Magellan® Motion Control IC
User's Guide

P M D

Performance Motion Devices, Inc.
80 Central Street
Boxborough, MA 01719

Revision 2.8 April 2015

Warranty

PMD warrants that its products shall substantially comply with the specifications applicable at the time of sale, provided that this warranty does not extend to any use of any PMD product in anUnauthorized Application (as defined below). Except as specifically provided in this paragraph, each PMD product is provided "as is" and without warranty of any type, including without limitation implied warranties of merchantability and fitness for any particular purpose.

PMD reserves the right to modify its products, and to discontinue any product or service, without notice and advises customers to obtain the latest version of relevant information (including without limitation product specifications) before placing orders to verify the performance capabilities of the products being purchased. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement and limitation of liability.

Unauthorized Applications

PMD products are not designed, approved or warranted for use in any application where failure of the PMD product could result in death, personal injury or significant property or environmental damage (each, an "Unauthorized Application"). By way of example and not limitation, a life support system, an aircraft control system and a motor vehicle control system would all be considered "Unauthorized Applications" and use of a PMD product in such a system would not be warranted or approved by PMD.

By using any PMD product in connection with an Unauthorized Application, the customer agrees to defend, indemnify and hold harmless PMD, its officers, directors, employees and agents, from and against any and all claims, losses, liabilities, damages, costs and expenses, including without limitation reasonable attorneys' fees, (collectively, "Damages") arising out of or relating to such use, including without limitation any Damages arising out of the failure of the PMD product to conform to specifications.

Disclaimer

PMD assumes no liability for applications assistance or customer product design. PMD does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of PMD covering or relating to any combination, machine, or process in which such products or services might be or are used. PMD's publication of information regarding any third party's products or services does not constitute PMD's approval, warranty or endorsement thereof.

Related Documents

Magellan Motion Control IC Programmer's Command Reference

Descriptions of all Magellan Motion Control IC commands, with coding syntax and examples, listed alphabetically for quick reference.

Magellan Motion Control IC Electrical Specifications

Booklets containing physical and electrical characteristics, timing diagrams, pinouts, and pin descriptions of each series:

MC58000 Series, for DC brush, brushless DC, Microstepping, and Pulse & Direction motion processors

MC55000 Series, for Pulse & Direction motion processors

MC58113 single axis, single IC motion processor with integrated current control.

Magellan Motion Control IC Developer's Kit Manuals

How to install and configure the DK58000 series and DK55000 series developer's kit board.

How to install and configure the DK58113 developer's kit board.

Atlas Digital Amplifier User's Manual

Description of the Atlas Digital Amplifier electrical and mechanical specifications along with a summary of its operational features.

Atlas Digital Amplifier Complete Technical Reference

Complete electrical and mechanical description of the Atlas Digital Amplifier with detailed theory of operations.

Pro-Motion User's Guide

User's guide to Pro-Motion, the easy-to-use motion system development tool and performance optimizer. Pro-Motion is a sophisticated, easy-to-use program which allows all motion parameters to be set and/or viewed, and allows all features to be exercised.

Other Documents

ION Digital Drive User's Manual

How to install and configure ION Digital Drives.

Prodigy-PCI Motion Card User's Guide

How to install and configure the Prodigy-PCI motion board.

Prodigy-PC/104 Motion Card User's Guide

How to install and configure the Prodigy-PC/104 motion board.

Prodigy/CME Standalone User's Guide

How to install and configure the Prodigy/CME standalone motion board.

Prodigy/CME Machine-Controller User's Guide

How to install and configure the Prodigy/CME machine controller motion board.

Table of Contents

1. The Magellan Family 11
1.1 Family Summary 11
1.2 Magellan Motion Control IC Products 11

2. System Overview 13
2.1 MC58000 and MC55000-Series IC Interconnections 13
2.2 MC58113-Series IC Interconnections 14
2.3 Functional Overview 14
2.4 Documentation Guide 15
2.5 Product P/N Referencing Guide 16

3. Control Modules 19
3.1 Control Flow Overview 19
3.2 Enabling and Disabling Control Modules 21
3.3 Reset Command 22
3.4 Setting the Cycle Time 22
3.5 The Time Register 23
3.6 GetVersion Command 24

4. Trajectory Generation 25
4.1 Trajectories, Profiles, and Parameters 25
4.2 Trapezoidal Point-to-Point Profile 26
4.3 S-curve Point-to-Point Profile 28
4.4 Velocity-Contouring Profile 30
4.5 Electronic Gear Profile 31
4.6 The SetStopMode Command 33
4.7 Disabling and Enabling the Trajectory Generator Module 33

5. Position Loop 35
5.1 Overview 35
5.2 Dual Encoder Support 38
5.3 Biquad Output Filters 41
5.4 Output Limit 43
5.5 Motor Bias 43
5.6 Disabling and Enabling the Position Loop Module 44

6. Parameter Update and Breakpoints 47
6.1 Parameter Buffering 47
6.2 Breakpoints 49

7. Status Registers 55
7.1 Overview 55
7.2 Event Status Register 55
7.3 Activity Status Register 57
7.4 Drive Status Register (ION, MC58113-Series, and Atlas-Connected Axes Only) 58
7.5 Signal Status Register 59

8. Motion Monitoring and Related Processing 61
8.1 SetEventAction Processing 61
8.2 Motion Error 63

8.3 Travel-Limit Switches....63
8.4 Tracking Window....65
8.5 Motion Complete Indicator....66
8.6 In-Motion Indicator....66
8.7 Settle Window....67
8.8 Trace Capture....68
8.9 Trace Buffer Architecture....68
8.10 Host Interrupts....74

9. Hardware Signals....77
9.1 The AxisOut Pin....77
9.2 The AxisIn Pin....77
9.3 Analog Input....78
9.4 The Synch Pin—Multiple Chip Synchronization....78

10. Encoder Interfacing....81
10.1 Incremental Encoder Input....81
10.2 High-Speed Position Capture....82
10.3 Parallel-Word Position Input....83

11. Motor Output....85
11.1 Disabling the Motor Output Module....85
11.2 Enabling the Motor Output Module....86
11.3 Motor Type....86
11.4 Motor Command Output....87
11.5 Setting PWM Frequency....94
11.6 Setting PWM Signal Interpretation (MC58113-Series Only)....94
11.7 Multi-Phase Motor Interfacing....95
11.8 Microstepping Motor Output....98
11.9 DC Servo Motor Output....100
11.10 Pulse & Direction Signal Generation....102

12. Host Communication....105
12.1 Host Communication Packets....105
12.2 Parallel Communications....107
12.3 Serial Port....110
12.4 Controller Area Network (CAN) Communications....114
12.5 Storing Communication Values Using ION....116
12.6 SPI (Serial Peripheral Interface) Communications....117

13. Brushless DC Motor Control....121
13.1 Overview....121
13.2 Number of Phases....122
13.3 Phasing Control Modes....122
13.4 Phase Counts....123
13.5 Phase Initialization....123
13.6 Phase Initialization Programming....125
13.7 Automatic Phase Correction....126
13.8 Encoder Prescaler....128
13.9 Sinusoidal Commutation....128
13.10 Field Oriented Control....129

14. Step Motor Control....133

14.1 Overview 133
14.2 Encoder Feedback 134
14.3 Stall Detection 135
14.4 Pulse & Direction Motor Control 135
14.5 Microstepping Motor Control 136

15. Drive Control 141
15.1 Current Loop 142
15.2 Current Loop Parameters. 142
15.3 Enabling and Disabling Current Loop 143
15.4 Current Feedback 143
15.5 Reading Current Loop Values. 143
15.6 Drive Safety Feature Overview. 144
15.7 Overcurrent Sense 144
15.8 Overtemperature Sense 145
15.9 Overvoltage Sense 146
15.10 Undervoltage Sense 147
15.11 Current Foldback 147
15.12 Drive Enable Signal 148
15.13 FaultOut Signal 148
15.14 Drive Fault Status Register 149
15.15 Atlas-Specific Commands 150
15.16 Setting Atlas Initialization Configuration 151

16. Memory Buffers and I/O 153
16.1 Memory Configuration 153
16.2 User I/O. 154

Index 157

This page intentionally left blank.

List of Figures

2-1 MC58000 and MC55000- Series IC Control and Data Paths 13
2-2 MC58113-Series IC Control And Data Paths 14
3-1 Magellan Control Flow Overview 19
3-2 Magellan/Atlas Control Flow Overview 20
4-1 Simple Trapezoidal Point-To-Point Profiles 26
4-2 Trapezoidal Profile With Non-Zero Starting Velocity 27
4-3 Simple Trapezoidal Point-To-Point Profile 27
4-4 Complex Trapezoidal Point-To-Point Profile, Showing Parameter Changes 28
4-5 Typical S-curve Point-To-Point Profile 29
4-6 S-curve That Does Not Reach Maximum Acceleration 29
4-7 S-curve With No Maximum-Velocity Segment 30
4-8 Velocity-Contouring Profile 31
4-9 Electronic Gear Profile 32
5-1 PID Loop And Biquad Filters 35
5-2 Position Loop Flow 36
5-3 Magellan Dual-Loop Flow 39
5-4 Magellan Dual-Loop Digital Filter 40
5-5 Biquad Algorithm Flow 41
5-6 Motor Control Paths, Trajectory Enabled/Disabled 45
8-1 Directional Limit Switch Operation 63
8-2 Tracking Window 65
8-3 Settle Window 67
10-1 Quadrature Encoder Timing 81
11-1 Sign/Magnitude PWM Encoding 89
11-2 50/50 PWM Encoding 90
11-3 PWM High/Low Encoding 90
11-4 High Level Magellan To Atlas Connection Diagram 93
11-5 SPI Atlas Communications Protocol Overview 94
11-6 Motor Output Waveform (Vout) 95
11-7 Brushless DC Motor (PWM High/Low With Current Control) Connection Scheme
(MC58113-Series ICs Only) 96
11-8 Brushless DC Motor (50/50 PWM Mode) Connection Scheme 97
11-9 Brushless DC Motor (DAC Mode) Connection Scheme 97
11-10 Two-Phase Step Motor (PWM High/Low With Current Control) Connection Scheme
(MC58113-Series ICs Only) 98
11-11 Two-Phase Motor (PWM Sign/Magnitude or DAC) Connection Scheme 99
11-12 Two-Phase Step Motor (Sign/Magnitude PWM With Current Control) Connection Scheme 100
11-13 Typical Amplifier Configuration For 3-Phase Step Motor 100
11-14 DC Brush Motor (PWM High/Low With Current Control) Connection Scheme
(MC58113-Series ICs Only) 101
11-15 DC Brush Motor (DAC) Connection Scheme 101
11-16 DC Brush Motor (Sign/Magnitude PWM With Current Control) Connection Scheme 102
11-17 Pulse and Direction Motor Output Connection Scheme 102
12-1 Parallel Communication Interconnec-tions 107
13-1 Commutation Waveforms 122
13-2 Hall-Based Phase Initialization 124
13-3 Sinusoidal Commutation 129
13-4 Control Flow Of FOC Control 130
13-5 Algorithmic Flow Of FOC Controller 131
14-1 Microstepping Waveform Generation 137
14-2 Microstepping Waveforms 138
15-1 Current Loop Control Flow 142
15-2 Host To Magellan To Atlas Packet Processing 150

This page intentionally left blank.

1. The Magellan Family

In This Chapter

- Family Summary
- Magellan Motion Control IC Products

1.1 Family Summary

The *Magellan Motion Control IC User's Guide* supports the Magellan Family of ICs from PMD, including the MC58000 Series (DC brush, brushless DC, and step motor) and the MC55000 Series (step motor). In addition, Magellan processors are used in a number of card and module-level products including ION Digital Drives, Prodigy motion control cards, and Prodigy/CME Machine Controller cards.

Magellans are complete IC-based motion processors, providing trajectory generation and related motion control functions. Depending on the type of motor to be controlled, they provide servo loop closure, on-board commutation for brushless motors, and high-speed pulse and direction outputs. When the MC58113 is used, when used in ION Digital Drives, or when used with Atlas Digital amplifiers, they also provide current control, short circuit protection, and many other amplifier-related functions. Together, these products provide a software-compatible family of dedicated motion control ICs that can handle a large variety of motion control applications.

1.2 Magellan Motion Control IC Products

The following table presents a feature summary of the various members of the Magellan Motion Control IC family:

	MC58000 Series (Except MC58113)	MC55000 Series	MC58113 Series
# of axes	1, 2, 3, 4	1, 2, 3, 4	1+ (primary & aux channel encoder input)
Motor types supported	DC Brush, Brushless DC, step motor	Step motor	DC Brush, Brushless DC, step motor
Output format	SPI Atlas, PWM, DAC, pulse & direction	Pulse & direction	SPI Atlas, PWM, DAC, pulse & direction
Parallel host communication	✓	✓	
Serial host communication	✓	✓	✓
CAN 2.0B host communication	✓	✓	✓
SPI host communication			✓
Incremental encoder input	✓	✓	✓
Parallel word device input	✓	✓	
Index & Home signals	✓	✓	✓
Position capture	✓	✓	✓
Directional limit switches	✓	✓	✓
PWM output	✓		✓
Parallel DAC output	✓		
SPI Atlas interface	✓		✓
SPI DAC output	✓		✓
Pulse & direction output	✓	✓	✓

	MC58000 Series (Except MC58113)	MC55000 Series	MC58113 Series
Digital current control	✓ (with Atlas)		✓
Field oriented control	✓ (with Atlas)		✓
Under/overvoltage sense	✓ (with Atlas)		✓
I^2T Current foldback	✓ (with Atlas)		✓
DC Bus shunt resistor control			✓
Overtemperature sense	✓ (with Atlas)		✓
Short circuit sense	✓ (with Atlas)		✓
Trapezoidal profiling	✓	✓	✓
Velocity profiling	✓	✓	✓
S-curve profiling	✓	✓	✓
Electronic gearing	✓	✓	✓
On-the-fly changes	✓	✓	✓
PID position servo loop	✓		✓
Dual biquad filters	✓		✓
Dual encoder loop	✓ (multi-axis configurations only)		✓
Programmable derivative sampling time	✓		✓
Feedforward (accel & vel)	✓		✓
Data trace/diagnostics	✓	✓	✓
Motion error detection	✓	✓ (with encoder)	✓
Axis settled indicator	✓	✓ (with encoder)	✓
Analog input	✓	✓	✓
Programmable bit output	✓	✓	✓
Software-invertible signals	✓	✓	✓
User-defined I/O	✓	✓	
Internal Trace Buffer			✓
External RAM support	✓	✓	
Multi-chip synchronization	✓		✓
Chipset configurations	MC58420 (4 axes, 2 ICs) MC58320 (3 axes, 2 ICs) MC58220 (2 axes, 2 ICs) MC58120 (1 axis, 2 ICs) MC58110 (1 axis, 1 IC)	MC55420 (4 axes, 2 ICs) MC55320 (3 axes, 2 ICs) MC55220 (2 axes, 2 ICs) MC55120 (1 axis, 2 ICs) MC55110 (1 axis, 1 IC)	MC51113 (1+ axis, 1 IC) MC53113 (1+ axis, 1 IC) MC54113 (1+ axis, 1 IC) MC58113 (1+ axis, 1 IC)
IC Package: CP chip	MC58x20: 144 pin TQFP MC58110: 144 pin TQFP	MC55x20: 144 pin TQFP MC55110: 144 pin TQFP	100 pin TQFP
IC Package: IO chip	MC58x20: 100 pin TQFP MC58110: NA	MC55x20: 100 pin TQFP MC55110: NA	N/A
Motion control IC developer's kit p/n's	DK58420 DK58320 DK58220 DK58120 DK58110	DK55420 DK55320 DK55220 DK55120 DK55110	DK51113 DK53113 DK54113 DK58113

2. System Overview

In This Chapter

- MC58000 and MC55000-Series IC Interconnections
- MC58113-Series IC Interconnections
- Functional Overview
- Documentation Guide
- Product P/N Referencing Guide

2.1 MC58000 and MC55000-Series IC Interconnections

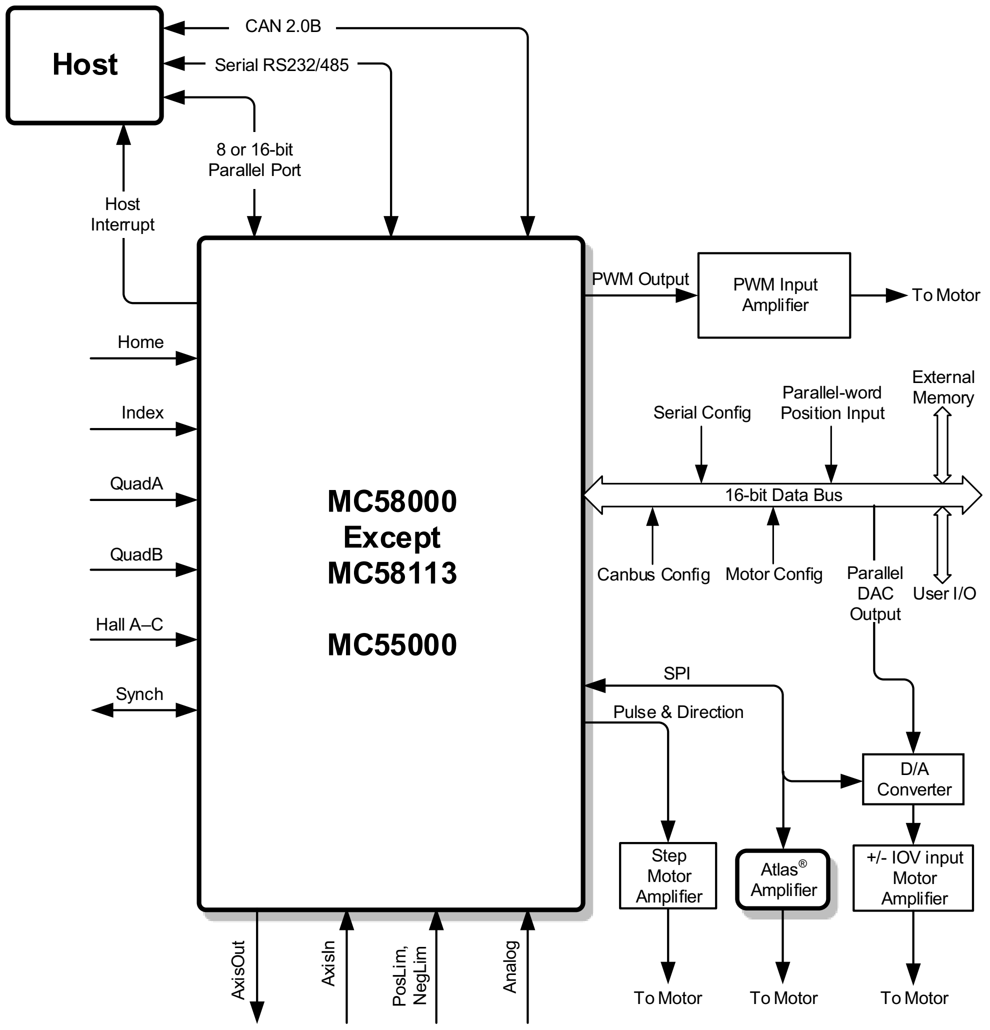
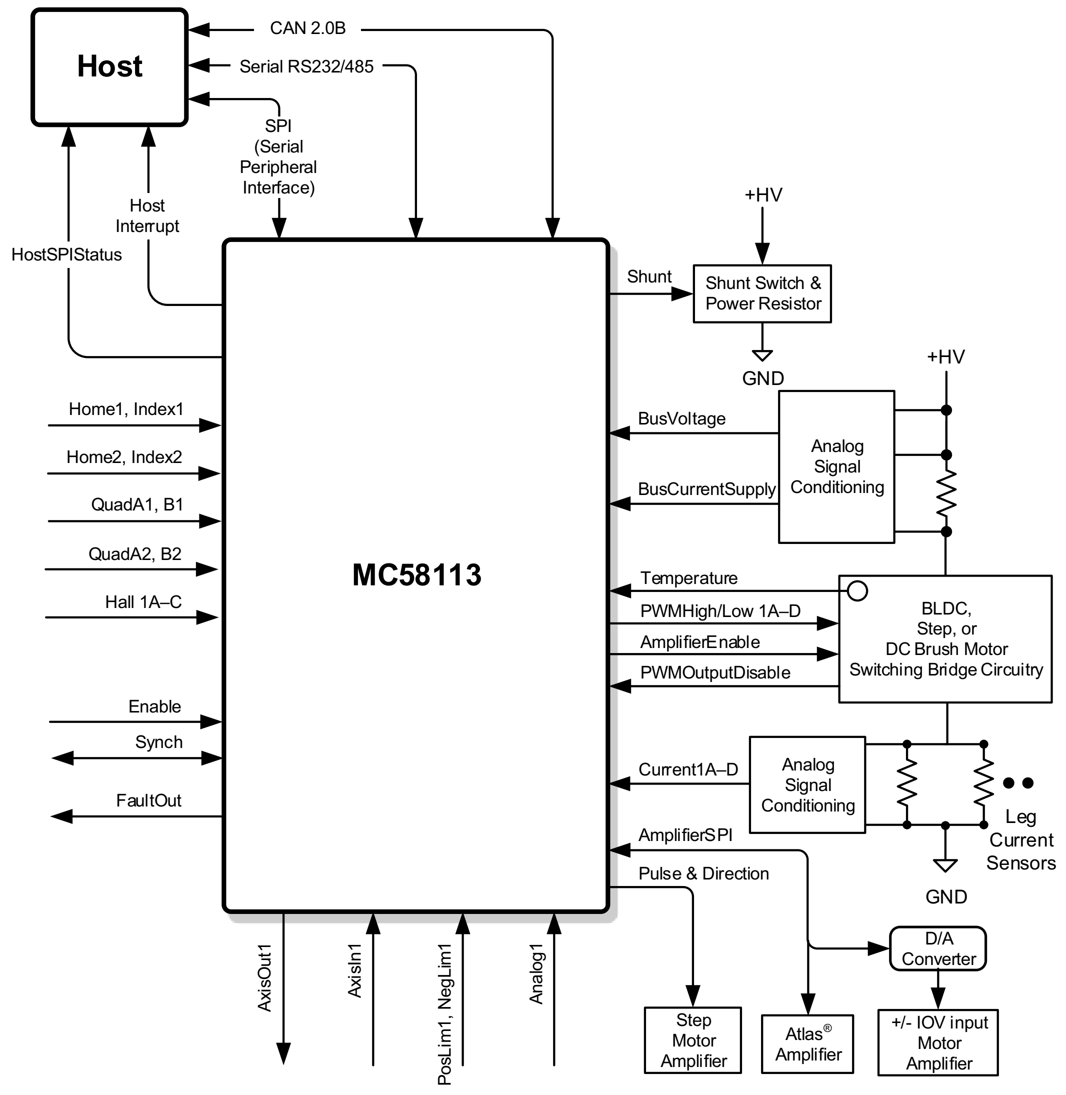


Figure 2-1: MC58000 and MC55000-Series IC Control and Data Paths

2.2 MC58113-Series IC Interconnections

Figure 2-2: MC58113-Series IC Control And Data Paths



2.3 Functional Overview

Figure 2-1 and Figure 2-2 show interconnection diagrams for the Magellan Motion Control ICs. Not all Magellan ICs support all of these interconnections. Refer to Section 1.2, "Magellan Motion Control IC Products," for a detailed summary of which ICs support which interfaces.

For chip-level designs, you will interface these interconnections with your own circuitry to create a complete motion card. For Magellan-based card and module-level products, some of these connections (such as encoder, limit switches, etc.) are available externally to the user, while some are connected to the internal card or module circuitry and do not require user interfacing. Refer to the card user's guide or module user's manual for more information.

Regardless of the hardware configuration, the overall control approach is similar. Each axis inputs the actual location of the axis using either incremental encoder signals or a parallel-word input device such as an absolute encoder, analog-to-digital converter, resolver, or laser interferometer. If incremental signals are used, the incoming A and B quadrature data stream is digitally filtered, and then passed on to a high-speed up/down counter. Using the parallel-word interface, a direct binary-encoded position word is read by the motion control IC. Regardless of the encoder input method, this position information is then used to maintain a 32-bit actual axis position counter.

Magellan contains a trajectory generator that calculates a new desired position at each cycle time interval, which is based on the profile modes and parameters programmed by the host, as well as on the current state of the system. The cycle time is the rate at which major system parameters are updated.

For motion control ICs with servo motor support (MC58000), the output of the trajectory generator is combined with the actual encoder position to calculate a 32-bit position error, which is passed through a PID position loop.

The resultant value is then output by the motion control IC to an external amplifier using either PWM, DAC, or SPI Atlas-compatible signals. If the axis is configured for a brushless DC motor, then the output signals are commutated, meaning they are combined with information about the motor phase angle to distribute the desired motor torque to two- or three-phased output commands. If the MC58113-series is used or if an ION Digital Drive is used, rather than being output to the amplifier the resultant value may be input to a current control loop, the output of which is then used to generate PWM output signals.

With an MC58000 axis configured for DC brush servo motors, the single-phase motor command is output directly. For axes configured for step motors, the output of the trajectory generator is converted to either microstepping signals (MC58000), or pulse and direction signals, and is then output accordingly. If not using pulse & direction signals step motor output is in PWM, DAC-compatible, or SPI Atlas-compatible format.

If Atlas Digital Amplifiers are used then the hardware connection format is always a four-signal SPI bus. Magellan automatically provides the protocol message formats needed by Atlas to support the motor type being used, either DC Brush, brushless DC, or step motor.

Host communication to and from Magellan Motion Control ICs is accomplished using a parallel-bus interface (all Magellan ICs except MC58113-series), an asynchronous serial port, a CAN 2.0B interface, or SPI (Serial Peripheral Interface, MC58113-series only). If parallel-bus communication is used, there is a further choice of 8-bit wide transfers or 16-bit wide transfers, allowing a range of microprocessors and data buses to be interfaced. If serial communications are used, then the user selects parameters such as baud rate, number of stop/start bits, and the transfer protocol. The transfer protocol may be either point-to-point (appropriate for single-motion control IC systems), or multi-drop (appropriate for serial communications to multiple motion control ICs). For CAN communication, the user selects the desired CAN data bus rate and the CAN node address.

For card-product communications through the bus, the parallel-bus interface is fixed to a 16-bit format. For example, the Prodigy-PCI card provides a complete PCI bus interface that connects to the motion control IC in its 16-bit parallel interface mode. PMD's motion cards and modules also provide serial, CANbus, and Ethernet communication options.

Regardless of the hardware interface method, communication to and from Magellan Motion Control ICs occurs using short commands sent or received as a sequence of bytes and words. These packets contain an instruction code word that tells the motion control IC which operation is being requested. It may also contain data sent to, or received from, the motion control IC.

These commands are sent by a host microprocessor or host computer executing a supervisor program that provides overall system control. The Magellan Motion Control IC is designed to function as the motion engine, managing high-speed dedicated motion functions such as trajectory generation, safety monitoring, etc., while the host software program provides the overall motion sequences.

2.4 Documentation Guide

Many functions are common across all Magellan Motion Control ICs. However, some sections of this user's guide describe features that apply to specific motor types, or to certain Magellan products. For example, position loop servo filtering applies only to axes configured for DC brush or brushless DC motors. The following table cross-references the applicable chapters for each motor type and Magellan-based product:

MC50000 Series Motion Control ICs

The following table describes the MC50000 Series Motion Control ICs.

Motor Type	Description	Applicable Chapters
DC brush	Servo motors with internal mechanical commutation, or connected to a commutating amplifier.	1–12, 15, 16
Brushless DC (2 phase, 3 phase)	Servo motors requiring external electrical commutation.	1–13, 15, 16
Microstepping motor	Step motor with microstepping drive.	1–4, 6–12, 14-16
Pulse & direction step motor	Step motor with pulse & direction drive.	1–4, 6–12, 14, 16

ION, MC58113, and Atlas-Connected Magellan Motion Control ICs

The following table describes the chapters that apply for IONs, the MC58113, and Atlas-connected axes.

Motor Type	Description	Applicable Chapters
DC brush	Servo motors with internal mechanical commutation.	1–12, 15
Brushless DC	3-phase brushless DC servo motors.	1–13, 15
Step motor	2-phase step motor.	1–12, 14, 15

2.5 Product P/N Referencing Guide

In this manual, various chapters, sections, or paragraphs give descriptions such as “MC55000 only.” The following table indicates the specific products that are referred to by each such reference:

Reference	Description	Parts Included
MC58000	Indicates all MC58000 series Magellan Motion Control ICs.	MC58420 MC58320 MC58220 MC58120 MC58110 MC58113 MC51113 MC53113 MC54113
MC55000	Indicates all MC55000 series Magellan Motion Control ICs.	MC55420 MC55320 MC55220 MC55120 MC55110

Reference	Description	Parts Included
MC50000	Indicates all Magellan Motion Control ICs.	MC58420 MC58320 MC58220 MC58120 MC58110 MC55420 MC55320 MC55220 MC55120 MC55110 MC58113
MC58113	Indicates all MC58113-series ICs	MC51113 MC53113 MC54113 MC58113

This page intentionally left blank.

3. Control Modules

In This Chapter

- Control Flow Overview
- Enabling and Disabling Control Modules
- Reset Command
- Setting the Cycle Time
- The Time Register
- GetVersion Command

3.1 Control Flow Overview

Figure 3-1 provides a control flow overview for the Magellan Motion Control IC. It shows how a final motor command is generated, starting with the profile generator and ending with the motor output module that generates amplifier-compatible output signals. Only IONs, MC58113-series ICs, and Atlas-connected axes support the current loop/FOC module.

Depending on the type of product and motor, some modules may not be used. For example, step motors do not use a position PID loop. In addition, depending on the nature of the control problem, some modules may be disabled by the user to tailor the control for their specific application.

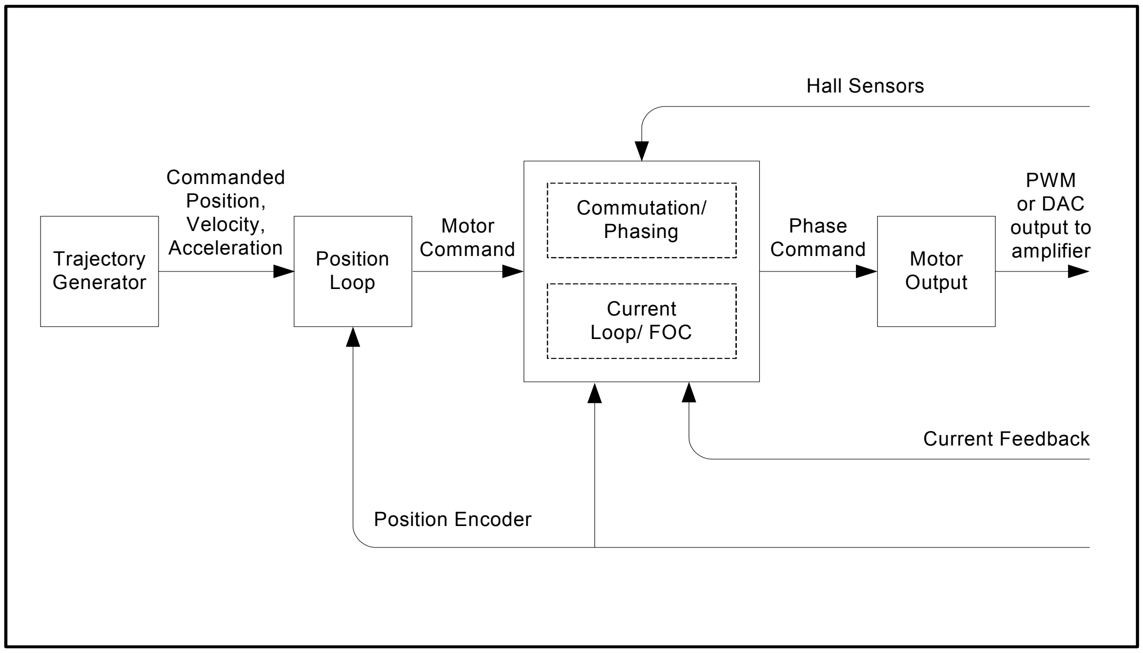
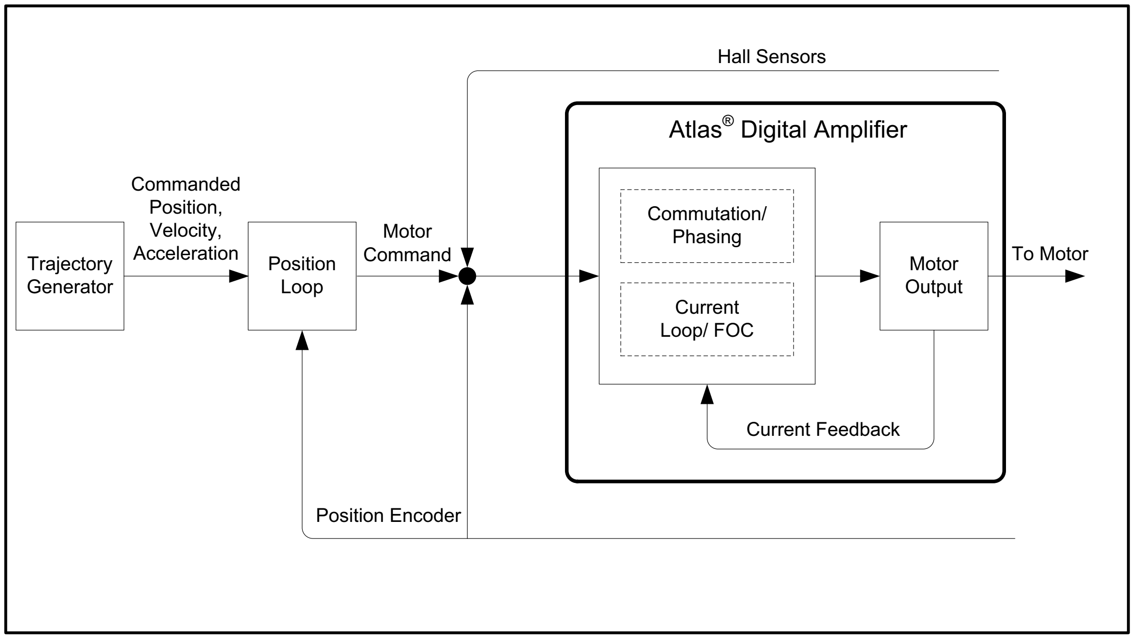


Figure 3-1: Magellan Control Flow Overview

If Atlas Digital Amplifiers are utilized, then the overall control flow is modified in that the actual current loop and related drive-specific functions are located in the Atlas unit, however the Magellan IC seamlessly integrates the combined

Magellan/Atlas system into one control structure which is accessed via the Magellan IC. Figure 3-2 shows the overall control flow when an Atlas amplifier is connected.

Figure 3-2: Magellan/ Atlas Control Flow Overview



In the ION module, a single, special ION-specific Magellan IC similar to the MC58113 provides all Magellan functions including current control and drive related functions.

Current Loop, FOC, and Current Feedback are available only when using an ION Drive, with the MC58113-series ICs, and with Atlas-connected axes.

Each of the major blocks within the control flow diagram is referred to as a module. The following table provides a brief description of each module.

Module Name	Function
Trajectory Generator	This module accepts user-specified parameters and generates a trajectory.
Position Loop	This module is used with servo motors only. It inputs the *commanded position* (the instantaneous desired axis position) and the *actual position* (the motor position measured by an encoder), and passes the resultant *position error* (the difference between the commanded and the actual position) through a PID filter along with dual biquad filters to generate a motor command.
Commutation/Phasing	This module is used with multi-phase motors such as brushless DC motors or microstepping step motors, and generates desired torque signals for each phase of the motor.
Current Loop/FOC	This module is used with ION drives, MC58113-series ICs, and Atlas-connected axes. It inputs the desired torque for each motor phase along with the actual measured current through each phase, and passes the resultant difference through a PI filter to generate a motor command.
Motor Output	This module inputs the desired motor phase command and generates the appropriate signals for the selected output format.

Each of these modules are described in detail in subsequent chapters. Beyond the functions provided by these major control modules, Magellan also provides numerous additional capabilities such as breakpoints, trace, and PLC-style signal control. These features are common to all motor types and motion control ICs, and are also described in detail in subsequent chapters.

3.2 Enabling and Disabling Control Modules

At various times during setup or operation of an axis, it may be desirable to selectively enable or disable specific control modules. This is accomplished using the command SetOperatingMode. To read back the status set using this command, the command GetOperatingMode is used. Generally speaking, if a module is disabled, Magellan skips whatever features and calculations are associated with that module, and the input from the previous module is passed directly to the subsequent module without modification.

The following table summarizes which modules may be disabled or enabled, and describes typical circumstances under which this might be useful. In addition to these specific modules it is possible to enable or disable an entire axis using the SetOperatingMode command. Note that the commutation/phasing module may not be disabled or enabled by the user. If a multi-phase motor type such as brushless DC or microstepping is selected, this module is always enabled.

Module	Description	Typical Uses
Trajectory Generator	If disabled, the commanded position will stay at its present value.	The trajectory generator is not usually disabled manually. For trajectory control, which requires an immediate stop, the SetStopMode command with an argument of *Abrupt Stop* is used instead.
Position Loop	Used with servo motors only. If disabled, this module's output is replaced by one of two sources, depending on whether the trajectory generator module is enabled or disabled. *Trajectory generator enabled.* If the trajectory generator is enabled, then the position loop is skipped, and the output of the profile generator is input directly to the subsequent module. *Trajectory generator disabled.* If the trajectory generator is also disabled, the output comes from the Motor Command register, which can be manually set using the command SetMotorCommand. See Section 5.6, "Disabling and Enabling the Position Loop Module," for details.	Disabling this module with trajectory generator enabled is useful if voltage or current-proportional positioning devices are used, such as certain kinds of galvanometers. It may also be useful if amplifier calibration with automated ramps is required. With trajectory generator disabled, disabling this module may be useful for amplifier or motor calibration.
Current Loop (ION modules, MC58113-series ICs, or Atlas-connected axes only)	If this module is disabled, the input motor command will be passed unmodified to the motor output module.	Disabling the current loop may be useful for calibration or when you are using a non-Atlas external amplifier that separately provides torque or velocity-based control.
Motor Output	Disabling this module sets all motor generation to a value of zero (0). The actual states of the associated motor output signals will depend on the selected signalling method (PWM sign/mag, PWM 50/50, PWM High/low, parallel DAC, serial DAC, or SPI Atlas) See Section 11.4, "Motor Command Output," for more information.	Disabling motor output is useful in connection with various safety-related conditions, or for amplifier calibration.

In addition to manually disabling modules, there are a number of circumstances where modules may be automatically disabled due to event-related issues, or breakpoints. See Section 8.1, “SetEventAction Processing,” and Section 6.2, “Breakpoints,” for more details.

GetOperatingMode returns the value set using the command **SetOperatingMode**, which sets the desired operating mode under normal operational circumstances. However this may differ from the actual operating mode for the reasons mentioned above. To determine the actual current status of the operating mode word use the command **GetActiveOperatingMode**.

Throughout this user’s guide various command mnemonics will be shown to clarify motion process or command usage or provide specific examples. See the *Magellan Motion Control IC Programmer’s Command Reference* for more information on host commands, nomenclature, and syntax.

3.3 Reset Command

In addition to enabling and disabling control modules, it is possible to entirely reset the motion control IC using the **Reset** command. This command will bring all registers to their default values and reinitialize all motion control functions. See the *Magellan Motion Control IC Programmer’s Command Reference* for details on the default values of various Magellan registers.

For MC50000 products a **Reset** command will have an equivalent effect as toggling the motion control IC’s ***Reset*** hardware signal. Also, in addition to manual resets or signal-based resets, a reset operation automatically occurs during motion control ICs powerup. See the *Magellan Motion Control IC Electrical Specifications* for details.

Due to the large number of operations required to complete a reset operation, **Reset** commands generally take substantially longer to process than standard Magellan commands. See the *Magellan Motion Control IC Electrical Specifications* or *ION Digital Drive User’s Manual* for details.

Note that in normal operation resets are not required. They are generally used during development or debugging to bring the system to a known initial state.

Executing a **Reset** command will result in the motor command for all axes being set to zero (0), and all motion control IC activity restarting from a default condition. It is the responsibility of the user to determine whether sending a **Reset** command is safe for a given operational condition.

3.4 Setting the Cycle Time

The motion control IC calculates all trajectory and servo information on a fixed, regular interval. This interval is known as the cycle time. For each enabled axis of the motion control IC, there is a required “time slice” of either 51.2 (MC50000) or 102.4 (ION) microseconds. In addition, for some motion control ICs there may be added overhead associated with the trace capture facility, and some internal overhead for multi-axis configurations. The minimum cycle times for various configurations of the Magellan Motion Control IC are provided in the following tables.

MC50000 Except MC58113-Series

# Enabled Axes	Minimum Cycle Time	Cycle Time w/ Trace Capture	Time per Axis	Maximum Cycle Frequency
1 (ION)	102.4 us	102.4 us	102.4 us	9.76 kHz
1 (Magellan Single-axis)	51.2 **µs**	102.4 µs	51.2 µs	19.53 KHz (9.76 w/ trace capture)
1 (Magellan Multi-axis)	102.4µs	102.4 µs	102.4 µs	9.76 kHz
2 (Magellan)	153.6 µs	153.6 µs	76.8 µs	6.51 KHz
3 (Magellan)	204.8 µs	204.8 µs	68.3 µs	4.88 KHz
4 (Magellan)	256 µs	256 µs	64 µs	3.91 KHz

ION & MC58113-Series

The minimum cycle time for ION and the MC58113 does not depend on whether trace has been selected, or any other factors. The following table summarizes this:

ION	Minimum Cycle Time	Maximum Cycle Frequency
ION	102.4 µs	9.76 KHz
MC58113	51.2 µs	19.53 KHz

The cycle rate determines the trajectory update rate for all motor types, as well as the servo loop calculation rate. It does not necessarily determine the commutation rate, the PWM rate, or the current loop rate.

An enabled axis receives its cycle time slice whether or not it is in motion, and whether or not all the modules are enabled. For multi-axis motion control ICs, if cycle time is critical, it is possible to reclaim that time slice by disabling an unused axis, and then resetting the loop rate with the instructions SetOperatingMode and SetSampleTime.

For example, using an MC55240, four axes are available. If only three of the axes will be used in a specific application, then the unused axis may be disabled using the command SetOperatingMode and a new, lower sample time may be set using the SetSampleTime command. This would improve the cycle frequency from 3.90 kHz to 4.88 kHz.

SetSampleTime may also be used to increase the cycle time to a value greater than the allowed minimum when required.

SetSampleTime cannot be used to set a sample time lower than the required minimum cycle time for the current configuration. Attempting to do so will set the required minimum as the sample time.

3.5 The Time Register

Magellan processors keep a 32-bit register that holds the current motion control IC time, measured as the number of cycles executed since powerup or reset. This continuously changing value can be read using the command GetTime.

The register has two primary purposes. It can be used as a comparison value for time-based breakpoints (See Section 6.2, “Breakpoints,” for details). In addition, it can be a useful way of keeping track of actual time elapsed by manually querying the time.

The Time register increases by a value of 1 for each cycle that the motion control IC executes until it reaches its largest possible value of FFFF FFFFh or 4,294,967,295 dec, at which point it wraps back to zero (0). The point at which the time wrap will occur depends on the cycle time set for the motion control IC. For example, for a 4-axis MC58000 with the default cycle time of 256 uSec, wrap will occur at 256 uSec * 4,294,967,295 = ~12.7 days. All motion control IC operations will continue normally, although if a time breakpoint has been set, care should be taken to correctly calculate the comparison time including any potential wrap.

3.6 GetVersion Command

All Magellan Motion Control ICs can be queried to provide a unique code that indicates the product type and (if applicable) version code. To retrieve this information use the command GetVersion. For a detailed description of the information provided, see the *Magellan Motion Control IC Programmer's Command Reference.*

4. Trajectory Generation

In This Chapter

- Trajectories, Profiles, and Parameters
- Trapezoidal Point-to-Point Profile
- S-curve Point-to-Point Profile
- Velocity-Contouring Profile
- Electronic Gear Profile
- The SetStopMode Command
- Disabling and Enabling the Trajectory Generator Module

4.1 Trajectories, Profiles, and Parameters

The trajectory generator performs calculations to determine the instantaneous position, velocity, and acceleration of each axis at any given moment. These values are called the commanded values. During a motion profile, some or all of these parameters will continuously change. Once the move is complete, these parameters will remain at the same value until a new move begins.

To query the instantaneous commanded profile values, use the commands **GetCommandedPosition**, **GetCommandedVelocity**, and **GetCommandedAcceleration**.

The specific profile created by the Magellan Motion Control IC depends on several factors, including the presently selected profile mode, the presently selected profile parameters, and other system conditions such as whether a motion stop has been requested. Four trajectory profile modes are supported: S-curve point-to-point, trapezoidal point-to-point, velocity contouring, and electronic gearing. The operation of these profile modes will be explained in detail in subsequent sections. The command used to select the profile mode is **SetProfileMode**. The command **GetProfileMode** retrieves the programmed profile mode.

The profile mode may be programmed independently for each axis. For example, axis #1 may be in trapezoidal mode, while axis #2 is in S-curve point-to-point mode.

Magellan Motion Control ICs can switch from one profile to another while an axis is in motion, with only one exception: when switching to the S-curve point-to-point profile from any other profile, the axis must be at rest.

4.1.1 Trajectory Parameter Representation

The Magellan Motion Control IC sends and receives trajectory parameters using a fixed-point representation. In other words, a fixed number of bits is used to represent the integer portion of a real number, and a fixed number of bits is used to represent the fractional component of a real number. The motion control IC uses the following three formats.

Format	Word Size	Range	Description
32.0	32 bits	−2,147,483,648 to +2,147,483,647	Unity scaling. This format uses an integer-only representation of the number.
16.16	32 bits	−32,768 to 32,767 + 65,535/65,536	Uses $1/2^{16}$ scaling. The motion control IC expects a 32-bit number scaled by a factor of 65,536. For example, to specify a velocity of 2.75, 2.75 is multiplied by 65,536, and the result is sent to the motion control IC as a 32-bit integer (180,224 dec. or 0002C000h).
0.32	32 bits	0 to +2,147,483,647/ 4,294,967,296	Uses $1/2^{32}$ scaling. The motion control IC expects a 32-bit number scaled by a factor of 4,294,967,296 (2^{32}). For example, to specify a value of .0075, .0075 is multiplied by 4,294,967,296 and the result is sent to the motion control IC as a 32-bit integer (32,212,256 decimal or 00EB8520h.).

4.2 Trapezoidal Point-to-Point Profile

The following table summarizes the host-specified profile parameters for the trapezoidal point-to-point profile mode.

Profile Parameter	Format	Word Size	Range
Position	32.0	32 bits	−2,147,483,648 to 2,147,483,647 counts.
Starting Velocity	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle.
Velocity	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle.
Acceleration	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle^2.
Deceleration	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle^2.

The host instructions SetPosition, SetStartVelocity, SetVelocity, SetAcceleration, and SetDeceleration load these values. The commands GetPosition, GetStartVelocity, GetVelocity, GetAcceleration, and GetDeceleration retrieve the programmed values.

For this profile, the host specifies an initial acceleration and deceleration, a velocity, and a destination position. The profile gets its name from the resulting curve (see Figure 4-1). The axis accelerates linearly (at the programmed acceleration value), until it reaches the programmed velocity. It continues in motion at that velocity, then decelerates linearly (using the deceleration value) until it stops at the specified position.

Figure 4-1: Simple Trapezoidal Point-To-Point Profiles

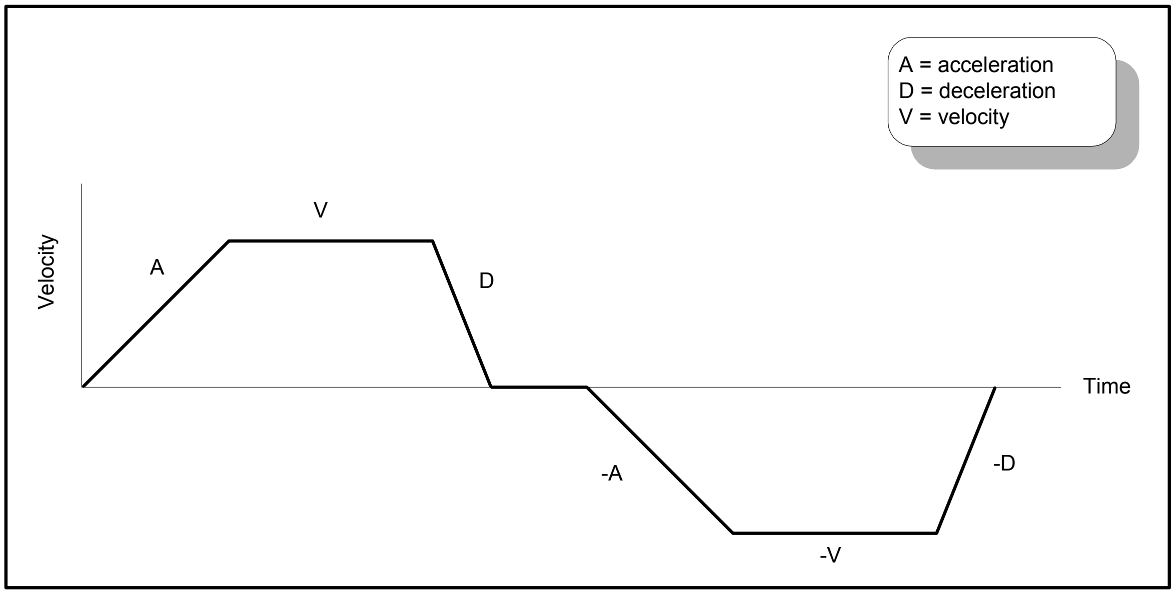


Figure 4-1 illustrates a trapezoidal profile with the starting velocity set at the default value of zero (0). When whole-stepping a step motor, it is sometimes desirable to define a non-zero starting velocity from which the motor will instantaneously begin motion. This is to avoid passing through the resonant frequency of a step motor. In the deceleration phase of the profile, rather than continuously decelerate to a velocity of zero (0), the velocity will transition from the start velocity to zero (0) velocity with no deceleration phase in between. Figure 4-2 shows a typical trapezoidal profile with non-zero starting velocity.

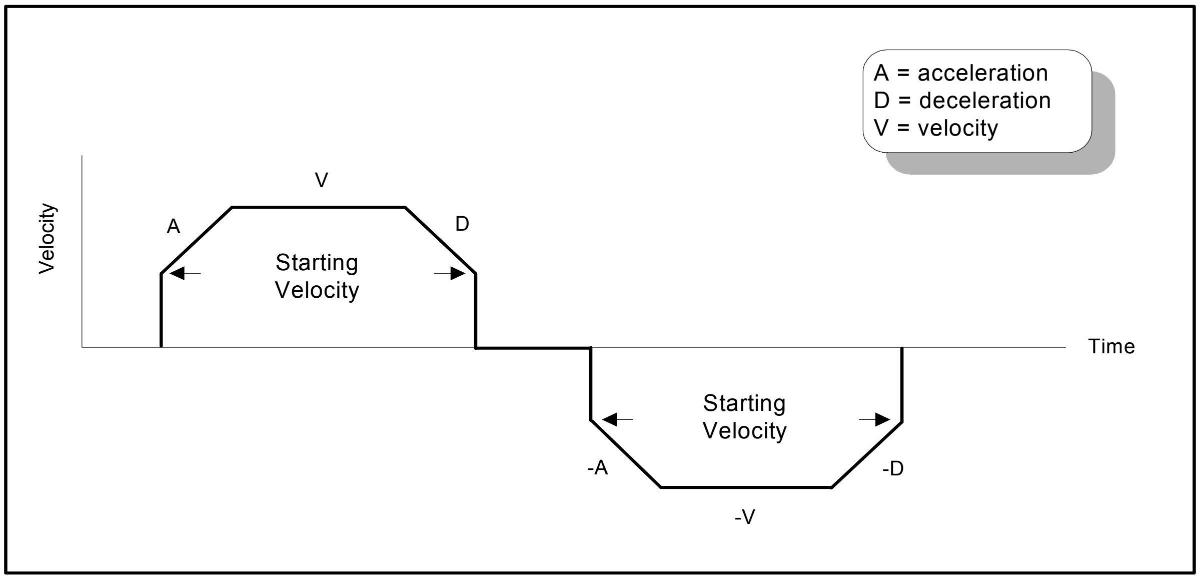


Figure 4-2: Trapezoidal Profile With Non-Zero Starting Velocity

Note that a programmable starting velocity is supported in Trapezoidal and Velocity Contouring profile modes only. It is not supported in Electronic Gear or S-curve profile modes.

If deceleration must begin before the axis reaches the programmed velocity, the profile will have no constant velocity portion, and the trapezoid becomes a triangle, as shown in Figure 4-3.

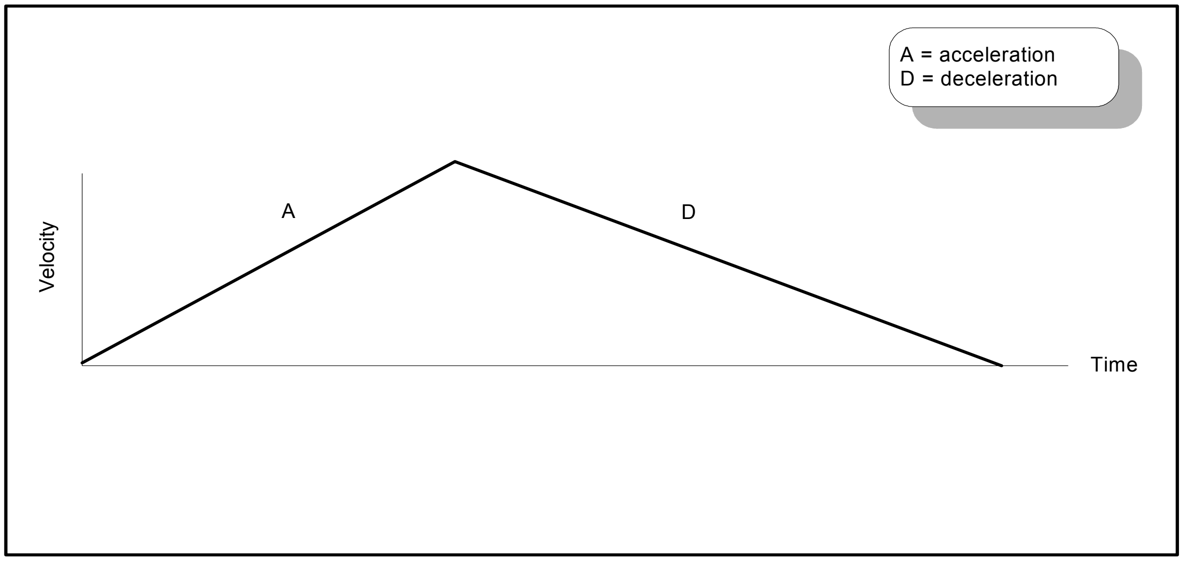
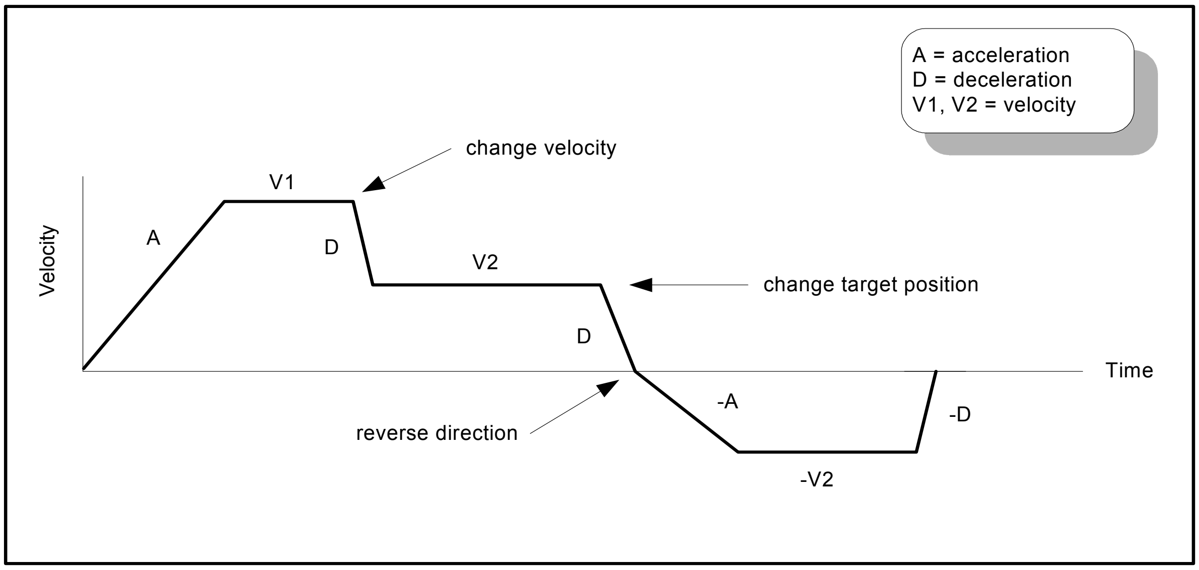


Figure 4-3: Simple Trapezoidal Point-To-Point Profile

The slopes of the acceleration and deceleration segments may be symmetric (if acceleration is equal to deceleration), or asymmetric (if acceleration is not equal to deceleration).

The acceleration parameter is always used at the start of the move. Thereafter, the acceleration value will be used when the absolute value of velocity is increasing, and deceleration will be used when the absolute value of velocity is decreasing. If no motion parameters are changed during the motion, then the acceleration value will be used until the maximum velocity is reached. The deceleration value will be used when ramping down to zero (0).

Figure 4-4: Complex Trapezoidal Point-To-Point Profile, Showing Parameter Changes



It is acceptable to change any of the profile parameters while the axis is moving in this profile mode. The profile generator will always attempt to remain within the legal bounds of motion specified by the parameters. If, during the motion, the destination position is changed in such a way that an overshoot is unavoidable, the profile generator will decelerate until stopped, then reverse direction to move to the specified position. This is illustrated in Figure 4-4.

If a deceleration value of zero (0) is programmed (or no value is programmed, leaving the motion control IC's default value of zero[0]), then the value specified for acceleration (**SetAcceleration**) will automatically be used to set the magnitude of deceleration.

4.3 S-curve Point-to-Point Profile

The following table summarizes the host-specified profile parameters for the S-curve point-to-point profile mode.

Profile Parameter	Format	Word Size	Range
Position	32.0	32 bits	–2,147,483,648 to 2,147,483,647 counts.
Velocity	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle.
Acceleration	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle^2.
Deceleration	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle^2.
Jerk	0.32	32 bits	0 to 2,147,483,647/4,294,967,296 counts/cycle^3.

The host instructions **SetPosition**, **SetVelocity**, **SetAcceleration**, **SetDeceleration**, and **SetJerk** load these respective values. The commands **GetPosition**, **GetVelocity**, **GetAcceleration**, **GetDeceleration**, and **GetJerk** retrieve the programmed values.

In S-curve profile mode, the same value must be used for both acceleration and deceleration. Asymmetric profiles are not allowed.

The S-curve point-to-point profile adds a limit to the rate of change of acceleration to the basic trapezoidal curve. A new parameter (jerk) is added which specifies the maximum change in acceleration in a single cycle.

In this profile mode, the acceleration gradually increases from 0 to the programmed acceleration value, then the acceleration decreases at the same rate until it reaches 0 again at the programmed velocity. The same sequence in reverse brings the axis to a stop at the programmed destination position.

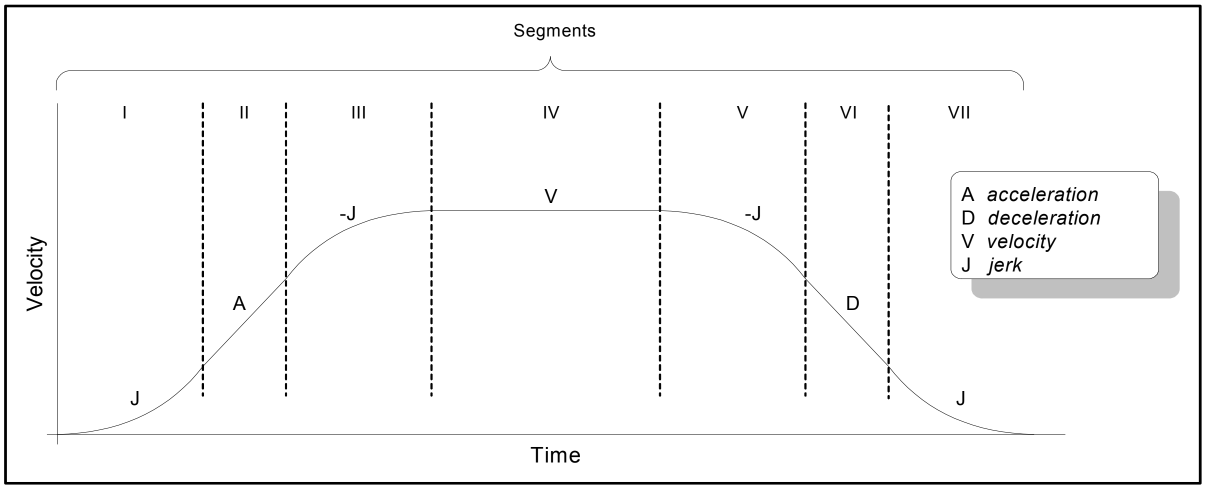


Figure 4-5: Typical S-curve Point-To-Point Profile

Figure 4-5 shows a typical S-curve profile. In Segment I, the S-curve profile drives the axis at the specified jerk (J) until the maximum acceleration (A) is reached. The axis continues to accelerate linearly (jerk = 0) through Segment II. The profile then applies the negative value of the jerk to reduce acceleration to 0 during Segment III. The axis is now at maximum velocity (V), at which it continues through Segment IV. The profile will then decelerate in a manner similar to the acceleration stage, using the jerk value first to reach the maximum deceleration (D) and then to bring the axis to a halt at the destination.

An S-curve profile might not contain all of the segments shown in Figure 4-5. For example, if the maximum acceleration cannot be reached before the “halfway” point to or from the velocity, the profile would not contain a Segment II or a Segment VI. Such a profile is shown in Figure 4-6.

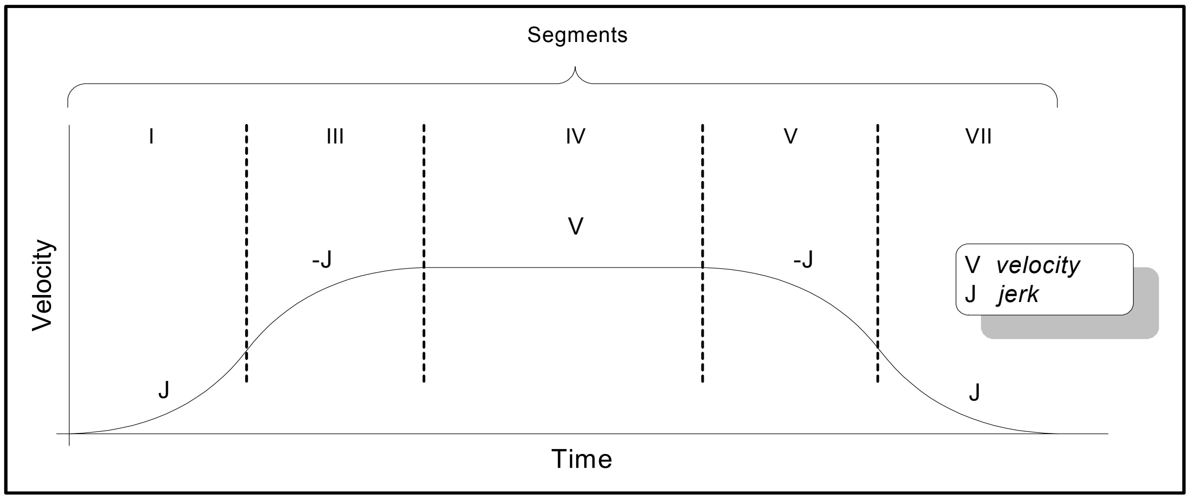
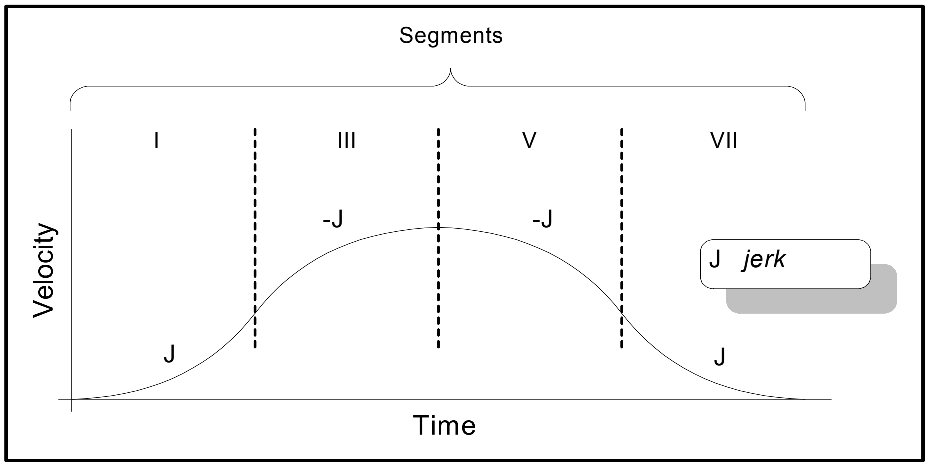


Figure 4-6: S-curve That Does Not Reach Maximum Acceleration

Similarly, if the position is specified such that velocity is not reached, there will be no Segment IV, as shown in Figure 4-7. There may also be no Segment II or Segment VI depending on where the profile is truncated.

Figure 4-7: S-curve With No Maximum-Velocity Segment



Unlike the trapezoidal profile mode, the S-curve profile mode does not support changes to any of the profile parameters while the axis is in motion.

An axis may not be switched into S-curve profile mode while the axis is in motion. It is legal to switch from S-curve mode to any other profile mode while in motion.

4.4 Velocity-Contouring Profile

The following table summarizes the host-specified profile parameters for the velocity-contouring profile mode.

Profile Parameter	Format	Word Size	Range
Start Velocity	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle.
Velocity	16.16	32 bits	–32,768 to 32,767 + 65,535/65,536 counts/cycle.
Acceleration	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle^2.
Deceleration	16.16	32 bits	0 to 32,767 + 65,535/65,536 counts/cycle^2.

The host instructions SetStartVelocity, SetVelocity, SetAcceleration, and SetDeceleration load these respective values. The commands GetStartVelocity, GetVelocity, GetAcceleration, and GetDeceleration retrieve the programmed values.

Unlike the trapezoidal and S-curve profile modes where the destination position determines the direction of initial travel, in the velocity-contouring profile mode, the sign of the velocity parameter determines the initial direction of motion. Therefore, the velocity value sent to the motion control IC can have positive values (for positive direction motion), or negative values (for negative direction motion).

In this profile, no destination position is specified. The motion is controlled entirely by changing the acceleration, velocity, and deceleration parameters while the profile is being executed.

In velocity-contouring profile mode, axis motion is not bounded by a destination. It is the host's responsibility to provide acceleration, deceleration, and velocity values that result in safe motion within acceptable position limits.

The trajectory is executed by continuously accelerating the axis at the specified rate until the velocity is reached. The axis starts decelerating when a new velocity is specified with a smaller value (in magnitude) than the present velocity, or a sign that is opposite to the present direction of travel. Figure 4-8 illustrates a more complicated profile, in which both the velocity and the direction of motion change twice.

As was the case for the Trapezoidal Profile mode, in addition to a maximum velocity, a starting velocity value can be specified which will cause the profile to instantly begin motion at that velocity, and instantly decelerate to zero (0) from that starting velocity.

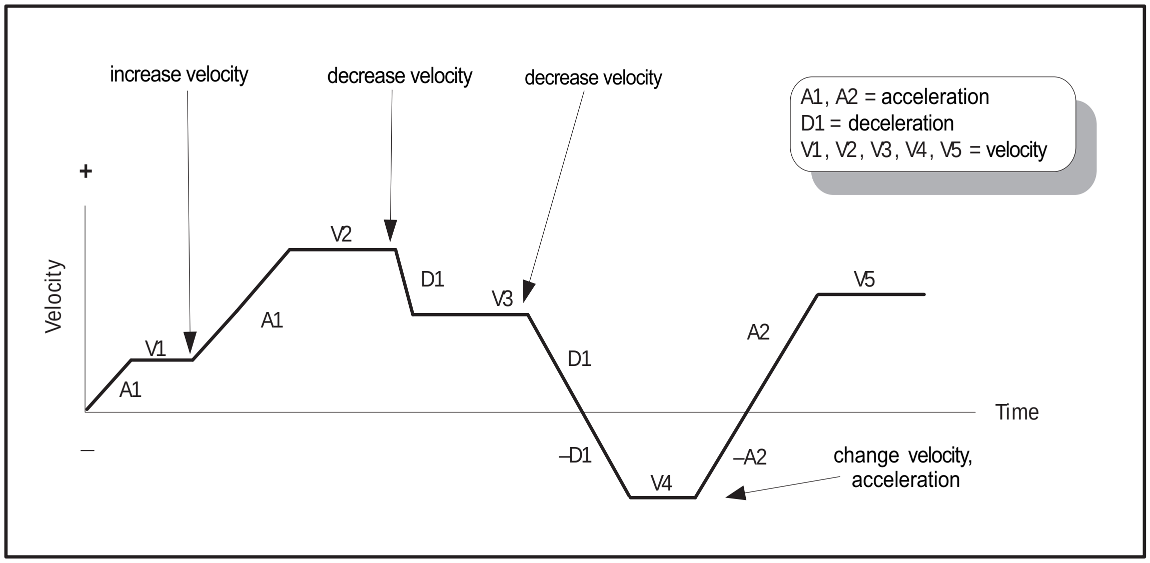


Figure 4-8: Velocity-Contouring Profile

4.5 Electronic Gear Profile

The following table summarizes the host-specified profile parameters for the electronic gear profile mode.

Profile Parameter	Format	Word Size	Range
Gear Ratio	16.16	32 bits	–32,768 to 32,767 + 65,535/65,536 counts/cycle.
Master Axis #	-	2 bits	0–3*
Master Source	-	1 bit	2 values; encoder or commanded (see below for details.)

The host instructions SetGearRatio and SetGearMaster load these respective values. The commands GetGearRatio and GetGearMaster retrieve the programmed values.

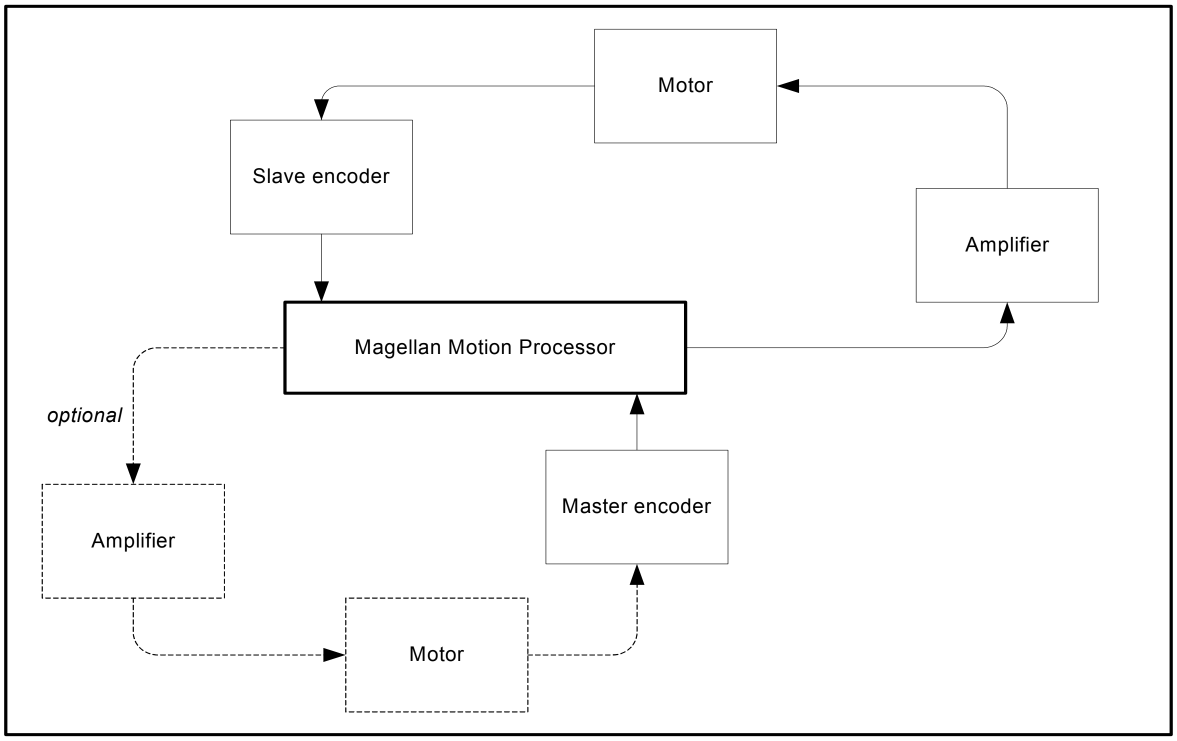
In this profile, the host specifies three parameters. The first is the master axis number. This is defined as the axis that will be the source of position information used to drive the slave axis, which is the axis in gear mode. The second is the gear source, which is either actual (the encoder position of the master axis), or commanded (the commanded position of the master axis). The third is the gear ratio, which specifies the direction and ratio of master gear counts to slave counts.

Normally, the slave axis is set to an axis different than the master axis. One allowed exception is when step motors are being used. In this case, the master axis may be set to the same axis as the slave, as long as the gear source is set to encoder. For servo motors, the master axis must be a different axis than the slave axis.

Note that for ION and the MC58113, the "auxiliary axis" is treated as the second axis.

Figure 4-9 shows the arrangement of encoders and motor drives in a typical electronic gearing application.

Figure 4-9: Electronic Gear Profile



A positive gear ratio value means that during an increase in either the master axis actual or commanded position, the slave commanded position will also increase. A negative gear ratio value has the opposite effect: increasing master position will result in decreasing slave axis commanded position.

For example, assume the slave axis is axis #1 and the master axis is set to axis #4. Also, assume the source will be actual with a gear ratio of –1/2. Then for each positive encoder count of axis 4, axis 1 commanded position will decrease in value by 1/2 count, and for each negative encoder count of axis 4, axis 1 commanded position will increase in value by 1/2 count.

The electronic gear profile requires two axes to be enabled. The single-axis motion control ICs do not support electronic gearing.

If the master axis source is set to actual, then this axis need not have a physical motor attached to it. Frequently, it is used only for its encoder input, for example, from a directly driven (open-loop) motor, or a manual control. It is possible to drive a motor on the master axis by enabling the axis and applying a profile mode *other* than electronic gear to the axis. The effect of this arrangement is that both master and slave can be driven by the same profile, even though the slave can drive at a different ratio and in a different direction if desired. The master axis will operate the same, whether or not it happens to be the master for some other geared axis. The "optional" components shown in Figure 4-9 illustrate this arrangement. Such a configuration can be used to perform useful functions such as linear interpolation of two axes.

The gear ratio parameter may be changed while the axis is in motion, but care should be taken to select ratios so that safe motion is maintained.

Note that unlike the trapezoidal, S-curve, and velocity contouring profile modes, electronic gearing profile mode does not have an explicit sense of whether motion is "completed" or not. Therefore, the "motion complete" bit of the Event Status register, as well as the "in-motion" bit of the Activity Status register, do not function when in this profile mode.

For selected ION modules and the MC58113, electronic gear can also be used with a pulse & direction input signal for the auxiliary encoder axis. In this mode, all the standard electronic gear commands are used, with one master axis input 'pulse' being equivelent to one master axis input encoder 'tick'. To operate in this mode the master axis number is set to #2 (auxiliary axis) and the encoder source is set to pulse & direction. See section 10.0 for more information on the SetEncoderSource command.

4.6 The SetStopMode Command

Normally, each of the trajectory profile modes will execute the specified trajectory, within the specified parameter limits, until the profile conditions are satisfied. For example, for the point-to-point profile modes this means that the profile will move the axis until the final destination position has been reached, at which point the axis will have a velocity of zero (0).

In some cases, it may be necessary to halt the trajectory manually, either for safety reasons, or simply to achieve a specific profile. This may be accomplished using one of two methods: *Abrupt Stop* or *Smooth Stop*.

To perform a stop, the command SetStopMode is used. To retrieve the current stop mode, the command GetStopMode is used. Using the SetStopMode command to set the mode to *Abrupt Stop* instantaneously stops the profile by setting the target velocity of the designated axis to zero (0). This is, in effect, an emergency stop with instantaneous deceleration.

Setting the stop mode to *Smooth Stop* brings the designated axis to a controlled stop, using the current deceleration parameter to reduce the velocity to zero (0).

In either mode, the target velocity is set to zero (0) after the SetStopMode command is executed. Before any other motion can take place, the velocity must then be reset using the SetVelocity command.

Abrupt Stop must be used with care. Sudden deceleration from a high velocity can damage equipment or cause injury.

Abrupt Stop functions in all profiles. *Smooth Stop* functions in all profiles except electronic gearing.

4.7 Disabling and Enabling the Trajectory Generator Module

There are a number of reasons why it might be desirable to disable the trajectory generator module. See Section 3.1, "Control Flow Overview," for more information on the functions of the Trajectory Generator. In addition, there are event-related actions that may result in this module being disabled. See Section 8.1, "SetEventAction Processing," for details.

If the trajectory generator module is disabled, the current commanded position will remain at its present value. All profile and other commands will be ignored. In addition, if the position loop is enabled, at the time the trajectory generator module is disabled, the position error will be set to 0 (equivalent to ClearPositionError command).

A previously disabled trajectory generator module may be re-enabled in a number of ways. If the module was disabled using the SetOperatingMode command, then another SetOperatingMode command may be issued. If the trajectory generator module was disabled as part of an automatic event-related action (see Section 8.1, “SetEventAction Processing,” for more information) then the command RestoreOperatingMode is used.

5. Position Loop

In This Chapter

- Overview
- Dual Encoder Support
- Biquad Output Filters
- Output Limit
- Motor Bias
- Disabling and Enabling the Position Loop Module

5.1 Overview

For Magellan motion control ICs that provide servo motor support, a position loop is used as part of the basic method of determining the motor command output. The function of the position loop is to match as closely as possible the commanded position, which comes from the trajectory generator, and the actual motor position. To accomplish this, the commanded value is combined with the actual encoder position to create position error, which is then passed through a digital PID-type servo filter. The scaled result of the filter calculation is the motor command, which is then passed to a "downstream" module, either the commutation/phasing module, the current loop/FOC module, or the motor output module, depending on the motor type chosen and Magellan product being used.

The overall position loop is split into two major sections, the PID loop, and the biquad filters. The PID loop generates an initial motor command, while the dual biquad filters can be used to perform various frequency-domain filtering such as notch, lowpass, and bandpass. Once the output of the PID and biquad filters is generated, it can be further limited to a prescribed range, thereby accommodating amplifiers, motors, or physical systems, as shown in Figure 5-1.

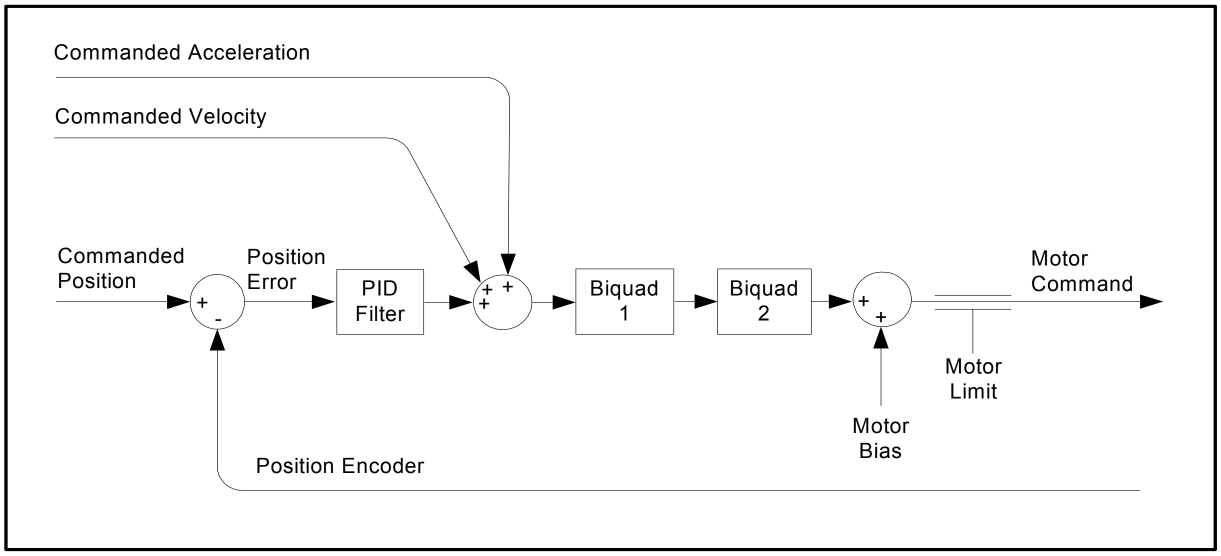


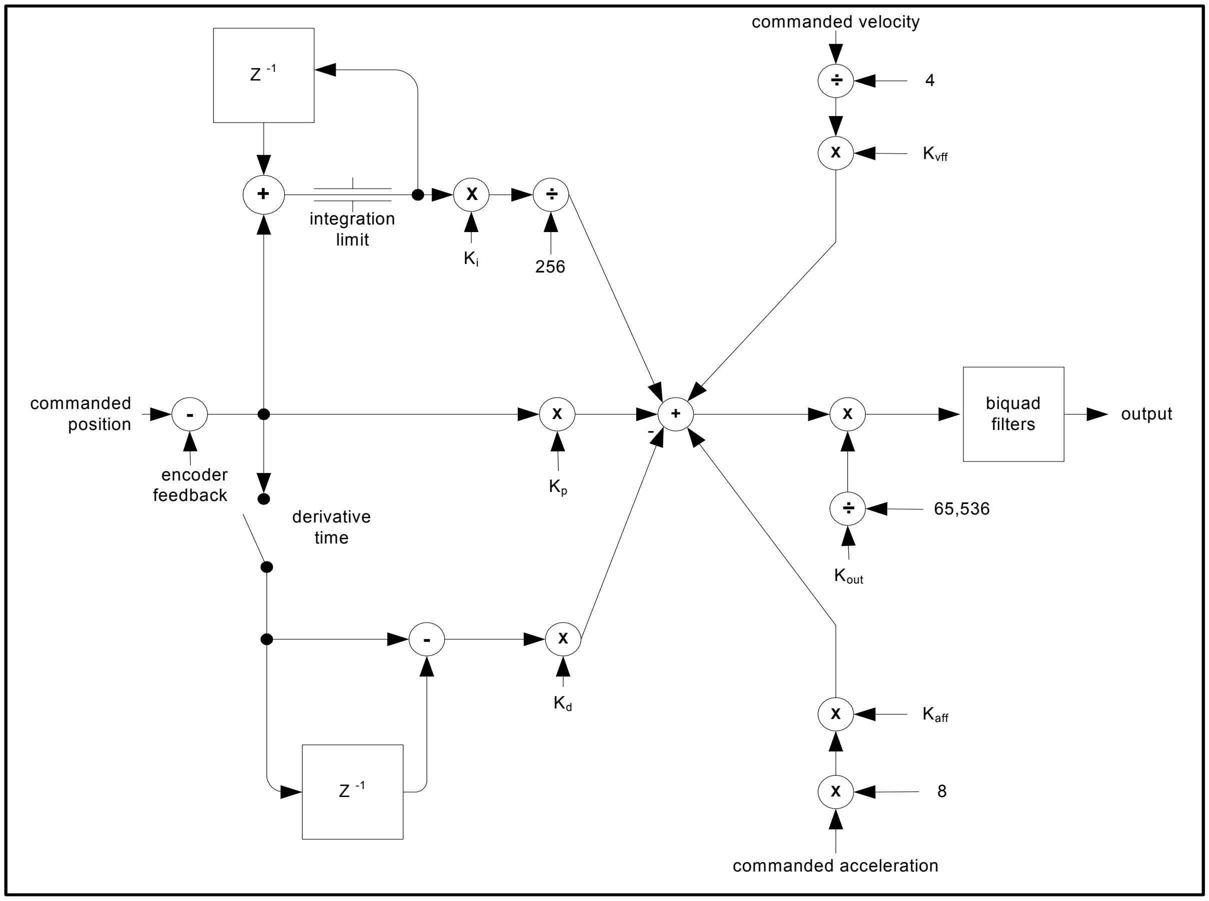
Figure 5-1: PID Loop And Biquad Filters

To perform position control, all servo applications require that safe and stable PID loop parameters be specified. Use of the dual biquad filters, on the other hand, is optional and will depend on the nature and complexity of the control problem. The more demanding the application, the more likely that the biquads will be useful.

5.1.1 PID Loop

The servo filter used with the Magellan Motion Control ICs is a proportional-integral-derivative (PID) algorithm, with velocity and acceleration feed-forward terms and an output scale factor. An integration limit provides an upper bound for the accumulated error. An optional bias value may be added to the filter calculation to produce the final motor output command. Figure 5-2 provides a control flow overview of the PID loop:

Figure 5-2: Position Loop Flow



The PID+V_{ff}+A_{ff} formula, including the scale factor and bias terms, is shown in the following equation:

$$Output_n = \left[K_pE_n + K_d(E_k - E_{k-1}) + \frac{K_i}{256} \times \sum_{j=0}^{n} E_j + K_{vff}\left(\frac{CmdVel}{4}\right) + K_{aff}(CmdAccel \times 8)\right] \times \frac{K_{out}}{65{,}536}$$

MC58113 Series ICs utilize a slightly different way of summing the integrator. The MC58113's position PID output equation is shown below:

$$Output_n = \left[K_pE_n + K_d(E_k - E_{k-1}) + \frac{1}{256} \times \sum_{j=0}^{n} (K_iE_j) + K_{vff}\left(\frac{CmdVel}{4}\right) + K_{aff}(CmdAccel \times 8)\right] \times \frac{K_{out}}{65{,}536}$$

where

k	= n – modulus (n/derivative time)
E_n	= position loop error at the sample time interval (Commanded Position – Actual Position)
E_k	= position loop error at the derivative sampling interval
K_i	= Integral Gain
K_d	= Derivative Gain
K_p	= Proportional Gain
K_{aff}	= Acceleration feed-forward
K_{vff}	= Velocity feed-forward
K_{out}	= scale factor for the output command

All filter parameters are programmable, so that the filter may be fine-tuned to any application. The parameter ranges, formats, and interpretations are shown in the following table.

Term	Name	Representation & Range
Ilimit	Integration Limit	unsigned 32 bits (0 to 2,147,483,647)
K_i	Integral Gain	unsigned 16 bits (0 to 32,767)
K_d	Derivative Gain	unsigned 16 bits (0 to 32,767)
K_p	Proportional Gain	unsigned 16 bits (0 to 32,767)
K_{aff}	Acceleration feed-forward	unsigned 16 bits (0 to 32,767)
K_{vff}	Velocity feed-forward	unsigned 16 bits (0 to 32,767)
K_{out}	Output scale factor	unsigned 16 bits (0 to 32,767)
DerivativeTime	Derivative Sampling Time	unsigned 16 bits (1 to 32,767)

To set servo parameters, use the command **SetPositionLoop**. To read back these same values, use the command **GetPositionLoop**.

5.1.2 Integration Limit

The integration limit is used to place a boundary on the absolute value that is contributed to the PID output by the integration term. Its default value after a reset is zero (0), which will result in the output from the integration term of the PID filter evaluating to zero (0). In order to use the K_i value, the integration limit must be programmed with a value greater than zero (0). For more information on the scaling of the integration limit, refer to the *Magellan Programmer's Command Reference*. As for other PID loop parameters, the integration limit can be set using the host instruction SetPositionLoop. It can be read using the command GetPositionLoop.

5.1.3 Output Scaling

The K_{out} parameter can be used to scale down the output of the PID filter by multiplying the filter result by $K_{out}/65,536$. It has the effect of increasing the usable range of Kp. The K_{out} value is set using the host instruction SetPositionLoop. It is read by using the command GetPositionLoop.

Unlike the default value of most PID loop parameters, which is zero (0), K_{out} has a power-up default value of 65,535, or 100%.

5.1.4 Derivative Sampling Time

Normally, the derivative term of the PID loop is recalculated at every servo cycle. Under some circumstances, however, it may be desirable to reduce the derivative sampling rate to a rate lower than this, to improve system stability, or simplify tuning. This can be accomplished using the command SetPositionLoop, and the value set can be read back using the command GetPositionLoop.

The specified value is the desired number of servo cycles per motion control IC sample time. For example, if the motion control IC's sample time (set using SetSampleTime command) has been set to 200 uSec (giving an effective sampling time of 204.8 uSec), a value of 1 programmed in the DerivativeTime register will result in a derivative sample time of 204.8 uSec, while a value of 10 will result in a sample time of 2.048 mSec, or once every 10 servo cycles.

Changing the derivative sample time has no effect on the overall motion control IC sample time set using the command SetSampleTime.

The default value for the derivative time is 1, meaning that by default the derivative term is calculated at each servo cycle.

5.2 Dual Encoder Support

The multi-axis MC58000 motion control ICs (MC58420, MC58320, MC58220), the ION, and MC58113 support a dual encoder PID configuration, which may be useful for applications where the position of the load is critical, but cannot be deterministically related to the motor position because of backlash or other forms of mechanical compliance. In this configuration, the encoder input for a second axis (other than the one being controlled) is incorporated as the derivative term in the servo loop as shown in Figure 5-3. For the ION and the MC58113, this second axis is known as the auxiliary encoder input.

For axes driving brushless DC motors in dual encoder mode, the auxiliary axis encoder is used for sinusoidal commutation.

Figure 5-3 provides an overall connection scheme for axes used in the dual loop configuration.

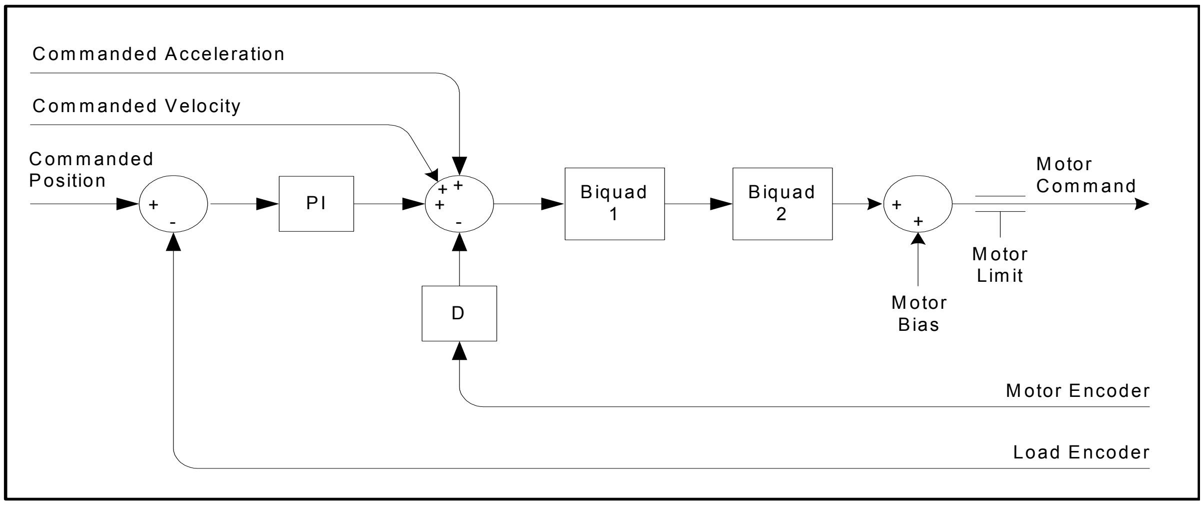


Figure 5-3: Magellan Dual-Loop Flow

5.2.1 Dual Encoder PID Loop Algorithm

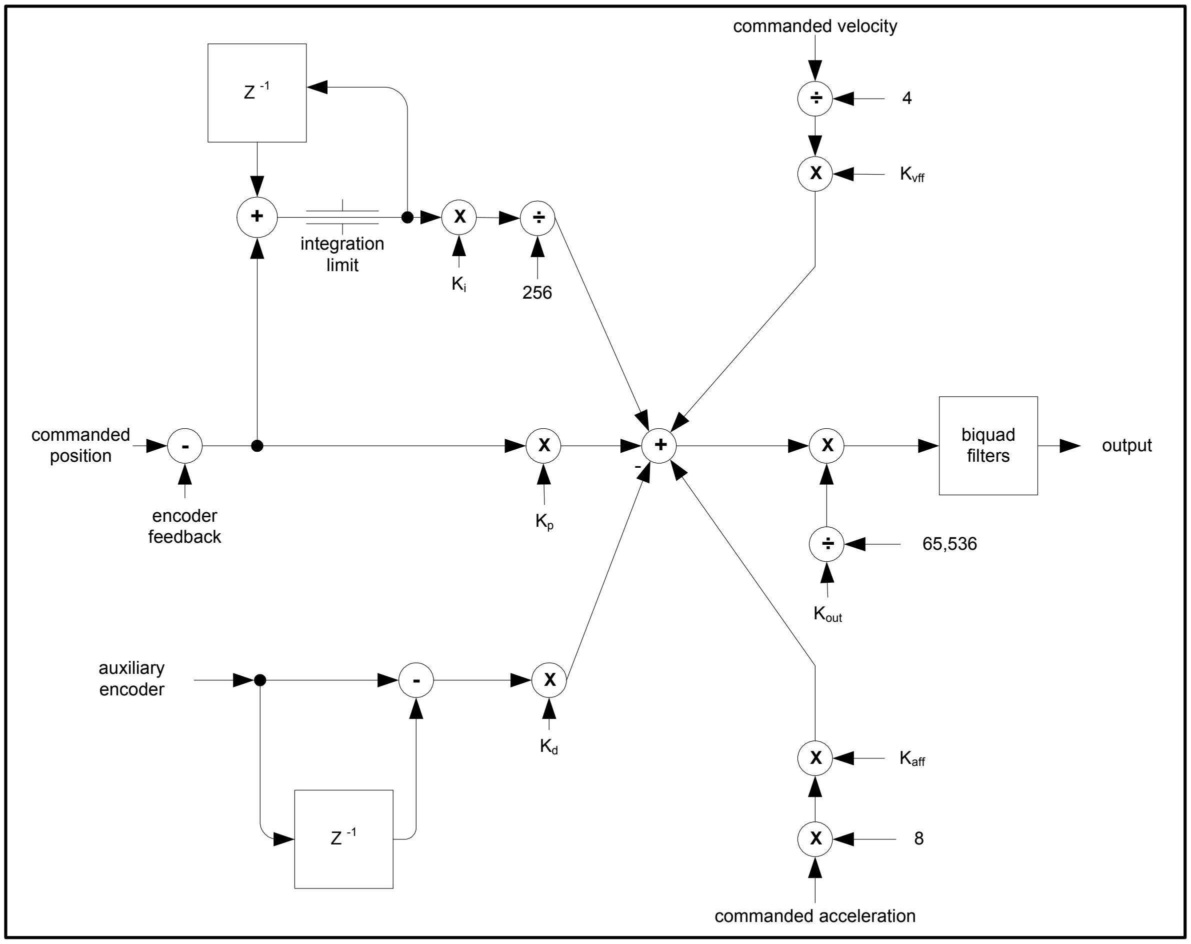
The structure of the servo filter used by the motion control IC in dual-encoder mode is slightly different from that used in single-encoder mode. The dual-encoder algorithm that follows is illustrated in Figure 5-4:

$$Output_n = \left[K_p E1_n - K_d(P_n - P_{n-1}) + \sum_{j=0}^{n} E1_j \times \frac{K_i}{256} + K_{vff}\left(\frac{CmdVel}{4}\right) + K_{aff}(CmdAccel \times 8) \right] \times \frac{K_{out}}{65,536}$$

where

$E1_n$	= the accumulated error terms from the main encoder
P_n	= position of the auxiliary encoder
K_i	= Integral Gain
K_d	= Derivative Gain
K_p	= Proportional Gain
K_{aff}	= Acceleration feed-forward
K_{vff}	= Velocity feed-forward
K_{out}	= scale factor for the output command

Figure 5-4: Magellan Dual-Loop Digital Filter



5.2.2 Configuring Dual Encoder Support

The SetAuxiliaryEncoderSource command is used to enable dual encoder processing for an axis. The AuxiliaryAxis parameter controls which axis' encoder input will be used to augment the primary (Load) encoder. The mode parameter enables or disables dual loop processing. Section 5.1.1, "PID Loop," describes how the servo processing loop functions when dual loop is disabled (the default condition).

If the application needs to determine the actual position of the auxiliary encoder, use the GetActualPosition command, and specify the axis of the auxiliary encoder.

The auxiliary encoder should always have a resolution which is greater than or equal to the resolution of the main encoder to avoid unstable operation in dual loop mode.

For MC50000 products the total number of encoder inputs supported by a given Magellan Motion Control IC is equal to the number of control axes supported. In other words, a four-axis motion control IC has four input encoder channels. Using an encoder channel for dual-loop, therefore, reduces the total number of available control axes by one. ION and the MC58113 is different in that they are a single axis products, but have an additional "auxiliary" encoder input channel.

5.3 Biquad Output Filters

A biquad is a generic digital filter structure. With the proper coefficients, it can be programmed to be a low-pass filter, high-pass filter, band-pass filter, notch filter, or custom filter. Programs such as Octave (www.octave.org) may be used to find the coefficients.

The Magellan Motion Control IC supports two programmable biquad output filters for each axis. These filters are chained. When both are enabled, the output of **Filter1** feeds the input of **Filter2.** If **Filter1** is disabled (the default state), the entire filter chain is bypassed, and the motion control IC output passes unfiltered to the motor.

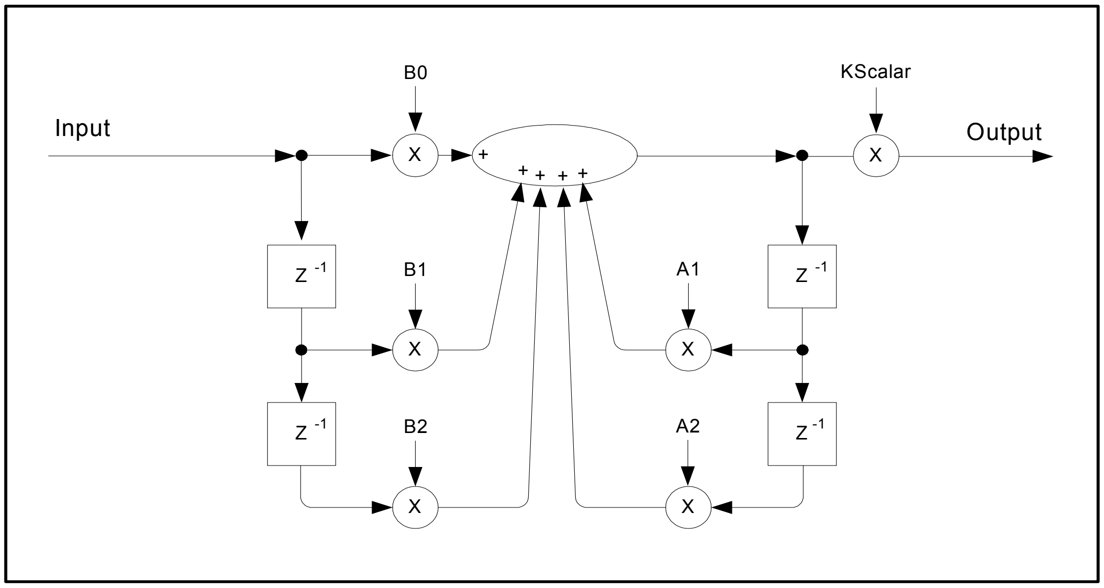


Figure 5-5: Biquad Algorithm Flow

The output of the filter at time *n* is determined with the following equation:

$$Y_n = K \times (B_0 \times X_n + B_1 \times X_{n-1} + B_2 \times X_{n-2} + A_1 \times Y_{n-1} \times A_2 \times Y_{n-2})$$

where:

Y_n	= output of the filter at time n
X_n	= input to the filter at time n
K	= positive scaling value used to avoid rounding errors
B_0	= programmable biquad coefficient
B_1	= programmable biquad coefficient
B_2	= programmable biquad coefficient
A_1	= programmable biquad coefficient
A_2	= programmable biquad coefficient

The following table shows biquad filter coefficients that are set using the command **SetPositionLoop**. They can be read back using the command **GetPositionLoop**. The following table summarizes the representation and range for the settable biquad parameters.

Term	Name	Representation & Range
K	biquad scalar	unsigned 16 bits (0 to 32,767)
A_1	coefficient A_1	signed 16 bits (–32,768 to 32,767)
A_2	coefficient A_2	signed 16 bits (–32,768 to 32,767)
B_0	coefficient B_0	signed 16 bits (–32,768 to 32,767)
B_1	coefficient B_1	signed 16 bits (–32,768 to 32,767)

Term	Name	Representation & Range
B_2	coefficient B_2	signed 16 bits (–32,768 to 32,767)

5.3.1 Determining Biquad Coefficients

Typically, coefficients used in biquad filter equations are small floating point numbers. To avoid rounding errors when storing these numbers as 16-bit values, the K coefficient is scaled by 2^{27} to allow the other coefficients to be entered as integers.

For example, in Octave, the coefficients for a second-order butterworth filter can be found as:

```
[b,a] = butter(2,0.1)
```

This results in the coefficients:

```
b0 = 0.020083
b1 = 0.040167
b2 = 0.020083
a1 = –1.56102
a2 = 0.64135
```

If the motion control IC's filter equation (shown at the top of this page) is compared to the filter equation used by Octave (type **help filter** in Octave), there is a slight difference in that the a_x components are subtracted in Octave, as opposed to being added in the motion control IC. The result is that the $\mathbf{a_1}$ and $\mathbf{a_2}$ coefficients from Octave (or Matlab) need to be multiplied by **–1** before being sent to the motion control IC. This results in:

```
b0 = 0.020083
b1 = 0.040167
b2 = 0.020083
a1 = 1.56102
a2 = –0.64135
```

Once the output scaling factor K is determined, these values will be scaled and set as the output filter coefficients.

5.3.2 Determining the Biquad Scaling Factor

To obtain maximum output precision, the programmable scaling value K should be set to a value which will scale the largest absolute value of the set of coefficients (a_1 in this case) closest to 32,767 (the largest positive value for a 16-bit signed integer). Note that in this step, if the coefficients are not chosen correctly there is the potential to create an overflow result. Using the coefficient values on the previous page (and accounting for the 2^{27} internal scaling factor), K is determined using this equation.

$$a_1 = K * 32767 * 2^{-27}$$

which can be easily resolved:

restated as:	$K = (a_1 * 2^{27}) / 32767$
substituting:	$K = (1.56102 * 2^{27}) / 32767$
evaluates to:	$K = 6394$

5.3.3 Scaling Biquad Coefficients

Once the optimal K scaling factor has been determined, the integer equivalents of the biquad coefficients must be calculated. These integer values (named B_0, B_1, B_2, A_1, and A_2) are calculated as shown in the following example.

	$b_0 = B_0 * K * 2^{-27}$
restated as:	$B_0 = (b_0 * 2^{27}) / K$
substituting:	$B_0 = (0.020083 * 2^{27}) / 6394$
evaluates to:	$B_0 = 422$

Using this formula ($X = (x * 2^{27}) / K$) for each of the coefficients yields:

$B_0 = 422$
$B_1 = 843$
$B_2 = 422$
$A_1 = 32767$
$A_2 = -13463$
$K = 6394$

5.4 Output Limit

The motor output limit prevents the filter output from exceeding a boundary magnitude in either direction. If the filter produces a value greater than the limit, the motor command takes the limiting value. The motor limit value is set using the host instruction SetMotorLimit. It is read by using the command GetMotorLimit. The value specified is a 16-bit unsigned number with a range of 0 to 32,767. The specified value is the maximum magnitude that will be output to the motor. For example, if the motor limit is set to 30,000 (or 91.6% output), then motor values greater than 30,000 will be output as 30,000, and motor values less than –30,000 will be output as –30,000.

The motor limit applies only when the position loop is enabled. In the case that the position loop is disabled, the output limit does not affect either the value passed on from the trajectory generator, or the value specified from the Motor Command register.

The default value of the motor command limit is 32,767, which corresponds to 100% output.

5.5 Motor Bias

When an axis is subject to a net external force in one direction (such as a vertical axis pulled downward by gravity), the servo filter can compensate for it by adding a constant DC bias to the filter output. As for the regular PID values, the bias value is set using the host instruction SetMotorBias. It can be read using the command GetMotorBias.

The motor bias is applied at all times that the position loop is enabled. If the position loop is disabled but the trajectory generator is enabled, the motor bias is also applied at all times.

If the position loop is disabled and the trajectory generator is also disabled, then the motor bias is applied only after a transition to this state, and any subsequent setting of the Motor Command register will be applied without motor bias. See Section 5.6, “Disabling and Enabling the Position Loop Module,” for more information on setting the Motor Command register. For example, if a motor bias value of +1,000 (= ~+3%) has been set, at the time a SetOperatingMode command is given to disable trajectory generator and position loop, or at the time a safety-related response such as motion error occurs and these modules are automatically disabled, the output motor command will be +1,000. If, in the process of recovering from this condition, a user sets the motor command to +2,000, this value, unaffected by the motor bias, will be output.

The default value of motor bias is zero (0).

If the specified bias value does not properly compensate for the external force, the axis may move suddenly in one direction or another after a SetOperatingMode command. It is the responsibility of the user to select a motor bias value which will maintain safe motion.

5.6 Disabling and Enabling the Position Loop Module

There are a number of reasons why it might be desirable to disable the position loop module. See Section 3.1, “Control Flow Overview,” for details. In addition, there are event-related actions that may result in this module being disabled. See Section 8.1, “SetEventAction Processing,” for details.

If the position loop module is disabled, the overall control flow of the motion control IC module will be altered in one of two ways, depending on whether the trajectory generator is enabled or disabled when the position loop module is disabled.

Trajectory generator disabled — If the trajectory generator module is disabled when the position loop is disabled, the output of the position loop module will be a 16-bit word derived from a programmable motor command register, set using SetMotorCommand, and read back using GetMotorCommand.

Trajectory generator enabled — If the trajectory generator module is enabled when the position loop is disabled, then the position loop module is bypassed, and the output value of the trajectory generator becomes the motor command value. Note that only the low word of the 32-bit commanded position is used. This is because the output of the trajectory generator is a 32-bit commanded position, while the output of the position loop, and therefore the input to subsequent modules, is a 16-bit motor command. Effectively this means that when used in this mode, the range of the trajectory generator is limited to 16 bits (–32,768 to +32,767).

Regardless of whether the position loop module is enabled or disabled, the actual effective motor command value may be queried using the command GetActiveMotorCommand. This value will indicate the motor command being sent to “downstream” processing modules such as commutation, current control (ION only), or motor output.

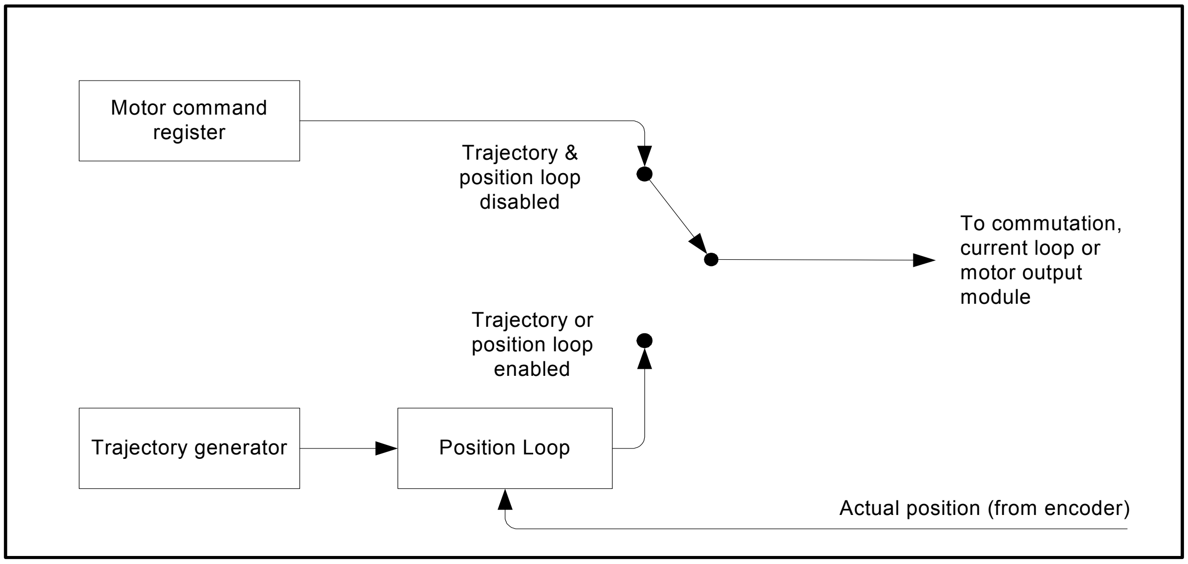

Motor command register
Trajectory & position loop disabled
To commutation, current loop or motor output module
Trajectory or position loop enabled
Trajectory generator
Position Loop
Actual position (from encoder)

Figure 5-6: Motor Control Paths, Trajectory Enabled/ Disabled

A previously disabled position loop module may be re-enabled in a number of ways. If the module was disabled using the **SetOperatingMode** command, then another **SetOperatingMode** command may be issued. If the position loop module was disabled as part of an automatic event-related action (see Section 8.1, "SetEventAction Processing," for more information) then the command **RestoreOperatingMode** is used.

Regardless of how the module is re-enabled, at the time that the re-enable operation is requested, certain special processing occurs to avoid unexpected axis movement. In particular, all position loop state variables are set to zero(0).

5.6.1 Reading Position Loop Values

As indicated in this chapter and others, there are a number of commands which can be used to read various position loop-related values. These include **GetCommandedPosition** to read the input command of the position loop, **GetActualPosition** to read the actual encoder position, **GetPositionError** to read the difference between these two quantities, and **GetActiveMotorCommand** to read the output of the position loop module. In addition to being readable through these commands, these variables can also be selected for tracing. See Section 8.8, "Trace Capture," for details on Magellan's capture trace facility.

Beyond these registers, to further facilitate tuning, there are a number of internal position loop values that can be read back as well as traced. To read back these values the command **GetPositionLoopValue** is used.

The variables within the position loop that can be read and traced are summarized in the following table.

Variable Name	Function
Integrator Sum	This register holds the sum of the integrator for the position PID loop.
Integral Contribution	This register holds the overall contribution of the integrator to the position PID loop.
Derivative	This register holds the position error derivative value. That is, the difference between the present position error and the previous one.
BiQuad1 Input	This register holds the input value to biquad filter 1.
BiQuad2 Input	This register holds the input value to biquad filter 2.

This page intentionally left blank.

6. Parameter Update and Breakpoints

In This Chapter

- Parameter Buffering
- Breakpoints

6.1 Parameter Buffering

Various parameters must be specified to the motion control IC for an axis to be correctly controlled. Some situations may require that a set of parameters take effect at the exact same time to facilitate precise synchronized motion. To support this, all profile parameters, most position loop parameters, and most current loop parameters, are loaded into the motion control IC using a buffered scheme. These buffered commands are stored in an area of the motion control IC that does not affect the actual motion control IC behavior until an Update event occurs. An Update results in buffered registers being copied to the active registers, thereby causing the motion control IC to act on the new parameters.

There are three separate types of buffered registers, each of which may be independently updated and made active. They are the profile parameters, the position loop parameters, and the current loop parameters. The command SetUpdateMask controls which of these three types will be copied upon receipt of an Update command. The command GetUpdateMask is used to read back the specified mask value. Separately updateable parameters are useful because they allow (for example) the profile parameters to be modified on-the-fly without affecting the servo parameters, or the current loop parameters to be changed without altering the position loop parameters. Many applications will not need this complexity, however, and it is not uncommon for the mask to be set just once, so that all three buffered register types are always updated.

The following command sequence illustrates the loading and updating of a set of parameters. In this case, a new profile mode, position, velocity, and acceleration (all of which are buffered commands) are loaded, followed by an update command to make them active.

```
SetProfileMode Axis1, trapezoidal      // set profile mode to trapezoidal for axis 1
SetPosition Axis1, 12345               // load a destination position for axis 1
SetVelocity Axis1, 223344              // load a velocity for axis 1
SetAcceleration Axis1, 1000            // load an acceleration for axis 1
SetUpdateMask Axis1, Profile           // specify that an update of profile parameters only
                                       // is to occur
Update                                 // Double buffered registers are copied into
                                       // the active registers, thereby initiating the move
```

In this sequence, the trajectory profile mode will actually be changed to trapezoidal and the specified parameters loaded into the trajectory generator only when the Update occurs. At that point the trajectory generator will begin the programmed motion.

The value set using the SetUpdateMask command is only altered by a subsequent SetUpdateMask command. That is, its value is not affected by the Update command itself or by any other commands.

The default value of the update mask for MC50000 motion control ICs is to update profile parameters and servo parameters. The default value for ION, the MC58113, and Atlas-connected axes is to update profile parameters, servo parameters, and current loop parameters.

6.1.1 Updates

Including the manual Update command described in Section 6.1, "Parameter Buffering," there are three different ways in which an update can occur.

1 Update command—The simplest way is to give an Update command as described in Section 6.1, "Parameter Buffering." This causes the parameters for the programmed axis to be updated immediately.

2 MultiUpdate command—The MultiUpdate command causes multiple axes to be updated simultaneously. This can be useful when synchronized multi-axis profiling is desired. This command takes a one-word argument which consists of a bit mask, with one bit assigned to each axis of the chipset. Note that for single axis Magellans this command will have no utility beyond the standard Update command. Executing the MultiUpdate command has the same effect as sending a set of Update commands to each of the individual axes selected in the MultiUpdate command mask. As was the case with the Update command, the command SetUpdateMask is used for each separate axis to be updated to control which parameters are copied from the buffered registers.

3 Breakpoints—A very useful facility supported by the motion control IC which may be programmed to generate an Update command automatically when a pre-programmed condition becomes true. The breakpoint facility is useful for performing operations such as "automatically change the velocity when a particular position is reached," or "stop the axis abruptly when a particular external signal goes active." Unlike the standard manual Update command, however, transfer from buffered to active registers during a breakpoint is controlled by the SetBreakpointUpdateMask command. See Section 6.2, "Breakpoints" for detailed information.

All three of the update methods execute in the same manner. At the time the update occurs, the selected type of buffered registers and commands are copied to the active registers. While most commands take place instantaneously upon an Update, it should be noted that depending on the calculations previously performed in the servo loop, these values may not be used until the next cycle. Also, as was illustrated in the preceding example, before the update occurs, sending buffered commands will have no effect on the system behavior.

In addition to profile generation, most servo parameter commands, most current control parameters, as well as a few other commands are buffered. The following table provides a complete list of buffered commands and values.

Double Buffered Commands

Trajectory	Position Loop	Current Loop
SetProfileMode	SetPositionLoop	SetCurrentLoop
ClearPositionError	SetMotorCommand	SetFOCLoop
SetStopMode		SetCurrentControlMode
SetPosition		
SetVelocity*		
SetAcceleration		
SetDeceleration		
SetJerk		
SetGearMaster		
SetGearRatio		

*Note that SetStartVelocity is not double-buffered.

6.2 Breakpoints

Breakpoints are a convenient way of programming a motion control IC event based upon a specific condition. Depending on the breakpoint instruction's arguments, a breakpoint can cause an update, an abrupt stop, a smooth stop, it can disable specific modules, or it can cause no action at all.

Each Magellan axis contains two breakpoints that may be programmed for that axis. In this manner, two completely separate conditions may be monitored and acted upon. These two breakpoints are known as breakpoint 1 and breakpoint 2. In addition, the buffered registers that are to be copied can be independently specified for each breakpoint using the command **SetBreakpointUpdateMask**. This value can be read back using the command **GetBreakpointUpdateMask**.

6.2.1 Defining a Breakpoint

Each breakpoint consists of six components: the breakpoint axis, the source axis for the triggering event, the event itself, the action to be taken, the *breakpoint update mask* associated with the action, and the comparison value. These components are described in the following table.

Breakpoint component	Description
Breakpoint axis	The axis on which the specified action is to be taken.
Source axis	The axis on which the triggering event is located. It can be the same as or different than the breakpoint axis. Any number of breakpoints may use the same axis as a *source axis*.
Trigger	The event that causes the breakpoint.
Action	The sequence of operations executed by the motion control IC when the breakpoint is triggered. After a breakpoint is triggered, the *action* is performed on the breakpoint axis, using the breakpoint update mask specified for the *breakpoint axis*.
Breakpoint update mask	The mask that controls whether profile and/or position loop and/or current loop parameters will be transferred into the active registers upon occurrence of a breakpoint trigger.
Comparison value	The value used in conjunction with the action to define the breakpoint event.

These parameters provide great flexibility in setting breakpoint conditions. By combining these components, almost any event on any axis can cause a breakpoint.

The command used to send the breakpoint axis, the trigger, the source axis, and the action is **SetBreakpoint**. To retrieve these values, the command **GetBreakpoint** is used.

Upon receipt of the **SetBreakpoint** command, the breakpoint will become active. That is, the motion control IC will begin to compare the conditions specified by the breakpoint with the actual conditions present in the motion control IC. This means that any other required information for the breakpoint to function (such as comparison value and breakpoint update mask) should already be loaded before this command is sent. See Section 6.2.7, "Breakpoint Examples" for examples of how to program breakpoints.

To set the comparison value, the command **SetBreakpointValue** is used. This comparison value can be retrieved using the command **GetBreakpointValue**. For each of these commands, the breakpoint number (1 or 2) must be specified. To specify the *breakpoint update mask* for a given breakpoint, the command **SetBreakpointUpdateMask** is used. This value can be read back using the command **GetBreakpointUpdateMask**.

The **SetBreakpointMask** and **SetBreakpointValue** commands should always be sent before the **SetBreakpoint** command when setting up a particular breakpoint.

6.2.2 Breakpoint Triggers

The Magellan Motion Control ICs support the following breakpoint trigger conditions.

Trigger Condition	Level or Threshold	Description
Greater or equal commanded position	threshold	Satisfied when the current commanded position is equal to or greater than the programmed compare value.
Lesser or equal commanded position	threshold	Satisfied when the current commanded position is equal to or less than the programmed compare value.
Greater or equal actual position	threshold	Satisfied when the current actual position is equal to or greater than the programmed compare value.
Lesser or equal actual position	threshold	Satisfied when the current actual position is equal to or less than the programmed compare value.
Commanded position crossed	threshold	Satisfied when the current commanded position crosses (is equal to) the programmed compare value.
Actual position crossed	threshold	Satisfied when the current actual position crosses (is equal to) the programmed compare value.
Time	threshold	Satisfied when the current motion control IC time (in number of cycles since power-up) is equal to the programmed compare value.
Event status	level	Satisfied when the Event Status register matches bit mask and high/low pattern in programmed compare value.
Activity status	level	Satisfied when the Activity Status register matches bit mask and high/low pattern in programmed compare value.
Drive status	level	Satisfied when the Drive Status register matches bit mask and high/low pattern in programmed compare value.
Signal status	level	Satisfied when the Signal Status register matches bit mask and high/low pattern set in programmed compare value.
None	—	Disables any previously set breakpoint.

To de-activate a breakpoint, specify "none" for the breakpoint trigger. Only one of the triggers listed in the preceding table may be selected at a time. See Section 6.2.4, "Level-Triggered Breakpoints" for detailed information.

6.2.3 Threshold-Triggered Breakpoints

Threshold-triggered breakpoints use the value set by the SetBreakpointValue command as a single 32-bit threshold value to which a comparison is made. When the comparison is true, the breakpoint is triggered.

For example, if it is desired that the trigger occur when the commanded position is equal to or greater than 1,000,000, then the comparison value loaded using SetBreakpointValue would be 1,000,000, and the trigger selected would be GreaterOrEqualCommandedPosition.

6.2.4 Level-Triggered Breakpoints

To set a level-triggered breakpoint, the host instruction supplies two 16-bit data words: a trigger mask, and a sense mask. These masks are set using the SetBreakpointValue instruction. The high word of data passed with this command is the trigger mask value and the low word is the sense mask value.

The trigger mask determines which bits of the selected status register are enabled for the breakpoint. A value of one in any position of the trigger mask enables the corresponding status register bit to trigger a breakpoint. A value of zero (0) in the trigger mask disables the corresponding status register bit. If more than one bit is selected, then the breakpoint will be triggered when any selected bit enters the specified state.

The sense mask determines which state of the corresponding status bits causes a breakpoint. Any status bit that is in the same state (i.e., 1 or 0) as the corresponding sense bit, is eligible to cause a breakpoint (assuming that it has been selected by the trigger mask).

For example, if the Activity Status register breakpoint has been selected, and the trigger mask contains the value 0402h and the sense mask contains the value 0002h, then the breakpoint will be triggered when bit 1 (the "at max velocity" indicator) assumes the value 1, or bit 10 (the "in motion" indicator) assumes the value zero (0).

6.2.5 Breakpoint Actions

Once a breakpoint has been triggered, the motion control IC may be programmed to perform one of the following instruction sequences.

Action	Description
None	No action taken, however breakpoint bit in Event Status register still set.
Update	Will transfer the double buffered registers specified by the SetBreakpointUpdateMask command.
Smooth Stop	Causes a smooth stop to occur at the current active decelaration rate. Velocity command will be set to zero (0) after breakpoint occurs.
Abrupt Stop	Causes an instantaneous halt of the trajectory generator. Velocity command will be set to zero (0) after breakpoint occurs.
Abrupt Stop with Position Error Clear	Causes an instantaneous halt of the trajectory generator as well as a zeroing of the position error (equivalent to ClearPositionError command). Velocity command will be set to zero (0) after breakpoint occurs.
Disable Position Loop & Higher Modules	Disables trajectory generator and position loop module.
Disable Current Loop & Higher Modules	Disables trajectory generator, position loop, and current loop modules.
Disable Motor Output & Higher Modules	Disables trajectory generator, position loop, current loop, and motor output modules.

For the Update action, the SetBreakpointUpdateMask determines which type of double buffered register will be updated upon occurrence of the breakpoint. The value set using SetBreakpointUpdateMask is not altered by occurrence of a breakpoint, thus it may be set once and left at the same value, or changed any number of times as the application requires.

For MC50000, the default value of SetBreakpointUpdateMask is to update trajectory and position loop parameters. For Magellans that support a current loop the default value of SetBreakpointUpdateMask is to update trajectory, position loop, and current loop parameters.

For specified actions that alter the operating mode, to restore normal operation the user must send a RestoreOperatingMode command.

Once a breakpoint condition has been satisfied, the Event Status bit that corresponds to the breakpoint is set, and the breakpoint is deactivated.

6.2.6 Breakpoint Latencies

The latency after a breakpoint condition exists and before the breakpoint occurs depends on the condition selected. Breakpoints that are conditional on internal Magellan registers will have a latency equivalent to the sample time. Breakpoints that are conditional on external Magellan signals will have a latency equivalent to twice the sample time.

6.2.7 Breakpoint Examples

Here are a few examples to illustrate how breakpoints may be used.

Example #1: The host would like axis 1 to change velocity when the encoder position reaches a particular value. Breakpoint #1 should be used.

This is accomplished using the following command sequence. Note that this sequence assumes that the UpdateMask has been left at its default value of trajectory and position loop update.

```
SetPosition Axis1, 123456              // Load destination
SetVelocity Axis1, 55555               // Load velocity
SetAcceleration Axis1, 500             // Load acceleration
SetDeceleration Axis1, 1000            // Load deceleration
Update Axis1                           // Make the move. Profile and position loop parameters are
                                       // updated, of which only profile parameters have been changed.
SetVelocity Axis1, 111111              // Load a new velocity of 111,111 but do not send an Update
SetBreakpointValue Axis1,              // Load 100,000 into the comparison register for breakpoint 1
   0, 100000
SetBreakPointUpdateMask Axis 1,        // Update profile parameters only
   0, Profile
SetBreakpoint Axis1,                   // Specify a positive actual position breakpoint on axis 1
   0, Axis1, Update,                   // which will result in an Update when satisfied for
   GreaterOrEqualActualPosition        // breakpoint 1. Note that this is the last command in the
                                       // breakpoint definition sequence, since this is the command
                                       // that results in the breakpoint comparison being initiated.
```

This sequence makes an initial move, and loads a breakpoint after the first move has started. The defined breakpoint will result in the velocity being updated to 111,111 when the actual position reaches a value of 100,000. Therefore, at 100,000, the axis will accelerate from a velocity of 55,555 to 111,111 with an acceleration value of 500. Note that any buffered values that are not re-sent will remain in the buffered registers. When the breakpoint performs an Update, the values for position, acceleration, and deceleration are unchanged and are therefore copied over to the active registers without modification.

Example #2: The host would like axis 1 to perform an emergency stop whenever the *AxisIn* signal for axis 3 goes high. In addition, the axis 1 acceleration and derivative gain factor should change whenever a particular commanded position is achieved on axis 4. This is accomplished using the following command sequence.

```
SetPosition Axis1, 123456              // Load destination
SetVelocity Axis1, 55555               // Load velocity
SetAcceleration Axis1, 500             // Load acceleration
SetDeceleration Axis1, 1000            // Load deceleration
Update Axis1                           // Make the move (assumes that
                                       // updatemask already set)

SetBreakpointValue Axis1,              // Load mask and sense word of 0x40, 0x40
   0, 0x400040                         // (bit 6 must be high) for breakpoint 1
```

```
SetBreakPointUpdateMask Axis 1,            // Update profile parameters
   0, Profile
SetBreakpoint Axis1, 0, Axis3,             // Specify a breakpoint to monitor the signal
   AbruptStop, SignalStatus                // status register of axis 3 to trigger when bit 6
                                           //(AxisIn) goes high for breakpoint 1
SetAcceleration Axis1, 111111              // Load a new acceleration of 111,111 but do
                                           // not send an Update
SetPositionLoop Axis1, Kd, 1250            // Load a new Kd
SetBreakpointValue Axis1, 1, 100000        // Load 100,000 into the comparison register
                                           // for breakpoint 2
SetBreakPointUpdateMask                    // Update profile parameters & Position Loop
   Axis1, 1, Profile 1 Position Loop       // for breakpoint 2

SetBreakpoint Axis1, 1, Axis4,             // Specify a positive commanded position
   Update, PositiveCommandedPosition       // breakpoint on axis 4 which will result in an
                                           // Update when satisfied for breakpoint 2
```

This sequence is similar to the previous example, except that an additional breakpoint has been defined which causes the abrupt stop. In addition, these breakpoints have been set to be triggered by events on axes 3 and 4. Both of these breakpoints were defined after the primary move was started. This may not be necessary depending on when the breakpoint is expected to occur. Breakpoints should be set to occur after the primary move, because there is only one set of buffered registers. It is impossible to load primary move parameters (position, velocity, etc.), and also breakpoint profile parameters (the profile parameters that take effect once the breakpoint occurs) before the primary move is updated.

This page intentionally left blank.

7. Status Registers

In This Chapter

- Overview
- Event Status Register
- Activity Status Register
- Drive Status Register (ION, MC58113-Series, and Atlas-Connected Axes Only)
- Signal Status Register

7.1 Overview

The Magellan Motion Control IC can monitor almost every aspect of the motion of an axis. There are various numerical registers that may be queried to determine the current state of the motion control IC, such as the current actual position (GetActualPosition command), the current commanded position (GetCommandedPosition command), etc.

In addition to these numerical registers, there are four bit-oriented status registers that provide a continuous report on the state of a particular axis. These status registers conveniently combine a number of separate bit-oriented fields for the specified axis. These four 16-bit registers are Event Status, Activity Status, Drive Status, and Signal Status.

The host may query these four registers, or the contents of these registers may be used in breakpoint operations to define a triggering event such as “trigger when bit 8 in the Signal Status register goes low.” These registers are also the source of data for the AxisOut mechanism (see Section 8.1, “SetEventAction Processing,”), which allows one or more bits within these four registers to be output as a hardware signal.

7.2 Event Status Register

The Event Status register is designed to record events that do not continuously change in value but rather tend to occur once due to a specific event. As such, each bit in this register is set by the motion control IC and cleared by the host.

The Event Status register is defined in the following table.

Bit	Name	Description
0	Motion complete	Set when a trajectory profile completes. The motion being considered complete may be based on the commanded position, or the actual encoder position.
1	Position wraparound	Set when the actual motor position exceeds 7FFF FFFFh (the most positive position), and wraps to 8000 0000h (the most negative position), or vice versa.
2	Breakpoint 1	Set when breakpoint #1 is triggered.
3	Capture received	Set when the high-speed position capture hardware acquires a new position value.
4	Motion error	Set when the actual position differs from the commanded position by an amount more than the specified maximum position error. The motion control IC can be configured to stop motion automatically when this flag is set.
5	Positive limit	Set when a positive limit switch event occurs.
6	Negative limit	Set when a negative limit switch event occurs.

Bit	Name	Description
7	Instruction error	Set when an instruction error occurs. For Atlas-connected axes this also signals a communication problem between the Magellan and the Atlas.
8	Disable	Set when the user disables the controller by making the enable signal inactive (ION and MC58113 only). See Section 15.12, "Drive Enable Signal," for more information.
9	Overtemperature fault	Set when an overtemperature fault occurs (ION, MC58113, and Atlas-connected axes only).
10	Drive Exception	Set when one of a number of drive exceptions, such as bus overvoltage or undervoltage fault occurs (ION, MC58113, and Atlas-connected axes only).
11	Commutation error	Set when a commutation error occurs (MC58000, ION only).
12	Current foldback	Set when current foldback occurs (ION, MC58113, and Atlas-connected axes only).
13	*Reserved*	*May contain 1 or 0.*
14	Breakpoint 2	Set when breakpoint #2 is triggered.
15	*Reserved*	*May contain 0 or 1.*

The command **GetEventStatus** returns the contents of the Event Status register for the specified axis.

Bits in the Event Status register are latched. Once set, they remain set until cleared by a host instruction or a system reset. Event Status register bits may be reset to 0 by the instruction **ResetEventStatus**, using a 16-bit mask. Register bits corresponding to 0s in the mask are reset; all other bits are unaffected.

The Event Status register may also be used to generate a host interrupt signal using the **SetInterruptMask** command. See Section 8.10, "Host Interrupts," for more information.

7.2.1 Instruction Error

Bit 7 of the Event Status register indicates an instruction error. Such an error occurs if an otherwise valid instruction or instruction sequence is sent by the host when the Magellan's current operating state makes the instructions invalid. Instruction errors can occur at the time the instruction is issued or at the time of an update.

Should an instruction error occur, the invalid parameters are ignored, and the **Instruction Error** indicator of the Event Status register is set. While invalid parameters checked at the time of the update are ignored, valid parameters are sent on. This can have unintended side effects depending on the nature of the motion sequence, so all instruction error events should be treated seriously.

In the following example, the negative velocity is not valid in the new profile mode.

```
SetProfileMode (axis2, Velocity)       // Set the profile mode to velocity contouring
SetVelocity (axis2, –4387)             // Set the velocity to a negative value
SetUpdateMask(axis2, Trajectory)       // Set the update mask
Update (axis2)                         // Perform the Update
SetProfileMode (axis2, Trapezoidal)    // Change the profile mode to trapezoidal
SetPosition (axis2, 123456)            // Load a position
Update (axis2)                         // Perform the Update
```

The **Update** is executed, but the **Instruction Error** bit is set. Legitimate parameters such as position are updated, and profile generation continues.

For non Atlas-connected axes, to determine the nature of the instruction error a **GetInstructionError** command is sent. This command indicates the specific nature of the most recent instruction error.

For Atlas-connected systems, an instruction error may also indicate a communication problem between the Magellan and the connected Atlas Amplifier. This error may occur with a specific command sent by the host to the Magellan, or it may occur as part of continuous background communications between the Magellan and the Atlas.

With an Atlas-connected axes, to determine the nature of an instruction error, in addition to the **GetInstructionError** command, the host should send a **GetDriveFaultStatus** command to determine if specific SPI checksum errors have occurred. Finally, a **GetInstructionError** command should be sent directly to the Atlas amplifier to determine if it has recorded an instruction error. See Section 15.15, "Atlas-Specific Commands" for information on how the host can send commands to a specific Magellan-connected Atlas.

All Instruction errors, whether indicating a problem between the host and the Magellan, or a problem between the Magellan and attached Atlas amplifiers, may indicate a serious control problem. It is the responsibility of the user to assess and correct any such problems.

7.3 Activity Status Register

Like the Event Status register, the Activity Status register tracks various motion control IC fields.

Activity Status register bits are not latched, however. They are continuously set and reset by the motion control IC to indicate the status of the corresponding conditions.

The Activity Status register is defined in the following table.

Bit	Name	Description
0	Phasing initialized	Set 1 when the motor's commutation hardware has been initialized. Cleared 0 if not yet initialized. Valid only for the MC58000 series motion control ICs.
1	At maximum velocity	Set 1 when the commanded velocity is equal to the maximum velocity specified by the host. Cleared 0 if it is not. This bit functions only when the profile mode is trapezoidal, velocity contouring, or S-curve. It will not function when the motion control IC is in electronic gearing mode.
2	Position tracking	Set 1 when the servo is keeping the axis within the Tracking Window. Cleared 0 when it is not. See Section 7.2, "Event Status Register," for more information.
3–6	Current profile mode	These bits indicate the profile mode currently in effect, which might be different from the value set using the **SetProfileMode** command if an **Update** command has not yet been issued. These three bits define the profile mode as follows:
		bit 6 · bit 5 · bit 4 · bit 3 · Profile Mode
		0 · 0 · 0 · 0 · trapezoidal
		0 · 0 · 0 · 1 · velocity contouring
		0 · 0 · 1 · 0 · S-curve
		0 · 0 · 1 · 1 · electronic gear
7	Axis settled	Set 1 when the axis has remained within the settle window for a specified period of time. Cleared 0 if it has not. See Section 7.5, "Signal Status Register," for more information.
8	Position Loop Enabled	Set 1 when either the position loop or trajectory generator is enabled. Cleared 0 when both the trajectory generator and position loop are disabled. The **SetOperatingMode** command is normally used to select what modules are enabled or disabled; however, modules can be automatically disabled by event actions (**SetEventAction** command) or breakpoints.
9	Position capture	Set 1 when a new position value is available to read from the high speed capture hardware. Cleared 0 when a new value has not yet been captured. While this bit is set, no new values will be captured. The command **GetCaptureValue** retrieves a captured position value and clears this bit, thus allowing additional captures to occur.
10	In-motion indicator	Set 1 when the trajectory profile commanded position is changing. Cleared 0 when the commanded position is not changing. The value of this bit may or may not correspond to the value of the motion complete bit of the Event Status register, depending on whether the motion complete mode has been set to commanded or actual.
11	In positive limit	Set 1 when the motor is in a positive limit condition. Cleared 0 when it is not.

Bit	Name	Description
12	In negative limit	Set 1 when the motor is in a negative limit condition. Cleared 0 when it is not.
13–15	Profile segment	Indicates the S-curve segment number, 1–7. See Section 4.3, "S-curve Point-to-Point Profile," for more information. A value of 0 in this field indicates the trajectory is not in motion. This field is undefined for other profile modes and may contain 0s or 1s.

The command **GetActivityStatus** returns the contents of the Activity Status register for the specified axis.

7.4 Drive Status Register (ION, MC58113-Series, and Atlas-Connected Axes Only)

The Drive Status register functions similarly to the Activity Status register in that it continuously tracks various motion control IC fields. In other words, Drive Status register bits are not latched; they are continuously set and reset by the motion control IC to indicate the status of the corresponding conditions. The specific status bits provided by the Drive Status register are defined in the following table.

Bit	Name	Description
0	*Reserved*	*May contain 0 or 1.*
1	In foldback	Set 1 when in foldback, cleared 0 if not in foldback. **Note:** Depending on the application, when this condition occurs it may be necessary to power down the system and check for proper operation or service.
2	Overtemperature	Set 1 when the axis is currently in an overtemperature condition. Cleared 0 if the axis is currently not in an overtemperature condition. **Note:** Depending on the application, when this condition occurs it may be necessary to power down the system and check for proper operation or service.
3	*Reserved*	*May contain 0 or 1.*
4	In holding	Set 1 when the axis is in a holding current condition, cleared 0 if not.
5	Overvoltage	Set 1 when the axis is currently in an overvoltage condition. Cleared 0 if the axis is currently not in an overvoltage condition. **Note:** Depending on the application, when this condition occurs it may be necessary to power down the system and check for proper operation or service.
6	Undervoltage	Set 1 when the axis is currently in an undervoltage condition. Cleared 0 if the axis is currently not in an undervoltage condition. **Note:** Depending on the application, when this condition occurs it may be necessary to power down the system and check for proper operation or service.
7	Disabled	Set 1 when the attached amplifier's Enable signal is not active. Cleared 0 when the amplifier Enable signal is active. (Atlas-connected axes only).
8-11	*Reserved*	
12	Output Clipped	Set 1 when the amplifier current command can not be met because of output clipping. This information is used by Magellan to prevent PID saturation (MC58113 and Atlas-connected axes only).
13, 14	*Reserved*	
15	Drive Not Connected	Set 1 when communication to connected amplifier has not yet been established. Cleared 0 when successfully communication with the amplifier has been established. (Atlas-connected axes only).

See Chapter 15, "Drive Control," for more information on undervoltage, in foldback, overtemperature, and overvoltage. See Chapter 14, "Step Motor Control," for more information on holding current.

The command **GetDriveStatus** returns the contents of the Drive Status register for the specified axis.

7.5 Signal Status Register

The Signal Status register provides real-time signal levels for various motion control IC I/O pins. The Signal Status register is defined in the following table.

Bit	Name	Description
0	A encoder	A signal of quadrature encoder input.
1	B encoder	B signal of quadrature encoder input.
2	Index encoder	Index signal of quadrature encoder input.
3	Home/Capture	For MC50000, this bit holds the home signal input. For ION, this bit holds the home signal, the HighSpeedCapture signal, or the Index signal, depending on which was set as the high speed capture. See Section 10.2, "High-Speed Position Capture" for details on setting high speed capture.
4	Positive limit	Positive limit switch input.
5	Negative limit	Negative limit switch input.
6	AxisIn	Generic axis input signal.
7	Hall1	Hall effect sensor input number 1.
8	Hall2	Hall effect sensor input number 2.
9	Hall3	Hall effect sensor input number 3.
10	AxisOut	AxisOut output.
11–12	*Reserved*	*May contain 0 or 1.*
13	/Enable	Enable signal input (ION and MC58113 only).
14	FaultOut	Fault signal output (ION and MC58113 only).
15	*Reserved*	*May contain 0 or 1.*

The command **GetSignalStatus** returns the contents of the Signal Status register for the specified axis. All Signal Status register bits are inputs except bit 10 (*AxisOut*) and bit 14 (*FaultOut*).

The input bits in the Signal Status register represent the actual hardware signal level combined with the state of the signal sense mask described in the next section. That is, if the signal level at the motion control IC is high, and the corresponding signal mask bit is 0 (do not invert), then the bit read using **GetSignalStatus** will be 1. Conversely, if the signal mask for that bit is a 1 (invert), then a high signal on the pin will result in a read of 0 using the **GetSignalStatus** command.

The outputs in the Signal Status register are not affected by the signal sense mask. For these signals a 1 indicates an active condition and a 0 indicates a non active condition.

The actual interpretation of the signal is dependent on its function. For example, *Index, Home, Negative Limit,* and *Positive Limit* are interpreted as active low, meaning that if a 0 is read using the **GetSignalStatus** command, the signal is active. Using the *Negative Limit* signal as an example, if **GetSignalStatus** indicates a value of 0, the motion control IC interprets this to mean that the axis is in the negative limit switch. Other signals such as *HallA, HallB, HallC, AxisIn,* and *AxisOut* do not have an active high or active low interpretation as such. Refer to the sections of the manual that describe these hardware functions for details on how these signals are used by the motion control IC.

7.5.1 Signal Sense Mask

The bits in the Signal Status register represent the high/low state of various signal pins on the motion control IC. It is possible to invert the incoming signal under software to match the signal interpretation of the user's hardware. This function is accessed via the command **SetSignalSense**, and can be read back using the command **GetSignalSense**.

The default value of the signal sense mask is "not inverted" except for the ***SPIEnable*** signal when the ***OutputMode0*** pin is grounded. See the *Magellan MC58000 Electrical Specifications* for more information on ***OutputMode0*** pin functions.

Bit	Name	Interpretation
0	A encoder	Set 1 to invert quadrature A input signal. Clear 0 for no inversion.
1	B encoder	Set 1 to invert quadrature B input signal. Clear 0 for no inversion.
2	Index encoder	Set 1 to invert, clear 0 for no inversion. This means that for active low interpretation of index signal, set to 0; and for active high interpretation, set to 1.
3	Home/Capture	Set 1 to invert, clear 0 for no inversion. This means that for active low interpretation of Home/Capture signal, set to 0; and for active high interpretation, set to 1.
4	Positive limit	Set 1 to invert, clear 0 for no inversion. This means that for active low interpretation of positive limit switch, set to 0; and for active high interpretation, set to 1.
5	Negative limit	Set 1 to invert, clear 0 for no inversion. This means that for active low interpretation of negative limit switch, set to 0; and for active high interpretation, set to 1.
6	AxisIn	Set 1 to invert AxisIn signal. Clear 0 for no inversion.
7	Hall A	Set 1 to invert HallA signal. Clear 0 for no inversion.
8	Hall B	Set 1 to invert HallB signal. Clear 0 for no inversion.
9	Hall C	Set 1 to invert HallC signal. Clear 0 for no inversion.
10	AxisOut	Set 1 to invert AxisOut signal. Clear 0 for no inversion.
11	Step, SPIEnable	Set 1 to define active transition as low-to-high. Clear 0 to define active transition as high-to-low (MC50000 only) of the *Pulse* and *SPIEnable* output signals.
12	Motor Direction	Set 1 to invert Motor Direction. Clear 0 for no inversion (MC50000 only).
13–15	*Reserved*	

When the capture source is set to Index with the SetCaptureSource command, the Index signal sense (bit 2) should be used to control the polarity of the index.

8. Motion Monitoring and Related Processing

In This Chapter

- SetEventAction Processing
- Motion Error
- Travel-Limit Switches
- Tracking Window
- Motion Complete Indicator
- In-Motion Indicator
- Settle Window
- Trace Capture
- Trace Buffer Architecture
- Host Interrupts

8.1 SetEventAction Processing

The Magellan Motion Control ICs provide a programmable mechanism for reacting to various safety or performance-related conditions.

The command **SetEventAction** is used to specify what action should be taken for a given condition. To define an event-related response, both a condition and an action must be specified. The following table lists the possible Event Status register conditions that can be used to define an event-related action.

Condition Name	Description
Motion error	A motion error occurs when the position error exceeds a programmable threshold.
Positive limit	A positive limit event occurs when the corresponding signal goes active while the motor velocity is positive.
Negative limit	A negative limit event occurs when the corresponding signal goes active while the motor velocity is negative.
Current foldback	(ION, MC58113-series, and Atlas-connected Magellan axes only). A current foldback event occurs when the amplifier current output goes into a foldback condition.

Both motion error processing and limit switch processing are described in detail later in this chapter. See Section 15.11, “Current Foldback” for more information on current foldback. In addition to these four monitored conditions it is possible to request an immediate safety action. See the *Magellan Motion Control IC Programmer's Command Reference* for more information.

The following table describes the actions that can be programmed for these conditions.

Action Name	Description
No Action	No action taken.
Smooth Stop	Causes a smooth stop to occur at the current active deceleration rate. The velocity command will be set to zero after event action occurs.
Abrupt Stop	Causes an instantaneous halt of the trajectory generator. The velocity command will be set to zero (0) after event action occurs.
Abrupt Stop with Position Error Clear	Causes an instantaneous halt of the trajectory generator as well as a zeroing of the position error (equivalent to ClearPositionError command). The velocity command will be set to zero (0) after event action occurs.
Disable Position Loop & Higher Modules	Disables trajectory generator and position loop module.
Disable Current Loop & Higher Modules	Disables trajectory generator, position loop, and current loop modules.
Disable Motor Output & Higher Modules	Disables trajectory generator, position loop, current loop, and motor output modules.

Once the event condition is programmed, the motion control IC monitors the specified condition continuously and executes the programmed action if it occurs. Upon occurrence, the programmed action is executed, and related actions may occur such as setting the appropriate bit in the Event Status register.

To recover from an event action, the command RestoreOperatingMode is used. This command will reset the motion control IC to the operating mode previously specified using SetOperatingMode command. Note that if the event condition is still present, then the event action will immediately occur again.

It is the responsibility of the user to safely and thoroughly investigate the cause of event-related events, and only restart motion operations when appropriate corrective measures have been taken.

If the event action programmed was either *No Action, Abrupt Stop*, or *Smooth Stop*, then the RestoreOperatingMode command will have no effect. It is intended to restore disabled modules only, and has no effect on trajectory generator parameters.

Once programmed, an event action will be in place until reprogrammed. The occurrence of the event condition does not reset the programmed event action.

Magellan provides default values for event-related processing. These defaults are intended to provide safe operation for many typical motion systems. Whether or not these defaults are appropriate must be determined by the user.

The default event actions are summarized in the following table.

Condition	Default Action
Motion Error	Disable position loop and trajectory generator.
Positive & Negative Limit	Abrupt Stop with Position Error Clear.
Current Foldback	Disable motor output and higher modules.

8.2 Motion Error

Under certain circumstances the actual axis position (encoder position) may differ from the commanded position (instantaneous output of the profile generator) by an excessive amount. Such an excessive position error often indicates a potentially dangerous condition such as motor failure, encoder failure, or excessive mechanical friction.

To detect this condition, as well as increasing safety and equipment longevity, the Magellan Motion Control ICs include a programmable maximum position error.

The maximum position error is set using the command **SetPositionErrorLimit**, and read using the command **GetPositionErrorLimit**. To determine whether a motion error has occurred, the position error limit is continuously compared against the actual position error. If the position error limit value is exceeded, then the axis is said to have had a motion error.

At the moment a motion error occurs, several events occur simultaneously. The motion error bit of the Event Status register is set. If the default event action for motion error has not been modified, then the trajectory generator and position loop are disabled. For servo axes this means the manually-set Motor Command register will be used as the motor output value. For step motors this means the axis will immediately stop. If a new event action for motion error has been specified, then whatever programmed action was entered will occur.

To recover from a motion error, the cause of motion error should be determined, and the problem corrected (this may require human intervention). If the event response resulted in a disabling of control modules, the host should then issue a **RestoreOperatingMode** command.

If an event action of *No Action* was programmed, then only the motion error status bit is set. In this case, no recovery sequence is required to continue operating the motion control IC. However, for safety reasons, the user may still want to manually stop motion, and determine the cause of the motion error.

Generally speaking, after a motion error the corresponding bit in the Event Status register is cleared using the command **ResetEventStatus**. For event actions of *Smooth Stop*, *Abrupt Stop*, and *Abrupt Stop with Position Error Clear*, clearing this bit is required to make a subsequent move. Although recommended, for event actions that disable modules, clearing this bit is not required to make a further move, nor is it required for an event action of *No Action*.

8.3 Travel-Limit Switches

The Magellan Motion Control ICs support motion travel limit switches which may be used to automatically recognize an end-of-travel condition. This is an important safety feature for systems with a defined range of motion.

The following figure shows a schematic representation of an axis with travel-limit switches installed, indicating the legal motion area and the over-travel, or illegal region.

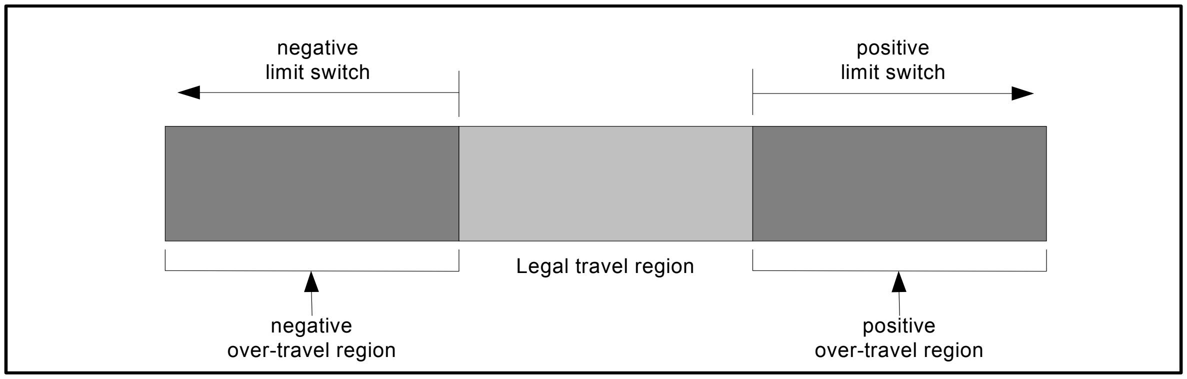


Figure 8-1: Directional Limit Switch Operation

For detailed information on interfacing to these signals, see the *Magellan Control ICElectrical Specification* (motion control IC users), or the motion card or *ION Digital Drive User's Manual* (card and module users).

At the moment a positive or negative limit switch event occurs, several events occur simultaneously. The corresponding overtravel bit of the Event Status register is set, along with a bit in the Activity Status register. If the default event action for positive and negative limits has not been modified, then the trajectory generator will undergo an abrupt stop and the position error will be cleared. If a new event action for overtravel has been defined, then whatever programmed action was entered will occur.

If an event action of *No Action* was programmed, then the limit status bit is not set, and no recovery sequence is required to continue operating the motion control IC.

To process limit switch events, the motion control IC will constantly monitor the limit switch input pins looking for a limit switch event. A limit switch event occurs when a limit switch goes active while the axis commanded position or torque is moving that motor in that limit switch's direction. If the axis is not moving, or if the trajectory generator (or torque command) is moving the motor in the opposite direction, then a limit switch event will not occur. For example, a positive limit switch will occur when the axis commanded position is moving in the positive direction, and the positive limit switch goes active. However, it will not occur if the axis commanded position is moving in the negative direction or is stationary.

The sense of the limit switch inputs (active high or active low) may be controlled using the **SetSignalSense** command.

Once an axis has entered a limit switch condition, the following steps should be taken to clear the limit switch event.

1. Unless limit switch events can occur during normal machine operation, the cause of the event should be investigated and appropriate safety corrections made.
2. The limit switch bit(s) in the Event Status register should be cleared by issuing the **ResetEventStatus** command. Unless the programmed action was set to *No Action*, no motion is possible in any direction while either of the limit switch bits in the Event Status register are set.
3. If the default event action of *Abrupt Stop with Position Error Clear* is not altered, a move should be made in the direction opposite to that which caused the limit switch event. This can be any profile move that backs the axis out of the limit. If the host attempts to move the axis further into the limit, a new limit event will occur, and an instruction error will be generated. See Section 12.2.5, "Instruction Errors" for more information on instruction errors. Note that as part of the event action associated with the abrupt stop, the Velocity register is loaded with zero (0). Thus a **SetVelocity** command must be sent along with any other desired profile parameters.
4. If the event action is altered from the default value such that a module is disabled, the host should issue a **RestoreOperatingMode** command to restore normal operation of the control loop. Then a reversing move should be made as described in step #3.

If an event action of *No Action* is defined for limit switches, then only step 1 is required.

For axes in elecronic gearing mode, the above steps are modified slightly. After the ResetEventStatus, a SetGearRatio command followed by an Update command are required. This is because the limit event sets GearRatio to zero (0).

If the limit switches are wired to separate switches, then it should not be possible for both limit switches to be active at the same time. However, if this does occur (presumably due to a special wiring arrangement), then both limit switch bits in the Activity Status register will be set, thus disabling moves in either direction. In this case, the **SetEventAction** command should be used with a value of *No Action* to temporarily disable limit switch processing while the motor is moved off of the switches.

Limit switch processing generally is used with the trajectory generator module enabled. When the trajectory generator is disabled (for example during amplifier calibration), limit switch processing still occurs but the possible event actions are limited to disabling other modules. The trajectory generator must be enabled for *Smooth Stop*, *Abrupt Stop*, or *Abrupt Stop with Position Error Clear* to be automatically executed by a limit event.

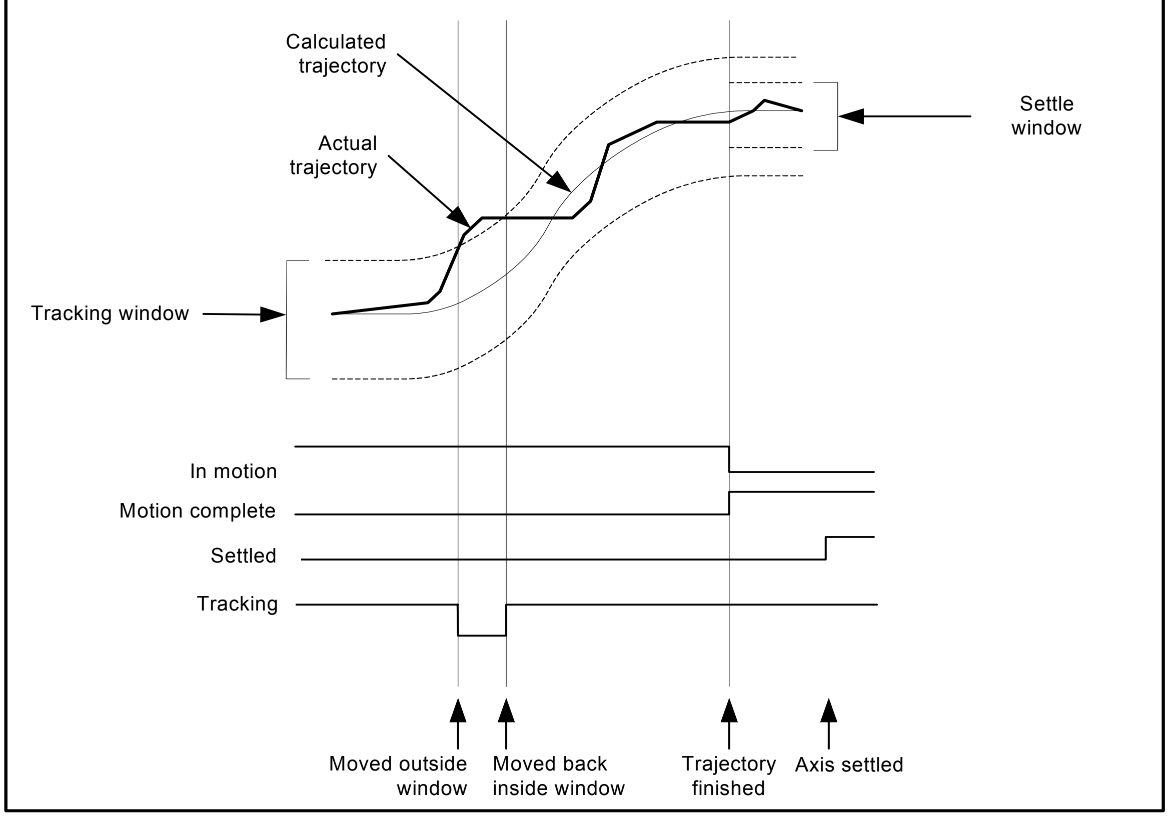
8.4 Tracking Window

The Magellan Motion Control IC provides a programmable tracking window to monitor servo performance outside the context of a motion error. The functionality of the tracking window is similar to the motion error in that there is a programmable position error limit within which the axis must remain. Unlike the motion error facility, should the axis move outside of the tracking window, the axis is not stopped. The tracking window is useful for external processes that rely on the motor's correct tracking of the desired trajectory within a specific range. The tracking window may also be used as an early warning for performance problems that do not yet qualify as a motion error.

To set the size of the tracking window (the maximum allowed position error while remaining within the tracking window), the command SetTrackingWindow is used. The command GetTrackingWindow retrieves this value.

When the position error is less than or equal to the window value, the tracking bit in the Activity Status register is set. When the position error exceeds the tracking window value, the tracking bit is cleared. See Figure 8-2 for details.

Figure 8-2: Tracking Window



8.5 Motion Complete Indicator

In many cases, it may be advantageous to have the motion control IC signal that a given motion profile is complete. This functionality is available in the motion complete indicator.

The motion complete indicator appears in bit 0 of the Event Status register. Like all bits in the Event Status register, the motion complete bit is set by the motion control IC and cleared by the host. When a motion is complete, the motion control IC sets the motion complete bit to on. The host can examine this bit by polling the Event Status register, or the host can program an automatic follow-on function using a breakpoint, a host interrupt, or an AxisOut signal. In either case, once the host has recognized that the motion has been completed, the host should clear the motion complete bit. This action will enable the bit to indicate the end of motion for the next move.

Motion complete can indicate the end of the trajectory motion in one of two ways. The first is commanded: the motion complete indicator is set based on the profile generator registers only. The other method is actual: the motion complete indicator is based on the actual encoder. The host instruction SetMotionCompleteMode determines which condition controls the indicator.

When set to commanded, the motion is considered complete when both trajectory generator registers for commanded velocity and acceleration become zero (0). This normally happens at the end of a move when the destination position has been reached. It may also happen as the result of a stop command (SetStopMode command), a change of velocity to zero (0), or when a limit switch event occurs, or after a motion error occurs.

When set to actual, the motion is considered complete when all of the following actions have occurred:

The profile generator (commanded) motion is complete.

The difference between the actual position and the commanded position is less than or equal to the value of the settle window. The settle window is set using the command SetSettleWindow. This same value may be read back using the command GetSettleWindow. See Section 8.7, "Settle Window" for more information on the settled window.

The two previous conditions have been met continuously for the last N cycles, where N is the programmed settle time. The settle time is set using the command SetSettleTime. This same value may be read back using the command GetSettleTime.

At the end of the trajectory profile, the cycle timer for the actual-based motion complete mechanism is cleared, so there will always be at least an N cycle delay (where N is the settle time) between the profile generator being completed point-to-point, and the motion complete bit being set. The motion complete bit functions in the S-curve point-to-point trapezoidal point-to-point, and velocity contouring profile modes only. It does not function when the profile mode is set to electronic gearing.

Appropriate software methods should be used with the actual motion complete mode, because it is possible that the motion complete bit will never be set if the servo is not tracking well enough to stay within the programmed position error window for the specified settle time.

8.6 In-Motion Indicator

The motion control IC can indicate whether or not the axis is moving. This function is available through the in-motion indicator.

The in-motion indicator appears in bit 10 of the Activity Status register. The in-motion bit is similar to the motion complete bit, however, there are two important differences. The first is that (like all bits in the Activity Status register) the in-motion indicator continuously indicates its status without interaction with the host. In other words, the in-motion bit cannot be set or cleared by the host. The other difference is that this bit always indicates the profile generator (commanded) state of motion, not the actual encoder.

The in-motion indicator bit functions in the S-curve point-to-point, trapezoidal point-to-point, and velocity contouring profile modes only. It does not function when the profile mode is set to electronic gearing.

It is recommended that the motion complete bit be used for determining when a motion profile has finished.

8.7 Settle Window

The motion control IC can also continuously indicate whether or not the axis has settled.

The settled indicator appears in bit 7 of the Activity Status register. The settled indicator is similar to the motion complete bit when the motion complete mode is set to actual. The differences are that the settled indicator continuously indicates its status (cannot be set or cleared).

The axis is considered to be settled when the axis is at rest (i.e., not performing a trajectory profile), and when the actual motor position has settled at the commanded position for the programmed settle time.

The settle window and settle time used with the settled indicator are the same as the motion complete bit when set to actual. Correspondingly, the same commands are used to set and read these values: **Set/GetSettleWindow** and **Set/GetSettleTime**.

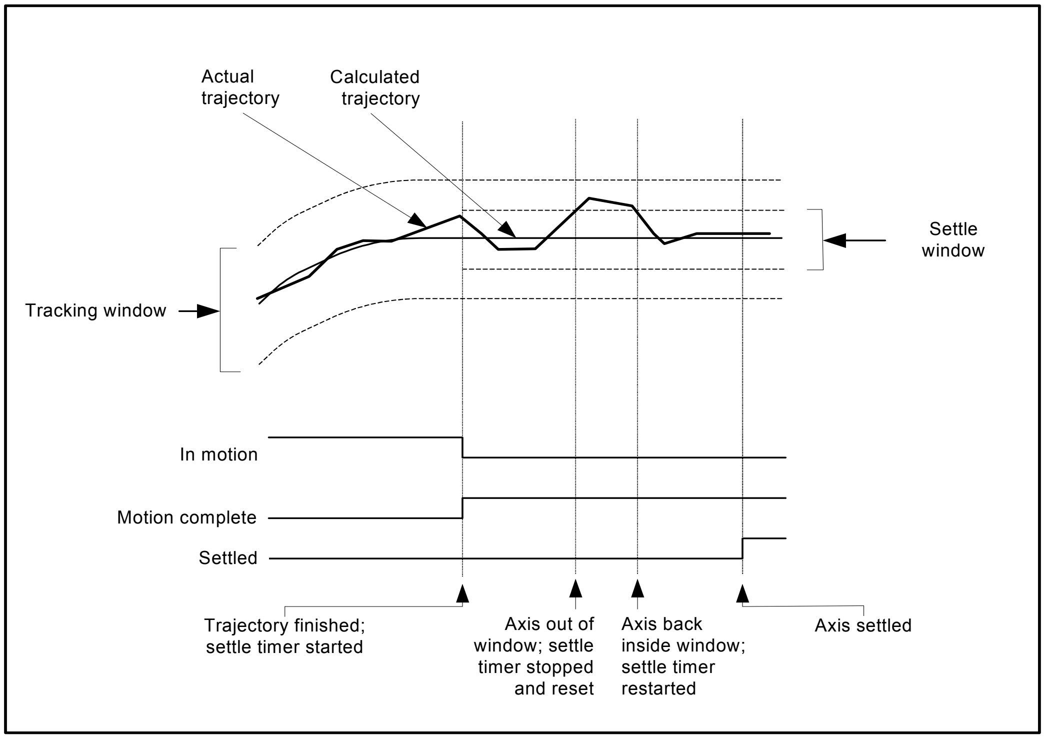

Actual trajectory
Calculated trajectory
Settle window
Tracking window
In motion
Motion complete
Settled
Trajectory finished; settle timer started
Axis out of window; settle timer stopped and reset
Axis back inside window; settle timer restarted
Axis settled

Figure 8-3: Settle Window

8.8 Trace Capture

Trace capture is a powerful feature of the Magellan Motion Control IC that allows various motion control IC parameters and registers to be continuously captured and stored to a memory buffer. The captured data may later be downloaded by the host using standard memory buffer access commands. Data traces are useful for optimizing servo performance, verifying trajectory behavior, capturing sensor data, or to assist with any type of monitoring where a precise time-based record of the system's behavior is required.

Maintaining the trace buffer requires the motion control IC to perform extra work during each cycle. In very high-performance scenarios (such as a single axis configuration running at maximum speed), the user may have to increase the chip cycle (servo loop) time. See Section 3.4, "Setting the Cycle Time" for more information on cycle time requirements. Most applications should disable trace capture once the motion profile has been debugged.

Trace data capture (by the motion control IC) and trace data retrieval (by the host) are executed as two separate processes. The host specifies which parameters will be captured, and how the trace will be executed. Then the motion control IC performs the trace. Finally, the host retrieves the data after the trace is complete. It is also possible to perform continuous data retrieval, even as the motion control IC is continuing to collect additional trace data.

8.9 Trace Buffer Architecture

The Magellan's powerful trace feature relies on the availability of a data memory pool which can be accessed by the motion control IC. For MC55000 series and MC58000 series card level designs, this memory is located externally to the motion control IC, and typically takes the form of single or dual ported static RAM. How much RAM, and what type of RAM, is up to the user that is designing the card, and generally is determined by the number of desired trace elements multiplied by the desired time duration of the trace.

For PMD's off-the-shelf motion cards using the MC55000 or MC58000 processors, the amount of external RAM is fixed, and is described in the corresponding user's guide for that product. Most of these cards provide the trace RAM in a dual port architecture, so that data can be read from the port at high speed directly through the parallel bus, even while the motion control IC is writing trace data to the buffer.

ION and MC58113-series ICs also support a trace buffer, however these buffers are located within the motion control IC itself. ION has a fixed size of 1,536 bytes (1.5 KBytes), while the MC58113 has a fixed size of 32,768 bytes (32 KBytes).

To start a trace, the host must specify a number of parameters. They are:

Parameter	Description
Trace buffer	The host must initialize the data trace buffer memory. The Magellan Motion Control IC provides special instructions to initialize external memory into buffers, allowing various sizes and start locations of external memory to be used for tracing. For ION and MC58113, buffer initialization is not required, as its size and location is fixed.
Trace variables	Depending on the motion control IC type, there are many dozens of separate items within the motion control IC that may be traced; for example, actual position, Event Status register, position error, etc. The user must specify the variables and the axes from which data will be recorded.
Trace period	The motion control IC can capture the value of the trace variables for every single motion control IC cycle, every other cycle, or at any programmed frequency. The period of data collection and storage must be specified.
Trace mode	The motion control IC can trace in one of two modes: one-time, or rolling mode. This determines how the data is stored, and whether the trace will stop automatically or whether it must be stopped by the host.

Parameter	Description
Trace start/stop conditions	To allow precise synchronization of data collection, it is possible to define the start and stop conditions for a given trace. The motion control IC monitors these specified conditions and starts or stops the trace automatically without host intervention.

8.9.1 The Trace Buffer

The Magellan Motion Control IC organizes external memory into data buffers. Each buffer is given a numerical ID. The trace buffer must always be ID 0 (zero). For MC55000 and MC58000 series processors except MC58113, before parameter traces may be used, memory buffer 0 must be programmed with a valid base address and length. For IONs and MC58113, memory buffer 0 has been pre-configured. No initialization is required, and the remaining comments in this section on the trace buffer do not apply.

The size of the trace buffer determines the maximum number of data points that can be captured. The maximum size of the trace buffer is only limited by the amount of physical memory in the system. The addressable memory space allows up to 2,048 megawords of RAM to be installed, all of which may be used to store trace information.

While trace data is being collected, it is not legal to change the trace buffer configuration. If an attempt is made to change the base address, length, or write pointer associated with buffer 0 while a trace is running, the change will be ignored and an error will be flagged. It is possible to change the read pointer and read data from the trace buffer while a trace is running. This allows the buffer to be constantly emptied while the trace runs.

8.9.2 The Trace Period

The tracing system supports a configurable period register that defines the frequency at which data is stored to the trace buffer. The tracing frequency is specified in units of motion control IC cycles, where one cycle is the time required to process all enabled axes.

The command SetTracePeriod sets the trace period, and the command GetTracePeriod retrieves it.

8.9.3 Trace Variables

When traces are running, one to four motion control IC parameters may be stored to the trace buffer for every trace period. The four trace variable registers are used to define which parameters are stored. The following commands are used to configure the trace variables.

The command SetTraceVariable selects which traceable parameter will be stored by the trace variable specified. The command GetTraceVariable retrieves this same value.

The value passed and returned by the two preceding commands specifies the variable number, axis, and type of data to be stored. The first 16-bit word specifies the variable number, and the second 16-bit word specifies the axis number and variable ID as follows:

Bits	Name	Description
0–3	TraceAxis	Selects the source axis for the parameter.
4–7	*Reserved*	Must be written as zero.
8–15	VariableID	Selects the parameter to be stored.

Many variables are available for trace on all Magellan ICs and on ION. Current or drive-related parameters are available for trace on ION and MC58113 only. Atlas-connected systems can trace current and drive-related parameters, however these traces must be executed on the Atlas amplifier rather than the Magellan. See Section 8.9.4, "Atlas Traces" for more information on this.

The supported variable IDvalues are defined in the following table.

ID	Name	Description
Trajectory Generator		
2	Commanded Position	The commanded position output from the profile generator.
3	Commanded Velocity	The commanded velocity output from the profile generator.
4	Commanded Acceleration	The commanded acceleration output from the profile generator.
Encoder		
5	Actual Position	The actual position of the motor.
6	Actual Velocity	The actual velocity (calculated using a simple low-pass filter).
9	Capture Register	The contents of the high-speed Capture register.
15	Phase Angle	The motor phase angle.
16	Phase Offset	The phase offset.
Position Loop		
1	Position Error	The difference between the actual and commanded position.
10	Position Loop Integral	The integral value used in the position loop PID filter.
11	Position Loop Derivative	The derivative value of the position loop PID filter.
57	Position Loop Integral Contribution	The contribution of the integral portion of the PID filter.
64	Biquad 1 Input	The value input to biquad 1 filter.
65	Biquad 2 Input	The value input to biquad 2 filter.
Status Registers		
12	Event Status	The Event Status register.
13	Activity Status	The Activity Status register.
14	Signal Status	The Signal Status register.
56	Drive Status	The Drive Status register.
79	Drive Fault Status register	The Drive Fault Status register (Atlas-connected axes and MC58113 only).
Commutation/Phasing		
7	Active Motor Command	The instantaneous motor command.
17	Phase A Command	The output command for Phase A.
18	Phase B Command	The output command for Phase B.
19	Phase C Command	The output command for Phase A.
29	Phase Angle Scaled	The normalized phase angle, scaled from 0 to 360 deg (0 to 0x7fff) rather than in encoder counts.
Current Loop (ION and MC58113 only)		
66	Phase A Reference	The current loop reference for phase A.
67	Phase B Reference	The current loop reference for phase B.
30	Phase A Error	The current loop Error for phase A.
35	Phase B Error	The current loop Error for phase B.
31	Phase A Actual Current	The current loop actual current for phase A.
36	Phase B Actual Current	The current loop actual current for phase B.
32	Phase A Integrator Sum	The current loop integral sum for phase A.
37	Phase B Integrator Sum	The current loop integral sum for phase B.
33	Phase A Integral Contribution	The current loop integrator contribution to PI filter for phase A.
38	Phase B Integral Contribution	The current loop integrator contribution to PI filter for phase B.
34	Phase A Current Loop Output	The current loop output for phase A.
39	Phase B Current Loop Output	The current loop output for phase B.
Field Oriented Control (ION and MC58113 only)		
40	D Reference	The FOC reference for D (direct) loop.
46	Q Reference	The FOC reference for Q (quadrature) loop.
41	D Error	The FOC D (direct) loop error.
47	Q Error	The FOC Q (quadrature) loop error.

ID	Name	Description
42	D Feedback	The FOC D (direct) feedback current.
48	Q Feedback	The FOC Q (quadrature) feedback current.
43	D Integrator Sum	The FOC integral for D (direct) loop.
49	Q Integrator Sum	The FOC integral for Q (quadrature) loop.
44	D Integral Contribution	The FOC integrator contribution for D (direct) loop.
50	Q Integral Contribution	The FOC integrator contribution for Q (quadrature) loop.
45	D Output	The FOC output for D (direct) loop.
51	Q Output	The FOC output Q (quadrature) loop.
52	FOC Alpha Output	The FOC output for phase Alpha.
53	FOC Beta Output	The FOC output for phase Beta.
31	Phase A Actual Current	The FOC actual current for phase A (same ID # as in current loop).
36	Phase B Actual Current	The FOC actual current for phase B (same ID # as in current loop).
Drive Related (ION and MC58113 only)		
75	PWM output A	Phase A PWM output (MC58113 only)
76	PWM output B	Phase B PWM output (used with Brushless DC motors only, MC58113 only)
77	PWM output C	Phase C PWM output (used with Brushless DC and step motors only, MC58113 only)
68	i^2t foldback energy	The i^2t total foldback energy
54	Bus Voltage	The bus voltage.
55	Temperature	The temperature of the drive's output stage.
86	Bus current supply	The DC bus supply current (MC58113 only)
87	Bus current return	The DC bus return current (MC58113 only)
69	Leg current A	Current1A reading (MC58113 only)
70	Leg current B	Current1B reading (MC58113 only)
71	Leg current C	Current1C reading (MC58113 only)
72	Leg current D	Current1D reading (MC58113 only)
Analog inputs, (MC5x000 ICs except MC58113)		
20	Analog0 input	The most recently read value from the Analog0 signal.
21	Analog1 input	The most recently read value from the Analog1signal.
22	Analog2 input	The most recently read value from the Analog2 signal.
23	Analog3 input	The most recently read value from the Analog3 signal.
24	Analog4 input	The most recently read value from the Analog4 signal.
25	Analog5 input	The most recently read value from the Analog5 signal.
26	Analog6 input	The most recently read value from the Analog6 signal.
27	Analog7 input	The most recently read value from the Analog7 signal.
Analog inputs (MC58113 only)		
20	Analog1 input	The most recently read value from the Analog1 signal.
Miscellaneous		
0	None	No trace variable selected.
8	Motion Control IC Time	The motion control IC time (units of servo cycles).

Setting a trace variable's parameter to zero will disable that variable and all subsequent variables. Therefore, if N parameters are to be saved each trace period, trace variables 0 to (N–1) must be used to identify the parameters to be saved, and trace variable N must be set to zero. Note that N ≤ 4.

For example, assume that the actual and commanded position values are to be stored for axis three for each cycle period. The following commands would be used to configure the trace variables.

```
SetTraceVariable 0, 0202h     // Sets trace variable 0 to store parameter 2 (commanded
                              // position) for axis 3.
SetTraceVariable 1, 0502h     // Sets trace variable 1 to store parameter 5 (actual
                              // position) for axis 3.
```

```
SetTraceVariable 2, 0002h          // Disables trace variables 2 (and higher).
```

8.9.4 Atlas Traces

Atlas amplifiers have a dedicated trace facility located inside the Atlas drive to record trace data. Traceable data includes current control-related variables, drive-related variables such as voltage and temperature, as well as many other variables.

Atlas can trace data at a user-specified rate, or at a rate synchronous with the Magellan capture rate. Atlas has the ability to start and stop traces based on specific user-specified conditions, or in synchrony with Magellan trace start and stop conditions.

Many users will utilize PMD's Windows-based software Pro-Motion to seamless specify, perform, and analyze traces on Magellan as well as Atlas. Users interested in programming these capabilities into their own system can learn how this is accomplished via the *Atlas Digital Amplifier Complete Technical Reference* as well as the *Magellan Motion Control IC Programmer's Command Reference.*

8.9.5 Trace Modes

As trace data is collected, it is written to sequential locations in the trace buffer. When the end of the buffer is reached, the trace mechanism will behave in one of two ways, depending on the selected mode.

If one-time mode is selected, then the trace mechanism will stop collecting data when the buffer is full.

If rolling-buffer is selected, then the trace mechanism will wrap around to the beginning of the trace buffer and continue storing data. Data from previous cycles will be overwritten by data from subsequent cycles. In this mode, the diagnostic trace will not end until the conditions specified in a **SetTraceStop** command are met.

Use the command **SetTraceMode** to select the trace mode. The command **GetTraceMode** retrieves the trace mode.

8.9.6 Trace Start/Stop Conditions

The command **SetTraceStart** is used to specify the conditions that will cause the trace mechanism to start collecting data. A similar command (**SetTraceStop**) is used to define the condition that will cause the trace mechanism to stop collecting data. Both **SetTraceStart** and **SetTraceStop** require a 16-bit word of data, which contains four encoded parameters. This is detailed in the following tables.

Bits	Name	Description
0–3	Trigger Axis	For trigger types other than *Immediate*, this field determines which axis will be used as the source for the trigger. Use 0 for axis 1, 1 for axis 2, etc.
4–7	Condition	Defines the type of trigger to be used. See the following table for a complete list of trigger types.
8–11	Trigger bit	For trigger types based on a status register, this field determines which bit (0–15) of the status register will be monitored.
12–15	Trigger State	For trigger types based on a status register, this field determines which state (0 or 1) of the specified bit will cause a trigger.

The Trigger Type field must contain one of the following values.

ID	Name	Description
0	Immediate	This trigger type indicates that the trace starts (stops) immediately when the SetTraceStart (SetTraceStop) command is issued. If this trigger type is specified, the trigger axis, bit number, and bit state value are not used.
1	Update	The trace will start (stop) on the next update of the specified trigger axis. This trigger type does not use the bit number or bit state values.

ID	Name	Description
2	Event Status	The specified bit in the Event Status register will be constantly monitored. When that bit enters the defined state (0 or 1), then the trace will start (stop).
3	Activity Status	The specified bit in the Activity Status register will be constantly monitored. When that bit enters the defined state (0 or 1), then the trace will start (stop).
4	Signal Status	The specified bit in the Signal Status register will be constantly monitored. When that bit enters the defined state (0 or 1), then the trace will start (stop).
5	Drive Status	The specified bit in the Drive Status register will be constantly monitored. When that bit enters the defined state (0 or 1), then the trace will start (stop).

8.9.7 Downloading Trace Data

Once a trace has executed and the trace buffer is full (or partially full) of data, the captured data may be downloaded by the host using the standard commands to read from the external buffer memory. See Section 16.1.2, "Memory Buffer Commands" for a complete description of external memory buffer commands.

At any time, the command **GetTraceCount** may be used to get the number of 32-bit words of data stored in the trace buffer. This value may be used to determine the number of **ReadBuffer** commands that must be issued to download the entire contents of the trace buffer.

During each trace period, each of the trace variables is used in turn to store a 32-bit value to the trace buffer. Therefore, when data is read from the buffer, the first value read would be the value corresponding to trace variable 1, the second value will correspond to trace variable 2, up to the number of trace variables used.

Both the length of the trace buffer and the number of trace variables set directly affect the number of trace samples that may be stored. For example, if the trace buffer is set to 1000 words (each 32-bits), and two trace variables are initialized (variables 0 and 1), then up to 500 trace samples will be stored. However, if three trace variables are used, then 333 full trace samples may be stored. In this case, the remaining word of data will store the first variable from the 334th sample.

If the trace mode is set to **RollingBuffer**, then the 334th trace sample will store the first word in the last location of the trace buffer, and the second and third words will be stored in locations 0 and 1, respectively. In this case, the first two words of sample 333 have been overwritten by the last two words of sample 334. When the trace is stopped, the read pointer will point to the oldest word of data in the buffer. This word may not correspond to the first word of a trace sample.

Therefore, it is recommended that the length of the trace buffer be set so that it is an even multiple of the number of trace variables being used. This will ensure that the read index is pointing to the first word in a complete trace sample; whether or not the trace buffer wraps. The simplest solution is to verify that the trace buffer length is an even multiple of 12 (since 12 is evenly divisible by all possible numbers of trace variables: 1, 2, 3, or 4).

8.9.8 Running Traces

The following is a summary of data trace operations to assist in getting started.

1 Specify the data to be stored. The command **SetTraceVariable** is used to specify up to four variables to be stored for each trace period. The location of trace variables must be used contiguously. For example, to trace two variables, use trace variables 1 and 2. To trace three variables, use trace variables 1, 2, and 3. The first trace variable found that is set to **none** (refer to **SetTraceVariable** in the *Magellan Motion Control IC Programmer's Command Reference*) is assumed to be the last variable being traced. If trace variable 1 is set to **none** and trace variable 2 is set to **actual position**, then no variables will be traced since the first variable (set to **none**) specifies the end of variables being traced. If four variables are being traced, do not set any variables to **none** since all variable locations are being used.

2 (MC50000 only) Set up the trace buffer. Using the commands **SetBufferStart** and **SetBufferLength**, define the location in external RAM where trace data should be stored, and the amount of RAM to be used to hold the trace data. The trace data will be stored in the buffer with an ID of zero. Be careful not to extend the buffer beyond the amount of available physical RAM. Keep in mind that **SetBufferStart** and **SetBufferLength** specify values based on a 32-bit word size.

3 Set the trace period. The command **SetTracePeriod** sets the interval between trace samples in units of cycles. The minimum is one cycle.

4 Set the trace mode. If the trace is to be started on a specific event, then the *One Time* mode should probably be used. This will allow one buffer full of trace data to be stored, beginning with the starting event (set using **SetTraceStart**). Alternatively, if the trace is to stop on an event (as specified using **SetTraceStop**), then the rolling buffer mode should be used. This will cause the system to constantly record data until the stopping event occurs. At that point, the data leading up to the event will be saved in the trace buffer.

5 Set the stopping mode (if desired). If a specific event will cause the trace to stop, then it should be programmed using the **SetTraceStop** command. However, if the trace is to be programmed to stop when the buffer fills up (by setting the trace mode to *One Time*), then it is not necessary to set another stopping event. Also, at any time while the trace is running, the **SetTraceStop** command may be issued to immediately stop the trace.

6 Start the trace. The **SetTraceStart** command may be used to start the trace directly (by specifying the *Immediate* trigger type). Alternatively, a triggering event may be specified to start the trace when this command occurs.

8.10 Host Interrupts

Interrupts allow the host to become aware of a special motion control IC condition without the need for continuous monitoring or polling of the status registers. The Magellan Motion Control ICs provide this service in the form of a *host interrupt*.

The actual signal generated to indicate an interrupt condition depends on the communication mode being used. For parallel and SPI host communications, a physical signal on the motion control IC or motion card indicates the event. For more information on this ***HostIntrpt*** signal, see the *Magellan Control ICElectrical Specifications* (if using the motion control IC directly) or the appropriate *Magellan Motion Controller Card User's Guide* (if using an off-the-shelf motion card). For CANbus communications, host interrupt events cause a CAN message to be sent. See Section 12.4, “Controller Area Network (CAN) Communications” for more information on CANbus communications. If serial communication is being used, then host interrupts are not sent out, which generally means that the host interrupt mechanism will not be used for processors running in that communication mode.

The events that trigger a host interrupt are the same as those that set the assigned bits of the Event Status register. Those events are outlined in the following table.

Bit	Event	Occurs when
0	Motion complete	The profile has reached its endpoint, or motion is otherwise stopped.
1	Position wraparound	The axis position has wrapped.
2	Breakpoint 1	Breakpoint 1 condition has been satisfied.
3	Capture received	Encoder index pulse or home pulse has been captured.
4	Motion error	Maximum position error set for a particular axis has been exceeded.
5	Positive limit	Positive over-travel limit switch violation has occurred.
6	Negative limit	Negative over-travel limit switch violation has occurred.
7	Instruction error	Instruction causes an error.
8	Disable	User enable signal has been brought inactive (ION, MC58113 only).
9	Overtemperature fault	Over temperature condition has occurred in drive (ION, MC58113, and Atlas-connected axes only).
10	Drive Exception	Set when one of a number of drive exceptions, such as bus overvoltage or undervoltage fault occurs (ION, MC58113, and Atlas-connected axes only).
11	Commutation error	Index pulse does not match actual phase (MC58000, ION only).
12	Current foldback	Current foldback has occurred (ION, MC58113, and Atlas-connected axes only).
14	Breakpoint 2	Breakpoint 2 condition has been satisfied.

Using a 16-bit mask, the host may condition any or all of these bits to cause an interrupt. This mask is set using the command **SetInterruptMask**. The value of the mask may be retrieved using the command **GetInterruptMask**. The mask bit positions correspond to the bit positions of the Event Status register. If a 1 is stored in the mask, then a 1 in the corresponding bit of the Event Status register will cause an interrupt to occur. Each axis supports its own interrupt mask, allowing the interrupting conditions to be different for each axis.

The motion control IC continually and simultaneously scans the Event status register and interrupt mask to determine if an interrupt has occurred. When an interrupt occurs, the *HostIntrpt* signal is made active.

At this point, the host can respond to the interrupt (although the execution of the current host instruction, including the transfer of all associated data packets, should be completed), but it is not required to do so.

Since it is possible for more then one axis to be configured to generate interrupts at the same time, the motion control IC provides the command **GetInterruptAxis**. This command returns a bitmasked value with one bit set for each axis currently generating an interrupt. Bit 0 will be set if axis 1 is interrupting, bit 1 is set for axis 2, etc. If no interrupt is currently pending, then no bits will be set.

To process the interrupt, normal motion control IC commands are used. The specific commands sent by the host to process the interrupt depend on the nature of the interrupting condition. At minimum, the interrupting bit in the Event Status should be cleared using the **ResetEventStatus** command. If this is not done, then the same interrupt will immediately occur once interrupts are re-enabled.

Once the host has completed processing the interrupt, it should send a **ClearInterrupt** command to clear the interrupt line, and re-enable interrupt processing. Note that if another interrupt is pending, the interrupt line will only be cleared momentarily and then reasserted.

When the motion control IC is communicating with the host via the CAN interface, motion control IC events that generate interrupts via the parallel interface are reported as CAN messages. See Section 12.4, “Controller Area Network (CAN) Communications” for further information.

The following provides a typical sequence of interrupts and host responses. In this example, an axis has hit a limit switch in the positive direction, causing a limit switch event and an abrupt stop. The abrupt stop causes a motion error. Assume that these events all occur more or less simultaneously. In this example, the interrupt mask for this axis has been set so that either motion errors or limit switch trips will cause an interrupt.

Event	Host action
Motion error & limit switch trip generates interrupt.	Sends GetInterruptAxis instruction. Note that for ION, this command is not necessary since there is only one axis.
A bitmask identifying all interrupting axes is returned by the motion control IC. This value identifies one axis as generating the interrupt.	Sends GetEventStatus instruction, detects motion error, and limit switch flags are set. Issues a ResetEventStatus command to clear both bits. Returns the axis to closed loop mode by issuing a RestoreOperatingMode command. Issues a ClearInterrupt command to reset the interrupt signal.
Motion Control IC clears motion error bit and disables host interrupt line.	Generates a negative direction move to clear the limit switch.
Motor moves off limit switch. Activity status limit bit is cleared.	None

At the end of this sequence, all status bits are cleared, the interrupt line is inactive, and no interrupts are pending.

This page intentionally left blank.

9. Hardware Signals

In This Chapter

- The AxisOut Pin
- The AxisIn Pin
- Analog Input
- The Synch Pin–Multiple Chip Synchronization

There are a number of signals that may be used to coordinate motion control IC activity with events outside the motion control IC, card, or module. In this section, these signals will be discussed.

9.1 The AxisOut Pin

Each axis has a general purpose axis output pin which can be programmed to track the state of any of the assigned bits in the Event Status, Activity Status, Drive Status, or Signal Status registers. One or more bits can be output using a selection mask, and their senses can be programmed using a sense mask. The command SetAxisOutMask is used to program these conditions, and the command GetAxisOutMask retrieves the programmed values.

The tracked bit(s) in one of these registers may be in the same axis or in a different axis as the axis of the *AxisOut* signal itself. This function is useful for triggering external peripherals by outputting hardware signals. For example, it allows the *AxisOut* signal to be driven when conditions such as "Drive *AxisOut* when motion complete bit is low, or when commutation error bit is high" occur. Whether *AxisOut* drives the signal with a high or low signal can be controlled using the command SetSignalSense.

It is possible to use the AxisOut pin as a software-programmed direct output bit under direct host control. This may be done by selecting zero as the register in the SetAxisOutMask command, and by adjusting the level of the resulting inactive output state to either high or low by using the SetSignalSense command.

For detailed information on interfacing to this signal, see the *Magellan Motion Control IC Electrical Specifications* (for motion control IC users), the *Magellan Motion Controller Card User's Guide* (for card users), or the *ION Digital Drive User's Manual* (for module users).

9.2 The AxisIn Pin

Each axis has an input pin (*AxisIn* signal) that can be used as a general purpose input. This is read using the GetSignalStatus command. It can also be used to trigger automatic events such as performing a motion change (stop, start, change of velocity, etc.) upon a signal transition using breakpoints.

For detailed information on interfacing to this signal, see the *Magellan Motion Control IC Electrical Specifications* (for motion control IC users), the *Magellan Motion Controller Card User's Guide* (for card users), or the *ION Digital Drive User's Manual* (for module users).

No special commands are required to set up or enable the *AxisIn* signals.

9.3 Analog Input

The Magellan Motion Control ICs provide general-purpose analog inputs. The MC58113 provides a single analog input, while all other MC50000 motion control ICs provide eight inputs. These inputs are connected to internal circuitry that converts the analog signal to a digital word with a precision of 12 bits for the MC58113 and 10 bits for all other Magellan motion control ICs.

To read the most recent value converted by any of the analog channels, the command ReadAnalog is used. The value returned by this command is the result of shifting the converted value to the left into the highest bit position. These analog inputs are general-purpose and are not used by the motion control IC in any calculations.

For detailed information on interfacing to these signals, see the *Magellan Motion Control IC Electrical Specifications* (for motion control IC users), or *Magellan Motion Controller Card User's Guide* (for card users).

9.4 The Synch Pin—Multiple Chip Synchronization

(MC58000 only)

The Magellan Motion Control ICs contain support for synchronizing their internal cycle time across multiple motion control ICs. This allows the start/stop and modification of motion profiles to be synchronized across more than one motion control IC when precise timing of movement is required. This may be necessary because the motion control ICs are remotely located, or the application may require greater than four axes of synchronized motion. In the most common configuration, one motion control IC is assigned as a master node, and the other motion control ICs are set into slave mode.

Multi-chip synchronization is not available if any axis is configured for a Pulse & Direction motor type. Therefore synchronization is not available with the MC55000 motion control IC.

Other configurations, however, are also possible, including using an external hardware device to generate timing signals for all motion control ICs that are set to slave mode. Alternatively, it is possible to use the signals from the master motion control IC to synchronize external user hardware.

When an axis is run in slave mode, the cycle time is explicitly controlled by the master, as are external parallel-word position encoder reads. Communications and PWM generation are not affected, however.

The input/output state of the *Synch* pin and its function are set using the command SetSynchronizationMode. The modes are summarized in the following table.

Upon sending the SetSynchronizationMode command with a mode setting of master or slave the motion control IC's Time Register is set to 0. This is useful to ensure that all synch-connected Magellans have the same value for the time register.

Encoding	Mode	Description
0	Disabled	In the disabled mode, the pin is configured as an input and is not used. This is the default mode after reset or power-up.
1	Master	In the master mode, the pin outputs a synchronization pulse that may be used by slave nodes or by other devices to synchronize with the internal chip cycle of the master node.
2	Slave	In the slave mode, the pin is configured as an input, and a pulse on the pin synchronizes the internal chip cycle.

For detailed information on interfacing to these signals, see the *Magellan Motion Control IC Electrical Specifications* (motion control IC users) or *Magellan Motion Controller Card User's Guide* (card users).

When synchronizing multiple motion control ICs, the following rules must be observed:

- All motion control ICs must have their sample time set to the same value. For example, if an MC58420 and MC58220 are to be synchronized, the sample time must be at least equal to the greater of the two sample times. See Section 3.4, "Setting the Cycle Time," for details on the motion control IC cycle time.
- Only one node in the network may be a master. For example, with three motion control ICs in a network, one motion control IC must be designated the master and the other two must be slaves.
- Slave nodes must be set into slave mode before the master is set. This ensures that the slave(s) cycle starts at precisely the moment that the master assumes its state as master.

Example:

A typical startup sequence to ensure that multiple Magellans are synchronized and that their time registors are all equal is to set all of the synch-connected Magellans to slave mode and then to set one of them as the master.

This page intentionally left blank.

10. Encoder Interfacing

In This Chapter

- Incremental Encoder Input
- High-Speed Position Capture
- Parallel-Word Position Input

All Magellan Motion Control ICs provide incremental quadrature encoder input, while the MC50000 processors (except the MC58113) also provide parallel-word feedback, which allows devices such A/D (Analog to Digital) converters, R/D (resolver to digital) converters, parallel encoders, and other devices to be used as the motor position source. In addition, ION/CME, ION 3000, and MC58113 ICs can select an input encoding format of pulse and direction for the auxiliary axis which is useful when operating with a motion controller that outputs pulse and direction signals. See Section 4.5 "Electronic Gear Profile," for more information on using pulse and direction as a command input.

To set the feedback type, the command SetEncoderSource is used. This value can be read using the command GetEncoderSource.

10.1 Incremental Encoder Input

Incremental encoder feedback provides two square-wave signals: A quadrature and B quadrature. There is also an optional index pulse, which indicates when the motor has made one full rotation. The A and B signals are offset from each other by 90°, as shown in Figure 10-1.

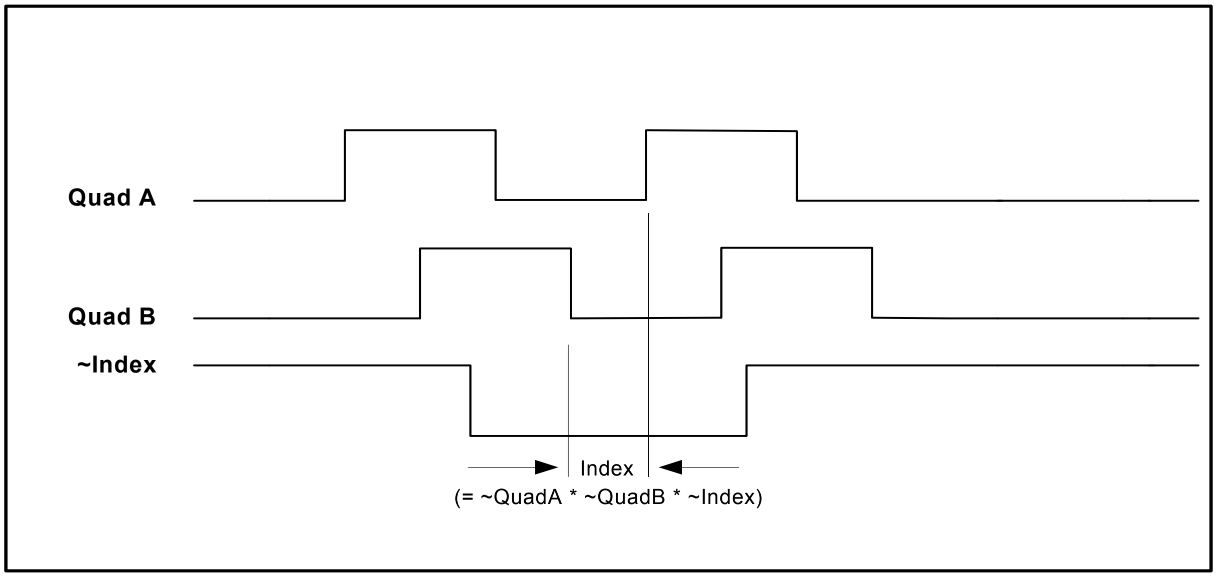


Figure 10-1: Quadrature Encoder Timing

For the quadrature incremental position to be properly registered by the motion control IC with the motor moving in the positive direction, *QuadA* should lead *QuadB*. When the motor moves in the negative direction, *QuadB* should lead *QuadA*. Due to the 90° offset, four resolved quadrature counts occur for one full phase of each A and B channel.

10.1.1 Actual Position Register

The motion control IC continually monitors the position feedback signals, and accumulates a 32-bit position value called the actual position. At power-up, the default actual position is zero. The actual position can be set with one of two commands: SetActualPosition, and AdjustActualPosition. The current actual position can be retrieved using the command GetActualPosition.

SetActualPosition sets the position to an absolute specified 32-bit value. Particularly when using incremental feedback, the actual position is generally set shortly after power-up, using a homing procedure to reference the actual position to a physical hardware location. AdjustActualPosition changes the current actual position by a signed relative value. For example a value of –25 specified using this command will subtract 25 from the current actual position.

In addition to retrieving the actual axis position, it is also possible to retrieve an estimation for the instantaneous velocity of the axis. This is accomplished using the command GetActualVelocity. Note that the provided velocity is an estimated quantity, created by subtracting the current position from the previous cycle's position. It is therefore subject to jitter or noise, particularly at low velocity.

10.1.2 Digital Filtering

All encoder inputs, as well as both the index and home capture inputs, are digitally filtered to enhance reliability. The filter requires that a valid transition be accepted only if it remains in its new (high or low) state for a specified amount of time. This ensures that brief noise pulses will not be interpreted as an encoder transition. See the Magellan electrical specifications for your motion control IC for detailed information

Although this digital filtering scheme can increase the overall reliability of the quadrature data, to achieve the highest possible reliability, additional external signal prcoessing techniques may be required. Techniques such as differential line drivers/receivers, or analog filtering have been proven in this area. Whether these additional schemes are required depends on the specific system, and the amount and type of noise sources.

10.2 High-Speed Position Capture

Each axis of the Magellan Motion Control ICs supports a high-speed Position Capture register that allows the current axis location (as determined by an attached encoder) to be captured when triggered by an external signal. For MC50000 motion control ICs, two signals may be used as the capture trigger: the Index signal or the Home signal. For ION, there is an additional available input source called *HighSpeedCapture*.

Product	Available position capture signal triggers
MC50000	Index Home
ION	Index Home HighSpeedCapture

These input triggers differ in how a capture is recognized. For all products except the MC58110, MC55110, and MC58113 when the *Index* signal is used as the trigger, a capture will be triggered when the *A*, *B*, and *Index* signals achieve a particular state (defined by the Signal Sense register using the SetSignalSense command). If the *Home* or *HighSpeedCapture* signal is selected as the capture trigger source, then only that signal need be in a particular state for the capture to be triggered. For the single chip motion control ICs (MC5x110), the *Index* operates like the *Home* or *HighSpeedCapture*.

For MC58113 the capture qualification scheme is somehwat different than for other Magellan ICs. See the *MC58113 Electrical Specifications* for details.

The command SetCaptureSource determines whether the *Index* signal, the *Home* signal, or the *HighSpeedCapture* signal will be used as the position capture trigger. The command GetCaptureSource retrieves this same value. When *Index* is selected as the trigger source, use the *Index* bit in the Signal Sense register to set the trigger state. When either *Home* or *HighSpeedCapture* are selected as the trigger source, use the Capture bit to set the trigger state.

When a capture is triggered, the contents of the Actual Position registers (derived from an attached encoder) are transferred to the Position Capture register, and the capture-received indicator (bit 3 of the Event Status register) is set. To read the Capture register, the command GetCaptureValue is used. The Capture register must be read before another capture can take place. Reading the Position Capture register causes the trigger to be re-armed, allowing for more captures to occur. As for all Event Status register bits, the Position Capture indicator may be cleared by using the command ResetEventStatus.

10.3 Parallel-Word Position Input

(MC50000 except MC58113-series only)

For feedback systems that do not provide incremental signals, but instead use a digital binary word, Magellan supports a parallel-word input mechanism that can be used with a large variety of devices including resolvers (after resolver to digital conversion), absolute optical encoders, laser interferometers with parallel word read-out, incremental encoders with external quadrature decoder circuit, and A/D converters reading an analog feedback signal.

In this position-input mode, the encoder position is read through the motion control IC's external bus by reading a 16-bit word. One word is read for each axis set to this mode. Depending on the nature of the feedback device, fewer than 16 bits of resolution may be available, in which case the unused high order data bits should be arranged to indicate a 0 value when read by the motion control IC. It is also acceptable to sign-extend these bits. Under no circumstances should unused bits of the parallel-word be left floating.

The value input by the motion control IC should be binary coded. The Magellan Motion Control ICs assume that the position data provided by the external device is a two's complemented signed number. If the value returned ranges from 0 to 2^n–1 (where n is the number of bits provided by the feedback device), then the difference in behavior will be the interpretation of the start location. This will be shifted by one-half of the full scale feedback range. If desired, this initial position may be altered using the SetActualPosition command.

When the encoder source is set to parallel, the first value read from the external bus is assumed to be an absolute position, and is directly stored in the Actual Position register.

10.3.1 Multi-Turn Systems

In addition to supporting position tracking across the full numeric feedback range of a particular device, the ability to support multi-turn systems is also provided. The parallel encoder values are continuously examined, and a position wrap condition is automatically recognized. This ranges from the largest encoder value to the smallest encoder value (negative wrap), or from smallest value to largest value (positive wrap).

Using this virtual multi-turn counter, the Magellan Motion Control ICs continuously maintain the axis location to a full 32-bit value. If the axis does not wrap around (non multi-turn system), the range will stay within a 16-bit value.

As the motor moves in the positive direction, the input value should increase to a maximum value, at which point it may wrap back to zero and continue increasing from there. Likewise, when the motor moves in the negative direction, the value should decrease to zero, at which point it may wrap back to its maximum. The value at which the parallel input device wraps is called the device's modulus, and should be set using the SetEncoderModulus command. Note that the SetEncoderModulus command takes as a parameter one-half of the value of the modulus.

For example, if a rotary motor uses a 12-bit resolver for feedback, the encoder modulus is 4,096 and therefore, the value sent to the SetEncoderModulus command would be 2,048. Once this is done, then each time the motor rotates and the binary word value jumps from the largest binary output to the smallest, the Magellan Motion Control IC will properly recognize the motor wrap condition, and accumulate actual encoder position with values larger than 4,096 or smaller than 0.

For systems using a position counter with a modulo smaller than the encoder counts per revolution, set the counts/rev value equal to the position counter size. For example, if a rotary laser interferometer is being used that provides a 16-bit output value, but provides 16,777,216 counts per revolution, use a counts/rev value of 32,768 ($2^{16}/2$).

The encoder modulus should always be set prior to setting the encoder source to parallel.

No high-speed position capture is supported in the parallel-word device input mode. Therefore the ***Index*** and ***Home*** signals, as well as the ***QuadA*** and ***QuadB*** signals are unused.

10.3.2 Parallel-Word Device Interfacing

For each axis set for parallel-word input, the motion control IC will use its peripheral bus to read the position feedback value for that axis. The motion control IC will perform a peripheral read at the corresponding address, but axes not in parallel-word mode will not be addressed. This read occurs every 50µs. See the *Magellan Control IC Electrical Specification* for details on parallel-word addresses and interfacing.

Motion cards such as Prodigy-PCI and Prodigy-PC/104 do not directly provide the ability to connect parallel-word encoder devices; however, it is possible to purchase or construct a daughter card that plugs into the main motion card and supports this interface.

11. Motor Output

In This Chapter

- Disabling the Motor Output Module
- Enabling the Motor Output Module
- Motor Type
- Motor Command Output
- Setting PWM Frequency
- Setting PWM Signal Interpretation (MC58113-Series Only)
- Multi-Phase Motor Interfacing
- Microstepping Motor Output
- DC Servo Motor Output
- Pulse & Direction Signal Generation

Magellan Motion Control ICs contain a motor output module to control how motor command signals are generated and output to the amplifier. For MC50000, these signals are output through the motion control IC and are used to interface to external circuitry. For ION, these signals remain internal to the drive. This chapter will detail how to enable and disable the motor output module, how to select a motor type, and how to select and interface to different amplifier signaling techniques.

11.1 Disabling the Motor Output Module

There are a number of reasons why it might be desirable to disable the motor output module. See Section 3.2, "Enabling and Disabling Control Modules," for an overall discussion of module enabling and disabling. In particular, the motor output module is generally disabled for safety-related condition, or for system calibration. To disable the motor output module the command SetOperatingMode is used. The value set using this command can be read using GetOperatingMode.

If the motor output module is disabled, a "zero" command will be sent to the motor, or to each phase of the motor for multi-phase motors such as brushless DC or microstepping motors. Depending on the motor output signal format, this zero command will be represented in different ways. See the subsequent sections of this chapter for more information on motor output signal representation.

Note that disabling the motor output module may or may not immediately stop the motor. Disabling this module has the effect of "free-wheeling" the motor, which means the motor may stop, coast, or even accelerate (if a constant external force exists such as on a vertical axis) depending on the load, inertia, and configuration of the axis mechanics.

For MC50000 except MC58113 or when the ***OutputMode0*** pin is tied low, the default condition of the motor output module is enabled.

For ION modules, MC58113-series ICs, and Atlas-connected axes the default condition is disabled, therefore to begin drive operations, a SetOperatingMode command must be sent to enable the motor output module.

11.2 Enabling the Motor Output Module

A previously disabled motor output module may be re-enabled in a number of ways. If the module was disabled using the SetOperatingMode command, then another SetOperatingMode command may be issued. If this module was disabled as part of an automatic event-related action (see Section 8.1, "SetEventAction Processing," for more information), then the command RestoreOperatingMode is used.

Regardless of how the module is re-enabled, at the time that the reenable operation is requested, the desired motor commands for each phase of the motor will immediately be output to the amplifier. Care should therefore be taken to re-enable this module when the axis is in a stable condition, such that no abrupt motion occurs. Along these lines, it is recommended that if additional modules are to be enabled such as the current loop, the position loop, and the trajectory generator, all modules be enabled at the same time, thereby insuring that internal current loop or position loop values (such as integrators) will be zeroed at the time of enabling.

11.3 Motor Type

Magellan Motion Control ICs provide support for a number of motor types including DC brush, brushless DC and step motors. In addition, in the case of Brushless DC and microstepping motors, both two and three phase motors are supported for some products. The following table shows this.

Product	Motor Types Supported
MC58000 ICs except MC58113	DC brush Brushless DC, 2-phase Brushless DC, 3-phase Microstepping, 2-phase Microstepping, 3-phase Step
MC55000 ICs	Step motor
MC58113 IC	DC brush Brushless DC, 3-phase Microstepping, 2-phase
ION (DC brush), MC51113 IC	DC brush
ION (brushless DC), MC53113 IC	Brushless DC, 3-phase
ION (microstepping), MC54113 IC	Microstepping, 2-phase

For products such as ION or the MC51113, MC53113, or MC54113 ICs, which support one motor type, selection of the correct motor type is not necessary since the motor type is already set to its default condition. For products such as MC58000 motion control ICs or MC58000-based cards, the motor type for each axis must be selected, and this value is used to determine a number of default operating conditions such as which control modules are enabled and the default amplifier interface format.

For Atlas-connected axes, the motor type must still be specified, and care should be taken to insure that the selected motor type matches the motor type of the Atlas connected to that axis. Selected motor types should be either DC brush, brushless DC 3-phase, or step.

During powerup, the Magellan processors except MC58113 read a motor configuration word at a specific memory address on its external bus to determine the default motor type setting for each axis. For motion control IC designs, see the *Magellan Motion Control IC Electrical Specifications* for more details. For MC58000-based cards, such as PMD's Prodigy cards, the logic used to respond to this read operation accesses on-card user-specified DIP switches, or is stored in NVRAM configuration memory. Consult the User's Guide for the PMD-based card product you are using for more information. The MC58113 allows internal NVRAM storage of startup parameters such as motor type. See the *MC58113 Electrical Specifications* for more information.

The value of the motor type read during powerup can also be set at a later time via the command **SetMotorType**. The value set can be read back using the command **GetMotorType**. If the motor type is to be set by the user in this way, the motor type should be sent first if a series of commands are to be used to configure the output method and amplifier interface format. See the *Magellan Motion Control IC Programmer's Command Reference* for more information.

11.4 Motor Command Output

In addition to selection of the motor type, to correctly interface to an amplifier or amplifier circuit, the motor command output format must be chosen. The Magellan Motion Control ICs provide a variety of methods to generate signals used to interface to the motor amplifier; however, the methods available vary with each product series. The following table shows the motor output options available for each supported motor type and motion control IC. For ION products, the motor amplifier is located internally, and the motor command output format need not be selected.

Magellan	Motor Type	Phases per Axis	Available Output formats
MC58000	DC brush	1	SPI Atlas
			Sign/magnitude PWM
			50/50 PWM
			PWM High/Low (MC58113 only)
			Parallel DAC - offset binary* (except MC58113)
			Parallel DAC - sign magnitude (except MC58113)
			SPI DAC - offset binary
			SPI DAC - two's complement
MC58000	Brushless DC	2 or 3	SPI Atlas
			Sign/magnitude PWM**
			50/50 PWM
			PWM High/Low (MC58113 only)
			Parallel DAC - offset binary* (except MC58113)
			Parallel DAC - sign magnitude (except MC58113)
MC58000	Microstepping	2 or 3	Sign/magnitude PWM**
			50/50 PWM
			PWM High/Low (MC58113 only)
			Parallel DAC - offset binary* (except MC58113)
			Parallel DAC - sign magnitude (except MC58113)
MC58000	Step	N/A	SPI Atlas
			Pulse and Direction
MC55000	Step	N/A	Pulse and Direction
ION	All (DC Servo, brushless DC, microstepping)		Not applicable. Amplifier is internal. No motor command signal output through hardware.

** Card products provide analog output +/-10V using the parallel DAC - offset binary output mode.*
*** Only when 2-phases are selected. This format is not supported for 3-phase motors.*

PWM High/Low, Sign/magnitude PWM, 50/50 PWM, parallel DAC, and SPI DAC all output a signed numerical motor command value while using different signal formats. Each of these signal formats encodes this signed numerical value, which represents the torque at which the motion control IC is commanding the motor.

SPI Atlas selects the four-signal Atlas SPI interface. This signal format encodes a complete bi-directional protocol that allows the Magellan to send torque commands as well as interact with Atlas's other control features and status registers. For more information on Atlas Amplifiers see Section 11.4.8, "SPI Atlas," as well as the *Atlas Digital Amplifier User's Manual.*

Pulse and direction is fundamentally different from the other output formats, because it makes no attempt to encode a motor torque. Instead, pulse and direction interfaces directly with step motor amplifiers which accept this format. It should be noted that pulse and direction is generally used to drive step motors, but can also be used with servo motors for amplifiers designed with this capability.

Choice of output mode generally determines the nature of the amplifier you will use. PWM output formats are most commonly used with IC-based switching circuits. Analog output formats such as parallel DAC or SPI DAC are most often used to generate a +/10V signal which is then connected to an external analog input amplifier. SPI Atlas is exclusively used to connect to PMD's Atlas Amplifiers

The command SetOutputMode controls which of the available output formats will be used. The command GetOutputMode retrieves this value.

In the next several sections we will look at each of these motor command output formats in more detail.

11.4.1 Sign/Magnitude PWM

In sign/magnitude PWM mode, two pins are used to output the motor command information for each motor phase. One pin carries the PWM magnitude, which ranges from 0 to 100%. This signal expresses the absolute magnitude of the desired motor command. A high signal on this pin means the motor coil should be driven with voltage. A second pin outputs the sign of the motor command by going high for positive sign, and low for negative.

For example, if the PWM resolution is one part per 2,048 and the motor commands that the motion control IC output for a given phase is +12,345, then the sign bit will be output as a high level, and the magnitude pin will output with a duty cycle of 2,048*12,345/32,768 = 771.56 = 772. This indicates that the magnitude signal will be high for 772 cycles, and low for the remaining 1,276 cycles. If it were desired that the output value be –12,345, then the magnitude signal would be the same, but the sign bit would be low instead of high.

Sign/magnitude PWM output is typically used with H-bridge type amplifiers. Most amplifiers of this type have separate sign and magnitude inputs, which allows the Magellan signals to be connected directly.

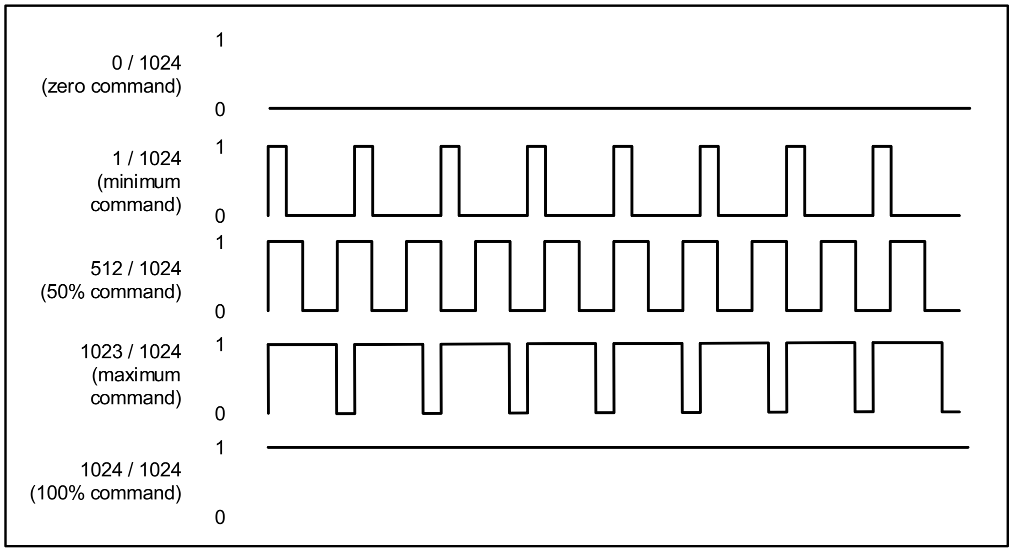


Figure 11-1: Sign/ Magnitude PWM Encoding

11.4.2 50/50 PWM

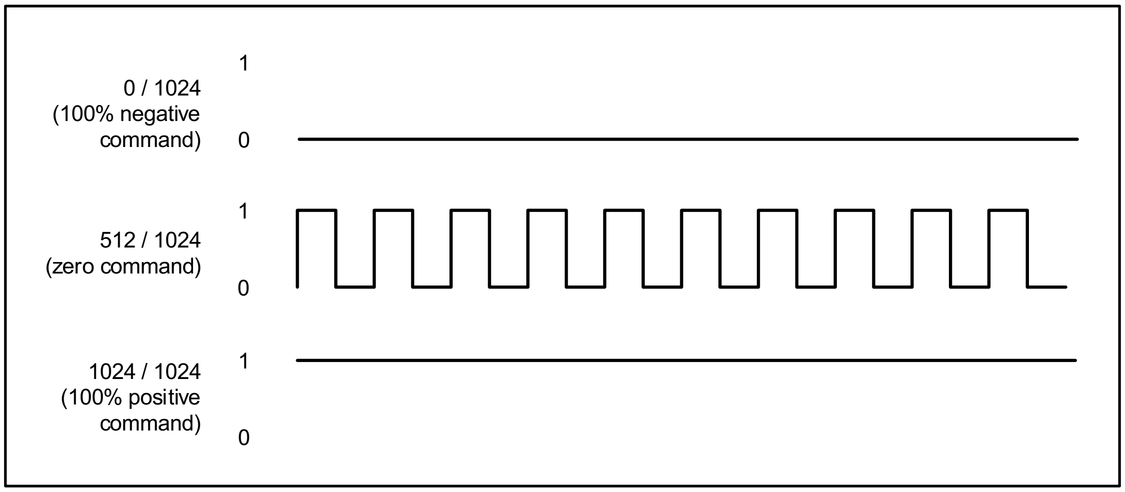
In 50/50 PWM mode, only one pin is used per motor output or per motor phase. This pin carries a variable duty cycle PWM signal, much like the magnitude signal for sign magnitude PWM. This PWM output method differs in that a 50% output signal (high half the time, low half the time) indicates a desired motor command of zero. Positive motor commands are encoded as duty cycles greater than 50% duty cycles, and negative motor commands are encoded as duty cycles less than 50%. In this mode, a full-on positive command is encoded as a 100% duty cycle (always high), and a full on negative command is encoded as a 0% duty cycle (always low).

For example, if the PWM resolution is one part per 2,048 and the motor commands that the motion control IC outputs for a given phase is +12,345, then the magnitude pin will output with a duty cycle of 1,024 + 1,024*12,345/32,768 = 771.56 = 1,409.78 = 1,410. This indicates that the magnitude signal will be high for 1,410 cycles, and low for the remaining 638 cycles. If it were desired that the output value be –12,345, then the magnitude signal would have a duty cycle of 1,024 + 1,024*–12,345/32,768 = 638.2 = 638; indicating that the magnitude signal will be high for 638 cycles, and low for the remaining 1,410.

50/50 PWM output is used with two different types of amplifiers. When driving a brushless PM (permanent magnet) motor, the magnitude signal is connected to a half-bridge driver. When driving a DC brushed motor, an H-bridge type amplifier is used. The magnitude signal of the H-bridge is always turned on, and the magnitude output of the motion control IC is connected to the sign input of the H-bridge. This alternative method of controlling an H-bridge is useful in situations where motor back-EMF during deceleration is a problem using the standard sign magnitude schemes.

Figure 11-2 illustrates 50/50 PWM output encoding.

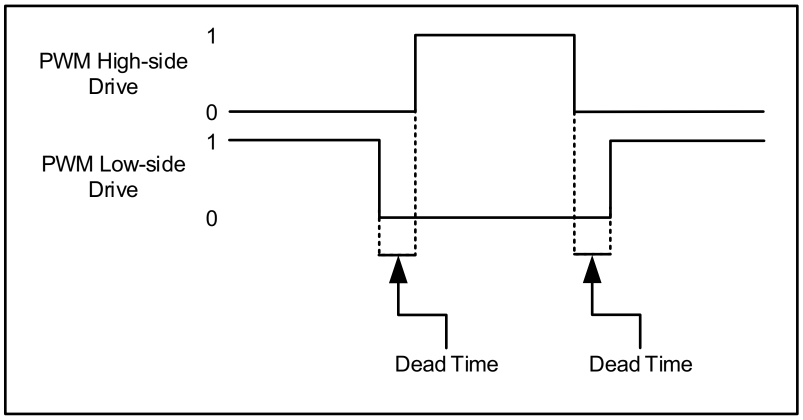
Figure 11-2: 50/50 PWM Encoding



11.4.3 PWM High/Low (MC58113 Only)

In PWM High/Low mode two pins are used per motor output or per motor phase, allowing separate high-side/low-side control of each bridge. Much like the 50/50 PWM mode, each signal carries a variable duty cycle PWM signal, and a zero desired motor command results in the high side and low side being active for the same amount of time.

Figure 11-3: PWM High/ Low Encoding



In this scheme, as Figure 11-3 shows, the high side output and the low side output are never active at the same time. In fact there is generally a period of time when neither output is active. This period of time is called the dead time, and provides a shoot through protection function for MOSFET or IGBT switches.

The dead time is specified in nSecs and can be programmed via the command **SetDrivePWM**. The correct value can generally be determined from the MOSFET or IGBT IC manufacturer's data sheet, or you can call PMD technical support if you have questions.

In addition to dead time, some high side switch drive circuitry requires a minimum amount of off time to allow the charge pump circuitry to refresh. This parameter, set using the **SetDrivePWM** command, specifies this refresh time and has units of nSecs. The related parameter of refresh time period, which is the time interval between these off time refreshes, and which is also set using the **SetDrivePWM** command, has units of current loop cycles.

> The default values for dead time, refresh time, and refresh period are set to be maximally safe but are not approriate to drive typical switching hardware. For proper amplifier function these values must be set with values appropriate for the connected switching circuitry.

PWM High/Low output mode is used with discrete component bridge designs or with integrated half bridge and H-bridge drivers ICs that provide this level of control input. The overall bridge configuration is the same as that for PWM 50/50; A triple half bridge configuration for three-phase Brushless DC, a single H-bridge for DC Brush motor drive, and two H-bridges for two-phase step motor drive.

PWM High/Low is the most commonly used mode when the MC58113's active current control is used, although PWM 50/50 can also be used with current control active.

11.4.4 Parallel DAC - Offset Binary

In parallel DAC - offset binary mode, the motor command for a given phase is output directly to the motion control IC's peripheral bus where it is assembled into an analog voltage using a DAC (Digital to Analog Converter). The motion control IC chip writes the DAC output value to each enabled channel in this mode at the commutation frequency of 10 kHz. For example, on an MC58420 with all four axes set to parallel DAC mode, DAC output will be written for each axis once per 100 µsec, with the writes in pairs separated by 50 µsec.

For one- or two-phase motors, one DAC output is used for each phase. For three-phase motors, only two DAC outputs are used. The third phase will always be an analog signal equal to –1 * (P1 + P2), where P1 is the output for phase 1, and P2 is the output for phase 2. If necessary, this third phase signal may be realized using an inverting summing amplifier in the external circuitry. Generally this is not necessary, since the majority of three-phase off-the-shelf amplifiers accept two phases and internally construct the third. The written output value has a 16-bit resolution. This value is offset by 8000h, so a value of 0 will correspond to the most negative output. A value of 8000h corresponds to zero output, and a value of FFFFh corresponds to the most positive output.

DACs with resolutions lower than 16-bit may be used. In this case, output values must be scaled to the high-order bits of the 16-bit data word. For example, to connect to an 8-bit DAC, pins **Data8–15** are used. The contents of the low-order 8 bits (**Data0–7**) are transferred to the data bus, but are ignored.

The motion control IC writes the DAC values using the external peripheral bus described in the *Magellan Motion Control IC Electrical Specifications*.

For more information on DAC signal timing and conditions, see the DAC pin descriptions and peripheral write interface timing diagram in the *Magellan Motion Control IC Electrical Specifications* document for your motion control IC.

If using a Magellan-based card, all interfacing circuitry and decoding are handled by the card, and a direct analog +/– 10V for each desired output phase will be provided through a connector. To interface to the signals, see the *Magellan Motion Controller Card User's Guide*.

11.4.5 Parallel DAC - Sign Magnitude

Parallel DAC - sign magnitude mode is very similar to parallel DAC - offset binary mode, however the encoding of the 16-bit word sent to each DAC channel has a different format. As indicated in the previous section, in offset binary mode the number output is a 16-bit two's complemented representation of the desired motor or phase command shifted by a value of 8000h. In sign magnitude mode, the encoding is different. In this format the representation is split into a 1 bit sign field and a 15 bit magnitude field.

This is encoded as follows: the top bit (bit 15) of the output DAC word represents the sign, with a 1 value indicating a negative output signal, and a 0 value indicating a positive output value. The low 15 bits (DAC outputs 0–14) encode an unsigned 15-bit magnitude of the desired motor or phase command. For example, in this scheme, the most negative possible command (–32,767) is encoded as a FFFFh, a zero output value is encoded as 0, and the most positive possible output value (+32,767) is encoded as 7FFFh.

In all other respects, this mode is identical to offset binary mode, and all other comments in the section above for that mode apply to parallel DAC - sign magnitude mode.

For Magellan cards, sign magnitude mode may not be used. Parallel DAC - offset binary mode must be selected for +/– 10V analog output.

11.4.6 SPI DAC - Offset Binary

In SPI DAC mode, single-phase drive data for DC brush motors (or brushless DC motors with external commutation) is output using the motion control IC's SPI (Serial Peripheral Interface) output port. The Magellan SPI output is a high-speed, synchronous serial I/O port that allows a serial bit stream of 16-bit data words to be shifted out of the motion control IC at a 8 Mbps rate. The **SetSPIMode** command controls the four possible output clocking schemes via the mode parameter as described in the following table.

SetSPIMode setting	Description
RisingEdge	Rising edge without phase delay: the *SPIClock* signal is active low. The SPIXmt pin transmits data on the rising edge of the *SPIClock* signal.
RisingEdgeDelay	Rising edge with phase delay: the *SPIClock* signal is active low. The SPIXmt pin transmits data one half-cycle ahead of the rising edge of the *SPIClock* signal.
FallingEdge	Falling edge without phase delay: the *SPIClock* signal is active high. The SPIXmt pin transmits data on the falling edge of the *SPIClock* signal.
FallingEdgeDelay	Falling edge with phase delay: the *SPIClock* signal is active high. The SPIXmt pin transmits data one half-cycle ahead of the falling edge of the *SPIClock* signal.

The output value is offset by 8000h, so a value of 0 will correspond to the most negative output. A value of 8000h corresponds to zero output, and a value of FFFFh corresponds to the most positive output.

Drive data for each axis is interleaved on the single SPI output pin. When the SPI data for an axis is being output, the corresponding **SPIEnable** line (**SPIEnable** for single-axis configurations, **SPIEnable1** – **SPIEnable4** for multiple axis configurations) is active; at all other times it is inactive. Note that some Magellan Motion ICs allow the ***SPIEnable*** signal sense to be user-programmed. See Section 7.5.1, "Signal Sense Mask," for details.

11.4.7 SPI DAC - Two's Complement

This mode is identical to the SPI DAC - offset binary mode described above except for the format of the output value. In this mode, the value is in standard two's complement format where zero (0) corresponds to zero output, 7FFFh corresponds to the most positive output and 8000h corresponds to the most negative output.

In all other respects, this mode is identical to offset binary mode, and all other comments in the section above for that mode apply to SPI DAC - two's complement mode.

11.4.8 SPI Atlas

This mode sets the motor output interface to SPI Atlas. The following section provides a general overview of Magellan/Atlas operations when in this mode, however most users will not need to know these details since all Magellan to Atlas operations are handled automatically by the Magellan Motion Control IC.

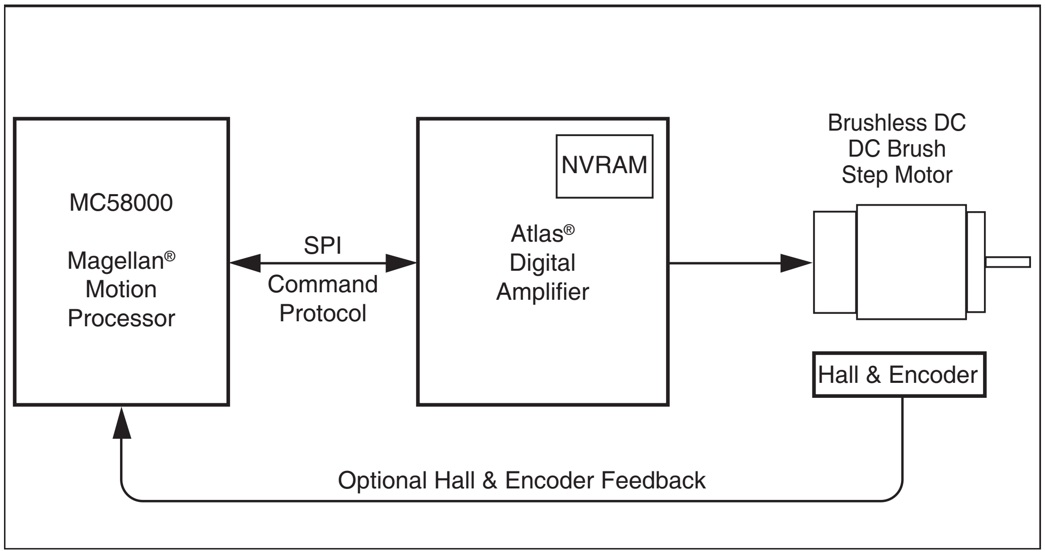


Figure 11-4: High Level Magellan To Atlas Connection Diagram

Atlas Digital Amplifiers are single-axis devices for torque or voltage-mode control of three-phase brushless DC motors, DC Brush motors, or two-phase step motors. They accept a stream of desired torque or voltage values from the Magellan and perform all current loop processing and switching bridge control to continuously drive the motor coils to the specified, commanded values.

In addition to providing a stream of torque or voltage commands, Magellan sets up operational parameters needed by Atlas such as control gains, safety-related parameters, and other information. These parameters may be provided to Atlas at each power up, or stored non-volatilely on Atlas so that they no longer need to be loaded at each power-up. See the *Atlas Digital Amplifier User's Manual* for more information.

Communication to/from Atlas occurs via an SPI interface and associated protocol that uses packet-oriented commands to specify various Atlas parameters, and, if desired, request status information from Atlas. This protocol has been designed for maximum speed and flexibility so that torque or voltage commands can be continuously sent to Atlas even while the external controller queries Atlas for various information. Please refer to the *Atlas Digital Amplifier Complete Technical Reference* for information on the SPI interface.

At power-up or reset, Atlas checks for the presence of stored configuration information in its non-volatile memory. If no such configuration information is found, default values are used.

SPI Atlas Communications Overview

Atlas uses an SPI (Serial Peripheral Interface) digital connection to communicate with Magellan. This connection is used to setup Atlas parameters, specify voltage or torque output values, monitor Atlas operation, as well as other functions.

SPI Atlas uses only 4 signals; SPIClk (Clock), SPIEnable (chip select), SPIRcv (slave in), and SPIXmt (slave out). Atlas utilizes standard SPI signaling and timing control for the hardware interface and implements a higher level protocol on top of this.

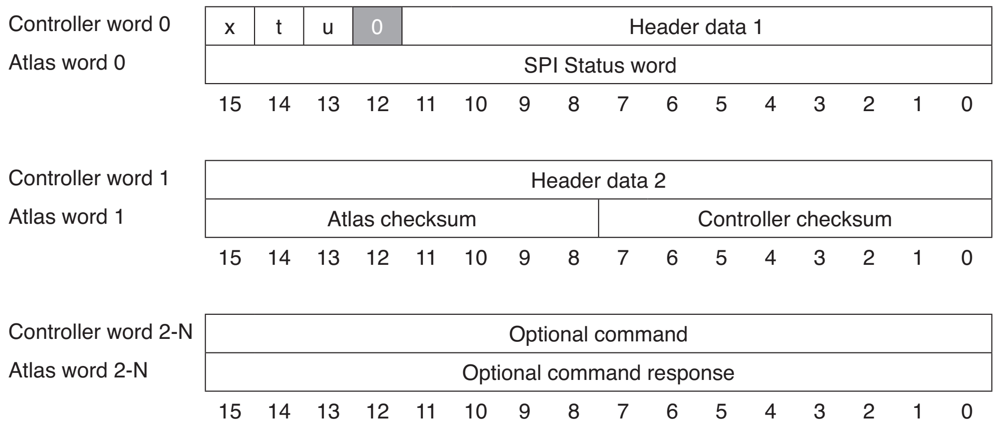


Figure 11-5: SPI Atlas Communications Protocol Overview

All communications to and from Atlas are in the form of a packet. Figure 11-5 shows the overall packet format. All Atlas SPI packets are comprised of a two word header and one or more optional command words. For more information, refer to the *Atlas Digital Amplifier User Manual.*

11.5 Setting PWM Frequency

If using the PWM output mode, under some circumstances, and for some products, the PWM output frequency may be specified using the command SetPWMFrequency. The value set using this command may be retrieved using GetPWMFrequency.

Product	PWM frequency choices
MC50000	20 kHz. 40 kHz (MC58113 only) 80 kHz.
ION	20 kHz. 40 kHz.

Choice of one of these frequencies over another depends on motor type and application. The default is 20 kHz, which is also the recommended frequency except for very low inductance motors.

For ION, 20kHz is the default PWM rate, and is recommended for most applications. 40kHz may be used effectively with some low-inductance motors.

11.6 Setting PWM Signal Interpretation (MC58113-Series Only)

The MC58113 allows output PWM signals to have their active/inactive interpretation programmed, meaning that normally active high PWM switch outputs can be changed to active low. This is accomplished via the SetDrivePWM command.

Note that this command does not affect the sign signal interpretation when motor output mode is set to Sign/Magnitude PWM.

11.7 Multi-Phase Motor Interfacing

(MC50000 only)

The general concepts of PWM or analog signal output apply for both single (DC brush) and multi-phase motors (brushless DC, step motors). Depending on the waveform and the motor output mode selected, either two or three phase commands per axis will be provided by the motion control IC.

For specific pin assignments of the PWM and DAC motor output signals, see the Pin Descriptions section of the appropriate *Magellan Motion Control IC Electrical Specifications* (IC users), or the *Magellan Motion Controller Card User's Guide* (card users).

For DC brush motors, which are single phase devices, each PWM or analog output drives the motor's single coil. For multi-phase motors the connections scheme differs somewhat depending on whether PWM or analog output has been chosen.

For Atlas-connected axes, there is no interface difference between single or multi-phase motors. All such differences are handled via the Magellan to Atlas protocol by the use of different command formats over the SPI Atlas interface. Therefore the following sections on multi-phase signal interfacing do not apply to Atlas based amplifier connections.

11.7.1 Multi-Phase Motor Command Interpretation

As for single-phase output, the commands for multi-phase motor phases represent a desired voltage or torque through the coil, and this command can be positive or negative. Unlike single-phase motors, the output waveforms for multi-phase motors are sinusoidal in shape, with the magnitude of the sinusoid reflecting the overall motor torque desired, and the angle of the sinusoid reflecting phasing requirements to correctly excite the motor coils. Figure 11-6 shows the desired output voltage waveform for a single phase of a multi-phase motor.

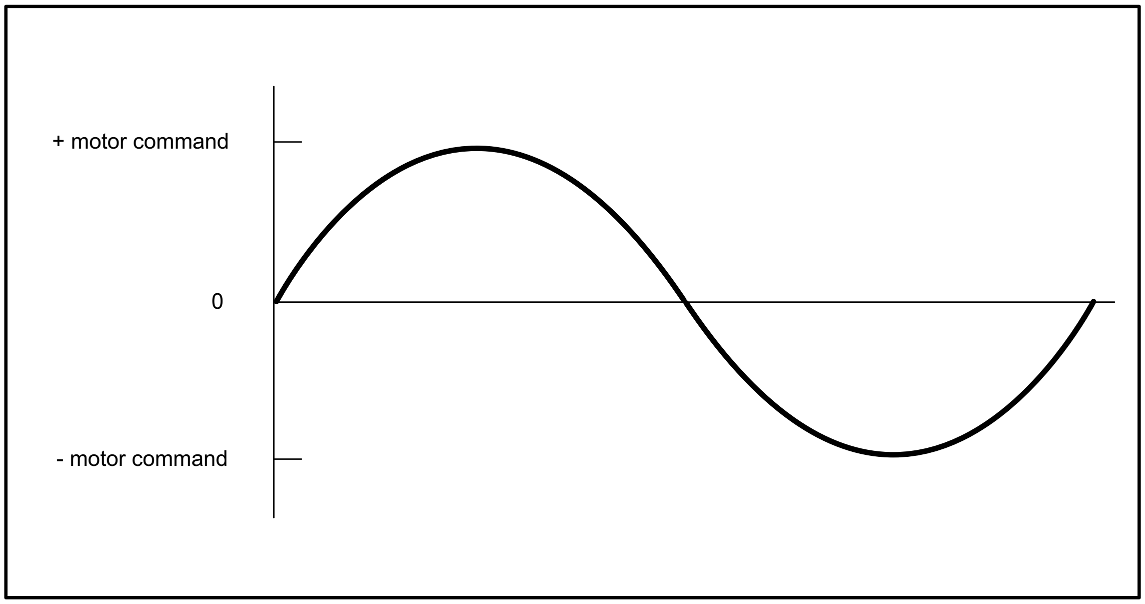


Figure 11-6: Motor Output Waveform (Vout)

The waveform is centered around a value of 0V. Depending on what control modules are enabled, the magnitude of the generated waveform is proportional to either the output of the position loop, the current loop, or the Motor Command register.

For example, if the motion control IC is connected to a DAC with output range of –10V to +10V, and the motion control IC has both the trajectory generator and position loop disabled with a motor command value of 32,767 (which is the maximum allowed value) loaded into the Motor Command register (SetMotorCommand), then as the motor rotates through a full electrical cycle, a sinusoidal waveform centered at 0V will be output with a minimum voltage of –10V and a maximum voltage of +10V.

11.7.2 Brushless DC Motor Output

Figure 11-7 illustrates a typical amplifier configuration for a three-phase brushless motor using PWM High/Low output with current control active. In this configuration the MC58113-series IC outputs six phased PWM magnitude signals per axis and uses three leg current input signals to form a complete bridge controller with high performance current control. See the MC58113 Electrical Specification for more details on this motor drive configuration.

Figure 11-7: Brushless DC Motor (PWM High/Low With Current Control) Connection Scheme (MC58113-Series ICs Only)

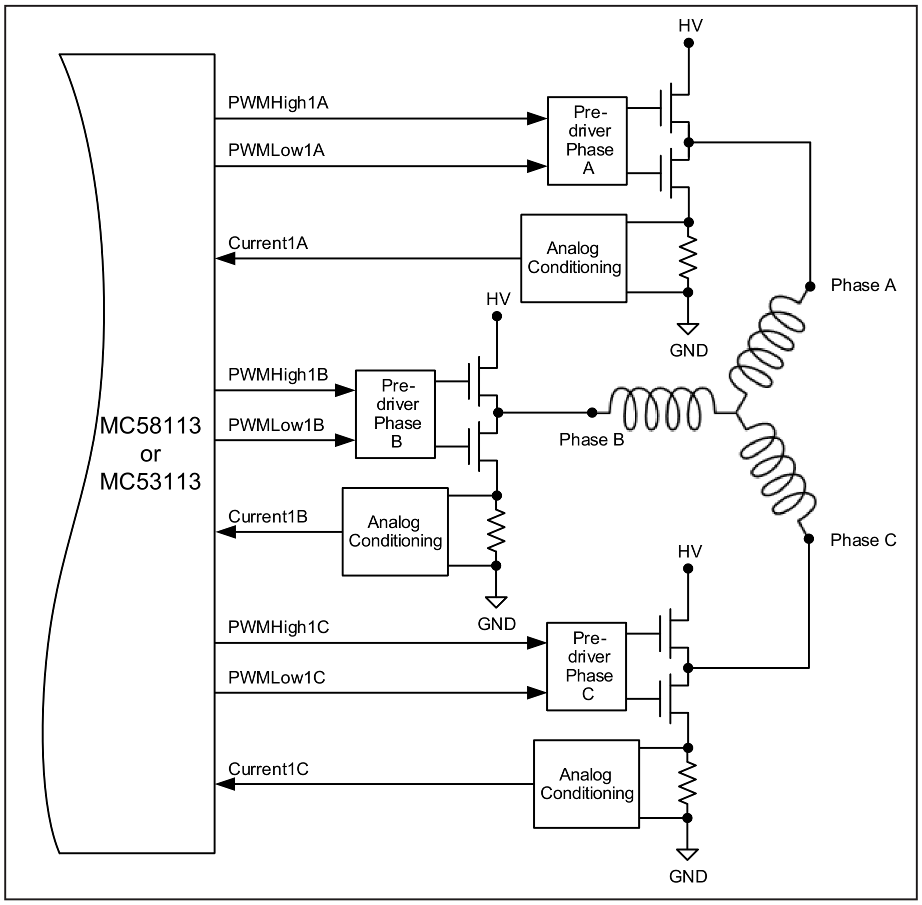


Figure 11-8 illustrates a typical amplifier configuration for a three-phase brushless motor using the 50/50 PWM output mode without current control. In this configuration, the motion control IC outputs three phased PWM magnitude signals per axis. These signals are then fed directly into three half-bridge type voltage amplifiers.

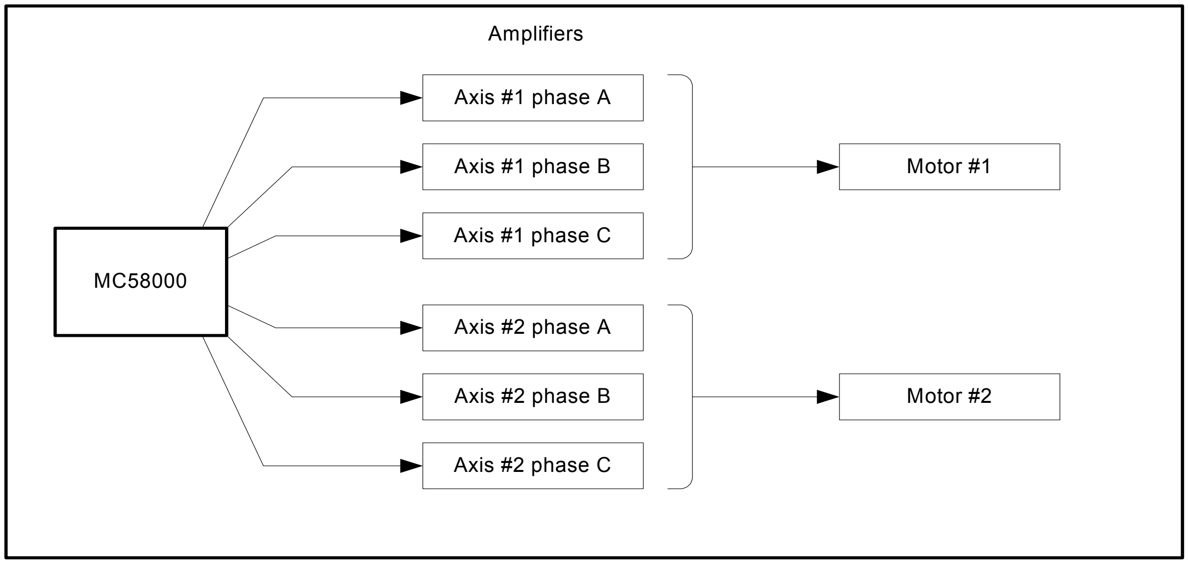


Figure 11-8: Brushless DC Motor (50/50 PWM Mode) Connection Scheme

Figure 11-9 illustrates a typical amplifier configuration for a three-phase brushless motor using the DAC output mode.

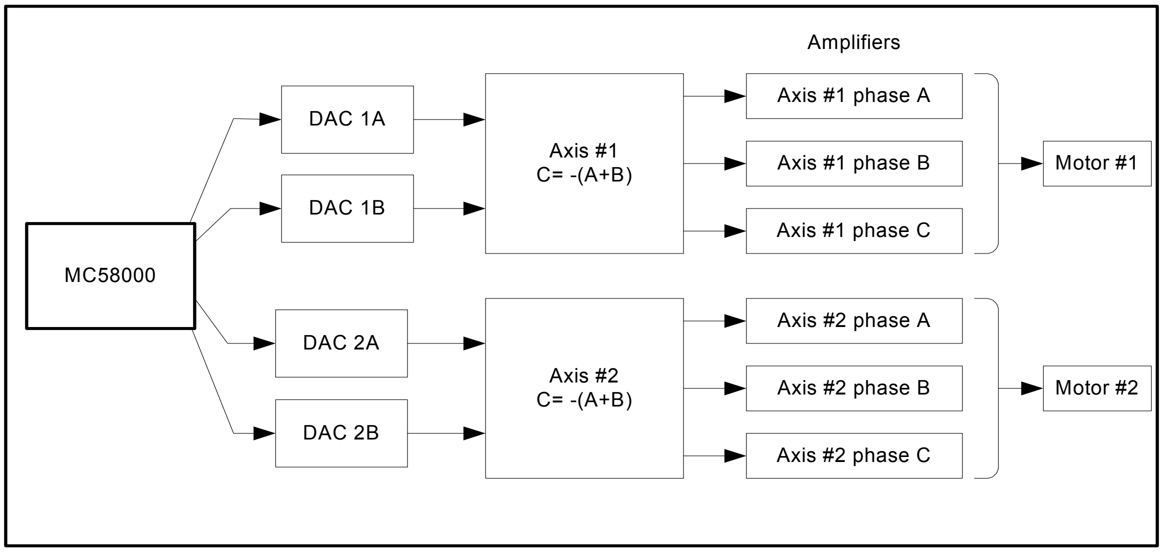


Figure 11-9: Brushless DC Motor (DAC Mode) Connection Scheme

When using DAC output mode, the digital word provided by the motion control IC must first be converted into a voltage using an external DAC. Two DAC channels are required per axis. To construct the third phase for a brushless motor (C phase), the sum of the A and B signals must be negated using C = –(A+B). This is usually accomplished with an op-amp circuit.

11.8 Microstepping Motor Output

Figure 11-10 illustrates a typical amplifier configuration for a two-phase step motor using PWM High/Low output with active current control. In this configuration the MC58113-series ICs output eight phased PWM magnitude signals per axis and use four leg current input signals to form a complete dual bridge controller with high performance current control. See the MC58113 Electrical Specification for more details on this motor drive configuration.

Figure 11-10: Two-Phase Step Motor (PWM High/ Low With Current Control) Connection Scheme (MC58113-Series ICs Only)

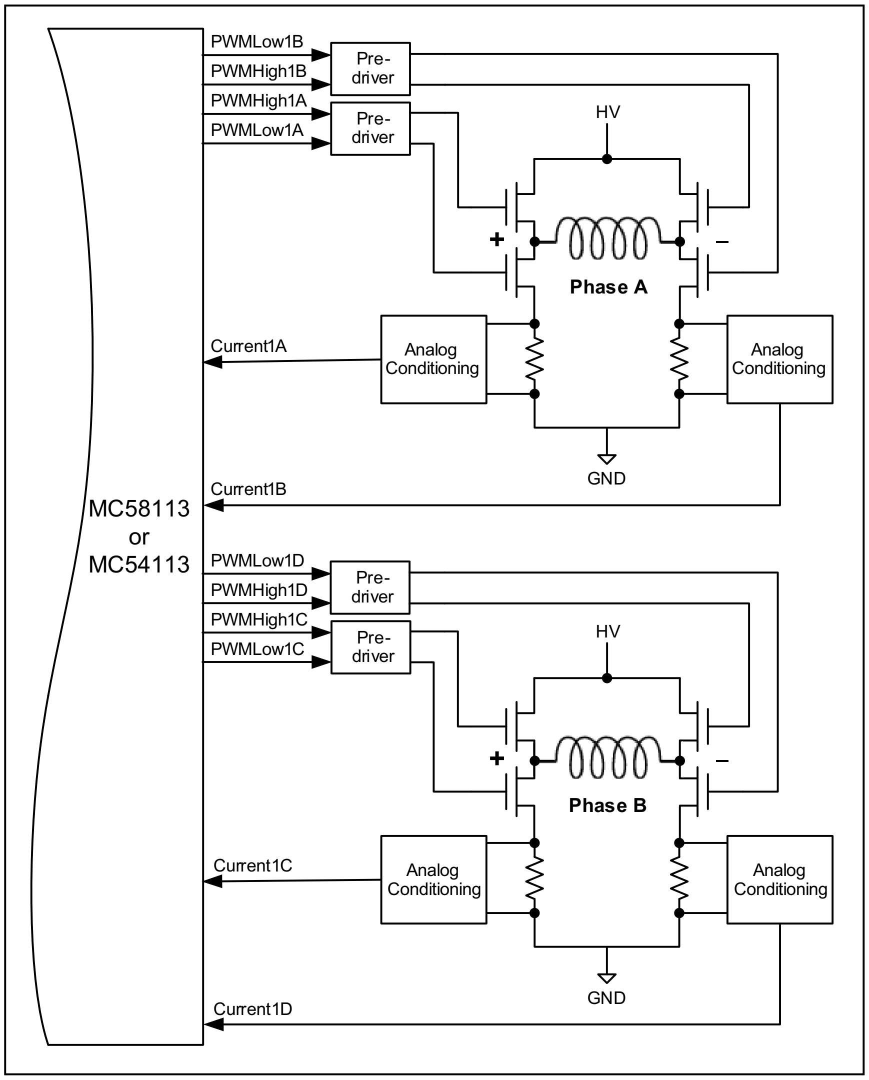


Figure 11-11 illustrates a typical amplifier configuration for a two-phase step motor, using either the sign/magnitude PWM or DAC output modes.

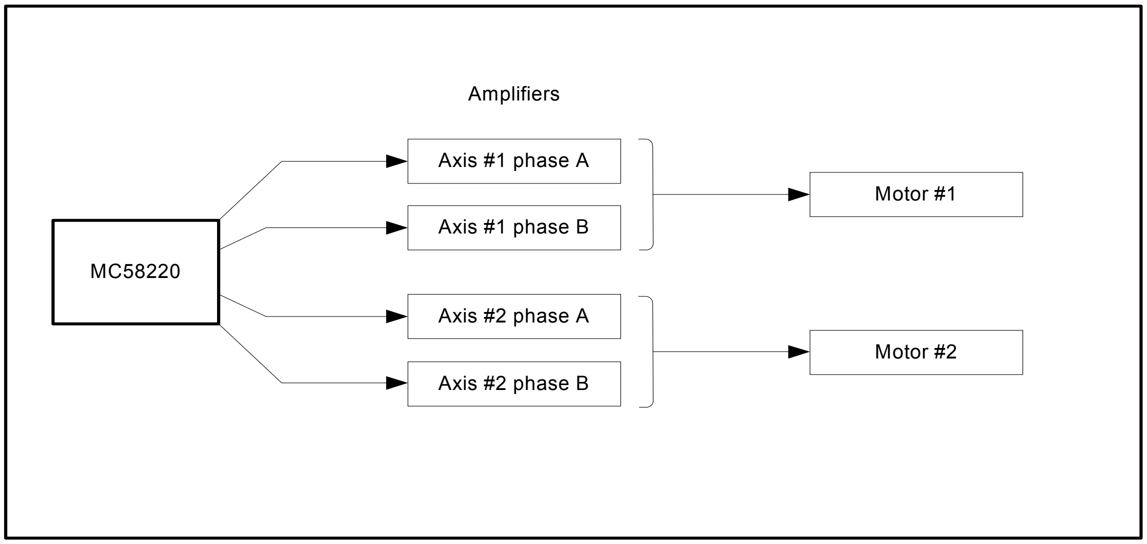


Figure 11-11: Two-Phase Motor (PWM Sign/ Magnitude or DAC) Connection Scheme

Using the DAC output mode, the digital motor output word for each phase is typically converted into a DC signal with a value ranging between –10V to +10V. This signal can then be input into an off-the-shelf DC-Servo type amplifier (one amplifier for each phase), or into any other linear or switching amplifier that performs current control and provides a bipolar, two-lead output.

In this scheme, each amplifier drives one phase of the step motor, with the motion control IC generating the required sinusoidal waveforms in each phase to perform smooth, accurate motion.

If the motion control IC's sign/magnitude PWM output mode is used, the PWM magnitude and sign signals can be directly connected to an H-bridge type device.

For maximum performance however, current control should be considered. Active current control minimizes the coil current distortion due to inductance and back-EMF. Although there are several methods which may be used to achieve current control with the PWM output mode, a common method is to pass the PWM magnitude signal through a low pass filter. This creates an analog reference signal that can be directly compared with the current through the coil. This scheme is shown in Figure 11-12.

Figure 11-12 illustrates this amplifier configuration.

Figure 11-12: Two-Phase Step Motor (Sign/ Magnitude PWM With Current Control) Connection Scheme

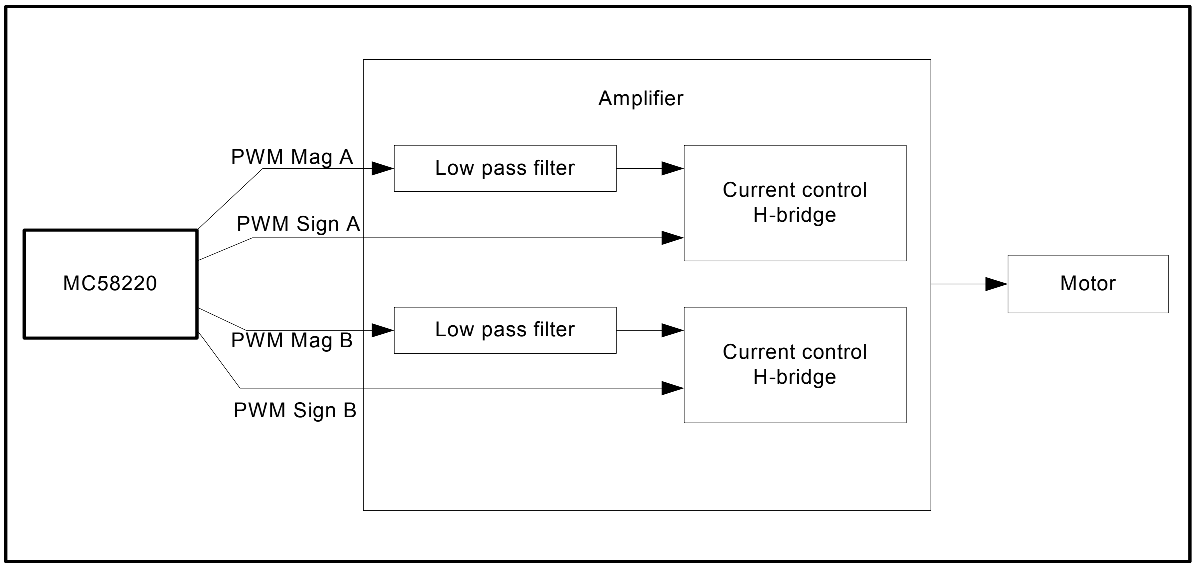
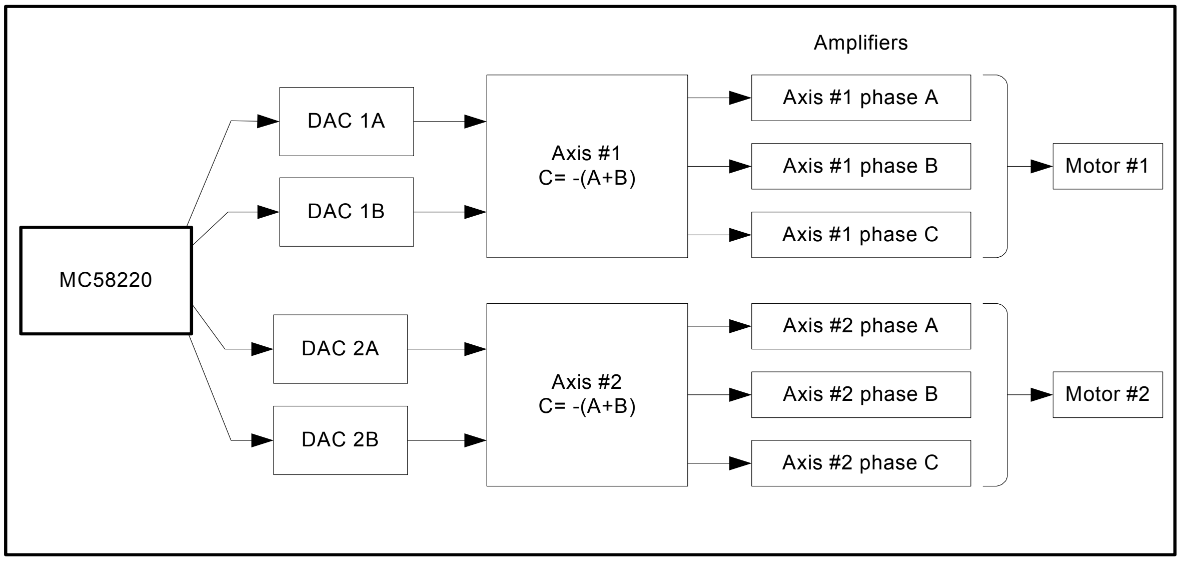


Figure 11-13 illustrates a typical amplifier configuration using the MC58220 in DAC mode for a three-phase step motor, or for a brushless DC motor with three phases.

Figure 11-13: Typical Amplifier Configuration For 3-Phase Step Motor



When using DAC output mode, the digital word provided by the motion control IC must first be converted into a voltage using an external DAC. Two DAC channels are required per axis. The third phase is constructed externally, using the expression C = –(A+B). This is usually accomplished with an op-amp circuit.

11.9 DC Servo Motor Output

Figure 11-14 illustrates a typical amplifier configuration for a DC Brush motor using PWM High/Low output with active current control. In this configuration the MC58113-series ICs output four phased PWM magnitude signals per axis and use two leg current input signals to form a complete bridge controller with high performance current control. See the MC58113 Electrical Specification for more details on this motor drive configuration.

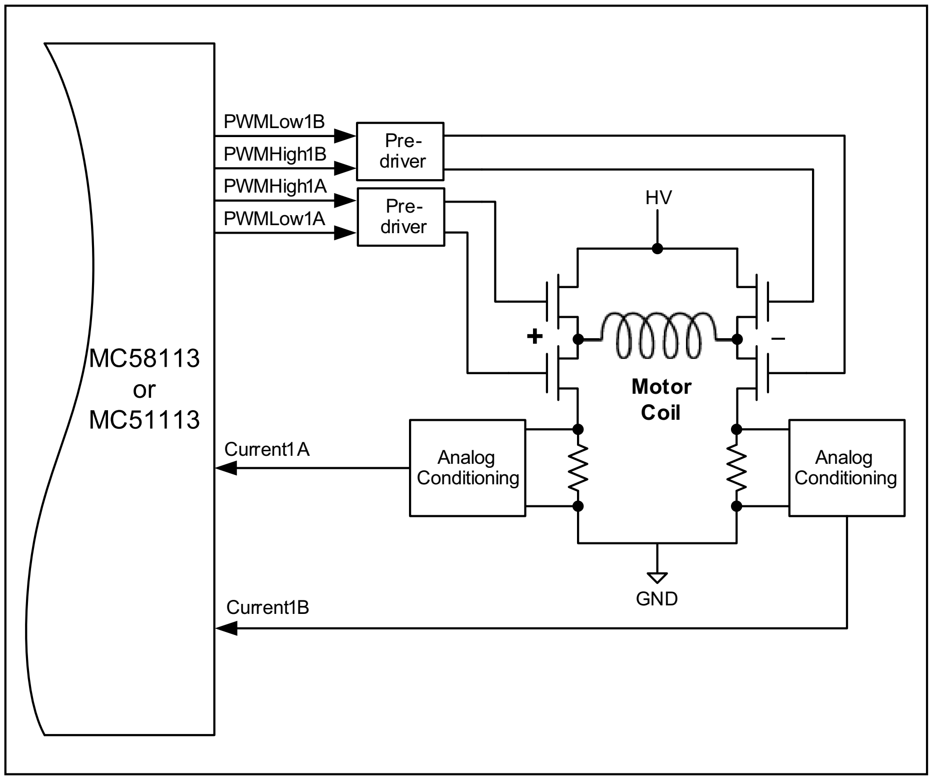


Figure 11-14: DC Brush Motor (PWM High/Low With Current Control) Connection Scheme (MC58113-Series ICs Only)

Figure 11-15 illustrates a typical amplifier configuration using DAC output mode.

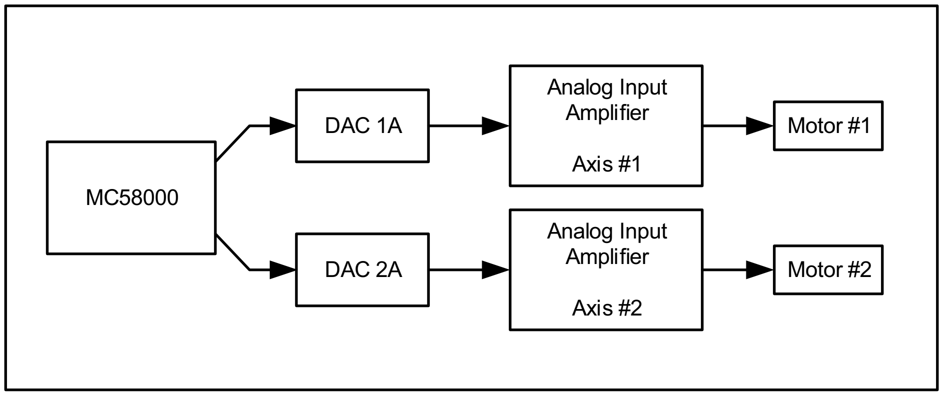

MC58000
DAC 1A
DAC 2A
Analog Input Amplifier Axis #1
Analog Input Amplifier Axis #2
Motor #1
Motor #2

Figure 11-15: DC Brush Motor (DAC) Connection Scheme

Using the DAC output mode, the digital motor output word is typically converted into a DC signal with a value ranging between –10V to +10V. This signal can then be input into an off-the-shelf DC-Servo type amplifier or into any other analog input linear or switching amplifier.

If the motion control IC's sign/magnitude PWM output mode is used there are two primary connection options. The first is to pass the PWM magnitude signal through a low pass filter, creating an analog reference signal that can be used with H-bridge amplifiers that provide current control. This is shown in Figure 11-16.

Alternatively the sign/magnitude outputs can directly drive an H-bridge in voltage mode (without active current control).

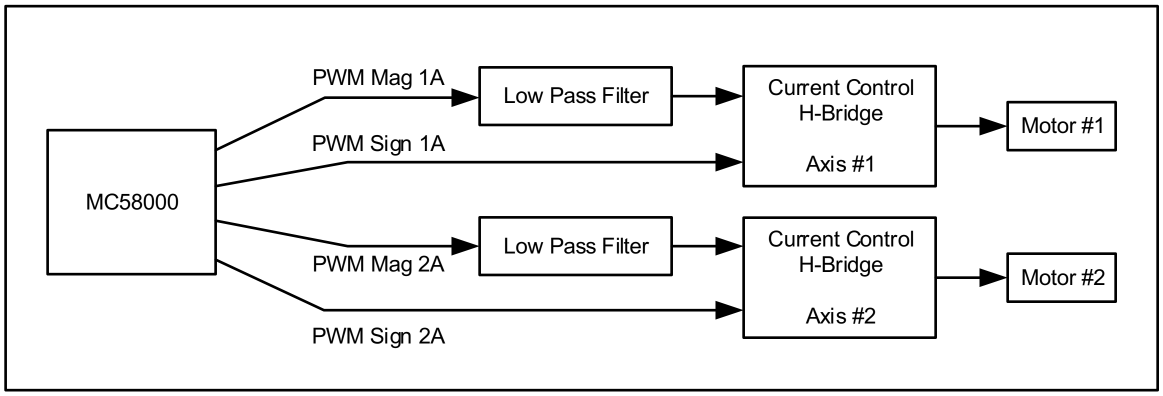


Figure 11-16: DC Brush Motor (Sign/ Magnitude PWM With Current Control) Connection Scheme

11.10 Pulse & Direction Signal Generation

(MC50000 only)

For step motors, in addition to signalling to drive each phase of a microstepping amplifier, step pulse and direction signals are provided to directly interface to amplifiers utilizing this format.

The step pulse signal, which is output by the motion control IC, consists of a precisely controlled series of individual pulses, each of which represents an increment of movement. This signal is always output as a square wave pulse train By default, a step pulse is considered to have occurred when the signal transitions from a low to a high output value. While not necessarily a 50% duty cycle, the square wave's rising edges are precisely timed. To invert the interpretation of the pulse signal, use the **SetSignalSense** command. See Section 7.5.1, "Signal Sense Mask," for more information.

The direction signal is synchronized with the pulse signal at the moment each pulse transition occurs. The direction signal is encoded so that a high value indicates a positive direction pulse, and a low value indicates a negative direction pulse. To invert this interpretation, use the **SetSignalSense** command. See Section 7.5.1, "Signal Sense Mask," for more information. Note that if the direction signal is inverted, then if the position feedback mode is internal loop-back, the "A" encoder signal must be inverted also to preserve the correspondence between direction and actual motor motion.

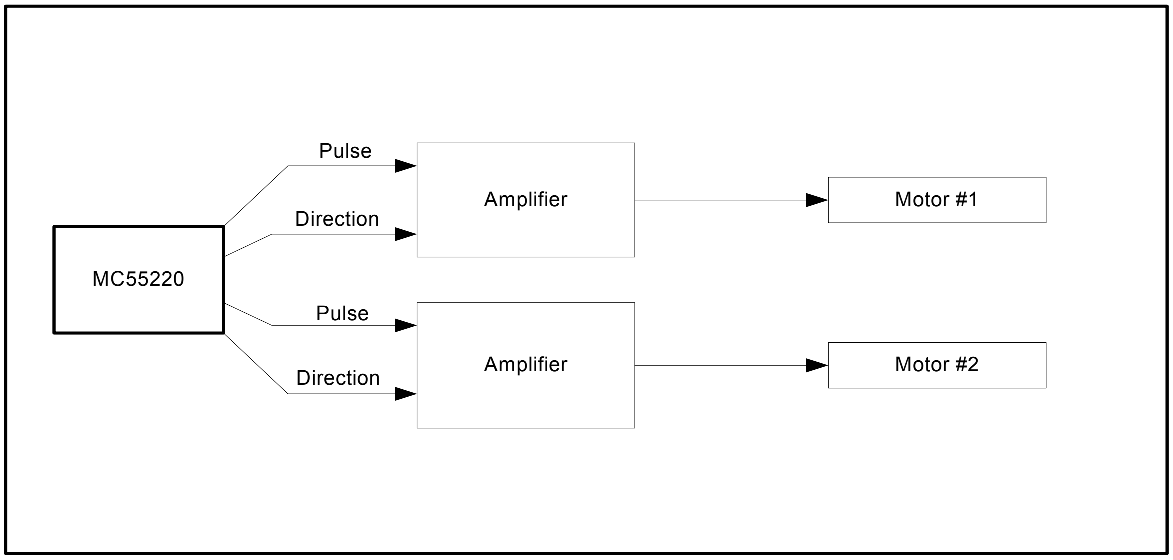


Figure 11-17: Pulse and Direction Motor Output Connection Scheme

The Magellan series of motion control ICs support separate pulse rate modes using the command SetStepRange. The following table shows the values and resultant step ranges available using this command.

Command	Frequency range of output pulses
SetStepRange 1	0 to 4.98 M steps per second.
SetStepRange 4	0 to 622.5 K steps per second.
SetStepRange 6	0 to 155.625 K steps per second.
SetStepRange 8	0 to 38.90625 K steps per second.

The maximum pulse output rate on the MC55110 and MC58110 is 100K steps per second. The MC58113 maximum rate is 1M steps per second. On these three devices the SetStepRange command cannot be used.

The ranges in the preceding example show the maximum and minimum ranges that can be generated by the motion control IC for the specified mode. For example, if the desired maximum step rate is 200 K steps per second, then the appropriate setting is SetStepRange 4.

For full-step and half-step applications, as well as pulse and direction applications that will have a maximum velocity of approximately 38 ksteps/sec, SetStepRange 8 should be used. For applications requiring higher pulse rates, one of the higher speed ranges should be specified.

A different step range can be programmed for each axis. To read the current step range setting, use the command GetStepRange.

The pulse generator is designed so that a step occurs when the pulse transitions from low to high. Systems using step motor drivers which interpret a pulse as a high-to-low transition should use the SetSignalSense command to invert the signal sense. SetSignalSense can also be used to set the polarity of the *Direction* output. Refer to the *Magellan Motion Control IC Programmer's Command Reference* for information on SetSignalSense command.

See Chapter 14, “Step Motor Control,” for more information on step motor control.

This page intentionally left blank.

12. Host Communication

In This Chapter

- Host Communication Packets
- Parallel Communications
- Serial Port
- Controller Area Network (CAN) Communications
- Storing Communication Values Using ION
- SPI (Serial Peripheral Interface) Communications

Magellan Motion Control ICs communicate with the host through one of four methods: a bi-directional parallel port, an asynchronous serial port, CANbus 2.0 standard, or SPI (Serial Peripheral Interface). However, not all Magellan processors support all communication modes. This is summarized in the table below.

	Parallel	Serial	CANbus	SPI
MC50000 except MC58113	yes	yes	yes	no
MC58113	no	yes	yes	yes
ION*	no	yes	yes	no

* ION as well as various PMD card products also support Ethernet communications, although this functionality is implemented external to the Magellan IC.

12.1 Host Communication Packets

All communications to and from the Magellan, whether parallel, serial, CANbus, or SPI are in the form of packets. A packet is a sequence of transfers to and from the host, resulting in a Magellan action or data transfer. Packets may consist of a command with no data (dataless command), a command with associated data that are written to the chipset (write command), or a command with associated data that are read from the chipset (read command).

Every command sent by the host has a structured format that does not change, even if the amount of data and nature of the command vary. Each command has an instruction word (16 bits) that identifies the command. There may be zero or more words of data associated with the command that the host writes to the motion control IC. This is followed by zero or more words of data that the host reads from the motion control IC. Finally, there is an optional checksum that may be read by the host to verify that communications are occurring properly. The type of command determines whether there are data written to the motion control IC and to the host.

Most commands with associated data (read or write) have one, two, or three words of data. See the *Magellan Motion Control IC Programmer's Command Reference* for more information on the length of specific commands. If a read or a write command has two words of associated data (a 32-bit quantity), the high word is loaded/read first, and the low word is loaded/read second.

The following tables show the generic command packet sequence for a dataless command, a write command, and a read command. The columns at the right of the tables list the corresponding hardware communication operation. These are either a command write, a data read, or a data write.

Dataless Command	Data Transfer
1	**Command Write:** Command Word
2 (optional)	**Data Read:** Packet Checksum

Write Command	Data Transfer
1	**Command Write:** Command Word
2	**Data Write:** Word 1
3 (optional)	**Data Write:** Word 2
4 (optional)	**Data Write:** Word 3
5 (optional)	**Data Read:** Packet Checksum

Read Command	Data Transfer
1	**Command Write:** Command Word
2 (optional)	**Data Write:** Word 1
3	**Data Read:** Word 1
4 (optional)	**Data Read:** Word 2
5 (optional)	**Data Read:** Packet Checksum

Design Note: While some users will develop their own low-level libraries for interfacing to MC50000 motion control ICs and cards, PMD's higher-level language tools, such as C-Motion and VB-Motion, provide convenient APIs (Application Programmmer Interface) for all Magellan commands.

12.2 Parallel Communications

(MC50000 except MC58113)

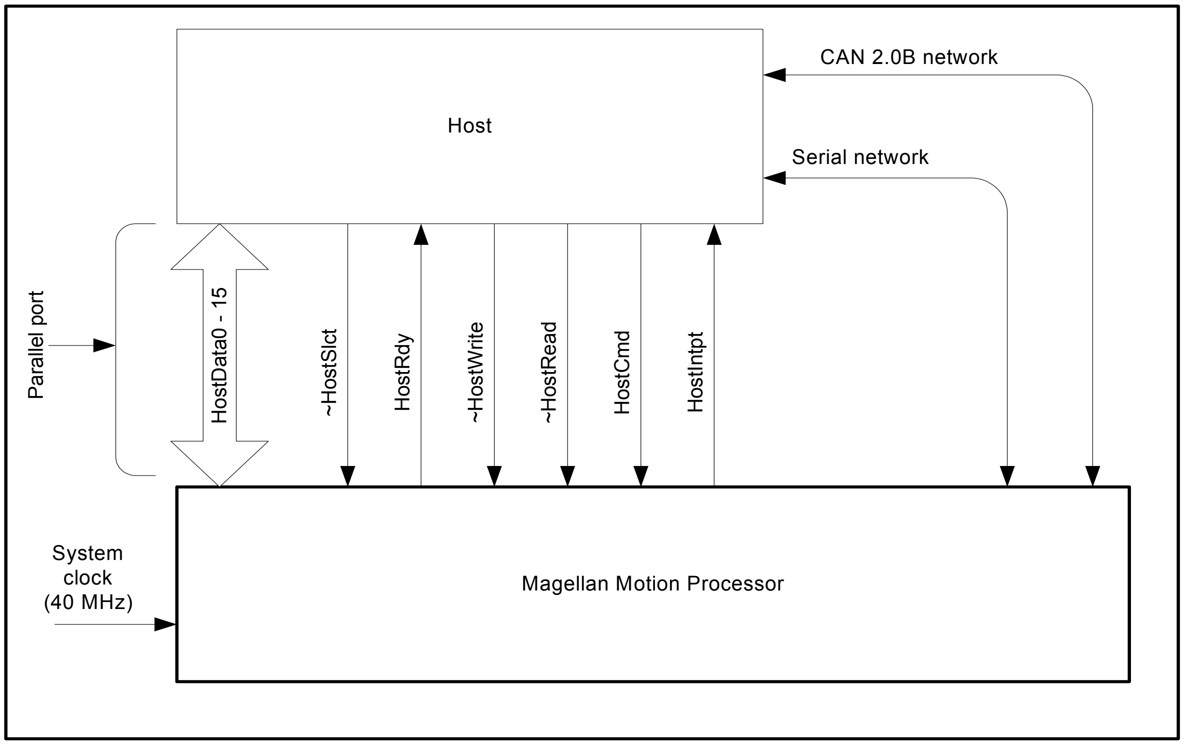


Figure 12-1: Parallel Communication Interconnections

MC58000 products except MC58113 offer a parallel word interface for connection to microprocessors or I/O ports that use this fornat. Figure 12-1 provides an overview of this connection scheme.

The parallel port is configured to operate in one of two modes, 16-bit and 8-bit, as described in the following table. Although 16-bit communication is faster and simpler to implement, 8-bit communication may be useful for direct interface with microprocessors that have an 8-bit external interface.

16-bit mode	The motion control IC transfers instructions and data as full 16-bit words, using the entire 16-bit data path.
8-bit mode	The motion control IC transfers instructions and data as full 16-bit words, using an 8-bit data path. Words are transferred in two successive bytes; the high-order byte of each word is transferred first in all cases. This mode allows access to all features of the Magellan instruction set, even when the host is limited to an 8-bit data path.

The parallel port configuration is determined by HostMode1 and HostMode0 pins on the MC5000's I/O chip. For more information see the *Magellan Motion Control IC Electrical Specifications.*

12.2.1 Interfacing

Five control signals synchronize communications through the parallel port: ~HostSlct, HostRdy, ~HostWrite, ~HostRead, and HostCmd. The descriptions of these signals are as follows.

Signal	Description
~HostSlct	Set by host—when this signal is asserted (low), the host parallel port is selected for operations.

Signal	Description		
HostRdy	Set by motion control IC—when high, indicates to the host that the motion control IC's host port is available for operations.		
~HostWrite	Set by host—when asserted low, allows a data transfer from the host to the motion control IC.		
~HostRead	Set by host—when asserted low, allows a data word to be read by the host from the motion control IC.		
HostCmd	Used in conjunction with the ~HostRead and ~HostWrite signals as follows:		
	When	and HostCmd is	Then
	~HostWrite is low	low	write data word to motion control IC
		high	write instruction word to motion control IC.
	~HostRead is low	low	read data word from motion control IC.
		high	read status word from motion control IC (see Section 12.2.3, "The Status Read Operation" for more information).

12.2.2 Parallel Port I/O Operations

Using the five parallel port control signals of ~HostSlct, HostRdy, ~HostWrite, ~HostRead, and HostCmd, it is possible to perform all necessary operations to send commands to the motion control IC.

The three operations are: the *instruction word write*, the *data word write*, and the *data word read*. By performing these operations in the correct sequence, the complete command packets can be assembled and sent to the motion control IC. Command format is discussed in the *Magellan Motion Control IC Programmer's Command Reference.*

instruction word write — In 16-bit bus mode, this is accomplished by asserting ~HostSlct and ~HostWrite low, asserting HostCmd high, and loading the data bus with the desired 16-bit instruction word value. In 8-bit bus mode, the control signals are the same, except that only the low 8 bits of the data bus hold data, and the operation is performed twice. On the first 8-bit write, the data bus should contain the high byte of the instruction word. On the second 8-bit write, the data should contain the low byte of the instruction word.

data word write —In 16-bit bus mode this is accomplished by asserting ~HostSlct, and ~HostWrite low, asserting HostCmd low, and loading the data bus with the desired 16-bit data word value. In 8-bit bus mode, the control signals are the same, except that only the low 8 bits of the data bus hold data, and the operation is performed twice. On the first 8-bit write, the data bus should contain the high byte of the data word. On the second 8-bit write, the data should contain the low byte of the data word.

data word read—In 16-bit bus mode this is accomplished by asserting ~HostSlct, and ~HostRead low, asserting HostCmd low, and storing the value asserted by the motion control IC on the 16-bit data bus. On the first 8-bit read, the data bus will contain the high byte of the data word. On the second 8-bit read, the data will contain the low byte of the data word.

At the beginning of each of these operations, the HostRdy signal must be high. This indicates that the motion control IC is ready to receive or transmit a new data or instruction word. In between 8-bit transfers, the HostRdy does not need to be checked. For example, after checking the HostRdy for the high byte of any of these 2-byte transfers (*instruction word write, data word write, or data word read*), the HostRdy does not have to be checked again to transfer the low byte.

For more detailed electrical information on these operations, see the pin descriptions and timing diagrams in the *Magellan Motion Control IC Electrical Specifications* document for your motion control IC model.

Before any parallel host I/O operation is performed, the user must make sure that the *HostRdy* signal is high (motion control IC ready). After each word (instruction or data) is read or written, this signal will go low (motion control IC busy). It will return to ready when the motion control IC is ready to transfer the next word.

12.2.3 The Status Read Operation

There is a special operation called a status read, which is not directly related to reading or writing to the motion control IC. The status read allows the user to determine the state of some of the motion control IC's host interface signals and flags without having to develop special decode logic.

A status read operation is performed by asserting ~HostRead and ~HostSlct low, HostCmd high, and reading the data bus. The resultant data word sent by the motion control IC contains the following information:

Bit number	Description
0–12	Not used.
13	Indicates whether an instruction error has occurred (see Section 12.2.5, "Instruction Errors" for more information).
14	Holds value of HostIntrpt signal. A value of one indicates the signal level is high.
15	Holds value of HostRdy signal. A value of one indicates the signal level is high.

Status reads may be performed at any time regardless of the state of the *HostRdy* signal.

12.2.4 Checksum

It is possible to retrieve a checksum at the end of each read and/or write command. The checksum may enhance reliability for critical applications, particularly in very noisy electrical environments. Checksums may also be helpful in situations where the communication signals are routed over a medium that may have data losses. The checksum consists of the low-order 16 bits of the sum of all preceding words transmitted in the command. For example, if a SetVelocity instruction (which takes two 16-bit words of data) is sent with a data value of FEDCBA98 (hex), the checksum would be:

0011h	code for SetVelocity instruction
+ FEDCh	first data word
+ BA98h	second data word
= 1B985h	checksum = B985 (low-order 16 bits only)

Reading the checksum is optional. Recovery from an incorrect command transfer (bad checksum) will depend on the nature of the packet. Buffered operations can always be re-transmitted, but a non-buffered instruction (one that causes an immediate action) might or might not be re-transmitted, depending on the instruction and the state of the axis.

12.2.5 Instruction Errors

There are a number of checks made by the motion control IC on the commands sent to it by the host, as well as checks during communication between the Magellan and any connected Atlas amplifiers. These checks improve safety of the motion system by eliminating incorrect command data values. All such checks are referred to as instruction errors. If any such error occurs, bit 13 of the I/O status read word is set. (See Section 12.2.3, "The Status Read Operation" for the definition of this word.) To determine the error's cause, the command GetInstructionError is used. Executing the GetInstructionError command also clears both the error code and the I/O error bit in the I/O status read word.

For Atlas-connected axes instruction errors may also occur in connection with a specific command sent by the host to the Magellan, or as part of continuous background communications between the Magellan and the Atlas. Therefore additional checks should be made to determine the nature of a reported instruction error. To accomplish this the host should send a GetDriveFaultStatus command to determine if specific SPI communication checksum errors have occurred. Then a GetInstructionError command should be sent directly to the Atlas amplifier to determine if it has recorded an instruction error. See Section 15.15, "Atlas-Specific Commands" for information on how the host can send commands to a specific Magellan-connected Atlas.

For more information on the fields encoded in this register, refer to the GetInstructionError command description of the *Magellan Motion Processor Programmer's Command Reference*. For more information on Atlas SPI-related command refer to the *Atlas Digital Amplifier Complete Technical Reference*.

12.3 Serial Port

All Magellan Motion Control ICs provide an asynchronous serial connection. This serial port may be configured to operate at baud rates ranging from 1200 baud to 460,800 baud and may be used in point-to-point or multi-drop mode.

12.3.1 Configuration

After reset, the motion control IC reads the default configuration of the serial port. For MC50000 except MC58113 external hardware is used to provide a configuration word via the peripheral I/O bus as described in the following table. See the *Magellan Motion Control IC Electrical Specifications* for details on peripheral bus I/O.

Bit	Parameter	Indications	
0 – 3	Transmission rate selector	0	1,200 bits per second
		1	2,400 bps
		2	9,600 bps
		3	19,200 bps
		4	57,600 bps
		5	115,200 bps
		6	230,400 bps
		7	460,800 bps
4 – 5	Parity Selector	0	None
		1	Odd parity
		2	Even parity
		3	*Reserved (do not use)*
6	Number of stop bits	0	1 stop bit
		1	2 stop bits
7 – 8	Protocol type	0	Point-to-point
		1	Multi-drop (idle line mode)
		2	*Reserved (do not use)*
		3	*Reserved (do not use)*
9 – 10	*Reserved*	--	

Bit	Parameter	Indications	
11 – 15	Multi-drop address selector Should be zero in point-to-point mode	0	Address 0
		1	Address 1
		...	
		31	Address 31

This word is read by the motion control IC at peripheral address 200h.

For Prodigy Motion Card users, this 16-bit value is set using a DIP switch or by other means. For more information refer to the appropriate motion card user's guide.

For ION users, the default communication parameters come from one of two sources. They are either fixed (RS-232), or they come from the ION's non-volatile memory (RS-485). The following table illustrates this:

Communications Mode	Default Value
RS-232	Fixed at 57.6K, 1 stop bit, no parity.
RS-485	Initially set at 57.6K, 1 stop bit, no parity, point-to-point mode. Thereafter set using SetDefault command. See Section 12.5, "Storing Communication Values Using ION" for more information.

For MC58113 the default values are the same as for ION, however storing new default is accomplished differently. See the *MC58113 Electrical Specifications* for details.

The motion control IC's serial port can also be configured using the **SetSerialPortMode** command. Refer to the *Magellan Motion Control IC Programmer's Command Reference* for more information.

12.3.2 Command Format

The basic unit of serial data transfer (both transmit and receive) is the asynchronous frame. Each frame of data consists of the following components.

l One start bit.

l Eight data bits.

l An optional even/odd parity bit.

l One or two stop bits.This data frame format is shown in the following figure.

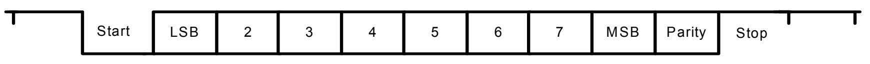


Figure 12-2: Typical Data Frame Format

The command format that is used to communicate between the host and motion control IC consists of a command packet sent by the host processor, followed by a response packet sent by the motion control IC. The host must wait for, receive, and decode the response packet.

Command packets sent by the host contain the following fields.

Field	Byte#	Description
Address	1	One byte identifying the motion control IC to which the command packet is being sent. This field should always be zero in point-to-point mode.
Checksum	2	One byte value used to validate packet data. See the table in Section 12.3.4, "Checksums" for detailed information.

Field	Byte#	Description
Instruction code	3–4	Two byte instruction, sent upper byte (axis number) first. The command codes are the same as those used in the parallel communication mode.
Data	5	Zero to six bytes of data, sent most significant byte (MSB) first. See the individual command descriptions for details on data required for each command.

In response to the command packet, the motion control IC will respond with a packet in the following format.

Field	Byte#	Description
Address	(1)*	One byte identifying the motion control IC sending the response. Present in multi-drop mode only.
Status	1 (2)	Zero if the command was completed correctly; otherwise, an error code specifying the nature of the error. See Section 12.2.5, "Instruction Errors" for more information.
Checksum	2 (3)	A one-byte checksum value used to validate the packet's integrity. See details in the preceding table.
Data	3 (4)	Zero to six bytes of data. No data will be sent if an error occurred in the command (i.e. the status byte was non-zero). If no error occurred, then the number of bytes of data returned would depend on the command to which the motion control IC was responding. Data are always sent MSB first.

* Note that the address byte is only present in the response packet when in multi-drop mode. In this case, its is also included in the checksum calculation.

12.3.3 Instruction Errors

There are a number of checks made by the chip on commands it receives. These checks improve safety of the motion system by eliminating some obviously incorrect command data values. All such checks associated with host I/O commands are referred to as instruction errors. The status byte in the response packet will contain one of the error codes. See the table in Section 12.2.5, "Instruction Errors" for more information.

12.3.4 Checksums

As for parallel communications, both command and response packets contain a checksum byte. The checksum is used to detect transmission errors, and allows the motion control IC to identify and reject packets that have been corrupted during transmission or were not properly formed.

Unlike the parallel interface however, checksums are mandatory when using serial communications. Any command packets sent to the motion control IC containing invalid checksums will not be processed and will result in a data packet being returned containing an error status code.

The serial checksum is calculated by summing all bytes in the packet (not including the checksum) and negating (i.e., taking the two's complement of) the result. The lower eight bits of this value are used as the checksum. To check for a valid checksum, all bytes of a packet should be summed (including the checksum byte), and if the lower eight bits of the result are zero, then the checksum is valid.

For example, if a command packet is sent to motion control IC address 3, containing command 0177h (SetMotorCommand for axis 2) with the one-word data value 1234h, then the checksum will be calculated by summing all bytes of the command packet (03h + 01h + 77h + 12h + 34h = C1) and negating this to find the checksum value (3Fh). On receipt, the motion control IC will sum all bytes of the packet, and if the lower eight bits of the result are zero, then it will accept the packet (03h + 3Fh + 01h + 77h + 12h + 34h = 100h).

12.3.5 Transmission Protocols

The Magellan Motion Control ICs support the ability to have more than one motion control IC on a serial bus, thereby allowing a chain, or network of motion control ICs, to communicate on the same serial hardware signals.

There are two methods supported by the serial port to resolve timing problems, transmission conflicts, and other issues that may occur during serial operations. These are point-to-point (used when there is only one device connected to the serial port) and multi-drop idle-line mode (used when there are multiple devices on the serial bus). The following sections describe these transmission protocols.

12.3.6 Point-to-Point Mode

Point-to-point serial mode is intended to be used when there is a direct serial connection between one host and one motion control IC. In this mode, the address byte in the command packet is not used by the motion control IC (except in the calculation of the checksum), and the motion control IC responds to all commands sent by the host.

When in point-to-point mode, there are no timing requirements on the data transmitted within a packet. The amount of data contained in a command packet is determined by the command code in the packet. Each command code has a specific amount of data associated with it. When the motion control IC receives a command code, it waits for all data bytes to be received before processing the command. The amount of data returned from any command is also determined by the command code. After processing a command, the motion control IC will respond with a data packet of the necessary length. No address byte is sent with this response packet.

When running in point-to-point mode, there is no direct way for the motion control IC to determine the beginning of a new command packet, except by context. Therefore, it is important for the host to remain synchronized with the motion control IC when sending and receiving data. To ensure that the processors remain synchronized, it is recommended that the host processor implement a time limit when waiting for data packets to be sent by the motion control IC. The suggested minimum timeout period is the amount of time required to send one byte at the selected baud rate plus one millisecond. For example, at 9600 baud each bit takes 1/9600 seconds to transfer, and a typical byte consists of 8 data bits, 1 start bit, and 1 stop bit. Therefore, one byte takes just over 1 millisecond, and the recommended minimum timeout is 2 milliseconds.

If the timeout period elapses between bytes of received data while the host is waiting on a data packet, then the host should assume that it is out of synchronization with the motion control IC. To resynchronize, the host should send a byte containing zero data and wait for a data packet to be received. This process should be repeated until a data packet is received from the motion control IC; at which point the two processors will be synchronized.

12.3.7 Multi-drop Idle-line Mode

This multi-drop protocol is intended to be used on a serial bus in which a single host processor communicates with multiple motion control ICs (or other subordinate devices). In this mode, the address byte that starts a command packet is used to indicate the device for which the packet is intended. Only the addressed device will respond to the packet. Therefore, it is important to properly set up the motion control IC address (using the serial configuration word previously described) and to include this address as the first byte of any command packet destined for the motion control IC.

When the idle-line mode is used, the motion control IC imposes tight timing requirements on the data sent as part of a command packet. In this mode, the motion control IC will interpret the first byte received after an idle period as the start of a new packet. Any data already received will be discarded.

The timeout period is equal to the time required to send ten bits of serial data at the configured baud rate—for example, roughly 1 millisecond at 9600 baud. If a delay of this length occurs between bytes of a command packet, then the bytes already received will be discarded, and the first character received after the delay will be interpreted as the address byte of a new packet.

Once the motion control IC receives an address byte and a command code, it waits for all data bytes to be received before processing the command. The amount of data returned from any command is also determined by the command code. After processing a command, the motion control IC will respond with a data packet of the necessary length. In multi-drop mode, the first byte of every response packet contains the address of the responding motion control IC. This prevents other devices on the network from interpreting the response as a command sent to them.

Note that this multi-drop protocol may also be used when the host and motion control IC are wired in a point-to-point configuration, as long as the host always transmits the correct address byte at the start of a packet and follows any additional rules for the selected protocol. This mode of operation allows the host to ensure that it will remain synchronized with the motion control IC without implementing the timeout and re-synch procedure previously outlined.

12.4 Controller Area Network (CAN) Communications

12.4.1 Overview

CAN is a serial bus system especially suited for networking "intelligent" devices as well as sensors and actuators within a system or sub-system.

All Magellan Motion Control ICs provide a CAN 2.0B network and will coexist (but not communicate) with CANOpen nodes on that network. Magellan uses CAN to receive commands, send responses, and (optionally) send asynchronous event notifications. Each message type has an address, as shown in the following table.

Message Type	CAN Address
Command received	0x600 + nodeID
Command response	0x580 + nodeID
Event notification	0x180 + nodeID

CAN nodes communicate via messages. Each message may carry a data payload of up to 8 bytes. The CAN interface layer automatically corrects transmission errors. Unlike the serial and parallel protocols, a checksum is not a part of the motion control IC's CAN interface protocol.

12.4.2 Message Format

Messages are transmitted and received using the standard format identifier length of 11 bits. All network messages that use the extended format 29-bit identifier are ignored by the motion control IC. The data formats for the three message types listed in the previous table are expressed in terms of the byte sequences for the parallel interface. Commands have varying data lengths; see the *Magellan Motion Control IC Programmer's Command Reference* for the data formats of particular commands. In the following table, bytes that will always be present independent of the command being processed are marked as Required.

The corresponding byte sequences in the CAN protocol for the three message types are described in the following tables.

Command Received		
Message data byte	Required?	Corresponding parallel byte
1	Y	Command word, high byte
2	Y	Command word, low byte
3	N	1st data word, high byte
4	N	1st data word, low byte
5	N	2nd data word, high byte

Command Received

Message data byte	Required?	Corresponding parallel byte
6	N	2nd data word, low byte
7	N	3rd data word, high byte
8	N	3rd data word, low byte

Command Response

Message data byte	Required?	Corresponding parallel byte
1	Y	*Reserved (always zero)*
2	Y	Instruction status code
3	N	1st data word, high byte
4	N	1st data word, low byte
5	N	2nd data word, high byte
6	N	2nd data word, low byte
7	N	3rd data word, high byte
8	N	3rd data word, low byte

The first word in a response will contain a value of zero in the upper byte, and the lower byte will contain a value that will also be zero in a no-error condition, but will be non-zero if an error occurred while processing the instruction. (See Section 12.2.5, “Instruction Errors” for more information.) The byte following the status byte will be the high byte of the 1st data word, followed by the low byte of the 1st data word and continuing as shown in the preceding table. The actual number of bytes returned is determined by the instruction that was issued; see the *Magellan Motion Control IC Programmer’s Command Reference* for the data lengths and formats of each command.

Event Notification

Message data byte	Data Interpretation
1	reserved (always zero)
2	axis number (0–3)
3	Event Status register value, high byte
4	Event Status register value, low byte

The first byte in a notification message will contain a value of zero with the second byte indicating the axis from which the notification was sent. The 3rd and 4th bytes are the high and low byte of the Event Status register from the notifying axis.

12.4.3 Configuring the CAN Interface

After reset, MC58000 motion control ICs except MC58113 read a 16-bit value from their peripheral bus (location 400h), to set the default configuration of the CAN interface. Refer to the *Magellan Motion Control IC Electrical Specifications* for details on peripheral bus I/O.

ION can permanently store interface parameters using the **SetDefault** command. See Section 12.5, “Storing Communication Values Using ION” for more information. MC58113 also allows permanent storage of communication parameters, but uses a different mechanism than ION. See the *MC58113 Electrical Specifications* for more information.

The motion control IC's CAN interface may also be configured via the command **SetCANMode**. This command is used to set the CAN nodeID of a particular motion control IC (0–127), as well as the transmission rate of the connected CAN network. The supported transmission rates are as follows:

SetCanMode Encoding	CAN Transmission Rate (bps)
0	1,000,000
1	800,000
2	500,000
3	250,000
4	125,000
5	50,000
6	20,000
7	10,000

12.4.4 CAN Event Notification

When communicating via the CAN interface, the motion control IC may (optionally) send messages when selected bits in the Event Status register are set active. This facility directly corresponds to the motion control IC's host interrupt facility when using the parallel interface. (See Section 8.10, "Host Interrupts" for more information.) These messages are sent with a CAN address of **0x180 + nodeID**.

This CAN notification facility is controlled with the command **SetInterruptMask**. For each **on** bit in the notify mask, a CAN message will be generated whenever the corresponding bit in the Event Status register becomes 1. See Section 8.10, "Host Interrupts" for more information.

12.5 Storing Communication Values Using ION

Particularly when configuring the motion control IC for its production connectivity, it is useful to be able to store communications parameters permanently, so that upon the next powerup, the motion control IC will utilize new communication parameters. ION allows this to be accomplished using the command **SetDefault**. The values set using this command can be read back using the command **GetDefault**.

The new communication parameters will take effect only after the next power cycle or reset. Therefore communication will continue at the present settings until this has occurred.

The following table defines which parameters can be stored using **SetDefault.**

Parameter	Description
CANMode	This parameter specifies the CAN configuration information, as described in Section 12.4.3, "Configuring the CAN Interface".
SerialPortMode	For RS485 only, this parameter specifies the serial port configuration word, as described in Section 12.3.1, "Configuration".

MC58113 also has the ability to permanently store values, but the mechanism used is different. See the *MC58113 Electrical Specifications* for more informaiton.

All other MC50000 ICs use external peripheral reads to determine communication settings,. See Section 12.3.1, "Configuration" and Section 12.4.3, "Configuring the CAN Interface" for details.

12.6 SPI (Serial Peripheral Interface) Communications

The MC58113-series ICs support an SPI interface for communication with a host microprocessor or other controller. Note that this interface is intended for on-card interconnections only. SPI should not be used for off-card communication.

The MC58113's SPI interface utilizes three digital input pins *HostSPIEnable*, *HostSPIClock* and *HostSPIRcv*, and two digital output pins *HostSPIXmt*, and *HostSPIstatus*. These signals represent the standard SPI enable, chip select, clock, and data functions, along with a protocol packet processing status indicator. When used with SPI host communications the MC58113 acts as an SPI slave, and the host processor acts as an SPI master.

12.6.1 SPI Protocol, Command Send

Each overall host SPI data exchange consists of the host asserting *HostSPIEnable*, the host sending two or more 16-bit data words, and the host de-asserting *HostSPIEnable*.

SPI commands that do not include data to send to the MC58113 (dataless commands) consist of two transmitted 16-bit words, commands that include one word of data consist of three transmitted 16 bit words, etc…

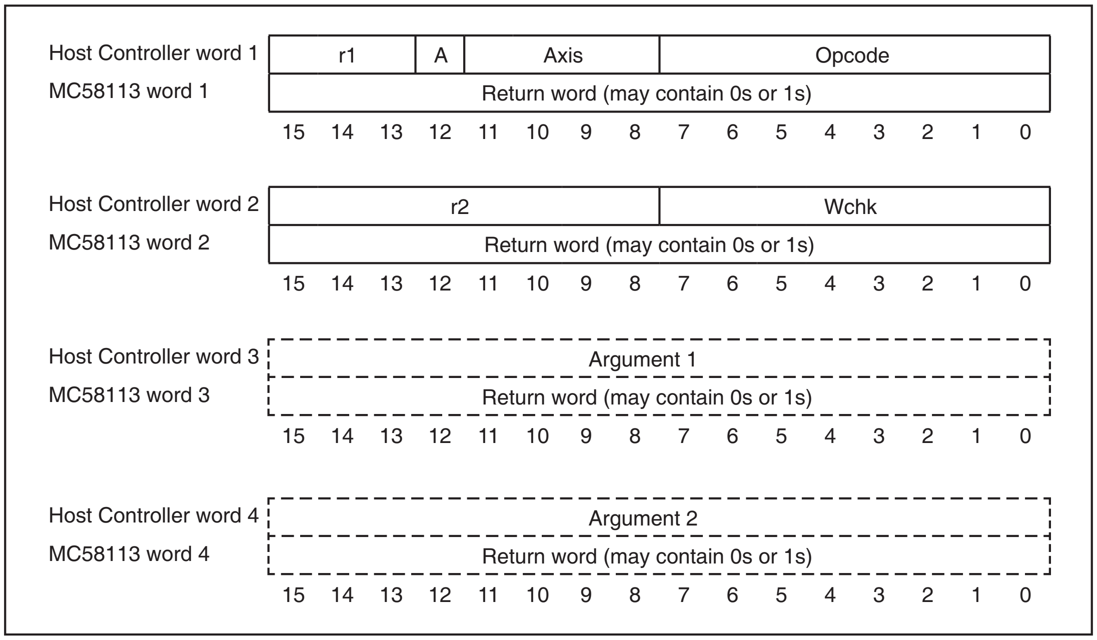


Figure 12-3: SPI Command Send Packet Sequence

This is shown in Figure 12-3 which shows the overall packets sequence and format for SPI command write communications. The following table details the content of these packets:

Field	Bit	Name	Description
Opcode	0-7	Opcode	Contains the 8 bit Magellan command opcode
axis	8-11	Axis	Contains the axis # that the command is directed to, must be either 0 (axis #1) or 1 (axis #2 for access to MC58113's auxiliary encoder input)
A	12	Atlas	This bit contains a 1 when the command being sent is being directed to an attached Atlas Digital Amplfier, otherwise contains a 0.
r1	13-15	Reserved	reserved, must contain 0

Wchk	0-7	Write check-sum	Contains the logical negation of an 8-bit ones complement checksum computed over all bits in the command packet except for the check-sum field, and a seed of 0xAA. If the checksum received by the MC58113 is incorrect (does not equal 0xff), the command will not be executed and will returns a checksum error code during the next data exchange.
r2	8-15	Reserved	reserved, must contain 0

The additional word writes argument1, argument2, shown in Figure 12-3 contain data (if any) associated with the command being sent to the MC58113. For example if the command **SetMotorCommand** is being sent, then a single 16-bit data word, consisting of the programmed Kp value is transmitted in argument1. Only the required number of argument data words should be sent.

12.6.2 SPI Protocol, Command Response

After receiving a command with a valid checksum and the appropriate number of argument words the MC58113 will process the command. After command processing is complete, the MC58113 will drive the *HostSPIStatus* signal high. The host reads this signal state change, and then initiates an SPI exchange to retrieve the command status as well as any return data words that may have been associated with the command that was sent by the host.

Note that the host should begin polling *HostSPIStatus* only after de-asserting *HostSPIEnable*. Polling earlier may result in incorrect interpretation of the *HostSPIStatus* signal.

Figure 12-4: SPI Response Packet Sequence

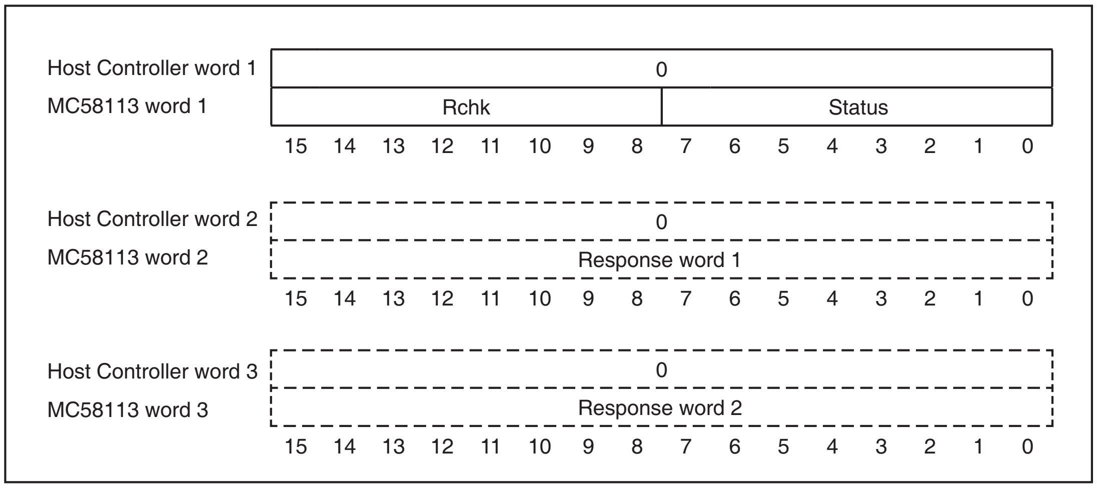


Figure 12-4 shows the packet format to receive a command response. In all cases the host sends 16-bit data words consisting of zeroes, while simultaneously retrieving data from the MC58113. The following table describes the fields of this returned data:

Field	Bit	Name	Description
Status	0-7	Command Sta-tus	This fields contains an 8-bit word which provides a status of the pro-cessed command. Positive values indicate that command processing occurred normally with the returned value holding the number of 16-bit words to be returned. Negative values contain an error code, and indicate that command processing did not occur normally.
Rchk	8-15	Response checksum	Contains the logical negation of an 8-bit ones complement checksum computed over all bits of the return packet except the checksum, and a seed of 0xAA. It is not required, although highly recommended, that the host confirm that the return checksum is correct (totals to 0xFF).

The additional word reads response1, response2, shown in Figure 12-4 contain data (if any) associated with the command that was sent to the MC58113. For example if the command GetMotorCommand was sent, then a single 16-bit data word, consisting of the current value of the MC58113's motor command register, will be returned.

If the host attempts to retrieve command status before it is ready an error will occur. The signal *HostSPIStatus* should always be used to synchronize command status checks.

Even if no data is being returned by the MC58113, the host must retrieve the command response. Failure to do so will result in SPI communciations becoming desynchronized.

12.6.3 Configuring Host SPI

There are no user-configurable parameters that need to be set for the Host SPI interface. For more information on HOSTSPI interface timing and signalling refer to the *MC58113 Electrical Specifications.*

This page intentionally left blank.

13. Brushless DC Motor Control

In This Chapter

- Overview
- Number of Phases
- Phasing Control Modes
- Phase Counts
- Phase Initialization
- Phase Initialization Programming
- Automatic Phase Correction
- Encoder Prescaler
- Sinusoidal Commutation
- Field Oriented Control

13.1 Overview

Magellan Motion Control ICs provide a number of special features for support of brushless motors. These include input of Hall sensors, support for sinusoidal commutation, and support for 2- or 3-phase motors. Magellan ICs that provide current control or Atlas-connected Magellan axes also provide additional features including field oriented control (FOC).

There are several approaches used to generate the correct phasing and motor excitation signal to drive brushless DC motors. The first uses Hall sensors to generate the desired output waveforms. The second uses the encoder, and utilizes sinusoidal or field oriented control to drive the phases. A "third" option for driving a brushless DC motor is using an amplifier that provides its own commutation. This effectively converts the controller into a single-phase DC brush motor type.

The following table shows the control modes available for various Magellan ICs.

Magellan Product	Control Modes Available	Current Control Available
MC58000	Hall-based commutation Sinusoidal commutation	No
ION, MC58113, or Atlas-connected Magellan axes	Hall-based commutation Sinusoidal commutation Field oriented control	Yes

Broadly speaking, the control tasks that are specific to multi-phase motors can be broken down into an initialization phase, when the exact correct phase angle may not yet be known, and motor operation, when phase initialization is complete and the motor is rotating and operating normally. For both of these modes, before correct phasing can occur, a number of parameters must be setup correctly. The following sections describe these parameters.

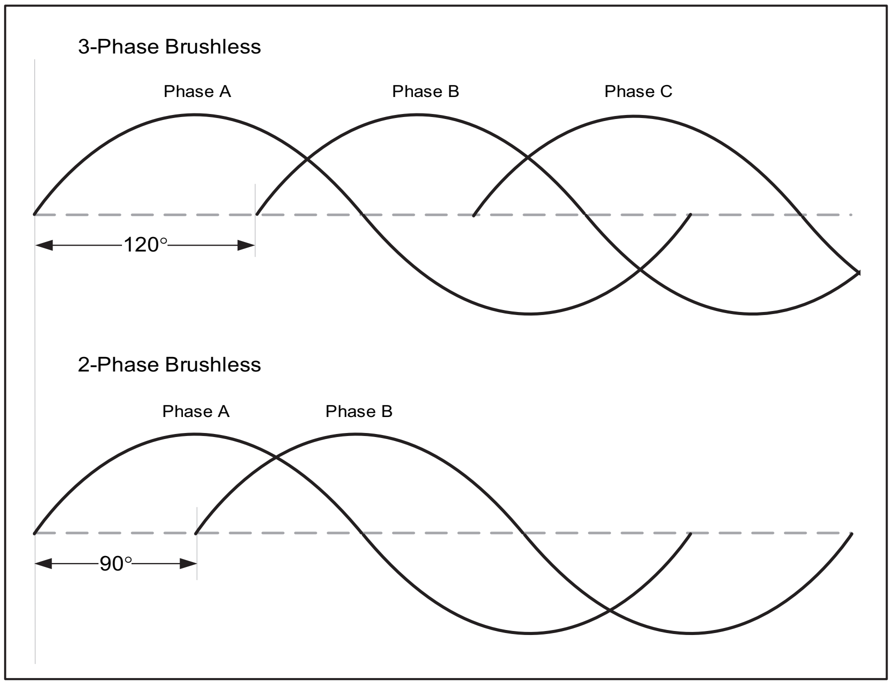
13.2 Number of Phases

The MC58000 supports two commutation waveforms: a 120-degree offset waveform appropriate for 3-phase brushless motors, and a 90-degree offset waveform appropriate for 2-phase brushless motors.

To specify the waveform, the command SetMotorType is used. This command is not required for ION, which defaults to the specific motor and waveform type that it supports.

Figure 13-1 illustrates the phase A, B, and C commutation signals for a 3-phase brushless motor, and the phase A and phase B signals for a 2-phase brushless motor.

Figure 13-1: Commutation Waveforms



13.3 Phasing Control Modes

To drive a brushless DC motor correctly, the motor's rotor angle must be known as it continually changes. This is accomplished using one of two methods. The first is by using Hall sensors, and the second is by using a position encoder. In both cases these sensors must be directly connected to the motor shaft. Generally speaking, if an encoder is available it should be used, as it will provide smoother motion and higher overall performance than Hall sensors.

To select whether the phasing of the motor will be Hall-based or encoder-based, the command SetCommutationMode is used. The value set can be read back using the command GetCommutationMode.

For ION or Atlas-connected axes, if field oriented control is desired, then the command SetCurrentControlMode is used with a value of FOC. The value set may be read back using the command GetCurrentControlMode. Selection of field oriented control does not affect selection of Hall-based or encoder-based phasing, although if an encoder is available, it is recommended that it be used rather than Hall sensors.

Note that frequently *both* Hall sensors and encoder feedback signals are used. The Hall sensors are used during phase initialization, and the encoder is used thereafter to determine correct waveform phasing during regular motor operation. See Section 13.5, "Phase Initialization" for more information on phase initialization.

13.4 Phase Counts

If the motor phasing will be determined by an encoder, the number of encoder counts per electrical cycle must be specified. This parameter indicates to the motion control IC the number of encoder counts required to complete a single full electrical rotation cycle. It is thus a way of indicating the relationship between mechanical motor rotation and electrical waveform generation.

To determine this value, the number of electrical cycles of the motor and the number of encoder counts per motor revolution must be known. The number of encoder counts per electrical cycle is then determined using the following equation:

Counts per cycle = Counts_per_rot/electrical cycles

where

Counts_per_rot is the number of encoder counts per motor rotation

electrical cycles is the number of motor electrical cycles

The number of electrical cycles can usually be determined by examining the motor manufacturer's specification. The number of electrical cycles is exactly half the number of poles. Note: Care should be taken not to confuse poles with pole pairs. For example, if a motor is documented as having 1,024 encoder counts per rotation and 4 poles, then it has a Counts per cycle value of:

Counts per cycle = 1,024 counts_per_rot/2 electrical cycles per rotation

Counts per cycle = 512

The command used to set the number of encoder counts per electrical cycle is SetPhaseCounts. To read this value, use the command GetPhaseCounts.

13.5 Phase Initialization

In addition to specifying the counts per electrical cycle, if the encoder will be used for motor phasing, then the motion control IC must determine the proper initial phase angle of the motor relative to the encoder position. This information is determined using a procedure called phase initialization. Note that a phase initialization procedure is not necessary if Hall-based commutation is selected.

The Magellan ICs provide three methods to perform phase initialization: Hall sensor-based, algorithmic, and direct-set.

13.5.1 Hall-Based Phase Initialization

The most common, and the simplest, method of phase initialization is Hall-based. To set the motion control IC for Hall-based initialization, use the command SetPhaseInitializeMode and specify Hall-based as the parameter.

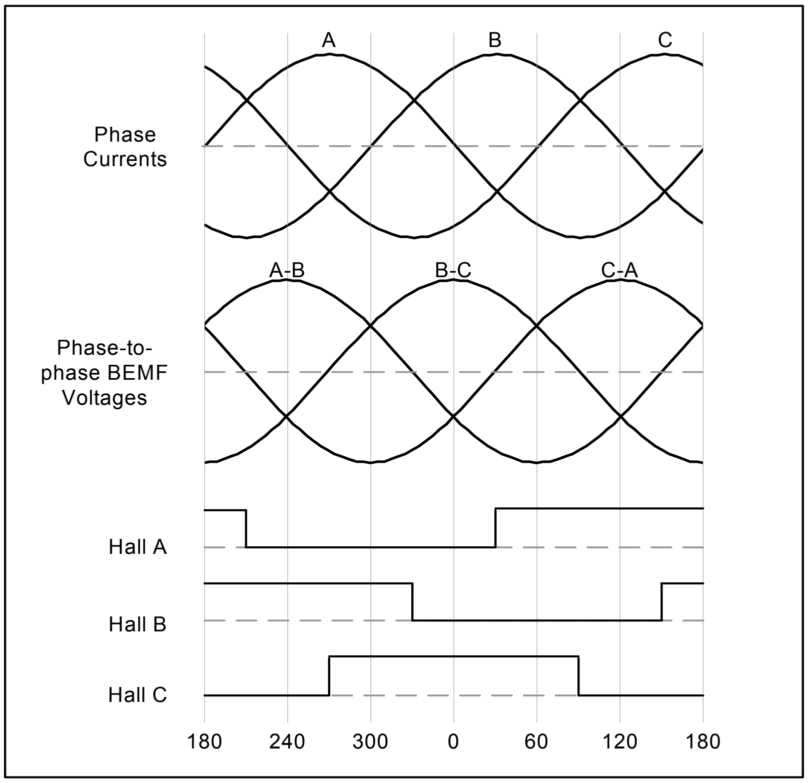
In this mode, three Hall sensor signals are used to determine the motor phasing. Sinusoidal commutation begins automatically after the motor has moved through a Hall state transition.

The Hall sensor signals are fed back into the motion control IC through the signals Hall1A-C (axis #1) and Hall2A-C (axis #2), etc. Care should be taken to connect these sensors properly. To read the current status of the Hall sensors, use the command GetSignalStatus.

Figure 13-2 illustrates the relationship between the state of the three Hall sensor inputs, the sinusoidally commutated phase current commands, and the motor phase-to-phase back EMF waveforms during forward motion of the motor. The motion control IC expects 120-degree separation between Hall signal transitions. To commutate using Hall sensors located 60 degrees apart, swap and invert the appropriate Hall signals and motor phases to generate the expected Hall states. This Hall to BEMF phasing diagram is the most common way of specifying the required alignment and a similar diagram is typically provided by the motor supplier.

With Hall-based phase initialization, no special motor setup procedures are required. Initialization is performed using the command InitializePhase, and occurs immediately, without any motor motion.

Figure 13-2: Hall-Based Phase Initialization



To accommodate varying types of Hall sensors, or sensors containing inverter circuitry, the signal level/logic interpretation of the Hall sensor input signals may be set through the host. The command SetSignalSense accepts a bit-programmed word that controls whether the incoming Hall signals are interpreted as active high or active low. To read this Hall interpretation value, use the command GetSignalSense. For details on the programming of this control word, see the *Magellan Motion Control IC Programmer's Command Reference.*

13.5.2 Algorithmic Phase Initialization

To set the motion control IC for algorithmic initialization, use the command SetPhaseInitializeMode and specify Algorithmic as the parameter.

In the algorithmic initialization mode, no additional motor sensors beyond the position encoder are required. To determine the phasing, the motion control IC performs a sequence that briefly stimulates the motor windings and sets the initial phasing using the observed motor response. From the resulting motion, the motion control IC can automatically determine the correct motor phasing.

Depending on the size and speed of the motor, the time between the start of motor phasing and the motor's coming to a complete rest (settling time) will vary. To accommodate these differences, the amount of time to wait for the motor to settle is programmable using the command SetPhaseInitializeTime. To read this value, use the command GetPhaseInitializeTime.

To minimize the impact on the system mechanics, this method utilizes a motor command value set by the host processor to determine the overall amount of power to introduce into the motor during phase initialization. Typically, the amount of power should be in the range of 5–25% of full-scale output, yet should be at least three times the breakaway starting friction. For best results, the initialization motor command value can be determined experimentally. The command used to set the motor output level is SetMotorCommand. To read this value, use the command GetMotorCommand.

To execute the initialization procedure, the host command InitializePhase is used. Upon executing this command, the phasing procedure will immediately be executed.

Before the phase initialization command is given (InitializePhase command), the trajectory generator and position loop must be disabled (SetOperatingMode command), a motor command output must be specified (SetMotorCommand command), and an initialization duration must be specified (SetPhaseInitializeTime command).

During algorithmic phase initialization, the motor may suddenly move in either direction. Proper safety precautions should be taken to prevent damage from this movement. To provide accurate results, motor movement must be unobstructed in both directions and the motor must not experience excessive starting friction.

13.5.3 Direct-Set Phase Initialization

If, after power-up, the location of the motor phasing is known, the phase angle can be directly set using the SetPhaseAngle command.

This typically occurs when sensors such as resolvers are used where the returned motor position information is absolute in nature (not incremental), and can then be used to generate a quadrature data stream, which is directly read by the host.

13.6 Phase Initialization Programming

The following examples illustrate typical host command sequences to initialize the commutation of a brushless motor for all three initialization methods.

13.6.1 Hall-Based Initialization Sequence

```
SetMotorType xx                 // Set the motor type to 3 or 2-phase brushless
SetOutputMode m                 // Set the motor output mode.
SetPhaseCounts uuuu             // Set number of encoder counts per electrical cycle.
SetSignalSense vvvv             // Set Hall sensor signal interpretation.
SetPhaseInitializeMode Hall     // Set phase initialization method to Hall based.
InitializePhase                 // Perform the initialization.
```

This sequence will cause the motion control IC to read the Hall sensor signals and initialize the phasing immediately. The motor will not move as a result of this sequence, and no delay is required for performing further motor operations.

13.6.2 Algorithmic Initialization Sequence

```
SetMotorType xx                      // Set the motor type to 3 or 2-phase brushless
SetOutputMode m                      // Set the output mode.
SetPhaseCounts uuuu                  // Set number of encoder counts per electrical cycle.
SetPhaseInitializeMode Algorithmic   // Set phase initialization method to algorithmic.
SetOperatingMode zz                  // set operating mode to "trajectory and position loop
                                      disabled"
SetPhaseInitializeTime wwww          // Set algorithmic phase init duration.
SetMotorCommand yyyy                 // Set initialization motor command level.
InitializePhase                      // Perform the initialization.
```

This sequence will cause the motor to begin the initialization procedure immediately, which will last for "wwww" servo loops. To determine if the procedure is completed, use the command GetActivityStatus. The phase initialization bit will indicate completion of the procedure. After the initialization procedure is complete, the position loop should be enabled using SetOperationMode if the motion control IC is to be run in closed-loop mode.

13.6.3 Direct-Set Initialization Sequence

```
SetMotorType xx          // Set the motor type to 3 or 2-phase brushless.
SetOutputMode m          // Set the output mode.
SetPhaseCounts xxxx      // Set the number of encoder counts per electrical cycle (hex).
SetPhaseAngle yyyy       // Set phase angle based on information from external sensor.
```

This sequence will directly set the phase angle to a value determined by another sensor. The set value must be between 0 and the number of encoder counts per electrical cycle.

13.7 Automatic Phase Correction

To enhance long-term commutation reliability, the Magellan Motion Control ICs provide the ability to utilize an index pulse input from the motor encoder or Hall sensor inputs as a reference point during commutation to perform automatic commutation phase correction. Automatic phase correction compensates for any long-term loss of encoder counts that might otherwise affect the accuracy of the commutation.

This feature is used with encoder-based commutation only. If Hall-based commutation is used, this feature is not necessary. To utilize automatic phase correction the motor encoder must provide an index pulse signal once per rotation or Hall sensor inputs (MC58113 only).

Automatic phase correction is recommended for all rotary brushless motors with quadrature encoders on the motor shaft. For linear brushless motors, although it may be used, automatic phase correction is less common.

The number of encoder counts per electrical cycle is not required to be an exact integer. In the case that this value is not an integer, the nearest integer should be specified for the value of SetPhaseCounts. Conversely, if automatic phase correction is not being used, then the number of counts per electrical cycle must be an exact integer with no remainder.

For example, if a 6-pole brushless motor is to be used, an encoder with 1,200 counts per rotation would be an appropriate choice, but an encoder with 1024 would not, because 1,024 cannot be evenly divided by 3.

The command SetPhaseCorrectionMode is used to specify index pulse phase correction, Hall-based correction or none.

Once selected, automated phase correction is performed by the motion control IC regardless of the initialization scheme (algorithmic, Hall-based, microstepping, or direct set).

13.7.1 Index Signal-Based Phase Correction

With an index signal properly installed the motion control IC will automatically correct any small losses of encoder counts that may occur.

If the index pulse does not arrive at the expected location within the commutation cycle, a commutation error occurs and bit (11) in the Event Status register will be set. Commutation errors occur when the required correction is greater than (PhaseCounts/128)+4.

Commutation errors are caused by a number of circumstances. The most common are:

- noise on the A or B encoder lines.
- noise on the index line.
- incorrect setting of encoder counts per electrical cycle.

For each commutation error occurrence, phase referencing will not occur for that index pulse. Depending on the cause of the error, the commutation error may be a one-time event, or it may occur continuously after the first event. To recover from a phasing error, bit 11 of the Event Status register is cleared by the host.

A phasing error may indicate a serious problem with the motion system, potentially resulting in unsafe motion. It is the responsibility of the host to determine and correct the cause of commutation errors.

13.7.2 Hall-Based Phase Correction (MC58113-Only)

Hall-based phase error detection functions similarly to index-based phasing. Bit 11 of the Event Status register signals an error and recovery occurs in the same manner. The main difference is that the absolute amount of error allowed before a commutation error occurs, normally Phase Counts/128 +4, is not fixed and increases as rotation speed increases.

13.7.3 Adjusting the Phase Angle

Magellan supports the ability to change the motor's phase angle directly, both when the motor is stationary and when it is in motion. Although this is not generally required, it can be useful during testing, or during phase initialization when the microstepping or direct-set methods are used.

To change the phasing angle when the motor is stationary, use the command **SetPhaseAngle**. To change the phasing angle while the motor is moving, the index pulse is required; and a different command, **SetPhaseOffset**, is used. **SetPhaseOffset** takes effect only when an index pulse occurs.

After phase initialization has occurred, the phase angle of the index pulse is stored in the phase offset register. This 16-bit offset register can be read using the command **GetPhaseOffset**.

For a given motor, the index pulse may be located anywhere within the commutation cycle, since it will usually vary in position from motor to motor. Only motors that have been mechanically assembled so that the index position is referenced to the motor windings will have a consistent index position relative to the commutation zero location.

Before phase initialization has occurred, the Phase Offset register will have a value of FFFFh. Once phase initialization has occurred and the motor has been rotated so that at least one index pulse has been received, the phase offset value will be stored as a positive number with a value between 0 and the number of encoder counts per electrical cycle.

Note that when an axis is in dual encoder loop mode with an auxiliary axis, the SetPhaseAngle, SetPhaseOffset, and SetPhaseCounts commands must be directed towards the main axis.

To convert the phase offset value, which is in encoder counts, to degrees, the following formula is used:

$$\text{Offset}_{\text{degrees}} = 360 * \text{Offset}_{\text{counts}}/\text{counts\_per\_cycle}$$

where

$\text{Offset}_{\text{degrees}}$ is the phase offset in degrees

$\text{Offset}_{\text{counts}}$ is the phase offset in encoder counts

counts_per_cycle is the # of counts per electrical cycle set using the **SetPhaseCounts** command

The phase offset value may also be changed any number of times while the motor is in motion. The changes that are made should be small; this will prevent sudden jumps in the motor motion.

The **SetPhaseOffset** and **GetPhaseOffset** commands may only be used when an index pulse from the encoder is connected. If no index pulse is used, the phase offset angle cannot be adjusted or read by the host.

Setting the phase offset value does not change the relative phasing of phases B and C to phase A. These phases are still set at either 90- or 120-degree offsets from phase A, depending on the selected waveform.

13.8 Encoder Prescaler

Particularly when used with linear motors, the range in the value of the number of encoder counts per electrical cycle can vary widely. Typical rotary motors can have a value between 1 and 32,767. Linear brushless motors can have values of 1,000,000 counts per cycle (or higher), because they often use high-accuracy laser-based encoders.

To accommodate this large range, the MC58000 series and ION supports a prescaler function which, for the purposes of commutation calculations, divides the incoming encoder counts by 64, 128, or 256. With the prescaler enabled, the maximum range for the number of encoder counts per electrical cycle is 8,388,352.

To enable the prescaler, use the command **SetPhasePrescale**.

The prescaler function only affects the commutation of the motion control IC. It does not affect the position used during servo filtering or requested by the command **GetActualPosition**.

The prescaler function should not be enabled or disabled once the motor has been set in motion.

13.9 Sinusoidal Commutation

Figure 13-3 provides an overview of the motion control IC control sequence when sinusoidal commutation is selected. To select sinusoidal commutation, encoder-based phasing should be selected using the **SetCommutationMode** command, and field oriented control should not be selected. For ION and Atlas-connected axes, the command **SetCurrentControlMode** should be used to deselect FOC mode.

Relative to Hall-based commutation, sinusoidal commutation provides improved smoothness and improved positioning stability due to lack of commutation torque discontinuities.

The sinusoidal commutation logic of the motion control IC uses as its input the motor command signal from either the position servo loop or the Motor Command register (depending on whether the position loop module is enabled or disabled). This pre-commutated command signal is then multiplied by commutation values derived from an internal lookup Sin/Cos table.

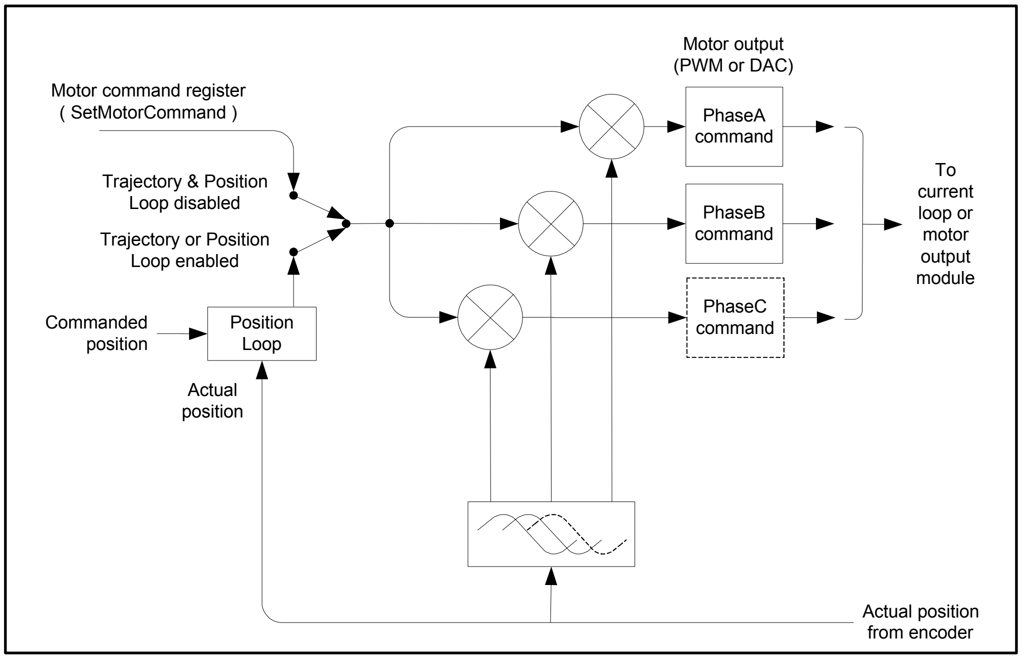

Motor command register
(SetMotorCommand)
Trajectory & Position Loop disabled
Trajectory or Position Loop enabled
Commanded position
Position Loop
Actual position
Motor output (PWM or DAC)
PhaseA command
PhaseB command
PhaseC command
To current loop or motor output module
Actual position from encoder

Figure 13-3: Sinusoidal Commutation

The commutation angle used in the Sin/Cos lookup is determined by the position encoder, as well as parameters set by the host processor that relate the specific encoder to the motor magnetic poles, such as counts per electrical cycle.

Two commutation waveforms are provided: one appropriate for 3-phase devices with 120-degree separation between phases (such as brushless motors), and one appropriate for 2-phase devices with 90-degree separation between phases (such as step motors). Some motion control ICs support both of these waveforms, while others, such as ION, MC58113, or Atlas-connected Magellan axes, support only 3-phase waveforms. See Section 1.1, "Family Summary" for more information.

Once commutated, the individual motor commands for the A, B, and C phases (3-phase motor) are output either directly to the amplifier or to the current control module. If output to the motor, they are converted to one of the hardware output formats such as PWM or DAC output. See Chapter 11, "Motor Output" for details. If output to the current control module (ION, MC58113, and Atlas-connected axes only), then additional calculations are performed using the measured current through each winding to determine a final amplifier command for each winding. See Section 15.1, "Current Loop" for details.

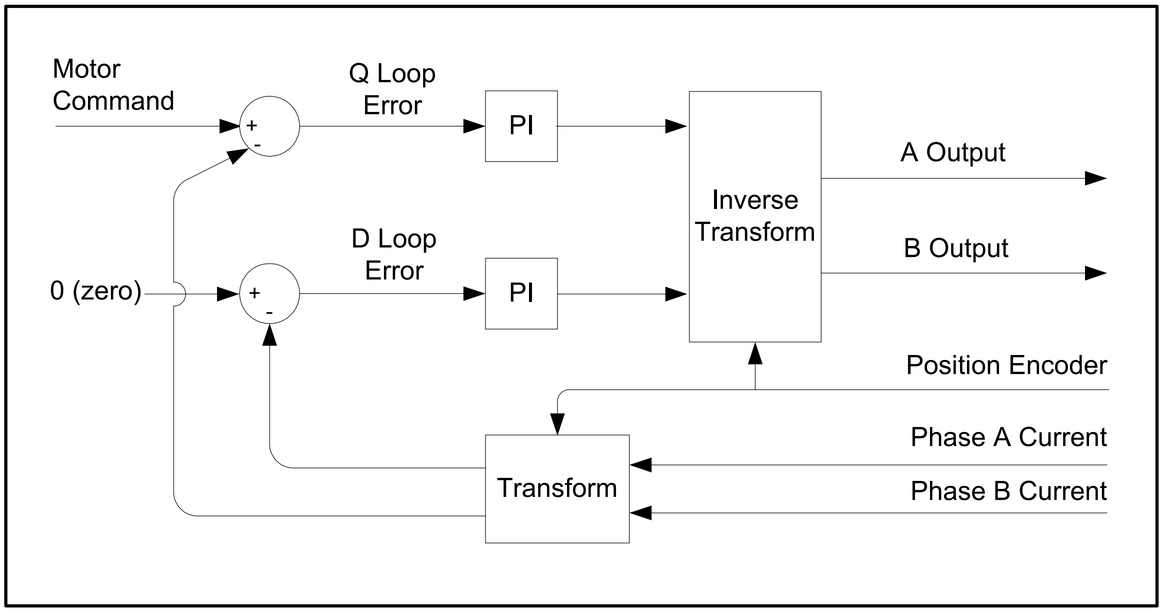
To read these individual phase commands, the command GetPhaseCommand is used.

13.10 Field Oriented Control

(ION, MC58113, and Atlas-connected axes only)

Figure 13-4 provides an overview of an alternate method for determining the commands for each motor coil, known as field oriented control (FOC). This technique is similar to sinusoidal commutation, but combines digital current control with phase calculation, whereas sinusoidal commutation separates these two operations. FOC is the recommended control approach when current control is available. Note that ION, MC58113, and Atlas-connected axes also support sinusoidal commutation, should this control method be preferred for a particular application.

Figure 13-4: Control Flow Of FOC Control



Compared to sinusoidal commutation, FOC uses an entirely different algorithmic method to determine the phase command for each motor winding. Instead of separating the phase lookup and current control operations as sinusoidal commutation does, it combines them and "re-references" them to what are known as D (direct torque) and Q (quadrature torque) reference frames. This difference in approach provides FOC with performance advantages over sinusoidal commutation at high motor speeds. At lower speeds there is very little performance difference between these two techniques.

Like sinusoidal control, FOC uses as its command input the motor command signal from either the position servo loop or the motor command register (depending on whether the position loop and trajectory modules are enabled or disabled). In addition to the motor command, however, FOC utilizes analog input signals to determine the instantaneous current flow through two of the three motor coils and combines this with the motor's rotor position to determine exact output commands for each motor coil.

These calculations and subsequent update of the phase commands are performed at 20 kHz, regardless of the motion control IC's PWM output rate, which can be either 20 kHz, 40 kHz, or 80 kHz in the case of the MC58113. See Section 11.5, "Setting PWM Frequency" for details on setting the PWM rate.

To enable field oriented control mode, the command SetCurrentControlMode should be used with an argument of FOC. The value set using this command can be read back using GetCurrentControlMode.

13.10.1 FOC Algorithm

Figure 13-5 details the algorithmic flow of the FOC controller. For each current loop (D & Q), three parameters are set by the user, Kp, Ki, and Ilimit. Two of these are gain factors for the PI (proportional, integral) controller that comprises the heart of the FOC controller, and the other is a limit for the integral contribution. These three parameters have the following ranges and formats.

Term	Name	Representation & Range
KpD, KpQ	D, Q proportional gain	unsigned 16 bits (0 to 32,767)
KiD, KiQ	D, Q integral gain	unsigned 16 bits (0 to 32,767)
IlimitD, IlimitQ	D, Q integration limit	unsigned 32 bits (0 to 2,147,483,647)

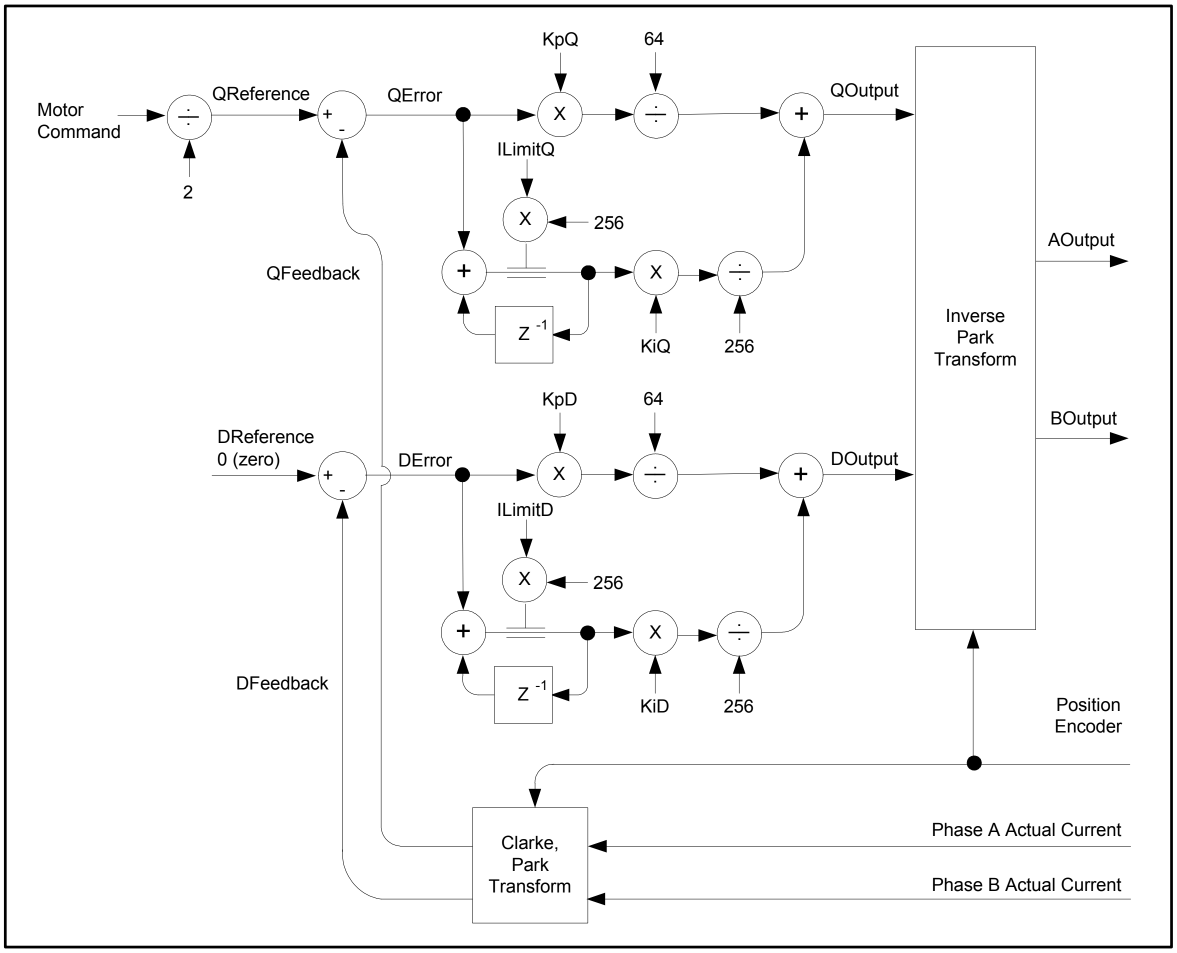


Figure 13-5: Algorithmic Flow Of FOC Controller

To set any of these parameters, the command SetFOC is used. To read back these parameters, the command GetFOC is used. The values set using this command are buffered and may be activated using the Update command. See Section 6.1, "Parameter Buffering" for details.

Determining correct parameters for the FOC controller gains can be done in a number of ways. The easiest is to utilize the auto-tuning facility provided within PMD's Pro-Motion software package. Parameters derived using this procedure may or may not be optimized for your system.

Please note that it is the responsibility of the user to determine the suitability of all parameters, including those determined by auto-tuning, for use in a given application.

Another method is through trial and error using the Magellan's built-in trace facility. Finally, it is possible to model your system and determine the best settings through simulation or analysis; however, a discussion of this approach is beyond the scope of this manual.

13.10.2 Reading FOC Loop Values

To facilitate tuning, there are a number of internal FOC loop values that can be read back as well as traced. To read back these values the command GetFOCValue is used. To specify these values for trace during automatic trace capture see the *Magellan Motion Control IC Programmer's Command Reference.*

Refer to the diagram in Section 13.10.1, "FOC Algorithm" for an overview of the FOC loop. The variables within the FOC loop that can be read or traced are summarized as follows:

Variable Name	Function
Q Reference, D Reference	Are the commanded values input into the Q and D loops. Note that D is always set to 0 (zero).
Q Feedback, D Feedback	Are the measured values for the Q (quadrature) and D (direct) force after re-referencing from the actual measured current in the phase A, phase B coils.
Q Error, D Error	Are the differences, for the Q loop and the D loop, between the loop reference and the loop measured value.
Q Integrator Sum, D Integrator Sum	Are the integrator sums for the D and Q loops.
Q Integral Contribution, D Integral Contribution	Are the contributions of the integral to the overall PI sum for the Q and D loop.
Q Output, D Output	Are the output commands of the Q and the D loops.
FOC A Output, FOC B Output	Are the phase A and phase B coil commands before output to the motor output module and PWM generator.
Phase A Actual Current, Phase B Actual Current	Are the measured currents for the phase A and phase B coils.

13.10.3 FOC With 2-Phase Step Motors

FOC on IONs, MC58113, and Atlas-connected Magellan axes is also designed to work with 2-phase step motors. When operating the 2-phase step motor in this mode (see Chapter 14, "Step Motor Control" for more information on Magellan operations with step motors), the basic method is identical. The same three FOC parameters described in Section 13.10.1, "FOC Algorithm" are set, and the readable parameters are also the same.

13.10.4 Hall-Based FOC

FOC on IONs, MC58113, and Atlas-connected Magellan axes can operate with both Hall-based motor position sensing and encoder-based motor position sensing. The former would only be used in the case that no encoder is available, as use of an encoder substantially improves smoothness and performance. To set the FOC controller for use with Hall sensors, use the command SetCommutationMode. The value set can be read back using the command GetCommutationMode.

14. Step Motor Control

In This Chapter

- Overview
- Encoder Feedback
- Stall Detection
- Pulse & Direction Motor Control
- Microstepping Motor Control

14.1 Overview

Magellan Motion Control ICs provide a number of special features for support of step motors. These include pulse and direction output, microstep signal generation, and a programmable holding current. Broadly speaking, two types of step motors are supported, pulse and direction motors and microstepping motors. Pulse and direction motors are step motors that are driven by an amplifier that accepts pulse and direction input signals. Microstepping motors are motors that are connected to an amplifier that controls the current through each phase explicitly, much like the way a brushless DC motor is controlled.

Overall, the control features of the Magellan when used with a step motor are similar to that used with servo motors. In particular trajectory generation, breakpoints, trace, and a number of other features are entirely unaffected by choice of motor type. The primary differences between servo motors and step motors however is that there is no position loop module used for step motors and that motor output signal generation differs from that of servo motors in some respects.

There are also a number of other features that are similar in concept, but different in implementation between servo motors and step motors. Motion error, which is equivalent to stall detection for step motors, is an example of this.

All of these differences will be explained in the upcoming sections of this chapter.

MC58000 Motion Control IC users are able to select between pulse and direction and microstepping motor modes. This choice will affect amplifiers interfacing and a number of other functions. MC55000 ICs are dedicated to pulse and direction output, and external amplifiers should be chosen accordingly. ION uses a microstepping motor mode. However, since the amplifier is located internally, there are no external amplifier choice issues.

When used with Atlas amplifiers, Magellan can drive step motors either in pulse and direction mode or via the SPI Atlas bus. Between these two modes SPI Atlas is recommended because the SPI bus provides a much greater level of information flow including safety-related information and the ability to select a wide range of control features. When used in pulse & direction mode the Atlas must first be configured by the user via SPI Atlas mode and then switched to pulse & direction signal mode. Once in pulse & direction mode, SPI communication is not available.

14.1.1 Trajectory Control Units

For servo motors, the units for measuring position are encoder steps, and for time they are cycles. For step motors, position units are measured as either steps or microsteps, depending on whether the motor type is pulse and direction step motor, or microstepping motor. The following table lists various commands and their corresponding units.

Command	Servo axes	Microstepping axes	Pulse & Direction axes
Set/GetPosition	counts	microsteps	steps
Set/GetVelocity	counts/cycle	microsteps/cycle	steps/cycle
Set/GetAcceleration	counts/cycle^2	microsteps/cycle^2	steps/cycle^2
Set/GetDeceleration	counts/cycle^2	microsteps/cycle^2	steps/cycle^2
Set/GetJerk	counts/cycle^3	microsteps/cycle^3	steps/cycle^3
Set/GetStartVelocity	-	microsteps/cycle	steps/cycle
GetCommandedPosition	counts	microsteps	steps
GetCommandedVelocity	counts/cycle	microsteps/cycle	steps/cycle
GetCommandedAcceleration	counts/cycle^2	microsteps/cycle^2	steps/cycle^2
Set/GetPositionErrorLimit	counts	microsteps	steps
GetPositionError	counts	microsteps	steps

14.2 Encoder Feedback

For MC50000, each step motor axis supports position feedback in one of the two standard ways, either by incremental encoder input, or parallel-word input. See Chapter 10, "Encoder Interfacing" for more information on position feedback. The command **SetEncoderSource** selects the type of position feedback. If no position feedback is used (something that is not unusual for step motors), then a value of none should be entered using **SetEncoderSource**. Doing so will allow the position feedback value to be ignored, thereby disabling stall detection, a feature that will be discussed in an upcoming section.

ION can input position using incremental encoder information only. See the *ION Digital Drive User's Manual* for details on interfacing to incremental quadrature encoders.

Regardless of the input method, most encoder commands operate as for servo motors. For example the current position is retrieved using the command **GetActualPosition**, the position capture location is retrieved using **GetCaptureValue**, and the **AdjustActualPosition** and **SetActualPosition** commands may be used to alter the current position. The default units of this command are encoder counts. To simplify program design and debugging, actual position units can be changed to steps/microsteps. This is done using the command **SetActualPositionUnits**. The following table lists the affected commands.

Command	Position Units = counts	Position Units = steps
Set/GetActualPosition	counts	steps/microsteps
AdjustActualPosition	counts	steps/microsteps
GetCaptureValue	counts	steps/microsteps

The **SetActualPositionUnits** command also affects the units of the trace variable Actual Position.

In many step motor systems, the ratio of steps to encoder counts is not necessarily exactly one. Magellan accommodates this by allowing the ratio of encoder counts to steps to be explicitly specified using the command **SetEncoderToStepRatio**. This value can be read back using the command **GetEncoderToStepRatio**.

If the units are set to counts, then the actual position commands are referenced to the encoder. If the units are set to steps, then the encoder input is converted to steps using the value specified by **GetEncodertoStepRatio** command.

See Chapter 10, "Encoder Interfacing"for additional information on interfacing to encoders.

14.3 Stall Detection

In addition to passively returning the position to the host with the GetActualPosition command, Magellan Motion Control ICs can actively monitor the target and actual position, and detect a motion error that results in a stall condition. Automatic stall detection allows the motion control IC to detect when the step motor has lost steps during motion. This typically occurs when the motor encounters an obstruction, or otherwise exceeds its rated torque specification.

Automatic stall detection operates continuously once it is initiated. The current desired position (commanded position) is compared with the actual position (from the encoder), and if the difference between these two values exceeds a specified limit, a stall condition is detected. The user-programmed register SetPositionErrorLimit determines the threshold at which a motion error is generated.

To initiate automatic stall detection, the host must specify the number of encoder counts per output step/microstep. This is accomplished using the command SetEncoderToStepRatio. This command accepts two parameters: the first parameter is the number of encoder counts per motor rotation, and the second parameter is the number of steps/ microsteps per motor rotation.

Parameter	Format	Word size	Range
Encoder counts per rev	16.0	16-bit	0 to 32,767
Steps/microsteps per rev	16.0	16-bit	0 to 32,767

For example, if a step motor with a 1.8 degree full step size is used with an encoder with 4,000 counts per motor rotation, the parameters would be:

```
SetEncoderToStepRatio 4000 200        // where the number of steps per rotation is derived
                                      // from 360/1.8.
```

In cases where the number of steps, microsteps, or encoder counts per rotation exceeds the allowed maximum of 32,767, the parameters may be specified as fractions of a rotation, as long as the ratio is accurately maintained. In other words specifying the ratio for a fraction of a rotation has the same accuracy as specifying it for a full rotation, as long as the ratio of counts to steps is correctly specified.

Processing of a motion error while using a step motor is identical to that for servo motors. See Section 8.2, "Motion Error" for details.

14.4 Pulse & Direction Motor Control

(MC50000 only)

If the motor type is set to pulse and direction motor, then the pulse generation circuitry of the Magellan Motion Control IC will be activated, and it will be used by the motor output module to drive the external amplifier. To set the motor type the command SetMotorType is used. The value set using this command can be read using GetMotorType.

The pulse signal output by the motion control IC consists of a precisely controlled series of individual pulses; each of which represents an increment of movement. This signal is always output as a square wave pulse train. By default, a step, or pulse, is considered to have occurred when the pulse signal transitions from a low to a high output value. (While the square wave is not guaranteed to have a 50% duty cycle, the rising edges will be correctly timed.) The direction signal is synchronized with the pulse signal at the moment each pulse transition occurs. The direction signal is encoded so that a high value indicates a positive direction pulse, and a low value indicates a negative direction pulse.

The rate of pulse output is usually determined by the particular trajectory profile parameters being requested by the host processor. However the Magellan Motion Control ICs support several ranges of pulse generation, to maximize accuracy for a given speed range. The overall pulse generation range can be specified using the command SetStepRange. The following table shows the values and resultant step ranges available using this command.

Command	Frequency range of output pulses
SetStepRange 1	0 to 4.98 M steps per second
SetStepRange 4	0 to 622.5 K steps per second
SetStepRange 6	0 to 155.625 K steps per second
SetStepRange 8	0 to 38.90625 K steps per second

The ranges in the preceding table show the maximum and minimum ranges which can be generated by the motion control IC for the specified mode. For example, if the desired maximum step rate is 200 Ksteps per second, then the appropriate setting is SetStepRange 4.

For full-step and half-step applications, as well as for pulse and direction applications which will have a maximum velocity of 38 Ksteps/sec, SetStepRange 8 should be used. For applications requiring higher pulse rates, one of the higher speed ranges should be specified.

A different step range can be programmed for each axis. To read the current step range setting, use the command GetStepRange.

The maximum pulse output rate on the MC58113 is 1.0 Msteps per second. On the MC55110 and MC58110 it is 100k steps per second. On all of these devices the SetStepRange command cannot be used.

14.5 Microstepping Motor Control

If the motor type is set to microstepping motor, rather than pulse and direction signals, multi-phase output signals will be generated for each axis set to this mode. Typical step motors have two phases, but some have three. MC58000 except MC58113 supports two or three phase drive, while ION and MC58113 supports only two-phase. To set the motor type the command SetMotorType is used. The value set using this command can be read using GetMotorType.

Note that for step motor ION Digital Drives, the motor type is automatically set to 2-phase microstepping. It is not necessary to specify the motor type manually.

Figure 14-1 shows an overview of the control flow of the microstepping scheme.

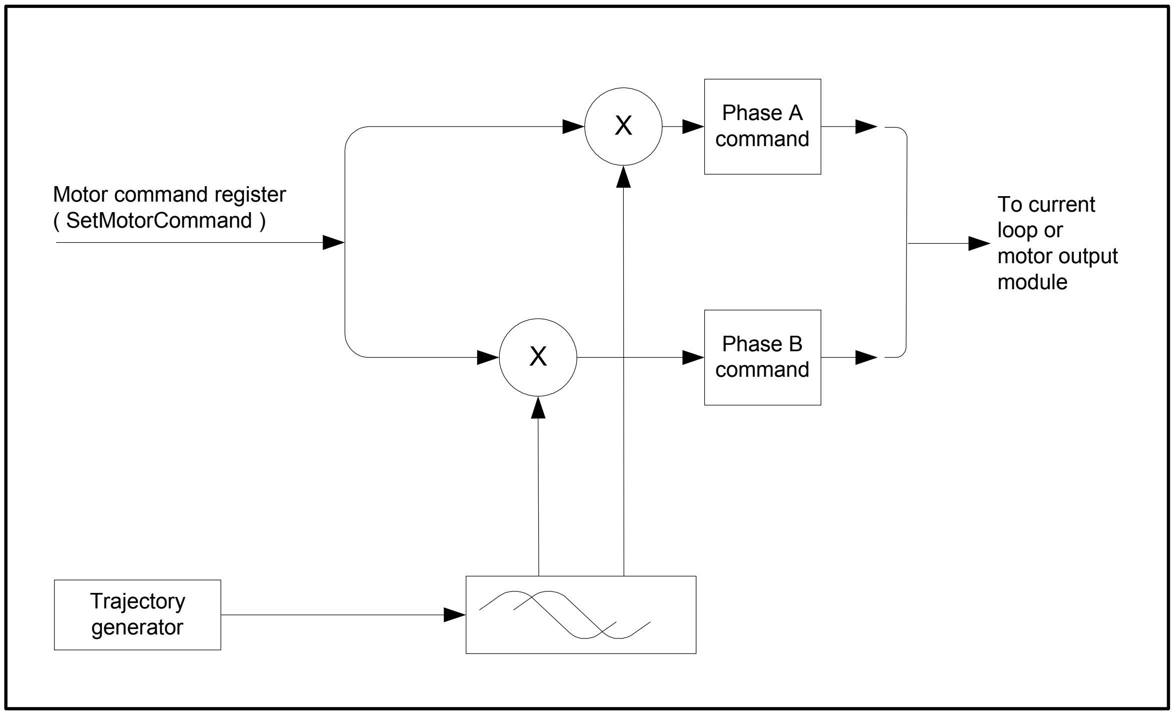


Figure 14-1: Microstepping Waveform Generation

Similar to sinusoidal commutation, the microstepping portion of the motion control IC generates a sinusoidal waveform with a number of distinct output values per full step (one full step is one quarter of an electrical cycle). The number of microsteps per full step is set using the command **SetPhaseCounts**. The parameter used for this command represents the number of microsteps per electrical cycle (four times the desired number of microsteps). For example, to set 64 microsteps per full step, the command **SetPhaseCounts 256** should be used. The maximum number of microsteps that can be generated per full step is 256, resulting in a maximum parameter for this command of 1024.

The output frequency of the microstepping signals are controlled by the trajectory generator, while the Motor Command register controls the amplitude of the microstepping signals. To set this register use the command **SetMotorCommand**. A value between 0 and 32,767 is set, representing an amplitude of zero to 100 percent. Since **SetMotorCommand** is double buffered, it requires an **Update** or a breakpoint to occur before it takes effect. This feature can be advantageous when the motor power changes are to be synchronized with other profile changes, such as at the start or the end of a move.

As described in a subsequent section, a special holding command limit can also be defined to allow different output levels for active and non-active operational modes of the motor. This is useful for reducing heat output while the motor is not moving.

Step motor Atlas amplifiers that are connected to the Magellan by the SPI Atlas bus provide microstepping control of the step motor. Two phase step motors only are supported, and the motor type should be specified as “step” rather than micrsotepping using the **SetMotorType** command.

14.5.1 Microstepping Waveforms

For MC58000, two microstepping motor types with associated waveforms are provided, one appropriate for traditional two-phase step motors with 90 degrees of separation between phases and one appropriate for three-phase step motors with 120 degree separation between phases. ION, MC58113, and Atlas-connected axes provide only two-phase microstep operation. To specify one of these two motor types the command **SetMotorType** is used. To read the value set using this command, use **GetMotorType**.

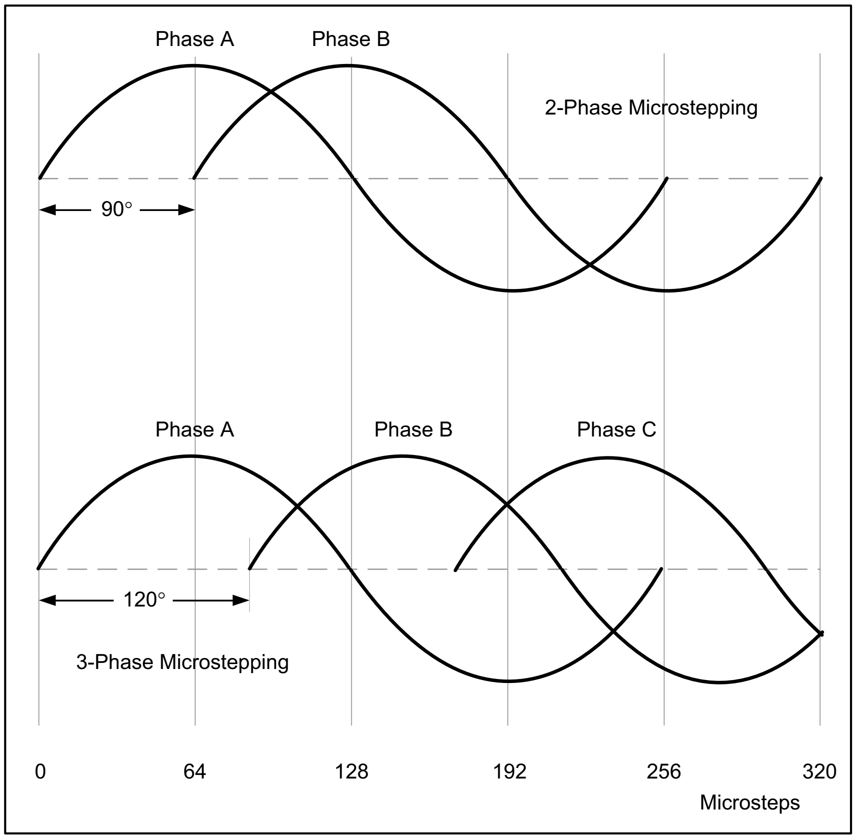
In addition, various motor output modes are available with different motor types. The following table summarizes this.

Motor type	Motor output mode	Number of phase signals & name
2-phase microstepping	PWMSign/Mag	2 (A, B)
2-phase microstepping	PWM High/Low	2 (A, B)
2-phase microstepping	DAC	2 (A, B)
3-phase microstepping	PWM50/50	3 (A, B, C)
3-phase microstepping	PWM High/Low	3 (A, B, C)
3-phase microstepping	DAC	2 (A, B)
step	SPI Atlas	2 (four signal Atlas SPI bus)

For specific pin assignments of the PWM and DAC motor output signals, see the *Magellan Motion Control IC Electrical Specifications*.

Figure 14-2 illustrates the phase A/phase B/phase C signals for a two-phase step motor, and the phase A/phase B signals for a three-phase step motor.

Figure 14-2: Microstepping Waveforms



14.5.2 Holding Current Functions

In addition to the standard pulse and direction output signals, MC50000 processors provide a signal output for each axis known as the *AtRest* signal, which indicates when the trajectory generator is in motion. This signal can be useful when interfacing with amplifiers that support a separate torque output level during motion as opposed to while it is holding (not moving).

For ION and MC58000 processors used with microstepping motor mode, a related facility exists to allow a specific holding current to be specified. Normally, the drive current used during microstepping operation motion is specified using SetMotorCommand. If desired, it is possible to have the output waveform be limited to a lower level while the motor is at rest. This current limit value can be set using the command SetHoldingCurrent. To read the value set use the command GetHoldingCurrent. Note that the value specified represents the limit of the output current while the motor is in a holding condition. For example if the current value normally output specified using SetMotorCommand is already lower than the at rest current limit, the lower motor command value will be used.

Whether represented as an *AtRest* signal, or output as a motor command reduction, the holding current condition will go active when the trajectory generator velocity is zero for a user-programmable amount of time. This parameter, if set to a non-zero value, allows a delay to be introduced between the time the trajectory finished, and the external signal goes active. Typically this is used to allow the motor to settle or come to a complete stop. This time delay can be set using the command SetHoldingCurrent. The value specified can be read back using GetHoldingCurrent.

A bit indicating whether the axis is currently in the holding condition or not is available in the Drive Status register. To read this register use the command GetDriveStatus.

14.5.3 Field Oriented Control and Current Control

For ION, MC58113, and Atlas-connected Magellan axes, current control of the step motor is achieved using either a field-oriented control technique, or a current control technique. See Section 13.10, "Field Oriented Control" for a detailed description of field-oriented control, and see Section 15.1, "Current Loop" for a detailed description of Magellan's current loop. To select field oriented control or current control the command SetCurrentControlMode is used. The value set can be read back using GetCurrentControlMode.

Once the overall current control mode has been selected, the specific loop gain and other parameters can be specified.

When the current loop is enabled, MotorCommand defines the amplitude of the phase current as a percentage of full-scale.

Determining correct parameters for the FOC or current control modules can be done in a number of ways. The easiest is to utilize the auto-tuning facility provided within PMD's Pro-Motion software package. Parameters derived using this procedure may or may not be optimized for your system.

Please note that it is the responsibility of the user to determine the suitability of all parameters, including those determined by auto-tuning, for use in a given application.

Another method is through trial and error using the Magellan's built-in trace facility. Finally, it is possible to model your system and determine the best settings through simulation or analysis; however, a discussion of this approach is beyond the scope of this manual.

This page intentionally left blank.

15. Drive Control

In This Chapter

- Current Loop
- Current Loop Parameters
- Enabling and Disabling Current Loop
- Current Feedback
- Reading Current Loop Values
- Drive Safety Feature Overview
- Overcurrent Sense
- Overtemperature Sense
- Overvoltage Sense
- Undervoltage Sense
- Current Foldback
- Drive Enable Signal
- FaultOut Signal
- Drive Fault Status Register
- Atlas-Specific Commands
- Setting Atlas Initialization Configuration

(ION, MC58113, and Atlas-connected Magellan axes only)

In addition to profiling, servo control, and other standard Magellan motion functions, ION, MC58113, and Atlas-connected axes provide digital current and drive control features. These additional capabilities comprise a complete drive capability for the Magellan IC architecture.

Digital current control is a technique used for DC brush, brushless DC, and step motors for controlling the current through each winding of the motor. By controlling the current, response times improve, motor efficiency can be increased, and motion smoothness can improve.

The digital current loop utilizes the desired current for each motor winding along with the actual measured current, which is input by analog feedback into the motion IC. This desired current and measured current are then subtracted to develop a current error, which is passed through a PI (proportional, integral) filter to generate an output voltage for each motor coil. The output command for each coil is then passed through additional motor output logic to generate the precise PWM (pulse width modulation) timing outputs signals, which are connected to external switching drives.

In addition to digital current control, ION, MC58113, and Atlas-connected axes integrate a number of drive safety features such as overcurrent sense, overtemperature sense, overvoltage sense, undervoltage sense, and others. These features will be described later in this chapter.

15.1 Current Loop

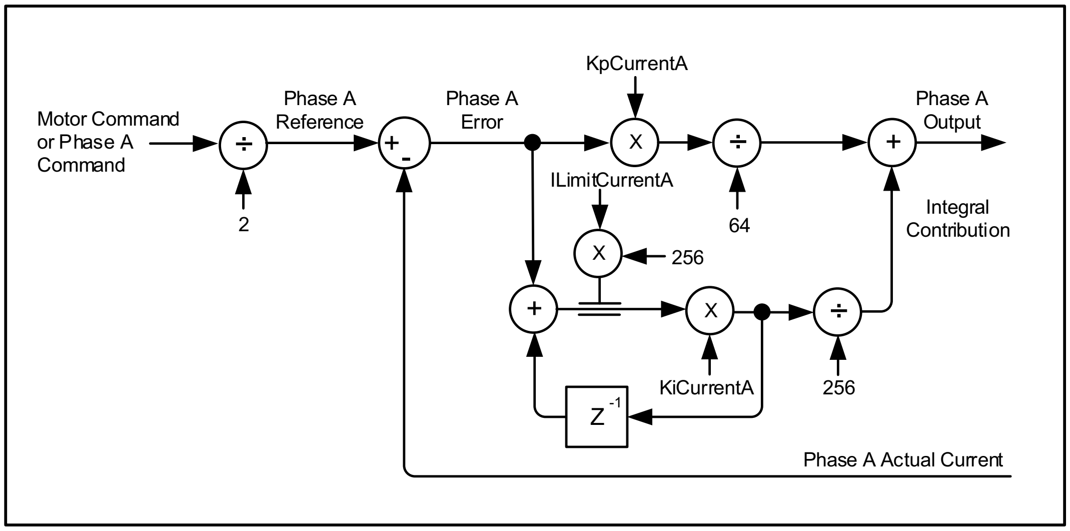
Figure 15-1 provides an overview of the current control loop for each phase of motor axis. For single-phase motors such as DC brush, one current loop per axis is used. For 3-phase brushless DC motors, two current loops are used, one for the A phase and one for the B phase, and the voltage of phase C is driven using the formula C = –(A+B), reflecting the fact that current entering any two coils must exit from the third. When driving 2-phase step motors, two current loops are used, one for the phase A coil, and the other for the phase B coil.

For safety reasons the default status of the current loop module as well as motor output module after power-on is disabled. To enable (or disable) current control as well as motor output, use the command **SetOperatingMode**.

There are two overall types of current control provided. The first is Field Oriented Control. See Section 13.10, “Field Oriented Control” for a description. The other method is referred to as “current loop” and is an alternate current control method that can be used with both single phase and multi-phase motors. See Section 15.2, “Current Loop Parameters” for more information.

DC brush motors can use current control, however they cannot use FOC. To select which type of current control will be used, use the command **SetCurrentControlMode**. To read the value set using this command, use **GetCurrentControlMode**. If FOC is selected, then Section 15.2, “Current Loop Parameters” through Section 15.5, “Reading Current Loop Values” do not apply.

Figure 15-1: Current Loop Control Flow



15.2 Current Loop Parameters

To control the current loop correctly, three parameters are set by the user, $Kp_{current}$, $Ki_{current}$, and $Ilimit_{current}$. Two of these are gain factors for the PI (proportional, integral) controller, and the other is a limit for the integral contribution. These three parameters have the following ranges and formats:

Term	Name	Representation & Range
$Kp_{current}$	Current loop proportional gain	Unsigned 16 bits (0 to 32,767)
$Ki_{current}$	Current loop integrational gain	Unsigned 16 bits (0 to 32,767)
$Ilimit_{current}$	Current loop integration limit	Unsigned 16 bits (0 to 32,767)

To set any of these three parameters, the command **SetCurrentLoop** is used. To read back these parameters, the command **GetCurrentLoop** is used. Note that for multi-phase motors, the values for the phase A and B loops can be set independently while for single-phase DC brush motors, only the phase A loop parameters are used. The values set using this command are buffered, and may be activated using the Update command. See Section 6.1, "Parameter Buffering" for details.

Determining correct parameters for the current loop controller gains can be done in a number of ways. The easiest is to utilize the auto-tuning facility provided within PMD's Pro-Motion software package. Parameters derived using this procedure may or may not be optimized for your system but will be a good starting point for most applications .

Please note that it is the responsibility of the user to determine the suitability of all parameters, including those determined by auto-tuning, for use in a given application.

Another method is through trial and error using the Magellan's built-in trace facility (see Section 8.8, "Trace Capture" for details). Finally, it is possible to model your system and determine the best settings through simulation or analysis; however, a discussion of this approach is beyond the scope of this manual.

15.3 Enabling and Disabling Current Loop

If during normal operation the current loop is disabled, then the output from the previous module will pass directly to the motor output module, with no current control being performed. The most common use of this configuration is to run the drive in voltage mode, which may be useful under some conditions for calibration or testing. Use the command SetOperatingMode to enable or disable the current loop module.

15.4 Current Feedback

For proper current loop operation feedback signals must be provided to the Magellan IC representing the current flowing through the motor coils. These signals are encoded as analog voltage levels, from 0.0V to 3.3V at the Magellan IC. The more accurate and noise-free these signals are, the higher the performance of the current loop will be.

For ION and Atlas-connected axes the current feedback signals are provided by dedicated circuitry located internal to the drive. For the MC58113 the motor's instantaneous current readings are provided via up to four signals depending on the motor type being controlled. These signals are *CurrentA*, *CurrentB*, *CurrentC*, and *CurrentD*. MC58113 expects the external circuitry to provide leg current signals, meaning ground-referenced current readings through the lower legs of the bridge. See the *MC58113 Electrical Specifications* for details and complete example schematics.

15.5 Reading Current Loop Values

To facilitate tuning, there are a number of internal current loop values that can be read back as well as traced. To read back these values the command GetCurrentLoopValue or GetFOCValue is used. To specify these values for trace during automatic trace capture, see the *Magellan Motion Control IC Programmer's Command Reference.*

The variables within the current loop that can be read or traced are summarized in the following table. Refer to Figure 15-1 for an overview of the current loop.

Variable Name	Function
Phase A Reference, Phase B Reference	*Brushless DC & microstepping motor:* These registers hold the commanded (reference) currents for the phase A and phase B coils. *DC brush motor:* Phase A Current holds the commanded (reference) current for the motor.
Phase A Current, Phase B Current	*Brushless DC & microstepping motor:* These registers hold the measured currents for the phase A and phase B coils. *DC brush motor:* Phase A Current holds the measured current for the motor.
Phase A Error, Phase B Error	*Brushless DC & microstepping motor:* These registers hold the difference between the current loop reference and the measured current value (Phase A Current, Phase B Current). *DC brush motor:* The Phase A Error register holds the difference between the current reference and the measured current value (Phase A Current).
Phase A Integrator Sum, Phase B Integrator Sum	*Brushless DC & microstepping motor:* These registers hold the sum of the integrator for the phase A and B current loops. *DC brush motor:* Phase A Integrator Sum holds the sum of the integrator for the current loop
Phase A Integral Contribution, Phase B Integral Contribution	*Brushless DC & microstepping motor:* These registers hold the overall contribution of the integrator for the phase A and B current loops. *DC brush motor:* Phase A Integral Contribution holds the overall contribution of the integrator for the current loop
Phase A Output, Phase B Output	*Brushless DC & microstepping motor:* These registers hold the output command for the phase A and B current loop. *DC brush motor:* Phase A Output holds the output command for the current loop.

15.6 Drive Safety Feature Overview

In addition to current control, a number of amplifier control features are provided that improve safety or reduce external circuitry required to build a complete drive.

The subsequent sections of this chapter describe these features.

15.7 Overcurrent Sense

The ION, MC58113, and Atlas-connected Magellans support automatic detection of major amplifier, voltage supply, or other electrical hardware problems. These serious fault conditions are described in the following table:

Fault Name	Description
Overcurrent	An overcurrent fault across the drive output is detected. This fault occurs when the motor, the wiring leading from the drive, or the internal circuitry of the drive becomes short circuited.

Fault Name	Description
Ground fault	This fault indicates that one or more of the motor connections are short circuited to the power supply ground. This fault occurs when the motor, the wiring leading from the digital drive to the motor, or the internal circuitry of the drive becomes short circuited to ground (ION only).
Electrical fault	This fault indicates that the supply voltage for drive logic components external to the motion control IC is too low or too high. This can occur when there is a problem with the circuitry external to the motion control IC, or when the motion control IC has experienced anomalous electrical conditions such as excessive supply voltage or excessive voltage on input or output signals (ION only).

Any of these faults will cause the following sequence of events:

- The Magellan's motor output module is disabled, thereby halting further motor output.
- Depending on the state of the FaultOut mask the ***FaultOut*** signal may be set to active (see Section 15.14, "Drive Fault Status Register" for details).
- The cause of the fault is saved in non-volatile memory (ION only).
- The motion control IC may enter a special state that requires that power be cycled (ION only). If this is the case, until power is cycled, no commands will be accepted and no further motion processing of any kind occurs. Check the *ION Digital Drive User's Manual* for details.

To recover from this condition, the user should determine the nature of the fault using the **GetDriveFaultStatus** command and, once the cause of the fault has been corrected, use the **ClearDriveFaultStatus** command to clear the condition. Although it is not required that this command be sent, it is often useful for safety and diagnostic reasons.

Overcurrent, ground fault, and electrical faults are serious conditions and warrant the utmost precaution before repowering and re-enabling the drive. It is the responsibility of the user to determine the cause and appropriate corrective action for an electrical fault. Refer to the *ION Digital Drive User's Manual*, the *Atlas Digital Amplifier User's Manual*, or the *MC58113 Electrical Specifications* for additional details on the procedure used to recover from an electrical fault.

15.8 Overtemperature Sense

ION, MC58113, and Atlas-connected axes provide the capability to continually monitor drive temperatures using sensors. A programmable value set using the command **SetOvertemperatureLimit** (ION) or **SetDriveFaultParameter** (MC58113 and Atlas-connected axes) is compared to a value read from a temperature sensor, and if the value read from the sensor exceeds the programmed threshold, an overtemperature fault occurs. To read the value set the command **GetOverTemperatureLimit** (ION) or **GetDriveFaultParameter** (MC58113 and Atlas-connected axes) is used.

ION temperature readings occur via a thermistor located inside the physical ION drive unit. The actual value read represents the temperature inside the ION drive. Similarly, the temperature reading for Atlas-connected axes is located inside the Atlas unit. For MC58113, the temperature is input by analog signal via the ***Temperature*** input pin. Therefore the physical location of the sensor, if one is used, is determined by the user. Most users will locate the temperature sensor on or near the amplifier, but this is not required.

When used with ION and Atlas, the value set using this threshold must have a value less than or equal to the rated maximum for the drive. See the *ION Digital Drive User's Manual* or *Atlas Digital Amplifier's User's Manual* for details.

When used with ION and Atlas, the value set using the **SetOverTemperatureLimit** command is in units of deg C/256. For example, a value of 12,800 indicates a threshold of 50 deg C. With the MC58113 there is no explicit correspondence between the values read and the actual temperature in deg C, since this may vary with the sensor being used.

Overtemperature faults indicate that the internal safe limits of the drive temperature range has been exceeded. This potentially serious condition can result from incorrect motor connections or from excessive torque demands placed on the drive.

An overtemperature fault will cause the following events:

- The motor output module is automatically disabled.
- The overtemperature bit in the Event Status register is set active.

To recover from this condition, the user should determine the reason for the fault and correct accordingly. It is always the responsibility of the user to maintain safe operating conditions of the drive and associated electronics. Once this has occurred, the overtemperature bit of the Event Status register should be cleared. This can be accomplished using **ResetEventStatus**. The normal operation of the control modules can then be restored using **RestoreOperatingMode**.

The instantaneous status of the overtemperature threshold comparison can be read using the command **GetDriveStatus**. If the overtemperature fault condition is still occurring at the time the overtemperature bit of the Event Status register is cleared, this bit will immediately be set again, and the recovery sequence must be executed again.

To read the current value of the temperature sensor, the command **GetTemperature** is used.

15.9 Overvoltage Sense

ION, MC58113, and Atlas-connected axes provide the capability to sense overvoltage conditions in the incoming main bus voltage. A programmable threshold set using the command **SetBusVoltageLimits** (ION) or **SetDriveFaultParameter** (MC58113 and Atlas-connected Magellan axes) is compared to the value read from the drive DC bus supply, and if the value read exceeds the programmed threshold, an overvoltage fault occurs. To read the value set, the command **GetBusVoltageLimits** (ION) or **GetDriveFaultParameter** (MC58113 and Atlas-connected Magellan axes) is used.

For ION and Atlas-connected axes the bus voltage readings and limits are provided by dedicated circuitry internal to the drive. For the MC58113 the bus voltage is input via the ***BusVoltage*** pin in connection with appropriate external circuitry.

The value set using this threshold must have a value equal to or less than a prescribed maximum for the drive. See the user's manual for the ION or Atlas drive product you are using for details. For the MC58113 there is no specific maximum setting because the scaling of the voltage will depend on the external circuitry used.

An overvoltage fault will cause the following events:

- The motor output module is disabled.
- The bus voltage sense bit in the Event Status register becomes active

To recover from this condition, the user should determine the reason for the fault and correct accordingly. It is always the responsibility of the user to maintain safe operating conditions for the ION drive and associated electronics. Once this has occurred, the bus voltage bit of the Event Status register should be cleared. This can be accomplished using **ResetEventStatus**. The normal operation of the control modules can then be restored using **RestoreOperatingMode**.

The instantaneous value of the overvoltage threshold comparison can be read using the command **GetDriveStatus**. If the overvoltage fault condition is still occurring while the overvoltage bit of the Event Status register is being cleared, this bit will immediately be set again, and the recovery sequence described above must be executed again.

The drive supply voltage can be monitored using the **GetBusVoltage** command. It returns the current supply voltage reading.

15.10 Undervoltage Sense

ION, MC58113, and Atlas-connected amplifiers provide a capability similar to the overvoltage sense except that it monitors undervoltage. To set the programmable threshold the command **SetBusVoltageLimits** (ION) or **SetDriveFaultParameter** (MC58113 and Atlas-connected Magellan axes) is used. This value is compared to the value read from the drive DC bus, and if the value read is less than the programmed threshold, an undervoltage fault occurs.

The value set using this threshold must have a value equal to or less than a prescribed maximum for the drive. See the user's manual for the ION or Atlas drive product you are using for details. To read the value set, the command **GetBusVoltageLimits** (ION) or **GetDriveFaultParameter** (MC58113 and Atlas-connected Magellan axes) is used.

For ION and Atlas-connected axes the bus voltage readings and limits are provided by dedicated circuitry internal to the drive. For the MC58113 the bus voltage is input via the ***BusVoltage*** pin in connection with appropriate external circuitry.

Threshold units, recovery procedure, and all other aspects of this feature are the same as for overvoltage sense except that the bit status location in the Drive Fault Status register is different. And just as for overvoltage conditions, it is the user's responsibility to determine the seriousness of, and appropriate response to, an undervoltage condition.

15.11 Current Foldback

ION, MC58113, and Atlas-connected axes support a current foldback feature, sometimes referred to as an I^2t foldback, which can be used to protect the drive output stage or motor windings from excessive current. I^2t current foldback works by integrating, over time, the difference of the square of the actual motor current and the square of the user-settable continuous current limit.

When this integrated value reaches a user-settable energy limit, the axis goes into current foldback. The axis will stay in foldback until the integrator returns to zero. When in current foldback, the current is clamped to the continuous current limit value, although it is also possible to program additional actions such as disabling motor output.

To set the event action associated with a current foldback event the command **SetEventAction** is used.

Each ION and Atlas drive has particular default values as well as maximum allowed settings for the continuous current limit and energy limit. These values are designed to protect the drive from excessive heat generation. See the ION Users manual for detailed information or the *Atlas Digital Amplifier Complete Technical Reference*. For the MC58113 there are no specific maximum settings because the limits will depend on the external circuitry used.

Setting continuous current limit and energy limit to less than the maximum supported by the ION drive or Atlas is useful if the required current limit is due to the motor, rather than to the ION drive. Continuous Current Limit and Energy Limit can be set using the command **SetCurrentFoldback**. The values set using this command can be read back using **GetCurrentFoldback**.

The instantaneous state of current foldback (whether the foldback limit is active or not) is available in the Drive Status register and can be read using the command **GetDriveStatus**. In addition, if a foldback event has occurred, this event is recorded in the Event Status register and can be read back using **GetEventStatus**.

As detailed in Section 8.1, “SetEventAction Processing” if desired, an event action can be programmed for current foldback using the command SetEventAction. Whether this is appropriate must be determined by the user.

Example I^2T calculations

A particular motor has an allowed continuous current rating of 3 amps. In addition, this motor can sustain a temporary current of 5 amps for 2 seconds.

In this example the *continuous current limit* would be set to 3 amps, and the energy limit would be set to:

Energy Limit = (peak current^2 - continuous current limit^2) * time

Energy Limit = ($5A^2$ - $3A^2$) * 2 Sec

Energy Limit = $32A^2$Sec

To determine the actual parameter values that should be sent to the SetCurrentFoldback command, consult the *ION Users Manual* or the *Atlas Digital Amplifier User's Manual* for the particular drive that you are using.

Current foldback, when it occurs, may indicate a serious condition affecting motion stability, smoothness, and performance. It is the responsibility of the user to determine the appropriate response to a current foldback event.

15.12 Drive Enable Signal

ION, the MC58113, and Atlas amplifiers support an ***Enable*** input signal that must be active for proper operation. This signal is useful for allowing external hardware to indicate a fault to the drive and thereby automatically shutting it down. The signal has an active low interpretation.

If the ***Enable*** signal becomes inactive (goes high) the following events occur:

- The motor output module is disabled.
- the disable bit in the Event Status register becomes active. For Atlas, the drive exception bit of the Event Status register becomes active and the Disabled bit of the Drive Fault Status register is set.

To recover from this condition, the user should determine the reason for the enable becoming inactive, and correct accordingly. It is always the responsibility of the user to maintain safe operating conditions of the drive and associated electronics. Once this has occurred, the appropriate bit of the Event Status register should be cleared. This can be accomplished using ResetEventStatus. The normal operation of the control modules can then be restored using RestoreOperatingMode.

If the ***/Enable*** signal is still inactive while the disable or drive exception bit of the Event Status register is being cleared, this bit will immediately be set again, and the recovery sequence must be executed again.

The status of the ***/Enable*** signal can be read using the command GetSignalStatus.

15.13 FaultOut Signal

ION and MC58113 support a ***FaultOut*** signal, which is used to indicate an occurrence of one or more of the faults indicated in the previous section. This signal is always active high, its sense cannot be changed using the command SetSignalSense. It is possible to use this signal to indicate additional motion control IC conditions. In particular, any bit condition of the Event Status register may be used to trigger activation of the fault signal. This is done using the command **SetFaultOutMask**. The value set using this command can be read back using **GetFaultOutMask**.

For example, programming **SetFaultOutMask** with a value of 20 (14 hex) configures the ***FaultOut*** signal to be driven high upon a Motion Error, BreakPoint1, or an electrical fault.

For Atlas-connected axes, the Atlas units' ***FaultOut*** signal is also programmable, but the conditions are internal to Atlas only. For more information on how to program the ***FaultOut*** behavior of the Atlas see the *Atlas Digital Amplifier Complete Technical Reference.*

15.14 Drive Fault Status Register

To simplify recovery from a drive fault, as well as to read other drive-related faults, Magellan provides a Drive Fault Status register that can be read using the command **GetDriveFaultStatus**. The bits in this register operate similarly to the bits in the Event Status register in that they are set by the motion control IC, and cleared by the user. The following table indicates the contents of this register:

Bit	Name	Description
0	Overcurrent	Set 1 to indicate an electrical fault due to a short circuit or overload in the drive output.
1	Ground fault	Set 1 to indicate an electrical fault due to a short in the drive output (ION only).
2	External logic fault	Set 1 to indicate an electrical fault located in the drive's output circuitry (ION only).
3	Mode mismatch	Set 1 to indicate that Atlas has received a torque or voltage command at a time when this is not a valid operation. (Atlas-connected axes only)
4	Internal logic fault	Set 1 to indicate an electrical fault located in the drive's internal logic circuitry (ION only).
5	Overvoltage	Set 1 to indicate an overvoltage condition of the external bus voltage input.
6	Undervoltage	Set 1 to indicate an undervoltage condition of the external bus voltage input.
7	Disabled	Set 1 to indicate that the amplifier enable signal input is inactive (Atlas-connected axes only).
8	Foldback	Set 1 to indicate that a current foldback event has occurred.
9	Overtemperature	Set 1 to indicate that an overtemperature fault has occurred.
10	SPI checksum error	Set 1 to indicate that the attached Atlas detected a checksum error or a disable command. (Atlas-connected axes only)
11	Communication watchdog	Set 1 to indicate that the drive has not received an expected transmission of data.Typically used to flag conditions where the motion controller stops sending commands to the amplifier. (Atlas-connected axes only)
12	*Reserved*	May contain 0 or 1
13	PWMOutputDisable	Set to 1 to indicate that the PWMOutputDisable signal input is active (MC58113 only).
14	SPI checksum error	Set 1 to indicate that Magellan has detected a checksum error while sending a communications packet to the drive. (Atlas-connected axes only)
15	Motor type mismatch	Set 1 to indicate that the motor type that Magellan is set to does not match the motor type of the attached Atlas amplifier. (Atlas-connected axes only)

To clear the bits in this register the command **ClearDriveFaultStatus** is used. For MC58113 and Atlas-connected axes at least one read of this register via the **GetDriveFaultStatus** command must occur before bits can be cleared.

For ION, unlike other registers in the Magellan Motion Control IC, a portion of the contents of this register are saved after a power cycle. Thus electrical faults (bits 0, 1, 2, and 4), which cause all communications with the drive to cease, and require the unit power to be cycled, can still be diagnosed once the condition has been corrected and the unit has been powered up normally. The Overvoltage and Undervoltage status bits are not saved but rather are cleared during a power cycle so they always reflect the latest voltage fault status.

In the case that an Atlas is connected to an MC58113, the Atlas will drive the above Drive Fault Status register flags, and the related MC58113 signals will not be utilized.

15.15 Atlas-Specific Commands

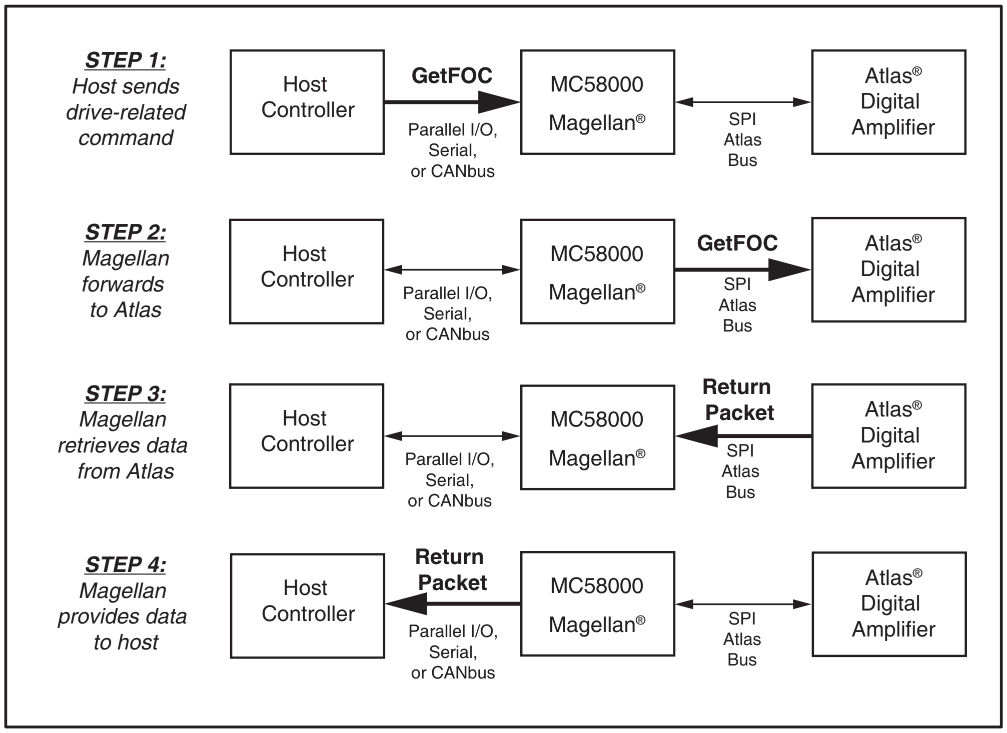
Atlas functions are seamlessly accessible from Magellan so that the user need not be versed in the internal details of the SPI Atlas protocol. Magellan accomplishes this by interpreting incoming commands from the host and routing them to internal destinations, or to appropriate Atlases depending on the nature of the host-requested function.

Although this automatic function distribution is convenient, there may be times when it is desirable for the host to send commands directly to the Atlas amplifier. To accomplish this, bit 5 of the Magellan command word is set active, and the appropriate axis number that corresponds to the Atlas being communicated to must be specified in the Magellan command word. Refer to the *Magellan Motion Control IC Programmer's Command Reference* for complete information on Magellan host I/O protocol.

Figure 15-2 shows this, showing a host -requested command being sent to the Magellan, which in turn forwards the command as well as its response back to the Magellan, which in turn provides the requested information to the host.

For a complete description of Atlas commands, see the *Atlas Digital Amplifier Complete Technical Reference.*

Figure 15-2: Host To Magellan To Atlas Packet Processing



15.16 Setting Atlas Initialization Configuration

Atlas drives allow many parameters, such as current gains, over and under temperature thresholds, and many other quantities to be stored permanently into an on-unit NVRAM (non volatile RAM). When stored on the unit, these parameters are automatically loaded and no longer need to be sent by the host. This is useful to reduce the number of startup host commands that must be sent to get the system ready for operation.

It is generally simplest to utilize PMD's Windows-based Pro-Motion program to configure the NVRAM memory. Nevertheless, it is also possible for the host to send these programming commands directly. Consult the *Atlas Digital Amplifier Complete Technical Reference* for detailed information on how this can be accomplished.

This page intentionally left blank.

16. Memory Buffers and I/O

In This Chapter

- Memory Configuration
- User I/O

16.1 Memory Configuration

Magellan Motion Control ICs are capable of accessing memory buffers for the storage of trace data or to be used for generic storage purposes. For MC50000 users except MC58113 this memory, if used, is located external to the Magellan IC and must be designed into the card by the user. Correspondingly the size of this memory buffer is determined by the user.

For card users, the amount of memory is fixed. See the *Magellan Motion Controller Card User's Guide* for details. ION and MC58113 users also have a fixed amount of buffer storage, however located internal to the Magellan IC. See the *ION Digital Drive User's Manual* or the *MC58113 Electrical Specifications* for details.

16.1.1 Memory Buffers

Magellan buffers are a contiguous block of memory with a base address and length. Once a buffer's base address and length have been defined, data values may be written to and read from the buffer.

When defining memory buffers the memory space is treated as a sequence of 32-bit memory locations. Each 32-bit value takes up two 16-bit memory locations in physical memory, with the high word of the 32-bit word stored in the upper memory location and the low word stored in the lower memory location. Buffer base addresses and lengths both deal with 32-bit quantities and therefore must be doubled to get the corresponding physical addresses.

When defining memory buffers the motion control IC will allow any values to be used for the base address and length, as long as the values result in legal addresses. Legal memory addresses range from 0 to 3FFF FFFFh (corresponding to physical address 7FFF FFFEh). Unless the full two gigawords of physical memory are present, it is possible to map a buffer to a memory location that does not contain physical memory.

In addition to the base address and length, each memory block maintains a read index and a write index. The read index may be assigned a value between 0 and L-1 (where L is the buffer length). It defines the location from which the next value will be read. Similarly, the write index ranges from 0 to L-1 and defines the location at which the next value will be written. When a value is read from the memory buffer, the read index is automatically incremented, thus selecting the next value for reading. The write index is incremented whenever a value is written to a buffer. If either index reaches the end of the buffer, it is automatically reset to 0 on the next read/write operation.

16.1.2 Memory Buffer Commands

This section details host commands that set up, access, and monitor memory buffers. Note that for ION users, these commands are not necessary, as the location and size of the available buffer are already set.

SetBufferStart bufferID, address
Sets the base address of a buffer. bufferID is a 16-bit integer in the range 0–31 that specifies the buffer to be mod-

ified. The variable address is a 32-bit integer in the range 0 to 3FFF FFFFh that defines the new base address of the buffer.

The motion control IC adds address to the current buffer length (as set by the SetBufferLength instruction) to ensure that the buffer will not extend beyond the addressable memory limit. If it would extend beyond the limit, the instruction is ignored and the instruction error bit is set..

GetBufferStart bufferID
Returns the base address of the specified buffer.

SetBufferLength, bufferID, length
Sets the length of a buffer. bufferID is a 16-bit integer in the range 0–31. length is a 32-bit integer in the range 1 to 3FFF FFFFh.

The motion control IC adds length to the current buffer base address (as set by the SetBufferStart instruction) to ensure that the buffer will not extend beyond the addressable memory limit. If the buffer would extend beyond the limit, the instruction is ignored and the instruction error bit is set.

SetBufferStart and SetBufferLength reset the buffer indexes to zero (0).

GetBufferLength bufferID
Returns the length of the specified buffer.

SetBufferReadIndex bufferID, index
Sets the read index for the specified buffer. index is a 32-bit integer in the range 0 to length–1, where length is the current buffer length. If index is not in this range, it is not set, and an instruction error is generated.

GetBufferReadIndex bufferID
Returns the value of the read index for the specified buffer.

SetBufferWriteIndex bufferID, index
Sets the write index for the specified buffer. index is a 32-bit integer in the range 0 to length–2, where length is the current buffer length. If index is not in this range, it is not set, and an instruction error is generated.

GetBufferWriteIndex bufferID
Returns the value of the write index for the specified buffer.

ReadBuffer bufferID
Returns a 32-bit value from the specified buffer. The location from which the value is read is determined by adding the base address to the read index. After the value has been read, the read index is incremented. If the result is equal to the current buffer length, the read index is set to zero (0).

WriteBuffer bufferID, value
Writes a 32-bit value to the specified buffer. The location to which the value is written is determined by adding the base address to the write index. After the value has been written, the write index is incremented. If the result is equal to the current buffer length, the write index is set to zero (0).

16.2 User I/O

Magellan ICs except MC58113 implement a peripheral device address space that is accessed with the ReadIO and WriteIO commands. This address space is used to access hardware peripherals external to the Magellan chipset. For a complete memory map of the peripheral address space see MC5X00 Series Electrical Specifications.

ReadIO and WriteIO take an address parameter which is an offset from location 1000h of the motion control IC's hardware peripheral device address space.

The format and interpretation of the 16-bit data word are dependent on the user-defined device being addressed. User-defined I/O can be used to implement a number of features, including additional parallel I/O, flash memory for non-volatile configuration information storage, or display devices such as LED arrays.

Note that User I/O commands use a different signalling protocol and different timing than external memory access commands. (See the *MC55000 Series Electrical Specifications* or the *MC58000 Series Electrical Specifications* for details).

16.2.1 User I/O Commands

This section details host I/O commands that cause Magellan to access its peripheral device address space.

ReadIO address
Reads one 16-bit word of data from the device at address, where address is an offset from location 1000h of the motion control IC's peripheral device address space.

WriteIO address, data
Writes one 16-bit word of data to the device at address, where address is an offset from location 1000h of the motion control IC's peripheral device address space.

This page intentionally left blank.

Index

Numerics
50/50 PWM mode 89

A
A/D converters 83
abrupt stop 33
absolute
  encoder 14
  magnitude 88
  optical encoders 83
acceleration
  parameter 27
  slope 27
  value 27
active registers 47
Activity Status register 55, 57
actual motor position 35
actual position 66, 82, 135
  units 134
additional trace data 68
addressable memory
  limit 154
  space 69
algorithmic initialization
  mode 124
  sequence 126
amplifier
  linear 101
  switching 101
analog
  filtering 82
  input 78
  reference signal 99, 101
  signal 78
analog-to-digital converter 14
asynchronous
  event notifications 114
  frame 111
  serial connection 110
  serial port 15, 105
at max velocity indicator 51
at rest indicator 139
Atlas 16
  drive control 141
  field-oriented control 129
  traces 72
automatic stall detection 135
auxiliary axis parameter 40
axis
  breakpoint 49
  input pin 77
  master 31
  output pin 77
  position 63
  settled 67
  slave 31
  source 49

B
backlash 38
bad checksum 109
band-pass filter 41
bi-directional parallel port 105
binary-encoded position 15
bi-quad 41
  coefficient scaling 43
  coefficients 42
  output filter 41
bit
  event status 51
  mask 48
  status 51
bitmasked value 75
bit-oriented
  fields 55
  status registers 55
bit-programmed word 124
breakaway starting friction 125
breakpoint 48, 66, 74
  axis 49
  deactivate 51
  defined 52
  instruction 49
  level-triggered 50

number 49
operations 55
threshold triggered 50
threshold-triggered 50
updatemask 49
breakpoint 1 49
breakpoint 2 49
Brushless DC motor control 121
automatic phase correction 126
encoder prescaler 128
field oriented control 129
number of phases 122
overview 121
phase counts 123
phase initialization 123
phase initialization programming 125
phasing control modes 122
sinusoidal commutation 128
buffer
base address 153
length 154
read index 154
write index 154
buffered
commands 47, 48
operations 109
butterworth filter 42

C

CAN 114
address 116
interface 116
interface layer 114
interface protocol 114
message 116
network 116
node ID 116
notification 116
CAN 2.0B
interface 15
network 114
CANOpen 114
capture
received 74
register 83
captured data 68, 73
checksum 105, 109
byte 112
calculation 112
invalid 112
valid 112
value 112
clear interrupt 75
coefficients 42
coil current distortion 99
command
buffered 47, 48
format 105, 111
MultiUpdate 48
packet 108, 112, 113
packet fields 111
packet sequence 106
read 105
update 48
values 25
write 105
commanded position 63, 66, 135
commutation
angle 129
error 74
frequency 91
rate 23
reliability 126
signals 122
waveforms 122, 129
comparison value 49
continuous data retrieval 68
control modules 19
control signals 107, 108
controlled stop 33
current
control 99
foldback 74
custom filter 41
cycle
frequency 23
rate 23
time 22
timer 66

D

DAC 15, 91
channels 97
outputs 91
signal timing 91
values 91

data
    buffers 69
    collection synchronization 69
    frame format 111
    packet 112
    packet time limit 113
    trace operations 73
    traces 68
    transfer 105
dataless command 105
DC servo motor output 100
debugging 134
deceleration slope 27
defined breakpoint 52
derivative term, servo loop 38
differential line drivers/receivers 82
digital
    filtering 82
    motor output word 99, 101
direct
    output bit 77
    set phase initialization 125
direction
    of motion 31
    signal, encoding 135
direction signal encoding 102
disable 74
disabling control modules 21
drive control 141
    current feedback 143
    current foldback 147
    disabling current loop 143
    dribe fault status register 149
    drive enable signal 148
    drive safety feature 144
    enabling current loop 143
    FaultOut signal 148
    overcurrent sense 144
    overtemperature sense 145
    overvoltage sense 146
    reading current loop lalues 143
    undervoltage sense 147
drive data 92
drive exception 74
Drive Status register 55, 58
dual encoder
    mode 39
    processing 40
dual encoder support 38
dual loop processing 40

E

electrical cycles 123
electronic
    gear profile 32
    gear profile mode 31
    gearing 25, 66
emergency stop 33
EMF 89, 99
enabling 86
enabling control modules 21
encode, incremental 83
encoder
    counts 126, 134
    modulus 84
    prescaler 128
    to step ratio 135
end-of-travel condition 63
error
    codes 112
    commutation 74
    instruction 74
    position 63
event 49
event status 55
    bit 51
Event Status register 55, 75
    clearing 75
external
    buffer memory 73
    peripherals 77
    quadrature decoder circuit 83

F

falling edge 92
feedback type 81
filter
    band-pass 41
    bi-quad output 41
    butterworth 42
    custom 41
    high-pass 41
    low-pass 41
    motor output 43
    parameters 37
    second-order butterworth 42
fixed-point representation 25

follow-on function 66
full scale feedback range 83
functional overview 14

G
gear ratio 32
generic command packet sequence 106
GetVersion command 24

H
half-bridge 97
  driver 89
Hall interpretation value 124
Hall-based
  initialization 123
  initialization sequence 125
hardware communication operation 106
H-bridge
  amplifiers 88
  device 99
high-pass filter 41
high-speed position capture register 82
home signal 82
host
  I/O commands 112, 153, 155
  instructions 28
  interrupt 66, 74
  interrupt events 74
  interrupt facility 116
host-specified profile parameters 26, 28

I
idle-line protocol 113
in motion indicator 51
incorrect command transfer 109
incremental
  encoder feedback 81
  encoder signals 14
  encoders 83
  feedback 82
  signals 14
index pulse
  input 126
  phase correction 126
  signal 126
index signal 82
inductance 99
initialization 124
  duration 125
  motor command value 125
  procedure 125
in-motion
  bit 66
  indicator 66
instantaneous
  commanded profile values 25
  deceleration 33
instruction
  error 56, 74
  error bit 154
  word 105
integration
  limit 38
  limit scaling 38
  term 38
interconnection diagram 14
interconnections 13
interface signals 109
internal scaling factor 42
interrupt 75
  mask 75
inverter circuitry 124
inverting summing amplifier 91

J
jerk 29

K
Ki value 38
Kout
  parameter 38
  value 38
Kp 38

L
laser interferometer 14, 83
level-triggered breakpoint 50
limit
  positive 74
  switch event 64, 66, 75
  switches 14
linear
  amplifier 99, 101
  interpolation 32
low pass filter 41, 99, 101

M
Magellan family 11
Magellan motion processing 16
mask
  interrupt 75
  sense 51
  trigger 50
master
  axis 31
  axis number 31
  mode 78
Matlab 42
maximum position error 63
memory buffer 68, 153
microstepping
  control flow 137
  output values 137
  signal output frequency 137
  signals, amplitude 137
microstepping motor output 98
minimum timeout 113
mode 25
  master 78
  slave 78
modulo 84
modulus 83
  encoder 84
motion
  complete 74
  complete bit 66
  complete indicator 66
  error 14, 63, 74
  error bit 63
  error cause 63
  error recovery 63
  error status bit 63
  processor address 113
  profile 25
  profile complete 66
motion complete indicator 66
motor
  bias 43
  command output 88, 125
  command register 129, 130, 137
  command value 95
  limit 43
  output limit 43
  output options 87
  output waveform 95
  phase angle 127
  phasing 125
  three-phase step 138
  two-phase step 138
  type 86
Motor Output module 86
  disabling 85
multi-drop idle-line mode 113
multi-turn systems 83
MultiUpdate command 48

N
negative gear ratio value 32
Negative Limit 74
noise
  pulses 82
  sources 82
non-buffered instruction 109
notch filter 41
notify mask 116

O
Octave 42
one-time trace mode 68
op-amp circuit 100
optional bias value 36
output
  clocking 92
  filter coefficients 42
  formats 88
  scale factor 36
  scaling factor 42
overshoot 28
overtemperature fault 74
over-travel 63

P
packet 105, 112
  command 112, 113
  data, limit time 113
  format 105
  response 111, 112
parallel port
  configuration 107
  control signals 108
parallel-bus interface 15

parallel-word 83
  feedback 81
  input 14
  input mechanism 83
parameter
  ranges, filter 37
  traces 69
parity bit 111
part numbers 16
peripheral
  bus 91
  read 84
phase
  angle 123
  cycle 127
phase initialization 123, 125, 127
  algorithmic 123
  bit 126
  command 125
  direct-set 123
  Hall sensor-based 123
phase offset
  register 127
  value 127
phase referencing, commutation error 127
phasing error 127
phasing procedure 125
PID
  algorithm 36
  filter 15
PID-type servo filter 35
point-to-point 15
  configuration 114
  mode 110
  protocols 113
  serial mode 113
position
  actual 135
  capture register 83
  commanded 135
  encoder 129
  error 63, 65
  error limit 65
  feedback value 84
  loop 35
  tracking 83
  wraparound 74
positive
  gear ratio value 32
  limit 74
  limit switch 64
pre-commutated command signal 129
prescaler 128
  function 128
product summary 11
profile 25
  generator 28, 66
  generator registers 66
  parameters 25, 28
programmable tracking window 65
programmed acceleration value 29
proportional-integral-derivative algorithm 36
pulse
  output rate 136
  rate modes 103
Pulse & Direction signal, generation 102
PWM 15
  magnitude 88
PWM frequency, setting 94
PWM signal interpretation, setting 94

Q

quadrature
  counts 81
  data 82
  data stream 125
  incremental position 81

R

ratio, encoder to step 135
read
  buffer 73
  command 105
  data 69
  index 73
  pointer 69
register
  capture 83
  motor command 129
  phase offset 127
  position capture 83
Reset command 22
resetting the loop rate 23
resolver 14, 83, 125
response packet 111, 112
resynchronize 113

rising edge 92
rolling
  buffer 73
  trace mode 68

S

S-curve
  mode 30
  point-to-point 25
  point-to-point profile 28
  profile 29
second-order butterworth filter 42
sense mask 51
separate pulse rate modes 103
serial
  checksum 112
  data transfer 111, 117, 118
  hardware signals 112
  operations 113
  peripheral
    interface 92
Serial Peripheral Interface 117
serial port, default configuration 110
servo
  filter 39
  loop 129, 130
  loop calculation rate 23
  parameter commands 48
  performance 65
set cycle time 22
SetEventAction command 61
settle
  time 66, 67
  window 66, 67
settled indicator 67
sign magnitude PWM 88
signal
  control 107
  encoded direction 135
  home 82
  index pulse 126
  interpretation 59
  parallel port control 108
  pre-commutated command 129
Signal Status mask 59
Signal Status register 55, 59
signed numerical value 88
sinusoidal waveform generation 99, 137
slave
  axis 31
  mode 78
smooth stop 33
source axis 49
SPI
  Atlas 93
  Atlas communications protocol 94
  Atlas, communications overview 93
  Atlas, functional overview 93
  communications 117
  communications overview 93
  DAC mode 92
  data 92
  output 92
  output pin 92
square wave pulse train 102, 135
stall
  condition 135
  detection 135
starting velocity 27
status
  bit 51
  event 55
  read operation 109
  registers, bit-oriented 55
step motor
  support 134
  three-phase 138
  two-phase 138
step motor control 133
  encoder feedback 134
  microstepping 136
  overview 133
  pulse & direction 135
  stall detection 135
stop
  abrupt 33
  bits 111
  command 66
  controlled 33
  emergency 33
  smooth 33
switching amplifier 99, 101
synchronization
  pin 78
  sample time 79

synchronized
  motion 78
  power changes 137
synchronizing multiple motion control ICs 79
system overview 13

T
three-phase
  AC induction motor 100
  step motor 138
threshold triggered breakpoints 50
Time register 23
time slice 23
timeout period 113
trace
  buffer 68, 69, 73
  buffer configuration 69
  capture 68
  capture overhead 22
  data 69
  data capture 68
  data retrieval 68
  data storage 153
  mode 72, 74
  period 74
  samples 73
  start 74
  start/stop 72
  stop 74
traceable parameter 69
tracing
  frequency 69
  system 69
tracking
  bit 65
  window 65
  window, size 65
trajectory
  generator 15, 25, 35, 47, 63
  generator registers 66
  motion 66
  parameters 25
  profile 66
  profile mode 47
  profile modes 25
  update rate 23
transfer protocols 15
transmission
  errors 112
  protocols 113
trapezoidal
  mode 30
  point to point 25
  point-to-point profile mode 26
trigger 49
  mask 50
triggering event 55
two-phase step motor 138

U
update command 48
User I/O 154

V
value
  phase offset 127
velocity
  contouring 25
  starting 27
velocity-contouring profile mode 30

W
waveform, magnitude 95
write command 105
write index 153

This page intentionally left blank.

For additional information, or for technical assistance,
please contact PMD at (978) 266-1210.

You may also e-mail your request to support@pmdcorp.com

Visit our website at http://www.pmdcorp.com

PMD

Performance Motion Devices
80 Central Street
Boxborough, MA 01719