code_2	Read Only	Unsigned 16 Bits
Block_2	Read Only	Unsigned 16 Bits
Actual_Position_2	Read Only	32 Bits
Commanded_Position_2	Read Only	32 Bits
Position_Error_2	Read Only	32 Bits
Strobe1_Position_2	Read Only	32 Bits
Strobe2_Position_2	Read Only	32 Bits
Actual_Velocity_2	Read Only	32 Bits
Commanded_Velocity_2	Read Only	32 Bits
Commanded_Torque_2	Read Only	32 Bits
User_Selected_Data1_2	Read Only	32 Bits
User_Selected_Data2_2	Read Only	32 Bits
UnAdjusted_Actual_Position_Cts_2	Read Only	32 Bits

Local Logic Variable Name	Attribute	Size
UnAdjusted_Strobe1_Position_Cts_2	Read Only	32 Bits
UnAdjusted_Strobe2_Position_Cts_2	Read Only	32 Bits
Axis_OK_2	Read Only	Bit Operand
Position_Valid_2	Read Only	Bit Operand
Strobe1_Flag_2	Read Only	Bit Operand
Strobe2_Flag_2	Read Only	Bit Operand
Drive_Enabled_2	Read Only	Bit Operand
Program_Active_2	Read Only	Bit Operand
Moving_2	Read Only	Bit Operand
In_Zone_2	Read Only	Bit Operand
Position_Error_Limit_2	Read Only	Bit Operand
Torque_Limited_2	Read Only	Bit Operand
Servo_Ready_2	Read Only	Bit Operand
Follower_Enabled_2	Read Only	Bit Operand
Follower_Velocity_Limit_2	Read Only	Bit Operand
Follower_Ramp_Active_2	Read Only	Bit Operand

Note:

1. These Digital Outputs must be configured for Local Logic control in Hardware Configuration in order to be write-able by Local Logic.
2. The Position_Increment_Cnts_n variable has a maximum range of ± 1023 counts.

Table 67: Axis 3 Variables

Local Logic Variable Name	Attribute	Size
FacePlate I/O		
Strobe1_Level_3	Read Only	Bit Operand
Strobe2_Level_3	Read Only	Bit Operand
Positive_EOT_3	Read Only	Bit Operand
Negative_EOT_3	Read Only	Bit Operand
Home_Switch_3	Read Only	Bit Operand
Digital_Output1_3 *	Write Only	Bit Operand
Digital_Output3_3 *	Write Only	Bit Operand
Analog_Input1_3	Read Only	Signed 16 Bits
Analog_Input2_3	Read Only	Signed 16 Bits
Control Variables		
Reset_Strobe1_3	Write Only	Bit Operand
Reset_Strobe2_3	Write Only	Bit Operand
Status Variables		
Error_Code_3	Read Only	Unsigned 16 Bits
Actual_Position_3	Read Only	32 Bits
Strobe1_Position_3	Read Only	32 Bits
Strobe2_Position_3	Read Only	32 Bits
Actual_Velocity_3	Read Only	32 Bits
Axis_OK_3	Read Only	Bit Operand
Position_Valid_3	Read Only	Bit Operand
Strobe1_Flag_3	Read Only	Bit Operand
Strobe2_Flag_3	Read Only	Bit Operand

* These Digital Outputs must be configured for Local Logic control in Hardware Configuration in order to be write-able by Local Logic.

Note: For Axis 3, the DSM314 Version 2.0 only supports the variables in Table 67.

Table 68: Axis 4 Variables

Local Logic Variable Name	Attribute	Size
FacePlate I/O		
Strobe1_Level_4	Read Only	Bit Operand
Strobe2_Level_4	Read Only	Bit Operand
Positive_EOT_4	Read Only	Bit Operand
Negative_EOT_4	Read Only	Bit Operand
Home_Switch_4	Read Only	Bit Operand
Digital_Output1_4 *	Write Only	Bit Operand
Digital_Output3_4 *	Write Only	Bit Operand
Analog_Input1_4	Read Only	Signed 16 Bits
Analog_Input2_4	Read Only	Signed 16 Bits

* These Digital Outputs must be configured for Local Logic control in Hardware Configuration, in order to be writeable by Local Logic.

Note: For Axis 4, the DSM314 Version 2.0 only supports the variables in Table 68.

Table 69: Global Variables

Local Logic Variable Name	Attribute	Size
Overflow ⁽¹⁾	Read / Write	Bit Operand
System_Halt ⁽¹⁾	Write Only	Bit Operand
Data_Table_Ptr ⁽²⁾	Read / Write	13 Bits
Data_Table_sint ⁽²⁾	Read / Write	Signed 8 Bits
Data_Table_usint ⁽²⁾	Read / Write	Unsigned 8 Bits
Data_Table_int ⁽²⁾	Read / Write	Signed 16 Bits
Data_Table_uint ⁽²⁾	Read/ Write	Unsigned 16 Bits
Data_Table_dint ⁽²⁾	Read / Write	32 Bits
Module_Error_Present	Read Only	Bit Operand
New_Configuration_Received	Read Only	Bit Operand
First_Local_Logic_Sweep ⁽¹⁾	Read Only	Bit Operand
Module_Status_Code	Read Only	Unsigned 16 Bits
CTL_1_to_32 ⁽³⁾	Read Only	32 Bits
P000-P255	Read / Write	32 Bits
D00-D07 ⁽⁴⁾	Read / Write	64 Bits

Note:

1. Refer to the Section on “Local Logic System Variables”.
2. Refer to the Section on “Local Logic User Data Table”.
3. The CTL_1_to_32 variable can be used to read all 32 CTL bits into a register.
4. Refer to the Section on “Double Precision 64 Bit Registers”.

Table 70: CTL Bits

Local Logic Variable Name	Attribute	Size
CTL01 **	Read / Write	Bit Operand
CTL02 **	Read / Write	Bit Operand
CTL03 **	Read / Write	Bit Operand
CTL04 **	Read / Write	Bit Operand
CTL05 **	Read / Write	Bit Operand
CTL06 **	Read / Write	Bit Operand
CTL07 **	Read / Write	Bit Operand
CTL08 **	Read / Write	Bit Operand
CTL09 **	Read / Write	Bit Operand
CTL10 **	Read / Write	Bit Operand
CTL11 **	Read / Write	Bit Operand
CTL12 **	Read / Write	Bit Operand
CTL13 **	Read / Write	Bit Operand
CTL14 **	Read / Write	Bit Operand
CTL15 **	Read / Write	Bit Operand
CTL16 **	Read / Write	Bit Operand
CTL17 **	Read / Write	Bit Operand
CTL18 **	Read / Write	Bit Operand
CTL19 **	Read / Write	Bit Operand
CTL20 **	Read / Write	Bit Operand
CTL21 **	Read / Write	Bit Operand
CTL22 **	Read / Write	Bit Operand
CTL23 **	Read / Write	Bit Operand
CTL24 **	Read / Write	Bit Operand
CTL25	Read / Write	Bit Operand
CTL26	Read / Write	Bit Operand
CTL27	Read / Write	Bit Operand
CTL28	Read / Write	Bit Operand
CTL29	Read / Write	Bit Operand
CTL30	Read / Write	Bit Operand
CTL31	Read / Write	Bit Operand
CTL32	Read / Write	Bit Operand

** CTL bits 1 through 24 are individually configurable in Hardware Configuration (refer to the Chapter 14 “Local Logic Configuration”). CTL01-24 can be written by Local Logic only if configured as “Local Logic Controlled” in Hardware Configuration.

Chapter 14: Local Logic Configuration

14.1 CTL Bit Configuration

The programming software environment allows you to configure the input source for CTL bits (CTL01-CTL24) using the Hardware Configuration screen. From the Hardware Configuration screen, select the DSM314 module you wish to configure. Refer to chapter 4 for information on using Hardware configuration. The DSM314 configuration screens contain a tab called CTL Bits. Selecting this tab results in a display similar to the one shown in Figure 142.

Figure 142: CTL Bits Configuration

Parameters	Values
CTL01 Config:	Local Logic Controlled
CTL02 Config:	IN10_A (Axis 1 - OT)
CTL03 Config:	IN11_A (Axis 1 Home SW)
CTL04 Config:	Strobe 1 Level (Axis 1)
CTL05 Config:	IN9_B (Axis 2 + OT)
CTL06 Config:	IN10_B (Axis 2 - OT)
CTL07 Config:	IN11_B (Axis 2 Home SW)
CTL08 Config:	Strobe 1 Level (Axis 2)
CTL09 Config:	%Q Bit Offset 12
CTL10 Config:	%Q Bit Offset 13
CTL11 Config:	%Q Bit Offset 14
CTL12 Config:	%Q Bit Offset 15
CTL13 Config:	IN9_C (Axis 3 + OT)
CTL14 Config:	IN10_C (Axis 3 - OT)
CTL15 Config:	IN11_C (Axis 3 Home SW)
CTL16 Config:	Strobe 1 Level (Axis 3)
CTL17 Config:	%Q Bit Offset 24
CTL18 Config:	%Q Bit Offset 25
CTL19 Config:	%Q Bit Offset 40
CTL20 Config:	%Q Bit Offset 41
CTL21 Config:	%Q Bit Offset 56
CTL22 Config:	%Q Bit Offset 57
CTL23 Config:	%Q Bit Offset 72

The configuration screen allows the user to select the CTL bit configuration that corresponds with the Motion Program and Local Logic program. The sections that follow provide additional information concerning the CTL bit configuration process.

14.2 CTL bits CTL01-CTL32

- CTL01 - CTL24 are configurable CTL bits.
- CTL25 - CTL32 are non-configurable CTL bits providing Local Logic read and Local Logic write.

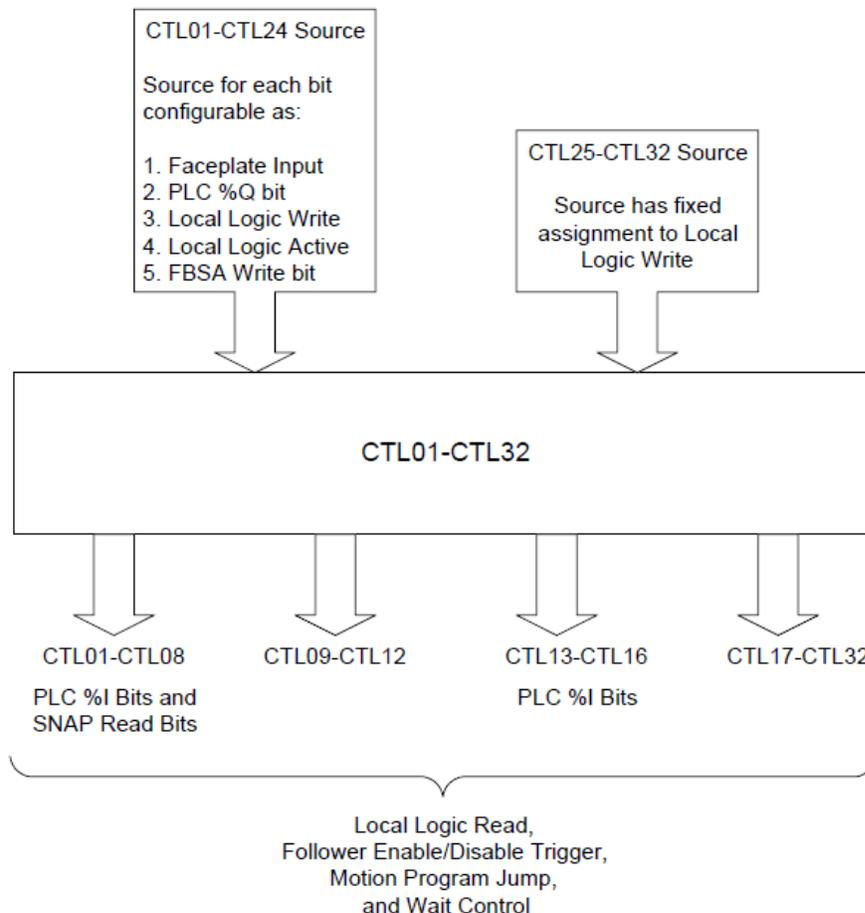
Table 71: CTL Bit Summary for DSM314

Identifier	%I Bit	Faceplate Inputs	%Q bit	Local Logic Read	Local Logic Write	SNAP ¹ Write	SNAP ¹ Read
CTL01-CTL08	X	Config	Config	X	Config	Config	X
CTL09-CTL12		Config	Config	X	Config	Config	
CTL13-CTL16	X	Config	Config	X	Config	Config	
CTL17-CTL24		Config	Config	X	Config	Config	
CTL25-CTL32				X	X		

¹ Series 90-30 only feature. SNAP is equivalent to Fast Backplane Status Access (FSBA). See GFK-0467L or later for details.

The figure below illustrates the sources that write to CTL bits and the destinations that read CTL bits:

Figure 143: CTL Bit Source/Destinations



14.3 CTL01-CTL24 Bit Configuration Selections

Each of the bits CTL01-CTL24 are individually configurable. CTL17-CTL24 default to the %Q digital output control bits for axis 1 - axis 4. The configuration choices are shown in the following table.

Table 72: CTL Bit Configuration Selections

CTL Bits	Allowed Configuration Values for Bit Source	Description
CTL01-CTL24	IN9_A	Overtravel (+) Axis 1
	IN10_A	Overtravel (-) Axis 1
	IN11_A	Home Switch Axis 1
	IN9_B	Overtravel (+) Axis 2
	IN10_B	Overtravel (-) Axis 2
	IN11_B	Home Switch Axis 2
	IN9_C	Faceplate 24v Input Axis 3
	IN10_C	Faceplate 24v Input Axis 3
	IN11_C	Home Switch Axis 3
	IN9_D	Faceplate 24v Input Axis 4
	IN10_D	Faceplate 24 v Input Axis 4
	IN11_D	Faceplate 24 v Input Axis 4
	Strobe1 Level Axis1	Input Strobe1 Level Axis 1
	Strobe2 Level Axis1	Input Strobe 2 Level Axis 1
	Strobe1 Level Axis2	Input Strobe 1 Level Axis 2
	Strobe2 Level Axis2	Input Strobe 2 Level Axis2
	Strobe1 Level Axis3	Input Strobe 1 Level Axis 3
	Strobe2 Level Axis3	Input Strobe 2 Level Axis 3
	IN5_D	Faceplate 5v Input Axis 4
	IN6_D	Faceplate 5v Input Axis 4
	Local Logic Write	CTL bit under Local Logic control
	Local Logic Active Flag	Local Logic Program Active
	SNAP Write Bit 1	Serial Non-Acknowledge Protocol (FBSA) Bit 1
	SNAP Write Bit 2	Serial Non-Acknowledge Protocol (FBSA) Bit 2
	SNAP Write Bit 3	Serial Non-Acknowledge Protocol (FBSA) Bit 3
	SNAP Write Bit 4	Serial Non-Acknowledge Protocol (FBSA) Bit 4
	%Q bit Offset 12	CTL09 Program Control
	%Q bit Offset 13	CTL10 Program Control
	%Q bit Offset 14	CTL11 Program Control
	%Q bit Offset 15	CTL12 Program Control
	%Q bit Offset 24	Faceplate 24v Output Control Axis 1 (OUT1_A)
	%Q bit Offset 25	Faceplate 5v Output Control Axis 1 (OUT3_A)

CTL Bits	Allowed Configuration Values for Bit Source	Description
	%Q bit Offset 40	Faceplate 24v Output Control Axis 2 (OUT1_B)
	%Q bit Offset 41	Faceplate 5v Output Control Axis 2 (OUT3_B)
	%Q bit Offset 56	Faceplate 24v Output Control Axis 3 (OUT1_C)
	%Q bit Offset 57	Faceplate 5v Output Control Axis 3 (OUT3_C)

14.4 FBSA Function and CTL Bit Assignments

The backplane Fast Backplane Status Access (FBSA) function will write 4 bits to the DSM and read 8 bits. The FBSA function is mapped as shown in the following table.

FBSA is a Series 90-30 only feature. For information on the FBSA service request, refer to the Series 90-30/20/Micro PLC CPU Instruction Set Reference Manual, GFK-0467L (or later).

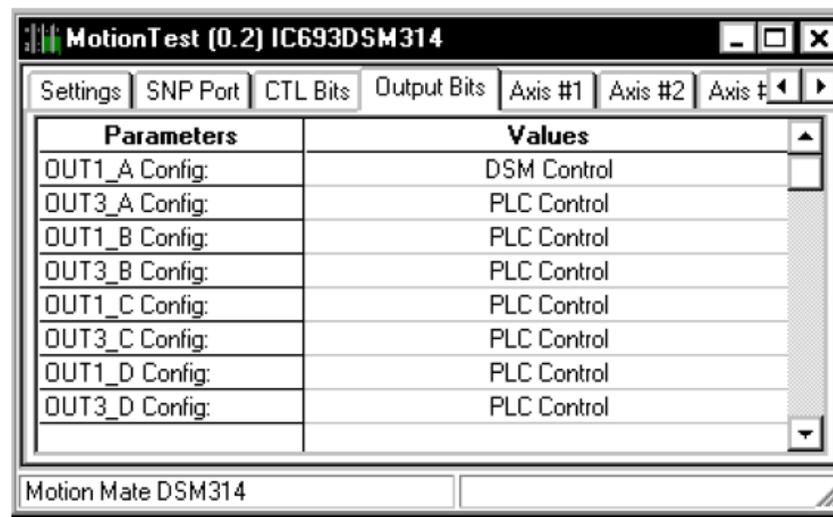
Table 73: FBSA Bit CTL Bit Assignments

FBSA Read	CTL01-CTL08	CTL01-CTL08 each have an individually configurable source that includes Local Logic or any DSM faceplate input. The bits are always readable as PLC %I bits and FBSA inputs.
FBSA Write	CTL01-CTL24 (Configurable)	FBSA Write Bits 1-4 can be configured as the source for any of the bits CTL01-CTL24. FBSA Write Bits 1-2 are the default source for CTL23-24.

14.5 Faceplate Output Bit Configuration

The programming environment, through Hardware configuration, allows you to configure the DSM314 faceplate digital outputs for either Local Logic program control or host controller program control. The DSM314 configuration screens contain a tab (Output Bits). Selecting this tab results in a display similar to the one shown in Figure 144.

Figure 144: Output Bit Configuration



The following table describes the faceplate outputs that can be controlled from Local Logic or the host controller.

Table 74: Faceplate Output Bit Description

Signal Name	Description
OUT1_A	Faceplate 24v (SSR) Output Axis 1
OUT3_A	Faceplate 5v Output Axis 1
OUT1_B	Faceplate 24v (SSR) Output Axis 2
OUT3_B	Faceplate 5v Output Axis 2
OUT1_C	Faceplate 24v (SSR) Output Axis 3
OUT3_C	Faceplate 5v Output Axis 3
OUT1_D	Faceplate 24v (SSR) Output Axis 4
OUT3_D	Faceplate 5v Output Axis 4

Chapter 15: Using the Electronic CAM Feature

This chapter describes the electronic CAM function, which was introduced in DSM314 release 2.0. An electronic CAM is analogous to a mechanical CAM. In most cases, an electronic CAM not only can replace the traditional mechanical CAM but also performs many functions not achievable with its mechanical counterpart. For example, an electronic CAM never mechanically wears out.

15.1 Electronic CAM Overview

Electronic CAMs are used in the machine industry to perform complex motions that require tight coordination between axes. There are many examples of applications that fit these requirements. Some examples are a simple rotary knife application shown in Figure 145 and Figure 146. In this application, the conveyor belt position serves as the master position, while the cutting knife is the slave. Since the knife position is linked to the master position, the knife always tracks the master position even when the line is accelerating or decelerating.

Figure 145: Rotary Knife Position to Position table

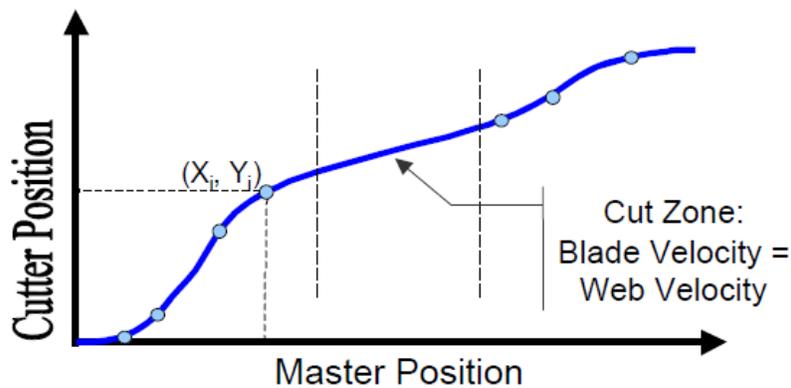
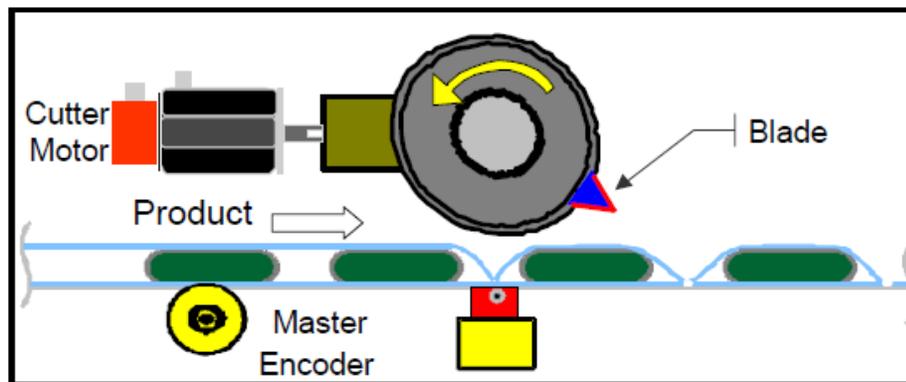


Figure 146: Rotary Knife Application



Another example application is a bottle filling line (reference Figure 147 and Figure 148). In this case, the lift that raises and lowers the bottles serves as the CAM master. The slave is the plunger that pushes the fluid into the bottle. In this example, the bottles have a curved shape. Thus the fill rate must be varied to account for this shape.

Figure 147: Filling Application Position to Position Table

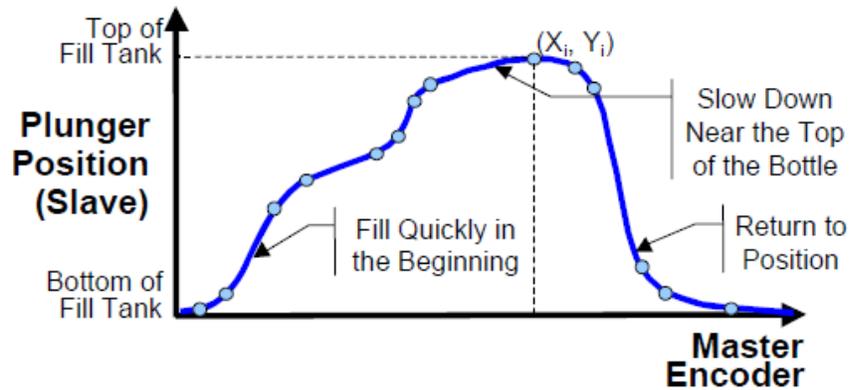
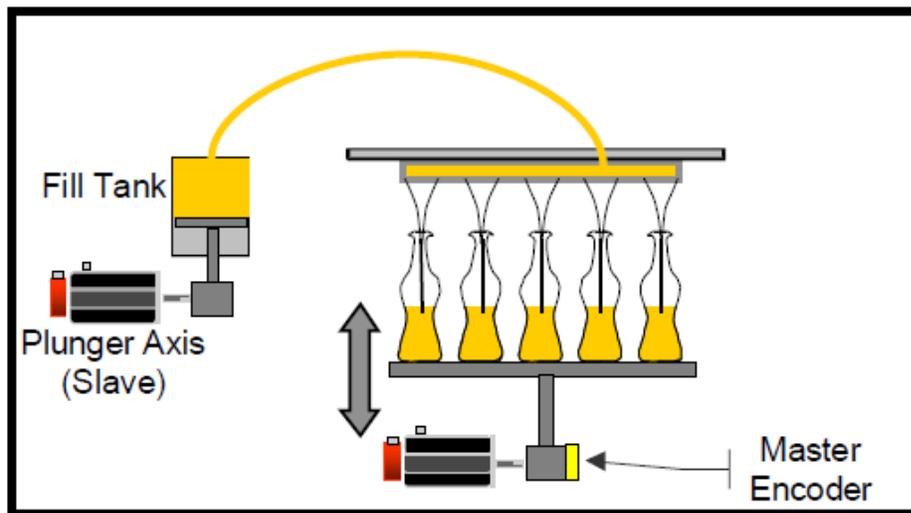


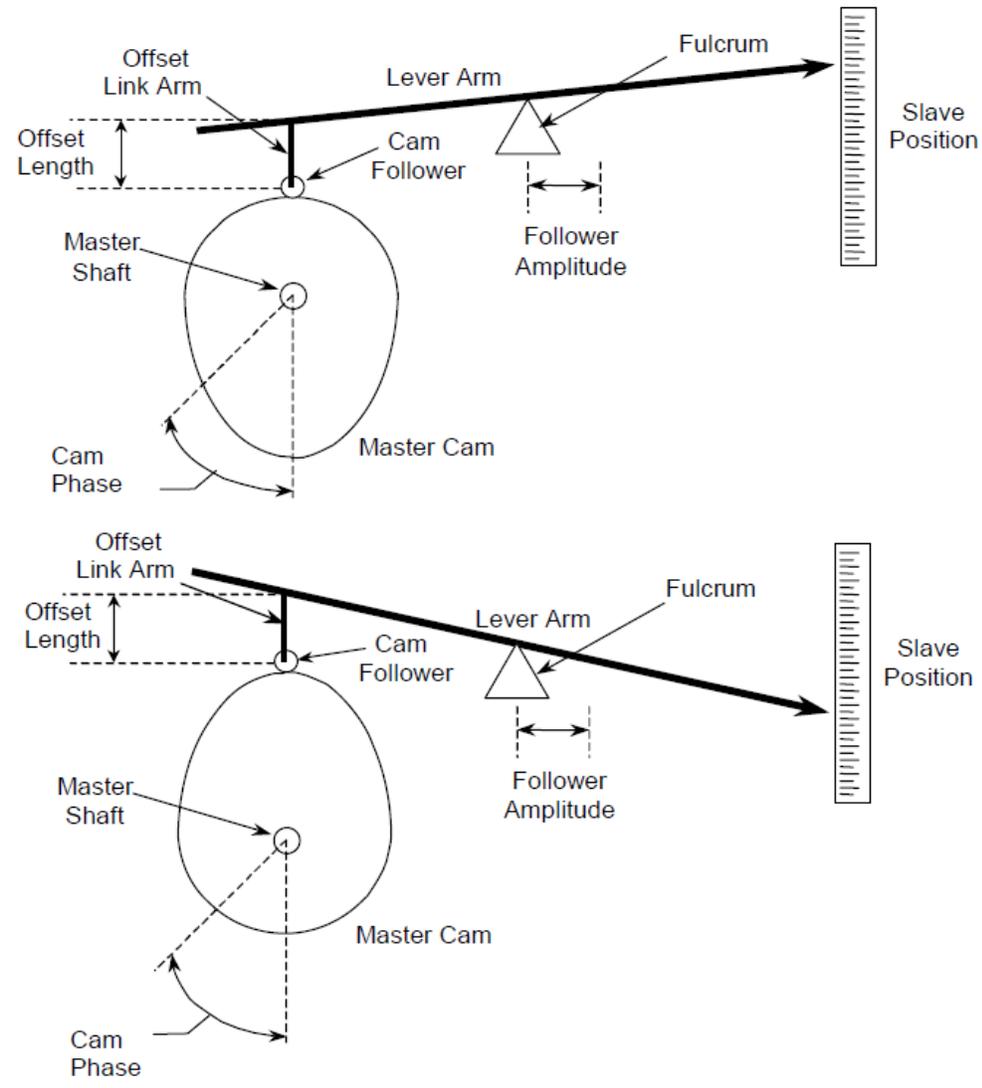
Figure 148: Filling Application



15.2 Basic Cam Shapes/Definition

Electronic Cams duplicate the behavior of their mechanical counterparts. The following figure illustrates the elements of a basic mechanical cam system and shows the Slave Position for two positions of the Master Cam. As the Master Shaft rotates, the Master Cam, which is fastened to the Master Shaft, rotates as well. The Cam Follower (which is a ball bearing mounted to the Offset Link Arm) rolls on the Master Cam as the Master Cam rotates. The Cam Follower either pushes up or pulls down on the Offset Link Arm, depending on the position of the Master Cam. The Lever Arm, which is coupled to the Offset Link Arm, moves up and down in turn, pivoting on the Fulcrum as the Offset Link Arm moves. Besides the Master Cam shape, additional parameters that affect slave motion are the Cam Phase value (the amount that the Master Cam position is offset from the Master Shaft position), the Offset Length of the Offset Link Arm, and the Follower Amplitude (based on the Fulcrum position). These mechanical parameters all have Electronic Cam counterparts.

Figure 149: CAM Model



15.3 CAM Syntax

This section covers some critical features of the CAM feature and introduces the CAM Motion Program statements and error codes.

15.3.1 CAM Types

An important concept concerning the CAM function is the different CAM types available. The CAM profiles can be one of the following types:

1. Non-Cyclic CAM
2. Linear Cyclic CAM
3. Circular Cyclic CAM

The following sections describe each of these CAM types.

Non-Cyclic CAM

A Non-Cyclic CAM has a unique non-repeating profile for the whole range of Master position values. The CAM exits when either boundary of the CAM profile is reached. The CAM can also exit if an external event is configured to trigger a conditional Jump. The User Units to Counts ratio specified for the Master and Slave axes when configuring a Non-Cyclical CAM must match the User Units:Counts ratio specified for the corresponding axes in Hardware Configuration. Also, the maximum and minimum position values for the slave and master axes must lie within the High/Low position limits specified for the corresponding axes in Hardware Configuration.

Linear Cyclic CAM

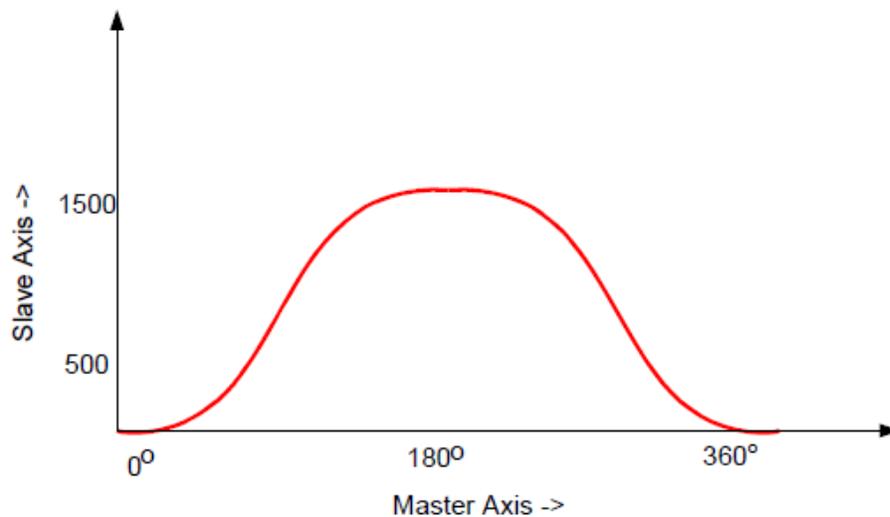
A Linear Cyclic CAM has a profile that keeps repeating until an event causes it to exit. Furthermore, the numerical and physical end points of the CAM slave axis are the same as the starting point of the cycle. A reciprocating crankshaft is an example of a Linear Cyclic CAM. The User Units to Counts ratio specified for the master and slave axes when configuring a CAM profile must match the User Units per Counts value for the corresponding axes in Hardware Configuration. Figures 145 and 146 show an example of a Linear Cyclic CAM application.

Constraint: The first and last slave point must be the same for a Linear Cyclic CAM. The CAM Editor will not display the option for “Linear Cyclic” in the “Cam Type” field unless this constraint is satisfied by the data in the CAM table.

Note:

1. For any Cyclic CAM, the master High/Low Position limits in Hardware Configuration must be set up according to the master rollover points in the CAM profile. The master axis Low Limit must equal the first master position in the profile. The master High Position Limit must be equal to the (Last Master Position – 1) in user units. This is because a cyclic profile's first and last point are the same on the physical device.
2. For any Cyclic CAM, the master High/Low Position limits in Hardware Configuration must be set up according to the master rollover points in the CAM profile. The master axis Low Limit must equal the first master position in the profile. The master High Position Limit must be equal to the (Last Master Position – 1) in user units. This is because a cyclic profile's first and last point are the same on the physical device.

Figure 150: Linear Cyclic CAM



Circular Cyclic CAM

A Circular Cyclic CAM has a profile that keeps repeating until an event causes it to exit. Furthermore, a Circular Cyclic CAM has different numerical start and end slave axis positions (see Figure 151). Both the master axis and the slave axis roll over at the profile end points. A rotary knife is an example of a Circular Cyclic CAM.

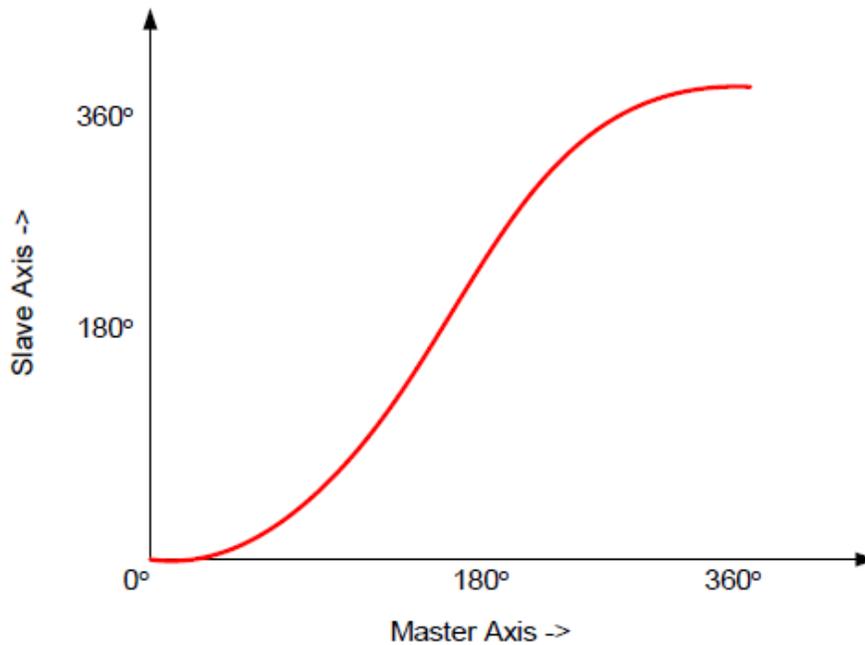
Constraint: The **entire** slave profile (including interpolated values) must lie between the minimum and maximum slave position limits, where the minimum and maximum slave limits are defined as follows:

Minimum Slave Value	Maximum Slave Value	Condition
First Slave Point	Last Slave Point	Last Point > First Point
Last Slave Point	First Slave Point	Last Point < First Point

Note:

1. The Editor will not display “Circular Cyclic” as an option in the “CAM Type” field unless the constraint described above is satisfied.
 2. For Cyclic CAMs, the master High/Low Position limits in Hardware Configuration must be set up according to the master rollover points in the CAM profile. The master axis Low Limit must equal the first master position in the profile. The master High Position Limit must be equal to the (Last Master Position – 1) in user units. This is because a cyclic profile's first and last point are the same on the physical device. (for example, 0° and 360° on a circular knife).
 3. For a Circular Cyclic CAM, both the master axis and the slave axis rolls over at the profile's end points. The High and Low position limits for the slave axis are set (in Hardware Config.) as follows:
 - If the minimum slave position is the profile's first point and the maximum slave position is the last point, set the Low Position Limit to the first point's slave value and the High Position limit to the (last point's slave value – 1).
 - If the minimum slave position is the profile's last point and the maximum slave position is the first point, set the Low Position Limit to the (last point's slave value + 1) and the High Position Limit to the first point's slave value.
-

Figure 151: Circular Cyclic CAM



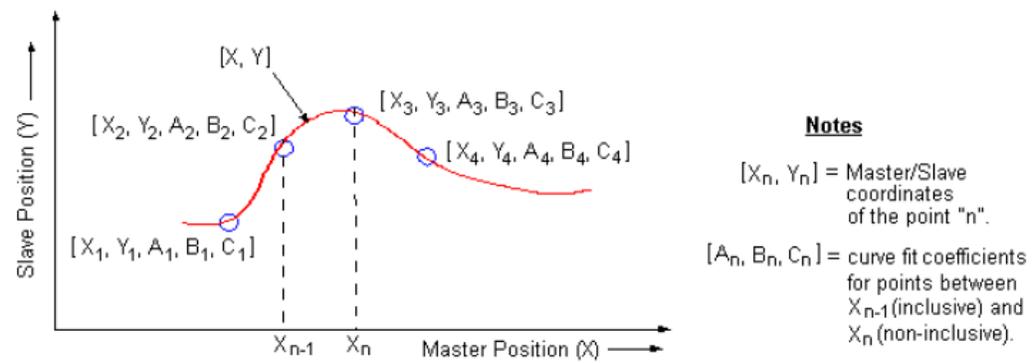
15.3.2 Interpolation and Smoothing

One key CAM feature is the interpolation scheme used to define the CAM profiles. The following is a reprint of a section from the CAM Editor help system. It is included in this section to not only introduce these important concepts, but also to encourage you to explore the CAM Editor on-line help for additional information.

The CAM editor employs spline polynomial interpolation to define regions of a profile that fall between user-defined points. This approach reduces the memory required for profile storage on the target motion module while providing accurate and smooth motion trajectories. Without this interpolation scheme, a large number of data points, thus a large amount of memory, would be needed to define each profile.

A CAM profile is defined with a minimum number of actual data points. After these points are defined they are grouped into sectors; a profile is composed of one or more sectors. For each sector, you specify the curve-fit order (1, 2 or 3). The higher the order, the smoother the curve-fit. The curve-fit order is the order of the polynomial curves used to define the regions of the sector not specified by user-defined points. Unique curve-fit polynomial coefficients are generated for each segment of a sector (that is, between each pair of user-defined points). The coefficients of the polynomials are calculated to include the user-defined points and to match the slope of the profile on either side of a user-defined point (except for 1st order sectors).

Figure 152: Windowing Strobes Example



The polynomial curves for a position profile are described by the following function:

$$Y(X) = A_{n-1}(X_n - X_{n-1})^3 + B_{n-1}(X_n - X_{n-1})^2 + C_{n-1}(X_n - X_{n-1}) + Y_{n-1}$$

Where:

Y = slave position value for a master position X .

X_{n-1} = master position value at point $n-1$.

$A_{n-1}, B_{n-1}, C_{n-1}$ = curve-fit coefficients at point $n-1$.

Note:

- For a given master position X , that lies between X_{n-1} and X_n , the coefficients A , B and C are selected for the point corresponding to X_{n-1} .
 - For a second order curve-fit, the A coefficient is always zero, and for a first order curve-fit, both the A and B coefficients are always zero.
-

Blending Sectors

The process applied to blend adjacent sectors depends on their curve-fit order. The following descriptions cover the possible scenarios.

1st order to 1st order

No action is taken to smooth the transition between successive linear sectors (that is, with curve-fit order of 1). The profile simply connects the end point of one sector to the start point of the next with a straight line.

1st order to 2nd order

When a quadratic (2nd order) sector follows a linear (1st order) sector, the polynomial coefficients for the first segment of the quadratic sector are calculated so that the slope of the profile is equal on either side of the starting point of that sector. That is, the initial slope of the quadratic sector is made equal to the final slope of the linear sector.

2nd order to 1st order

When a linear (1st order) sector follows a quadratic (2nd order) sector no action is taken to smooth the transition. This type of transition is not recommended (if it is avoidable) as it may result in drastic velocity or acceleration changes on the controlled servo.

2nd order to 2nd order

When a quadratic (2nd order) sector follows another quadratic (2nd order) sector the polynomial coefficients for the first segment of the second quadratic sector are calculated so that the slope of the profile is equal on either side of the starting point of that sector. That is, the initial slope of the second quadratic sector is made equal to the final slope of the first quadratic sector.

2nd order to 3rd order

When a cubic (3rd order) sector follows a quadratic (2nd order) sector the polynomial coefficients for the first segment of the cubic sector are calculated so that the slope of the profile is equal on either side of the starting point of that sector. That is, the initial slope of the cubic sector is made equal to the final slope of the quadratic sector.

3rd order to 2nd order

When a quadratic (2nd order) sector follows a cubic (3rd order) sector the polynomial coefficients for the first segment of the quadratic sector are calculated so that the slope of the profile is equal on either side of the starting point of that sector. That is, the initial slope of the quadratic sector is made equal to the final slope of the cubic sector.

3rd order to 3rd order

When two cubic (3rd order) sectors are adjacent, the slopes of the profile before and after the point they meet are made equal. Also, the 2nd derivatives of the profile before and after the point the sectors meet are made equal. The curve-fit polynomial coefficients for the two adjacent segments are calculated simultaneously.

Boundary Conditions

For non-cyclic profiles it is necessary to define some condition at the start and end of a profile for the purpose of calculating curve-fit polynomial coefficients. The start or end boundary condition can be:

- The numerical value of the profile's 1st derivative (slope).
- The numerical value of the profile's 2nd derivative.
- Based on a default calculation.

The default calculations are as follows:

- **Start Boundary.** The slope at the start point of the profile is calculated by temporarily fitting a polynomial curve to the first three (2nd order sector) or four points (3rd order sector) and calculating the slope of the temporary polynomial at the first point.
- **End Boundary.** The slope at the end point of the profile is calculated by temporarily fitting a polynomial curve to the last four points (3rd order sector) and calculating the slope of the temporary polynomial at the end point.

15.3.3 Interaction of Motion Programs with CAM

CAM motion shall be initiated in the DSM314 using instructions in the motion program. The following new motion instructions are required to support CAM motion programming:

1. **CAM:** Used in the motion program to start CAM motion and specify exit conditions.
2. **CAM-LOAD:** Used to load a parameter register with the starting location for a CAM slave axis. The PMOVE command can be used in conjunction with the CAM-LOAD command to move a slave axis to the starting point.
3. **CAM-PHASE:** Used to specify a Phase for CAM commands. The phase value may be specified either through a parameter register or as a constant.

The following sections describe the syntax and functionality of each of the above instructions in more detail. The convention used to specify the command syntax is as follows:

'<>' brackets- indicates a required field.

'[]' brackets- indicates an optional field.

'{}' brackets- indicates a field that is required for multi-axis programs and subroutines but is illegal for single axis programs and subroutines.

15.3.4 CAM Command

The CAM command is used to program a CAM move using the specified CAM profile.

Syntax:

CAM <"CAM Profile Name">, <distance>, <master mode>, [Cyclic Exit Condition]

Parameter	Description
<"CAM Profile Name">	Name of the CAM Profile from the CAM Library (the profile must be linked to the CAM Download block). This name is limited to 20 characters maximum (also, see Note 3 below). Note that the quotes around the name are required.
<distance>	Maximum distance the master axis can travel once the CAM is active (in user units) Distance can be a constant , a parameter or the keyword NONE. Allowed Range: -MaxPosn (MaxPosn-1) uu/cts
<master mode>	Master mode can be declared as ABS (absolute) or INCR (incremental); this indicates how the master position data is interpreted. In ABS mode, an absolute master axis position is used to determine a slave value from the CAM table. In INCR mode, the master value at the starting point of a CAM command is assumed to be equal to the "CAM-phase" value, and the slave values calculated during CAM motion are relative to this start master value.
[Cyclic Exit Condition]	Specifies an exit condition for Cyclical CAMs. The allowed range is CTL01-CTL32. If the CTL bit evaluates to True, the Cam exits at the end of the current cycle. Note that this parameter must not be used for a Non-Cyclic CAM profile.

Note:

1. The CAM instruction is not permitted in a Multi-Axis motion program.
2. A CAM command counts as two instructions towards the 1000 instruction limit in a motion program.
3. The Profile Names UDT_CAM_1, UDT_CAM_2, UDT_CAM_3 and UDT_CAM_4 are reserved for future usage.

The <distance> argument in the CAM command is used to define the maximum distance the master can travel through the profile before exiting. For Cyclic CAMs, the distance can be either greater than or less than the length of the CAM table (defined as the absolute difference between the first and last master positions in the CAM table). If the distance is less than the length of the table, the CAM command exits once the distance has been traversed. If the distance is greater than the length of the table, the CAM will cycle through the CAM table until the distance is reached. Thus, the user may set up the number of times a Cyclic CAM should be executed. For example, a distance of 2.5 times the length of the CAM Table Master Position will cause the CAM profile to execute two and one-half times and then exit. For non-Cyclic CAMs, the specified distance cannot exceed the length of the table.

The distance can also be specified as "NONE". For a Cyclical CAM, this will result in continuous CAM motion until a CTL bit triggers an exit or motion is aborted. For a non-

cyclical CAM, specifying “NONE” means the CAM will exit when it reaches either the minimum or maximum master position of the profile.

The [master mode] is used to specify whether the master axis operates in Absolute or Incremental mode. The master axis may be operated in absolute or incremental mode for both Cyclic and Non-Cyclic CAMs. In Absolute mode, the master positions in the table represent the absolute positions of the master axis. In Incremental mode, the slave axis positions in the table are relative to the master axis position when the CAM instruction is initiated.

The [Cyclic Exit Condition] is used to specify an exit condition for a Cyclic CAM profile. If the CTL condition evaluates to TRUE, the CAM will exit at the end of the current cycle.

Bi-directional and **Unidirectional** CAMs can be defined by using the +Vlim and -Vlim master axis velocity limit parameters. For Unidirectional operation, the appropriate velocity limit must be set to zero for the direction in which motion is prohibited. For example, if motion in the negative direction is prohibited, then -Vlim must be set to zero.

15.3.5 CAM-LOAD Command

The CAM-LOAD command is used to load the slave axis position into a parameter register. A regular PMOVE command can then be used to move the slave axis to the loaded position. The CAM-LOAD command uses the CAM profile name, actual master position and phase (specified using the CAM-PHASE command) to determine the starting point for the slave axis.

Syntax:

CAM-LOAD <"CAM profile name">, <Parameter Number>, <master mode>

Parameter	Description
<"CAM Profile Name">	Name of the CAM Profile from the CAM Library (the profile must be linked to the CAM Download block). This name is limited to a maximum length of 20 characters (also, see Note 3 below). Note that the quotes around the name are required.
<Parameter Number>	Specifies the Parameter Number to load.
<master mode>	Master mode can be declared as ABS (absolute) or INCR (incremental); this indicates how the master position data is interpreted in the slave start position calculation. In ABS mode the absolute master axis position is used to determine the corresponding slave starting position value from the CAM table. In INCR mode, the master value is assumed to be equal to the CAM-Phase in the calculations.

Note:

1. A CAM-LOAD command counts as two instructions towards the 1000 instruction limit in a motion program.
2. When a CAM-LOAD command is executed, the following sequence of actions is performed:
 - A. The current master position is read.
 - B. Using the master position, CAM-Phase value, the CAM profile table, and CAM configuration table, the appropriate position of the slave axis is calculated and loaded into the designated parameter register.
 - C. The motion program can use a PMOVE instruction to move the slave axis to the position calculated in step B.
3. The names UDT_CAM_1, UDT_CAM_2, UDT_CAM_3 and UDT_CAM_4 are reserved for future use and cannot be used for CAM Profile Names.

15.3.6 CAM-PHASE Command

The CAM-PHASE command is used to specify a phase for CAM commands. This command lets you offset or shift the phase relationship between the master position and follower position. The phase value may be specified either through a parameter register or as a constant. Note that a phase value is active for all CAM instructions that follow it, until modified by another CAM-PHASE command. The default Cam Phase value for a motion program is 0.

Syntax:

CAM-PHASE <Phase>

Parameter	Description
<Phase>	The CAM phase value specified as a constant or a Parameter Register. Allowed Range for constant: -MaxPosn (MaxPosn-1)

15.3.7 CAM and MOVE Instructions

A series of CAM commands may execute without any dwells or interruptions. To obtain smooth motion you must ensure that the starting point on each subsequent CAM profile is the same as the ending point of the preceding CAM profile. This ensures a continuous position and velocity trajectory. For a sequence of Non-Cyclic CAMs, the starting and ending points may be adjusted in the CAM Editor to obtain smooth transitions. Transitions between CAM and MOVE commands while the slave axis is moving are not permitted at this time. Consequently, the slave axis must have a start velocity equal to 0 at the transition point between a CAM and MOVE command. When a CAM command exits, if it is not immediately followed by another CAM command, the axis will use the programmed acceleration rate to decelerate to a stop.

15.3.8 Time-Based CAM Motion

The implementation of a time-based CAM profile employs the same mechanism as a regular CAM (position-based master). In order to program a time-based CAM profile, the CAM master source should be configured as “Commanded Position” of Axis 3 in the DSM314 module hardware configuration, with the Axis 3 mode set to “Auxiliary Axis.” A constant velocity command is then initiated on Axis 3. The effective time scale of the CAM motion is determined by the scaling of the master in the profile source file and the User Units-to-Counts conversion factor defined in Hardware Configuration. A time-based CAM motion command can be executed simultaneously on multiple axes.

15.3.9 CAM Scaling Editor and Hardware Configuration

The DSM module allows the user to scale the position feedback device resolution versus the module programming units. For example suppose 1 motor revolution corresponded to 1 inch of travel for the driven load. In this example, the motor connected to the driven load has an encoder that produces 8,192 counts per motor revolution. Thus, 8,192 feedback counts equals 1 inch of load travel. Some users would find it easier to program motion in inches rather than in feedback counts. In this case, you could set up the scaling to program motion in thousandths of an inch. To obtain this result, set the User Units to

Counts ratio to be 1000 to 8192 in the DSM hardware configuration. Additional information on specifying these values is located in Chapter 5.

The CAM feature also supports application-specific units. However, you are required to manually transfer the values entered in hardware configuration to the appropriate area within the CAM editor.

Note: *You must transfer these values for both Master and Slave axes. Building on the prior example, suppose both the master and the slave axes had equivalent motors. Therefore, each feedback device has the same 8,192 counts per revolution. However, for the master, one motor revolution equals 1 inch of load travel, while for the slave, one motor revolution equals .5 inches of load travel. To make the programs easier to understand, you should program the master and slave in the same units. In this example, units of 0.001 inch are used. To obtain this result, first determine the correct user units to counts ratios for the master and the slave.*

To determine the ratio, apply the following equation

$$\frac{\text{User Units}}{\text{Counts}} = \left(\frac{\text{Load Movement per Motor Rotation}}{\text{Desired Resolution}} \right) \cdot \frac{1}{\text{Feedback Counts Per Motor Revolution}}$$

For the master axis in the example:

$$\frac{\text{User Units}}{\text{Counts}} = \left(\frac{1 \text{ in}}{1000} \right) \cdot \frac{1}{8192 \text{ Counts}}$$

$$\frac{\text{User Units}}{\text{Counts}} = \left(\frac{1000}{8192} \right) \cdot \frac{\text{in}}{\text{Counts}}$$

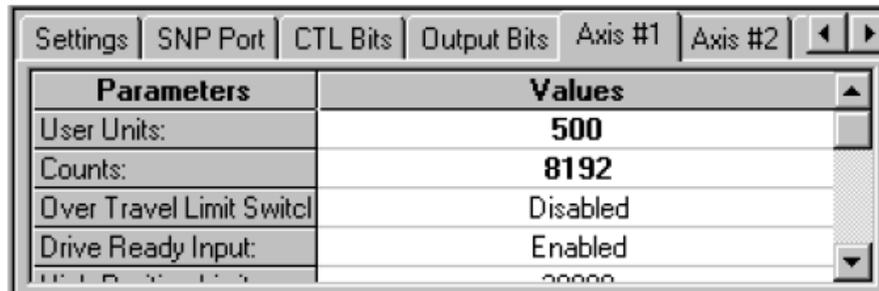
For the slave axis in the example:

$$\frac{\text{User Units}}{\text{Counts}} = \left(\frac{.5 \text{ in}}{1000} \right) \cdot \frac{1}{8192 \text{ Counts}}$$

$$\frac{\text{User Units}}{\text{Counts}} = \left(\frac{500}{8192} \right) \cdot \frac{\text{in}}{\text{Counts}}$$

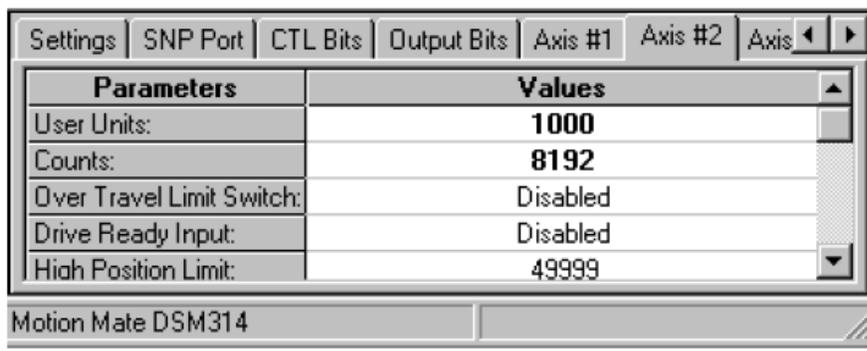
You then need to enter these values in the appropriate locations in hardware configuration. In this example, Axis #2 is the master and Axis #1 is the slave. Therefore, enter the User Units and Count values into hardware configuration for the slave as shown in Figure 153.

Figure 153: Slave User Units/Counts Hardware Configuration



The Master User Units/Counts value would be entered as shown in Figure 154

Figure 154: Windowing Strobes Example



This tells the module the correct scaling to use when it runs motion programs. However, the CAM Editor also needs to know the correct scaling to perform the proper transformations from user units to counts. For this example, this data must be entered into any CAM profiles that are to run on these axes. An example is shown in Figure 155.

Figure 155: Slave and Master User Units/Counts CAM Editor

- Profile Link	
Profile Name	Scale_Non_Cyc
CAM Type	Non-Cyclic CAM
Start Type	Ignored
End Type	Ignored
Master Device Counts	8192
Master User Counts	1000
Slave Device Counts	8192
Slave User Counts	500
Inspector	

It is recommended that the scaling operations be performed before programming any CAMs that do not have one-to-one scaling. This is suggested to avoid the need to reenter data. The CAM editor displays the master/slave data in User Units. Therefore, if you do not define your scaling and enter all the CAM data, by default you have chosen 1 to 1 scaling. When you finally correct this error and enter the correct scaling, you will note that all non-zero numbers in the CAM data tables have changed to reflect the new User Units values.

The following section discusses how the CAM editor rounds values when you are entering data. This function is performed automatically and does not require you to perform or configure the editor in any special way.

Note that, internally, the DSM works in native feedback units and converts the native units to User Units automatically for the user. The module performs this operation to take full advantage of all the available feedback resolution. This includes when a user has chosen to program motion in units that are not the full resolution of the feedback device. The CAM Editor also seeks to maintain all the resolution that is available (without showing false resolution) for a given motor/feedback set. Therefore, when you specify scaling within the CAM editor, the editor will, in some cases, add decimal places to the data table. Additionally, it automatically rounds numbers to values that can be represented as integer numbers of feedback unit counts. The sample cam table in Table 75 is based on the previous example. Note that the CAM Editor displays values in User Units, but always rounds them to an integer value in counts.

Table 75: CAM Example Data Scaled in Inches

Master Position (Inches)	Slave Position (Inches)
0	0
0.075	0.075
0.5	0.25
1	0.5

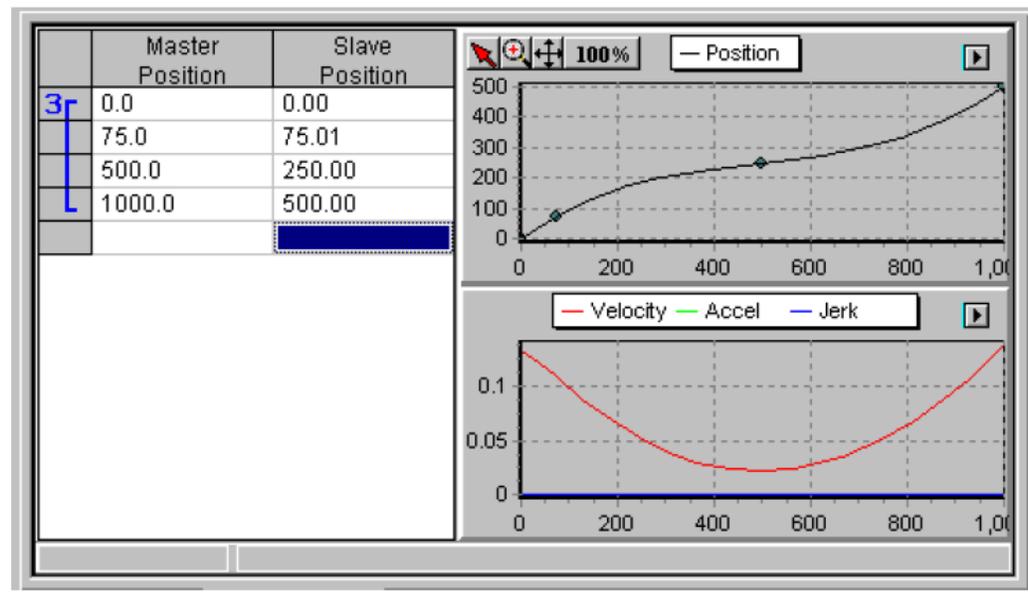
The table above is shown in inches. In this example, the CAM is programmed in 1000th of an inch. Therefore, convert the values and enter the data into the CAM Editor as shown in Table 76.

Table 76: CAM Example Data Scaled in .001 Inches

Master Position (1000 th of In)	Slave Position (1000 th of In)
0 in = 0 thousandth of in	0 in = 0 thousandth of in
.075 in = 75 thousandth of in	.075 in = 75 thousand of in
.5 in = 500 thousandth of in	.25 in = 250 thousandth of in
1 in = 1000 thousandth of in	.5 in = 500 thousandth of in

When you enter the data into the CAM editor (Figure 156), some values are automatically changed by the CAM editor.

Figure 156: CAM Data Table User Units Example



The CAM Editor automatically changes the values to correspond to an integer number of feedback counts. The Editor also rounds the displayed values to limit clutter within the table. Note that the editor maintains the variable's precision and it is only the display that is rounded. The functions that are automatically performed by the CAM editor are illustrated below. First determine the integer feedback counts that are the closest to the desired values. For this example, only two numbers cannot be exactly represented. These are the Master Position of 75 and the Slave Position of 75. The closest integer count value to these values is 614 counts and 1,229 counts for the master and slave respectively. The relationship between the Master Position and Slave Position with respect to User Units and Counts is shown in Table 77.

Table 77: Relationship Between User Units and Counts in Scaling Example

Master Position (User Units)	Master Position (Counts)	Slave Position (User Units)	Slave Position (Counts)
0	0	0	0
74.951171875	614	75.01000000000001	1229
500	4096	250	4096
1000	8192	500	8192

This agrees with the functions that were automatically performed by the editor. Note that for the Master Position of 74.951171875, the editor rounds to 75.0. For the Slave Position of 75.01000000000001, the editor rounds to 75.01.

15.3.10 Synchronization of CAM Motion with External Events

The following mechanisms allow the programmer to synchronize CAM motion with external events:

- The start of CAM motion can be synchronized with an external event by using the existing WAIT command in a motion program.
- A Cyclic CAM can be synchronized with a strobe using Local Logic variables. Refer to Chapter 11-14 for additional information concerning Local Logic.

15.3.11 CAM-Specific DSM Error Codes

Table 78: CAM Specific Error Codes

Error Code (hex)	Response	Description	Error Type	Possible Cause
Cam Program Error Codes				
2A	Normal Stop	Cyclic CAM CTL Exit condition specified for Non-Cyclic CAM	Axis	CTL exit conditions are permitted for Cyclic CAMs only. The motion program contains a non-cyclic CAM instruction with a CTL exit condition.
2B	Normal Stop	CAM Phase out of range	Axis	The CAM PHASE value is outside the axis position range.
CAM Configuration Error Codes				
2D	Normal Stop	CAM Master Axis Config Error – master profile does not match master axis configuration	Axis	The User-Units:Counts ratio specified for the master axis in the Editor and Hardware Config are not compatible and/or The High/Low Position Limit specified for the master axis in Hardware Config is not compatible with the profile. Refer to the section on CAM Types for a detailed description on setting up the High/Low Position Limits.
2E	Normal Stop	CAM Slave Axis Config Error – slave profile does not match slave axis configuration	Axis	The User-Units : Counts ratio specified for the slave axis in the Editor and Hardware Config are not compatible and/or The High/Low Position Limit specified for the slave axis in Hardware Config is not compatible with the profile. Refer to the section on CAM Types for a detailed description on setting up the High/Low Position Limits.
2F	Normal Stop	CAM Slave Axis SW EOT mode cannot be enabled for Cyclic Circular CAM	Axis	
Configuration Parameter Error Codes				
1D	Normal Stop	Attempt to use CAM, CAM-Load, or CAM- Phase commands with Follower Mode	Axis	If using CAM, ensure that Follower Mode is not configured (Follower Mode cannot be used when using CAM).

Error Code (hex)	Response	Description	Error Type	Possible Cause
				If using Follower Mode, ensure that CAM commands are not present in motion program (CAM cannot be used when Follower Mode is configured).
CAM Execution Error Codes				
66	Normal Stop	CAM Profile not found in CAM Download Block	Axis	The Cam profile was not linked to the CAM Download block in the CAM Editor and/or the CAM Download block name was not specified in Hardware Config.
67	Normal Stop	CAM Exit Distance out of range (Non-Cyclic CAMs)	Axis	The exit distance for a Non-Cyclic CAM was greater than the modulus for the CAM.
68	Status Only	(Correction Enabled) Velocity Command Limited due to Velocity Limit violation or Position Error Limit violation	Axis	
68	Normal Stop	(Correction Disabled) CAM velocity command above configured axis velocity limit	Axis	
6A	Normal Stop	CAM Position Error Limit Violation (with Correction Disabled)	Axis	
6B	Status Only	CAM commanded position at the exit different from CAM profile value due to position error or velocity limit	Axis	
6C	Normal Stop	CAM master value out of profile master range for Non-Cyclic profile (CAM and CAM-LOAD commands)	Axis	
6D	Normal Stop	Absolute mode CAM after incremental mode CAM in the sequence	Axis	
6F	Fast Stop	CAM trajectory calculation error	Axis	Contact Emerson

15.4 Electronic Cam Programming Basics

This section contains an introduction to the basic electronic CAM programming concepts. The Local Logic function, and motion programming are not discussed in detail in this section, since they are discussed in other chapters in this manual.

15.4.1 Requirements

The Local Logic, CAM editor, and Motion Program editors are integrated within the programming software environment. You need one of the following software packages:

- CIMPLICITY Machine Edition Logic Developer – PLC version 2.1 or later
- VersaPro version 1.1 or later (Series 90-30 only. For details, refer to Appendix H.) The CAM feature requires DSM314 firmware release version 2.0 or later.

15.4.2 Introduction to Electronic Cam Programming

The electronic CAM function works with the DSM314 motion program, DSM314 Local Logic program, and the Host Controller programming environment. Specifically, the electronic CAM function allows you to specify precise position-to-position relationships between a master axis and a slave axis. This ability is critical to many applications where very tight synchronization between axes is an absolute requirement.

The CAM Editor tool allows you to specify these position-to-position relationships, called profiles, graphically, in tabular form, or a combination of both. These profiles are stored to the DSM module where they are accessed through the DSM motion programs.

CAM Profiles must be linked to their associated CAM block. The CAM block is linked to the DSM via the CAM Block entry in Hardware configuration.

A CAM block can contain numerous CAM profiles. The DSM has two limits that affect the number of profiles. The maximum CAM block size is 50Kbytes, and the maximum number of linked profiles for an individual block is 100. The CAM Profile library is only limited by available disk space on the host computer.

The basic CAM concepts are illustrated with a simple example.

Creating a CAM Application Example

Basic Steps

1. Open the project folder or create a new one
2. Create a CAM block
3. Create a CAM profile
4. Link the CAM profile to the CAM block
5. Configure the CAM profile
6. Specify the CAM Type
7. Specify the Correction Property
8. Save the CAM profile

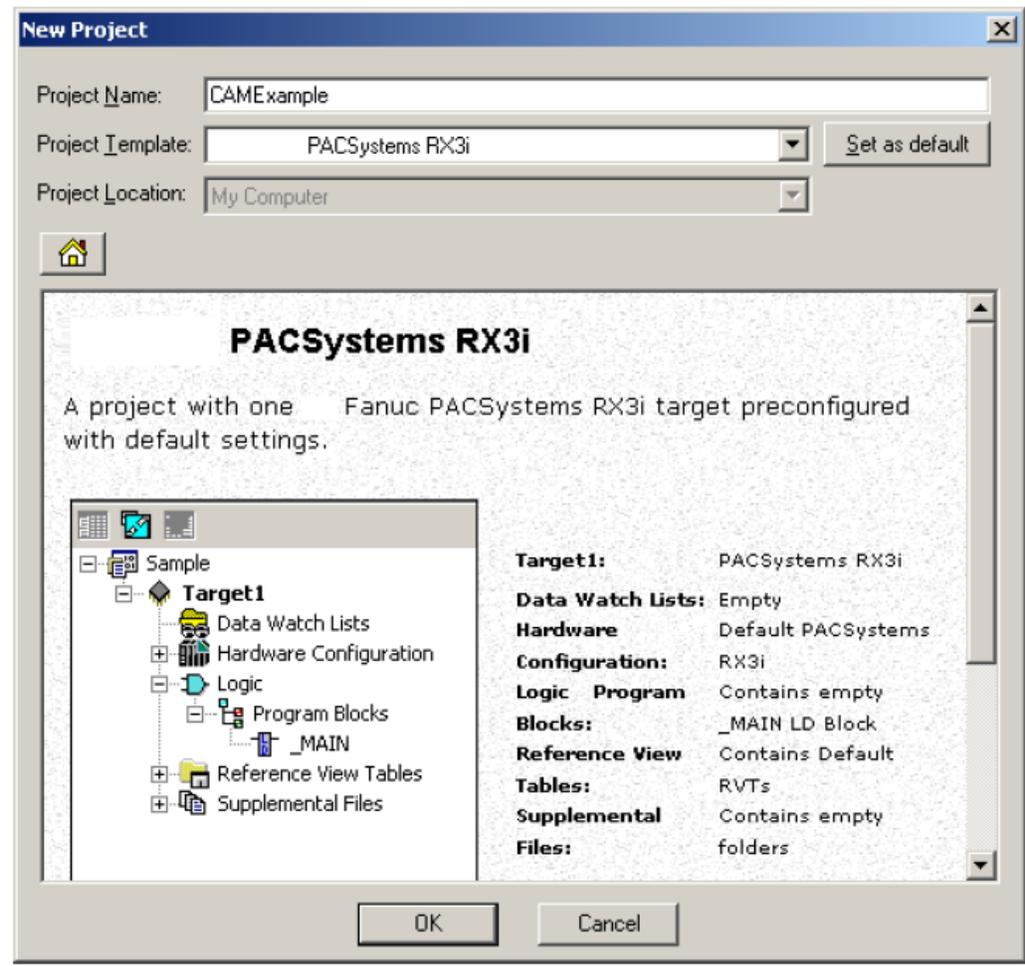
9. Generate motion and Local Logic programs
10. Set up hardware configuration in the configuration/programming software
11. Execute (test) the application

Step 1: Create a Project

For details on creating a project, refer to the on-line help or the software user's manual.

PAC Machine Edition Logic Developer-PLC Getting Started, GFK-1918

Figure 157: New project

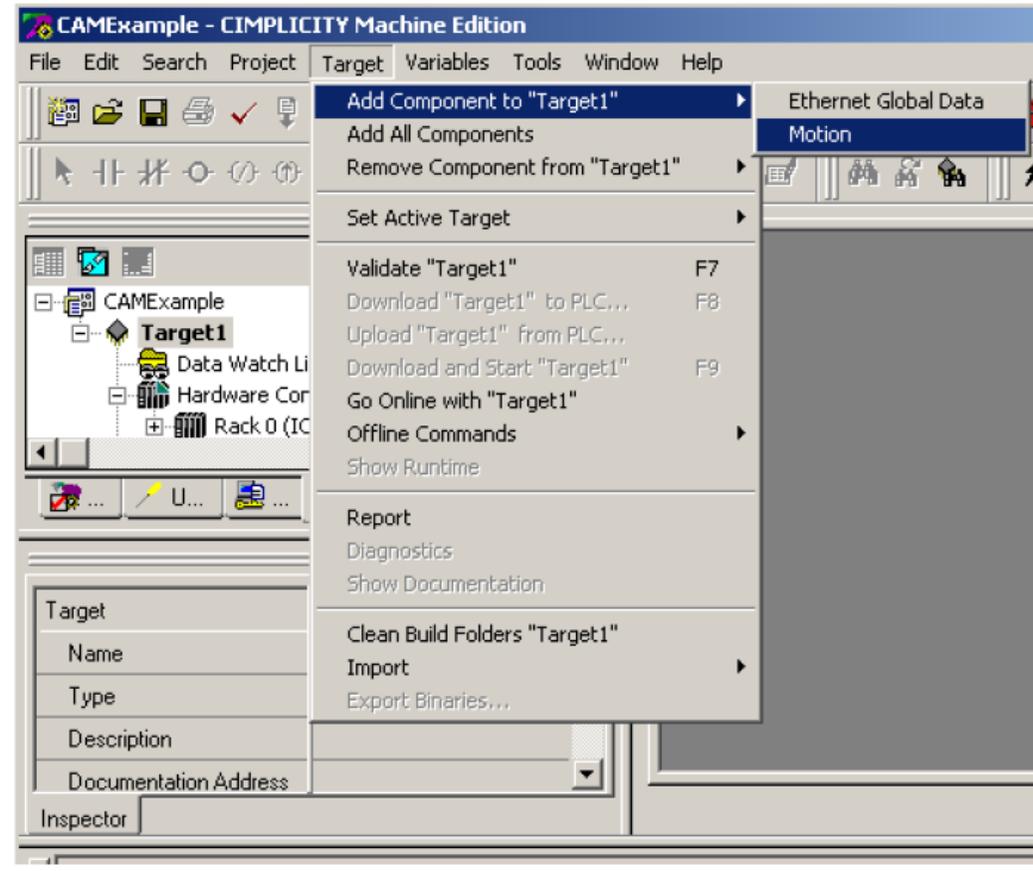


Step 2: Create a CAM Block Using the CAM Editor

The CAM editor is integrated into the Logic developer environment. The editor allows you to easily create, edit, store, and download CAM blocks.

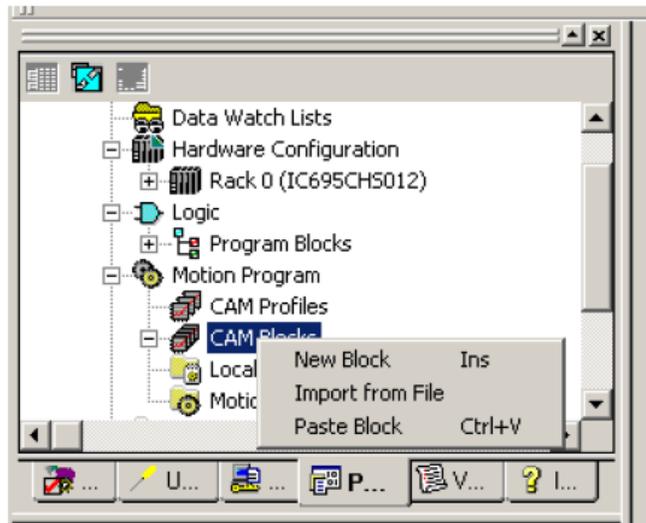
From the Target menu, select Add Component to Target1, then select Motion. This adds the Motion Target to the Navigator portion of the Logic Developer window. If Motion programs or Local Logic programs have already been defined, this step is not necessary.

Figure 158: Create CAM Program



In the Navigator window right-click “CAM blocks” and select “New Block.”

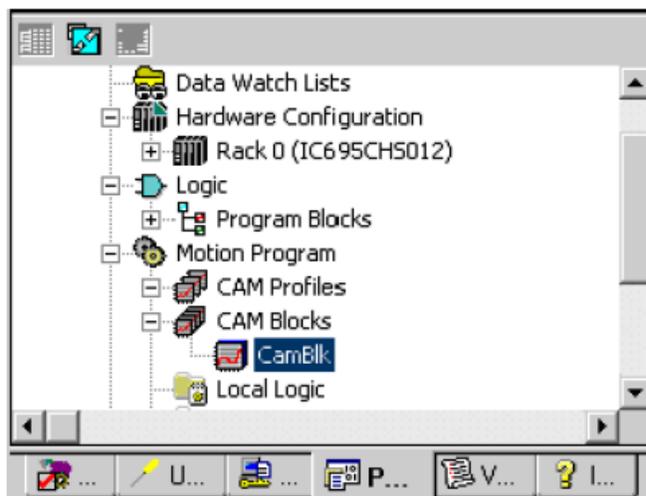
Figure 159: Create New CAM Block



Creating the new block opens up an edit field that allows you to name the block. The rules for naming a CAM block are:

- Only the characters A-Z, a-z, 0-9, and _ (underscore symbol) are allowed. Consecutive underscores are not allowed.
- The block name must begin with a letter or underscore symbol.
- A block cannot have the same name as another block that exists in an open folder.
- A CAM block name may contain up to twenty characters.

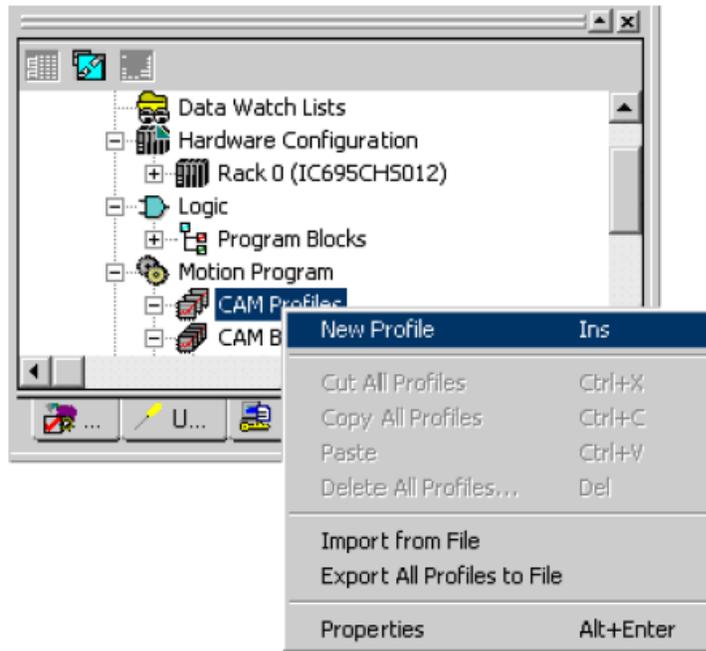
Figure 160: CAM Block Screen



Step 3: Create a CAM Profile

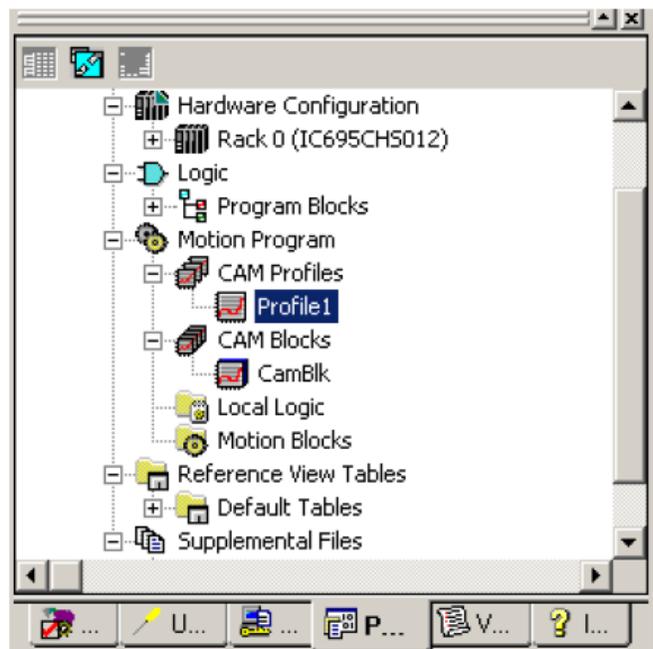
The next step is to create a simple CAM profile. The CAM profiles are linked to CAM blocks. Additional information on this interlinking is contained within the on-line help. To create a CAM profile, right-click the CAM Profiles icon in the Navigator window and select New Profile as shown in the following figure.

Figure 161: Create New CAM Profile



This inserts a new profile named “Profile1” into the library as seen in the following figure.

Figure 162: New Profile Creation



You can rename this profile to a name more suitable to the application if desired. The naming rules are:

- Any alpha-numeric character or the underscore (_) symbol may be used.
- The first character in a profile name must be a letter.
- A profile name cannot be more than 20 characters long.
- A profile is referenced by name in a motion program. NOTE: Logic Developer is not case-sensitive when referencing a profile name.

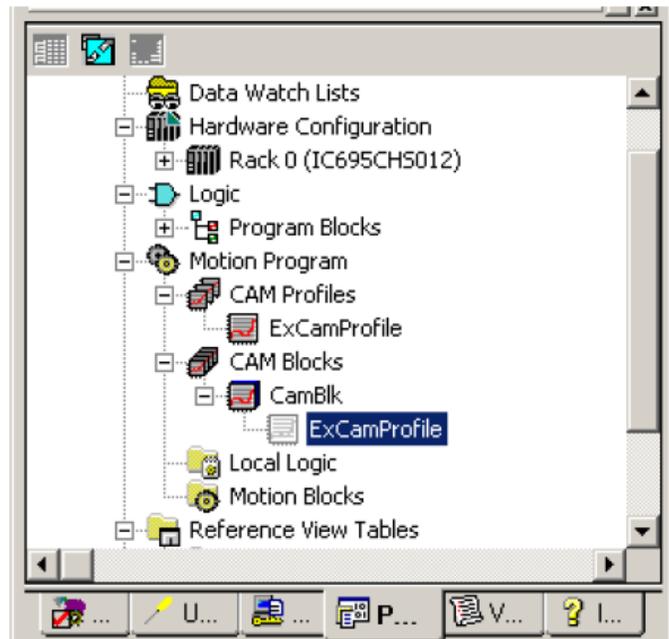
To rename the profile, right-click the profile name in the Navigator window and choose Rename Profile from the short-cut menu. Type a name for the profile and press ENTER to finish. The profile and any CAM profile links to it are renamed. For this example, the profile is renamed to ExCamProfile. Refer to the on-line help for additional information.

Step 4: Link the CAM Profile to the CAM Block

Although there is more than one way to link a CAM profile to a CAM block, the easiest method is to click the desired CAM profile in the Navigator, then drag it and drop it on the applicable CAM block. The result is shown in the next figure.

Note: Logic Developer limits the download block total size (Motion, Local Logic, and CAM combined) to 32K.

Figure 163: Linking a Profile to a CAM Block



Step 5: Configure CAM Profile Data Points

Once these operations are complete, you must configure the CAM profile. A CAM profile is composed of a series of Points that defines the relationship between the master position and the slave position. Each point is defined by two coordinates. When viewing the graphical representation, the Master coordinate represents the horizontal axis and the Slave coordinate represents the vertical axis, as shown in the next figure.

Begin by double-clicking the profile to open it in the Profile Editor window (see next figure), which has two editors:

- **Table Editor** is similar to a spreadsheet. In the table, each point has its own row with two columns, one for the Master position and one for the corresponding Slave position. When a new profile is opened, there are, by default, only two points, a start point and an end point. The start point is the top point of the table and the end point is at the bottom.
- **To edit points with the Graphical Editor**, click the point on the graph and drag it to the desired location. (NOTE: The point data in the table editor will update to the new position.) To perform other tasks in the graphical editor, right-click in the graph and select the applicable task from the short-cut menu.

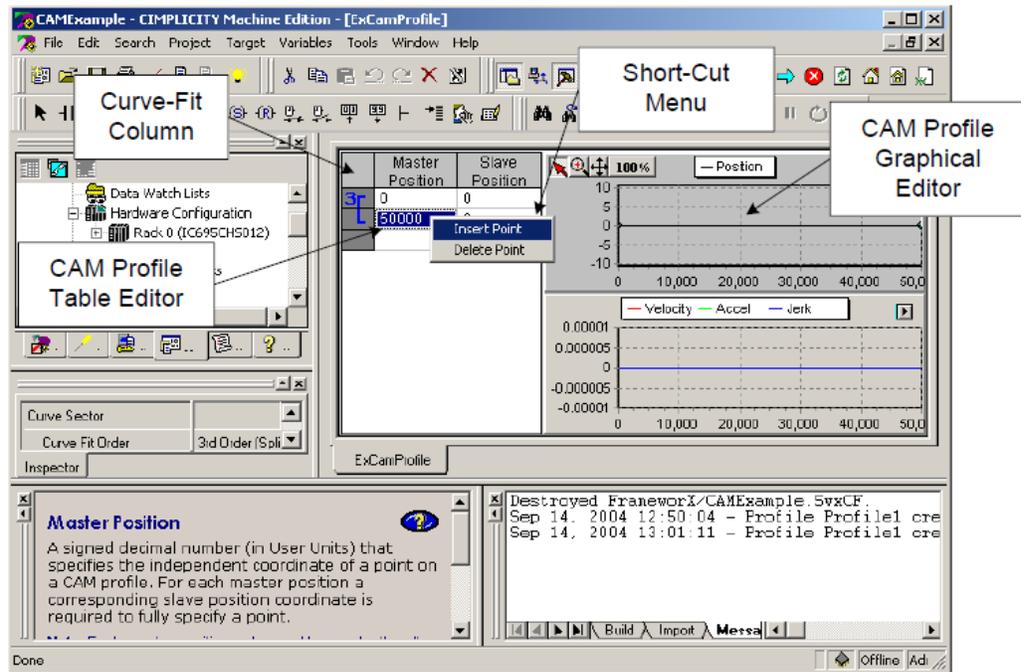
The next step is to edit the end point (the bottom point in the table) for the Master and Slave. In the Table Editor, click in the end point's Master column and enter the value 50000; then click in the end point's Slave column and enter the value 0. (NOTE: As points are added or changed in the Table Editor, the graph in the Graphical Editor will update accordingly.)

Next, insert an additional point into the Editor table. Right-click in the Master column of the end point and choose Insert Point from the short-cut menu (shown in the next figure). A new row is added above the end point row, specifying a new point with master and slave values, by default, midway (25000 and 0, respectively) between the values of the two existing adjacent (above and below) points. Change the values for this point to 47500 for the Master and 11000 for the Slave. To change a point value, click it, type in the new number, then either press the Enter key, or click outside of the table.

To change the Curve-Fit order, click the Curve-Fit column, then select the Curve-Fit Order in the Property Inspector window. Also, a profile can be split into multiple sectors or multiple sectors merged into one by right clicking on the Curve-Fit display and choosing from the short-cut menu.

Note: A CAM profile is limited to 400 points if it contains second or third order sectors. A CAM profile is limited to 5000 points if it only contains first order sectors.

Figure 164: Inserting a Point in the Profile Editor Window



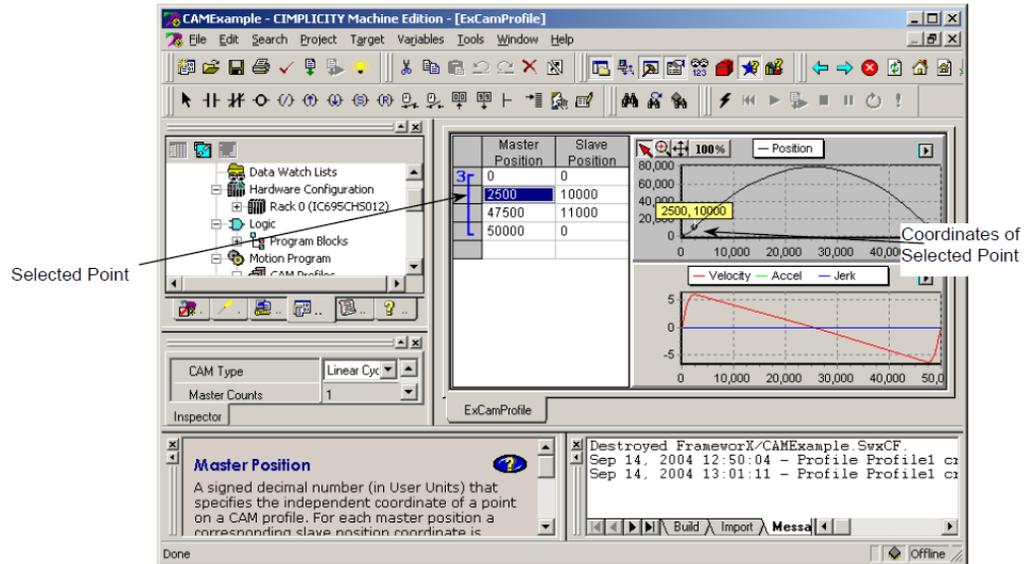
Since the Slave Position end point is the same value (0) as the initial Slave Position point, this CAM meets the requirements for a Linear Cyclic CAM. (For more information on the different CAM types, refer to “CAM Types” on page 326.) Note that the CAM Editor has several “Smart” edit fields that will ONLY display the choices that are valid for a given data set. For example, since a requirement for a Linear Cyclic CAM is that the Slave Position start point and Slave Position end point are the same, the editor only allows the Linear Cyclic CAM choice if these criteria is met.

Next, insert a new point into the profile and then edit the point. The point can be edited either in the profile table or graphically on the plot. Insert the point as shown above and in Figures 165 and 166 by right clicking the point below the insertion position and selecting Insert Point from the menu. Then change the default values to 2500 for the Master and 10000 for the Slave.

Figure 165: CAM Profile Table Data

	Master Position	Slave Position
3	0	0
	2500	10000
	47500	11000
	50000	0

Figure 166: CAM Editor Example (Linear Cycle CAM)



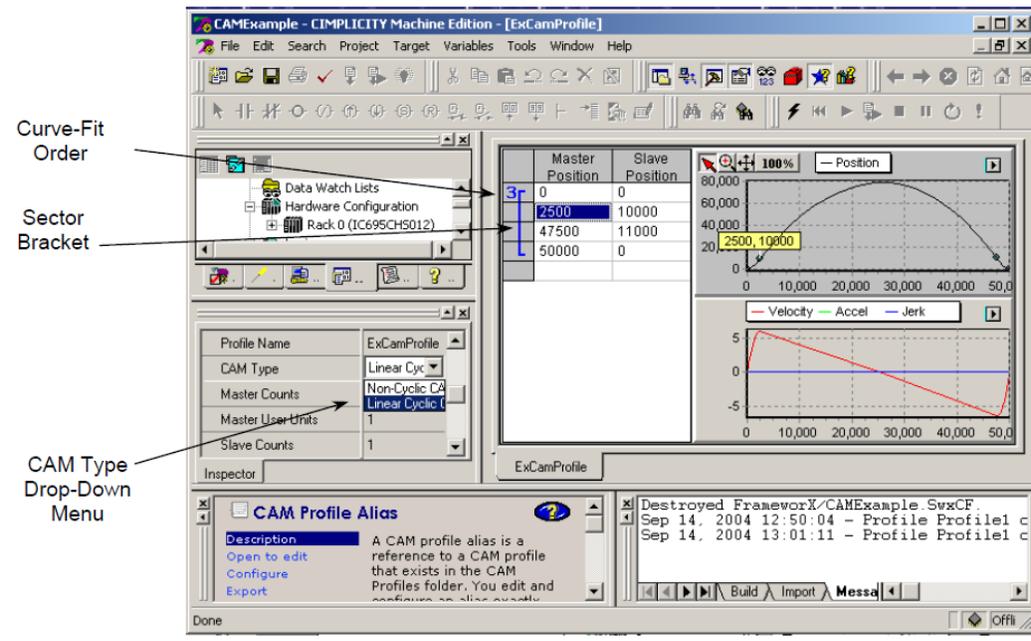
There are numerous other features in the editor. These include being able to define additional sectors that each have a different curve fit method. These editor features are discussed in the programming software's on-line help. Please refer to this source for additional information.

Step 6: Specify the CAM Type

For this example, the CAM will be Linear Cyclic, as discussed previously. Use the following procedure:

- In the Project tab of the Navigator, right-click a CAM profile. The short-cut menu appears.
- From the short-cut menu, choose Properties. The Inspector opens showing the CAM profile's properties.
- In the Inspector, click the arrow in the CAM Type field. The CAM Type drop-down list appears.
- Choose 'Linear Cyclic CAM' from the list (Figure 167).

Figure 167: CAM Editor CAM Type Selection

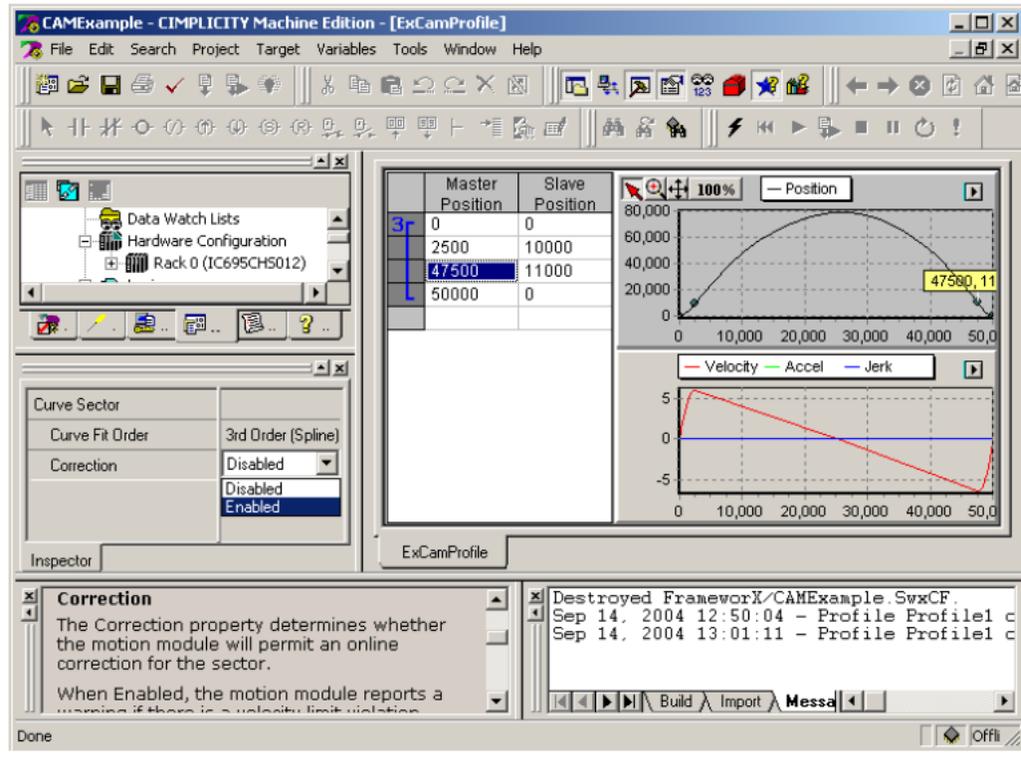


Step 7: Specify the Correction Property

The last item to be specified for this example is the correction status. The Correction property determines whether the motion module will permit an online correction for a specific sector. A sector is a region of a CAM profile defined by at least two adjacent user-defined points. The sector includes the user-defined points, the curve connecting them and also up to, but not including, the first point defined for the next adjacent sector. The points included in a given sector are denoted by the Sector Bracket, are shown in the figure above. Each sector is assigned a curve-fit order number, also shown in the figure above. The segments of the profile between user-defined points are defined by polynomials of the curve-fit order specified. A unique polynomial is used to interpolate between each pair of adjacent user-defined points. Although the actual polynomial coefficients can be different for each segment, the curve-fit order is the same throughout the sector. A sector is indicated in the CAM profile table as a bar spanning the user-defined Master Position values included in the sector. Initially, all points defined in a profile are included in a single sector. This single initial sector can be subdivided as required to facilitate smoothing a CAM profile. When the Correction property is Enabled, the motion module reports a warning if there is a velocity limit violation. When the Correction property is set to Disabled, the motion module reports an error for these violations and stops the slave axis.

For this example, correction should be enabled. To enable correction, select the sector from the CAM profile table by clicking it. This will cause the Inspector window to display the sector properties and allow them to be edited. Select the Correction drop down box and choose Enabled (Figure 168).

Figure 168: CAM Editor Correction Enable



Step 8: Save the CAM Profile

At this point, a simple CAM profile is defined. To save the CAM blocks/profiles, select the File main menu item followed by the Save Project submenu selection. The CAM editor has many more additional features and functionality. Refer to the online documentation for a detailed description of these features.

Step 9: Generate Motion and Local Logic Programs

The next items to be generated are a motion program and Local Logic program that will work with this CAM profile. For this example, the logic must work with a DSM3214 controlling two axes. Axis #1 will be the slave, and Axis #2 will be the master. Therefore, there will be two motion programs. The Axis 1 program, for the slave, will do some base initialization, load the slave starting point for the given CAM profile, and then execute the CAM command. The Axis 2 program, for the master source, is a simple program that will initialize and then wait for the slave to be ready. It will then execute a series of moves.

The program stops at points described within the CAM master such that it is easy to verify that the slave axis is correctly executing the CAM profile. This example also requires a Local Logic program. In this example the Local Logic program serves a supervisory role over the CAM slave and CAM master motion programs. Thus, the Local Logic synchronizes the two programs.

Consult the applicable chapters in this manual for additional details on these features. The motion program and Local Logic programs for this example are as follows:

// Motion program for example CAM block

// Slave Axis

```
Program 1 AXIS1
VELOC 10000 // Set Velocity
ACCEL 10000 // Set Acceleration
100: WAIT CTL01 // Wait For LL to Say Master is ready
110: CAM-LOAD "ExCamProfile", P006, ABS // Load Param. Reg. with Slave Pnt that
corresponds to current Master Position
120: PMOVE P006, ABS, LINEAR // Move Slave Axis to the Position that
corresponds to Start of Table
130: CAM "ExCamProfile", 50000, ABS // Execute CAM Statement
140: PMOVE 0,ABS,S-CURVE // Move back to zero
150:
ENDPROG
```

// Master Axis Program

Program 2 AXIS2

```
VELOC 10000 // Set Velocity
ACCEL 10000 // Set Acceleration
200: PMOVE 0 ,ABS,S-Curve // Start at zero
210: WAIT CTL08 // Master Waits Until Slave in Position
220: PMOVE 2500,ABS,LINEAR // Move 1st Master Point in Table
230: DWELL 5000 // Wait 5 Sec
240: PMOVE 47500,ABS,LINEAR // Move to 2nd Point
250: DWELL 5000 // Wait 5 Sec
260: PMOVE 2500,INC,LINEAR // Finish Distance Specified in CAM Cmd 1st CAM
Complete
270: PMOVE 0,ABS,LINEAR // Move back to zero
280:
ENDPROG
```

// Local Logic Program for CAM Example

```

CTL01 := 0; // Outputs written when logic completes initialize to
// zero to allow toggle to true

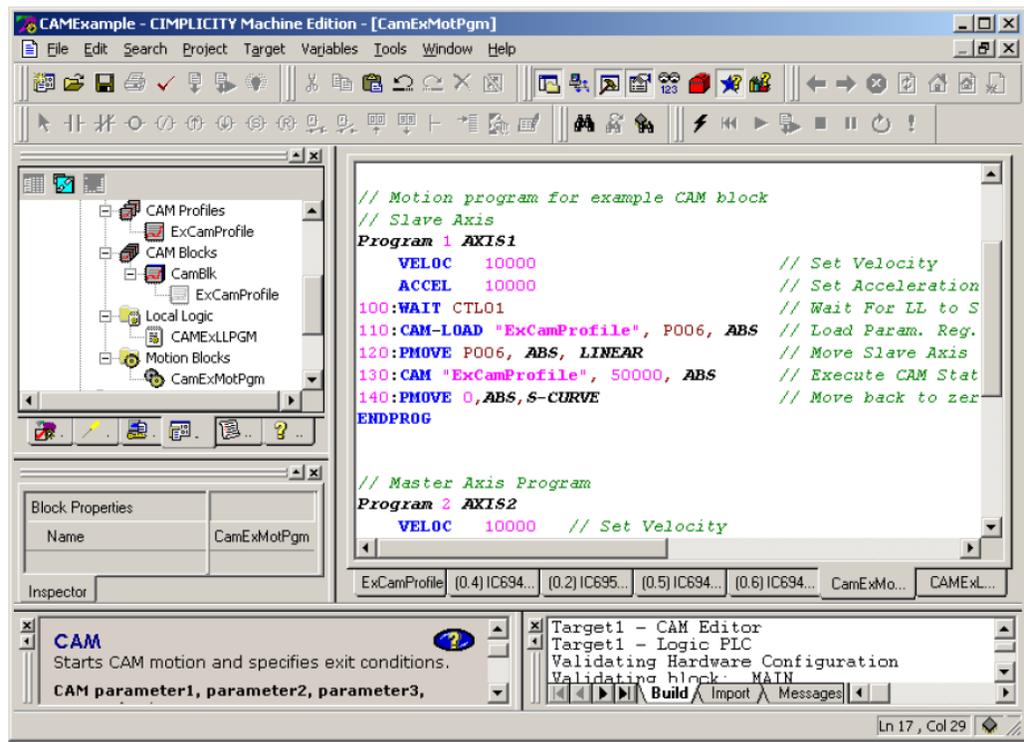
CTL08 := 0; // Outputs written when logic completes initialize to
// zero to allow toggle to true

// Control Logic for Program 1 and 2
// Program 1 = Slave
// Program 2 = Master
IF PROGRAM_ACTIVE_2=1 THEN // Make sure Program 2 is active
  IF BLOCK_2=210 THEN // Indicates Master is Ready to Start CAM
    IF PROGRAM_ACTIVE_1=1 THEN // Check that Program on Axis 1 is active
      IF BLOCK_1=100 THEN // Block indicates Slave ready for CAM-Load
// Sequence
        CTL01:=1; // Signal Slave to perform CAM Load sequence
      END_IF;
      IF BLOCK_1=130 THEN // Block indicates Slave has completed initialization
// and CAM-Load Sequence
        CTL08:=1; // Signal Master and Slave that both Axes are ready
// to start CAM sequence
      END_IF;
    END_IF;
  END_IF;
END_IF;
// End Control Logic for Program 1 & 2

```

After completing the program entry, the resulting Logic Developer screen should look similar to Figure 169.

Figure 169: CAM Editor Correction Enable



Step 10: Set up Hardware Configuration in Logic Developer

Once a successful syntax check is completed for the local logic and motion programs, you need to set up the hardware configuration that will allow the example program to be downloaded to the DSM314 module. Most users will first set up their hardware configuration and then generate the programming statements. However, because this example is intended to illustrate the CAM feature the order is reversed to better illustrate the link the CAM block name and the DSM314 hardware configuration.

Configure the power supply, CPU, and DSM314 module that are appropriate for your installation. For general information on hardware configuration, refer to chapter 4.

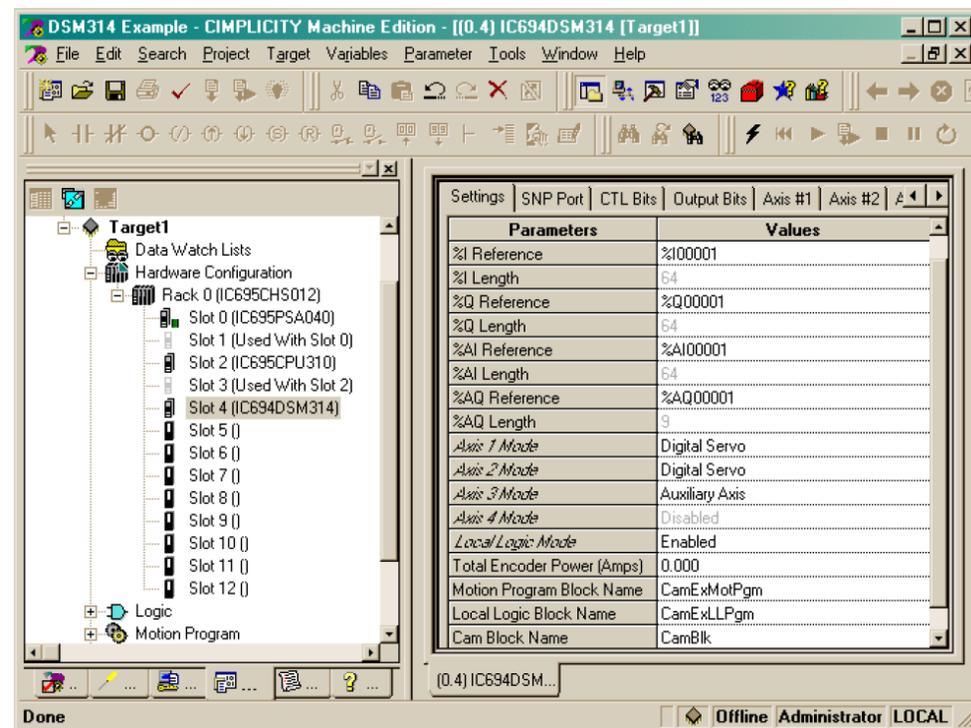
Change the following Settings tab parameters to the values shown. (Axis 1 and Axis 2 modes are set to digital servo because this example uses the βis 0.5 digital servo.)

Axis 1 Mode	Digital Servo
Axis 2 Mode	Digital Servo
Local Logic Block Name	CamExLLPgm
Cam Block Name	CamBlk
Local Logic Mode	Enabled

The resulting Settings tab will be as shown in Figure 170 .

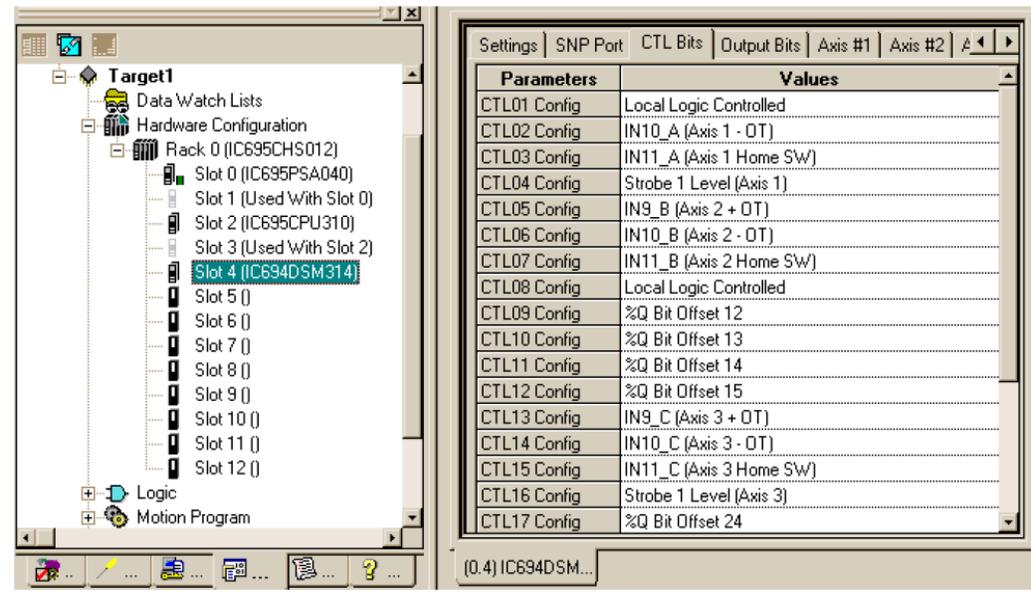
Note: This example uses only one DSM314. The DSM314 executes the files (CAM, Local Logic, and Motion Program) pointed to by the configuration. Multiple DSM314 modules can run the same Local Logic program, motion programs, or CAM Blocks. This allows you to have one source file for multiple DSM314 modules. Note that this does not prevent DSM314s from executing different programs.

Figure 170: Hardware Configuration 90-30 rack DSM314 Settings Tab



In this example, the Local Logic program will control CTL01 and CTL08. Because CTL01 and CTL08 are used to signal the Motion Programs, you must configure these CTL bits to be under Local Logic Control. To do this, access the CTL Bits tab in the hardware configuration and set CTL01 Config and CTL08 Config to Local_Logic_Controlled. The resulting Hardware Configuration screen is shown in Figure 171.

Figure 171: Hardware Configuration 90-30 rack DSM314 CTL Bits Tab



You also need to indicate to Axis #1 that it will use the Axis #2 commanded position as its CAM Master source. To do this select, the Axis #1 tab in hardware configuration. Go to the CAM Master Source data entry field. From the drop-down box, select Cmd Position 2. This will configure Axis #1 to use the Axis #2 commanded position as its CAM master source (Figure 172). While in this tab, change the Home Mode to Move + and OverTravel Switch to Disabled.

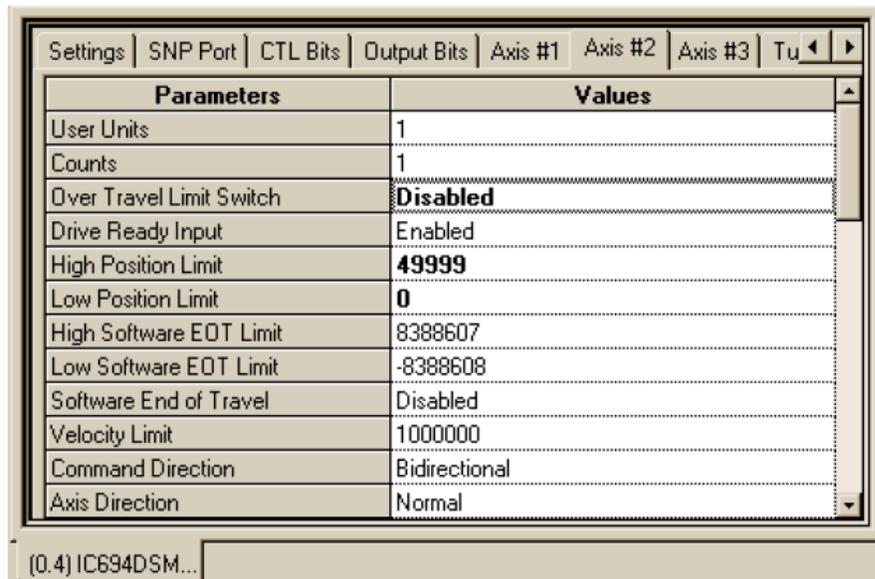
Figure 172: CAM Slave Master Source Selection

Parameters	Values
Over Travel Limit Switch	Disabled
Drive Ready Input	Enabled
High Position Limit	8388607
Low Position Limit	-8388608
High Software EOT Limit	8388607
Low Software EOT Limit	-8388608
Software End of Travel	Disabled
Velocity Limit	1000000
Command Direction	Bidirectional
Axis Direction	Normal
Feedback Source	Default
Feedback Mode	Incremental
Reversal Compensation	0
Drive Disable Delay (ms)	100
Jog Velocity	1000
Jog Acceleration	100000
Jog Acceleration Mode	Linear
Home Position	0
Home Offset	0
Find Home Velocity	2000
Final Home Velocity	500
Home Mode	Move +
Return Data 1 Mode	0h
Return Data 1 Offset	0
Return Data 2 Mode	0h
Return Data 2 Offset	0
Cam Master Source	Cmd Position 2
Follower Control Logic	Disabled

(0.4) IC694DSM...

You also need to indicate to Axis #2, the rollover points for the Master axis position reference. To do this, select the Axis #2 tab in hardware configuration. Input 49,999 into the High Position Limit and 0 into the Low Position Limit data entry fields. Note that since this is a Cyclic CAM, the master source high limit, by definition, must be one less than the last point in the master data table. In this example, this is point 50,000. Thus, the high limit is equal to 49,999. One way to envision this principle is to think of a Cyclic CAM Master as a continuous circular strip where the first point on the strip is the same as the last point on the strip. Therefore, in this example, 50,000 is the same point as zero. While in this tab, change the Home Mode: to Move + and OverTravel Switch to Disabled.

Figure 173: CAM Master Axis Scaling



Parameters	Values
User Units	1
Counts	1
Over Travel Limit Switch	Disabled
Drive Ready Input	Enabled
High Position Limit	49999
Low Position Limit	0
High Software EOT Limit	8388607
Low Software EOT Limit	-8388608
Software End of Travel	Disabled
Velocity Limit	1000000
Command Direction	Bidirectional
Axis Direction	Normal

To finish the configuration, go to the Tuning#1 and Tuning #2 tabs and enter the following values:

- **Motor Type:** 281
- **Position Error Limit:** 200 (Optional; see Configuration information for additional information)
- **In Position Zone:** 20 (Optional; see Configuration information for additional information)
- **Pos Loop Time Const:** 200 (Note: Based upon application/mechanics reference Chapter 4 and Appendix D)
- **Velocity FeedForward:** 9000 (Note: Based upon application/mechanics reference Chapter 4 and Appendix D)
- **Vel Loop Gain:** 32 (Note: Based upon inertia attached to motor. Typical demo cases have a indicator wheel attached that represents approximately this inertia to the motor)

The resulting display should be similar to Figure 174.

Figure 174: Hardware Configuration Tuning#1 Tab

Parameters	Values
Motor Type	281
Position Error Limit	200
In Position Zone	20
Pos Loop Time Constant (0.1ms)	200
Velocity at Max Cmd	100000
Velocity Feed Forward (.01%)	9000
Acceleration Feed Forward (.01%)	0
Integrator Mode	Off
Integrator Time Constant (ms)	0
Velocity Loop Gain	32

The Tuning tab for Axis #2 should also be set up as shown for Axis #1.

This completes the configuration changes necessary for the example.

The link between the sample CAM Block, Motion program and Local Logic program, and the DSM314 module are now complete. Create any required Host Controller ladder logic programming, then Validate the programs and download them to the Host Controller. Additional information concerning the download operation is shown in the Logic Developer on-line help.

Step 11: Execute (Test) Your CAM-Based Motion Program

⚠ WARNING

Before testing your application on actual machinery, you must first verify that it is safe to do so. This includes insuring that all devices are securely mounted, all safety equipment is installed and operational, and personnel in the area have been notified. Failure to address all safety-related issues could result in injury to personnel and damage to equipment.

Once the download operation is complete, the module is ready to execute the CAM Blocks, motion programs and Local Logic program. Use the following procedure:

1. Place the Host Controller in run mode.
2. Enable the servo drives. To enable Axis #1, toggle the %Q offset 18 bit. To enable Axis #2, toggle The %Q offset 34 bit. Based upon the current module error status, you may also have to initiate a clear error routine by toggling the %Q offset 0 bit.
3. Have both axes perform a find home routine by toggling the %Q offset 19 bit (find home Axis #1) and the %Q offset 35 bit (find home Axis #2). At this point, both axes

will perform a find home cycle. Wait until this completes for both axes and the Position Valid %I bits turn on. The Position Valid %I bit for Axis 1 is the %I offset 17 bit (the 18th %I bit), and for Axis 2 is the %I offset 33 bit (the 34th %I bit). The resulting display is shown in the following figure.

Figure 175: RVExample Screen

	Binary	00000000	%Q00002	Address
	00000000 00100001 00000100 00100111 00000100 00100111 10000011 00000100			%I00001
	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001			%I00065
	00000000 00000000 00000001 00000100 00000000 00000100 01100000 00000000			%Q00001
	+0, +0, 0, 16#0000, +0, +0, +0, 16#0000			%AT0001
	+0, +0, +0, +0, +0, +0, +0, +0			%AT0011
	+0, +0, 0, 16#0000, +3, +0, +0, +0			%AT0021
	+0, +0, +0, +0, +0, +0, +0, +0			%AT0031
	+0, +0, 0, 16#0000, -141, +0, +0, +0			%AT0041
	+0, +0, +0, +0, +0, +0, +0, +0			%AT0051
	+0, +0, 0, 16#0000, +0, +0, +0, +0			%AT0061
	+0, +0, +0, +0, +0, +0, +0, +0			%AT0071
	+0, +0, 0, 16#0000, +0, +0, +0, +0			%AT0081
	+0, +0, +0, +0, +0, +0, +0, +0			%AT0091
	16#6550, +13, 16#6450, +10000, 16#4027, +0, 16#4027, +0			%A00001
	+0, +0, +0, +0, +0, +0, +0, +0			%A00011

4. Enable Local Logic by setting the %Q offset 1 bit from the Host Controller. If there are no errors, you can then execute the motion programs.
5. Execute Program 1 by toggling %Q offset 2 bit. The motor connected to Axis #1 should then begin to execute Motion Program #1.
6. Execute Program 2 by toggling %Q offset 3 bit. The motor connected to Axis #2 should begin to execute Motion Program #2.
7. The motors will execute the statements until they reach the first DWELL, where you can visually verify that it followed the CAM profile correctly. The display should be similar to the following figure. Note that the commanded position for Axis#2 equals 2500, while the commanded position for the slave corresponds to the CAM table and has the value 10,000.

Figure 176: RVExample Screen First Dwell

Signed Decimal		00000000000000000001001110001000						%AI0007	Address
00000000	00100001	00000100	00101111	00000100	00111111	10000011	00000110	%I00001	
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000001	%I00065	
00000000	00000000	00000001	00000100	00000000	00000100	01100000	00001110	%Q00001	
+10000	+10000	130	16#0000	+0	+0	+0	16#0000	%AI0001	
+0	+0	+0	+0	+0	+0	+0	+0	%AI0011	
+2500	+2500	230	16#0000	-22	-22	-22	-22	%AI0021	
-4	+0	-2	+0	+0	+0	+0	+0	%AI0031	
+0	+0	0	16#0000	-214	-214	-214	-214	%AI0041	
+0	+0	+0	+0	+0	+0	+0	+0	%AI0051	
+0	+0	0	16#0000	+0	+0	+0	+0	%AI0061	
+0	+0	+0	+0	+0	+0	+0	+0	%AI0071	
+0	+0	0	16#0000	+0	+0	+0	+0	%AI0081	
+0	+0	+0	+0	+0	+0	+0	+0	%AI0091	
16#6550	+13	16#6450	+10000	16#4027	+0	16#4027	+0	%AQ0001	
+0	+0	+0	+0	+0	+0	+0	+12	%AQ0011	

Once the dwell time is finished, the motors will continue executing the statements until they reach the second DWELL where you can visually verify that it followed correctly. The display should be similar to Figure 177. Note that the commanded position for Axis#2 equals 47500, while the slave commanded position corresponds to the CAM table and has the value 11,000.

Figure 177: RVExample Screen Second Dwell

Signed Decimal		000000000000000001010101111000		%AI007	Address			
00000000	00100001	00000100	00101111	00000100	00111111	10000011	00000110	%I0001
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000001	%I0005
00000000	00000000	00000001	00000100	00000000	00000100	01100000	00001110	%Q0001
+11000	+11000	130	16#0000	+0	+0	+0	16#0000	%AI0001
+0	+0	+0	+0	+0	+0	+0	+0	%AI0011
+47500	+47500	250	16#0000	+27	+27	+27	+27	%AI0021
+0	+0	+0	+0	+0	+0	+0	+0	%AI0031
+0	+0	0	16#0000	-136	-136	-136	-136	%AI0041
+0	+0	+0	+0	+0	+0	+0	+0	%AI0051
+0	+0	0	16#0000	+0	+0	+0	+0	%AI0061
+0	+0	+0	+0	+0	+0	+0	+0	%AI0071
+0	+0	0	16#0000	+0	+0	+0	+0	%AI0081
+0	+0	+0	+0	+0	+0	+0	+0	%AI0091
16#6550	+13	16#6450	+10000	16#4087	+0	16#4027	+0	%A0001
+0	+0	+0	+0	+0	+0	+0	+0	%A0011

When the master axis reaches 50000 (47500 +2500), the CAM command will exit, the slave axis will decelerate at the programmed acceleration rate and come to a halt, and both axes will return to zero.

Details on the DSM314's %AI, %AQ, %I, and %Q memory are provided in Chapter 5.

Appendix A: Error Reporting

A-1 DSM314 Error Codes

The DSM314 reports error codes in these %AI table locations:

%AI Table Location	Data Reported	Usage
00	Module Status Code	Errors not related to a specific axis
04	Axis 1 Error Code	Errors related to Axis 1
24	Axis 2 Error Code	Errors related to Axis 2
44	Axis 3 Error Code	Errors related to Axis 3
64	Axis 4 Error Code	Errors related to Axis 4

Each error code is a hexadecimal word that describes the error indicated when the Module Error Present %I status bit is set.

A-1.1 Module Status Code Word

The **Module Status Code** %AI status word reports the following two categories of errors:

- Module errors that are not related to a specific axis. Examples of such errors would be a self-test detected hardware failure or a request to run an empty or invalid program. A new Module Status Code will not replace a previous Module Status Code unless the new Module Status Code has Fast Stop or System Error priority. These can be cleared with the %Q Clear Error bit.
- System Status Errors. These are of the format Dxxx, Exxx, and Fxxx. If one of these codes is present, the module will not operate and the %Q Clear Error bit will not clear the error. See the section “System Status Errors” later in this appendix for details.

A-1.2 Axis Error Code Words

All axis-specific motion related errors are reported in the proper Axis Error Code %AI status word. Whenever the Module Error Present %I status bit is set, all error words (including Module Status Code) should be checked for a reported error. A new Axis Error Code will replace a previous Axis Error Code if it has equal or higher priority (Warning, Normal Stop, Fast Stop) compared to the previous Axis Error Code.

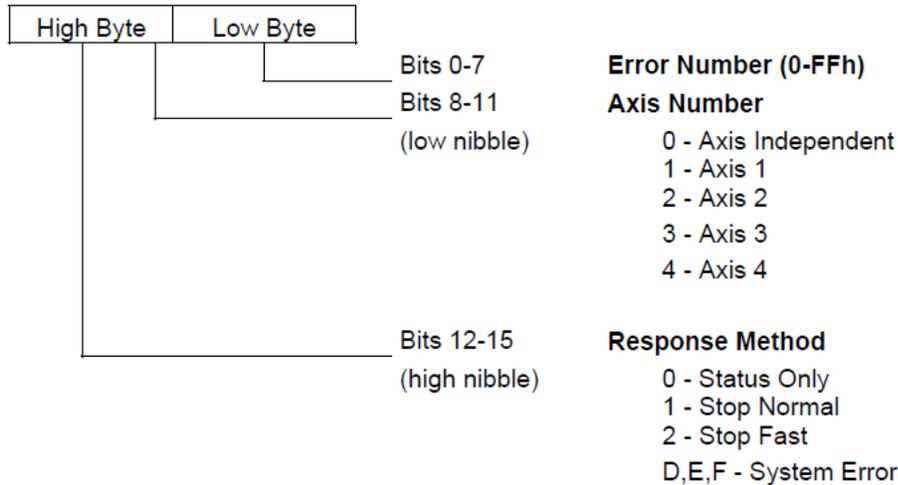
Error codes that stop the axis will clear the Axis OK %I bit for that axis. User logic that sends %Q or %AQ commands to an axis should normally be qualified by the applicable %I Axis OK bit. If Axis OK is off, the axis will not respond to any %Q bit or %AQ commands other than Clear Error or Load Parameter. The %Q Clear Error bit will always clear the Axis Error Code; however, if the condition that caused the error still exists, the error will immediately be reported again.

Note: *The STAT LED on the faceplate of the module flashes slow (four times/second) for Status Only errors and fast (eight times/second) for errors that cause the servo to stop. In the case of a fatal hardware error being detected at power-up, the STAT LED will flash an error code, which should be reported to Emerson. See “LED Indicators” later in this chapter for more details.*

A-1.3 Error Code Format

All error codes are represented as hexadecimal data with the following format:

Figure 178: Status Code Organization



A-1.4 Response Methods

1. **Status Only Errors:** Set the Module Error Present %I bit and Module Status Code or Axis Error Code %AI word, but do not affect motion.

Note: Unless otherwise noted, any command that causes a Status Only Error is ignored.

2. **Stop Normal Errors:** Perform an internal abort of any current motion using current **Jog Acceleration** and **Jog Acceleration Mode** (LINEAR or S-CURVE). The Drive Enabled and Axis Enabled %I bit are each turned OFF after the configured **Drive Disable Delay**.
3. **Stop Fast Errors:** Instantly abort all motion by setting the servo velocity command to zero. The Drive Enabled and Axis Enabled %I bits are each turned OFF after the configured **Drive Disable Delay**.

System Errors (displayed in Module Status Code only): The DSM is disabled and will not respond to PLC control. System errors cannot be cleared until a new configuration is sent to the DSM.

Table 79: DSM314 Error Codes

Error Code (hex)	Response	Description	Error Type	Possible Cause
00	None	No Error	All	
Configuration Errors				
02	Status Only	Scaled data too big, maximum value in range used	Axis	Check DSM axis configuration in HWCFG
03	Status Only	Home Position > Positive EOT, Positive EOT used	Axis	Check DSM axis configuration in HWCFG
04	Status Only	Home Position < Negative EOT, Negative EOT used	Axis	Check DSM axis configuration in HWCFG
05	Status Only	Tuning parameter row 1 invalid; data ignored	Axis	Check DSM tuning configuration in HWCFG Advanced Tab Row 1
06	Status Only	Tuning parameter row 2 invalid; data ignored	Axis	Check DSM tuning configuration in HWCFG Advanced Tab Row 2
07	Status Only	Tuning parameter row 3-16 invalid; data ignored	Axis	Check DSM tuning configuration in HWCFG Advanced Tab One or more than one parameter in row 3-16 is invalid
0A	System Error	Output written by Local Logic is not configured for Local Logic control	Module	A Local Logic block name is specified in the configuration and the Hardware Configuration (Output Bits Tab) for the module does not have the required output configured for Local Logic control.
0B	System Error	CTL bit written by Local Logic is not configured for Local Logic control	Module	A Local Logic block name is specified in the configuration and the Hardware Configuration (CTL Bits Tab) for the module does not have the required CTL bit configured for Local Logic control.
Configuration Parameter Errors				
17	Status Only	EOT Adjust Error	Axis	Software End of Travel is enabled in configuration and the High or Low Software End of Travel values are set outside the High or Low Count Limits. Configuration should be changed by either disabling the Software End of Travel or setting the End of Travel values within the Count Limits.
18	Status Only	(Aux only) Scaled rotary EOT count modulus is not an integer	Axis	Check DSM axis configuration in HWCFG.
19	Status Only	Scaled rotary Hi/Lo limit count modulus is not an integer	Axis	Check DSM axis configuration in HWCFG.
1C	Status Only	Unsupported AQ command mode	Axis	The AQ commands that configure Torque Mode variables are not available in Analog Velocity Mode.

Error Code (hex)	Response	Description	Error Type	Possible Cause
1D	Normal Stop	Attempt to use CAM, CAM-Load, or CAM-Phase commands with Follower Mode	Axis	If using CAM, ensure that Follower Mode is not configured (Follower Mode cannot be used when using CAM). If using Follower Mode, ensure that CAM commands are not present in motion program (CAM cannot be used when Follower Mode is configured).
1E	Status Only	Immediate command Jog Velocity out of range, command ignored	Axis	The AQ immediate Jog Velocity command that was sent is too large. Re-enter the command using a smaller value
1F	Status Only	Immediate command Jog Acceleration out of range, command ignored	Axis	The AQ immediate Jog Acceleration command that was sent is too large. Re-enter the command using a smaller value
Program Errors				
20	Status Only	Program Acceleration over-range, acceleration set to maximum value	Axis	The acceleration programmed in the motion program currently executing is too large. Maximum value (1,073,741,823 cts/sec/sec at 1:1 scaling) is being used in the motion program.
21	Status Only	Program Acceleration too small, defaulted to 32 cts/sec/sec	Axis	The acceleration programmed in the motion program currently executing is too small. Default value (32 cts/sec/sec) is being used in the motion program
22	Status Only	Scaled Velocity greater than 1 million cts/sec, 1 million cts/sec is used	Axis	Check scaling in configuration, velocity in program
23	Status Only	Program Velocity is zero, set to minimum value of 1 count/sec.	Axis	The program velocity in the motion program currently executing is zero. The minimum value (1 count/sec) is being used
24	Status Only	Motion Program Velocity > Configured Velocity Limit, limit value used	Axis	The programmed velocity in the currently executing program is greater than the Velocity Limit set in axis configuration.
25		Reserved – not used in DSM314	Axis	
26	Stop Normal	Jump Mask error	Axis	Contact Emerson
27	Stop Normal	Wait Mask error	Axis	Contact Emerson
28	Stop Normal	Parameter Position too large	Axis	The position contained in the parameter referenced by the current PMOVE or CMOVE was greater than the maximum position range (-536,870,912 to +536,870,911 at 1:1 scaling)
29	Status Only	Dwell time greater than 60 seconds, 5 seconds used	Axis	The executing motion program encountered a DWELL statement where the DWELL time is greater than 60 seconds. This

Error Code (hex)	Response	Description	Error Type	Possible Cause
				value is larger than allowed. The DWELL time used for the program is 5 seconds. The user should open the motion program and correct the DWELL time statement to be less than 60 seconds. If more DWELL time is needed, consider multiple DWELL statements
2A	Normal Stop	Cyclic CAM CTL Exit condition specified for Non-Cyclic CAM	Axis	CTL exit conditions are permitted for Cyclic CAMs only. The motion program contains a non-cyclic CAM instruction with a CTL exit condition.
2B	Normal Stop	CAM Phase out of range	Axis	The CAM PHASE value is outside the axis position range.
Position Increment Errors				
2C	Status Only	Position Increment Over-range error, increment ignored	Axis	Position Increment in AQ command must be in range -128 to 127 user units
2D	Normal Stop	CAM Master Axis Configuration Error – master profile does not match master axis configuration	Axis	<ol style="list-style-type: none"> 1) The User-Units:Counts ratio specified for the master axis in the Editor and Hardware Configuration are not compatible and/or 2) The High/Low Position Limit specified for the master axis in Hardware Configuration is not compatible with the profile. Refer to the section on CAM Types for a detailed description on setting up the High/Low Position Limits.
2E	Normal Stop	CAM Slave Axis Configuration Error – slave profile does not match slave axis configuration	Axis	<ol style="list-style-type: none"> 1) The User-Units : Counts ratio specified for the slave axis in the Editor and Hardware Configuration are not compatible and/or 2) The High/Low Position Limit specified for the slave axis in Hardware Configuration is not compatible with the profile. Refer to the section on CAM Types for a detailed description on setting up the High/Low Position Limits.
2F	Normal Stop	CAM Slave Axis SW EOT mode cannot be enabled for Cyclic Circular CAM	Axis	
Find Home Errors				
30	Status Only	Find Home while Drive Not Enabled error	Axis	The Find Home command was executed when the Drive Enable bit was not on. The user should enable the drive and re-execute the command.

Error Code (hex)	Response	Description	Error Type	Possible Cause
31	Status Only	Find Home while Program Selected error	Axis	The Find Home command was executed while a Motion Program was selected for execution. The motion program must be halted (Program Active I bit off) prior to executing the Find Home command.
32	Status Only	Find Home while Force Digital Servo Velocity or Force Analog Output	Axis	The Find Home command was executed while the user was sending the Force Digital Servo Velocity (34h) or Force Analog Output (24h) AQ command. The user needs to clear this command prior to executing the Find Home command
33	Status Only	Find Home while Jog error	Axis	The user executed the Find Home command while the servo was being jogged. The user must halt the Jog command prior to executing the Find Home command.
34	Status Only	(1) Find Home while Move at Velocity error, or (2) Find Home while another Find Home Cycle is still active	Axis	User executed the Find Home command (1) while executing a Move at Velocity (22h) AQ command or (2) while another Find Home cycle was in progress. For (1), halt the Move at Velocity operation (Moving I bit off) prior to executing the Find Home command. For (2), verify that axis is In Zone and not Moving before executing a Find Home command.
35	Status Only	Find Home While Follower Enabled	Axis	The user executed the Find Home command while the follower function was enabled. The user must disable the follower (Follower Enabled I bit off) prior to executing the Find Home command
36	Status Only	Find Home while Abort bit set error	Axis	The user executed the Find Home command while the Abort bit was set. The user must clear the Abort bit prior to executing the Find Home command
37	Status Only	Find Home on first PLC sweep error	Axis	The Find Home Q bit was set during the first PLC sweep. The PLC program must be corrected to prevent this command from being sent on the first PLC sweep.
Move at Velocity Errors				
38	Status Only	Move at Velocity on First PLC sweep error	Axis	The Move at Velocity command (22h) was sent during the first PLC sweep. The PLC program must be corrected to prevent this command from being sent on the first PLC sweep.

Error Code (hex)	Response	Description	Error Type	Possible Cause
39	Status Only	Move at Velocity while Drive Not Enabled error	Axis	The Move at Velocity command (22h) was sent when the Drive Enable bit was not on. The user should enable the drive and re-execute the command.
3A	Status Only	Move at Velocity while Program Selected error	Axis	The Move at Velocity command (22h) was sent while a Motion Program was selected for execution. The motion program must be halted (Program Active I bit off) prior to sending the Move at Velocity Command
3B	Status Only	Move at Velocity while Home Cycle active error	Axis	The Move at Velocity command (22h) was sent while the module was executing a Home Cycle. The user needs to either abort the Home Cycle or wait until the Home Cycle completes prior to sending the Move at Velocity command
3C	Status Only	Move at Velocity while Jog error	Axis	The Move at Velocity command (22h) was sent while the Jog Q bit was active. The user must halt the Jog command prior to sending the Move at Velocity command
3D	Status Only	Move at Velocity while Abort All Moves bit is set error	Axis	The Move at Velocity command (22h) was sent while the Abort All Moves Q bit was set. The user must clear the Abort bit prior to sending the Move at Velocity command
3E	Status Only	Move at Velocity Data greater than 8,388,607 user units/sec	Axis	The user sent a Move at Velocity command (22h) where the commanded Velocity was greater than 8,388,607 user units/sec. The user needs to make the commanded Velocity smaller prior to re-executing the command.
3F	Status Only	Move at Velocity Data greater than 1 million cts/sec error	Axis	The user executed a Move at Velocity command (22h) where the scaled commanded Velocity was greater than 1 million cts/sec. The user needs to make the commanded Velocity smaller prior to re-executing the command. Check scaling
Jog Errors				
40	Status Only	Jog while Find Home error	Axis	The user executed a Jog while the module was executing a Find Home function. Either abort the Find home function or wait until it completes prior to executing the Jog function
41	Status Only	Jog while Move at Velocity error	Axis	The user set a Jog Q bit while the module was executing a Move at Velocity (22h) command. The Move at Velocity action must be halted before executing a Jog.

Error Code (hex)	Response	Description	Error Type	Possible Cause
42	Status Only	Jog while Force Digital Servo Velocity error	Axis	The user set a Jog Q bit while the module was executing a Force Digital Velocity (34h) or Force Analog Output (24h) AQ command. The AQ command must be removed before executing a Jog.
43	Status Only	Jog while Program Selected and not Feedholding error	Axis	If a program is running, the DSM can only Jog if the Feedhold Q bit is set.
Force Digital Servo Velocity Errors				
47	Status Only	Force Digital Servo Velocity or Force Analog Output while Jog error	Axis	The user executed a Force Digital Servo Velocity (34h) or Force Analog Output (24h) AQ command while the module is executing a Jog function. The Jog function must be halted prior to executing Force Digital Servo Velocity or Force Analog Output.
48	Status Only	Force Digital Servo Velocity or Force Analog Output while Move at Velocity error	Axis	The user executed a Force Digital Servo Velocity (34h) or Force Analog Output (24h) AQ command while the module is executing a Move at Velocity function. The Move at Velocity command must be halted prior to executing Force Digital Servo Velocity or Force Analog Output.
49	Status Only	Force Digital Servo Velocity or Force Analog Output while Program Selected error	Axis	The user executed a Force Digital Servo Velocity (34h) or Force Analog Output (24h) AQ command while the module is executing a motion program. The motion program must be halted (Program Active I bit off) prior to executing Force Digital Servo Velocity or Force Analog Output.
4A	Status Only	Force Digital Servo Velocity or Force Analog Output while Follower Enabled error	Axis	The user executed a Force Digital Servo Velocity (34h) or Force Analog Output (24h) AQ command while the follower was enabled. The follower must be disabled (Follower Enabled I bit off) prior to executing Force Digital Servo Velocity or Force Analog Output.
4B	Status Only	Force Analog Output while in Analog Torque mode	Axis	The user executed a Force D/A (24h) AQ command while the servo was configured for Analog Torque Mode. Force Analog Output. is not supported in Analog Torque Mode
Set Position Errors				
50	Status Only	Set Position while Program Selected error	Axis	The user executed a Set Position command while a Motion Program was selected to execute. The motion program must be

Error Code (hex)	Response	Description	Error Type	Possible Cause
				halted (Program Active I bit off) prior to executing the Set Position command.
51	Status Only	Set Position Data over-range error	Axis	The user executed a Set Position command with a value greater then the maximum position range (-536,870,912 to +536,870,911 at 1:1 scaling)
52	Status Only	Set Position while Moving or Set Position while not Moving, not In Zone and velocity > 100 cts/sec.	Axis	Set Position is not allowed if the Moving %I bit is on. If the Moving bit is off and the In Zone %I bit is also off, the Actual Velocity must be < 100 cts/second.
53	Status Only	Attempt to initialize position before digital encoder passes reference point.	Axis	The absolute digital encoder was not rotated past the zero reference point after the first application of power. The encoder must be rotated past the reference point (up to 1 revolution) before Set Position is allowed in absolute mode.
54	Status Only	Digital encoder position invalid, must use Find Home or Set Position.	Axis	<ol style="list-style-type: none"> 1. Absolute encoder position has not been initialized since first application of power 2. Configuration for Encoder mode has been changed from incremental to absolute. 3. Configuration for Axis Direction (normal or reverse) has been changed. 4. Encoder resolution (Set with Advanced configuration tab parameter) has been changed. 5. Encoder alarm has occurred
55	Status Only	Digital Encoder moved too far while power off	Axis	The digital absolute encoder was moved more than 16,383 revolutions while power was off.
End of Travel and Count Limit Errors				
56	Status Only	Commanded Position > Positive End of Travel or High Count Limit	Axis	The user executed a command that resulted in the Commanded Servo Position exceeding the Positive End of Travel or High Count Limit. Either fix the command to be less than these values or make the values higher in Hardware Configuration.
57	Status Only	Commanded Position < Negative End of Travel or Low Count Limit	Axis	The user executed a command that resulted in the Commanded Servo Position exceeding the Negative End of Travel or Low Count Limit. Either fix the command to be greater than these values or make the values more negative in Hardware Configuration

Error Code (hex)	Response	Description	Error Type	Possible Cause
58	Status Only	Absolute Encoder Position > High Software EOT Limit	Axis	This error is reported at power up or re-configuration if the absolute digital encoder has been moved beyond the High Software EOT Limit.
59	Status Only	Absolute Encoder Position < Low Software EOT Limit	Axis	This error is reported at power up or re-configuration if the absolute digital encoder has been moved beyond the Low Software EOT Limit.
Drive Disable Errors				
5B	Stop Normal	Drive Disabled while Moving	Axis	The Enable Drive Q bit was turned off while the servo was performing a Jog or Move at Velocity (Moving I bit set). The PLC program should be corrected to prevent this error. Consider using the Moving Bit in the logic that disables the drive.
5C	Stop Normal	Drive Disabled while Program Active	Axis	The Enable Drive Q bit was turned off while the servo was executing a motion program (Program Active I bit set). The PLC program should be corrected to prevent this error. Consider using the Program Active Bit in the logic that disables the drive.
Software Errors				
5F	Status Only	Software Error (Call Emerson Field Service)	Axis	Contact Emerson
60	Status Only	Absolute Encoder Rotary Position Computation error	Axis	Contact Emerson
Program and Subroutine Errors				
61	Stop Normal	Invalid subroutine number	Axis	The Motion Program called a subroutine that was not contained in the module program space. If the call instruction references a parameter that contains the subroutine number, confirm that the parameter data is correct.
62	Stop Normal	Call Error (subroutine already active on axis)	Axis	A Motion Subroutine called itself or called another subroutine that called the original subroutine.
63	Stop Normal	Subroutine End command found in Program	Axis	The Motion Program contains an invalid Subroutine end command within the main Motion Program (Program 1-10). Modify the Motion Program to remove this statement.
64	Stop Normal	Program End command found in Subroutine	Axis	The Motion Subroutine contains an invalid Program end command within the Motion Subroutine (Subroutine 1-40) . Modify the Subroutine to remove this statement

Error Code (hex)	Response	Description	Error Type	Possible Cause
65	Stop Normal	Sync subroutine encountered by non-sync program	Axis	The Motion Program encountered a Sync block in a program that was not multi-axis and setup for sync blocks.
66	Normal Stop	CAM Profile not found in CAM Download Block	Axis	The Cam profile was not linked to the CAM Download block in the CAM Editor and/or the CAM Download block name was not specified in Hardware Configuration.
67	Normal Stop	CAM Exit Distance out of range (Non-Cyclic CAMs)	Axis	The exit distance for a Non-Cyclic CAM was greater than the modulus for the CAM.
68	Status Only	(Correction Enabled) Velocity Command Limited due to Velocity Limit violation or Position Error Limit violation	Axis	
69	Normal Stop	(Correction Disabled) CAM velocity command above configured axis velocity limit	Axis	
6A	Normal Stop	CAM Position Error Limit Violation (with Correction Disabled)	Axis	
6B	Status Only	CAM commanded position at the exit different from CAM profile value due to position error or velocity limit	Axis	
6C	Normal Stop	CAM master value out of profile master range for Non-Cyclic profile (CAM and CAM-LOAD commands)	Axis	
6D	Normal Stop	Absolute mode CAM after incremental mode CAM in the sequence	Axis	
6F	Fast Stop	CAM trajectory calculation error	Axis	Contact Emerson
Program Execution Errors				
70	Status Only	Execute Program on first PLC sweep	Module	An Execute Program Q bit was set on the first PLC sweep. The PLC program must be corrected to prevent these Q bits from being set on the first sweep.
71	Status Only	Too many programs requested in same PLC sweep	Module	The number of Execute Program Q bits that transitioned ON in 1 sweep is greater than the configured number of axes. This error is also reported if the number of programs requested in a PLC sweep is less than or equal to the number of configured

Error Code (hex)	Response	Description	Error Type	Possible Cause
				axes but greater than the number of axes that are NOT already executing programs.
72	Status Only	Execute multi-axis program with multi-axis program already active	Module	An Execute Program Q bit was set for a multi-axis program when a multi-axis program was already executing. <i>Note:</i> Error 0075 will be reported instead of Error 0072 if the DSM is configured for only 1 axis. Error 0071 will be reported instead of Error 0072 if the DSM is configured for only 2 axes.
73	Status Only	Execute Program for axis configured as Limited Aux axis	Module	Motion Programs cannot be executed on an axis configured as Limited Aux. A Limited Aux axis performs position feedback processing only and does not have an internal motion path generator.
74		Reserved - not used in DSM314		
75	Status Only	Empty or Invalid Program requested	Module	An Execute Program Q bit was set for a program number not defined in the configured motion program block. This error is also reported if the DSM is configured for fewer axes than the axis number of the requested program. Check the configuration for a correct motion program block name. Make sure the requested program number is defined in the configured program block. Make sure the DSM is configured for a number of axes greater than or equal to the axis number of the requested program.
76	Status Only	AQ Move Command Position Out of Range	Axis	The user sent an AQ Move command (27h) with a position value greater than the maximum position range. (-536,870,912 to +536,870,911 at 1:1 scaling)
77	Status Only	AQ move command on first PLC sweep	Axis	An AQ Move command (27h) was commanded on the first PLC sweep. The PLC program must be corrected to prevent AQ Move commands from being sent on the first sweep.
Program Execution Conditions Errors				
80	Status Only	Execute Program while Home Cycle active	Axis	The PLC set an Execute Program Q bit while the module was executing a home cycle. The user either needs to wait until the home

Error Code (hex)	Response	Description	Error Type	Possible Cause
				cycle completes or abort the home cycle prior to executing the Motion Program.
81	Status Only	Execute Program while Jog	Axis	The PLC set an Execute Program Q bit while the module was performing a Jog operation. The Jog bits (from PLC or local logic) must be turned off prior to executing a Motion Program.
82	Status Only	Execute Program while Move at Velocity	Axis	The PLC set an Execute Program Q bit while the module was executing a Move at Velocity (22h) command. The Move at Velocity command must be halted prior to executing the Motion Program.
83	Status Only	Execute Program while Force Digital Servo Velocity or Force Analog Output	Axis	The PLC set an Execute Program Q bit while the module was executing a Force Digital Velocity (34h) or Force Analog Output (24h) command. The Force Digital Velocity or Force Analog Output command must be removed prior to executing the Motion Program.
84	Status Only	Execute Program while Program Active	Axis	The PLC set an Execute Program Q bit for an axis that was already running a motion program. The current program must be completed (Program Active I bit off) before executing another program on the same axis.
85	Status Only	Execute Program while Abort All Moves bit set	Axis	The PLC set an Execute Program Q bit while the module was executing an Abort All Moves. The Abort Q bit, the Moving I bit and the Program Active I bit must all be off before executing a program.
86	Status Only	Execute Program while Position Valid not set	Axis	The PLC set an Execute Program Q bit when the Position Valid I bit was off. Position Valid must be set by a Find Home cycle or Set Position command.
87	Status Only	Execute Program while Drive Enabled not set	Axis	The PLC set an Execute Program Q bit when the drive was not enabled (Drive Enabled I bit off). The Enable Drive Q bit must be set in order to enable the drive.
Program Synchronous Block Errors				
8C	Status Only	Sync Block Error during CMOVE	Axis	Program execution encountered a CMOVE identified by a sync block even though the other axis had not yet reached the sync block.

Error Code (hex)	Response	Description	Error Type	Possible Cause
8D	Status Only	Sync Block Error during Jump	Axis	Program execution jumped to a CMOVE or PMOVE identified by a sync block even though the other axis had not yet reached the sync block.
EEPROM Errors				
90	Status Only	Flash EEPROM memory programming failure	Module	Contact Emerson
Local Logic Errors				
91	Stop Fast	Local Logic System Halt	Module	The Local Logic program executed a statement that wrote to the System_Halt variable (e.g. System_Halt := 1;)
92	Stop Fast	Local Logic Time-Out Error	Module	The Local Logic Program exceeded the allocated execution time of 300 Microseconds. Decrease the Local Logic execution time by reducing the number of Local Logic statements or by modifying the program structure. Consult Appendix E for more information on local logic execution time.
93	Stop Fast	Local Logic Divide By Zero Error	Module	The Local Logic program performed a divide by zero or a Modulus by zero. Check the Local Logic program divide statements for error source. Parameter registers that contain zero values are possible sources for this error.
94	Stop Fast	Local Logic Divide/Modulus Overflow Error	Module	The Local Logic program performed a divide (or modulus) of a 64 bit integer and the result could not fit in a 32 bit integer. Check the Local Logic program divide statements for error source.
95	Status Only	Local Logic Add/Subtract Overflow Warning	Module	The Local Logic program added or subtracted numbers that caused an overflow condition to occur. The allowable range is -2,147,483,648 to +2,147,483,647. Change the local logic program to prevent overflow or set the Overflow variable to 0 at the end of each local logic cycle.
96	Status Only	Local Logic Absolute(ABS) Overflow warning	Module	The Local Logic program attempted to perform an ABS operation on -2,147,483,648 resulting in an overflow.

Error Code (hex)	Response	Description	Error Type	Possible Cause
97	Status Only	Local Logic Timeout Warning	Module	The Local Logic program execution time is close (greater than 275 Microseconds) to the maximum allowable execution time (300 Microseconds). Decrease the Local Logic execution time by reducing the number of Local Logic statements or by modifying the program structure. Consult Appendix E for more information on local logic execution time.
98	Status Only	Local Logic Execute on First Sweep Error	Module	The user attempted to execute Local Logic on the first PLC sweep (e.g. if the Local Logic enable Q bit is on when the PLC is switched from Stop to Run Mode).
99	Status Only	Local Logic Invalid Program Name or Not Enabled in Configuration	Module	The Local Logic Program Name specified in Hardware Configuration is not valid (or empty) or Local Logic is not enabled in Hardware Configuration.
9A	Stop Fast	Local Logic Stop Error (Per-Axis)	Axis	A Local Logic Stop Fast Error occurred (error codes 91-94).
Hardware Limit Switch Errors				
A0	Stop Fast	Limit Switch (+) error	Axis	The Positive Overtravel Limit Switch input is off. If Overtravel Limit switches are not used, set the Overtravel Limit Switch configuration to Disabled.
A1	Stop Fast	Limit Switch (-) error	Axis	The Negative Overtravel Limit Switch input is off. If Overtravel Limit switches are not used, set the Overtravel Limit Switch configuration to Disabled.
Hardware Errors				
A8	Stop Fast	Out of Sync error	Axis	Position Error has exceeded the Position error limit. Possible sources for this error are: <ol style="list-style-type: none"> 1. Position error limit being set too low for the application. 2. Feedback device being disconnected or slipping on controlled device 3. Incorrect Feedback device wiring. (i.e. positive rotation indicated as negative by feedback device)
A9	Status Only	Loss of Position Feedback	Axis	A Quadrature Error has been detected on an incremental quadrature encoder. Check the encoder wiring and ensure that the encoder is not operated beyond its rated speed.
B0-BE		See Table 81		Digital Servo Alarms, documented in Table 81

Error Code (hex)	Response	Description	Error Type	Possible Cause
Encoder Alarms				
C0	Stop Fast	Servo not ready	Axis	For analog servos, the Drive Ready faceplate input must be set on (0 volts) within 1 second after turning on the Enable Drive Q bit. If the Drive Ready input for analog servos is not used, the input configuration must be set to Disabled. For Digital servos, the amplifier E-Stop input may be activated or an amplifier fault may have occurred.
C1	Status Only	Serial Encoder Battery Low	Axis	The Serial Encoder battery voltage is low. The battery must be replaced or the encoder can be configured for Incremental (instead of Absolute) operation.
C2	Stop Normal	Serial Encoder Battery Failed	Axis	The Serial Encoder battery has failed. The battery must be replaced or the encoder can be configured for Incremental (instead of Absolute) operation.
C3	Stop Normal	Servo Motor Over Temperature	Axis	The Servo Motor or Control Firmware has reported an over temperature condition. The user needs to check the motion program to make sure that the duty cycle rating for the motor is not being exceeded. The user needs to also check the motor mounting to make sure the heat sink for the motor is adequate and ventilation for the motor is adequate
C4	N/A	Not used.	N/A	
C5	Stop Fast	Loss of Encoder	Axis	The module is not communicating with the encoder. Make sure the servo amplifier is on. Check encoder cabling to make sure cable is connected. Additionally, check grounding to ensure that grounding is correct.
C6	Stop Fast	Error in encoder pulse detection	Axis	The encoder pulse detection circuit has encountered an error. Make sure that the motor is properly grounded. If error persists consult factory.
C7	Stop Fast	Encoder counter error	Axis	The encoder counter circuit has encountered an error. Make sure that the motor is properly grounded. If error persists consult factory.
C8	Stop Fast	Encoder LED is disconnected	Axis	The encoder LED is disconnected. Consult factory.

Error Code (hex)	Response	Description	Error Type	Possible Cause
C9	Stop Fast	Encoder CRC checksum failure	Axis	The encoder communications circuit has detected a CRC error. Check the encoder cable grounding and the motor grounding for possible error sources. Check for other electrical noise sources in the area of the motor and encoder cabling. Isolate these sources from motor/encoder cabling if possible. If error persists consult factory
CA	Stop Fast	Unsupported encoder, linear or Type A	Axis	The motor encoder connected to the module is not supported. Motor is either not supported by the DSM module or has an incorrect encoder attached to the motor. Check motor label and verify motor is a supported model. If problem persists consult factory
CB	Stop Fast	Unsupported encoder, Type C	Axis	The motor encoder connected to the module is not supported. Motor is either not supported by the DSM module or has an incorrect encoder attached to the motor. Check motor label and verify motor is a supported model. If problem persists consult factory
CC	Normal Stop	Missed DZ pulse when DS transitioned from 1 to 0	Axis	Position data may be incorrect. Power-cycle motor and amplifier. If problem persists consult factory.
DSP Alarms				
D1	Stop Fast	Over current Detected	Axis	The Motor Control firmware detected an over current condition. Possible sources for this error include: - Incorrect Motor Type selected in Hardware configuration - Machine back driving motor excessively - Over Duty cycle conditions
D2	N/A	Not Used		
D3	Stop Fast	Over Acceleration Detected	Axis	The Motor Control firmware detected an acceleration value that exceeded allowed values. This error is not encountered under normal operating conditions. Possible error causes include encoder failure, encoder slippage, incorrect position reported from encoder. If error is not explained by physical hardware consult factory.

Error Code (hex)	Response	Description	Error Type	Possible Cause
D4	Stop Fast	Over Velocity Detected	Axis	The Motor Control firmware detected a velocity value that exceeded allowed values. This error is not encountered under normal operating conditions. Possible error causes include encoder failure, encoder slippage, incorrect position reported from encoder. If error is not explained by physical hardware consult factory.
D5	Status Only	Velocity Loop Gain for Kp Too Large	Axis	The Proportional Gain for the Velocity Loop has exceeded allowed values. Value limited to valid range. This error should not be encountered during normal operation. Possible error sources include incorrect motor type selected in hardware configuration, or Velocity Loop Gain values that are too large. If motor type is correct in hardware configuration, then reduce velocity loop gain. If problem persists, or velocity loop gain is too small for the application consult factory.
D6	Status Only	Integrator Gain Too Large	Axis	The Integral Gain for the Velocity Loop has exceeded allowed values. Value limited to valid range. This error should not be encountered during normal operation. Possible error sources include incorrect motor type selected in hardware configuration, or Velocity Loop Gain values that are too large. If motor type is correct in hardware configuration, then reduce velocity loop gain. If problem persists, or velocity loop gain is too small for the application consult factory.
D7	Status Only	Alpha Calculation Overflow G.S.	Axis	Internal Velocity Loop calculation has exceeded allowed values. Value limited to valid range. This error should not occur during normal operation. Reduce Velocity Loop Gain. If problem persists consult factory
D8	Status Only	Integrator Gain Calculation Overflow	Axis	Integral Gain for the Current Loop has exceeded allowed range. Calculation limited to valid range. This error should not occur during normal operation. If error encountered consult factory.

Error Code (hex)	Response	Description	Error Type	Possible Cause
D9	Status Only	Kp Calculation Overflow	Axis	Proportional Gain for the Current Loop has exceeded allowed range. Calculation limited to valid range. This error should not occur during normal operation. If error encountered consult factory.
DA	Stop Fast	FPGA Error Detected	Axis	An error was detected when the Field Programmable Gate Array was initialized. This error should not be encountered during normal operating conditions. If error encountered consult factory.
Special Purpose Errors				
E2	Stop Fast	DSP Interrupt failure	Module	Contact Emerson
Follower Ramp Errors				
E8	Status Only	Follower Registration Distance (from parameter register) is out of allowed range - follower stops using ramp acceleration.	Axis	When Follower Disable Action = Incremental Position, the incremental distance (registration distance) specified in the associated parameter register must be greater than the stopping distance. The stopping distance depends on the present slave axis velocity and follower ramp acceleration. Negative slave axis velocities require negative registration distances.
E9		Reserved - not used in DSM314		
EA	Status Only	Master velocity greater than 0.8 * velocity limit-no distance compensation	Axis	The master velocity when converted to slave axis units is greater than 0.8 * the configured velocity limit. The velocity limit must be increased or the master velocity must be decreased.
EB	Stop Fast	Error in calculation during follower ramp-up	Axis	Contact Emerson
EC	Status Only	Follower makeup time is not long enough	Axis	The configured Ramp Makeup Time is too small so that actual makeup time is longer. The makeup time of follower ramp acceleration should be increased.
ED	Status Only	Velocity limit violation during follower ramp	Axis	Follower ramp makeup requires a velocity greater than 0.8 * the configured axis velocity limit, so that actual makeup time is longer than the configured value. Increase the velocity limit, makeup time or ramp acceleration.

Error Code (hex)	Response	Description	Error Type	Possible Cause
EE	Status Only	Time limit violation during acceleration sector of the follower distance correction	Axis	Ramp makeup required an acceleration time > 64000 position loop sample times. The follower ramp acceleration must be increased.
Position Loop Errors				
F0	Status Only	Attempt to enable follower with drive disabled	Axis	Follower has been enabled on an axis that the drive is not enabled. Drive must be enabled prior to enabling follower.
F1	Status Only	Follower Position Error Limit Encountered	Axis	The position error has reached the position error limit and the follower loop is no longer position-locked to the master axis. The position error limit must be increased or velocity feedforward must be used.
F2	Status Only	Velocity Limit Condition Encountered	Axis	The sum of all command inputs (internal cmds + follower master + local logic) to the position loop has exceeded the configured velocity limit. The axis is no longer position-locked to the commands. The command velocities must be decreased or the velocity limit must be increased.
F3	Status Only	Follower Ratio B value = 0	Axis	Follower Ratio B values < 0 are not allowed.
F4	Status Only	Follower Ratio B value < 0	Axis	A Follower Ratio B value of 0 is not allowed.
F5	Status Only	Follower ratio A:B > 32:1 or < 1:10000	Axis	The Follower Ratio A / Ratio B values must represent an A/B ratio in the range 32:1 to 1:10000.
Internal Errors				
FB	Status Only	Control Loop execution time > 500 microseconds	Axis	Contact Emerson
FC	Status Only	Control Loop execution > 400 microseconds, more than 5 times in a row	Axis	Contact Emerson
FD	Stop Fast	System software error	Axis	Contact Emerson
FE	Stop Fast	Unrecognized encoder, not supported	Axis	Error can indicate defective encoder cable – check cable. If cable checks out correctly, contact Emerson

A-1.5 System Error Codes

If the DSM encounters errors with the configuration, a motion program, or local logic block, it will place a System Error code in the Module Status Code register (the first AI word). When a System Error occurs, the DSM will not update any %I bits or %AI data and will not respond to any %Q bit or %AQ commands.

So the %Q Clear Error bit has no effect on a System Error. A System Error can only be cleared by sending a new configuration to the DSM

The following system error codes indicate that the user has entered an invalid DSM configuration in the configuration/programming software. If one of these errors occurs, you must change the configuration and store the new configuration to the PLC. Any other errors of the format **Dxxx**, **Exxx** or **Fxxx** not documented in the table below are unexpected and should be reported to Emerson.

Table 80: System Error Codes

Error Code (hex) (x = axis number)	System Error Type	Description
D008	Module	Axis 4 not disabled when Axis 1,2 = Digital Servo
Dx65	Axis	Feedback Source is invalid or not supported
Dx68	Axis	Follower Disable action is not supported
Dx69	Axis	Follower Ramp Makeup Mode is not supported
Dx71	Axis	Invalid digital servo motor type
Dx81	Axis	Analog Servo Cmd mode (Torque mode) not supported.
		<i>Note: DSM314 version 3.0 or later supports Torque Mode.</i>

A-2 DSM Digital Servo Alarms (B0–BE)

α and β digital servo systems have built in detection and safety shut down circuitry for many potentially dangerous conditions. The table below reflects that three different models of servo amplifiers may be used with the DSM, the β Series, the α Series SVU and the α Series SVM. The following table indicates alarms supported by a particular servo amplifier and the corresponding DSM error code. Table entries that are blank in the amplifier columns indicate amplifier alarms not supported by the particular amplifier series. To clear a servo alarm, amplifier power cycle reset is required. Additionally, a “Clear Error” %Q discrete command is required to clear the DSM Error Code. Amplifier alarms not cleared by power cycle of the amplifier will continue to be reported to the DSM module. A brief diagnostics section for servo alarms appears at the end of the error alarm tables.

Table 81: DSM Digital Servo Alarms

Error Number (Hex)	Servo Alarm Name	Description	Amplifier Alarm Display		
			SVM 7 SEG	SVU 7 SEG	β ALM LED
B0	HV	Over- Voltage DC LINK	07†	1	ON
B1	LV	Low Voltage Control Power	06†	2	
B2	DBRLY	Dynamic Brake Circuit Failure † SVM PSM DC LINK Low Charge	05†	7	
B3	LVDC	Low Voltage DC LINK	04†	3	ON
B4	OH	Amplifier Over Heat	03†	ON	
B5	FAL	Cooling Fan Failure	02†	ON	
B6	† SVM PSM IPM Alarm or Over Current		01†		
B7	DCSW	Regenerative Circuit – Failure Alarm	08†	4	ON
	DCOH	Regenerative Circuit – Discharge Alarm		5	
B9	LV5V	SVM Servo Module +5 V Low	2		
BA	IPML	IPM Over Current, High Temp or Low Volt (L axis, M axis, N axis, L & M axes, M & N axes, N & L axes or L & M & N axes)	8.	8.	
	IPMM		9.	9.	
	IPMN		A.	A.	
	IPMLM		b.	b.	
	IPMMN		C.	C.	
	IPMNL		d.	d.	
IPMLMN	E.	E.			
BB	LVDC	SVM Servo Module Low DC LINK	5		
BD	FAL	SVM Servo Module Fan Failure	1		
BE	HCL	Abnormally High Motor Current (L axis, M axis, N axis, L & M axes, M & N axes, N & L axes or L & M & N axes)	8	8	ON
	HCM		9	9	
	HCN		A	A	
	HCLM		b	b	
	HCMN		C	C	
	HCNL		d	d	
	HCLMN		E	E	

† The two segment display on the SVM power supply module (PSM) indicates power supply alarms.

A-3 Troubleshooting Digital Servo Alarms

The guidelines below are intended to assist in isolating problems associated with various servo alarms. If the items below do not fit the case or resolve the alarm, replace the servo amplifier, or Contact Emerson Technical support. The appropriate amplifier and motor, Maintenance Manual or Description Manual, will include more detailed trouble shooting procedures.

HV (High-voltage) Alarm: This alarm occurs if the high voltage DC level (DC LINK) is abnormally high.

1. The AC voltage supplied to the amplifier may be higher than the rated input voltage. The β Series amplifier, three-phase supply voltage should be between 200 VAC to 240 VAC.
2. The external regeneration resistor may be wired incorrectly. Carefully check the connections of the regeneration resistor to the amplifier. Check that the resistance of the regeneration resistor is within 20% of the rated value. Replace the regeneration unit if the resistance is out of tolerance.
3. The regeneration resistor may not be capable of dissipating excess generated voltage. Review the calculations for selecting the regenerative discharge unit and replace with a resistor of higher wattage rating as needed. Reducing acceleration values and position loop gains (larger value Position Loop Time Constant) will additionally reduce regenerated voltage levels.

LVDC (Low Voltage DC Link): This alarm occurs if the high voltage DC level (DC LINK) voltage is abnormally low.

The AC voltage supplied to the amplifier may be missing or lower in value than the rated input voltage. The β Series amplifier, three-phase supply voltage should be between 200 VAC to 240 VAC. Verify that the proper level of AC voltage is supplied to the line input (L1, L2 and L3) connections of the amplifier.

DCOH or DCSW (Regeneration Alarm): The DCOH alarm occurs if the temperature of the regeneration resistors is too high. The DCSW alarm indicates problems in the switching portion of the regeneration circuitry.

1. If the external regeneration resistor is not used check that the temperature sensor input to the amplifier is shorted or jumped. The β Series amplifier jumper T604 should be installed on connector CX11-6.
2. The external regeneration resistor may be wired incorrectly. Carefully check the connections of the regeneration resistor to the amplifier. Check that the resistance of the regeneration resistor temperature sensor is near zero ohms at room temperature. Replace the regeneration resistor if the temperature sensor indicates an open condition.
3. The regeneration resistor may not be capable of dissipating excess generated voltage. Review the calculations for selecting the regenerative discharge unit and replace with a resistor of higher wattage rating as needed. Reducing acceleration

values and position loop gains (larger value Position Loop Time Constant) will additionally reduce regenerated voltage levels.

OH (Over-heat Alarm): The temperature of the amplifier heat sink is too high or motor temperature is excessive.

1. Ambient temperature may be too high, consider a cooling fan for the servomotor. Emerson supplies fan kits for most motors.
2. The motor may be operating in violation of duty cycle restrictions. Calculate the amount of cooling time needed based on the duty cycle curves published for the particular motor.
3. The motor may be over loaded. Check for excessive friction or binding in the machine.
4. For all the above problems, allow ten minutes cooling of the amplifier with minimum or no motor loading then cycle amplifier power to reset.

FAL (Fan Alarm): The cooling fan has failed.

1. Check the fan for obstructions or debris. With amplifier power removed attempt to manually rotate the fan.
2. For SVM type amplifier systems the power supply module (PSM) and the servo amplifier module each include a cooling fan. The alarm code will indicate which unit failed.
3. Some amplifiers have field replaceable fan units. If a replacement fan unit is not available, replace the amplifier.

HC, HCL, etc. (High Current Alarm): Motor current is excessive. For α Series amplifiers the suffix (L, M, N, etc.) indicates which axis is in alarm

1. Motor power wiring (U, V and W) may be shorted to ground or connected with improper phase connections. Check the wiring and connections. Check the servomotor for shorts to motor frame. Replace the motor if shorted.
2. Improper motor type code may be configured or excessive values for tuning parameters. Confirm that the proper motor is configured and lower gain values.
3. The amplifier maintenance manual will describe the procedure for monitoring motor current signals (IR and IS). If the waveforms are abnormal replace the amplifier. If excessive noise is observed check grounds and especially the cable shield grounds for the command cable (K1) to the amplifier.
4. The motor may be operating in violation of duty cycle restrictions. Calculate the amount of cooling time needed based on the duty cycle curves published for the particular motor.
5. The motor may be over loaded. Check for excessive friction or binding in the machine.
6. For all the above problems, allow ten minutes cooling of the amplifier with minimum or no motor loading then cycle amplifier power to reset.

LV (Low Voltage Control Power Alarm): The control voltage used to operate the low-voltage circuitry in the amplifier is too low.

1. α Series SVU type amplifiers will be shipped with default jumpers to use a single phase of the 220 VAC power to the amplifier. Optionally the user may remove the jumpers and connect 220 VAC control power separately. Check that a minimum 200VAC is available on terminals L1C and L2C for default installation or on connector CX3 (Y Key) for separate control power.
2. Check the amplifier fuse. If the fuse is open replace with a new fuse after checking control power voltage. If the second fuse blows open, replace the amplifier.

DBRLY (Dynamic Brake Relay Failure): This alarm indicates that the contacts of the braking relay are welded together. Replace amplifier immediately.

IPML, IPMM, etc. (IPM Alarm): The Intelligent Power Module (IPM) is the high current switching device in the amplifier. The IPM can detect over-current, over-heat or low-voltage conditions in the power switching circuitry. The suffix (L, M, N, etc.) indicates which axis is in alarm.

1. Motor power wiring (U, V and W) may be shorted to ground or connected with improper phase connections. Check the wiring and connections. Check the servomotor for shorts to motor frame. Replace the motor if shorted.
2. Improper motor type code may be configured or excessive values for tuning parameters. Confirm that the proper motor is configured and lower gain values.
3. The amplifier maintenance manual will describe the procedure for monitoring motor current signals (IR and IS). If the waveforms are abnormal replace the amplifier. If excessive noise is observed check grounds and especially the cable shield grounds for the command cable (K1) to the amplifier.
4. The motor may be operating in violation of duty cycle restrictions. Calculate the amount of cooling time needed based on the duty cycle curves published for the particular motor.
5. The motor may be over loaded. Check for excessive friction or binding in the machine.
6. For all the above problems, allow ten minutes cooling of the amplifier with minimum or no motor loading then cycle amplifier power to reset.

A-4 LED Indicators

There are seven LEDs on the DSM314 module that provide status indications. These LEDs are described below.

STAT Normally ON. FLASHES to provide an indication of operational errors. Flashes slow (four times/second) for Status-Only errors. Flashes fast (eight times/second) for errors that cause the servo to stop.

ON: When the LED is steady ON, the DSM314 is functioning properly. Normally, this LED should always be ON.

OFF: When the LED is OFF, the DSM314 is not functioning. This is the result of a hardware or software malfunction that will not allow the module to power up.

Flashing: When the LED is FLASHING, an error condition is being signaled.

Constant Rate, CFG LED ON:

The LED flashes slow (four times / second) for Status Only errors and fast (eight times / second) for errors that cause the servo to stop. The Module Error Present %I status bit will be ON. An error code (hex format) will be placed in the Module Status Code %AI word or one of the Axis Error Code %AI words.

Constant Rate, CFG LED Flashing:

If the STAT and CFG LEDs both flash together at a constant rate, the DSM314 module is in boot mode waiting for a new firmware download. If the STAT and CFG LEDs both flash alternately at a constant rate, the DSM314 firmware has detected a software watchdog timeout due to a hardware or software malfunction.

Irregular Rate, CFG LED OFF:

If this occurs immediately at power-up, then hardware or software malfunction has been detected. The module will blink the STAT LED to display two error numbers separated by a brief delay. The numbers are determined by counting the blinks in both sequences. Record the numbers and contact Emerson for information on correcting the problem.

OK The OK LED indicates the current status of the DSM314 module.

ON: When the LED is steady ON, the DSM314 is functioning properly. Normally, this LED should always be ON.

OFF: When the LED is OFF, the DSM314 is not functioning. This is the result of a hardware or software malfunction that will not allow the module to power up.

CFG	This LED is ON when a module configuration has been received from the PLC.
EN1	When this LED is ON, the Axis 1 Drive Enable relay output is active
EN2	When this LED is ON, the Axis 2 Drive Enable relay output is active.
EN3	When this LED is ON, the Axis 3 Drive Enable relay output is active.
EN4	When this LED is ON, the Axis 4 Drive Enable relay output is active.

Appendix B: DSM314 Communications Request Instructions

This appendix describes two types of Communications Request (abbreviated COMM REQ in this appendix) ladder instructions used with the DSM314:

- **Parameter Load Type:** Used to load DSM Parameter Memory. An advantage of the COMM REQ instruction is that each one can load up to 16 parameters, and multiple COMM REQ instructions may be used in one host controller sweep. By comparison, each Load Parameter Immediate Command can load only one parameter per sweep, with from one to four Load Parameter Immediate commands allowed per sweep, depending upon the number of %AQ words configured (which, in turn, depends upon the number of axes configured - see Table 47). Therefore, the COMM REQ is most useful for loading several or many parameters, and the Load Parameter Immediate Command is most useful if you only need to load a few (one to four).
- **User Data Table (UDT) Type:** Used to access the DSM314's Local Logic User Data Table. The User Data Table is an 8192-byte memory area that Local Logic programs can use for data storage and retrieval. The UDT COMM REQ can copy data either from host controller word memory to the UDT or from the UDT to host controller word memory.

In general, a COMM REQ is used in a host controller ladder program to communicate with a variety of intelligent modules. This appendix first discusses the COMM REQ instruction in general in Sections 1 and 2, then in Sections 3 – 5, discusses how it specifically applies to the DSM314 module. This appendix is divided into the following sections:

- Section 1: Communications Request Overview
- Section 2: The COMM REQ Ladder Instruction
- Section 3: The User Data Table (UDT) COMM REQ
- Section 4: The Parameter Load COMM REQ
- Section 5: COMM REQ Ladder Logic Example (uses Parameter Load COMM REQ)

B-1 Communications Request Overview

The Communications Request uses the parameters of the COMM REQ Ladder Instruction and an associated Command Block to define the characteristics of the request. An associated Status Word reports the results of each request.

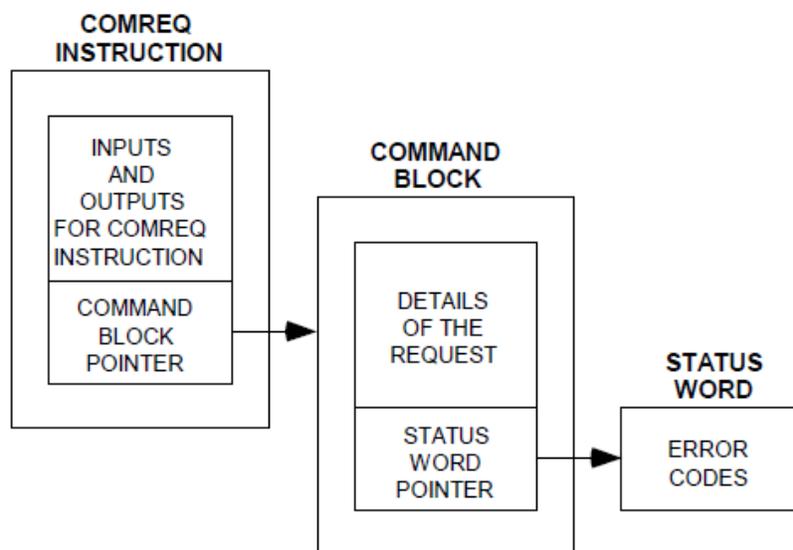
B-1.1 Structure of the Communications Request

The Communications Request is made up of three main parts:

- The COMM REQ Ladder Instruction
- The Command Block, which is a block of host controller memory (usually %R memory) that contains instructions and data for the COMM REQ.
- The Status Word, which is one word of memory that status/error codes are written to.

The figure below illustrates the relationship of these parts:

Figure 179: Structure of the COMM REQ



The COMM REQ Ladder Instruction: The COMM REQ Ladder Instruction is the main structure used to enter specific information about a communications request. This information includes the rack and slot location of the DSM module associated with the request, and a parameter that points to the starting address of the Command Block. Note that in programming this instruction, the command block data should be initialized in the ladder program before the rung containing the COMM REQ instruction is executed.

The Command Block: The Command Block consists of several words of host controller memory that contain additional information about the communications request. This information includes timing parameters, a pointer to the Status Word, a Data Block, memory types and sizes, and a specific command code. The Data Block specifies the direction of the data transfer (via the Command Code) and location and type of data to be transferred.

The Status Word: The Status Word is a single location in host controller data memory where the CPU reports the result of the communications request. The Status Word address is specified in the Command Block by the user. The following table lists the status codes reported in the Status Word:

Table 82: DSM COMM REQ Status Word Codes

Code Name	Code #	Description	Possible Corrective Action
IOB_SUCCESS	1	All communications proceeded normally.	None required.
IOB_PARITY_ERR	-1	A parity error occurred while communicating with an expansion rack.	Retry. Check hardware – expansion cables, DSM module, etc.
IOB_NOT_COMPL	-2	After the communication was over, the module did not indicate that it was complete.	Retry. Verify the COMM REQ parameters.
IOB_MOD_ABORT	-3	The module aborted the communication.	Retry. Verify the COMM REQ parameters.
IOB_MOD_SYNTAX	-4	The module indicated that the data sent was not in the correct sequence.	Verify the COMM REQ parameters.
IOB_NOT_RDY	-5	The RDY bit in the module’s status was not active.	Retry. Check DSM module.
IOB_TIMEOUT	-6	The maximum response time elapsed without receiving a response from the module.	Check DSM module. Verify the COMM REQ parameters.
IOB_BAD_PARAM	-7	One of the parameters passed was invalid.	Verify the COMM REQ parameters.
IOB_BAD_CSUM	-8	The checksum received from the DMA protocol module did not match the data received.	Retry. Check installation for proper grounding, shielding, noise suppression, etc.
IOB_OUT_LEN_CHGD	-9	The output length for the module was changed, so normal processing of the reply record should not be performed.	Verify the COMM REQ parameters.

Corrective Action

The type of corrective action to take depends upon the application. If an error occurs during the startup or debugging stage of ladder development, the advice to “Verify the COMM REQ parameters” is appropriate. The same is true if an error occurs right after a program is modified. But, if an error occurs in a proven application that has been running successfully, the problem is more likely to be hardware related. The host controller fault tables should be checked for possible additional information when troubleshooting Status Word errors.

B-1.2 Monitoring the Status Word

Error Detection and Handling

Figure 180



As shown in the table above, a value of 1 is returned to the Status Word if communications proceed normally, but if any error condition is detected, a negative value is returned. If you require error detection in your ladder program, you can use a Less Than (LT) compare instruction to determine if the value in the Status Word is negative (less than zero). An example of this is shown in the following figure. If an error occurs, the Less Than's output (Q) will go high. A coil driven by the output can be used to enable fault handling or error reporting logic.

The FT output of the COMM REQ, described later in this appendix, goes high for certain faults and can be used for fault detection also. Additionally, the Status Word can be monitored by error message logic for display on an Operator Interface device, in which case, Status Word codes would correspond to appropriate error messages that would display on the operator screen. For example, if a -1 was detected in the Status Word, a message could be displayed that says something like "Error communicating with the DSM module in an expansion rack."

To dynamically check the Status Word, write a non-significant positive number (0 or 99 are typically used) into the Status Word each time before its associated COMM REQ is executed. Then, if the instruction executes successfully, the CPU will write the number 1 there. This method lets you know that if the number 1 is present, the last COMM REQ definitely executed successfully, and that the 1 was not just "left over" from a previous execution. In the example presented at the end of this appendix, the number 99 is moved into the Status Word (%R0195) in a rung prior to the rung that contains the COMM REQ instruction.

When multiple DSM COMM REQs are used, it is recommended that each be verified for successful communications before the next is enabled. Monitoring the Status Word is one way to accomplish this.

Verifying that the DSM Received Correct Data

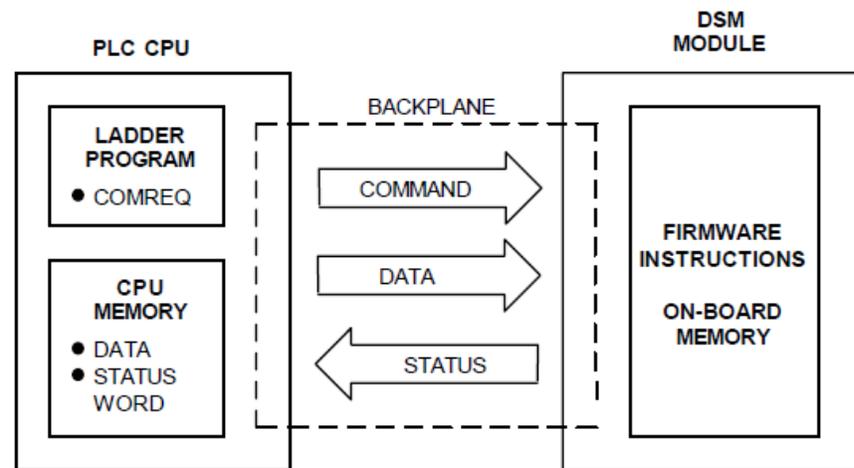
For critical applications, it may be advisable to verify that certain parameter values were communicated correctly to the DSM module before operation is allowed to continue. To accomplish this, first program the Select Return Data %AQ Immediate Command to specify a DSM parameter number to be read into the applicable User Selected Data %AI double word (there is one User Selected Data %AI double word for each axis). **Note that at least three host controller sweeps or 20 milliseconds, whichever represents more time, must elapse before the new User Selected Data is available in the host controller.** This requires programming some time delay logic to ensure that this requirement is met. Then, program a Double Integer type Equal instruction to compare the value returned in the User Selected Data double word with the value sent. Section 5 of this appendix shows an example of this. Also, refer to Chapter 5 for more information on the User Selected Data word and the Select Return Data command.

B-1.3

Operation of the Communications Request

The figure below illustrates the flow of information from the host controller CPU to the DSM module:

Figure 181: Operation of the DSM Communications Request



A Communications Request is initiated when a COMM REQ ladder instruction is activated during the host controller scan. At this time, details of the Communications Request, consisting of command and data, are sent from the host controller CPU to the DSM module.

- In the case of a Parameter Load COMM REQ, the command data specifies that data is to be read from host controller memory and copied into specific DSM parameter memory locations.
- In the case of a UDT COMM REQ, the command data either specifies that data is to be read from host controller memory and copied into a specific UDT memory Segment or read from a specific UDT memory Segment and copied into host controller memory.

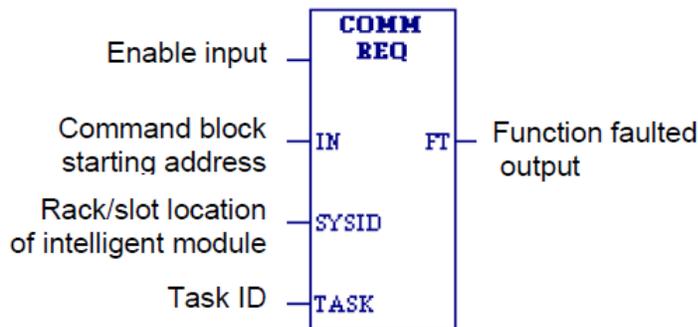
The order in which these instructions are sent is critical, so the Command Block for each type of COMM REQ should be programmed exactly as instructed later in this appendix. In the figure above, the DSM module is shown in the CPU rack and communications occur over the host controller backplane. If the DSM module is located in an expansion or remote rack, the commands and data are sent over the CPU rack’s backplane, through the expansion or remote cable to the rack containing the DSM module, and across that rack’s backplane to the DSM.

At the conclusion of every request, the host controller CPU reports the status of the request to the Status Word, which is a location in host controller memory that is designated by the Status Word Pointer in the Command Block.

B-2 The COMM REQ Ladder Instruction

This section discusses the COMM REQ instruction in general. More information is provided in the PACSystems CPU Reference Manual, GFK-2222 and the Series 90-30/20/Micro PLC CPU Instruction Set Reference Manual, GFK-0467. The Communications Request begins when the COMM REQ Ladder Instruction is activated. The COMM REQ ladder instruction has four inputs and one output:

Figure 182: COMM REQ Ladder Instruction



Enable Input: Must be Logic 1 to enable the COMM REQ Instruction. It is recommended that the enabling logic be a contact from a transition (“one-shot”) coil.

IN: The memory location of the first word of the Command Block. It can be any valid address in word-type memory (%R, %AI, or %AQ).

SYSID: A hexadecimal value that gives the rack and slot location of the module that the COMM REQ is targeting. The high byte (first two digits of the hex number) contains the rack number, and the low byte contains the slot number. The table below shows some examples of this:

SYSID Examples

Rack	Slot	Hex Word Value
0	4	0004h
3	4	0304h
2	9	0209h

TASK: The number 0 should always be entered here for a DSM module.

FT Output: The function’s FT (fault) output can provide an output to optional logic that can verify successful completion of the Communications Request. The FT output can have these states:

Table 83: COMM REQ Instruction FT Output Truth Table

FT Output		
Enable Input Status	Does an Error Exist?	FT output
Active	No	Low
Active	Yes	High
Not active	No execution	Low

- The FT output will be set High if:
 - The specified target address is not present (for example, specifying Rack 1 when the system only uses Rack 0).
 - The specified task number is not valid for the device (the TASK number should always be 0 for the DSM).
 - Data length is set to 0.

DSM COMM REQ Programming Requirements and Recommendations

- It is recommended that DSM COMM REQ instructions be enabled with a contact from a transition coil.
- If using more than one DSM COMM REQ in a ladder program, verify that a previous COMM REQ executed successfully before executing another one. This can be done by checking the Status Word and the FT (Fault) output, explained earlier in this appendix under the heading “Monitoring the Status Word.”
- As seen in the table above, the FT output will be held False if the Enable Input is not active. This means that if the COMM REQ is enabled by a transitional (one-shot) contact and a fault occurs, the FT output will only be High for one host controller scan. Therefore, to “capture” the fault, you can program the fault output as a Set coil, which would not be automatically reset at the end of a scan. Additional logic would then be needed to reset the fault output coil after the fault is acknowledged.
- Programming a device, such as a Set Coil, on the FT output of the COMM REQ is optional.
- It is necessary to initialize the data in the Command Block prior to executing the COMM REQ instruction. Since the normal host controller sweep order is from top to bottom, initializing the Command Block in an earlier rung (or rungs) than the rung that contains the COMM REQ will facilitate this requirement. See the example at the end of this appendix.
- Recommendation: If you use MOVE instructions to load values into Command Block registers, use a Word-type MOVE to load a hexadecimal number, and an Integer-type MOVE to load a decimal number. You will see this applied in the example at the end of this appendix for a Parameter Load COMM REQ, where the E501h code is

loaded via a Word-type MOVE instruction, and the remaining decimal values are loaded via Integer-type MOVEs.

B-3 The User Data Table (UDT) COMM REQ

The DSM314 has an 8192-byte memory area called the User Data Table (UDT) that is designated for use with Local Logic (LL) programs. LL Programs can access all or part of this memory to store and retrieve data. The UDT is useful for storing and retrieving large amounts of data such as large batches of setup data.

The host controller CPU can write to or read from the UDT via a User Data Table Communications Request (UDT COMM REQ) instruction in the host controller ladder program. A single UDT COMM REQ reads or writes 2048 bytes of memory at a time. Therefore, the UDT is logically divided into four 2048-byte segments, called Segments 1-4, that can be accessed individually by a UDT COMM REQ. There is a unique Read and a unique Write command for each of the four Segments, for a total of 8 possible UDT COMM REQ commands.

B-3.1 User Data Table COMM REQ Features and Usage Information

- Reads or Writes 2K (2048) bytes at a time to the Local Logic User Data Table. No other value is permitted.
- Only works with the DSM314 module (will not work with the DSM302)
- Cannot be used to download parameter data to the DSM314
- This instruction adds about 15 ms to host controller scan (sweep) time for one scan if the host controller's Communication Window Sweep Control parameter is set to COMPLETE (Run to Completion). If the Communication Window Sweep Control parameter is set to LIMITED, the COMM REQ will be executed over several scans, with a smaller impact on scan time. However, the COMM REQ probably will not be executed repeatedly – it will only be executed when there is a need to change data. Therefore, if it was sent on the First Scan, or during a job setup, it would not have an impact while the application is running.
- To avoid memory access conflicts, it is recommended that a Periodic Subroutine not be used during the time this COMM REQ is active.
- This COMM REQ does not support discrete memory for its host controller Data Type.

B-3.2 The UDT COMM REQ Command Block

Table 84: User Data Table Command Block

User Data Table COMM REQ Command Block for DSM314 Module		
Description	Address Offset	Word No. and Value
Data Block Header Length	Address + 0	Word 1, always set to 4
WAIT/NOWAIT Flag	Address + 1	Word 2, always set to 0
Status Word Memory Type (see Status Word Memory Type Codes table below)	Address + 2	Word 3, chosen by user (see Memory Type Codes table, below)
Status Pointer Offset	Address + 3	Word 4, chosen by user
Idle Timeout Value	Address + 4	Word 5, always set to 0
Maximum Communication Time	Address + 5	Word 6, always set to 0
Command Code	Address + 6	Word 7, see Command Code Table
Parameter Data Size, in bytes	Address + 7	Word 8, always 2048.
Memory Type for Host Controller Data	Address + 8	Word 9, chosen by user (see Memory Type Codes table, below)
Start of Host Controller Data (Data Offset)	Address + 9	Word 10, chosen by user

Data Block Length (Word 1): The length of the Data Block header portion of the Command Block. It should be set to 4. The Data Block header is stored in Words 7 through 10 of the Command Block

WAIT/NOWAIT Flag (Word 2): This must always be set to logic zero for the DSM.

Status Word Memory Type (Word 3): This word specifies the memory type that will be used for the Status Word. Each memory type has its own specific code number, shown in the Memory Type Codes table below. So, for example, if you want to use %R memory for the Status Word, you would put either the decimal code number 8 or the hexadecimal code number 08h in this word.

Note that if you select a discrete memory type (%I or %Q), 16 consecutive bits will be assigned to the Status Word, beginning at the address specified in the Status Word Pointer Offset word, described below.

Table 85: Status Word Memory Type Codes

Memory Type Abbreviation	Memory Type	Code Number to Enter	
		Decimal	Hexadecimal
%I	Discrete input table	70	46h
%Q	Discrete output table	72	48h
%R	Register memory	8	08h
%AI	Analog input table	10	0Ah
%AQ	Analog output table	12	0Ch

Status Word Pointer Offset (Word 4): This word contains the offset within the memory type selected. Note: The Status Word Pointer Offset is a zero-based number. In practical terms, this means that you should subtract one from the address number that you wish to specify. For example, to select %R0001, enter a zero ($1 - 1 = 0$). Or, if you want to specify %R0100, enter a 99 ($100 - 1 = 99$). Note that the memory type, %R in this example, is specified by the previous word (see the “Status Word Pointer Memory Type” explanation above).

Idle Timeout Value (Word 5): Since the DSM always uses the NOWAIT mode (WAIT/NOWAIT flag always set to zero), this Idle Timeout Value parameter is not used for the DSM. Set it to zero.

Maximum Communication Time (Word 6): Since the DSM always uses the NOWAIT mode (WAIT/NOWAIT flag always set to zero), this Maximum Communication Time parameter is not used for the DSM. Set it to zero.

Command Code (Word 7): Use one of the eight Command Codes from the table below. The Command Codes are given as hexadecimal numbers.

Table 86: UDT COMM REQ Command Codes

User Data Table (UDT) COMM REQ Commands	
Command Code	Command Description
D001h	Write to UDT Segment 1
D101h	Write to UDT Segment 2
D201h	Write to UDT Segment 3
D301h	Write to UDT Segment 4
D804h	Read from UDT Segment 1
D904h	Read from UDT Segment 2
DA04h	Read from UDT Segment 3
DB04h	Read from UDT Segment 4

UDT Segment Data Size (Word 8): Specifies the memory size, in bytes, of the UTP Segment to be accessed. This value should always be 2048 bytes (800h for hexadecimal).

Data Memory Type (Word 9): This word specifies the memory type that will be used for host controller data. Each memory type has a unique code number, shown in the Memory Type Codes table below. So, for example, to specify %R memory, you would put either the decimal code number 8 or the hexadecimal code number 08h in this word.

Note: *The UDT COMM REQ does not support discrete memory (%I or %Q) for the Data Memory Type.*

Table 87: Data Memory Type Codes for UDT COMM REQ

Memory Type Abbreviation	Memory Type	Code Number to Enter	
		Decimal	Hexadecimal
%R	Register memory	8	08h
%AI	Analog input table	10	0Ah
%AQ	Analog output table	12	0Ch

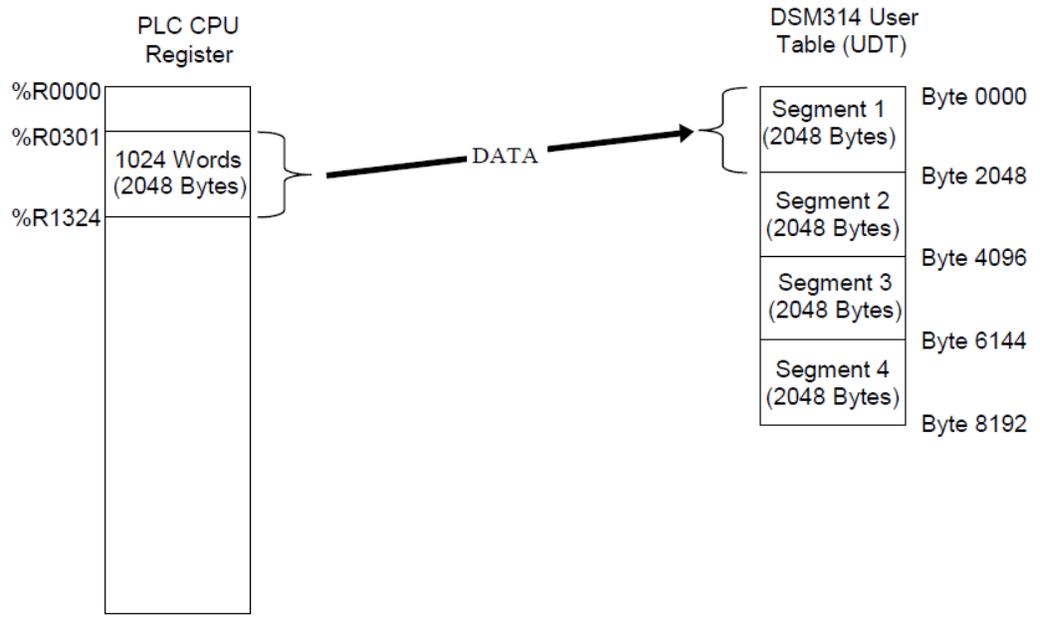
Data Start Pointer Offset (Word 10): This word contains the offset within the memory type selected in the Data Memory Type word (Word 9). Note: The Data Start Pointer Offset is a zero-based number. In practical terms, this means that you should subtract one from the address number that you wish to specify. For example, to select %R0001 as the Data Start location, enter zero ($1 - 1 = 0$). Or, to select %R0100, enter 99 ($100 - 1 = 99$). Note that the memory type, %R in this example, is specified in the previous word. The starting address designated by this word will be the first of 1024 contiguous words of memory used in the COMM REQ.

B-3.3 User Data Table COMM REQ Example

In this example, the following specifications are given:

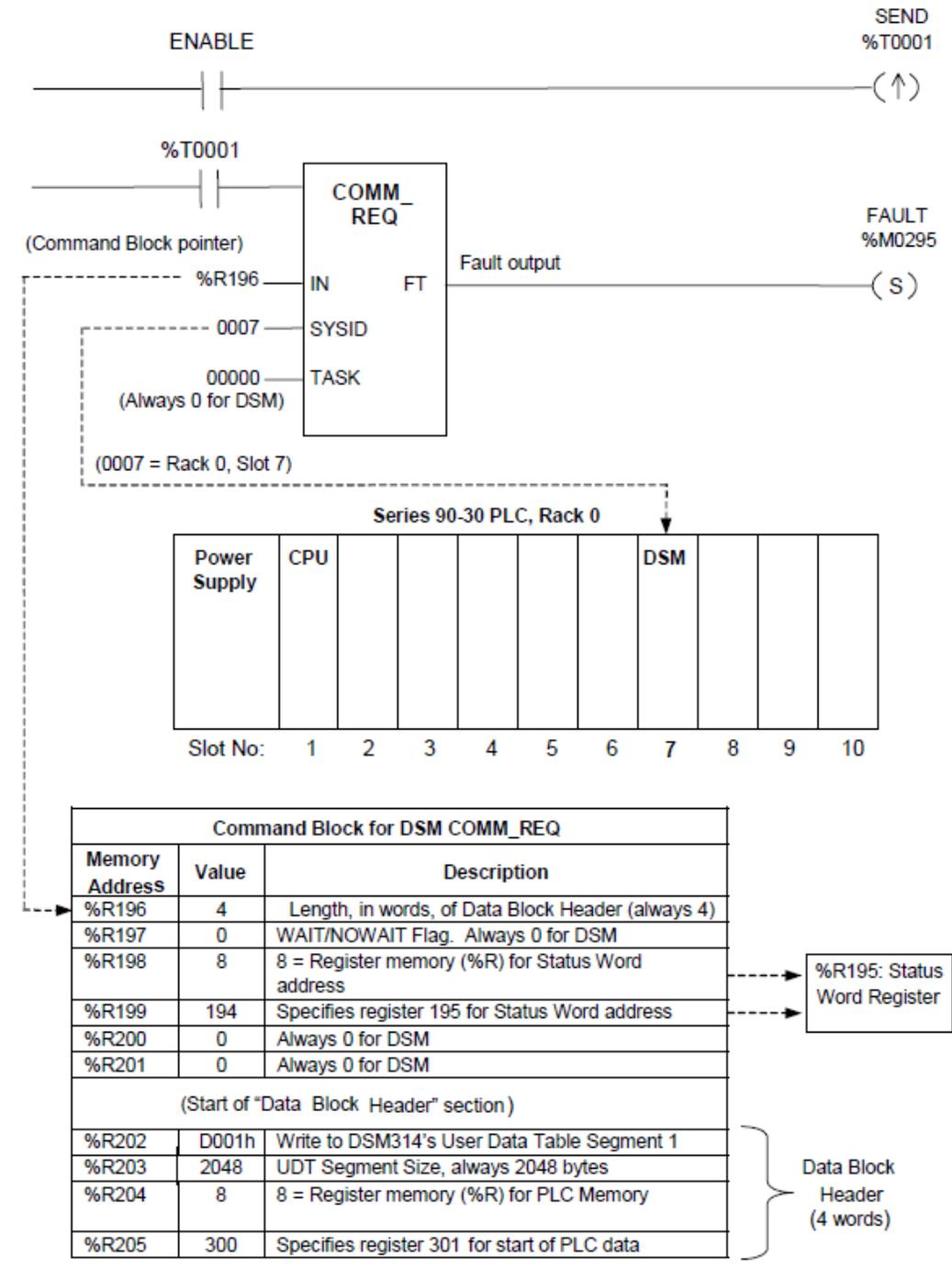
- The DSM314 module is mounted in Rack 0, Slot 7 of the PLC.
- The Command Block's starting address is %R0196.
- The Status Word is located at %R0195.
- The COMM REQ's FT (fault) output drives a Set Coil.
- Segment 1 of the DSM314 User Data Table is to be Written to. This is specified by the Command Code D001 in Word 7 of the Command Block.
- The data in a 1024-word (2048 byte) portion of register memory, %R0301 through %R1324, is copied and written into Segment 1 (2048 bytes) of the User Data Table. (Note that each %R word is two bytes in length.) This transfer of data is illustrated in the next figure:

Figure 183: Data Transfer for Command Code D001 (Write to Segment 1)



B-3.4 User Data Table COMM REQ Example

Figure 184: DSM314 UDT COMM REQ Example



B-4 The Parameter Load COMM REQ

B-4.1 The Command Block

The Command Block contains the details of a Communications Request. The first address of the Command Block is specified by the IN input of the COMM REQ Ladder Instruction. This address can be in any word-oriented area of memory (%R, %AI, or %AQ). The Command Block structure can be placed in the designated memory area using an appropriate programming instruction (the BLOCK MOVE instruction is recommended). The DSM Command Block has the following structure:

Table 88: DSM Parameter Load COMM REQ Command Block

Parameter Load COMM REQ Command Block for DSM Module		
Description	Address + Offset	Word No. and Value
Data Block Header Length	Address + 0	Word 1, always set to 4
WAIT/NOWAIT Flag	Address + 1	Word 2, always set to 0
Status Pointer Memory Type (see Memory Type Codes table, below)	Address + 2	Word 3, chosen by user (see Memory Type Codes table)
Status Pointer Offset	Address + 3	Word 4, chosen by user
Idle Timeout Value	Address + 4	Word 5, always set to 0
Maximum Communication Time	Address + 5	Word 6, always set to 0
Command Code	Address + 6	Word 7, always E501 (hex.)
Parameter Data Block Size, in bytes (must include 4 bytes for Parameter Specifier Words)*	Address + 7	Word 8, always set to 68 (44 hex)
Parameter Data Memory Type (for Word 11)	Address + 8	Word 9, chosen by user (see Memory Type Codes table)
Parameter Data Offset (for Word 11)	Address + 9	Word 10, chosen by user
Starting parameter number (0 - 255) in DSM Parameter Table	Address xyz (Address specified in Words 9 and 10)	Word 11, chosen by user
Number of parameters to load	Address xyz + 1	Word 12, chosen by user
1st parameter data	Address xyz+ 2/3	Word 13 and Word 14
2nd parameter data	Address xyz + 4/5	Word 15 and Word 16
...	Address xyz +
16th parameter data (4 bytes)	Address xyz + 32/33	Word 43 and Word 44

* Parameter Data Block size equals 4 bytes for the Parameter Specifier Words plus 4 bytes for each Parameter

Data Block Length (Word 1): The length of the Data Block header portion of the Command Block. It should be set to 4 for the DSM. The Data Block header is stored in Words 7 through 10 of the Command Block

WAIT/NOWAIT Flag (Word 2): This must always be set to logic zero for the DSM.

Status Word Pointer Memory Type (Word 3): This word specifies the memory type that will be used for the Status Word. Each memory type has its own specific code number, shown in the Memory Type Codes table below. So, for example, if you want to use %R memory for the Status Word, you would put either the decimal code number 8 or the hexadecimal code number 08h in this word.

Note that if you select a discrete memory type (%I or %Q), 16 consecutive bits will be assigned to the Status Word, beginning at the address specified in the Status Word Pointer Offset word, described below.

Status Word Pointer Offset (Word 4): This word contains the offset within the memory type selected. Note: The Status Word Pointer Offset is a zero-based number. In practical terms, this means that you should subtract one from the address number that you wish to specify. For example, to select %R0001, enter a zero ($1 - 1 = 0$). Or, if you want to specify %R0100, enter a 99 ($100 - 1 = 99$). Note that the memory type, %R in this example, is specified by the previous word (see the “Status Word Pointer Memory Type” explanation above).

Idle Timeout Value (Word 5): Since the DSM always uses the NOWAIT mode

(WAIT/NOWAIT flag always set to zero), this Idle Timeout Value parameter is not used for the DSM. Set it to zero.

Maximum Communication Time (Word 6): Since the DSM always uses the NOWAIT mode (WAIT/NOWAIT flag always set to zero), this Maximum Communication Time parameter is not used for the DSM. Set it to zero.

Command Code (Word 7): This is always E501(hexadecimal) for the DSM. To enter this value directly as a hexadecimal value, use a Word-type MOVE instruction. Also, since this value is 58,625 in decimal, an Integer-type MOVE instruction (limited to a maximum decimal value of 32,767 because bit 16 is used for the sign) does not have the capacity to contain it. A Word-type MOVE instruction can hold a decimal number up to 65,535 (FFFF in hex.).

Parameter Data Size (Word 8): Specifies the Parameter Data size in bytes. This value is always 68, which provides 4 bytes (for the first two words of the Parameter Data section) plus 4 additional bytes for each parameter loaded.

Parameter Data Memory Type (Word 9): This word specifies the memory type that will be used for Parameter Data. Each memory type has a unique code number, shown in the Memory Type Codes table below. So, for example, to specify %R memory, you would put either the decimal code number 8 or the hexadecimal code number 08h in this word.

Note that if you select a discrete memory type (%I or %Q), a group of 32 consecutive bits will be required for each parameter, and a group of 16 consecutive bits each will be required for Words 11 and 12.

Parameter Data Start Pointer Offset (Word 10): This word contains the offset within the memory type selected in the Parameter Data Memory Type parameter. Note: The Parameter Data Pointer Offset is a zero-based number. In practical terms, this means that you should subtract one from the address number that you wish to specify. For example, to select %R0001 as the Parameter Data Start location, enter zero ($1 - 1 = 0$).

Or, to select %R0100, enter 99 ($100 - 1 = 99$). Note that the memory type, %R in this example, is specified in the previous word.

Starting Parameter Number (Word 11): Specifies the number of the first parameter to be loaded to the DSM Parameter Table. Valid values are 0 – 255. However, to load all 16 parameters, the value of Word 11 must be 240 or less.

Number of Parameters to Send (Word 12): This parameter must always be set to 16.

Parameter Data (Words 13 - 44): The size of this Parameter Data area depends on the value in Word 12 (Number of Parameters to Send). Two words (4 bytes) of data are required for each parameter. Since the valid number of Double Integer parameters is 1 through 16, the Parameter Data area can be between 2 and 32 words.

COMM REQ Memory Type Codes: The codes in the following table are used in Word 3 (Status Word Pointer Memory Type), and Word 9 (Parameter Data Memory Type).

Table 89: Parameter Load COMM REQ Memory Type Codes

Parameter Load COMM REQ Memory Type Codes			
Memory Type Abbreviation	Memory Type	Code Number to Enter	
		Decimal	Hexadecimal
%I	Discrete input table	70	46h
%Q	Discrete output table	72	48h
%R	Register memory	8	08h
%AI	Analog input table	10	0Ah
%AQ	Analog output table	12	0Ch

B-4.2 DSM Parameter Load COMM REQ Example

This example is used as the basis for the following section, “Section 5: COMM REQ Ladder Logic Example.” In this example, the following specifications are given:

- The DSM module is mounted in Rack 0, Slot 7 of the PLC.
- The Command Block’s starting address is %R0196.
- The Status Word is located at %R0195.
- 16 parameters are to be sent.
- The COMM REQ’s FT (fault) output drives a Set Coil.
- DSM Parameter 1 is considered critical in this example application. The last two rungs of the “COMM REQ Ladder Logic Example” (see Section 5) verify that Parameter 1 received the correct value via the COMM REQ.
- The data in 32 words (16 double words) of memory, %R0208 through %R0239, are copied to 16 double word parameter registers, P001 through P016, in DSM314 parameter memory. This transfer of data is illustrated in the next figure:

Figure 185: Data Transfer for Parameter Load COMM REQ Example

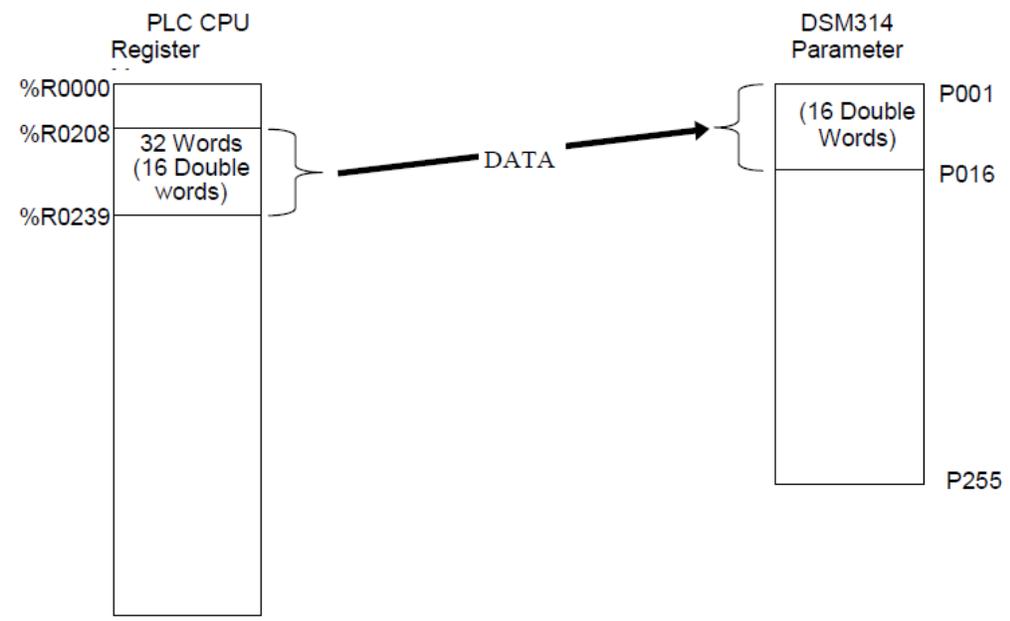
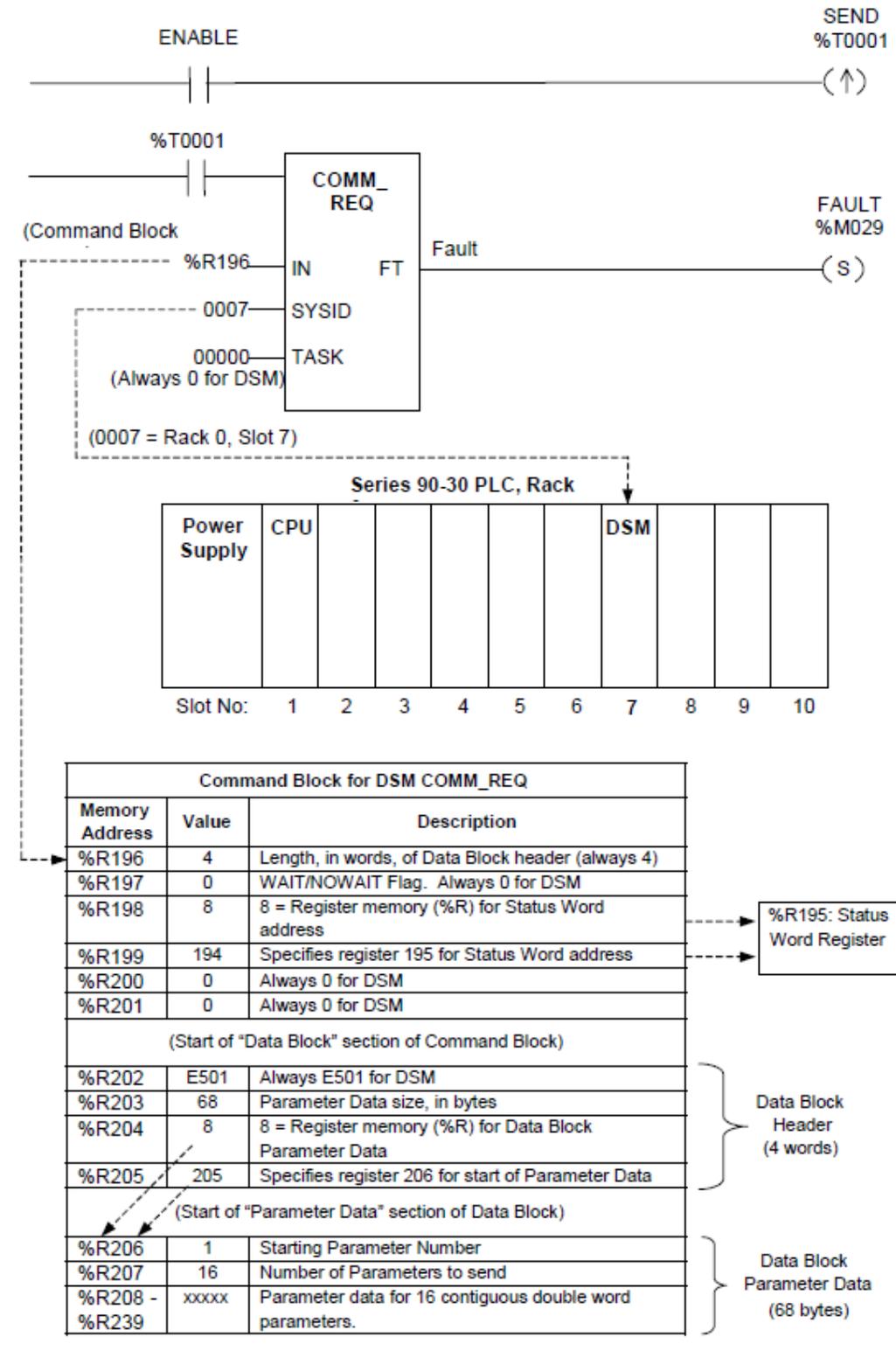


Figure 186: Overview of the Parameter Load COMM REQ Example



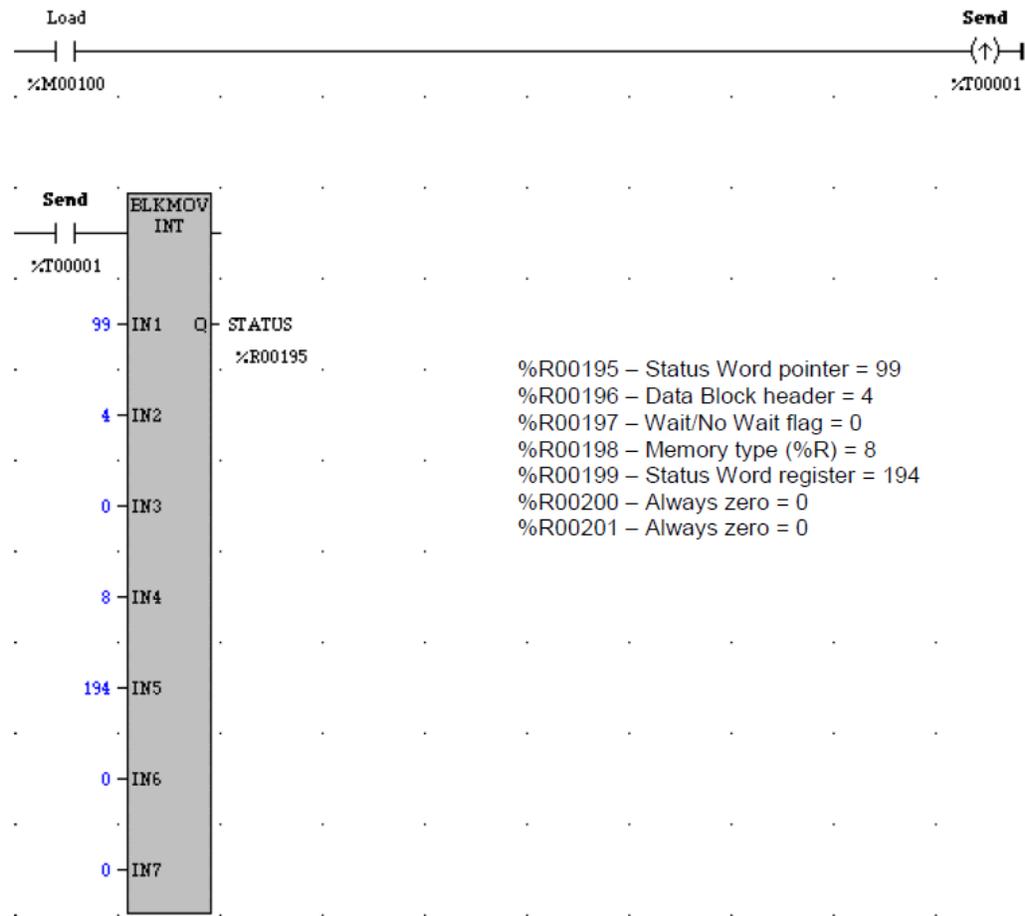
B-5 COMM REQ Ladder Logic Example

The following ladder logic example is based upon the Parameter Load COMM REQ example in the previous section. Refer to the table on the previous page for the Command Block listing.

Setting up the COMM REQ Command Block Values

The next two rungs load the appropriate values into the first seven words of the COMM REQ's Command Block.

Figure 187



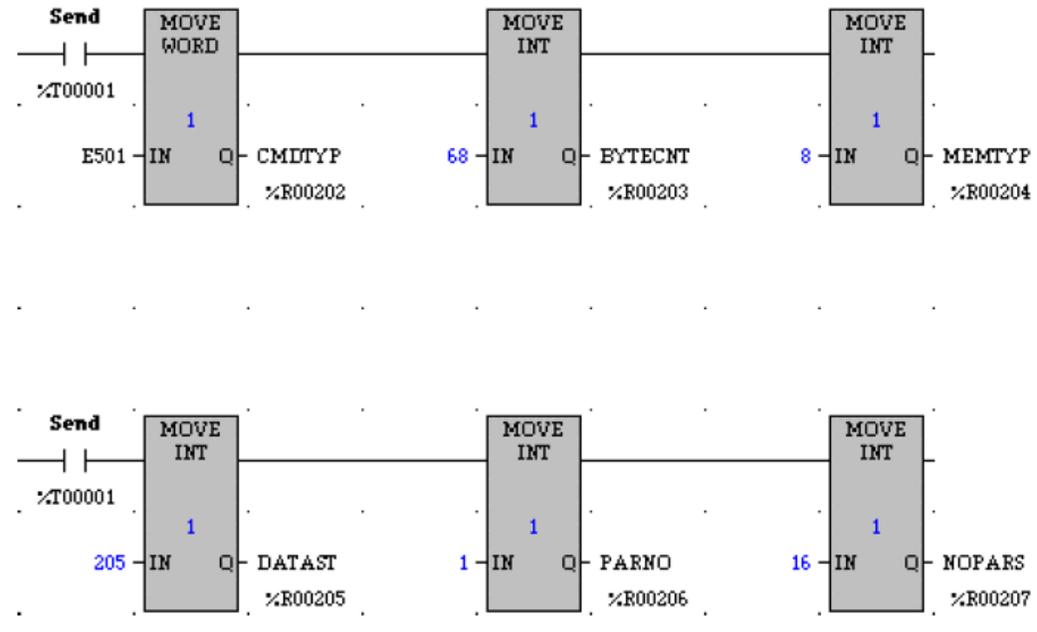
In the following two rungs, the remainder of the Command Block data is loaded. This data is listed next:

- %R00202 – Command. For DSM, it's always = E501 (hex)
- %R00203 – Parameter data size, in bytes = 68
- %R00204 – Memory type code for %R memory = 8
- %R00205 – Starting register for Parameter Data (offset by one) = 205
- %R00206 – Starting Parameter Number = 1

%R00207 – Number of Parameters to send = 16

%R00208 - %R00239 – Parameter data to be sent

Figure 188



Logic for Parameter Data (not Shown)

Additional logic will be required to load your data into registers %R00208 - %R00239 so that it can be sent to the DSM314 parameters. (The value in double word %R00208/%R00209 will be sent to Parameter 1, the value in %R00210/%R00211 will be sent to Parameter 2, and so on, until finally, the value in %R00238/%R00239 will be sent to Parameter 16.) The method to be used for loading the data into these registers depends upon your application. If the data values will not change, constants can be moved into the registers using Block Move and/or Move instructions. If the values are to change, they could be moved into the registers from an operator interface device.

Handling Double Integer Parameter Values and Input Value Scaling

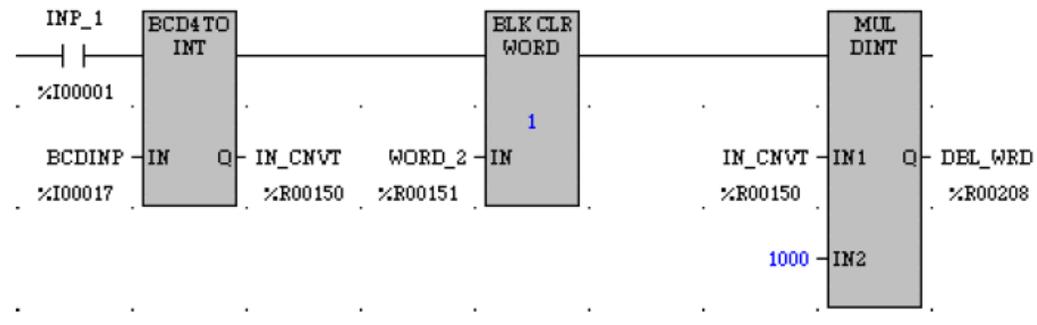
The data in the single precision registers (16 bits) needs to be converted to double-integer (32 bits) form because the DSM's parameters are double-integer size. A convenient way to do this is to use a Double Integer Multiply (MUL DINT) instruction to move input data into the registers whose contents will be sent to the DSM. There are two possible advantages to this approach:

- This is an easy way to convert single integer registers to double-integer form.
- It lets you easily scale the input values if you should need to. The term scaling refers to multiplying and/or dividing a value to create a new value that is proportional to the original value. For example, multiplying an input value by two, then dividing it by 3 would result in an output value that is always 2/3 the size of the input value. Scaling is often required in a servo system to match the actual distance moved to the distance commanded. It is doing so, it provides the function of an "electronic

gearbox.” It can be used to allow for gear ratio, ballscrew pitch, encoder resolution, and customer input value preference.

In the example below, the integer value from an Operator Input device (a 4-digit BCD thumbwheel switch) will be multiplied by a factor of 1000, then placed into the double-integer word %R00208/%R00209 (for Parameter 1).

Figure 189



In the example above, when switch %I00001 is closed, the Binary Coded Decimal (BCD) value in BCDINP (%I00017-%I00032) from a BCD Operator Input device is converted to an integer value (by the BDC4 TO INT instruction), and the integer value is placed in register %R00150. Next, %R00151 is cleared to zero by the BLK CLR instruction. Note that on the output of the BCD4 TO INT instruction, %R00150 is a single integer value. However, when %R00150 is used as an input (IN1) for the double integer Multiply instruction (MUL DINT), the CPU automatically combines it with the next %R address (%R00151) to form a double-integer value. Word %R00150 becomes the Least Significant Word, and %R00151 becomes the Most Significant Word in this double-integer word. The MUL DINT instruction multiplies the value in %R00150/%R00151 by 1000 and places the result in double word %R00208/%R00209 (DBL_WRD).

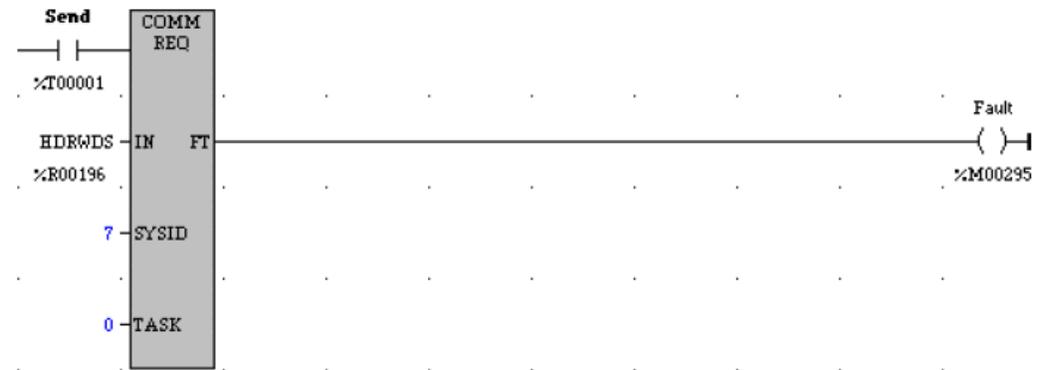
When used this way, %R00151 is called an “implied address” since it is not shown on the screen. Be aware that you must not use %R00151 for any other purpose (it should be held to a value of zero); otherwise, the value placed into %R00150 from the BDC4 to INT instruction would be altered. The same principle applies in the case of double word %R00208/R00209. Here, the use of %R00209 is implied by the fact that %R00208 is displayed as the output of the Double Integer Multiply (MUL DINT) instruction. So %R00209 should be reserved for this use only.

In this rung, the MUL DINT instruction performs two functions: (1) it converts the value in %R00150 from single integer form to double integer form, and (2) it scales the value in %R00150/%R00151 by multiplying it by 1000. If scaling had not been desired, a value of 1 would be used instead of 1000 at IN2 of the MUL DINT instruction; this would provide conversion to double integer without changing (scaling) the value.

The Communications Request Instruction

The next figure shows the Communications Request (COMM REQ) instruction. The IN input contains the address of the first word of the command block. The SYSID input contains the rack and slot number (rack 00, slot 07) of the DSM314 targeted by this COMM REQ. The TASK input is always zero for the DSM314. The FT output connects to a coil (%M00295) that will be energized if a fault is detected.

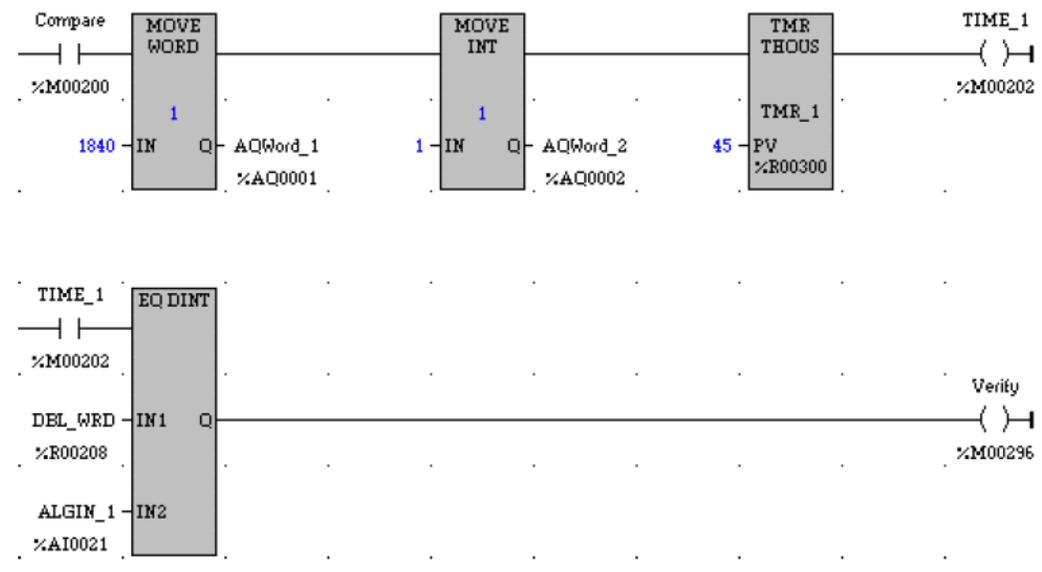
Figure 190



Verifying the Data Sent to Parameter 1

In this example, the value in DSM Parameter 1 is critical because it specifies a move distance that, if incorrect, could result in machine damage. So, the logic in the following two rungs verifies that Parameter 1 received the correct value. If the value is not correct, contacts (not shown) from output coil “VERIFY” in the second rung will prevent the DSM from producing motion.

Figure 191



First Rung: The MOVE WORD instruction moves hexadecimal number 1840 into %AQ00001, the first word of the Immediate Command. The low byte value (40) of this number specifies the Select Return Data Immediate Command. The high byte value (18) specifies the Mode selection for Parameter Data.

The MOVE INT instruction moves a decimal value of 1, indicating Parameter 1, into %AQ00002. This commands that the value in DSM Parameter 1 be written to the User Selected Data double word for Axis 1, %AI00021/AI00022 in this example.

Note: *The actual %AI addresses used for any DSM module are specified when the module is configured.*

The TMR THOUS (thousandths) timer instruction produces a 45-millisecond time delay after the Select Return Data Immediate Command is sent. This is required because User Selected Data is not available in the ladder until at least 3 sweeps or 20 milliseconds (whichever is greater) elapses after the Select Return Data Immediate Command is sent. Since the sweep time in this example is 14 milliseconds, this 45-millisecond delay ensures that the Parameter 1 data will be present in the User Selected Data double word before the Equal instruction in the next rung executes. Note that contact %M00200 must stay ON long enough for the TMR timer to time out and enable the second rung.

Second Rung: After the 45-millisecond delay in the previous rung elapses, contact %M00202 closes and enables this rung. In this rung, a double integer EQUAL instruction compares the value in %R00208/R00209 (the source of the value sent by the COMM REQ to DSM Parameter 1) with the value returned from Parameter 1 in %AI00021/AI00022. If the values are equal, coil “Verify” will turn on.

Appendix C: Position Feedback Devices

Four α and β Series Digital serial encoder models function with the DSM314:

Table 90: Digital Serial Encoder Resolutions

8K	(8,192 cts/rev)	- No longer available on new motors
32K	(32,768 cts / rev)	- Standard on β Series motors
64K	(65,536 cts/rev)	- Standard on α Series motors
1000K	(1,048,576 cts/rev)	- Optional on α Series motors

Note: *The older “A” or “C” Series million count serial encoder will not operate with the DSM314. An error will be reported if this encoder is connected.*

For position control purposes, by default, the DSM314 treats all encoders as 8192 counts/rev. The additional resolution of 32K, 64K and 1000K encoders will still be used in the digital servo velocity controller to provide smooth operation at low speeds. To use the increased position feedback resolution, refer to the Tuning Parameters section of Chapter 4.

C-1 Digital Serial Encoder Modes

The Digital serial encoders can be operated in either **Incremental** mode or **Absolute** mode. The mode is configured using the **Feedback Mode** selection in the configuration software. Proper operation of the **Absolute** mode requires an external battery pack that must be connected to the servo amplifier. Refer to the appropriate amplifier manual for selection and installation of the battery pack.

C-2 Incremental Encoder Mode Considerations

The digital serial encoder can be used as an incremental encoder returning 8192 counts per shaft revolution, with no revolution counts retained through a power cycle. The equivalent of a marker pulse will occur once each motor shaft revolution. All **Home Modes** (Home Switch, Move+, Move-) and Set Position %AQ commands reference the axis, and set the Position Valid %I bit upon successful completion. The configured **High Position Limit** and **Low Position Limit** are valid and the Actual Position %AI status word as reported by the DSM314 will wrap from high to low count or from low to high count values. This is an excellent mode for continuous applications that will always operate via incremental moves, in the same direction. **Home Offset** and **Home Position** configuration items allow simple referencing to the desired location.

C-3 Absolute Encoder Mode Considerations

The Digital serial encoder can be used as an absolute type encoder by adding a battery pack to retain servo position while system power is off. A Find Home cycle or Set Position %AQ command must be performed initially or whenever encoder battery power is lost with the servo amplifier also in a powered down state. **Feedback Mode** set to ABSOLUTE must be selected in the configuration software for proper operation with a battery pack.

C-3.1 Absolute Encoder - First Time Use or Use After Loss of Encoder Battery Power

The absolute encoder temporarily provides incremental data during the first use or after restoring encoder battery power. The incremental data is lost when motor shaft rotation causes the encoder to pass a reference point (similar to a marker signal) within one revolution of the motor shaft. The Digital Absolute serial encoder must be rotated up to one full revolution after the absolute mode battery has been reattached to the amplifier. The encoder will reference itself within one revolution and report a referenced status to the DSM314.

C-3.2 Absolute Encoder Mode - Position Initialization

When a system is first powered up in Absolute Encoder mode, a position offset for the encoder must be established. Using the %Q Find Home cycle or the Set Position % AQ command can accomplish this.

Find Home Cycle - Absolute Encoder Mode

The Find Home Mode can be configured for Move (+), Move (-) or Home Switch operation. Refer to Chapter 4 for additional details of Home Cycle operation. The **Home Offset** and **Home Position** configuration items function the same as in Incremental Encoder mode. At the completion of the Home Cycle, the Actual Position %AI status word is set to the configured **Home Position** value. The DSM314 internally calculates the encoder Absolute Feedback Offset needed to produce the configured **Home Position** at the completion of the Home Cycle. This Absolute Feedback Offset is immediately saved in the DSM314 non-volatile (capacitor backup) memory.

Once an absolute position is established by successful completion of a Find Home cycle, the DSM314 will automatically initialize the Actual Position %AI status word after a power cycle and set the Position Valid %I bit.

Note: *If the Position Valid %I bit is set before initiating a Home Cycle, the Home Cycle clears Position Valid and then sets Position Valid again when the cycle completes. If the Home cycle is halted by an Abort All Moves %Q bit command, Position Valid will remain off. However cycling power will cause a valid Actual Position to be restored and Position Valid will be automatically set.*

Set Position Command - Absolute Encoder Mode

The Set Position %AQ command functions the same way as in incremental encoder mode. At the completion of the Set Position operation, Actual Position is set to the Set Position value. The DSM314 internally calculates the encoder Absolute Feedback Offset needed to produce the commanded Set Position value. This Absolute Feedback Offset is immediately saved in the DSM314 non-volatile (capacitor backup) memory.

If a Set Position AQ command is received before the encoder has been referenced, Error Code 53(hex) “Attempt to initialize position before digital encoder passes reference point” will be reported. This error code is only reported if the Feedback Mode is set to **Absolute**. Serial Encoders configured for **Incremental** mode do not have this restriction.

Once an absolute position is established by a Set Position command, the DSM314 automatically initializes Actual Position after a power cycle and sets the Position Valid %I bit.

C-3.3 Absolute Encoder Mode - DSM314 Power-Up

The battery pack attached to the servo subsystem maintains power to the encoder counter logic. Once the encoder has referenced through first time start up, the encoder automatically maintains the actual position, even if the axis is moved during servo power loss. The encoder monitors the status of the battery pack, and reports loss of battery power or low battery power to the DSM314.

The DSM314 completes a power-on diagnostic, and when configured for absolute encoder mode, interrogates the referenced status of the Digital serial encoder. A valid referenced status from the encoder signals the DSM314 to read the encoder absolute position. The DSM314 reports the Actual Position %AI status as the sum of the encoder position and the Absolute Feedback Offset established by the initial Find Home cycle or Set Position %AQ command.

C-3.4 Incremental Quadrature Encoder

Incremental Quadrature Encoders provide three output signals to the DSM314: Channel A, Channel B, and Marker. The Channel A and Channel B signals transition as the encoder turns, allowing the DSM314 to count the number of signal transitions and calculate the latest encoder position change and direction of rotation.

Incremental Quadrature Encoders are incremental feedback devices; they do not provide a continuous indication of absolute shaft angle as the input shaft rotates. For this reason, the DSM314's Actual Position %AI status word must be initialized with a known physical position before positioning control is allowed. This position alignment can be accomplished using the Set Position %AQ Immediate command or the %Q Find Home cycle. The home cycle makes use of the encoder marker channel, which is a once per revolution pulse produced at a known encoder shaft angle. Successful completion of the %Q Find Home cycle or a Set Position %AQ command causes the DSM314 to set the axis Position Valid %I bit. Position Valid must be set before motion programs will be allowed to execute. Position Valid is only cleared by an encoder Quadrature Error (Channel A and Channel B switching at the same time) or by turning on the Find Home and Abort %Q bits simultaneously.

Note: *In Digital Mode, only incremental quadrature encoders are supported for the Follower mode master axis.*

Appendix D: Tuning Digital and Analog Servo Systems

This appendix provides a procedure for starting up and tuning a Digital or Analog servo system. For Digital servos systems, there are two control loops in the DSM314 that require tuning, the velocity loop and the position loop. Always begin with module configuration then proceed to the velocity loop setting and finally the position loop. For Analog servo systems, there are a series of Start-Up Procedures to follow.

D-1 Start-Up and Tuning Information for Digital Servo Systems

There are three major sections covered:

- Validating Home Switch, Over Travel Inputs and Motor direction.
- Tuning the Velocity Loop.
- Tuning the Position Loop.

D-1.1 Validating Home Switch, Over Travel Inputs and Motor direction

1. Connect the motor, amplifier and DSM314 module following the procedures in Chapter 2.
2. If Over travel Limit switches are used (**Overtravel Limit Switch** = Enabled in configuration), wire them to the correct 24V terminal board points (refer to Chapter 3). The overtravel inputs are operated in the fail-safe mode i.e. a normally closed or PNP type switching device should be used. Current must be sourced to the input to maintain a logic level 1 on the input while the axis is NOT at the overtravel position or an alarm condition (Error A9) will be returned. Otherwise the **Overtravel Limit Switch** configuration must be set to Disabled using the configuration software.
3. If a Home switch is used (Home Mode = Home Switch in configuration), wire it to the correct 24V terminal board points (refer to Chapter 3). The Home switch must be wired and actuated so that it is ALWAYS ON (closed) when the axis is on the negative side of home and ALWAYS OFF (open) when the axis is on the positive side of home. Typically, the Home switch is mounted at or near one end of the axis travel. It is important to verify the operation of the home switch prior to attempting a home cycle. It may be necessary to reverse the motor direction (Motor1 or Motor2 Dir = POS/NEG) in the module configuration.

- Use the configuration software to set the desired user scaling factors and other configurable parameters. The following items **MUST** be changed from the default configuration settings:

Configuration Item	Setting
Axis 1 Mode	Digital Servo
Motor Type:	Select from Table in Chapter 4
Position Loop Time Constant:	60 ms
Velocity Loop Gain:	(Load Inertia / Motor Inertia) * 16
User Units : Counts (Standard Mode Only)	See Chapter 3
Position Error Limit:	30000 x User Units / Counts

Set the configuration parameters in the order shown above.

- Store the configuration to the host controller.
- Clear the program from the host controller, turn off all DSM314 %Q bits and place the host controller in RUN mode. Monitor the %I CTL bits for Home Switch, (+) Overtravel and (-) Overtravel and confirm that each bit responds to the correct switch (Refer to Chapter 5 for %I bit definitions).
- Turn on the Enable Drive %Q bit and confirm that the servo amplifier is enabled. If a brake is used on the servomotor it should be released at this time.
- Send the %AQ command code for Force Digital Servo Velocity 100 (rpm). Confirm that the motor moves in the desired POSITIVE direction and the Actual Velocity reported in the %AI table is POSITIVE. If the motor moves in the wrong direction, use the **Axis Direction** parameter in the configuration software to swap the positive and negative axis directions.
- Remove the Force Digital Servo Velocity command from the %AQ table. Use a low **Jog Velocity** and **Jog Acceleration** in the configuration, values may be increased later. Turn on the Jog Plus %Q bit. Confirm that the servo moves in the proper direction and that the Actual Velocity reported by the DSM314 in the %AI table matches the configured **Jog Velocity**. If Motion Programs will use an acceleration higher than **the Jog Acceleration**, it may be necessary to increase **Jog Acceleration** so that Abort All Moves and Normal Stop actions will operate as expected.
- Use a low value for **Find Home Velocity** and **Final Home Velocity** in the module configuration, values may be increased later. Check for proper operation of the Find Home cycle by momentarily turning on the Find Home %Q bit (the Drive Enabled %Q bit must also be maintained on). The axis should move towards the Home Switch at the configured **Find Home Velocity**, then seek the Encoder Reference point at the configured **Final Home Velocity**. If necessary, adjust the configured velocities and the location of the Home Switch for consistent operation. The final Home Switch **MUST** transition at least 10 milliseconds before the encoder reference point is encountered. The physical location of **Home Position** can be adjusted by changing the **Home Offset** value with the configuration software.

11. Monitor servo performance and use the Jog Plus and Jog Minus %Q bits to move the servomotor in each direction. Placing the correct command code in the %AQ table can temporarily modify the Position Loop Time Constant. For most systems the **Position Loop Time Constant** can be reduced until some servo instability is noted, then increased to a value approximately 50% higher. Once the correct time constant is determined, the DSM314 configuration should be updated using the configuration software. **Velocity Feedforward** can also be set to a non-zero value (typically 90 – 100 %) for optimum servo response. Refer to Tuning a Digital Servo for information on setting the digital servo **Velocity Loop Gain**.
12. If Follower mode is used with an Incremental Quadrature Encoder, confirm that Actual Position (Aux Axis 3) represents the encoder position. **Make sure the desired Follower axis slave: master ratio has been programmed as the A:B ratio using the configuration software.**

Digital Servo System Startup Troubleshooting Hints

1. The DSM314 requires a Series 90-30 CPU with firmware release 10.0 or later, or a PACSystems RX3i CPU (version 2.8 or later).
2. DSM support for Beta M1 and Beta M0.5 motors requires DSM firmware version 3.0 or later.
3. The default DSM314 configuration for the **Overtravel Limit Switch** inputs is ENABLED. Therefore, 24 VDC must be applied to the Overtravel inputs or the DSM314 will not operate. If Overtravel inputs are not used, the DSM314 configuration should be set to **Overtravel Limit Switch** inputs DISABLED.
If the Axis Enabled %I bit is OFF, the axis will not respond to any %Q bits or %AQ commands. When a servomotor is not used with a Servo Axis, the **Motor Type** must be set to 0 or Axis Enabled will stay OFF. A **Motor Type** of 0 disables the axis servo loop processing and sets Axis Enabled ON, allowing the axis to accept commands such as Load Parameter Immediate and Set Analog Output Mode.
4. **The Enable Drive %Q control bit must be set continuously to ON or no motion other than jogs will be allowed.** If no STOP errors have occurred, the Drive Enabled %I status bit will mirror the state of the Enable Drive %Q bit. A STOP error will turn off Drive Enabled even though Enable Drive is still ON. The error condition must be corrected, and the Clear Error %Q control bit turned ON for one host controller sweep to re-enable the drive.
5. If the Module Error Present %I status bit is ON and the Axis Enabled and Drive Enabled %I status bits are OFF, then a STOP error has occurred (Status LED flashing fast). **In this state, the Servo Axis will not respond to any %Q bits or %AQ commands other than the Clear Error %Q bit.**
6. The Clear Error %Q control bit uses one-shot action. Each time an error is generated, the bit must be set OFF then ON for at least one sweep to clear the error.

- The CFG OK LED must be ON or the DSM314 will not respond to host controller commands. If the LED is OFF then a valid DSM314 configuration has not been received from the host controller, or there may be a recognized configuration error. Check the %AI error code words for Dxxx errors, which are documented in the “System Error Codes” section of Appendix A. Also check the PLC fault tables for reported configuration errors.

D-1.2 Tuning a Digital Servo Drive

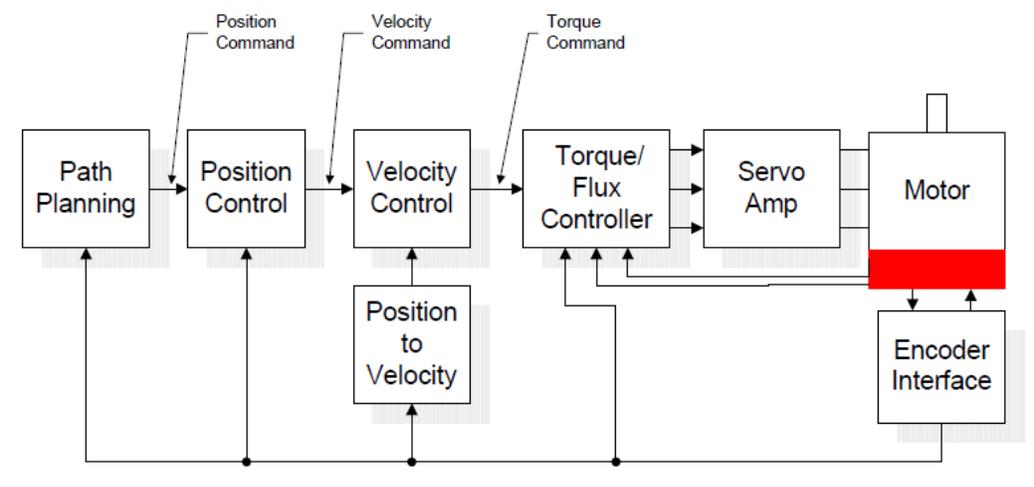
The following pages provide you with an introduction to the basics required for tuning a Digital servo drive. This introduction shows one method for tuning a servo drive. The method will not work in all applications, and you should modify the approach based on the application. In order to display and measure the necessary signal waveforms, the DSM314 analog outputs must be connected to an oscilloscope. Without an oscilloscope to measure the signals, tuning the servo drive with the following approach will not be possible. The Select Analog Output Mode %AQ command (47h) is used to select the data that is sent to the analog outputs during servo tuning. Refer to Chapter 5 for a discussion of this %AQ command.

Tuning Requirements

The module has three main parameters that are adjusted during tuning. The parameters are the **Position Loop Time Constant**, **Velocity Feed Forward Gain**, and **Velocity Loop Gain**. The **Position Loop Integrator Time Constant** gives the position loop an additional degree of freedom but in typical applications is not required.

The approach to tuning the control loops is to tune the inner control loops first. In this example, the inner control loop that requires tuning is the velocity loop. As shown in the figure below, the position loop is the outer loop and sends velocity commands to the velocity loop.

Figure 192: Control Loops Block Diagram



Tuning the DIGITAL MODE Velocity Loop

The proper method to tune the velocity loop is to separate the velocity loop from the position loop. To achieve this separation, a method must be used to directly send velocity commands without using the position loop control. The DSM module has several modes that will allow the user to send a velocity command directly to the velocity loop. Two methods are as follows:

Method #1:

The Force Digital Servo Velocity %AQ immediate command (34h) will send a velocity command directly to the velocity loop. This command is different from the Move at Velocity Command, which uses the position loop to generate the command. This is important since the position loop should not be interacting with the velocity loop at this point in the tuning process. The Force Digital Servo Velocity %AQ command allows the user to generate a step change in the velocity. The velocity command step is then used to generate the velocity loop step response. The user should note that when a velocity command step change is performed the acceleration is limited only by the bandwidth of the velocity loop. In some applications this can cause damage to the controlled device due to the high acceleration rate.

Method #2:

In some applications, method #1 introduces too large a shock to the device under control. In these cases, another method to generate a velocity command is needed. The method requires that the user set the position loop to an open loop configuration. The position loop is set to open loop by setting the **Position Loop Time Constant** to zero and the **Velocity Feedforward Gain** to 100 percent. You can then use the Move at Velocity Command or a motion program to generate velocity commands to the servo drive.

The first parameter that needs to be adjusted is the **Velocity Loop Gain**. The parameter adjusts the velocity loop bandwidth. As a starting point use the following formula (also reference the Velocity Loop Gain Section):

Equation 1

$$\text{Velocity Loop Gain} = \frac{J_l}{J_m} 16$$

Where :

J_l = Load Inertia

J_m = Motor Inertia

The **Velocity Loop Gain** calculated in equation 1 in many cases will not need to be altered. However, due to the application (for example, machine resonance) the value may need to be adjusted. To tune the **Velocity Loop Gain** the following procedure can be used:

1. Choose the method to introduce velocity command to the velocity loop. Method #1 and Method #2 (above) are examples of methods to perform this task.

2. Connect an oscilloscope to the analog outputs for Motor Velocity feedback and Torque Command. See Section 4.25 of Chapter 5 for analog output configuration instructions.
3. Set the Velocity Loop Gain to zero. This is a conservative approach. If the application is known to not have resonant frequencies from zero to approximately 250 Hz, you can start with a higher value, but do not exceed the value calculated in equation 1 at this point.
4. Generate a velocity command step change. At this point the step change should be relatively small compared to the full speed of the machine. Ten to 20 % of the rated machine speed is a good start.
5. Observe the Motor Velocity and Torque Command on the oscilloscope. The objective is to obtain a critically damped velocity loop response. Pay particular attention to any oscillations that are occurring in the velocity feedback signal.
6. Increase the **Velocity Loop Gain** in small steps and repeat 4 and 5 until instability in the Motor Velocity feedback signal is observed. Once this point is reached, decrease the **Velocity Loop Gain** by at least 15 %. As a general rule, the lower the **Velocity Loop Gain** value that meets the system requirements the more robust the control. You should carefully observe the velocity feedback signal. In some applications, running the **Velocity Loop Gain** high enough to create instability can cause machine damage. If in doubt, adjust the **Velocity Loop Gain** to be no greater than the value calculated in equation 1. If oscillations are observed in the Motor Velocity feedback signal prior to this point, decrease the **Velocity Loop Gain** and continue with step 7 below.
7. The velocity loop is tuned at this point. However, the robustness of the loop must be checked. To perform this test, introduce velocity command steps in increments of 20% Rated Machine Speed, 40% Rated Machine Speed, 60% Rated Machine Speed, 80% Machine Rated Speed, and 100% Rated Machine Speed. Observe the Motor Velocity and Torque Command signals for any instability. If an instability or resonance is observed, reduce the Velocity Loop Gain and repeat the test.

Note: For Digital servos, the %AQ Force Analog Output command can provide Torque Command or Commanded Motor Velocity. ($Velocity = 750 \text{ rpm/volt}$ and $\%TqCmd = (100/1.111111 \text{ Volt}) * X \text{ Volt}$ or $Torque \text{ Cmd} = 100\% \text{ Torque Command} = 1.111 \text{ Volts}$, $100\%TqCmd = \text{MaxCur Amplifier}$. For instance: $Beta \ 0.5 \ \text{MaxCurAmp} = 12 \ \text{amps} \Rightarrow 1.111111 \ \text{volt} = 12 \ \text{amps}$.)

Sample DIGITAL MODE Velocity Loop Tuning Session

A sample velocity loop tuning session is shown in the plots that follow.

Figure 193: Velocity Loop Step Response Velocity vs. Time VLGN = 0

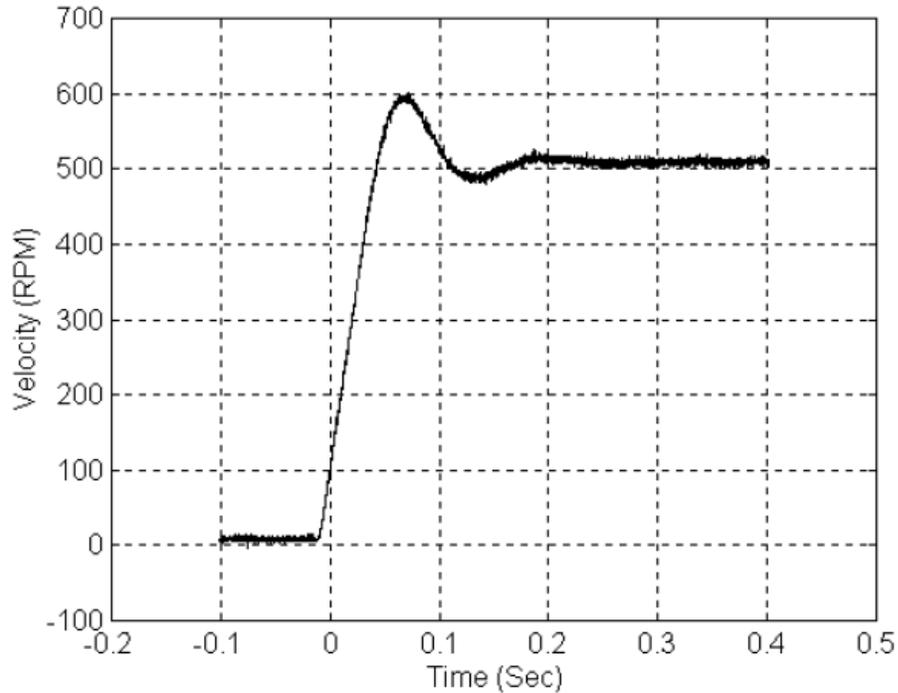
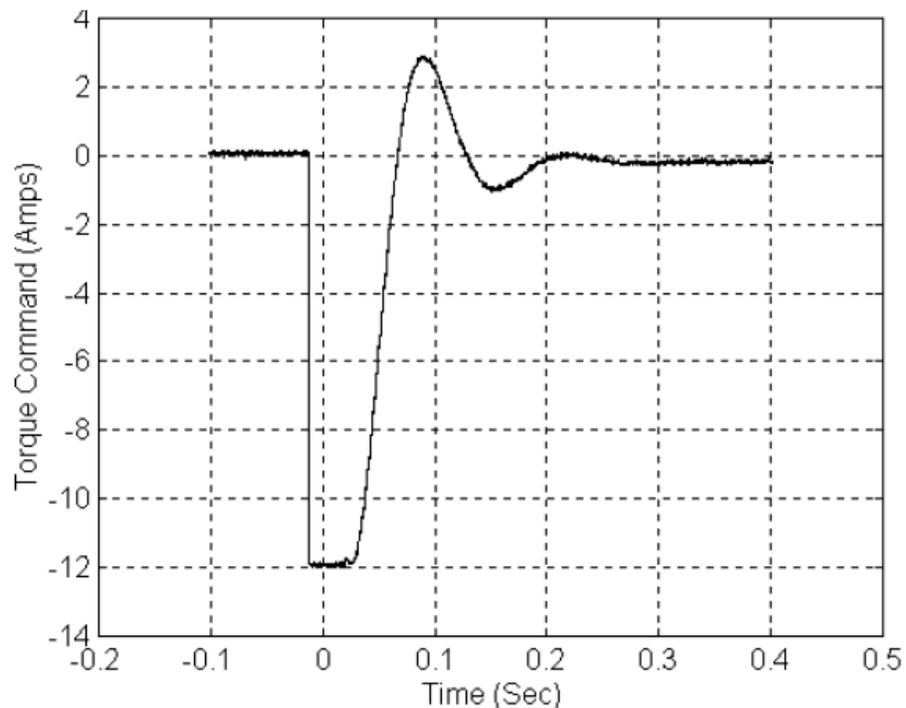


Figure 194: Velocity Loop Step Response Torque Command vs. Time VLGN = 0



Note that in Figures 193 and 194 the system does not have enough damping. In this case the controller does not have the required bandwidth and the Velocity Loop Gain must be increased.

Figure 195: Velocity Loop Step Response Velocity vs. Time VLGN = 24

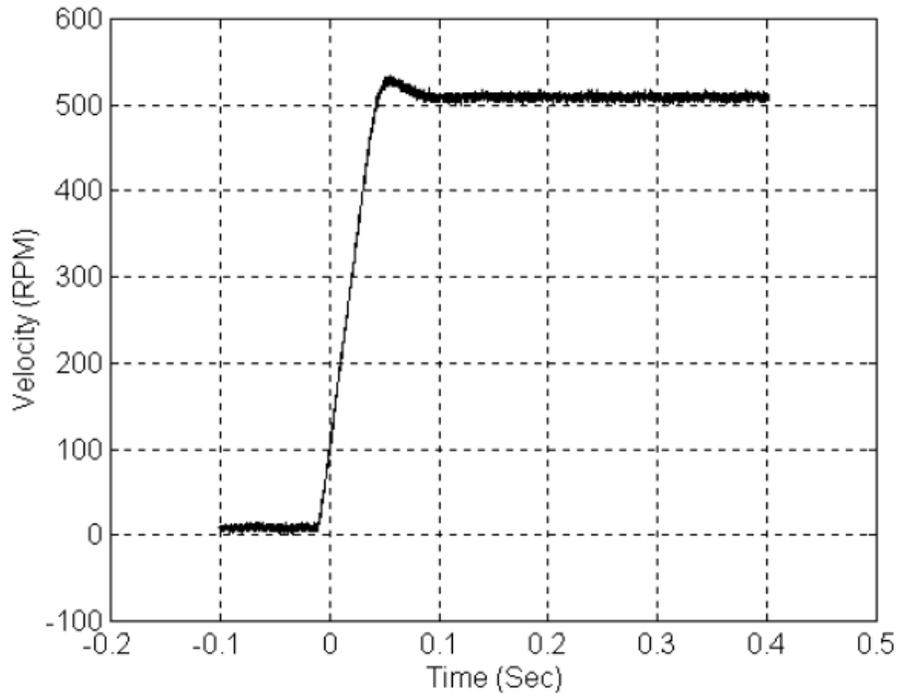
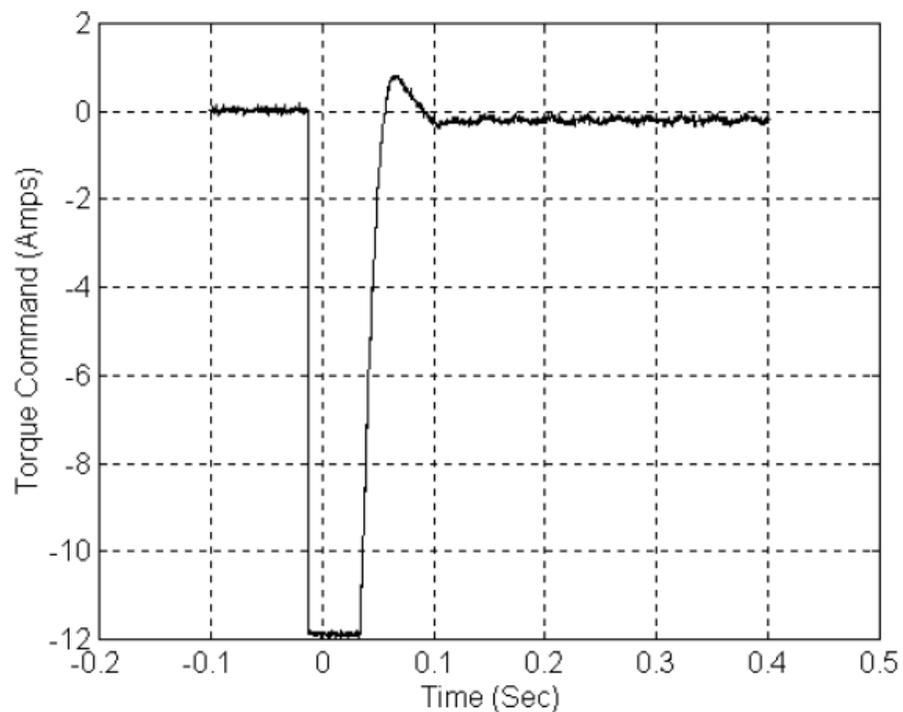


Figure 196: Velocity Loop Step Response Torque Command vs. Time VLGN = 24



Note that in Figures 195 and 196, the system is beginning to look acceptable. The only problem is the velocity overshoot.

Figure 197: Velocity Loop Step Response Velocity vs. Time VLG = 48

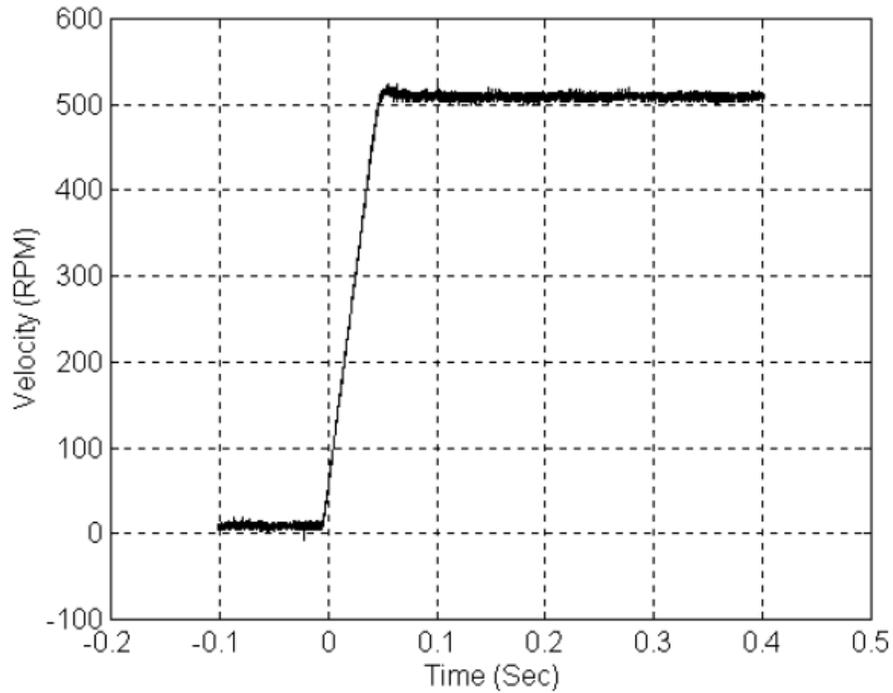
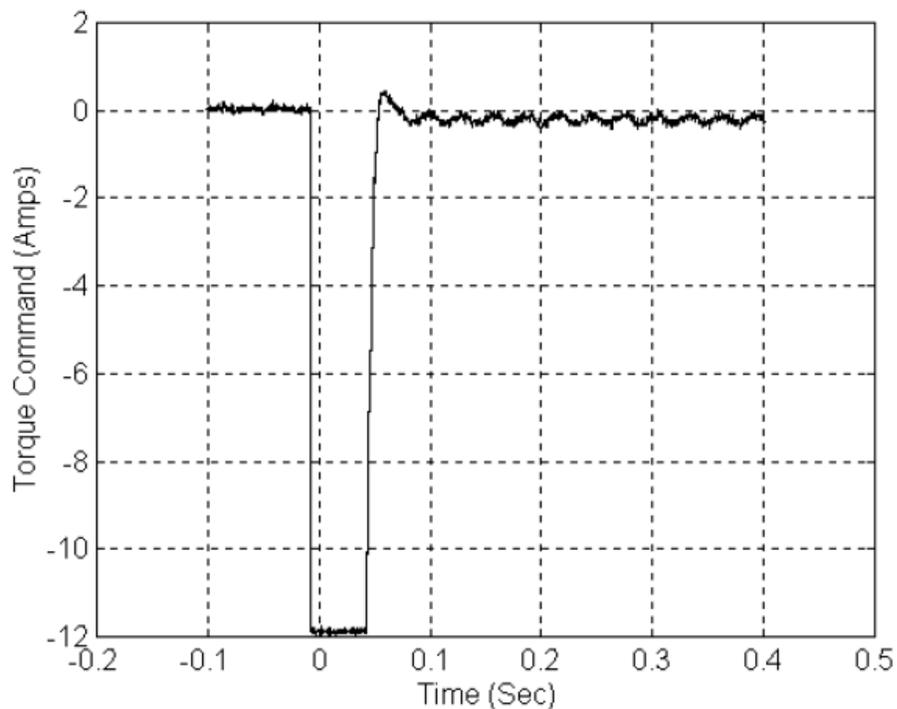


Figure 198: Velocity Loop Step Response Torque Command vs. Time VLG = 48



The response shown in Figures 197 and 198 is good.

Figure 199: Velocity Loop Step Response Velocity vs. Time VLGN = 64

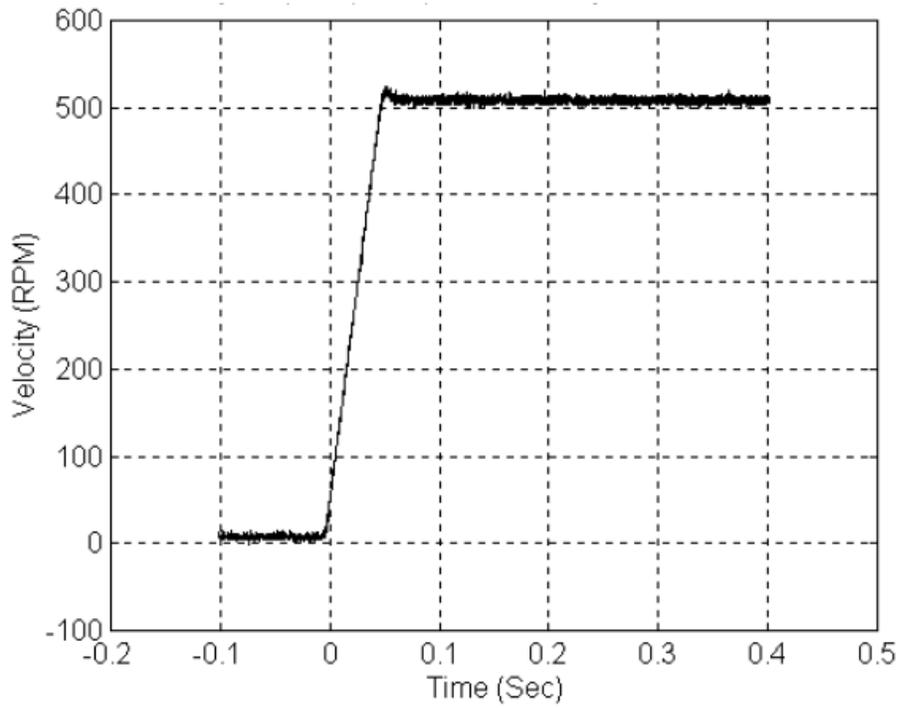
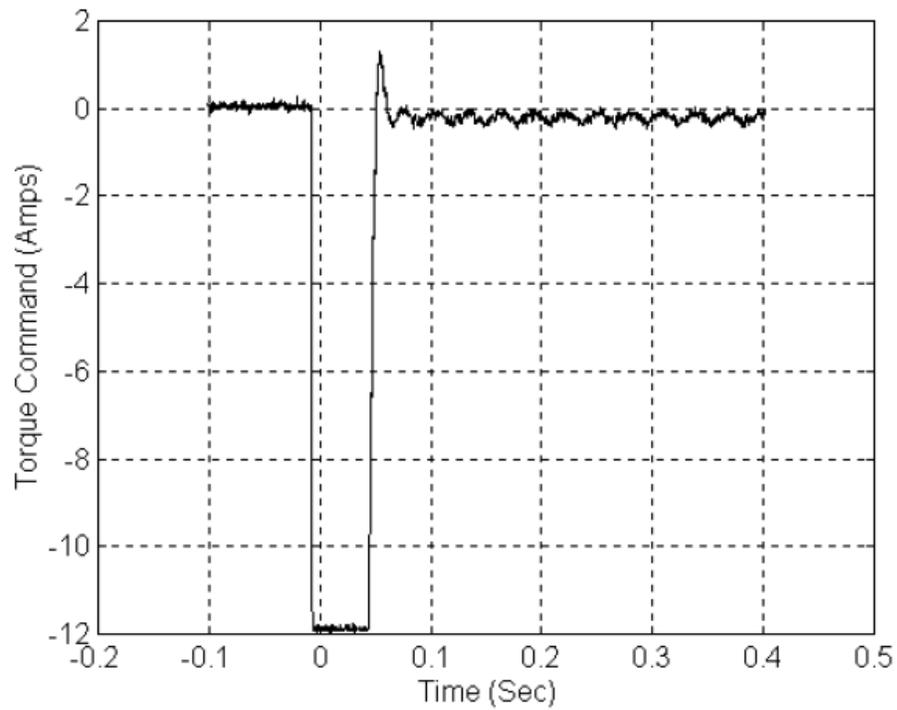


Figure 200: Velocity Loop Step Response Torque Command vs. Time VLGN = 64



The response shown in Figures 199 and 200 is acceptable.

Figure 201: Velocity Loop Step Response Velocity vs. Time VLGn = 208

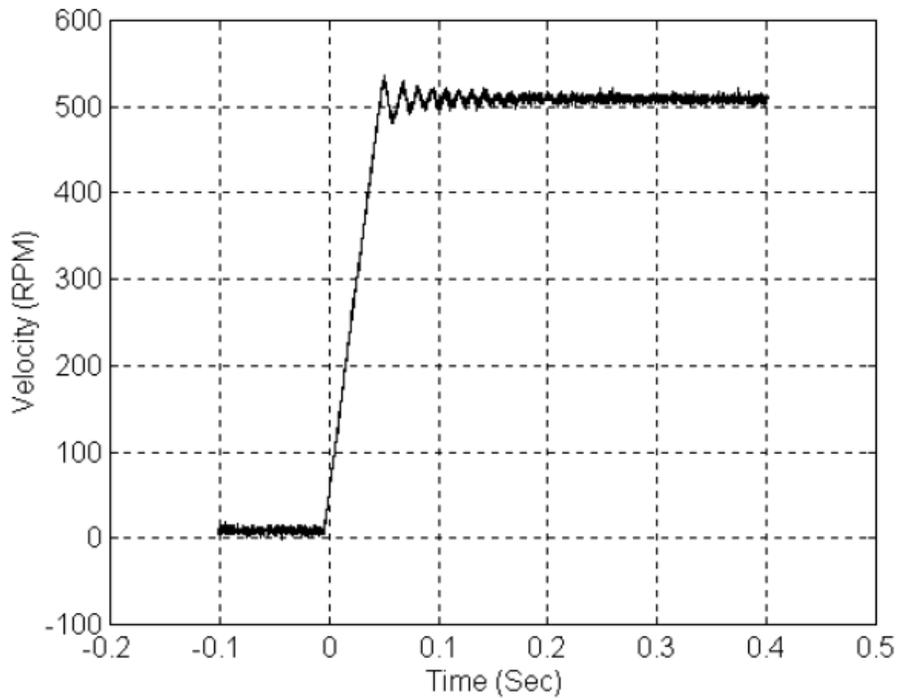
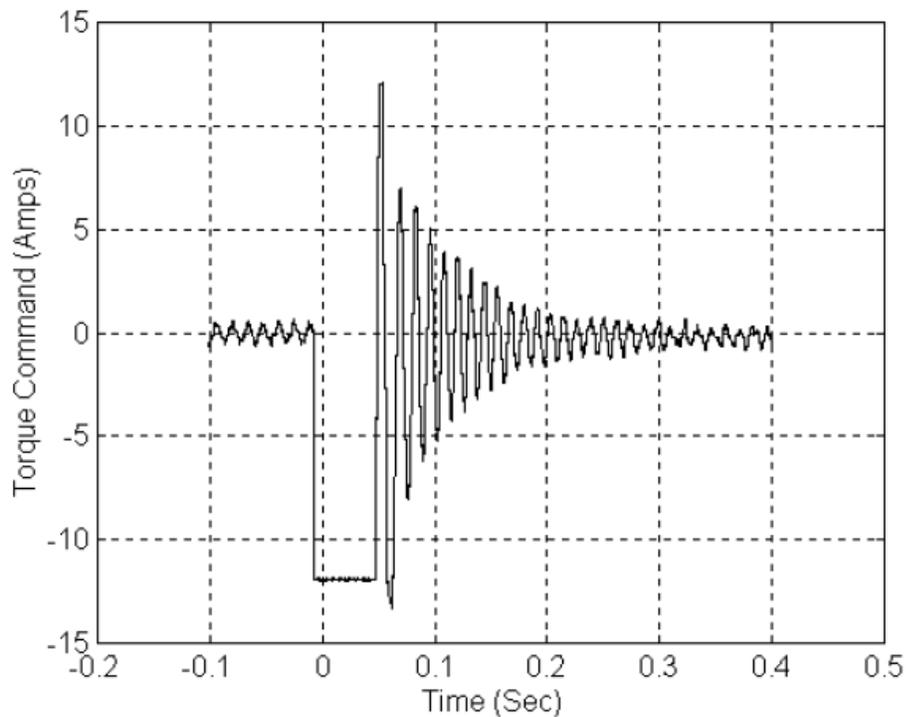


Figure 202: Velocity Loop Step Response Torque Command vs. Time VLGn = 208



The response shown in Figures 201 and 202 is marginally stable and would be unacceptable in many applications. The plots are shown for reference only.

Tuning the Position Loop

The very first step in adjusting the tuning for the position loop is to insure that the velocity loop is stable and has response suitable to the application. Refer to the previous section for methods of setting the velocity loop.

Preliminary Position Loop Settings for Tuning Session.

1. If using Standard Mode control loop settings, set the User Unit and Counts configuration to values appropriate to the mechanical configuration for the axis. See the discussion and examples in Chapter 4 for details.
2. Set the Velocity at 10 Volt value as described in Chapter 4.
3. Set the Integrator Mode selection to “OFF”.
4. Set the Feed Forward % to zero.
5. Set the Position Error Limit to near maximum value. The maximum is 60,000 (User Units / Counts).

Setting the Position Loop Gain

The position control loop is primarily a “PI” (Proportional, Integral) algorithm with optional Feed Forward. Begin tuning the position loop by setting the proportional gain (Pos Loop TC) to provide a stable response with sufficient gain (bandwidth) to meet the motion profile requirements. Setting the Integrator Mode to “OFF” as described in the previous section creates a proportional-only control loop. There are two suggested methods of setting the proportional gain (Pos Loop TC).

Position Loop Proportional Gain Method 1

Calculating the position loop proportional gain assumes that the mechanical design of the machine will have sufficient bandwidth to remain stable and that any resonant frequencies are higher than the bandwidth required by the motion profile.

Terminology

A large mismatch between the load and motor inertia can cause a **RESONANCE** in the system. Resonance is oscillatory behavior caused by mechanical limitations and aggravated by gearing backlash or torsion windup of mechanical members like couplings or shafts. Resonance is eliminated by improving the mechanics, reducing load/motor inertia mismatch or reducing servo gains (reduce performance).

BANDWIDTH is a figure of merit used to compare control system or mechanical performance. As the frequency of command increases, the system response will begin to lag. The bandwidth is defined as the frequency range over which system response (gain) is at least 70% (-3 decibels) of the desired command.

High Bandwidth

- Allows the servo to more accurately reproduce the desired motion
- Allows accurate following of sharp corners in motion paths and high machine cycle rates
- Rejects torque disturbances from mechanics or outside influences improving system accuracy
- Can expose machine resonance, which occur at frequencies near or below the bandwidth

The response of a proportional only system, which is set up by setting Integrator Mode to “OFF”, is an exponential rise. A time constant for an exponential curve represents 68% of the remaining rise. For instance, starting at zero velocity, the response of the position loop to a change in command will require one time constant to reach 68% of the commanded velocity. The second time constant will reduce 68% of the remaining command. Subsequent time constants will reduce 68% of remaining command. For example $100\% - 68\%$ (one time constant) = 32%, $32\%(68\%)=21.8\%$, 68% (first time constant) + 21.8% (second time constant) = 89.8%. Two-time constants eliminate 89.8% of the command necessary. Three-time constants will account for 96.7% of the rise in command. Four-time constants account for 98.9% of the rise. Typically, three time constants are sufficient for most motion applications.

You can use your knowledge of time constants to predict the required system response. For instance, if the fastest acceleration required in the motion profiles must occur within 200 mSec, the 200 mSec response to the change in command will be 98.9% complete in three-time constants. Dividing the 200 mSec by 3 results in a time constant of about 67 mSec. **The Pos Loop TC configuration field represents one time constant in mSec.** In the example above one time constant is 67msec.

Position Loop Proportional Gain Method 2

Similar to the Velocity loop tuning method above. Use an oscilloscope and gradually lower the Pos Loop TC value (increasing gain). Monitor the Motor Velocity analog output for performance characteristics are appropriate.

D-2 Start-Up and Tuning Information for Analog Servo Systems

There are two major sections covered;

- Validating Home Switch, Over Travel Inputs, and Motor direction.
- Velocity at Max Cmd, Position Loop Time Constant, and Velocity Feedforward determination

D-2.1 Analog Mode Velocity Interface System Startup Procedures

Startup Procedures

1. Connect the motor to the analog velocity interface servo amplifier according to the manufacturer's recommendations.
2. Connect the DSM314 **Drive Enable** Relay and **Velocity Command** outputs to the servo amplifier. Connect the position feedback device (Incremental Quadrature Encoder) to the Motion Mate DSM314 encoder inputs.

Note: *If these connections are incorrect or there is slippage in the coupling to the Feedback Device, an Out of Sync error condition can occur when motion is commanded.*

3. Connect the servo amplifier Ready output (if available) to the DSM314 Drive Ready input (IN_4). This signal must switch to 0v when the amplifier is ready to control the servo. **The DSM starts checking the Drive Ready input one second after the Drive Enable relay turns on in response to the Enable Drive %Q bit.** If the servo amplifier does not provide a suitable Ready output, this input to the DSM314 must be connected to 0v or the Drive Ready input can be disabled in the module configuration. If a **Home** switch is used (24 Vdc), wire it to the correct DSM314 input. The Home switch must be wired so that it is ALWAYS ON when the axis is on the negative side of home and ALWAYS OFF when the axis is on the positive side of home.
4. Use the configuration software to set the desired configurable parameters. Store the configuration to the host controller.
5. Turn on the %Q Enable Drive bit and place the command code for Force D/A Output equal to 0 in the %AQ table. Confirm that the servo amplifier is enabled (the motor should exhibit holding torque). If the motor moves, adjust the amplifier command offset adjustment until the motor stops moving. Note: The %Q Enable Drive bit must be maintained ON in order for the Force D/A Output command to function.
6. Send the command code for Force D/A Output equal to +3200 (+1.0v). Confirm that the motor moves in the desired POSITIVE direction (based on the **Axis Direction** configuration parameter setting) and the Actual Velocity reported in the DSM314 %AI table is POSITIVE. If the motor moves in the wrong direction, consult the servo amplifier manufacturer's instructions for corrective action. The **Axis Direction** parameter in the Configuration Software can also be used to swap the positive and

negative axis directions. If the motor moves in the POSITIVE direction but the DSM314 reports that Actual Velocity is NEGATIVE, then the encoder channel A and channel B inputs must be swapped.

7. Record the actual motor velocity reported by the Motion Mate DSM314 with a 1.0 volt velocity command. Multiply this velocity by 10 and update the **Velocity at Max Cmd** entry in the DSM314 configuration, if necessary. Initially set the **Pos Loop Time Constant (0.1 ms)** configuration parameter to a high value (typically 100 ms or a value of 1000 in the configuration).
8. Turn on the %Q Jog Plus bit. Confirm that the servo moves in the proper direction and that the Actual Velocity reported by the Motion Mate DSM314 in the %AI table matches the configured **Jog Velocity**. If Motion Programs will use an acceleration higher than **the Jog Acceleration**, it may be necessary to increase **Jog Acceleration** so that Abort All Moves and Normal Stop actions will operate as expected.
9. With the Drive Enabled %Q bit ON and no servo motion commanded, adjust the servo drive command offset adjustment for zero Position Error. The integrator should be OFF during this process.
10. Check for proper operation of the Find Home cycle by momentarily turning on the %Q Find Home bit (the Drive Enabled %Q bit must also be maintained ON). The axis should move towards the Home Switch at the configured **Find Home Velocity**, then seek the Encoder Marker at the configured **Final Home Velocity**. If necessary, adjust the configured velocities and the location of the **Home Switch** for consistent operation. The final **Home Switch** transition MUST occur at least 10 ms before the Encoder Marker Pulse is encountered. The physical location of **Home Position** can then be adjusted by changing the **Home Offset** value in the Configuration Software.
11. Monitor servo performance and use the %Q Jog Plus and Jog Minus bits to move the analog servo motor in each direction. The **Position Loop Time Constant** can be temporarily modified by placing the correct command code in the %AQ table. For most systems **the Position Loop Time Constant** can be reduced until some servo instability is noted, then increased to a value approximately 50% higher. Once the correct time constant is determined, the DSM314 configuration should be updated using the Configuration Software. **Velocity Feedforward** can also be set to a non-zero value (typically 90-100 %) for optimum servo response.

Note: *For proper servo operation, the Configuration entry for Velocity at Max Cmd MUST be set to the actual servo velocity (in User Units/sec) caused by a 10 Volt Velocity command to the amplifier.*

D-2.2 Analog Mode Torque Interface System Startup Procedures

Startup Procedures

1. Connect the motor to the analog torque interface servo amplifier according to the manufacturer's recommendations.

Note: *The amplifier must be configured to accept voltage (+-10 volt) that corresponds to motor torque.*

2. Connect the DSM314 **Drive Enable** Relay and **Torque Command** outputs to the servo amplifier. Connect the position feedback device (Incremental Quadrature Encoder) to the Motion Mate DSM314 encoder inputs.

Note: *If these connections are incorrect or there is slippage in the coupling to the Feedback Device, an Out of Sync error condition can occur when motion is commanded.*

3. Connect the servo amplifier Ready output (if available) to the DSM314 Drive Ready input (IN_4). This signal must switch to 0v when the amplifier is ready to control the servo. **The DSM starts checking the Drive Ready input one second after the Drive Enable relay turns on in response to the Enable Drive %Q bit.** If the servo amplifier does not provide a suitable Ready output, this input to the DSM314 must be connected to 0v or the Drive Ready input can be disabled in the module configuration. If a Home switch is used (24 Vdc), wire it to the correct DSM314 input. The **Home** switch must be wired so that it is ALWAYS ON when the axis is on the negative side of home and ALWAYS OFF when the axis is on the positive side of home.
4. Use the configuration software to set the desired configurable parameters. Store the configuration to the host controller. Specific parameters that the user will need to reference are as follows:

Analog Servo Command -configuration must be set to Torque. This is not the default value. This configuration parameter configures the module to produce a torque command on the analog output.

Note: *DSM firmware revision 3.0 or later is required for Analog Torque mode to function.*

Velocity at Max Command - The configuration setting velocity at maximum command determine the maximum velocity the servo will be commanded to run. In the early tuning stages it is advisable to set this value relatively low. This will allow the system to be brought up in stages. Once basic operation and tuning has been verified, the maximum value can be raised to the value that is determined by either the process limitations or servo amplifier/motor set

Torque Limit - The torque limit value determines the maximum analog torque command that will be sent to the servo amplifier. In the early tuning stages it is advisable to set this value relatively low. This torque limit is set using %AQ command. Refer to Chapter 5 for information on this command. Once basic operation is verified, the torque limit value can then be set to the value desired for the application.

Advanced Tuning Parameters: The advanced tuning parameter section contains many parameters that are used to configure torque mode to operate correctly. The advanced tuning parameters are discussed in detail in Chapter 4. For a complete reference consult this chapter. The tuning parameters of interest are as follows:

Tuning Parameter 6: Sets the encoder resolution parameter. The parameter is only used in torque mode. For correct torque mode operation, this value must be set to the number of quadrature encoder counts generated by the motor feedback device per revolution. The user can determine the value from the feedback device specification. As a double check, the user may wish to connect the feedback device to the DSM and manually rotate the motor shaft one revolution. The reading on the DSM %AI data for actual position should closely match (variations are caused by the accuracy of manual turning shaft one revolution) the value placed in this parameter. The allowed range is 100-32767 counts/revolution. The default value is 4096 counts per revolution

Tuning Parameter 7: Sets the velocity regulator proportional gain. The parameter is only used in torque mode. The proportional gain is multiplied by velocity error (velocity command - velocity feedback) to generate the portion of the torque command due to the proportional term. Correctly setting this value will determine how well the velocity regulator performs in the control system. The following sections will discuss how to set this value.

Tuning Parameter 8: Sets the velocity regulator integral gain. The parameter is only used in torque mode. The integral gain is the term multiplied by the area of the velocity error (velocity command - velocity feedback) to generate the portion of the torque command due to the integral term. Correctly setting this value will determine how well the velocity regulator performs in the control system. The following sections will discuss how to set this value.

Note: *For proper servo operation, the Configuration entry for Encoder Resolution MUST be set to the correct value for the servo amplifier/motor set. If this value is not set correctly instabilities can result.*

5. Turn on the %Q Enable Drive bit and place the command code for Force Servo Velocity equal to 0 in the %AQ table. Confirm that the servo amplifier is enabled (the motor should exhibit holding torque). If the motor moves, adjust the amplifier until the motor stops moving.
6. Make sure that the motor shaft is not connected to the load when first performing the following operation. The user needs to now verify basic control functionality. Send the command code for Force Servo Velocity equal to 10 RPM. Confirm that the motor moves in the desired POSITIVE direction (based on the **Axis Direction** configuration parameter setting) and the Actual Velocity reported in the DSM314 %AI table is POSITIVE. If the motor moves in the wrong direction, consult the servo amplifier manufacturer's instructions for corrective action. The **Axis Direction** parameter in the Configuration Software can also be used to swap the positive and negative axis directions. If the motor moves in the POSITIVE direction but the DSM314 reports that Actual Velocity is NEGATIVE, then the encoder channel A and channel B inputs must be swapped.

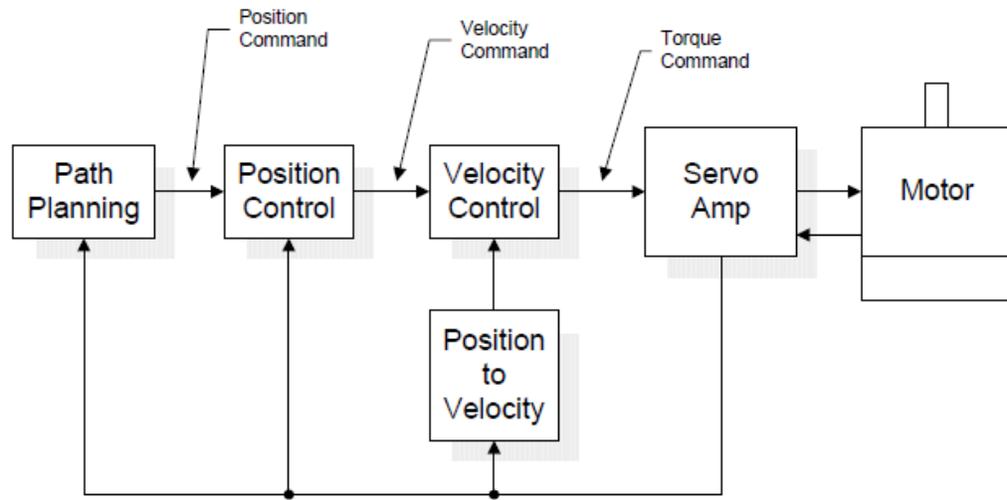
7. With the Drive Enabled %Q bit ON and no servo motion commanded, adjust the servo drive so no motion is generated. The velocity loop integral term **MUST** be set to 0 to properly complete this step.
8. Once correct basic operation has been achieved, the velocity loop requires tuning. The section "Tuning the Torque Mode Velocity Loop" contains a basic procedure for tuning the loop. **NOTE:** The tuning procedure for Torque Mode velocity regulators is **DIFFERENT** from Digital Mode Velocity regulators. The user should **NOT** proceed to tuning the Position Loop until the velocity loop tuning is complete.
9. Once the velocity regulators have been tuned, the position loop tuning and setup can be completed. Initially set the **Pos Loop Time Constant (0.1 ms)** configuration parameter to a high value (typically 100 ms or a value of 1000 in the configuration).
10. Turn on the %Q Jog Plus bit. Confirm that the servo moves in the proper direction and that the Actual Velocity reported by the Motion Mate DSM314 in the %AI table matches the configured **Jog Velocity**. If Motion Programs will use acceleration higher than **the Jog Acceleration**, it may be necessary to increase **Jog Acceleration** so that Abort All Moves and Normal Stop actions will operate as expected.
11. Check for proper operation of the Find Home cycle by momentarily turning on the %Q Find Home bit (the Drive Enabled %Q bit must also be maintained ON). The axis should move towards the Home Switch at the configured **Find Home Velocity**, then seek the Encoder Marker at the configured **Final Home Velocity**. If necessary, adjust the configured velocities and the location of the **Home Switch** for consistent operation. The final **Home Switch** transition **MUST** occur at least 10 ms before the Encoder Marker Pulse is encountered. The physical location of **Home Position** can then be adjusted by changing the **Home Offset** value in the Configuration Software.
12. Monitor servo performance and use the %Q Jog Plus and Jog Minus bits to move the analog servo motor in each direction. The **Position Loop Time Constant** can be temporarily modified by placing the correct command code in the %AQ table. For most systems **the Position Loop Time Constant** can be reduced until some servo instability is noted, then increased to a value approximately 50% higher. Once the correct time constant is determined, the DSM314 configuration should be updated using the Configuration Software. **Velocity Feedforward** can also be set to a non-zero value (typically 90-100 %) for optimum servo response.

Note: For proper servo operation, the Configuration entry for Velocity at Max Cmd **MUST** be set to maximum servo velocity (in User Units/sec) that the system or process allows.

Tuning the Torque Mode Velocity Loop

The proper method to tune the velocity loop is to separate the velocity loop from the position loop. To achieve this separation, a method must be used to directly send velocity commands without using the position loop control. The DSM module has several modes that will allow the user to send a velocity command directly to the velocity loop. Two methods are as follows:

Figure 203: Analog Mode Torque Interface Control Loops Block Diagram



Method #1:

The Force Servo Velocity %AQ immediate command (34h) will send a velocity command directly to the velocity loop. This command is different from the Move at Velocity Command, which uses the position loop to generate the command. This is important since the position loop should not be interacting with the velocity loop at this point in the tuning process. The Force Servo Velocity %AQ command allows the user to generate a step change in the velocity. The velocity command step is then used to generate the velocity loop step response. The user should note that when a velocity command step change is performed the acceleration is limited only by the bandwidth of the velocity loop. In some applications this can cause damage to the controlled device due to the high acceleration rate.

Method #2:

In some applications, method #1 introduces too large a shock to the device under control. In these cases, another method to generate a velocity command is needed. The method requires that the user set the position loop to an open loop configuration. The position loop is set to open loop by setting the **Position Loop Time Constant** to zero and the **Velocity Feedforward Gain** to 100 percent. You can then use the Move at Velocity Command or a motion program to generate velocity commands to the servo drive.

1. The following procedure tunes the velocity regulator. It is suggested that initially, this be done with the motor NOT connected to the driven load. The tuning associated with the load will be performed in a later step. The first parameter that needs to be adjusted is the **Velocity Loop Proportional Gain**. The velocity loop proportional gain is multiplied by velocity error (velocity command - velocity feedback) to generate the portion of the torque command due to the proportional term. The proportional term should be set low to begin the process. Depending on the bandwidth of the controlled servo amplifier, the default value of 1500 may represent a good starting point. However, if the servo amplifier has a low bandwidth or is very sensitive to changes in the torque command the initial value may need to be set lower. The tuning procedure will allow the user to iterate to get the final value. Thus, if there is any concern start with a very low value (100 for example)
2. Choose the method to introduce velocity command to the velocity loop. Method #1 and Method #2 (above) are examples of methods to perform this task.
3. Connect an oscilloscope to the analog outputs for Motor Velocity from the servo amplifier.
4. Per the earlier discussion, set the initial velocity loop proportional gain value.
5. Generate a velocity command step change. At this point the step change should be relatively small compared to the full speed of the machine. Ten to 20 % of the rated machine speed is a good start.
6. Observe the Motor Velocity on the oscilloscope. The objective is to obtain a critically damped velocity loop response. There will most likely be a steady state error in the velocity at this point. This is expected at this point in the tuning process. The velocity integral term will be introduced in steps that follow to cancel this error. Pay particular attention to the 1st peak that occurs and any oscillations that are occurring in the velocity signal.
7. Increase the **Velocity Loop Proportional Gain** in small steps and repeat 5 and 6 until the desired response is achieved. Depending on the application this may be a critically damped system or may have a slight overshoot. As a general rule, the lower the **Velocity Loop Proportional Gain** value that meets the system requirements the more robust the control. The user should carefully observe the velocity feedback signal. In some applications, running the **Velocity Loop Gain** high enough to create instability can cause machine damage. If oscillations are observed in the Motor Velocity feedback signal prior to this point, decrease the **Velocity Loop Proportional Gain**.

8. The next parameter to be adjusted is the **Velocity Loop Integral Gain**. The Velocity Loop integral gain is the term multiplied by the area of the velocity error (velocity command - velocity feedback) to generate the portion of the torque command due to the integral term. The integral gain term is typically used to compensate for steady state error in velocity. To begin the tuning process the **Velocity Loop Integral Gain** should be set to zero. The tuning procedure will be to slowly increase this value until steady state error is eliminated without incurring large overshoot or excessive ringing in the response.
9. Choose the method to introduce velocity command to the velocity loop. Method #1 and Method #2 (above) are examples of methods to perform this task.
10. Connect an oscilloscope to the analog outputs for Motor Velocity from the servo amplifier.
11. Per the earlier discussion, set the initial **Velocity Loop Integral Gain** value.
12. Generate a velocity command step change. At this point the step change should be relatively small compared to the full speed of the machine. Ten to 20 % of the rated machine speed is a good start.
13. Observe the Motor Velocity on the oscilloscope. The objective is to eliminate steady state error without introducing excessive overshoot or ringing. While tuning the integral term pay particular attention to any oscillations that occur in the response. Excessive oscillations are an indication of instability in the control loop due to excessive integral gain.
14. Increase the **Velocity Loop Integral Gain** in small steps and repeat 12 and 13 until the desired response is achieved. Depending on the application this may be a critically damped system or may have a slight overshoot. As a general rule, the lower the **Velocity Loop Integral Gain** value that meets the system requirements the more robust the control. The user should carefully observe the velocity feedback signal. In some applications, running the **Velocity Loop Integral Gain** high enough to create instability can cause machine damage. If oscillations are observed in the Motor Velocity feedback signal prior to this point, decrease the **Velocity Loop Integral Gain**. The basic velocity loop is tuned at this point. The next step will be to connect the motor to the load and adjust the **Velocity Loop Gain** parameter to adjust for the motor load.
15. With the base Velocity Loop tuned, connect the motor to the load. The **Velocity Loop Gain** parameter adjusts the velocity loop response to compensate for the load. Specifically, the **Velocity Loop Gain** parameter adjusts the velocity loop bandwidth. As a starting point use the following formula shown below.

Equation 2

$$\text{Velocity Loop Gain} = \frac{J_l}{J_m} 16$$

Where:

J_l Motor Inertia

J_m Load Inertia

The Velocity Loop Gain calculated above in many cases will not need to be altered. However, due to the application (for example, machine resonance) the value may need to be adjusted. To tune the Velocity Loop Gain the following procedure can be used:

16. Choose the method to introduce velocity command to the velocity loop. Method #1 and Method #2 (above) are examples of methods to perform this task.
17. Connect an oscilloscope to the analog outputs from the Servo Amplifier for Motor Feedback Velocity.
18. Set the Velocity Loop Gain to zero. This is a conservative approach. If the application is known to not have resonant frequencies from zero to approximately 250 Hz, you can start with a higher value, but do not exceed the value calculated in equation 2 at this point.
19. Generate a velocity command step change. At this point the step change should be relatively small compared to the full speed of the machine. Ten to 20 % of the rated machine speed is a good start.
20. Observe the Motor Velocity on the oscilloscope. The objective is to obtain a critically damped velocity loop response. Pay particular attention to any oscillations that are occurring in the velocity feedback signal.
21. Increase the **Velocity Loop Gain** in small steps and repeat 19 and 20 until instability in the Motor Velocity feedback signal is observed.

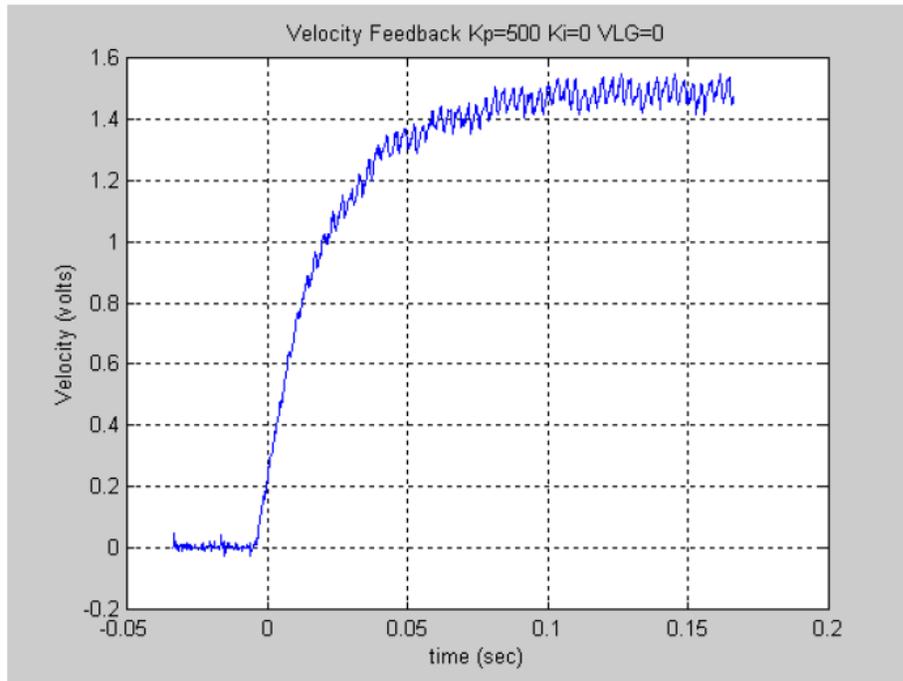
Note: Care should be taken in this step that an instability does not cause damage to the machine. Once this point is reached, decrease the Velocity Loop Gain by at least 15 %. As a general rule, the lower the Velocity Loop Gain value that meets the system requirements the more robust the control. You should carefully observe the velocity feedback signal. In some applications, running the Velocity Loop Gain high enough to create instability can cause machine damage. If in doubt, adjust the Velocity Loop Gain to be no greater than the value calculated in equation 1. If oscillations are observed in the Motor Velocity feedback signal prior to this point, decrease the Velocity Loop Gain and continue with step 22 below.

22. The velocity loop is tuned at this point. However, the robustness of the loop must be checked. To perform this test, introduce velocity command steps in increments of 20% Rated Machine Speed, 40% Rated Machine Speed, 60% Rated Machine Speed, 80% Machine Rated Speed, and 100% Rated Machine Speed. Observe the Motor Velocity and Torque Command signals for any instability. If an instability or resonance is observed, reduce the **Velocity Loop Gain** and repeat the test.

Sample Velocity Loop Tuning Session

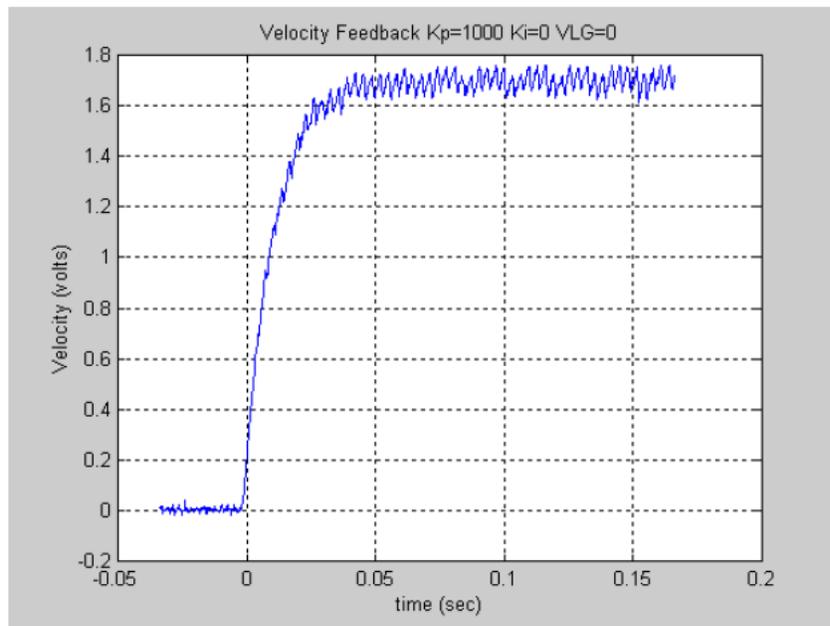
A sample velocity loop tuning session is shown in the plots that follow. To begin the process the Velocity Loop Proportional Gain is tuned

Figure 204: Velocity Loop Step Response Velocity Feedback vs. Time $K_p=500$ $K_i=0$ $V_LG=0$



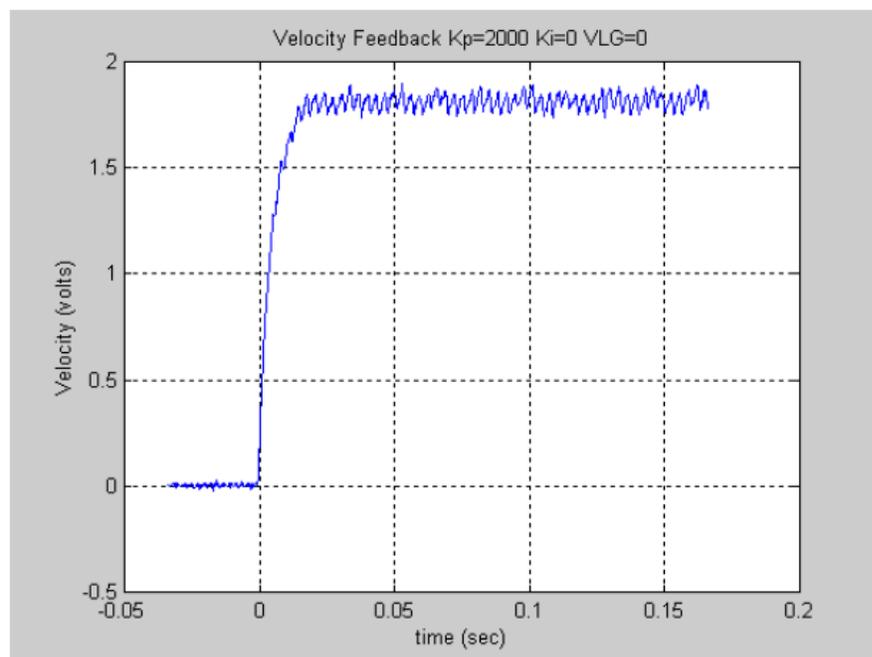
Note the system has a relatively slow response. Also based the desired velocity, there is a steady state error. In this case, the Velocity Loop Proportional Gain can be increased to help generate a faster response.

Figure 205: Velocity Loop Step Response Velocity Feedback vs. Time Kp=1000 Ki=0 VLG=0



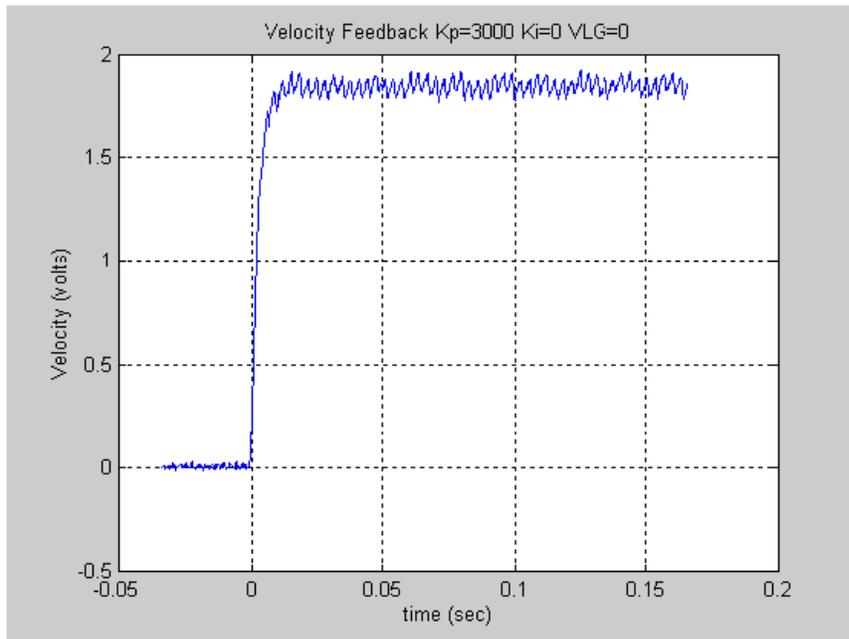
The **Velocity Loop Proportional Gain** has been increased in the figure above. The rise time has been decreased. However, the system can still be enhanced by adding additional **Velocity Loop Proportional Gain**. The steady state error is still present.³

Figure 206: Velocity Loop Step Response Velocity Feedback vs. Time Kp=2000 Ki=0 VLG=0



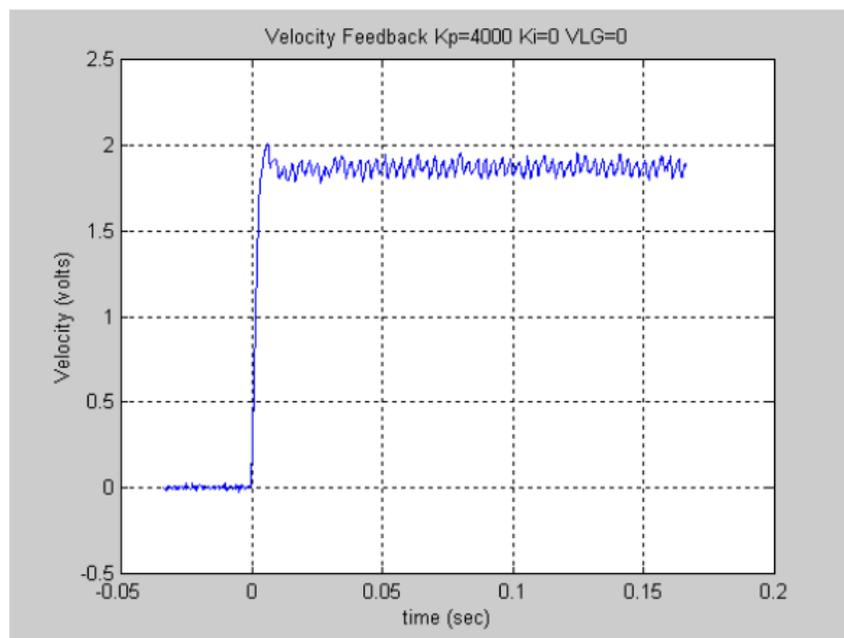
The Velocity Loop Proportional Gain has been increase again. The response shown is starting to look very acceptable. However, the rise time can be improved further.

Figure 207: Velocity Loop Step Response Velocity Feedback vs. Time $K_p=3000$ $K_i=0$ $V_LGN = 0$



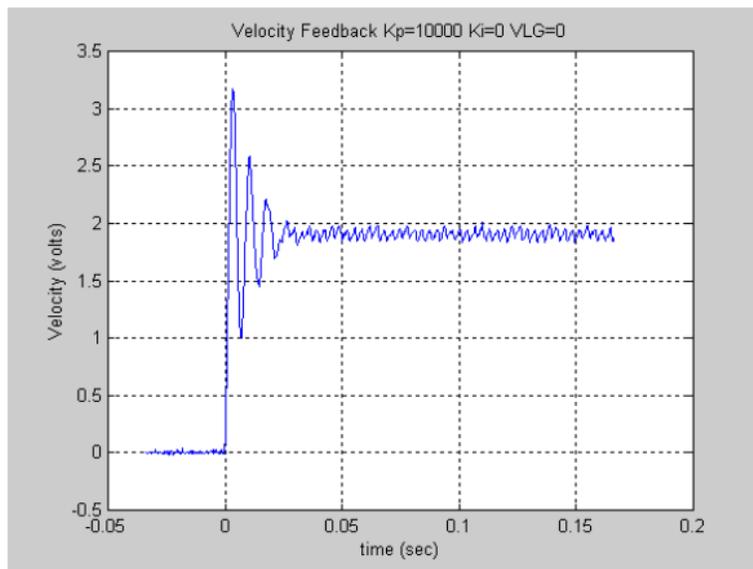
The response shown above in is looking very good. Note the slight peak in the response. To experiment with the response, the Velocity Loop Proportional Gain will be increased more.

Figure 208: Velocity Loop Step Response Velocity Feedback vs. Time $K_p=4000$ $K_i=0$ $V_LGN = 0$



The response shown in the figure has a slight overshoot. This or the previous response would be very acceptable in many applications. However, the tuning should be determined based upon the machine abilities. The plots are shown for reference only.

Figure 209: Velocity Loop Step Response Velocity Feedback vs. Time Kp=10000 Ki=0 VLG=0

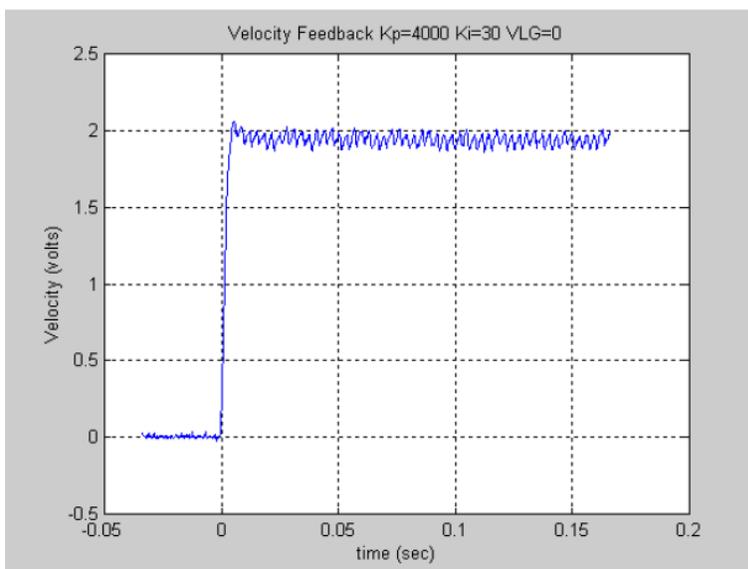


The plot shown above represents an unacceptable response. The loop is exhibiting signs of instability. Not the Overshoot and ringing following the first peak. The **Velocity Loop Proportional Gain** should be significantly decreased to achieve a more stable response.

For this exercise, the response shown corresponding to the **Velocity Loop Proportional Gain** (Kp) =4000 will be chosen as the desired response for the system. This value will be used when tuning the **Velocity Loop Integral Gain**.

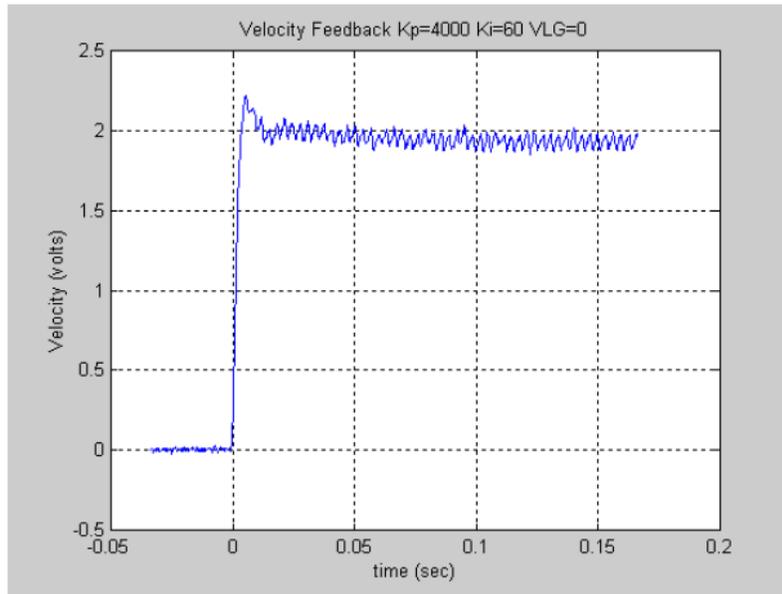
The **Velocity Loop Integral Gain** is initially zero. You can make a small change to the value and observe the response.

Figure 210: Velocity Loop Step Response Velocity Feedback vs. Time Kp=4000 Ki=30 VLG=0



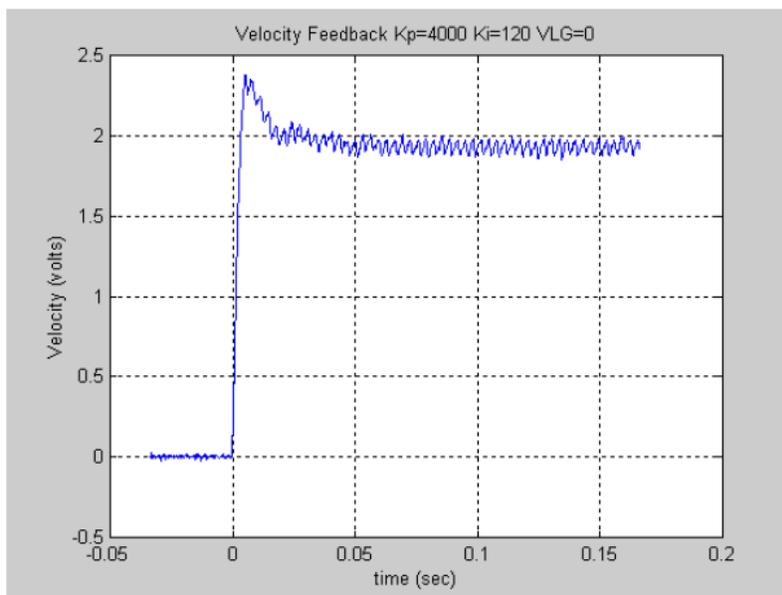
The response above indicates that the **Velocity Loop Integral Gain** has resulted in a more desirable response. Specifically, the steady state error is being reduced.

Figure 211: Velocity Loop Step Response Velocity Feedback vs. Time Kp=4000 Ki=60 VLG=0



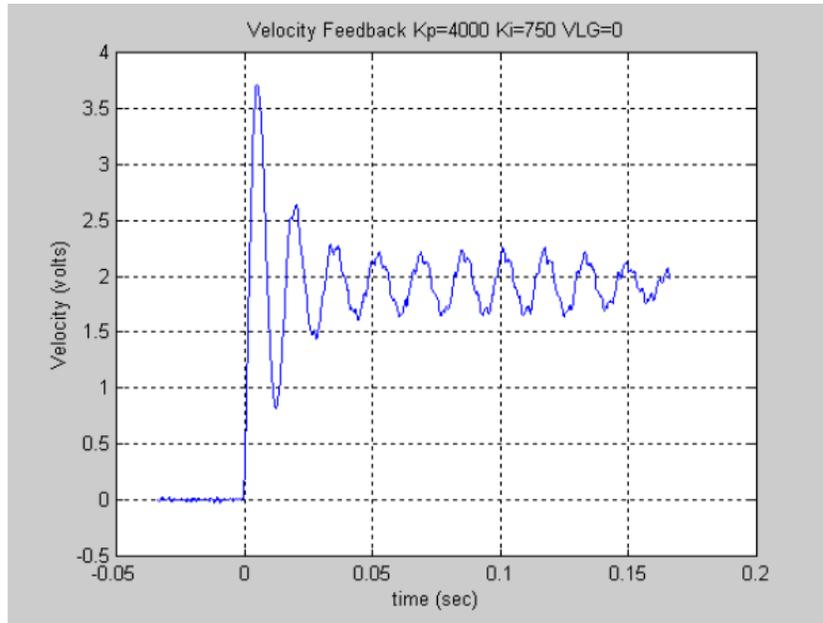
When you increase the **Velocity Loop Integral Gain** further, you can see the beginning of an overshoot due to the integral gain. However, the responses in the previous two figures are both acceptable. The final values chosen to depend on the capabilities of the driven load. In general, the lower the **Velocity Loop Proportional Gain** and **Velocity Loop Integral Gain** that meet the system requirements the more robust the control.

Figure 212: Velocity Loop Step Response Velocity Feedback vs. Time Kp=4000 Ki=120 VLG=0



The response shown above illustrates too much **Velocity Loop Integral Gain** and in most applications this would be considered unacceptable.

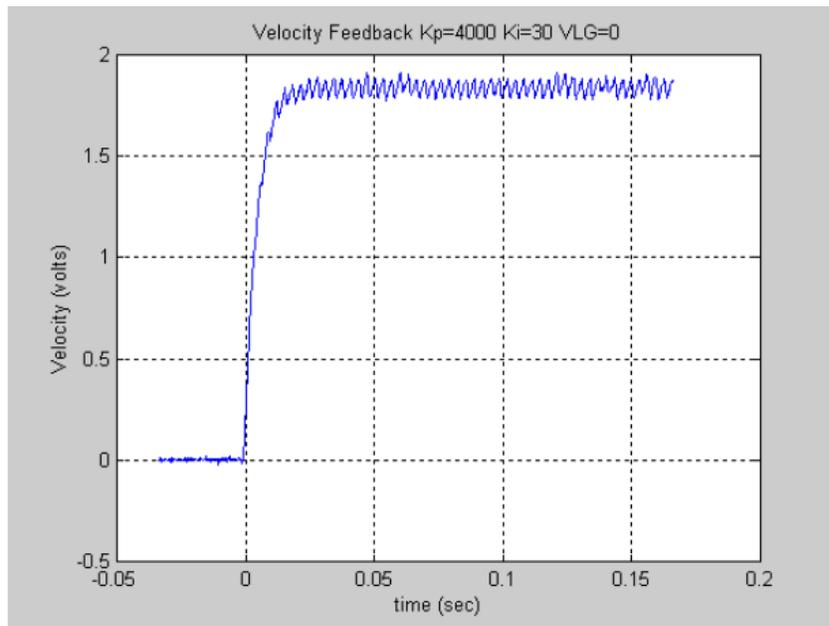
**Figure 213: Velocity Loop Step Response Velocity Feedback vs. Time Kp=4000
Ki=7500 VLG= 0**



The result shown above represents a marginally stable system. In this response, there is not only a significant overshoot, but also a ringing in the velocity response that is slowly being damped out. The response is unacceptable.

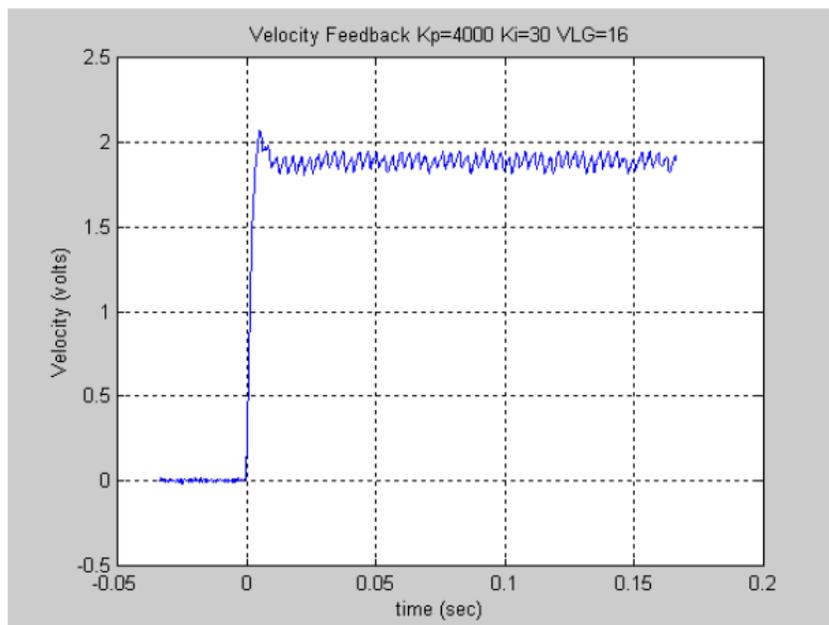
The next step in the tuning process is to connect the motor to the load and then adjust the control to achieve the desired performance. The Velocity Loop Gain parameter allows the user to adjust the controller parameters to account for the motor load. As in the procedure above, start with the **Velocity Loop Gain** equal to zero.

Figure 214: Velocity Loop Step Response Velocity Feedback vs. Time $K_p=4000$ $K_i=30$ $VLGN = 0$



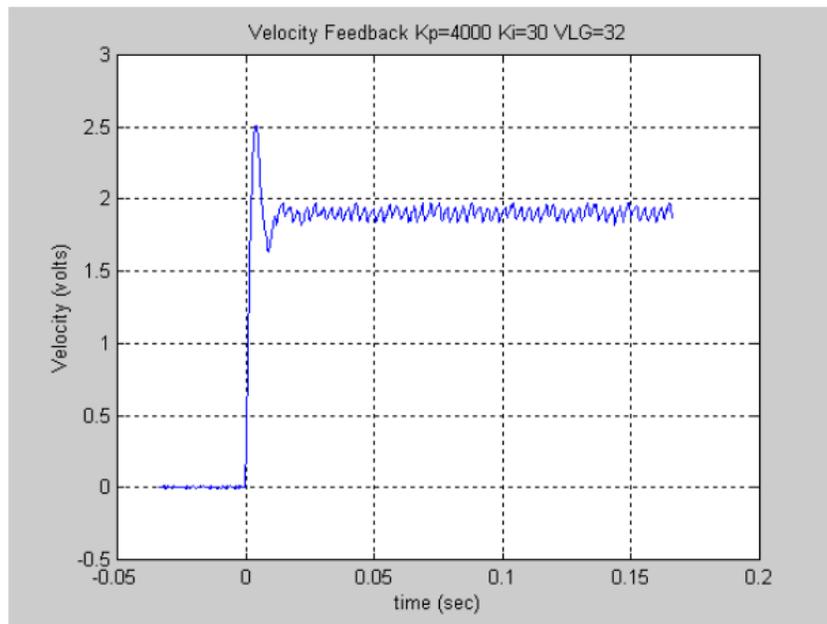
The figure above shows the motor velocity response with a load connected to the motor and the motor tuned per the exercise above. The performance is acceptable, but by increasing the Velocity Loop Gain the rise time can be decreased.

Figure 215: Velocity Loop Step Response Velocity Feedback vs. Time $K_p=4000$ $K_i=30$ $VLGN = 16$



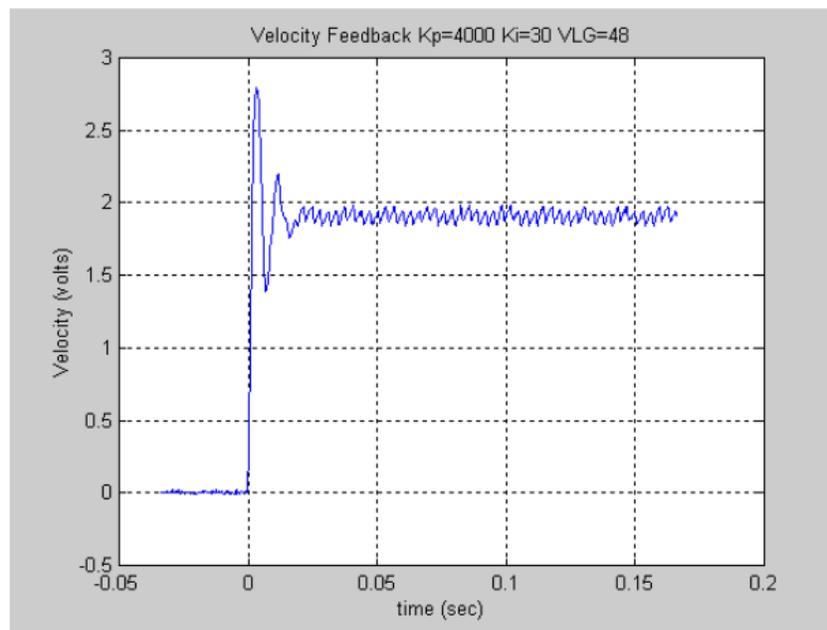
The response shown above is acceptable. The response has a slight overshoot but no sustained oscillation or ringing.

Figure 216: Velocity Loop Step Response Velocity Feedback vs. Time Kp=4000 Ki=30 VLG=32



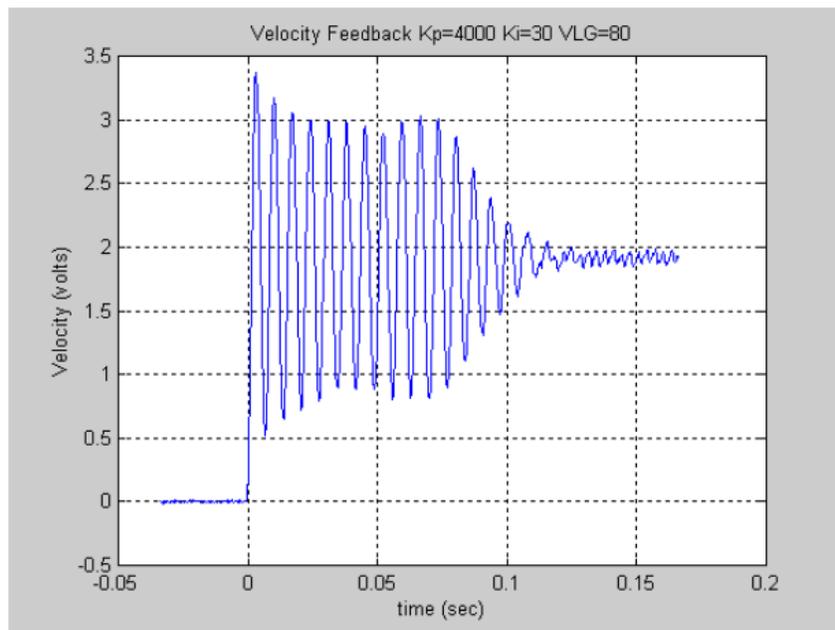
The response shown above has a rather large overshoot, however there are no adverse effect beyond the initial overshoot and oscillation. The overshoot indicates that the user may wish to reduce the Velocity Loop Gain.

Figure 217: Velocity Loop Step Response Velocity Feedback vs. Time Kp=4000 Ki=30 VLG=48



The response shown above exhibits an overshoot and notable ringing in the response. This response is starting to indicate that the velocity loop gain is greater than necessary.

Figure 218: Velocity Loop Step Response Velocity Feedback vs. Time $K_p=4000$ $K_i=30$ $VLGN = 80$



The response shown above represents a marginally stable system. The Velocity Loop Gain is significantly too large. Notice the significant overshoot and sustained ringing in the response. This response would not be acceptable.

D-3 System Troubleshooting Hints (Analog Mode)

1. The DSM314 requires a Series 90-30 CPU with firmware version 10.0 or later, or a PACSystems RX3i CPU with version 2.8 or later.
2. The DSM Torque Mode function requires DSM firmware version 3.0 or higher.
3. If the Drive Ready input is enabled in the module configuration, the input must be connected to 0v within 1 second after the Drive Enable relay turns on or the Motion Mate DSM314 will not operate. Incorrect Drive Ready configuration or wiring will cause Error Code C0h to be reported in the Axis Error Code %AI data.
4. The ENABLE DRIVE %Q control bit must be set continuously to 1 or no motion other than Jog moves will be allowed. If no STOP errors (see Appendix A for error codes) have occurred, the DRIVE ENABLED %I status bit will mirror the state of the ENABLE DRIVE %Q bit. A STOP error will turn off the DRIVE ENABLED output bit even though ENABLE DRIVE input bit is still a 1. The error condition must be corrected and the CLEAR ERROR %Q control bit turned on for one host controller sweep to re-enable the drive.
5. If the ERROR %I status bit is 1 and the AXIS ENABLED and DRIVE ENABLED %I status bits are 0, then a STOP error has occurred (Status LED flashing fast). In this state, the DSM314 will not respond to any commands other than the CLEAR ERROR %Q control bit.

6. The CLEAR ERROR %Q control bit uses one-shot action. Each time an error is generated, the bit must be set to 0 then set to 1 for at least one sweep to clear the error.
7. The CFG OK LED must be ON or the DSM314 will not respond to host controller commands. If the LED is OFF then a valid DSM314 configuration has not been received from the host controller, or there may be a recognized configuration error. Check the %AI error code words for Dxxx errors, which are documented in the “System Error Codes” section of Appendix A. Also check the PLC fault tables for reported configuration errors.
8. Host controller logic should not send the following %Q bit commands to the DSM314 on the first sweep: Find Home, Execute Motion Program, Execute Local Logic. If these commands are sent on the first sweep, an error will be reported, and the action will not be performed.
9. Host controller logic should not send the following %AQ commands to the DSM314 on the first sweep: Move at Velocity, Move Command. If these commands are sent on the first sweep, an error will be reported, and the action will not be performed.

Appendix E: Local Logic Execution Time

This appendix contains information necessary to determine a local logic program's execution time.

E-1 Local Logic Execution Timing Data

Local Logic program in the DSM is constrained to complete execution within **300 Microseconds**. Exceeding the execution time limit will result in a watchdog timeout and an error being reported. The watchdog timeout error will stop axes motion and Local Logic execution. The timing data supplied in the tables below allows the programmer to compute the worst-case execution time for a program. Note that the data below represents **execution** times, not response times. For example, the execution time required to write a value to the follower ratio variables is 0.30 microseconds, however the time required to observe the resulting change in the axes motion would be in the order of

2 to 5 milliseconds. Similarly, for the digital inputs the hardware filter delays must be taken into account when computing the response time.

Note: *If the program execution time is between 300 and 350 microseconds a watchdog timeout may not occur, depending on the task loading in the module. The user should keep his program execution time within 300 microseconds to ensure that it runs without any timeouts.*

The tables below can be used to compute the worst case execution times and therefore predetermine that a program will not cause a watchdog timeout. The examples below illustrate the computation of execution times for a program.

E-2 Example 1

```
P001 := P002 + 3500;          (* Instruction Line 1 *)
IF P001 > 5000 THEN          (* Instruction Line 2 *)
    Torque_Limit_1 := 75;    (* Instruction Line 3 *)
    Jog_Plus_1 := Strobels_Level_1; (* Instruction Line 4 *)
END_IF;                      (* Instruction Line 5 *)
```

Execution Time for Instruction Line 1=>

(Time to Load P002) + (Time to load Constant) + (Time to perform Addition) + (Time to write P001)

=> 0.60 (from Table 97) + 0.50 (from Table 97) + 0.90 (from Table 91) + 0.60 (from Table 97)

=> **2.60 microseconds**

Execution Time for Instruction Line 2 =>

(Time to Load P001) + (Time to load Constant) + (Time to perform > Conditional)

=> 0.60 (from Table 97) + 0.50 (from Table - 97) + 2.50 (from Table 92)

=> **3.60 microseconds**

Execution Time for Instruction Line 3 (assuming Conditional evaluates to TRUE)=>

(Time to load Constant) + (Time to write Torque_Limit_1)

=> 0.50 (from Table 97) + 0.30 (from Table 93)

=> **0.80 microseconds**

Execution Time for Instruction Line 4 (assuming Conditional evaluates to TRUE)=>

(Time to load Strobe1_Level_1) + (Time to write Jog_Plus_1)

=> 1.40 (from Table 93) + 1.70 (from Table 93)

=> **3.10 microseconds**

Execution Time for Instruction Line 5 => **0.0 microseconds** (from Table 92)

Total Execution Time => 2.60 + 3.60 + 0.80 + 3.10 + 0.0 = **10.10 Microseconds**

E-3 Example 2

```
D00 := P100 * 1000;           (* Instruction Line 1 *)
P101 := D00 / 55;           (* Instruction Line 2 *)
Enable_Follower_1 := CTL01 BWAND CTL02; (* Instruction Line 3 *)
Follower_Ratio_A_1 := P101; (* Instruction Line 4 *)
```

Execution Time for Instruction Line 1 =>

(Time to Load P100) + (Time to Load Constant) + (Time to Multiply) + (Time to write D00)

=> 0.60 (from Table 97) + 0.50 (from Table 97) + 1.30 (from Table 91) + 0.70 (from Table 97)

=> **3.10 microseconds**

Execution Time for Instruction Line 2 =>

(Time to Load D00) + (Time to load constant) + (Time to perform divide) + (Time to write P101)

=> 0.70 (from Table 97) + 0.50 (from Table 97) + 2.90 (from Table 91) + 0.60 (from Table 97)

=> **4.70 microseconds**

Execution Time for Instruction Line 3 =>

(Time to Load CTL01) + (Time to load CTL02) + (Time to perform BWAND) + (Time to store Enable_Follower_1)

=> 1.40 (from Table 97) + 1.40 (from Table 97) + 0.20 (from Table 91) + 1.70 (from Table 93)

=> **4.70 microseconds**

Execution Time for Instruction Line 4 =>

(Time to Load P101) + (Time to write Follower_Ratio_A_1)

=> 0.60 (from Table 97) + 0.30 (from Table 93)

=> 0.90 microseconds

Total Execution Time => 3.10 + 4.70 + 4.70 + 0.90 = 13.40 Microseconds

Table 91: Local Logic Math/Logical Operation execution times

Local Logic Math and Logical Operations (Assignment, :=)	Local Logic Execution Time (Microseconds)
Add (+)	0.90*
Subtract (-)	0.90*
Multiply (*)	1.30
Divide (/)	2.90
Modulus (MOD)	2.90
Absolute (ABS)	1.70*
BWAND	0.20
BWOR	0.30
BWXOR	0.20
BWNOT	0.50

* Execution times for Addition, Subtraction and Absolute value (ABS) assume there are no computation overflows.

Table 92: Local Logic Conditional Operation Execution Times

Local Logic Conditional Operations (IF...THEN)	Local Logic Execution Time (Microseconds)
Greater Than (>)	2.50
Less Than (<)	2.50
Greater/Equal (>=)	2.50
Less/Equal (<=)	2.50
Equal (=)	2.30
Not Equal (<>)	2.30
BWAND	1.40
BWOR	1.40
BWXOR	1.40
BWNOT	1.60
Null operator (IF var THEN)	1.10
END_IF	0.00

Note: The execution time for the conditionals is for the case where the IF...THEN operation evaluates to FALSE. This represents the worst-case execution time, since the execution time required to evaluate a conditional that is TRUE is less. Note that the END_IF instruction does not require any execution time.

Table 93: Axis 1 Local Logic Variable Execution Times

X- Not Applicable.

Local Logic Variable Name	Local Logic Execution Time (In Microseconds)	
	Read	Write
Strobe1_Level_1	1.40	X
Strobe2_Level_1	1.40	X
Positive_EOT_1	1.40	X
Negative_EOT_1	1.40	X
Home_Switch_1	1.40	X
Digital_Output1_1	X	1.80
Digital_Output3_1	X	1.80
Analog_Input1_1	0.80	X
Analog_Input2_1	0.80	X
Position_Loop_TC_1	X	0.30
Follower_Ratio_A_1	X	0.30
Follower_Ratio_B_1	X	0.30
Torque_Limit_1	X	0.30
Position_Increment_Cts_1	X	0.30
Velocity_Loop_Gain_1	0.80	0.20
Reset_Strobe1_1	X	1.70
Reset_Strobe2_1	X	1.70
Enable_Follower_1	X	1.70
Jog_Plus_1	X	1.70
Jog_Minus_1	X	1.70
FeedHold_1	X	1.70
Error_Code_1	0.80	X
Actual_Position_1	0.70	X
Strobe1_Position_1	0.80	X
Strobe2_Position_1	0.80	X
Actual_Velocity_1	0.80	X
Block_1	0.90	X
Commanded_Position_1	0.60	X
Position_Error_1	0.60	X
Commanded_Velocity_1	0.60	X
User_Selected_Data1_1	0.60	X
User_Selected_Data2_1	0.60	X
UnAdjusted_Actual_Position_Cts_1	0.80	X
UnAdjusted_Strobe1_Position_Cts_1	0.80	X
UnAdjusted_Strobe2_Position_Cts_1	0.80	X

Local Logic Variable Name	Local Logic Execution Time (In Microseconds)	
	Read	Write
Commanded_Torque_1	0.80	X
Axis_OK_1	1.40	X
Position_Valid_1	1.40	X
Strobe1_Flag_1	1.40	X
Strobe2_Flag_1	1.40	X
Drive_Enabled_1	1.40	X
Program_Active_1	1.40	X
Moving_1	1.40	X
In_Zone_1	1.40	X
Position_Error_Limit_1	1.40	X
Torque_Limited_1	1.40	X
Servo_Ready_1	1.40	X
Follower_Enabled_1	1.40	X
Follower_Ramp_Active_1	1.40	X
Follower_Velocity_Limit_1	1.40	X

Table 94: Axis 2 Local Logic Variable Execution Times

X- Not Applicable.

Local Logic Variable Name	Local Logic Execution Time (In Microseconds)	
	Read	Write
Strobe1_Level_2	1.40	X
Strobe2_Level_2	1.40	X
Positive_EOT_2	1.40	X
Negative_EOT_2	1.40	X
Home_Switch_2	1.40	X
Digital_Output1_2	X	1.80
Digital_Output3_2	X	1.80
Analog_Input1_2	0.80	X
Analog_Input2_2	0.80	X
Position_Loop_TC_2	X	0.30
Follower_Ratio_A_2	X	0.30
Follower_Ratio_B_2	X	0.30
Torque_Limit_2	X	0.30
Position_Increment_Cts_2	X	0.30
Velocity_Loop_Gain_2	0.80	0.20
Reset_Strobe1_2	X	1.70
Reset_Strobe2_2	X	1.70

Local Logic Variable Name	Local Logic Execution Time (In Microseconds)	
	Read	Write
Enable_Follower_2	X	1.70
Jog_Plus_2	X	1.70
Jog_Minus_2	X	1.70
FeedHold_2	X	1.70
Error_Code_2	0.80	X
Actual_Position_2	0.70	X
Strobe1_Position_2	0.80	X
Strobe2_Position_2	0.80	X
Actual_Velocity_2	0.80	X
Block_2	0.90	X
Commanded_Position_2	0.60	X
Position_Error_2	0.60	X
Commanded_Velocity_2	0.60	X
User_Selected_Data1_2	0.60	X
User_Selected_Data2_2	0.60	X
UnAdjusted_Actual_Position_Cts_2	0.80	X
UnAdjusted_Strobe1_Position_Cts_2	0.80	X
UnAdjusted_Strobe2_Position_Cts_2	0.80	X
Commanded_Torque_2	0.80	X
Axis_OK_2	1.40	X
Position_Valid_2	1.40	X
Strobe1_Flag_2	1.40	X
Strobe2_Flag_2	1.40	X
Drive_Enabled_2	1.40	X
Program_Active_2	1.40	X
Moving_2	1.40	X
In_Zone_2	1.40	X
Position_Error_Limit_2	1.40	X
Torque_Limited_2	1.40	X
Servo_Ready_2	1.40	X
Follower_Enabled_2	1.40	X
Follower_Ramp_Active_2	1.40	X
Follower_Velocity_Limit_2	1.40	X

Table 95: Axis 3 Local Logic Variable Execution Times

X- Not Applicable.

Local Logic Variable Name	Local Logic Execution Time (In Microseconds)	
	Read	Write
Strobe1_Level_3	1.40	X
Strobe2_Level_3	1.40	X
Positive_EOT_3	1.40	X
Negative_EOT_3	1.40	X
Home_Switch_3	1.40	X
Digital_Output1_3	X	1.80
Digital_Output3_3	X	1.80
Analog_Input1_3	0.80	X
Analog_Input2_3	0.80	X
Reset_Strobe1_3	X	1.70
Reset_Strobe2_3	X	1.70
Error_Code_3	0.80	X
Actual_Position_3	0.70	X
Strobe1_Position_3	0.80	X
Strobe2_Position_3	0.80	X
Actual_Velocity_3	0.80	X
Axis_OK_3	1.40	X
Position_Valid_3	1.40	X
Strobe1_Flag_3	1.40	X
Strobe2_Flag_3	1.40	X

Table 96: Axis 4 Local Logic Variable Execution Times

X- Not Applicable.

Local Logic Variable Name	Local Logic Execution Time (In Microseconds)	
	Read	Write
Strobe1_Level_4	1.40	X
Strobe2_Level_4	1.40	X
Positive_EOT_4	1.40	X
Negative_EOT_4	1.40	X
Home_Switch_4	1.40	X
Digital_Output1_4	X	1.80
Digital_Output3_4	X	1.80
Analog_Input1_4	0.80	X
Analog_Input2_4	0.80	X

Table 97: Global Local Logic Variable Execution Times

X- Not Applicable

Local Logic Variable Name	Local Logic Execution Time (In Microseconds)	
	Read	Write
Local Logic Program Constants	0.50	X
Overflow	2.40	1.30
System_Halt	X	1.80
Data_Table_Ptr	0.60	0.70
Data_Table_sint	2.10	1.70
Data_Table_usint	1.80	1.70
Data_Table_int	2.30	2.20
Data_Table_uint	2.30	2.20
Data_Table_dint	3.80	4.00
Module_Error_Present	1.40	X
New_Configuration_Received	1.40	X
First_Local_Logic_Sweep	1.40	X
Module_Status_Code	0.50	X
CTL_1_to_32	0.50	X
P000-P255	0.60	0.60
D00-D07	0.70	0.70
CTL01-CTL32	1.40	1.80

Appendix F: Updating Firmware in the DSM314

The DSM314 operating firmware is stored in on-board FLASH memory. The Winloader update utility requires Windows 95, Windows NT, or Windows 98, or Windows 2000. The hardware required to run these operating systems should suffice to also run Winloader. Winloader requires about 500Kbytes of hard disk space.

The DOS-based PC Loader utility controls downloading the new firmware from the floppy to the DSM314 FLASH memory. PC Loader requires an IBM AT/PC compatible computer with at least 640K RAM, one floppy drive, MS-DOS 3.3 (or higher), and one RS-232 serial port. In order to run this utility within an MS-DOS box under Windows® 3.1, Windows 95 or Windows NT, the processor should be at least a Pentium 133. If not, the computer should be rebooted into MS-DOS mode. PC Loader functions optimally with a hard drive with at least 1 MB available space.

WARNING

The user MUST determine that the PC is connected to a DSM (and not a Host Controller CPU or other module that supports FLASH firmware upgrades) before entering Boot mode. Failure to do so can cause loss of Host Controller CPU Program and Configuration.

To Install the New Firmware, Perform the Following Steps:

1. Save or back up any programs or data resident in the module before performing the update function.
2. Place the Host Controller in STOP/NOIO Mode. (Clear any faults.)
3. Ensure that the module's SNP serial port baud rate is set to 19200 baud.
4. Using a Station Manager to PC cable, IC693CBL316, connect the appropriate serial port of your computer (master) to the DSM314 module to be updated (slave).

F-1 Windows Update (for Windows 95/NT/98/2000)

Note: *This section only applies to those using the Winloader update software with Windows 95, NT, 98 or 2000. If using the DOS operating system, see the section "DOS Update."*

1. Insert a labeled floppy disk in drive A: or B. Ensure that the floppy is not write protected. Run the self-extracting archive specifying drive A: or B: as the destination when prompted with "Unzip to folder".
2. Invoke the Winloader software package by double clicking on its icon located in drive A: or B: (depending on the drive designation for the 3.5" floppy disk) in Windows Explorer or simply execute it by going to the start menu and selecting RUN. In the RUN window type A (or B): winloader.exe.
3. Begin storing firmware by single clicking the "Update" button.
4. Upon completion of the update a window will "pop up" indicating the status of the update. If the update was successful, power cycle the Host Controller and indicate that another device is NOT to be updated by left clicking on "No". If not successful, consult the on-line help for additional information.

F-2 DOS Update

Note: *This section only applies to those running the DOS Loader update program from DOS. For those using Windows software, refer to "Windows Update."*

1. Insert a labeled floppy disk in drive A: or B: Ensure that the floppy is not write protected. Run the self-extracting archive specifying drive A: or B: as the destination when prompted with "Unzip to folder:".
2. At the C:\> prompt, type A: install (or B: install if your floppy drive is B:). The install program will copy several files to the hard drive then invoke the PC Loader. Install can also be run from the floppy drive directly if there is not enough space on the hard drive. To run from the floppy, type install at the A:\> or B:\> prompt.
3. From the main menu, press the F3 key to configure the correct serial port if the cable is not connected to COM1. Press the TAB key to toggle through the options and ENTER to accept the displayed choice.
4. From the main menu, press the F1 key to attach to the DSM312 slave device.
5. Once the slave device is attached, the boot mode menu will appear - press F1 to enter BOOT MODE and press the 'Y' key to confirm the operation. The STAT and CFG LED's on the front of the module should now be flashing in unison.
6. Once in boot mode, press the F1 key to download the new firmware.
7. Press the Y key to confirm the operation. The download should take about 4 minutes. If the download fails, refer below to Restarting An Interrupted Firmware Upgrade.

8. When the download is complete, the PC loader will instruct you to power cycle your module. At this time, power cycle the module. If the module is installed in an expansion or remote rack, it is necessary to also power-cycle the main rack.
9. Label the unit with the installed firmware version. If the firmware is Beta or an Engineering Release, indicate so on the label.

F-3 Restarting an Interrupted Firmware Upgrade

- A. Connect all cables as described in step 4 of the procedure above.
- B. Power cycle the rack containing the module. If a partial or erroneous download was performed, the module will power up with the STAT and CFG LED's on the module flashing in unison.
- C. If you are still running the PC Loader or Winloader program on your PC, skip to step D below; otherwise, follow steps 5 and 6 above.
- D. Follow step 7 above. Note that you will automatically be placed in BOOT MODE. E. Follow steps 9 through 12 above.
- E. If the update still fails, repeat the process with a lower baud rate.
- F. Label your unit with the installed firmware version.

MS-MS-DOS, Windows, and Windows NT are registered trademarks of Microsoft Corporation; Pentium is a trademark of Intel Corporation; IBM-AT and IBM-PC are registered trademarks of International Business Machines Corporation.

Appendix G: Strobe Accuracy Calculations

In general the accuracy of the strobe position value can be expressed as +/- 2 counts with an additional variance of 10 microseconds. However, the actual accuracy of the strobe position value may be better than that depending upon axis configuration, motor acceleration during a strobe event, and the number of counts per revolution of the encoder used. The first consideration is whether the axis configuration is Digital or Analog.

G-1 Analog Mode

In Analog mode, when a strobe event occurs, the quadrature counter value is latched into a holding register immediately. This means that the position capture inaccuracies are based primarily on the input filtering and sampling delay for the strobe input which can total up to 10 microseconds (or the number of counts that can occur in 10 microseconds). Note that the value may be one count off based on when the strobe event occurred in relation to when the count value changed.

G-2 Digital Mode

In Digital mode the encoder is read as serial data. Because this data is only acquired once every 250 microseconds, latching the position value read from the encoder will only allow an accuracy of 250 microseconds. To overcome this limitation, the strobe event is time stamped in relation to the last encoder position reading that occurred within the DSM314. This value is used to estimate the axis position at the instant that the strobe event occurred based on the actual servo axis velocity at the time of the strobe. The velocity used for the calculation is derived from the difference in the two encoder position readings around the strobe event (see the formula below).

$$\text{Velocity} = \frac{(\text{Position Sample after Strobe}) - (\text{Position Sample before Strobe})}{250 \text{ microseconds}}$$

Therefore, changes in velocity (i.e. acceleration or deceleration of the motor) between position samples are not taken into account thus causing inaccuracies in the captured strobe position value. For strobe events that occur during when velocity is constant during the sampling period, the interpolation algorithm will be accurate to within one count and the position capture inaccuracy will be primarily determined by the filtering and sampling delays.

The following example can be used to calculate the worst case inaccuracies due to acceleration given a particular servo motor:

Given the following values/constant for this example:

Encoder Resolution = 8192 cnts/rev

A = Acceleration/deceleration during the strobe event which is 250,000,000 cnts/sec² (assumed to be constant over the entire 250µs period; Larger acceleration values will increase the amount of error in the calculation)

T_p = Position sampling period which is 250 microseconds

V_i = Initial velocity just before the strobe event which will be 0 for this example.

The change in the number of encoder counts (Cnts) for a given amount of time (t) can be calculated using the following formula:

$$P_{act} = V_i t + \frac{1}{2} A t^2$$

Therefore the total number of counts to occur during the sampling period for this example is approximately 8 counts (actual calculated values is 7.8125) or 0.343 degrees of motor rotation.

The average velocity for the sample period given the change in position would be as follows:

$$V_{avg} = \frac{\text{Change in counts}}{\text{Sampling Period}} = \frac{7.8125 \text{ cnts}}{250 \mu\text{sec}} = 31250 \text{ cnts/sec}$$

The following formula can be used to estimate the strobe position using the velocity derived above:

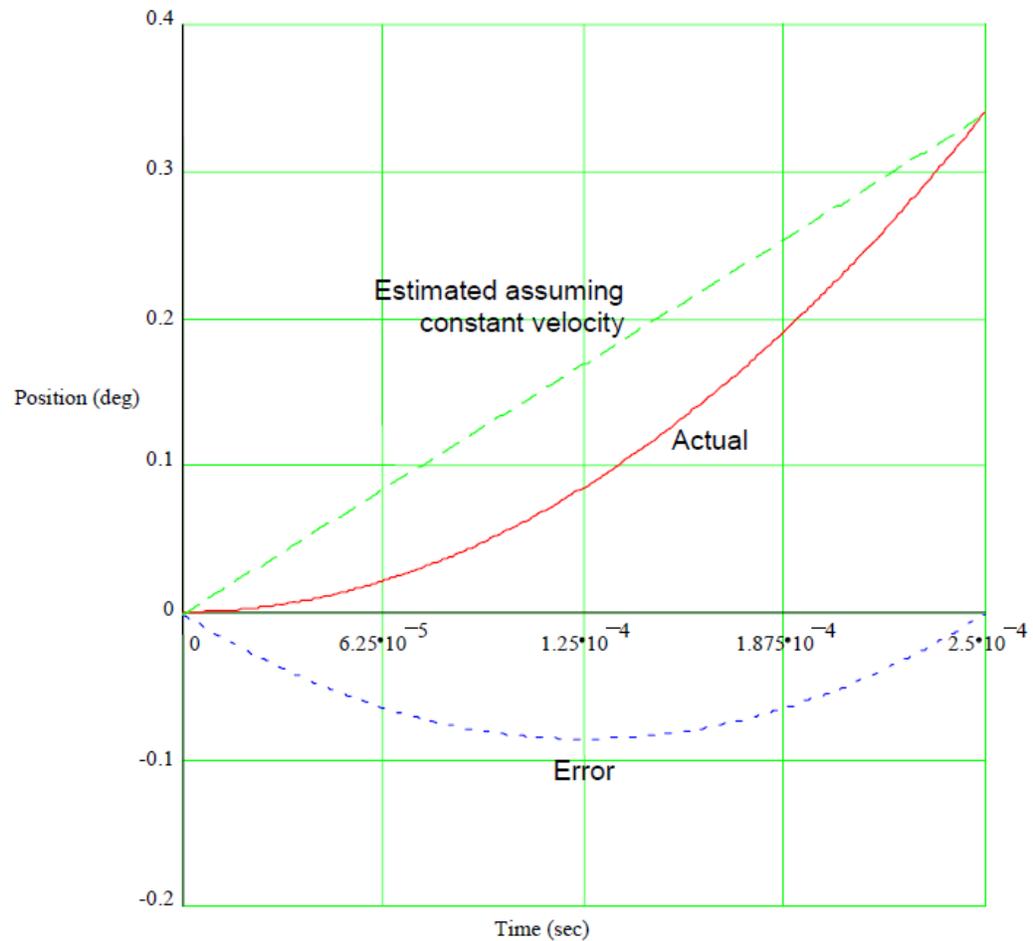
$$P_{est} = V_i t + V_{avg} t$$

Therefore, the error between the estimated strobe position and the actual strobe position is as follows:

$$\text{Error} = P_{act} - P_{est}$$

The graph below contains plots of the actual position, the estimated position, and the resulting strobe position count error for the 250-microsecond sample period. The graph shows that the greatest count error occurs in the middle (i.e. at 125 microseconds) of the period.

Figure 219: Example axis position capture error due to acceleration



Since the initial velocity is equal to 0, the formula for calculating P_{act} can be manipulated to determine the time that the count actually occurred at (T_{act}) as follows:

$$T_{act} = \sqrt{\frac{2P_{act}}{A}}$$

Likewise, the formula for estimating the strobe position (P_{est}) can be solved for time (T_{est}) as well (assuming that the initial velocity is 0):

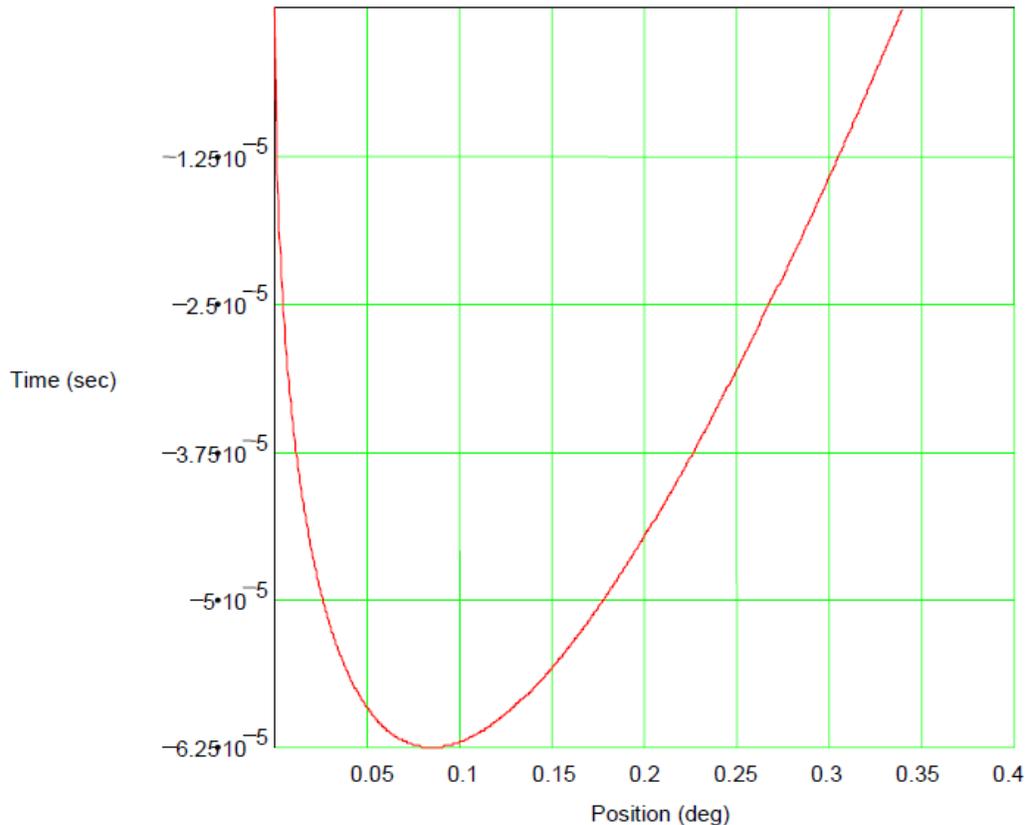
$$T_{est} = P_{est} / V_{avg}$$

Using these formulas, the difference in time between when the strobe occurred and when the reported count occurred (i.e. the effective delay) can be calculated as follows:

$$\text{Effective Delay} = T_{est} - T_{act}$$

The effective delay for the maximum strobe position error (i.e. at 125 microseconds) is equal to -62.5 microsecond. This value is negative because the estimated/reported strobe position occurred prior to the actual position when the strobe event happened. The following graph represents the effective delay that would be seen across the change in position for the sampling period in this example.

Figure 220: Effective response time delay



Therefore, in the example above, the worst-case error due to acceleration/deceleration can be expressed as +/- 0.086 degrees (approximately 2 counts) of position or as 62.5 microseconds of delay (given that the initial velocity is 0). **Note that the DSM cannot deal with fractional units and therefore the error will be rounded to the nearest count or user unit.**

The formulas for determining the strobe error due to acceleration/deceleration on a Digital axis are as follows:

Counts_of_error =

$$\text{Effective_delay} = \frac{T_p(V_i + A T_p)}{4(2 V_i + A T_p)}$$

Where:

A = Acceleration/deceleration during the strobe event

T_p = Position sampling period which is 250 microseconds

V_i = Velocity just before the strobe

Note that the formulas above assume constant acceleration throughout the sampling period. The formulas for determining the error for the cases where acceleration is not constant during the sampling period are too complex for the context of this manual.

Note that an additional error as much as 10 microseconds (or the number of degrees or position counts that can occur in 10 microseconds) may also be seen due to input filtering/sampling delays in the hardware.

 WARNING

Note that user wiring and the type of device used for the strobe input may also cause inaccuracies in the strobe value.

Appendix H: Using VersaPro with the DSM314

The examples shown in this chapter are specific to the VersaPro programming software. Users of Machine Edition software should refer to other chapters in this manual and the on-line help for instructions on configuring and programming the DSM314 controller.

H-1 Getting Started

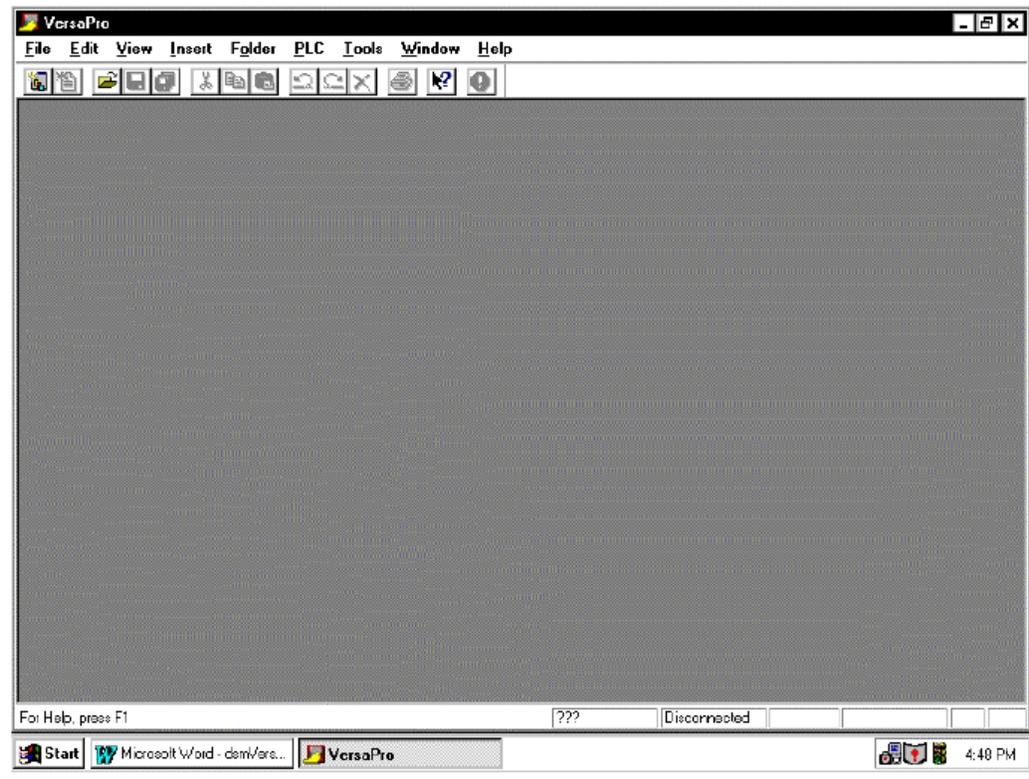
Note: VersaPro Version 1.1 or later is required for use with the DSM314.

This document discusses how to use the VersaPro software to access the DSM314 configuration, motion programming, and Local Logic programming screens. It does not tell you specifically what values to configure, or what commands to use in motion or Local Logic programs. That information is covered elsewhere in this manual. Additional VersaPro information can be found in the VersaPro Programming Software User's Guide, GFK-1670, as well as in VersaPro's on-line help.

H-1.1 Starting VersaPro

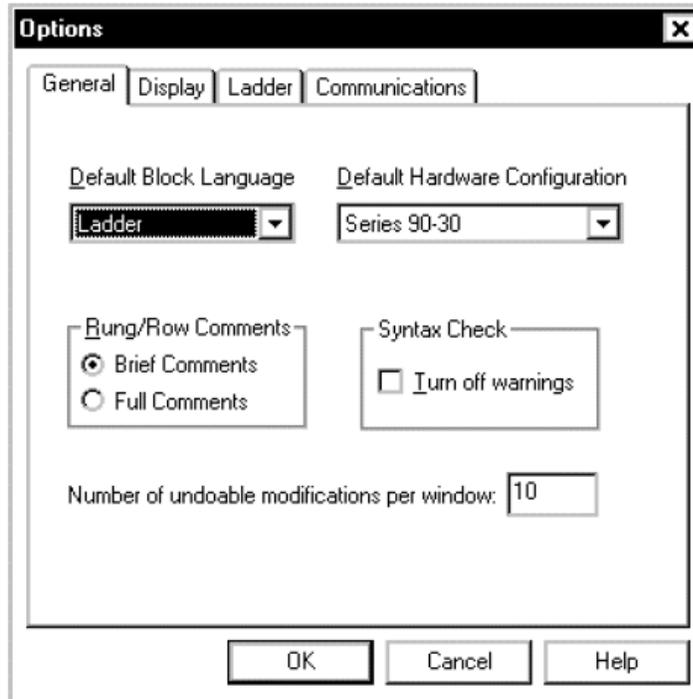
Double click the VersaPro icon on your Windows desktop to start the software running. VersaPro will start with a blank screen called the "Workbench."

Figure 221: VersaPro Startup Screen



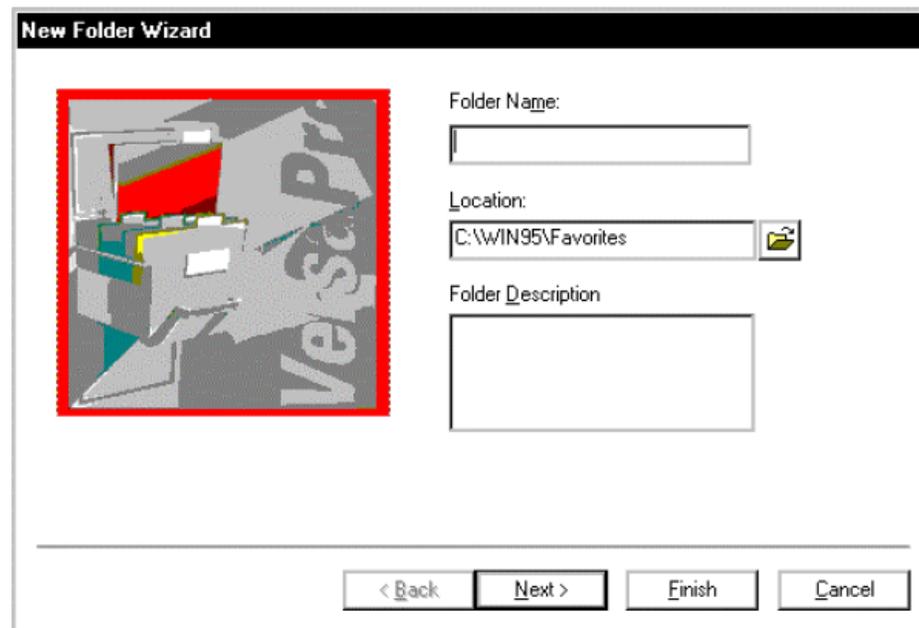
- Before creating a file, check the Workbench default settings to make sure that Series 90-30 is the default PLC. To do this, click Tools on the Menu bar (see Figure 15-4), then click the Options selection. The Options dialog box will appear, as shown next:

Figure 222: Checking the VersaPro Default Settings in the Tools/Options Dialog Box



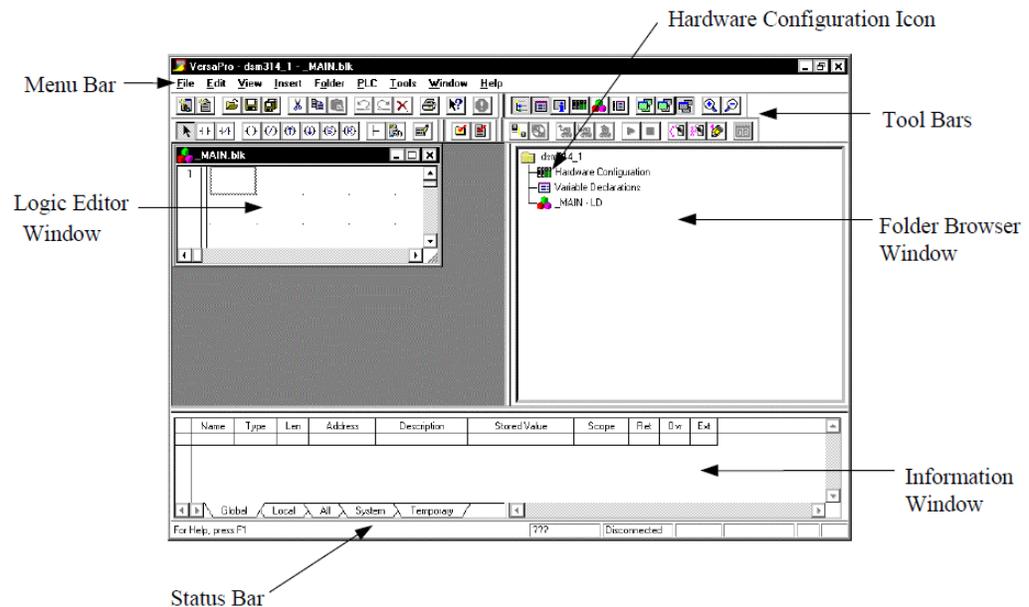
- Make sure Series 90-30 is shown in the Default Hardware Configuration box, then click the OK button.
- To open a folder, click File on the Menu bar, then either click Open Folder to open an existing one, or click New Folder to create a new one. You can also import an existing Series 90-30 folder that was originally created in Logicmaster or Control. See the “Folder Operations” section of Chapter 2 in the VersaPro User’s Guide, GFK-1670 for details. For this example, click New Folder. A New Folder Wizard dialog box will appear as shown in the next figure.

Figure 223: The New Folder Wizard Dialog Box



- Enter a name for your folder. Although you can use up to 255 characters to name the folder, only the last 7 characters will be used as the folder name in the PLC. These last 7 characters of the Folder Name are called the Folder Nickname. This being the case, you may wish to carefully name your folder so that its nickname is meaningful. For example, if your Folder Name is “Pumphouse_Number_1,” the Nickname stored to the PLC is “umber_1” which may not convey the meaning you desire. Better names might be something like “PHouse1” or “PH1.” Chapter 2 of the VersaPro user’s manual (GFK-1670) has a section that explains the rules for creating Folder Names, including which characters are allowed.
- You may also change the folder location path from the default path shown in the Location field if you wish to store your folder in a different location. Also, there is a Description field that allows you to enter up to 64 characters of description information. When finished entering information in this dialog box, click the Finish button. (If you wanted to import a Logicmaster or Control folder, you would click Next, which would give you another dialog box with the import choices.) You will now see the Main LD (Ladder Diagram) screen.

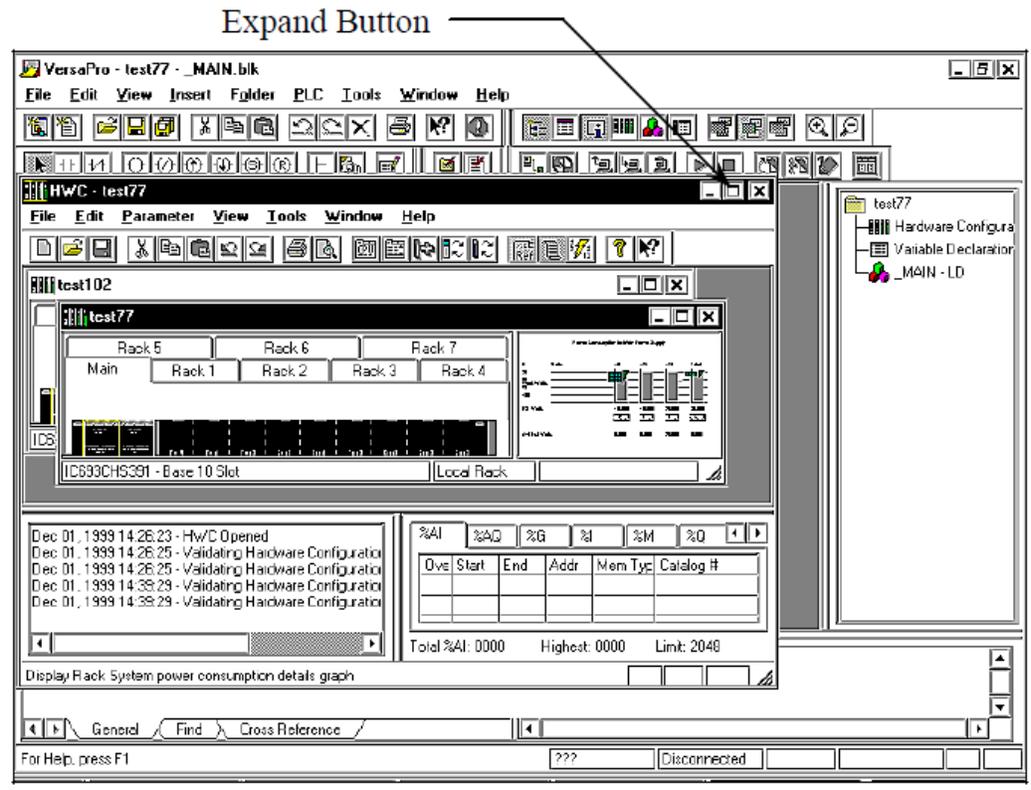
Figure 224: VersaPro's Main Ladder Diagram (LD) Screen



H-2 Starting the Configuration Process

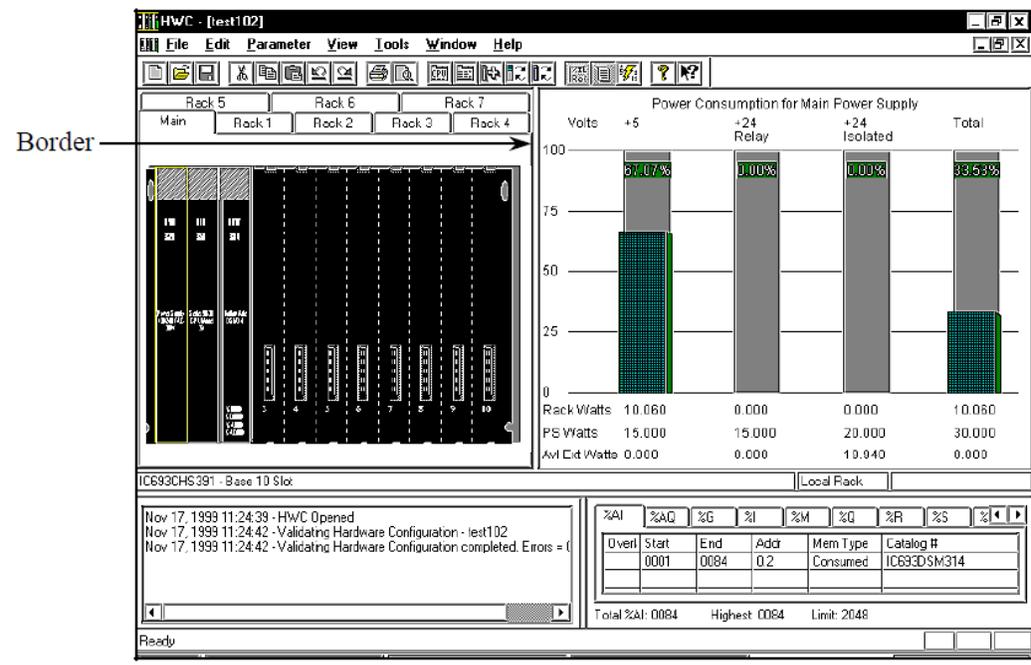
The configurator is actually a separate program that you can launch from the Main screen (shown in the previous figure). To begin, double click the Hardware Configuration icon to launch the HWC (Hardware Configuration) program. The HWC screen may appear as a window in or on top of the VersaPro Workbench, as shown below. If so, click the Expand button to expand it to full size. (You may also have to click the Expand Button in the smaller window to expand it also.)

Figure 225: The Hardware Configuration (HWC) Startup Screen



The Configuration Window will expand to its full size:

Figure 226: The Expanded Hardware Configuration Screen

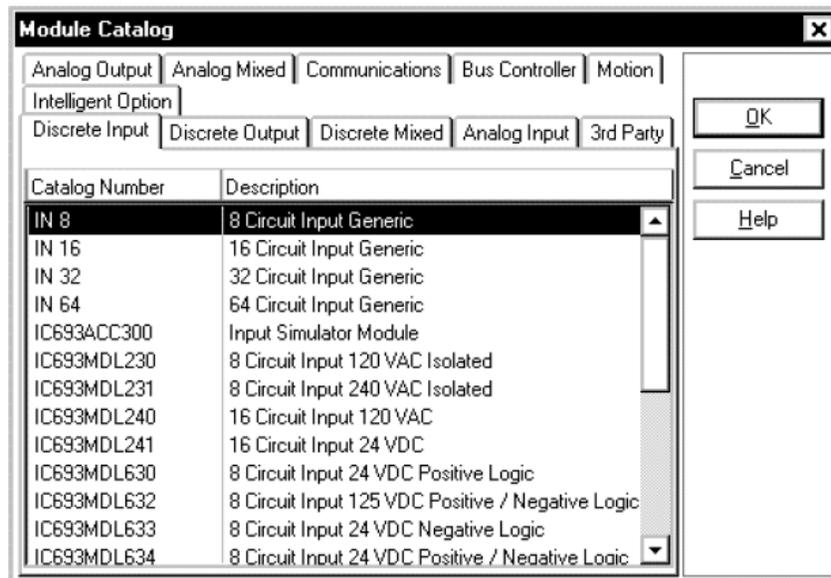


H-3 Configuring the DSM314

The following information discusses configuring the DSM314. For configuring other hardware, please refer to the VersaPro User's Guide, GFK-1670, and the VersaPro on-line help.

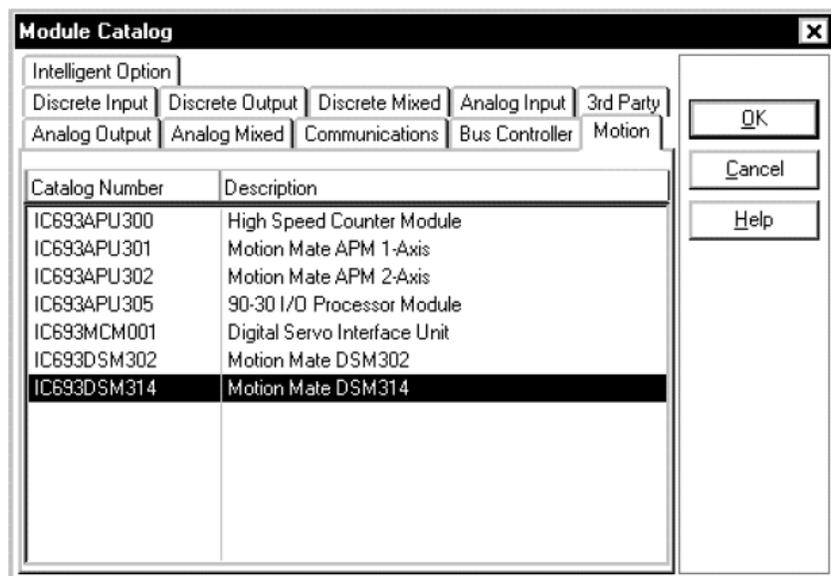
- With the Configuration window open, as shown in the previous figure, double click the empty slot where the DSM314 is to be installed. You will see a Module Catalog window appear with a list of module categories:

Figure 227: Module Catalog Window for Hardware Configuration



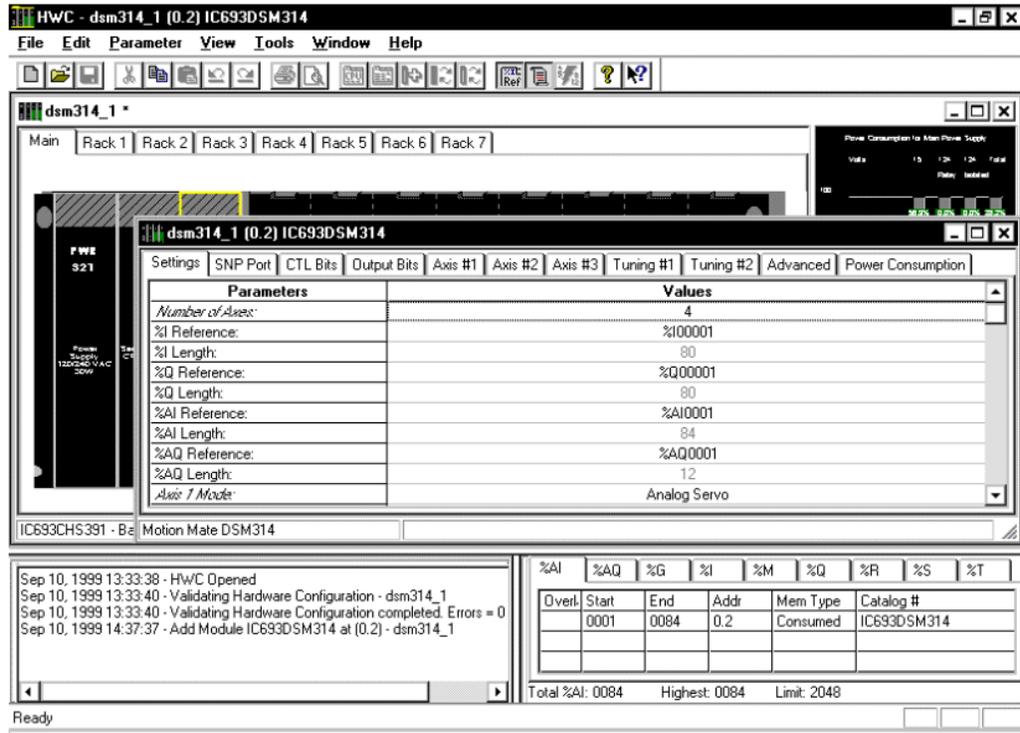
- Click the Motion tab to access a list of motion module choices:

Figure 228: Motion Tab for Hardware Configuration



- Double click the IC693DSM314 or highlight it as shown and click the OK button. The DSM314 will be added to the on-screen rack, and its Configuration window will appear:

Figure 229: DSM314 Hardware Configuration Window



The figure above shows the DSM314 default configuration settings. Only 11 of the selection tabs are displayed. Other tabs not shown will appear if their associated parameters are selected. For details on individual configuration settings, refer to Chapter 4. Here is a summary of the tabs:

Table 98: DSM314 Hardware Configuration Window Selection Tabs

Tab Name	Description
Settings	Contains PLC Reference assignments and lengths, DSM Axis setup, and other global data.
SNP Port	Setup for the DSM front panel SNP port (labeled COMM).
CTL Bits	Configuration for 24 Control bits used inside the DSM.
Output Bits	Configuration for the 8 DSM faceplate digital outputs.
Axis #1	Configuration of axis parameters such as Position Limits, Find Home Velocity, and Jog Acceleration.
Axis #2	
Axis #3	
Axis #4	
Tuning #1	Configuration of servo loop tuning items such as Motor Type, Position Loop Time Constants, and Velocity Feedforward parameters.
Tuning #2	
Tuning #3	
Tuning #4	
Advanced	Allows user entry of custom tuning parameters for any axis.
Power Consumption	Lists DSM power consumption required from the backplane supply (4.0 watts plus encoder power).

- When finished configuring the module, click the DSM314 configuration window's close button (the button in the upper right corner of the configuration window with an X) to return to the "Rack View." At this point, your configuration settings are not yet saved to disk. They only reside in your computer's volatile RAM memory.

Saving Your Configuration Settings to Disk

- Click File on the Menu bar, then click Save on the drop-down File menu. The configuration settings will be written to the applicable file in your program folder. Once a file is saved, the Save selection on the Menu file becomes inactive (it changes from black to a light green color). If you make any further changes to the configuration, the Save selection on the File menu will return to its active state (and its color will change back to black).
- After saving your configuration file, click File on the Menu bar, then click Exit to return to the Main LD screen.

H-4 Connecting to and Storing Your Configuration to the PLC

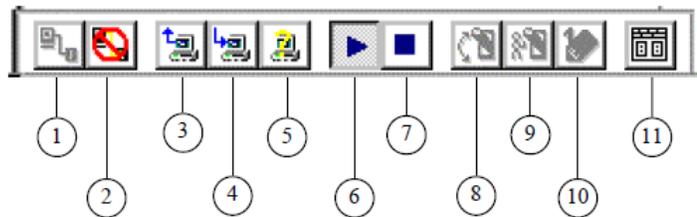
Note: You cannot store your configuration file to the PLC from within the configurator program. You must be on VersaPro's Main LD screen in order to store to the PLC.

Useful Tool Bar Icons

Several toolbar icons will be used in the next several steps to initiate such operations as Connect, Stop the PLC, and Store. The following figure identifies these toolbar icons:

Figure 230: VersaPro Toolbar Icons

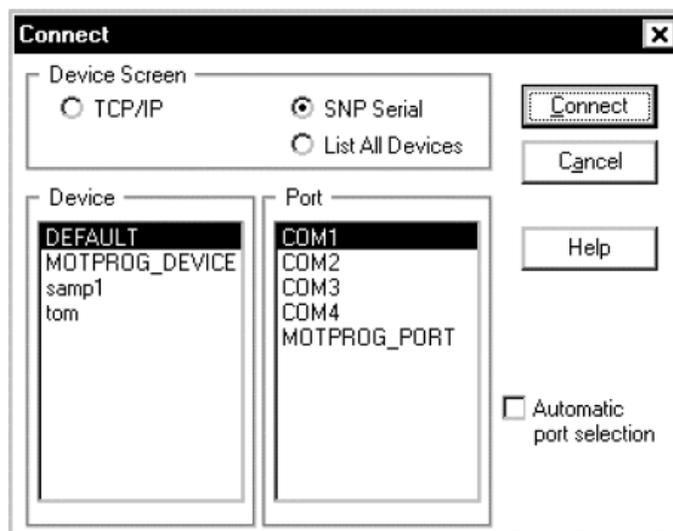
1. Connect
2. Disconnect
3. Store to PLC
4. Load from PLC
5. Verify with PLC
6. Run PLC
7. Stop PLC
8. Toggle a Reference
9. Override a Reference
10. Write a Value to a Reference
11. View PLC Status



Connecting to the PLC

- On the main VersaPro screen, click the Connect icon on the Toolbar. The Connect dialog box will appear.

Figure 231: The Connect Dialog Box



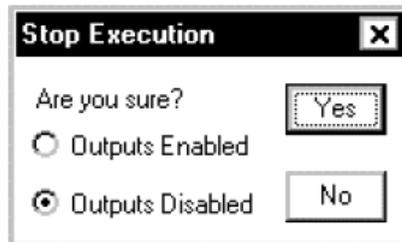
- If connecting directly to the PLC programmer port from the COM1 serial port on your computer, use the DEFAULT settings shown in the figure above.

- Make sure your serial cable is connected between your computer and the serial port on the PLC. Then click the Connect button on the Connect dialog box to begin connecting to the PLC. The message bar at the bottom of the VersaPro screen will display a “Connecting” message with a horizontal bar graph. Once the connection is made, the Status bar message will change from Disconnected to Connected.

Stopping the PLC

- The PLC must be stopped to store configuration files, so click the Stop icon on the Tool bar. The Stop Execution dialog box will appear.

Figure 232: The Stop Execution Dialog Box

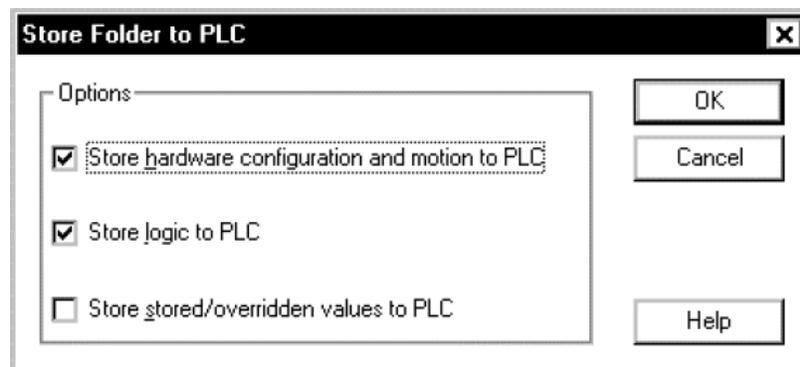


- Click Yes to stop the PLC. The Status bar message at the bottom of the screen will change from Run Enabled to Stop Disabled.

Store Operation

- Click the Store to PLC icon on the Tool bar. The Store Folder to PLC dialog box will appear.

Figure 233: The Store Folder to PLC Dialog Box



- Make sure the “Store hardware configuration and motion to PLC” item is checked as shown, then click the OK button to store to the PLC. Once the store is complete, the message on the Status bar at the bottom of the screen will change from Not Equal to Equal.

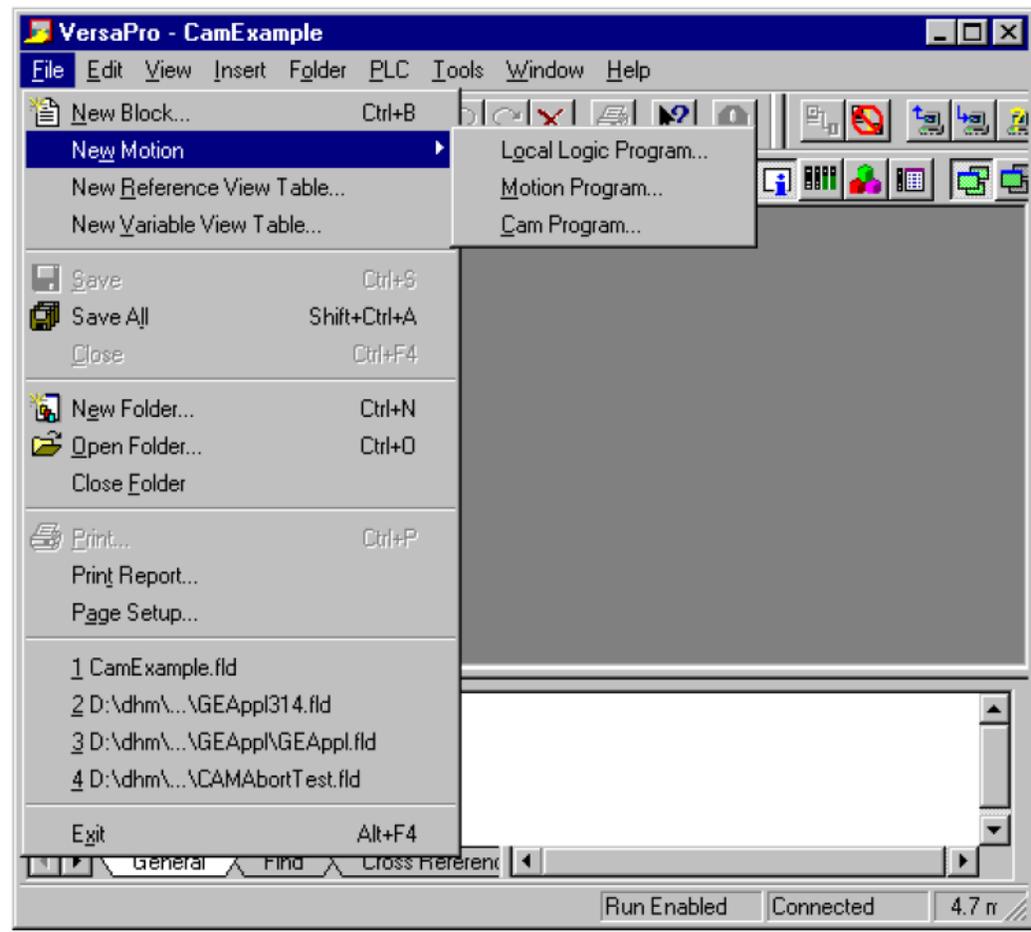
H-5 Creating a Motion Program

H-5.1 Accessing the Motion Editor Screen

Both the Motion Editor and Local Logic Editor are accessed from the VersaPro Folder Browser window. However, once created and saved, motion programs and Local Logic programs become part of the PLC CPU Hardware Configuration and are Stored to the PLC with the other configuration information.

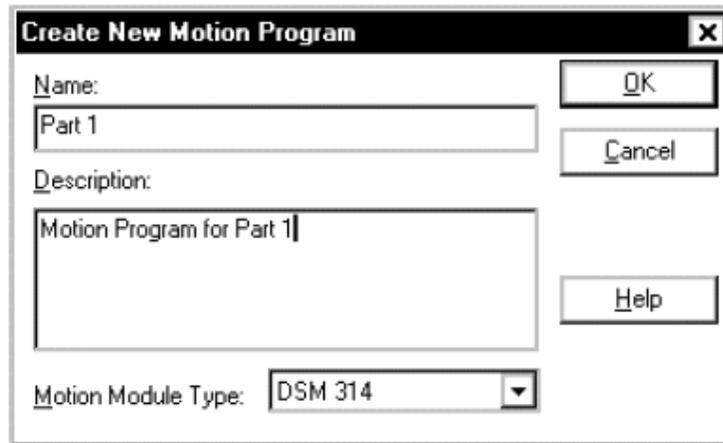
- On the Main LD screen, click File on the Menu bar, then select New Motion. Then, on the side menu, click Motion Program (see next figure).

Figure 234: Creating a New Motion Program from the File Menu



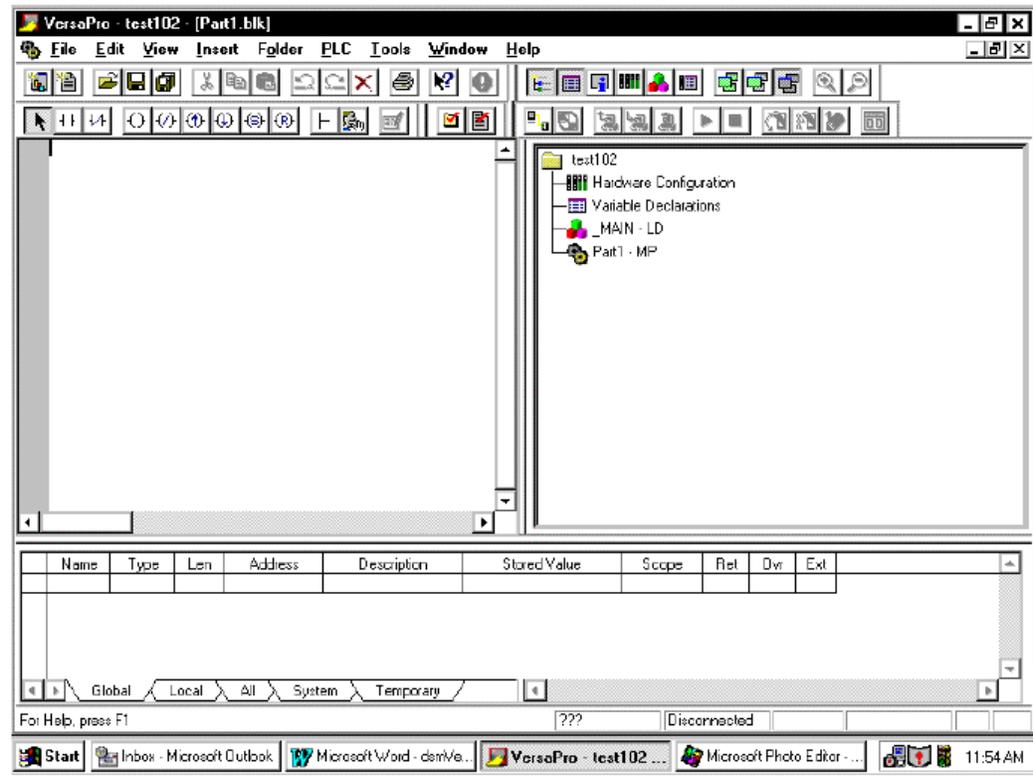
- The Create New Motion Program dialog box will appear.

Figure 235: The Create New Motion Program Dialog Box



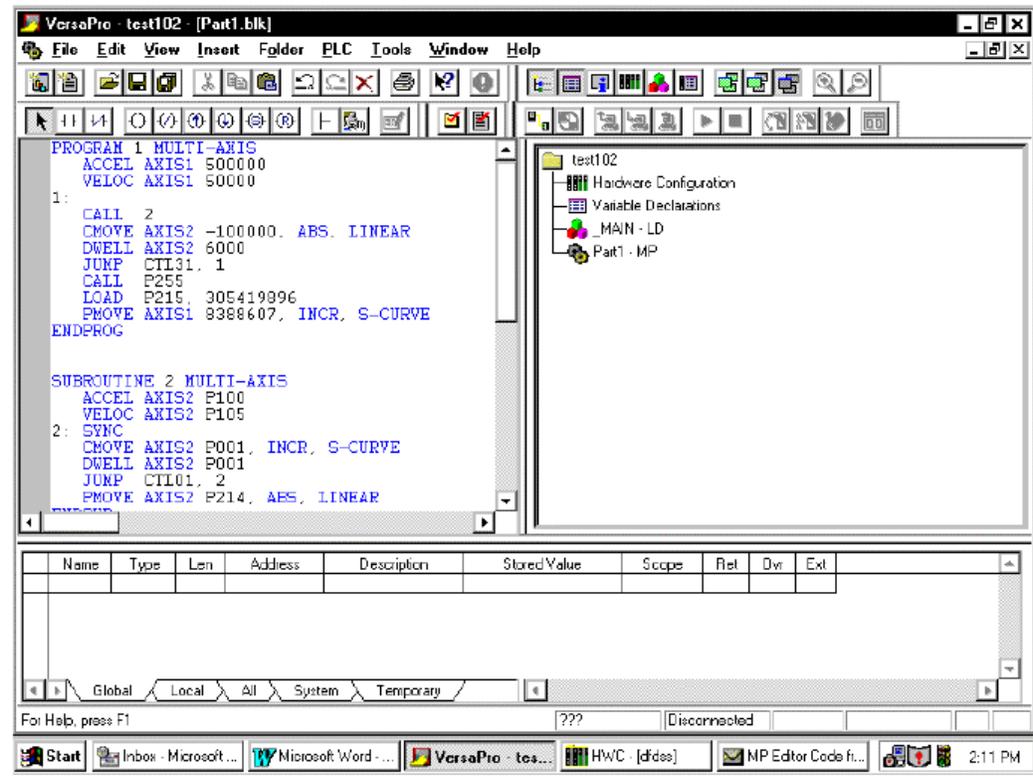
- Enter the motion program Name and Description, then click the OK button (leave the Motion Module Type box set at its default DSM314 setting). A window for the new motion program block will open. As shown in the next figure, the window title is based upon the folder name, Test102 in this case, and motion program name, Part1 in this case. Notice also in the next figure that an icon for the new motion program, called “Part1 – MP” (Motion Program), appears in the Folder Browser window.

Figure 236: A New Motion Editor Window



- The text-based motion programs and subroutines are created in the Motion Editor window, as shown in the following figure. Up to 10 motion programs and 40 subroutines, separated by their identifying headers (such as “PROGRAM 1 MULTI-AXIS”), are programmed in the same window and are stored in the same file. Details on motion program commands and syntax are covered in Chapter 7.

Figure 237: Motion Editor Window with Programmed Code



H-5.2 Saving your Motion Program

- When ready to save your motion program/subroutine file to your computer’s hard disk, either click the Save icon on the tool bar (looks like a floppy diskette), or click File from the Menu bar and click Save.

H-5.3 Storing your Motion Programs and Subroutines to the PLC

Since the Motion Program/Subroutine file is considered part of the Configuration file group, use the procedure under the heading “Connecting to and Storing Your Configuration to the PLC” on page 473.

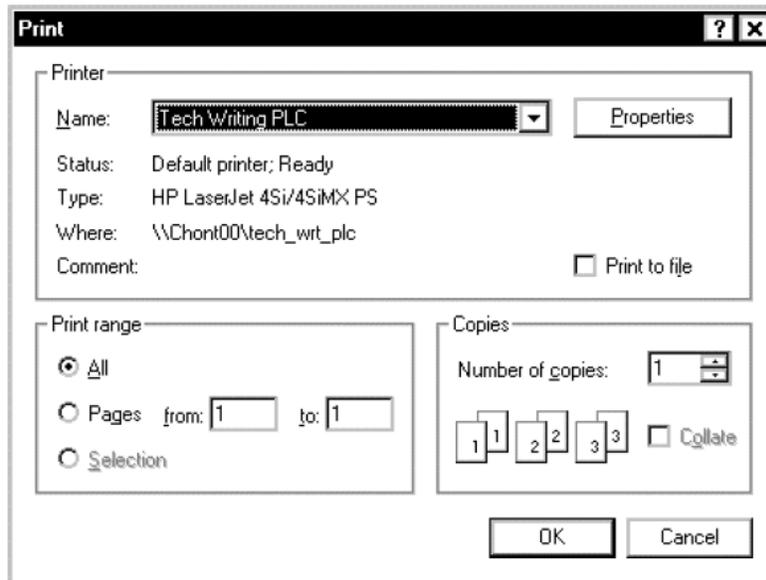
H-5.4 Printing a Hardcopy of your Motion Programs and Subroutines

There are two print selections on the File menu: Print and Print Report.

Print

- This item describes how to print your entire motion program file (block). While the Motion Editor is active, click File on the Menu bar and select Print. The Printer dialog box will display. Make any desired printer setup changes, then click the OK button.

Figure 238: Print Dialog Box

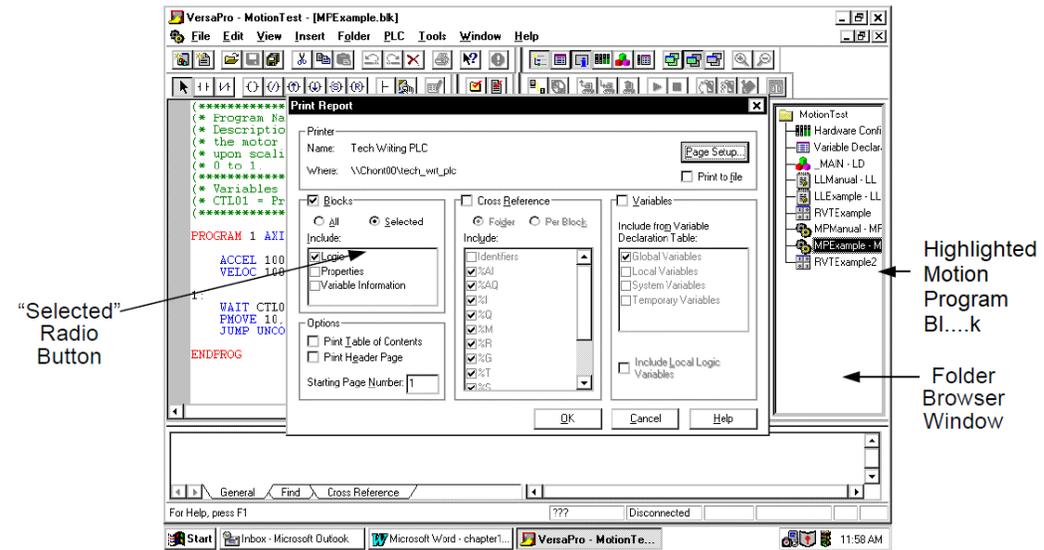


- This item describes how to print just a selected portion of your motion program/subroutine file. In the Motion Editor window, use your mouse to select the portion you wish to print, click File on the Menu bar, then select Print. In the Print dialog box (shown above), make sure the Selection radio button in the Print range section is selected (has a dot in the middle). Click the OK button.

Print Report

- To print all motion program blocks (if you have more than one) as part of a report with the other information in the folder, click File on the Menu bar and select Print Report. The Print Report dialog box will appear. Click the Blocks checkbox on the Print Report dialog box. Make sure the All radio button is selected. (You can also select other items and features for the report such as Table of Contents, Cross References, Variables, etc.) Click the OK button to start printing. Motion program, Local Logic, and Ladder Diagram blocks will be printed as part of this report.
- To print only selected blocks, highlight them in the Folder Browser window. Click File on the Menu bar and select Print Report. Click the Blocks checkbox, then choose the Selected radio button. This limits the reports to only those blocks that you have highlighted in the Folder Browser window.

Figure 239: The Print Report Dialog Box

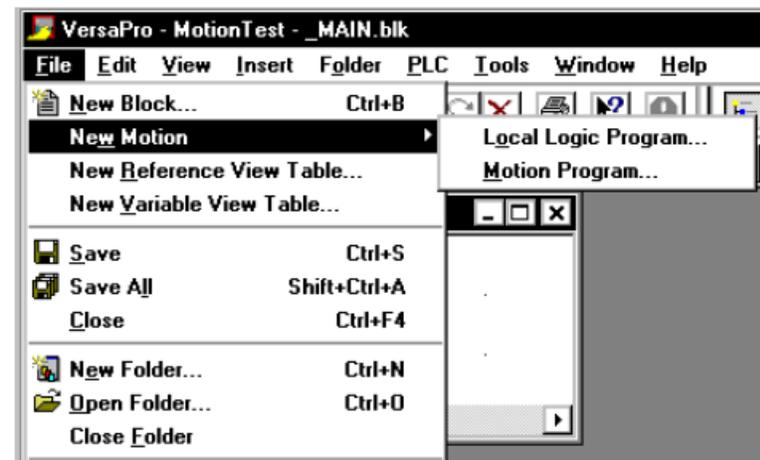


H-6 Creating a Local Logic Program

Both the Motion Editor and Local Logic Editor are accessed from VersaPro’s Folder Browser window. However, once created and saved, motion programs and Local Logic programs become part of the PLC CPU Hardware Configuration and are Stored to the PLC with the other configuration information.

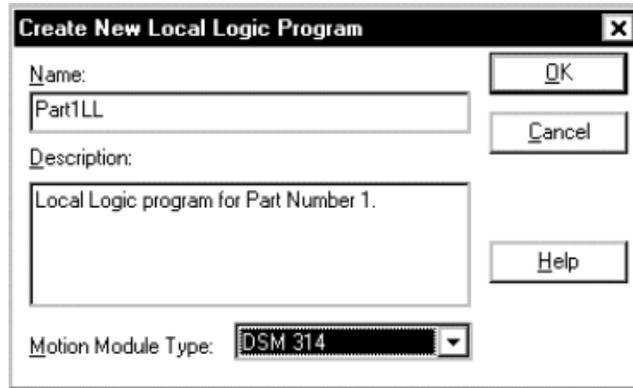
- On the Main LD screen, click File on the Menu bar, then select New Motion. Then, on the side menu, click Local Logic Program.

Figure 240: Creating a New Local Logic Program



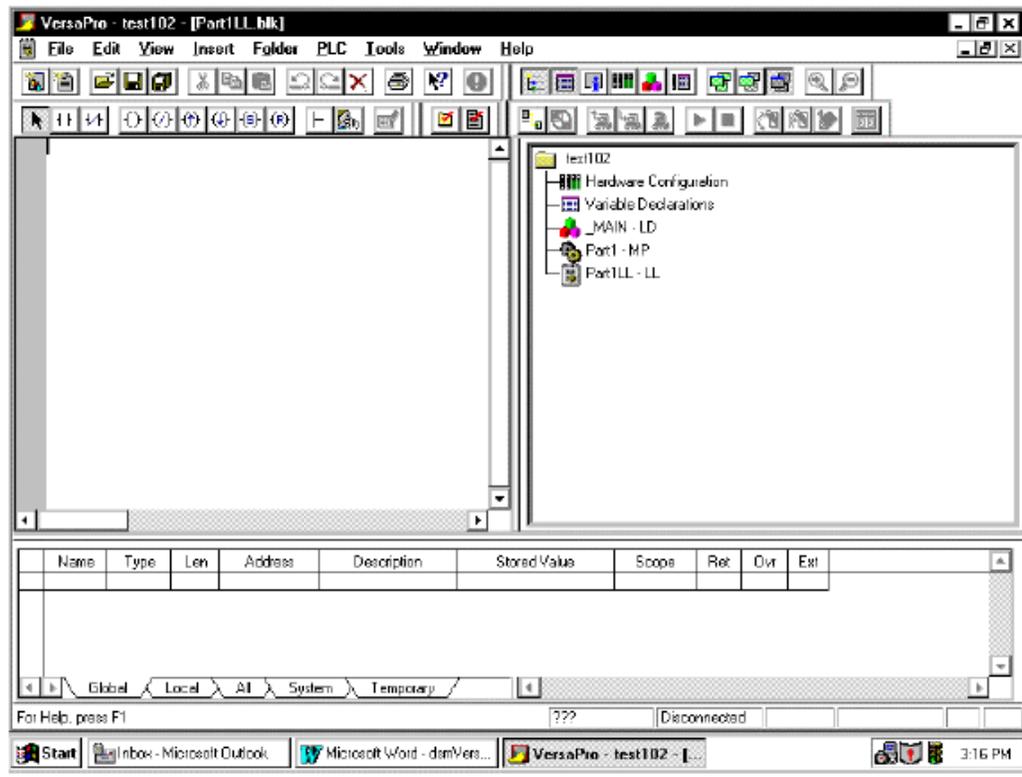
The Create New Local Logic dialog box will appear.

Figure 241: Create New Local Logic Dialog Box



- Type the Local Logic program Name and Description, then click the OK button (leave the Motion Module Type box set at its default DSM314 setting). A window for the new Local Logic program block will open. As shown in the next figure, the window title is based upon both the folder name, Test102 in this case, and Local Logic program name, Part1LL in this case. The Local Logic program in this example could not be called “Part1” because that name had already been used as the Motion Program block name. Notice also in the next figure that an icon for the new motion program, called “Part1LL – LL” (Local Logic), appears in the Folder Browser window on the right side of the screen.

Figure 242: New Local Logic Program Window



The text-based Local Logic program is created the left window (Local Logic Editor window). Details on Local Logic commands and syntax are covered in Chapters 10–14.

Saving your Local Logic Program

When ready to save your Local Logic file to your computer's hard disk, either click the Save icon on the tool bar (looks like a floppy diskette), or click File from the Menu bar and click Save.

Storing your Local Logic Program to the PLC

Since the Local Logic file is considered part of the Configuration file group, use the procedure under the heading “Connecting to and Storing Your Configuration to the PLC,” documented earlier in this appendix.

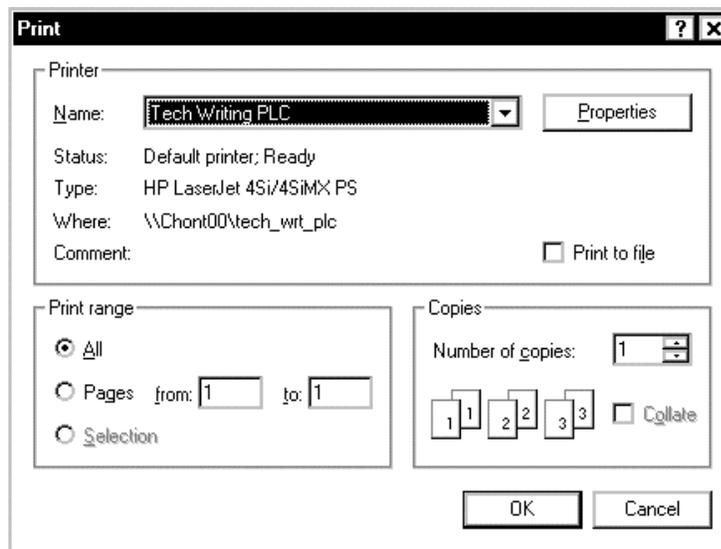
Printing a Hardcopy of your Local Logic Program

There are two print selections on the File menu: Print and Print Report.

Print:

- **Printing your entire Local Logic file (block):** While the Local Logic Editor is active, click File on the Menu bar and select Print. The Printer dialog box will display. Make any desired printer setup changes, then click the OK button.

Figure 243: Print Dialog Box

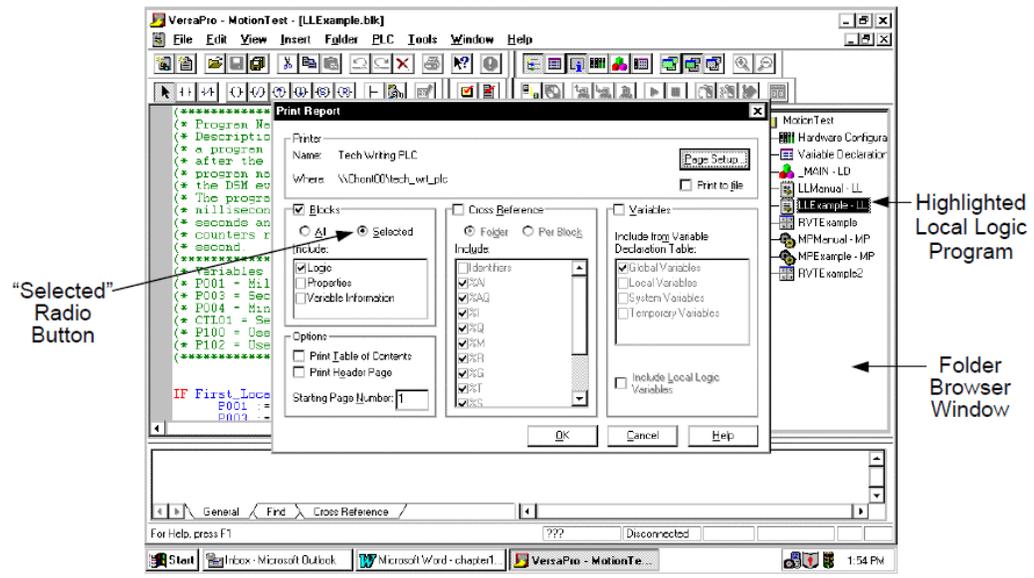


- **Printing just a selected portion of your Local Logic file:** In the Local Logic Editor window, use your mouse to select the portion you wish to print, click File on the Menu bar, then select Print. In the Print dialog box (shown above), make sure the Selection radio button in the Print range section is selected (has a dot in the middle). Click the OK button.

Print Report:

- To print all Local Logic blocks as part of a report with the other information in the folder, click File on the Menu bar and select Print Report. The Print Report dialog box will appear. Click the Blocks checkbox on the Print Report dialog box. Make sure the All radio button is selected. (You can also select other items and features for the report such as Table of Contents, Cross References, Variables, etc.) Click the OK button to begin printing. Motion program, Local Logic, and Ladder Diagram blocks will be printed as part of this report.
- To print only selected Local Logic blocks as part of a report, highlight the desired Local Logic blocks in the Folder Browser window. Click File on the Menu bar and select Print Report. Click the Blocks checkbox, then choose the Selected radio button. This limits the reports to only those blocks that you have highlighted in the Folder Browser window. See the next figure for an example of this.

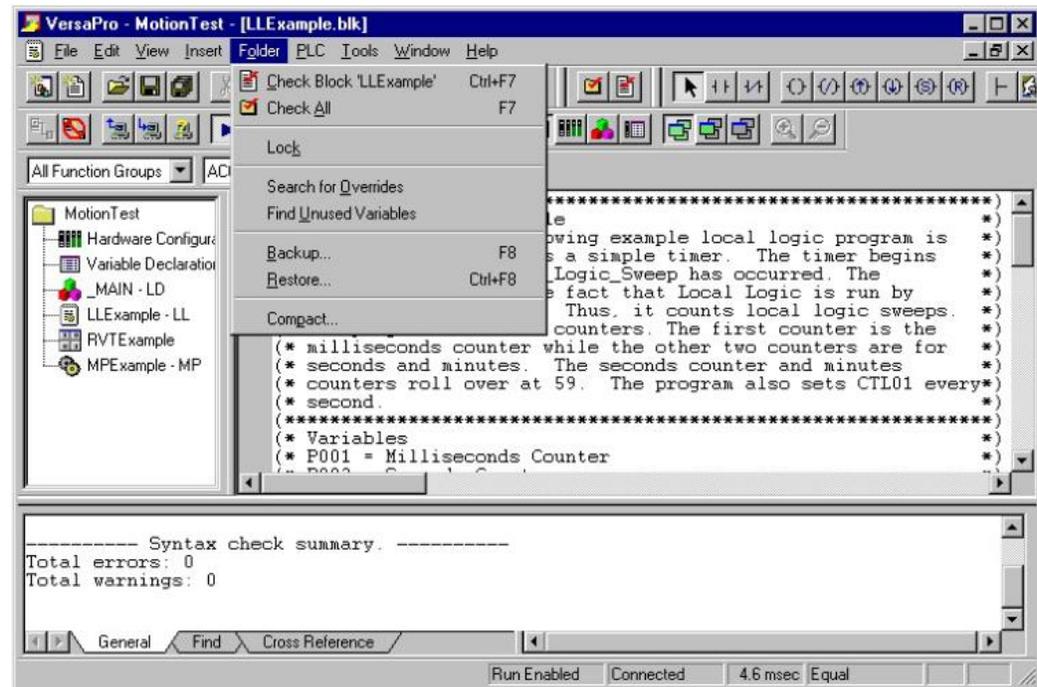
Figure 244: The Print Report Dialog Box



H-6.1 Checking Local Logic Syntax

To check the language syntax in VersaPro, select Folder from the main menu, then the submenu Check Block 'LLExample'. This causes the syntax check routines to run on the specified Local Logic program. Note: To check all the blocks within the folder, select Check All. (Figure 245.)

Figure 245: LLExample Syntax Check Command



Checking the blocks causes the information window to appear if it was not previously displayed. Note the Information window can be toggled on and off by pressing the  information toolbar icon.

The information window displays the output of the syntax check operation. If the sample program has been entered correctly, you should receive a message indicating zero errors and zero warnings.

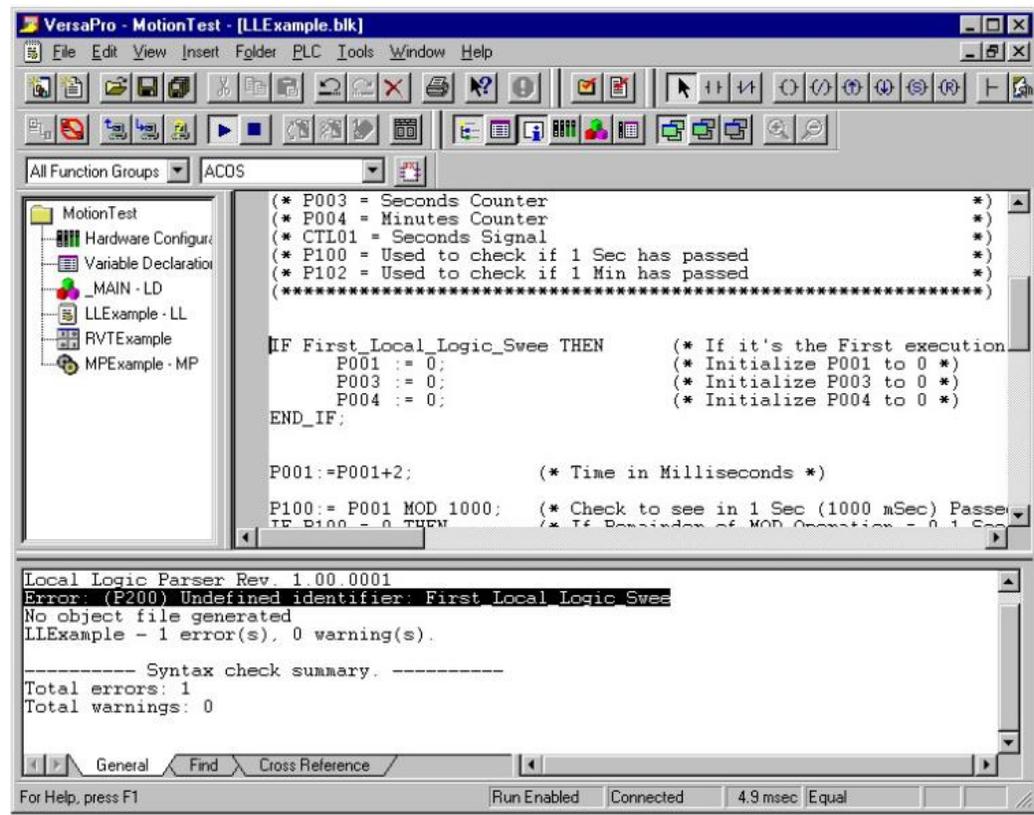
If the information window indicates a syntax error has occurred, you can scroll the information window to the line that contains the error message. While the information window has focus, double click the error message. This causes the editor window to automatically go to the line within the program that caused this error. For example, if in the example program you incorrectly typed “First_Local_Logic_Sweep” as “First_Local_Logic_Swee” a syntax error will be generated. As follows:

```
Error: (P200) Undefined identifier: First_Local_Logic_Swee
```

You can then go to the error message in the information window and double click the line. The Local Logic editor automatically goes to the beginning of the line that caused the error message so the user can fix the error.

If you are actively editing a local logic program and want to find out your approximate line number, click the vertical scrollbar on the right side of the local logic editor. It displays the line number of the line at the top of the window. If the cursor is several lines below the top of the window, either scroll the window until the current line is at the top of the window or count down the number of lines from the top and add them to the current line number.

Figure 246: LLExample Syntax Check Failure

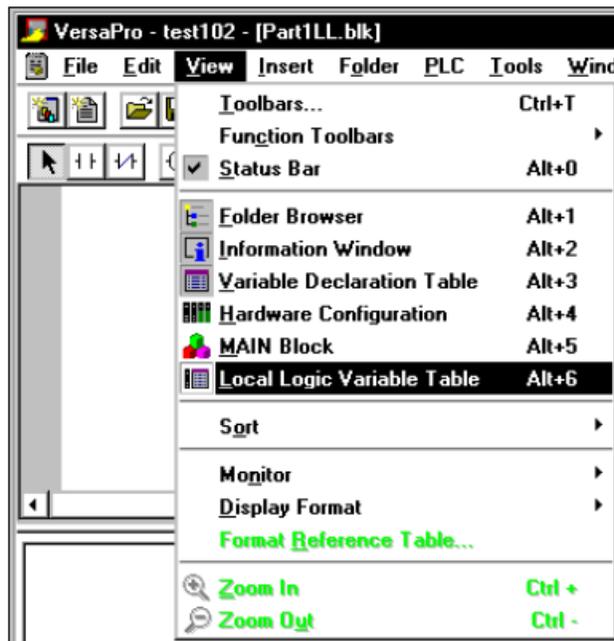


Chapter 12 contains details and corrective actions for syntax errors and warnings.

H-6.2 Viewing the Local Logic Variable Table

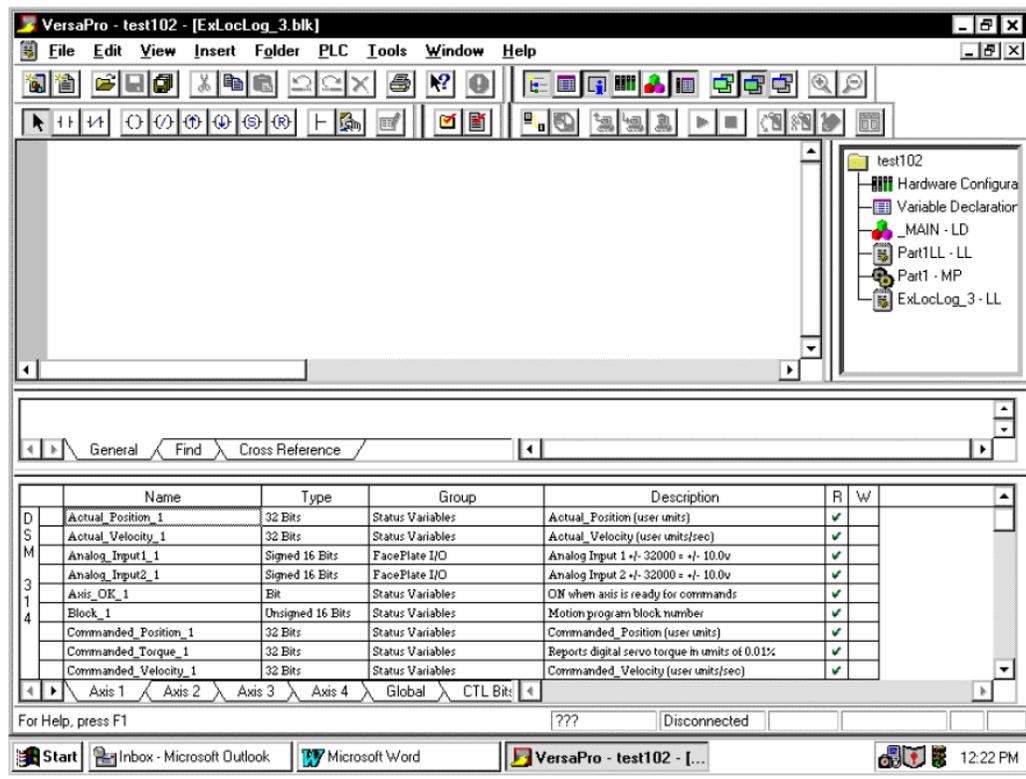
The Local Logic Variable Table appears in the Information Window area of the screen and contains information on the variables used in your Local Logic program. To use this feature, a Local Logic block must exist. If none exist, create a new one. To display the table, click View on the Menu bar, and select Local Logic Variable Table from the dropdown menu, shown in the following figure:

Figure 247: Selecting the Local Logic Variable Table from the View Menu



Once the Local Logic Variable Table appears near the bottom of the VersaPro screen, you can drag its top border or column borders to size them to your preference. See the next figure.

Figure 248: View Showing Local Logic Variable Table near Bottom of Screen



This table is useful when creating a Local Logic program because it allows you to copy and paste variable names, such as “Actual_Position_1,” into your program.

H-7 Creating a Cam Block

For information about Cam operation, refer to chapter 15.

Basic Steps

1. Open the project folder or create a new one
2. Create a CAM block
3. Create a CAM profile
4. Link the CAM profile to the CAM block
5. Configure the CAM profile
6. Specify the CAM Type
7. Specify the Correction Property
8. Save the CAM profile
9. Generate motion and Local Logic programs
10. Set up hardware configuration in the configuration/programming software
11. Execute (test) the application

Step 1: Create a Project

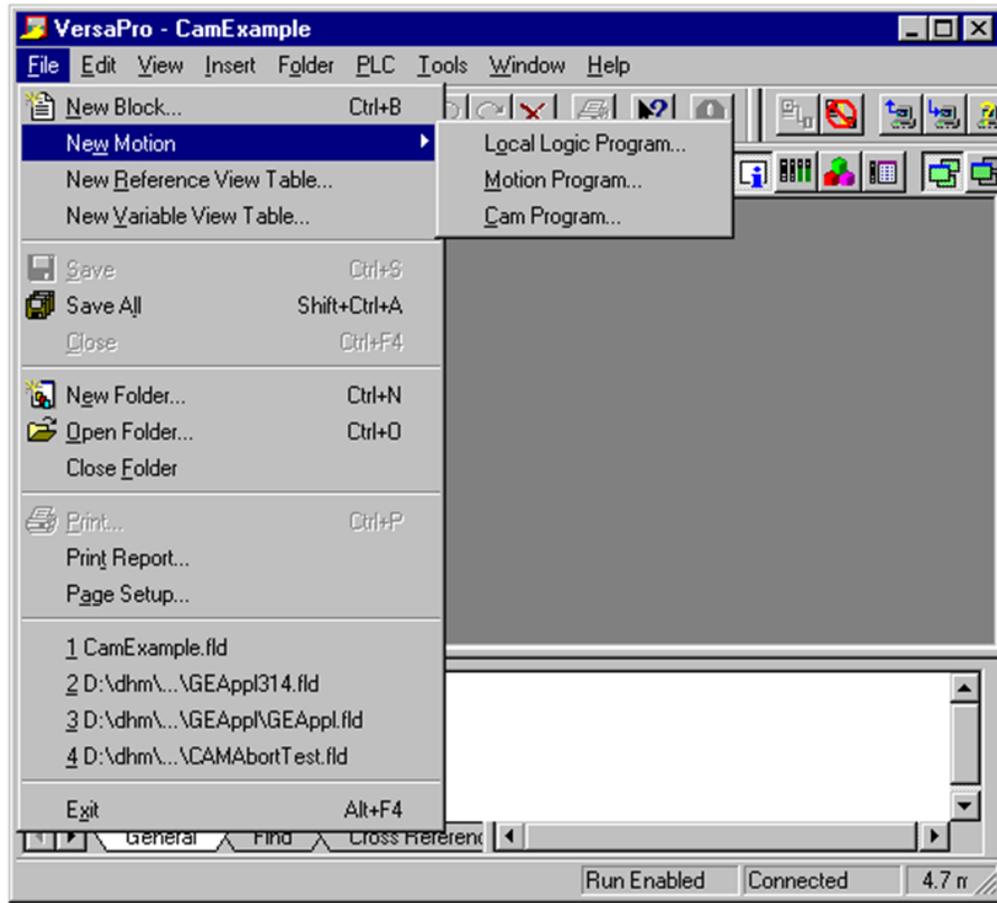
For details on creating a project, refer to the on-line help or the software user’s manual.

VersaPro™ Programming Software User’s Guide, GFK-1970

Step 2: Create a CAM Block Using the CAM Editor

The CAM editor is integrated into the VersaPro environment. The editor allows you to easily create, edit, store, and download CAM blocks. To create a CAM block, you must open or create a new VersaPro folder (see Step 1). Refer to the VersaPro User’s Manual, GFK-1670 for how to create or open a folder. Once the VersaPro folder is open, select the File menu selection, then the New Motion menu selection, followed by the CAM Program... menu selection. (Figure 249)

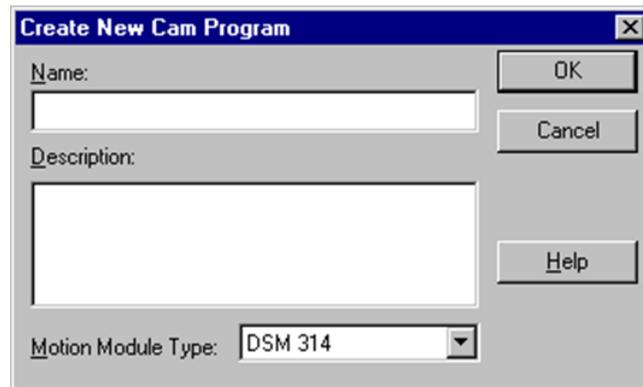
Figure 249: Create CAM Program



A “Create New Program” dialog box appears. Give the CAM block a name and an optional descriptive comment. At this time, the CAM feature is supported only on the DSM314 (release 2.0 or later). Therefore, the default selection for Motion Module Type should not be changed. (Figure 250). The rules for naming a CAM block are:

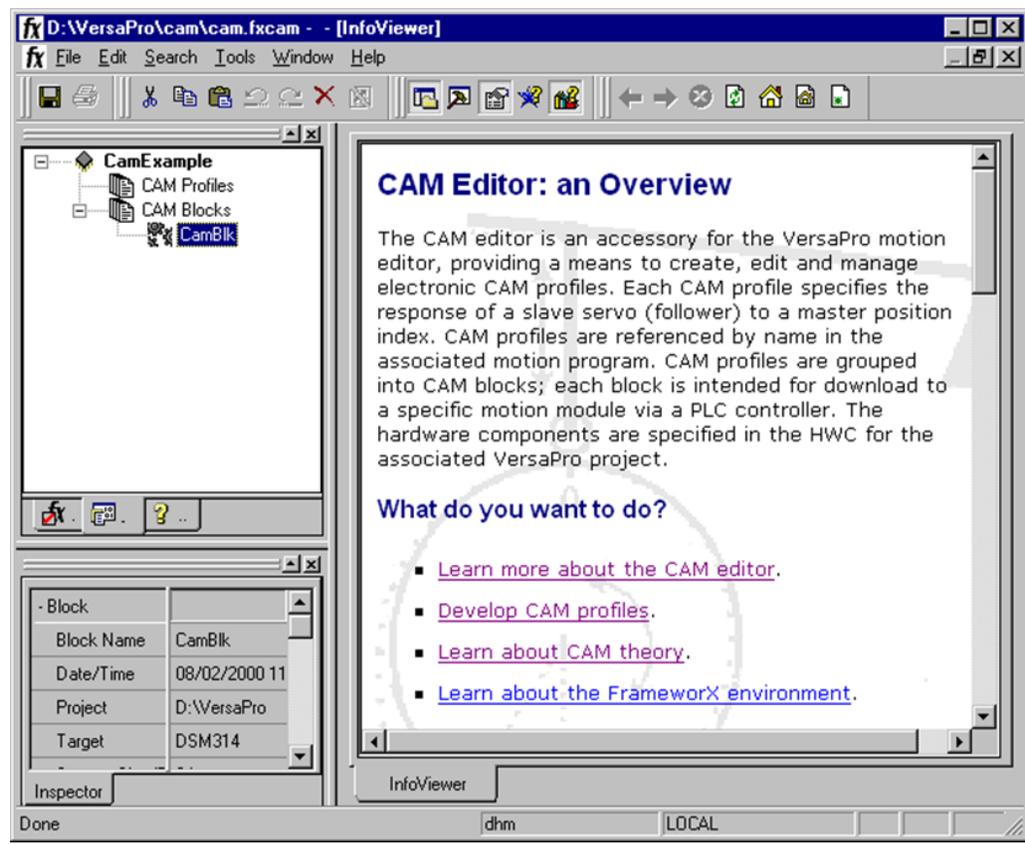
- Only the characters A-Z, a-z, 0-9, and _ (underscore symbol) are allowed. Consecutive underscores are not allowed.
- The block name must begin with a letter or underscore symbol.
- A block cannot have the same name as another block that exists in an open folder.
- A CAM block name may contain up to twenty characters.

Figure 250: Create New CAM Program



Enter the data in the “Create New CAM Program” box, then click the OK button to create the CAM Block. VersaPro launches the CAM editor program. (Figure 251)

Figure 251: Initial CAM Editor with InfoViewer Screen

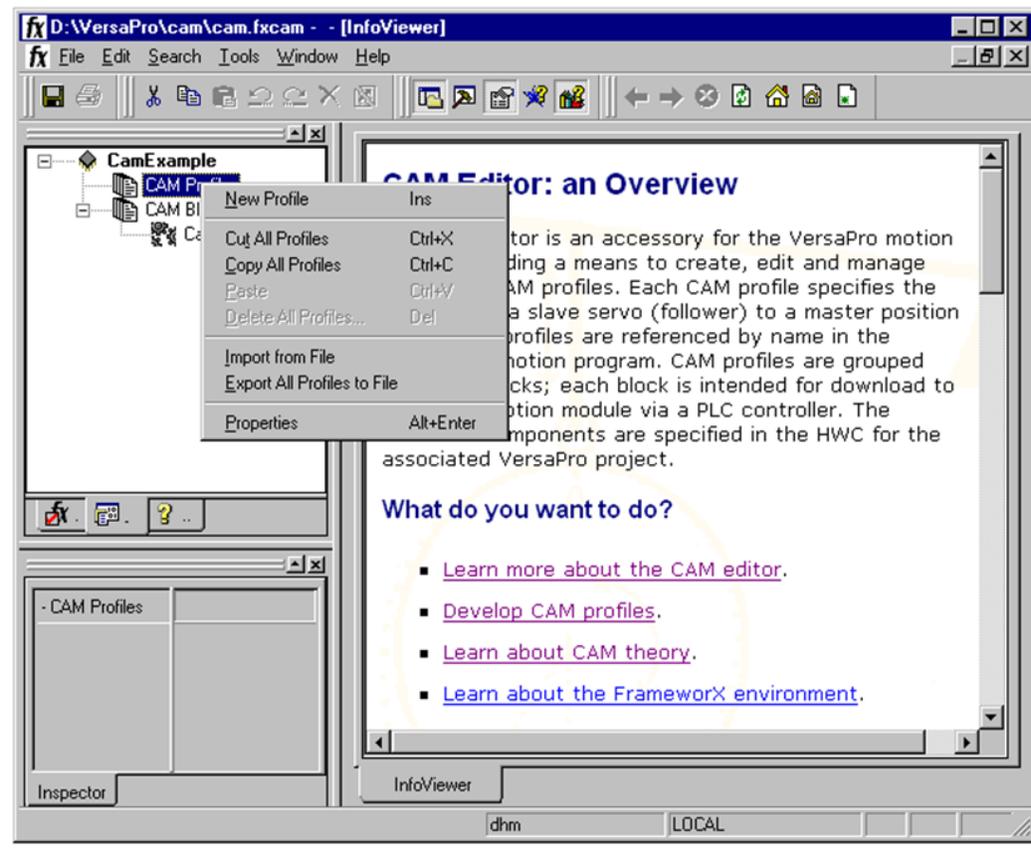


The CAM Editor contains extensive on-line hypertext help. This manual only attempts to introduce some of these concepts. It does not try to cover all the editor features, so it is strongly suggested that you review the programming software’s on-line material.

Step 3: Create a CAM Profile

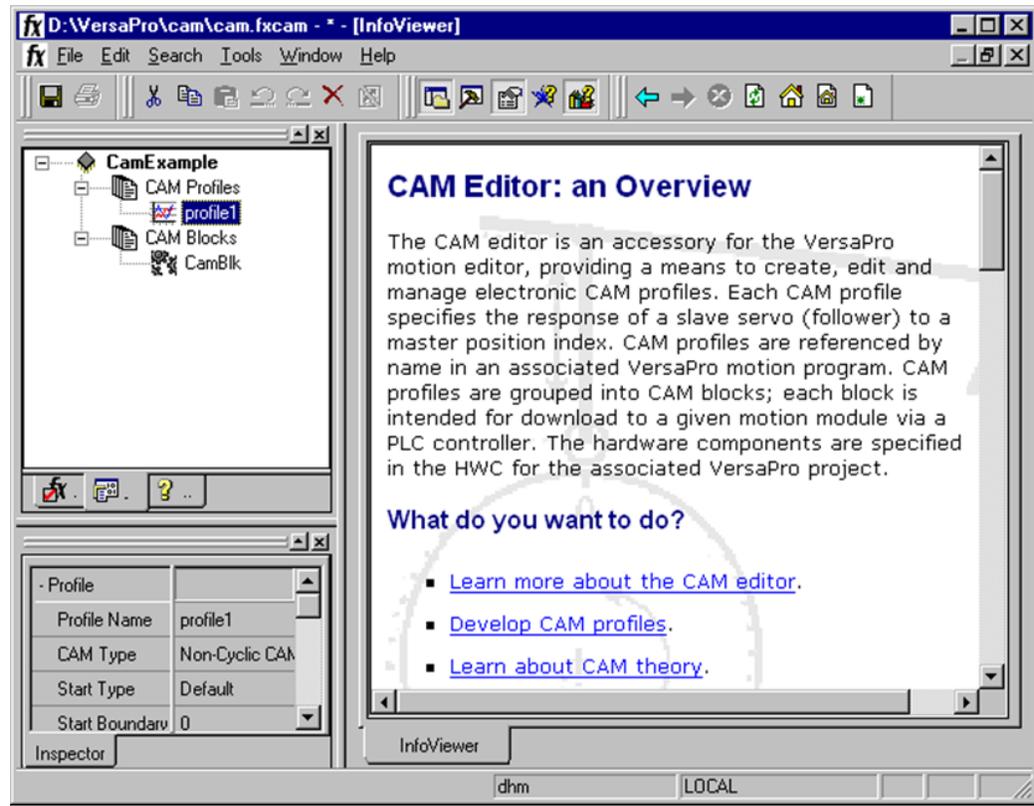
The next step is to create a simple CAM profile in the CAM editor. The CAM Editor has a CAM profile library that is created by the user. The CAM profiles within the library are then linked to the CAM blocks. Additional information on this interlinking is contained within the on-line help. For this example, a profile must first be created in the library. One method to perform this step is to right-click the “CAM Profiles” icon in the Navigator window. The short-cut menu appears. Select New Profile as shown in Figure 252.

Figure 252: Create New CAM Profile



This will insert a new profile into the library named “profile1” as seen in Figure 253.

Figure 253: New Profile Creation

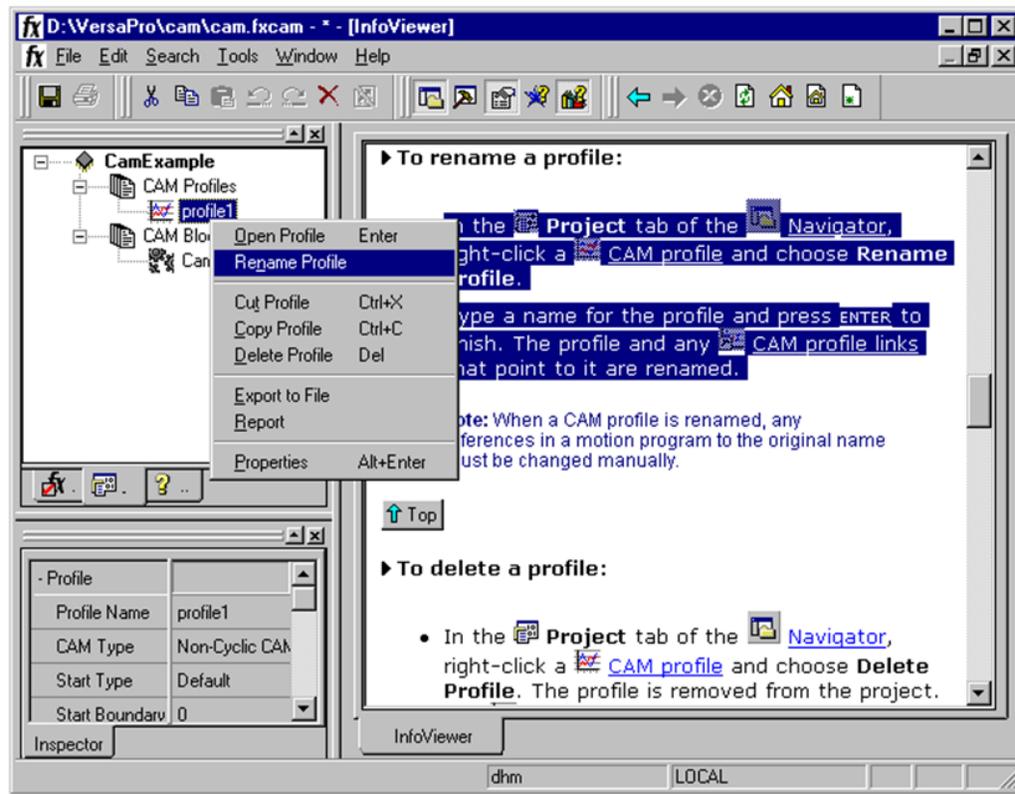


You can then rename this profile to a name more suitable to the application if desired. The naming rules are:

- Any alpha-numeric character or the underscore (_) symbol may be used.
- The first character in a profile name must be a letter.
- A profile name cannot be more than 20 characters long.
- A profile is referenced by name in a VersaPro motion program. NOTE: VersaPro is not case-sensitive when referencing a profile name.

One method to rename the profile is to right-click the profile name in the Navigator window and choose Rename Profile (Figure 254) from the short-cut menu. Type a name for the profile and press ENTER to finish. The profile and any CAM profile links to it are renamed. For this example, the profile is renamed to ExCamProfile. Refer to the on-line help for additional information.

Figure 254: Rename Profile



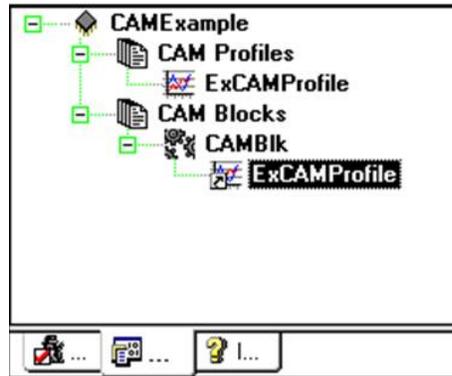
Step 4: Link the CAM Profile to the CAM Block

- CAM Profiles must be linked to their associated CAM block. A CAM block can contain numerous CAM profiles. The DSM has two limits that affect the number of profiles. The maximum CAM block size is 50K, and the maximum number of linked profiles for an individual block is 100. The CAM Profile library is only limited by available disk space on the host computer. The CAM block is linked to the DSM via the CAM Block entry in Hardware configuration.

Although there is more than one way to link a CAM profile to a CAM block, the easiest method is simply to drag and drop the desired CAM profile onto the applicable CAM block. The result is shown in the next figure.

Note: VersaPro limits the download block total size (Motion, Local Logic, and CAM combined) to 32K.

Figure 255: Linking a Profile to a CAM Block



Step 5: Configure CAM Profile Data Points

Once these operations are complete, you must configure the CAM profile. The CAM profile is a relationship between the master position and the slave position. A CAM profile is composed of a series of Points. Each point is defined by two coordinates. In the graphical representation, the Master coordinate represents the horizontal axis and the Slave coordinate represents the vertical axis, as shown in the next figure.

Begin by double-clicking the profile to open it in the Profile Editor window (see next figure), which has two editors:

- **The Table Editor** is similar to a spreadsheet. In the table, each point has its own row with two columns, one for the Master position and one for the corresponding Slave position. When a new profile is opened, there are, by default, only two points, a start point and an end point. The start point is the top point of the table and the end point is at the bottom.
- **To edit points with the Graphical Editor**, click the point on the graph and drag it to the desired location. The point data in the table editor will update to the new position. To perform other tasks in the graphical editor, right-click in the graph and select the applicable task from the short-cut menu.

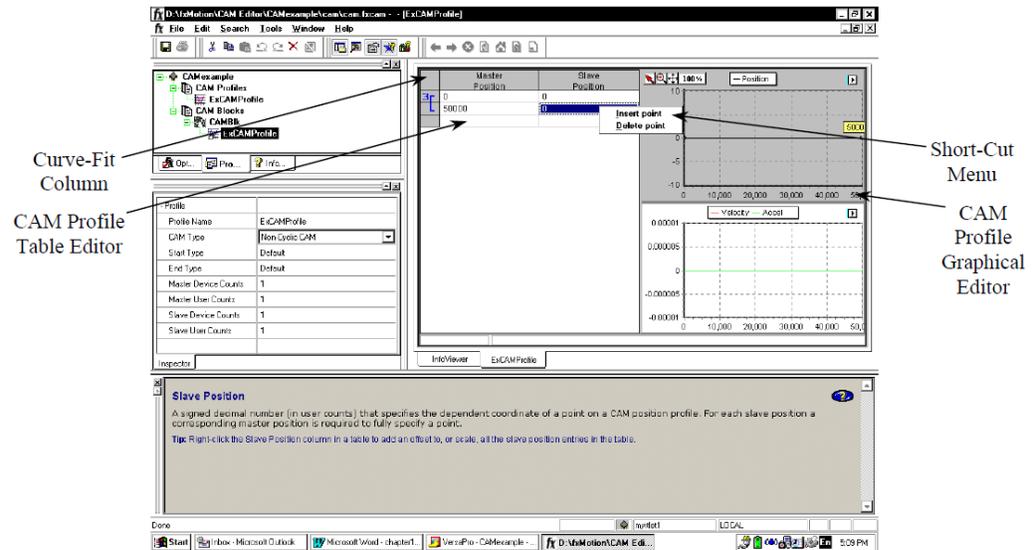
The next step is to edit the end point (the bottom point in the table) for the Master and Slave. In the Table Editor, click in the end point's Master column and enter the value 50000; then click in the end point's Slave column and enter the value 0. (NOTE: As points are added or changed in the Table Editor, the graph in the Graphical Editor will update accordingly.)

Next, insert an additional point into the Editor table. Right-click in the Master column of the end point and choose Insert Point from the short-cut menu (shown in the next figure). A new row is added above the end point row, specifying a new point with master and slave values, by default, midway (25000 and 0, respectively) between the values of the two existing adjacent (above and below) points. Change the values for this point to 47500 for the Master and 11000 for the Slave. To change a point value, click it, type in the new number, then either press the Enter key, or click outside of the table.

To change the Curve-Fit order, click the Curve-Fit column, then select the Curve-Fit Order in the Property Inspector window. Also, a profile can be split into multiple sections or multiple sections merged into one by right clicking on the Curve-Fit display and choosing from the short-cut menu.

Note: A CAM profile is limited to 400 points if it contains second or third order sectors. A CAM profile is limited to 5000 points if it only contains first order sectors.

Figure 256: Inserting a Point in the Profile Editor Window



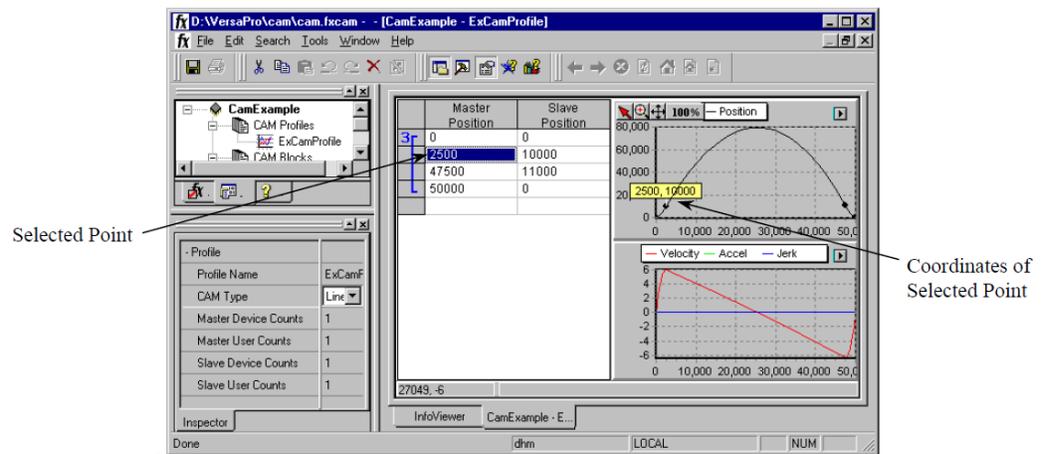
Since the Slave Position end point is the same value (0) as the initial Slave Position point, this CAM meets the requirements for a Linear Cyclic CAM. (If desired, refer to Section 2 for more information on the different CAM types.) Note that the CAM Editor has several “Smart” edit fields that will ONLY display the choices that are valid for a given data set. For example, since a requirement for a Linear Cyclic CAM is that the Slave Position start point and Slave Position end point are the same, the editor only allows the Linear Cyclic CAM choice if these criteria is met.

Next, insert a new point into the profile and then edit the point. The point can be edited either in the profile table or graphically on the plot. Insert the point as shown above and in Figure 47HH-9 and Figure 48HH-10 by right-clicking the point below the insertion position and selecting Insert Point from the menu. Then change the default values to 2500 for the Master and 10000 for the Slave.

Figure 257: CAM Profile Table Data

	Master Position	Slave Position
3r	0	0
	2500	10000
	47500	11000
	50000	0

Figure 258: CAM Editor Example



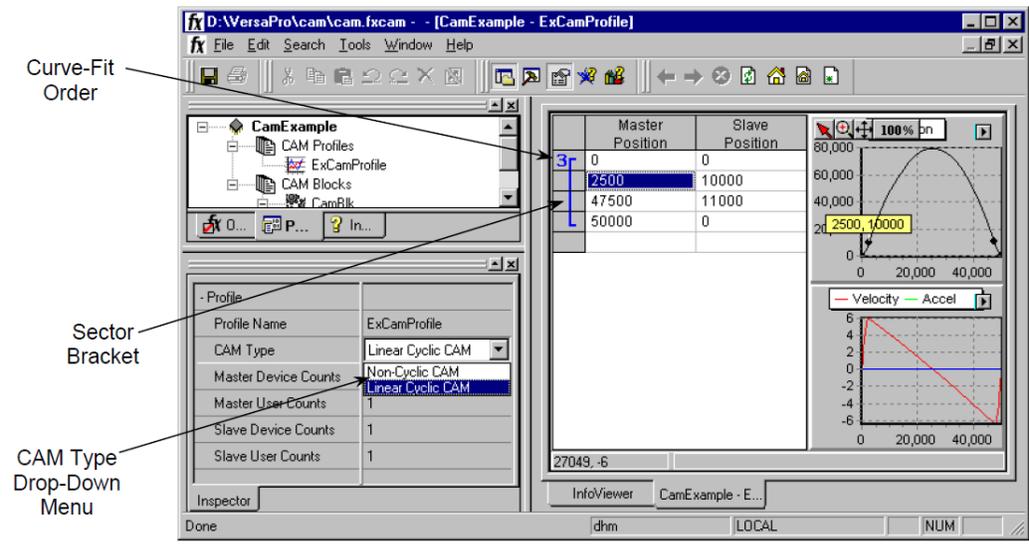
There are numerous other features in the editor. These include being able to define additional sectors that each have a different curve fit method. These editor features are discussed in the programming software's on-line help. Please reference this source for additional information.

Step 6: Specify the CAM Type

For this example, the CAM will be Linear Cyclic, as discussed previously. Use the following procedure:

- In the Project tab of the Navigator, right-click a CAM profile. The short-cut menu appears.
- From the short-cut menu, choose Properties. The Inspector opens showing the CAM profile's properties.
- In the Inspector, click the arrow in the CAM Type field. The CAM Type drop-down list appears.
- Choose 'Linear Cyclic CAM' from the list (Figure 289).

Figure 259: CAM Editor CAM Type Selection

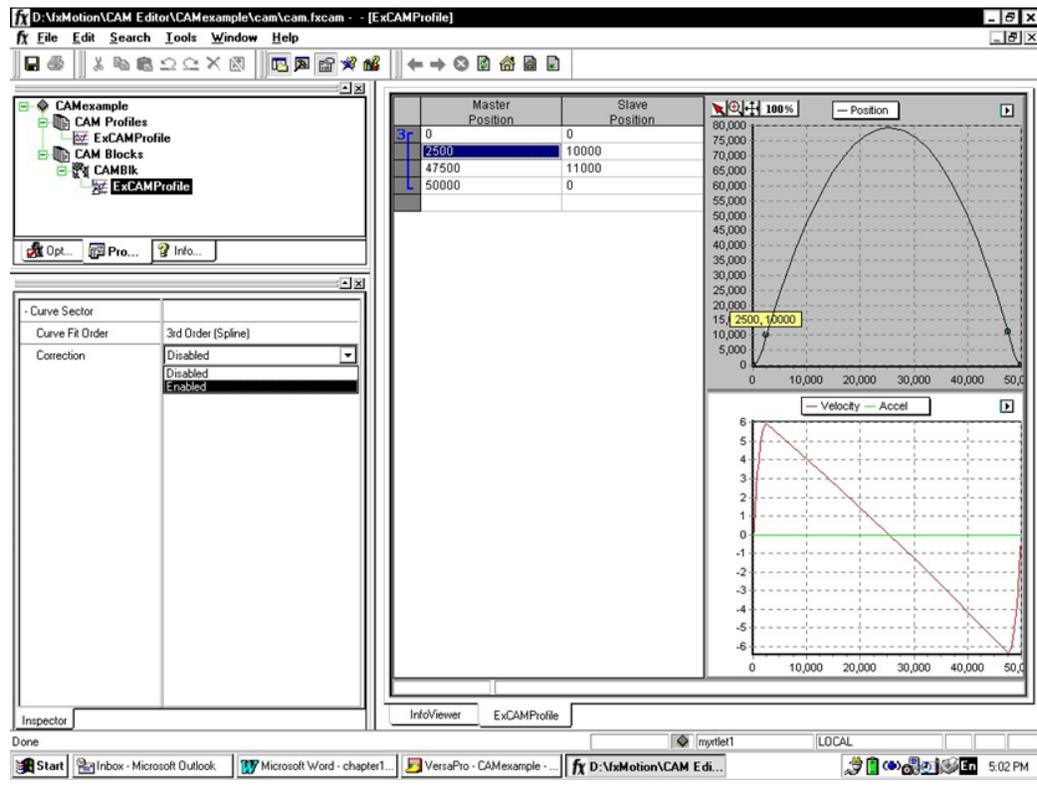


Step 7: Specify the Correction Property

The last item to be specified for this example is the correction status. The Correction property determines whether the motion module will permit an online correction for a specific sector. A sector is a region of a CAM profile defined by at least two adjacent user-defined points. The sector includes the user-defined points, the curve connecting them and also up to, but not including, the first point defined for the next adjacent sector. The points included in a given sector are denoted by the Sector Bracket, shown in the figure above. Each sector is assigned a curve-fit order number, also shown in the figure above. The segments of the profile between user-defined points are defined by polynomials of the curve-fit order specified. A unique polynomial is used to interpolate between each pair of adjacent user-defined points. Although the actual polynomial coefficients can be different for each segment, the curve-fit order is the same throughout the sector. A sector is indicated in the CAM profile table as a bar spanning the user-defined Master Position values included in the sector. Initially, all points defined in a profile are included in a single sector. This single initial sector can be subdivided as required to facilitate smoothing a CAM profile. When the Correction property is Enabled, the motion module reports a warning if there is a velocity limit violation. When the Correction property is set to Disabled, the motion module reports an error for these violations and stops the slave axis.

For this example, correction should be enabled. To enable correction, select the sector from the CAM profile table by clicking it. This will cause the Inspector window to display the sector properties and allow them to be edited. Select the Correction drop down box and choose Enabled (Figure 260).

Figure 260: CAM Editor Correction Enable



Step 8: Save the CAM Profile

At this point, a simple CAM profile is defined. To save the CAM blocks/profiles, select the File main menu item followed by the Save Project submenu selection. You could also select Exit, which causes an automatic save. The CAM editor has many more additional features and functionality. Refer to the online documentation for a detailed description of these features.

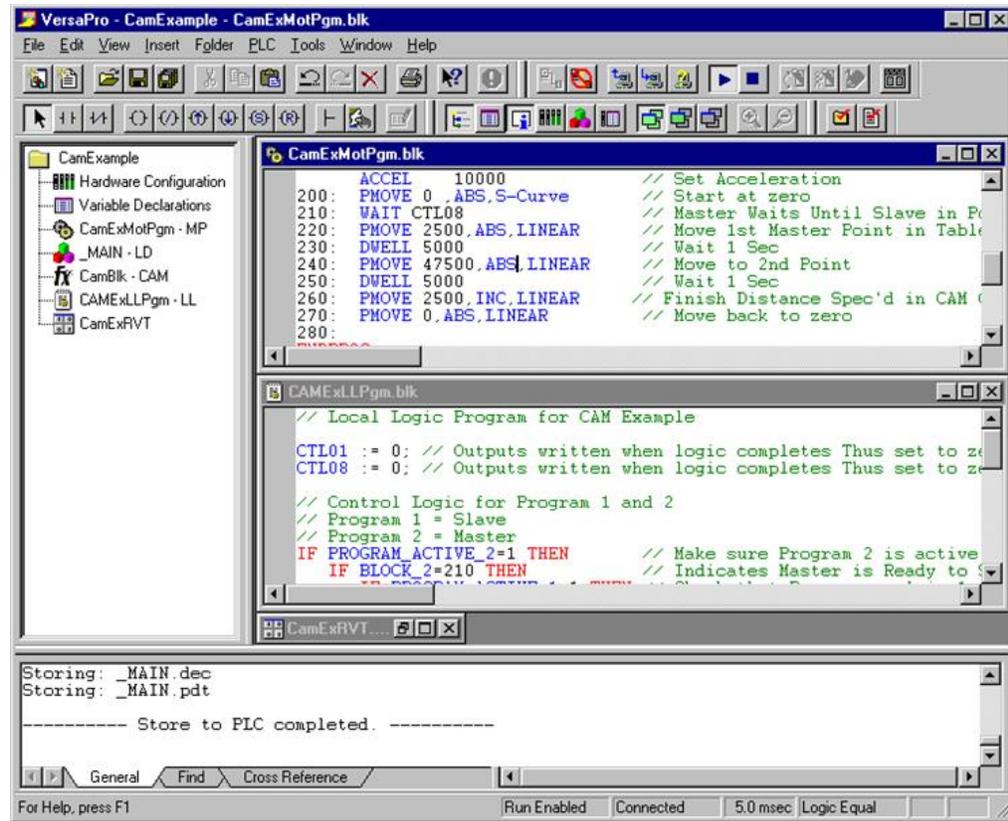
Step 9: Generate Motion and Local Logic Programs

The next items to be generated are a motion program and Local Logic program that will work with this CAM profile. For this example, the logic must work with a DSM314 controlling two axes. Axis #1 will be the slave, and Axis #2 will be the master. Therefore, there will be two motion programs. The Axis 1 program, for the slave, will do some base initialization, load the slave starting point for the given CAM profile, and then execute the CAM command. The Axis 2 program, for the master source, is a simple program that will initialize and then wait for the slave to be ready. It will then execute a series of moves. The program stops at points described within the CAM master such that it is easy to verify that the slave axis is correctly executing the CAM profile. This example also requires a Local Logic program. In this example the Local Logic program serves a supervisory role over the CAM slave and CAM master motion programs. Thus, the Local Logic synchronizes the two programs.

Consult the applicable chapters in this manual for additional details on these features. The motion program and Local Logic programs for this example are provided in chapter 15.

After completing the program entry, the resulting VersaPro screen should look similar to the figure shown below (Figure 261).

Figure 261: CAM Example VersaPro Screens



Step 10: Set up Hardware Configuration in VersaPro

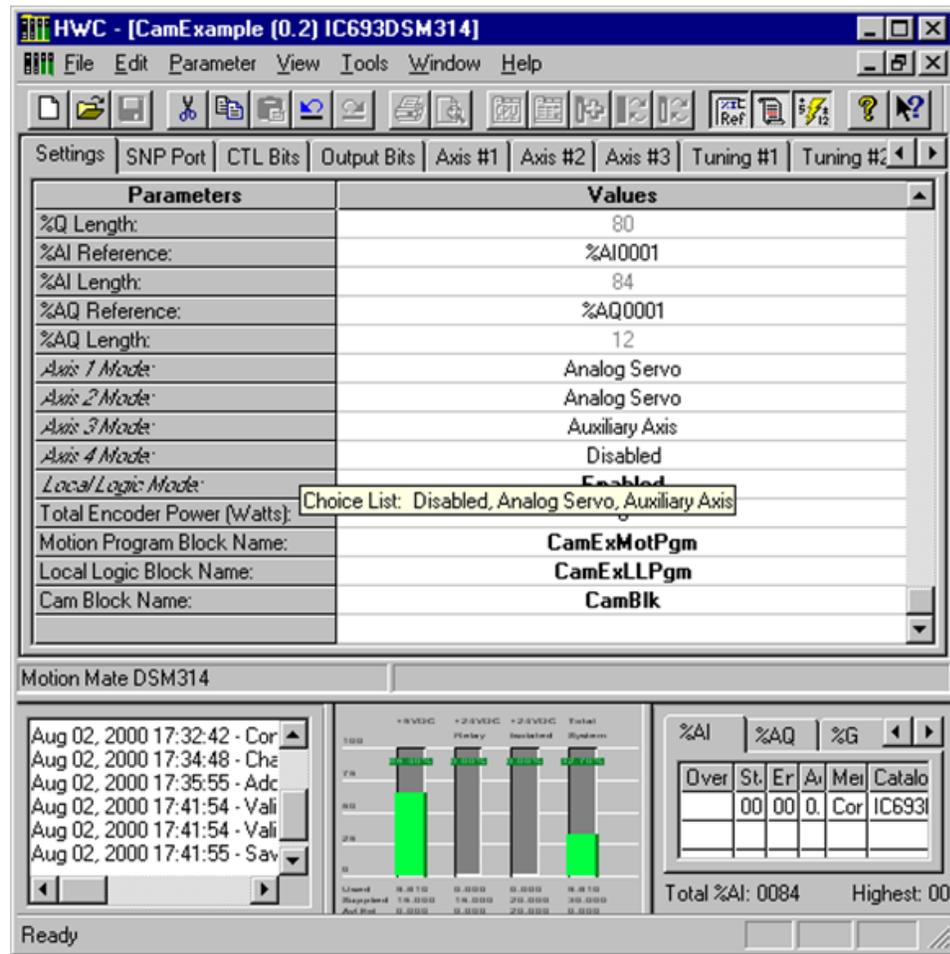
Change the following Settings tab parameters to the values shown. (Axis 1 and Axis 2 modes are set to digital servo because this example uses the β is 0.5 digital servo.)

Axis 1 Mode	Digital Servo
Axis 2 Mode	Digital Servo
Local Logic Block Name	CamExLLPgm
Cam Block Name	CamBlk
Local Logic Mode	Enabled

Note: This example uses only one DSM314. The DSM314 executes the files (CAM, Local Logic, and Motion Program) pointed to by the configuration. Multiple DSM314 modules can run the same Local Logic program, motion programs, or CAM Blocks. This allows you to have one source file for multiple DSM314 modules. Note that this does not prevent DSM314s from executing different programs.

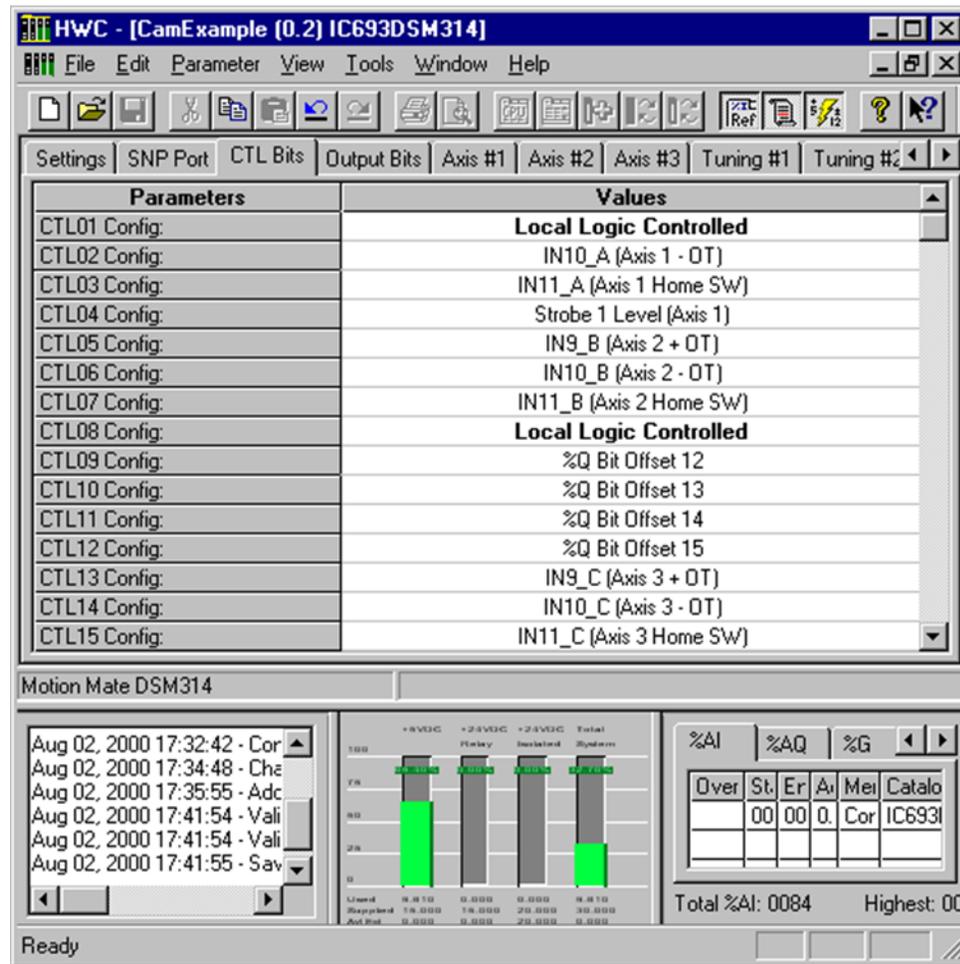
The resulting Settings tab will be as shown in Figure 262.

Figure 262: Hardware Configuration 90-30 rack DSM314 Settings Tab



In this example, the Local Logic program will control CTL01 and CTL08. Because CTL01 and CTL08 are used to signal the Motion Programs, you must configure these CTL bits to be under Local Logic Control. To do this, access the CTL Bits tab in the VersaPro hardware configuration. Select “CTL01 Config” and choose Local_Logic_Controlled. Repeat the procedure for CTL08. The resulting Hardware Configuration screens are shown in Figure 263.

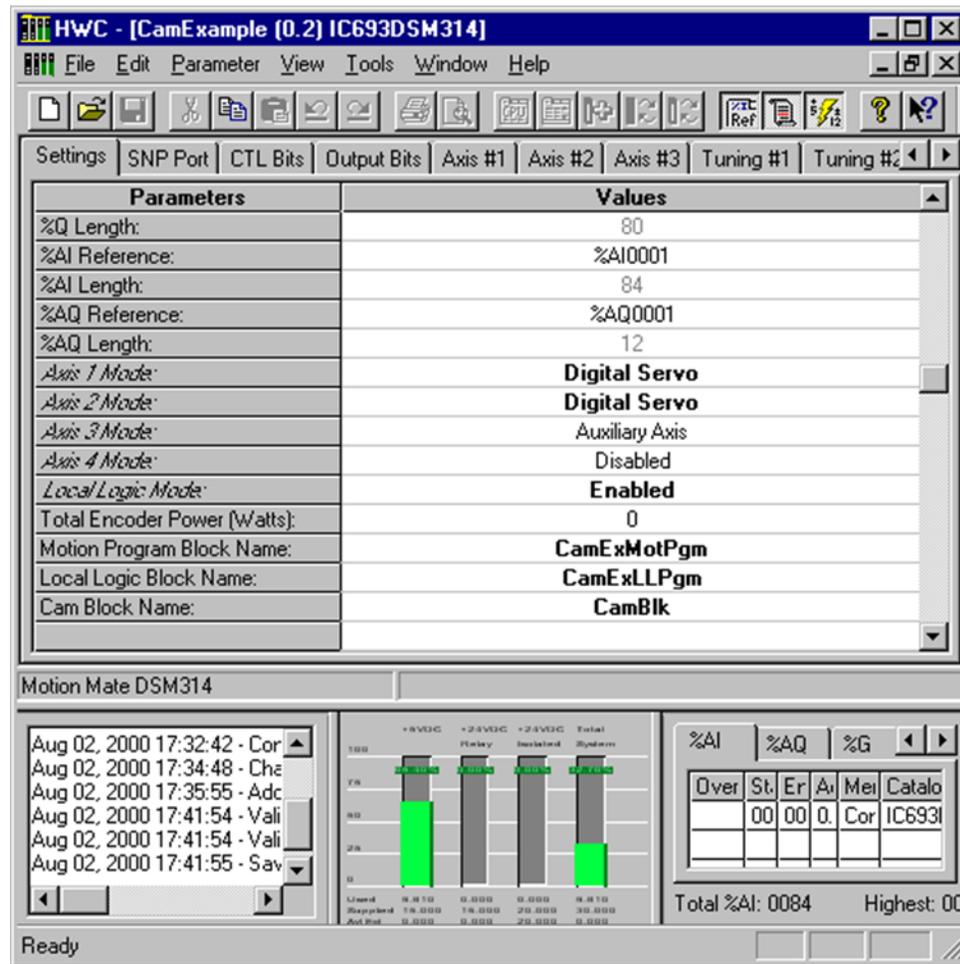
Figure 263: Hardware Configuration 90-30 rack DSM314 CTL Bits Tab



Since this example uses the Beta 0.5 digital servo, Axis 1 and Axis 2 Mode should be set to Digital Servo.

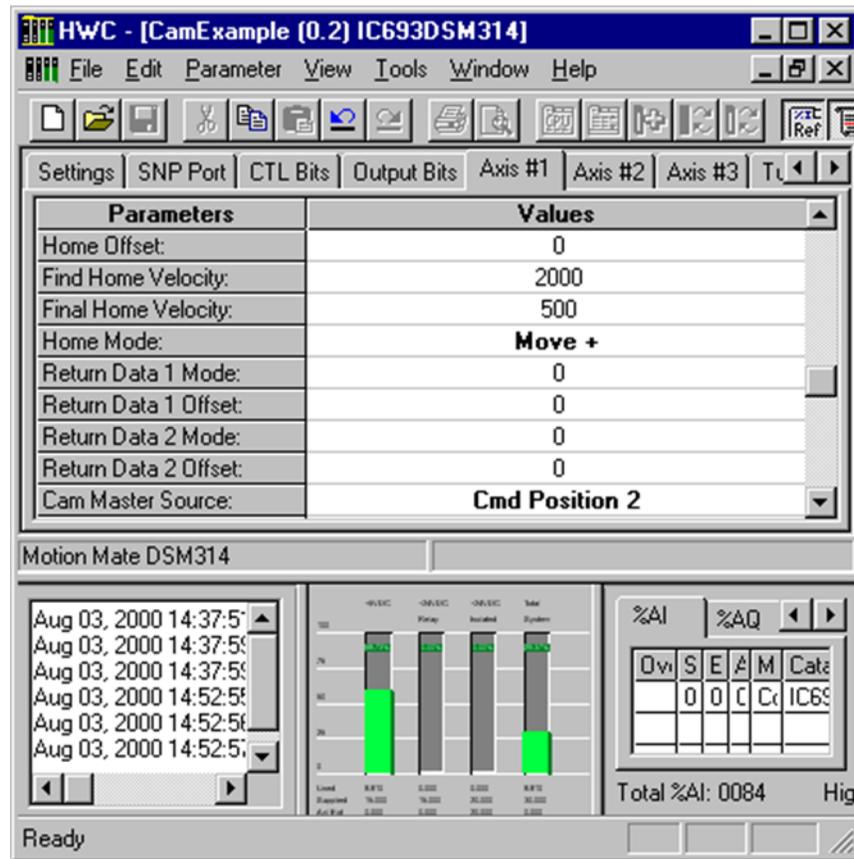
The resulting Hardware Configuration screens are shown in Figure 264.

Figure 264: Hardware Configuration DSM314 Settings Tab



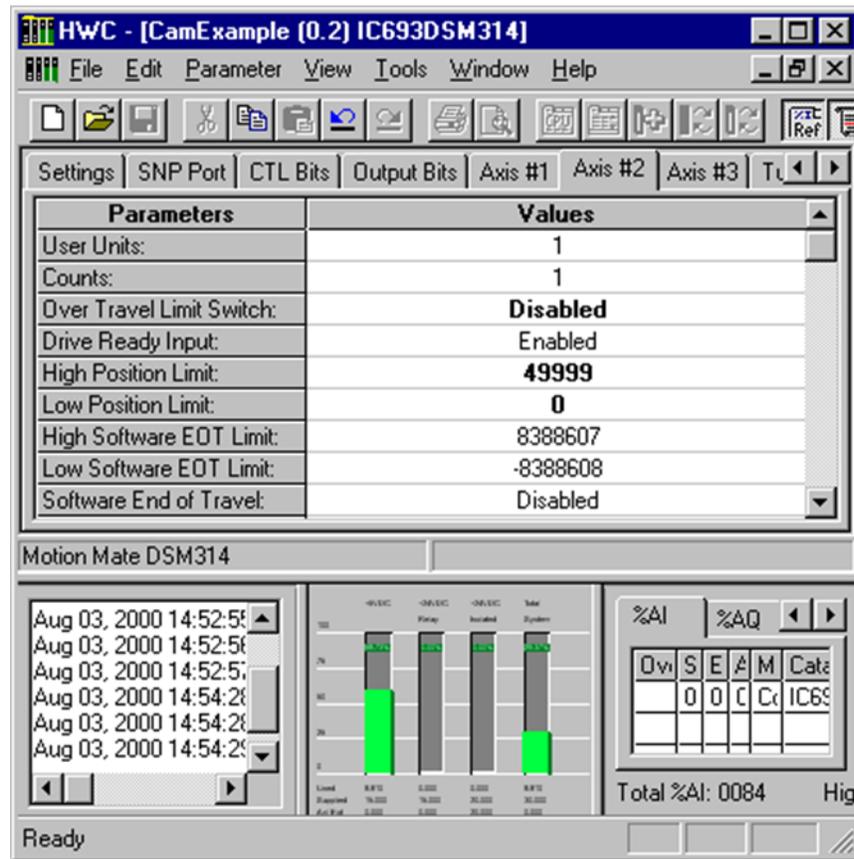
You also need to indicate to Axis #1 that it will use the Axis #2 commanded position as its CAM Master source. To do this select, the Axis #1 tab in hardware configuration. Go to the CAM Master Source data entry field. From the drop-down box, select Cmd Position 2. This will configure Axis #1 to use the Axis #2 commanded position as it's CAM master source (Figure 265). While in this tab, change the Home Mode: to Move + and OverTravel Switch to Disabled.

Figure 265: CAM Slave Master Source Selection



You also need to indicate to Axis #2, the rollover points for the Master axis position reference. To do this, select the Axis #2 tab in hardware configuration. Input 49,999 into the High Position Limit and 0 into the Low Position Limit data entry fields (Figure 266). Note that since this is a Cyclic CAM, the master source high limit, by definition, must be one less than the last point in the master data table. In this example, this is point 50,000. Thus, the high limit is equal to 49,999. One way to envision this principle is to think of a Cyclic CAM Master as a continuous circular strip where the first point on the strip is the same as the last point on the strip. Thus, for this example, 50,000 is the same point as zero. While in this tab, change the Home Mode: to Move + and OverTravel Switch to Disabled.

Figure 266: CAM Master Axis Scaling



To finish the configuration, you need go to the Tuning#1 and Tuning #2 tabs and enter the following data:

- **Motor Type:** 13
- **Position Error Limit:** 200 (Optional; see Configuration information for additional information)
- **In Position Zone:** 20 (Optional; see Configuration information for additional information)
- **Pos Loop Time Const:** 200

Note: (Based upon application/mechanics reference Chapter 4 and Appendix D)

- **Velocity FeedForward:** 9000

Note: (Based upon application/mechanics reference Chapter 4 and Appendix D)

- **Vel Loop Gain:** 32

Note: (Based upon inertia attached to motor. The typical Beta Demo case has an indicator wheel attached that represents approximately this inertia to a Beta 0.5)

The resulting display should be similar to Figure 267.

Figure 267: Hardware Configuration Tuning#1 Tab

Parameters	Values
Motor Type:	13
Analog Servo Command:	Velocity
Position Error Limit:	200
In Position Zone:	20
Pos Loop Time Constant (0.1ms):	200
Velocity at Max Cmd:	100000
Velocity Feed Forward (.01%):	9000
Acceleration Feed Forward (.01%):	0
Integrator Mode:	Off
Integrator Time Constant (ms):	0
Velocity Loop Gain:	32

Motion Mate DSM314

The Tuning tab for Axis #2 should also be set up as shown for Axis #1.

The link between the example CAM Block, Motion program, Local Logic program, and the DSM314 module is now complete. Create any required PLC ladder logic programming, then perform a Check All on the programs and download them to the PLC. Additional information concerning the download operation is provided in the VersaPro manual, GFK-1670, or the on-line help.

Step 11: Execute (Test) Your CAM-Based Motion Program

⚠ WARNING

Before testing your application on actual machinery, you must first verify that it is safe to do so. This includes insuring that all devices are securely mounted, all safety equipment is installed and operational, and personnel in the area have been notified. Failure to address all safety-related issues could result in injury to personnel and damage to equipment.

Once the download operation is complete, the module is ready to execute the CAM Blocks, motion programs and Local Logic program. Use the following procedure:

1. Place the PLC in run mode.
2. Enable the servo drives. To enable Axis #1, toggle the %Q offset 18 bit. To enable Axis #2, toggle The %Q offset 34 bit. Based upon the current module error status, you may also have to initiate a clear error routine by toggling the %Q offset 0 bit.
3. Have both axes perform a find home routine by toggling the %Q offset 19 bit (find home Axis #1) and the %Q offset 35 bit (find home Axis #2). At this point, both axes will perform a find home cycle. Wait until this completes for both axes and the Position Valid %I bits turn on. The Position Valid %I bit for Axis 1 is the %I offset 17 bit

(the 18th %I bit), and for Axis 2 is the %I offset 33 bit (the 34th %I bit). The resulting display is shown in Figure 268.

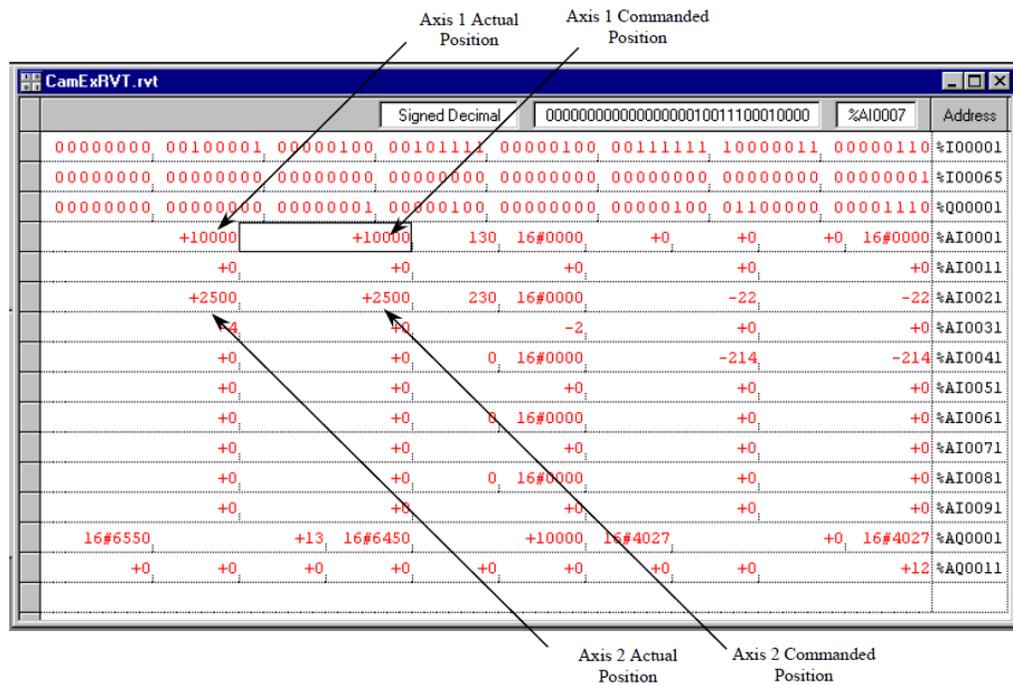
Figure 268: RVTEExample Screen

The screenshot shows a software window titled 'CamExRVT.rvt'. At the top, there are two input fields: 'Binary' with the value '00000000' and '%Q00002' with the value '00000000'. Below these is a table with 8 columns of binary data and an 'Address' column. Two arrows point to specific bits in the first row: one to the 18th bit (labeled 'Axis 2 Position Valid Bit') and one to the 34th bit (labeled 'Axis 1 Position Valid Bit'). Below the binary data is a table of numerical values in hexadecimal and decimal formats, corresponding to the addresses in the first table.

Binary	%Q00002	Address
00000000, 00100001, 00000100, 00100101, 00000100, 00100101, 10000011, 00000100	00000000	%I00001
00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000000, 00000001		%I00065
00000000, 00000000, 00000001, 00000100, 00000000, 00000100, 01100000, 00000000		%Q00001
+0, +0, 0, 16#0000, +0, +0, +0, 16#0000		%AI0001
+0, +0, +0, +0, +0, +0, +0, +0		%AI0011
+0, +0, 0, 16#0000, +3, +3, +3, +3		%AI0021
+0, +0, +0, +0, +0, +0, +0, +0		%AI0031
+0, +0, 0, 16#0000, -141, -141, -141, -141		%AI0041
+0, +0, +0, +0, +0, +0, +0, +0		%AI0051
+0, +0, 0, 16#0000, +0, +0, +0, +0		%AI0061
+0, +0, +0, +0, +0, +0, +0, +0		%AI0071
+0, +0, 0, 16#0000, +0, +0, +0, +0		%AI0081
+0, +0, +0, +0, +0, +0, +0, +0		%AI0091
16#6550, +13, 16#6450, +10000, 16#4027, +0, 16#4027, +0		%AQ0001
+0, +0, +0, +0, +0, +0, +0, +0		%AQ0011

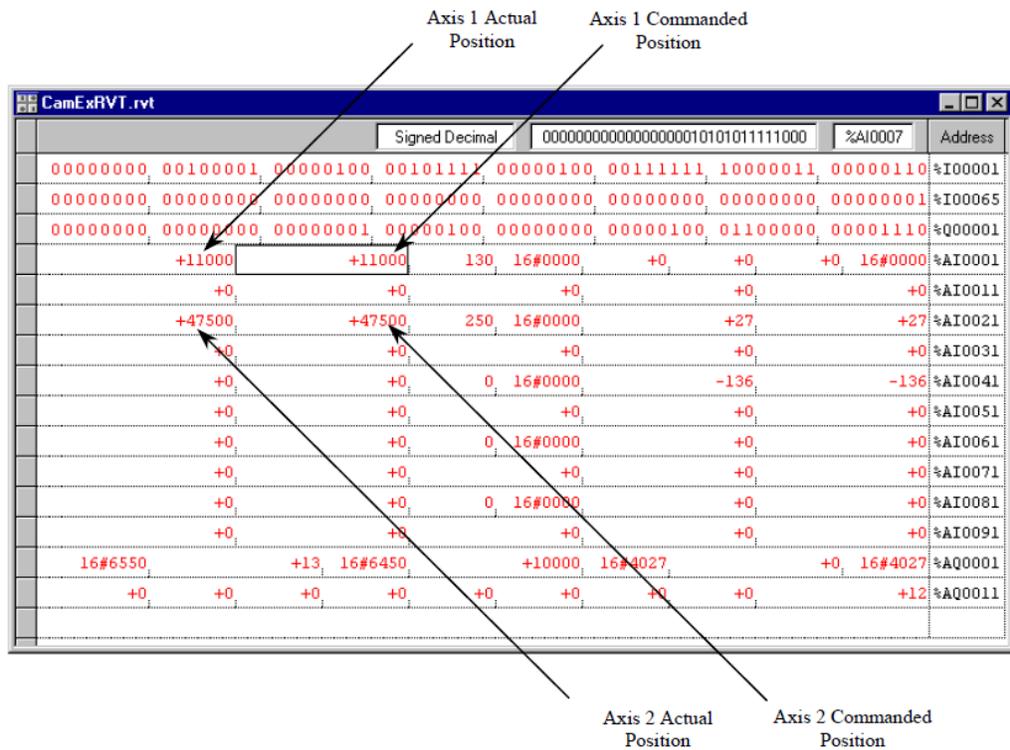
4. Enable Local Logic by setting the %Q offset 1 bit from the PLC. If there are no errors, you can then execute the motion programs.
5. Execute Program 1 by toggling %Q offset 2 bit. The motor connected to Axis #1 should then begin to execute Motion Program #1.
6. Execute Program 2 by toggling %Q offset 3 bit. The motor connected to Axis #2 should begin to execute Motion Program #2.
7. The motors will execute the statements until they reach the first DWELL, where you can visually verify that it followed the CAM profile correctly. The display should be similar to Figure 269. Notice how the commanded position for Axis#2 equals 2500, while the commanded position for the slave corresponds to the CAM table and has the value 10,000.

Figure 269: RVExample Screen First Dwell



Once the dwell time is finished, the motors will continue executing the statements until they reach the second DWELL where you can visually verify that it followed correctly. The display should be similar to Figure 270. Notice how the commanded position for Axis#2 equals 47500, while the slave commanded position corresponds to the CAM table and has the value 11,000.

Figure 270: RVTEXample Screen Second Dwell



When the master axis reaches 50000 (47500 +2500), the CAM command will exit, the slave axis will decelerate at the programmed acceleration rate and come to a halt, and both axes will return to zero.

Details on the DSM314's %AI, %AQ, %I, and %Q memory are found in Chapter 5.

Technical support & Contact Information

Home link: <http://www.Emerson.com/Industrial-Automation-Controls>

Knowledge Base: <https://www.emerson.com/Industrial-Automation-Controls/support>

Note: If the product is purchased through an Authorized Channel Partner, please contact the seller directly for any support.

Emerson reserves the right to modify or improve the designs or specifications of the products mentioned in this manual at any time without notice. Emerson does not assume responsibility for the selection, use or maintenance of any product. Responsibility for proper selection, use and maintenance of any Emerson product remains solely with the purchaser.

© 2019 Emerson. All rights reserved.

Emerson Terms and Conditions of Sale are available upon request. The Emerson logo is a trademark and service mark of Emerson Electric Co. All other marks are the property of their respective owners.

