

MC73110

Advanced 3-Phase Motor Control IC

Product Manual



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



NOTICE

This document contains proprietary and confidential information of Performance Motion Devices, Inc., and is protected by federal copyright law. The contents of this document may not be disclosed to third parties, translated, copied, or duplicated in any form, in whole or in part, without the express written permission of PMD.

The information contained in this document is subject to change without notice. No part of this document may be reproduced or transmitted in any form, by any means, electronic or mechanical, for any purpose, without the express written permission of PMD.

Copyright 1998–2008 by Performance Motion Devices, Inc.

Magellan, ION, Magellan/ION, Pro-Motion, Pro-Motor, C-Motion, and VB-Motion are trademarks of Performance Motion Devices, Inc.

Warranty

PMD warrants performance of its products to the specifications applicable at the time of sale in accordance with PMD's standard warranty. Testing and other quality control techniques are utilized to the extent PMD deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Performance Motion Devices, Inc. (PMD) reserves the right to make changes to its products or to discontinue any product or service without notice, and advises customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

Safety Notice

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage. Products are not designed, authorized, or warranted to be suitable for use in life support devices or systems or other critical applications. Inclusion of PMD products in such applications is understood to be fully at the customer's risk.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent procedural hazards.

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

MC73110 Advanced 3-Phase Motor Control IC Developer's Kit Manual

This document guides you through installation and operation of the MC73110 Developer's Kit. It describes the developer's kit card and software, and provides complete schematics for the card.

Table of Contents


Chapter 1. Product Overview	9
Chapter 2. Specifications	11
2.1 Configurations, Parameters, and Performance	11
2.2 Physical Characteristics and Mounting Dimensions	12
Chapter 3. Electrical Specifications	13
3.1 Absolute Maximum Ratings	13
3.2 Recommended Operating Conditions	13
3.3 AC Characteristics	14
3.4 Timing Diagrams	15
3.5 Pin Descriptions	17
3.6 Phase Lock Loop (PLL)	20
Chapter 4. Theory of Operations	21
4.1 Functional Overview	21
4.2 Internal Block Diagram	23
4.3 Connection Summary	23
4.4 Control Loop Overview	24
4.5 Motor Output and Signal Generation	27
4.6 Current Loop	30
4.7 Commutation	32
4.8 Field Oriented Control (FOC)	35
4.9 Velocity Loop	36
4.10 Velocity Integrator	40
4.11 Profile Generation	42
4.12 Loop Rate	43
4.13 Status Words	44
4.14 Programmable Conditions	47
4.15 Temperature Sensor	50
4.16 Bus Voltage Sensor	51
4.17 Serial Port	51
4.18 Incremental Encoder Input	56
4.19 Serial EEPROM	57
4.20 Synchronous Serial Input (SPI Port)	59
4.21 Analog Signal Processing	60
4.22 GetLoop Commands and Variables	62
Chapter 5. Instruction Reference	67
5.1 How to Use This Reference	67



This page intentionally left blank.

List of Figures

2-1	MC73110 physical dimensions	12
3-1	MC73110 Clock timing diagram	15
3-2	MC73110 Quad encoder timing diagram	15
3-3	MC73110 Reset timing diagram	16
3-4	MC73110 SPI timing diagram	16
3-5	MC73110 chip pin layout and descriptions	17
3-6	PLL circuitry design	20
4-1	MC73110 internal block diagram	23
4-2	With an external motion controller	24
4-3	As a complete intelligent motion controller	24
4-4	Control loop flow	25
4-5	Configuration for a torque-mode amplifier	26
4-6	Configuration for a velocity-mode amplifier	27
4-7	Intelligent motion controller configuration	27
4-8	PWM waveforms and currents	28
4-9	Six-signal mode with dead time delay	29
4-10	Motor current sensing	30
4-11	Sinusoidal commutation	32
4-12	Motor command phasing vs. Hall sensors	34
4-13	FOC current control flow	36
4-14	Velocity feedback and scaling	37
4-15	Velocity integrator source select	41
4-16	Typical velocity profile	43
4-17	Typical data frame format	55
4-18	QuadA, QuadB, and Index signals	57
4-19	The current loop	62
4-20	The velocity loop	63
4-21	The velocity integrator loop	65



This page intentionally left blank.

1. Product Overview

1

	MC73110 Motor Control IC	Navigator/ Pilot	Magellan	Motion Cards	ION Digital Drive
Number of axes	1	1, 2, 4	1, 2, 3, 4	1, 2, 3, 4	1
Package	64-pin TQFP	132-pin PQFP 100-pin PQFP	144-pin LQFP 100-pin LQFP	PCI PC/104	Fully enclosed module
Voltage	3.3V	5V	3.3V	3V	12–56V
Function	Velocity control Torque control Commutation Encoder input	Position control Encoder input Profile generation Commutation	Position control Encoder input Profile generation Commutation Network communications Multi-motor	Position control Encoder input Profile generation Commutation Signal conditioning Analog output Trace buffer	Position control Profile generation Commutation Network communications Field oriented control Torque/current control Trace buffer MOSFET amplifier
Motor types	Brushless DC	DC brush Brushless DC Microstepping Pulse & direction	DC brush Brushless DC Microstepping Pulse & direction	DC brush Brushless DC Microstepping Pulse & direction	DC brush Brushless DC Microstepping
Communication	Standalone Serial	Parallel Serial point-to- point Serial multi-drop	Parallel Serial point-to- point Serial multi-drop CANbus	PCI, PC104	CANbus RS232/485
Loop rate	20 kHz	100–150 μ Sec/ axis	50–75 μ Sec/axis	50–75 μ Sec/axis	20kHz

The **MC73110 Motor Control IC** is a single-chip, single-axis device ideal for use in intelligent three-phase brushless DC motor amplifiers. It provides sophisticated programmable digital current control with direct analog input of feedback signals. It can be operated in voltage, torque, or velocity modes. The MC73110 also supports standalone operation for use with PMD's motion processors, other off-the-shelf servo controllers, or via a serial port.

Navigator/Pilot-family Motion Processors provide programmable chip-based positioning control for DC brush, brushless DC, microstepping, and pulse & direction motors. They are available in 1-, 2-, and 4-axis configurations, and in both single-chip and dual-IC chipset configurations.

Magellan Motion Processors are state-of-the-art programmable chip-based positioning controllers for DC brush, brushless DC, microstepping, and pulse & direction motors. They are similar to the Navigator Motion Processors, but provide increased capabilities including faster loop rate, CANBus communications, software-selectable motor type, and direct SPI bus output for serial DACs. They are available in 1-, 2-, 3-, and 4-axis configurations, and in both single-chip and dual-IC chipset configurations.

Magellan PCI and PC/104-bus motion cards are high performance general purpose motion cards for controlling DC brush, brushless DC, microstepping, and pulse & direction motors. Utilizing PMD's Magellan Motion Processors, these products are available in 1-, 2-, 3-, and 4-axis configurations and have advanced features such as 16-bit D/A analog output, and on-board high-speed performance tracing.

ION Digital Drives are compact, fully enclosed modules that provide high performance motion control, network connectivity, and power amplification for DC brush, brushless DC or step motors. Using advanced MOSFETs and surface mount technology, ION drives provide very high power density in a rugged, flexible form factor. They perform profile generation, servo compensation, stall detection, field oriented control, digital torque control and many other motion control functions. These single-axis drives are based on the Magellan Motion Processor and provide CANbus or serial communications.

2. Specifications

In This Chapter

- Configurations, Parameters, and Performance
- Physical Characteristics and Mounting Dimensions

2.1 Configurations, Parameters, and Performance

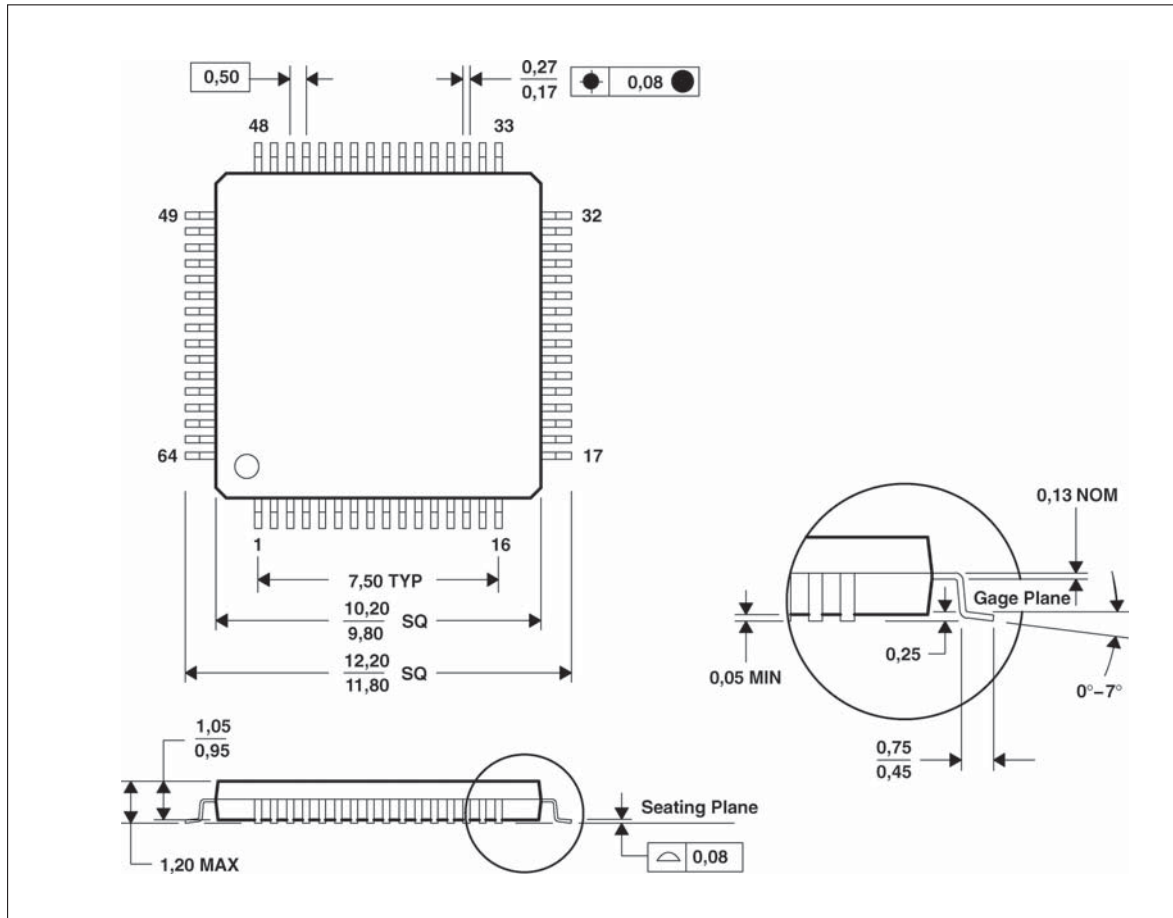
Available configurations	1 axis (MC73110)
Motors supported	3-phase brushless DC
Motor output modes	6-signal high/low digital outputs with dead time protection 3-signal digital outputs
Commutation modes	6-step (with Hall sensors) Sinusoidal (with Hall sensors and quadrature encoder input) FOC (with Hall sensors or Hall sensors and quadrature encoder input)
Current loop rate	20 kHz (19.53 kHz)
Commutation rate	20 kHz (19.53 kHz)
Velocity loop rate	10 kHz (9.766 kHz)
Operating modes	Standalone using serial EEPROM boot or on-board Flash for configuration upload, serial-command-mode (commands sent by host processor)
Serial communication modes	Point-to-point asynchronous Multi-drop asynchronous
Serial baud rate range	1,200 to 460,800
Programmable profile parameters	Velocity (32-bit resolution) Acceleration (32-bit resolution)
Current feedback	Two analog signals (10-bit A/D internal resolution)
Velocity feedback	One analog tachometer signal (10-bit A/D internal resolution), quadrature encoder, or Halls
Velocity/torque/voltage command options	From analog signal (10-bit A/D internal resolution) From digital SPI datastream (16-bit resolution) From serial port (live commands from host processor)
Bus voltage monitor	From analog signal(10-bit A/D internal resolution)
Temp sensor I/O	I ² C bus
Serial EEPROM I/O	I ² C bus
SPI input format	16-bit binary-encoded word
SPI input rate	10 MHz (1.6 μ sec total transmission time)
Input signals	EStop HallA –Hall C PWMOutputDisable
Output signals	AmplifierDisable
Quadrature input signals	A, B, Index
Max quadrature input rate	10 MCounts/sec
PWM resolution	10 bits @ 20 kHz (19.53 kHz) 9 bits @ 40 kHz (39.06 kHz)

PWM output method	Symmetric 3-phase
Internal A to D resolution	10-bit

2.2 Physical Characteristics and Mounting Dimensions

All dimensions are in millimeters.

Figure 2-1:
MC73110
physical
dimensions



3. Electrical Specifications

In This Chapter

- ▶ Absolute Maximum Ratings
- ▶ Recommended Operating Conditions
- ▶ AC Characteristics
- ▶ Timing Diagrams
- ▶ Pin Descriptions
- ▶ Phase Lock Loop

3.1 Absolute Maximum Ratings

Parameter	Rating
Supply Voltage Limits (V_{CC} , $PLL V_{CC}$)	-0.3V to +4.6V
V_{CCP} Range	-0.3V to 5.5V
Input/Output Voltage (V_I)	-0.3V to +4.6V
Operating Temperature: extended (T_a)	-40°C to 125°C
Package Thermal Impedance, θ_{JA} (Junction-to-ambient)	42° C/W
Free-air Temperature Range: Standard (T_a)	-40°C to 85°C
Free-air Temperature Range: Extended (T_a)	-40°C to 125°C
Junction Temperature Range: (T_J)	-40°C to 150°C
Storage Temperature (T_S)	-65°C to 150°C

3.2 Recommended Operating Conditions

(V_{CC} and T_a per operating ratings, either standard or extended temperature, $F_{clk} = 10.0$ MHz)

Symbol	Parameter	Minimum	Maximum	Conditions
V_{CC}	Supply Voltage	3.00V	3.6V	
V_{CCP}	V_{CCP} Supply Voltage	4.75V	5.25V	
I_{dd}	Supply Current		120 mA	all I/O pins floating
F_{clk}	Clock Frequency		10.0 MHz	Nominal

3.2.1 Input Voltages

Symbol	Parameter	Minimum	Maximum	Conditions
V_{ih}	Logic 1 Input Voltage	2.0V	$V_{cc} + 0.3V$	
V_{il}	Logic 0 Input Voltage		0.8V	

3.2.2 Output Voltages

Symbol	Parameter	Minimum	Maximum	Conditions
V_{oh}	Logic 1 Output Voltage	2.4V	V_{cc}	$I_o = -2 \text{ mA}$
V_{ol}	Logic 0 Output Voltage		0.4V	$I_o = 2 \text{ mA}$

3.2.3 Currents and Capacitance

Symbol	Parameter	Minimum	Maximum	Conditions
I_{in}	Input Current	$-30 \mu A$	$2 \mu A$	$V_{in} = 0 \text{ or } V_{cc}$
I_{out}	Tri-state Output Leakage Current	$-2 \mu A$	$2 \mu A$	$V_{in} = 0 \text{ or } V_{cc}$
C_{io}	Input/Output Capacitance	2/3 pF		typical
I_{vccp}	V_{ccp} Input Current		15 mA	

3.2.4 Analog Input

Symbol	Parameter	Minimum	Maximum	Conditions
Analog V_{cc}	Analog Supply Voltage	3.0V	3.6V	The difference between Analog V_{cc} and V_{cc} should be less than 0.3V.
I_a	Analog Supply Current		22 mA	
I_{refhi}	V_{refhi} Input Current		1.5 mA	
Z_{ai}	Analog Input Source Impedance		700 Ohms	
C_{ai}	Analog Input Capacitance		30 pF	typical
E_{zo}	Zero-offset Error		$\pm 2 \text{ LSB}$	typical
E_{dnl}	Differential Nonlinearity Error		$\pm 2 \text{ LSB}$	
	Difference Between the Step Width and the Ideal Value			
E_{inl}	Integral Nonlinearity Error		$\pm 2 \text{ LSB}$	
	Maximum Deviation from the Best Straight Line through the A/D Transfer Characteristics, Excluding the Quantization Error			

3.3 AC Characteristics

See timing diagrams on the opposite page for T_n numbers. The symbol “~” indicates active low signal.

Timing interval	T_n	Minimum	Maximum
Clock frequency (F_{clk})		4 MHz	10 MHz ^a

Timing interval	T _n	Minimum	Maximum
Clock period ^b	T2	100 nsec	250 nsec
Encoder pulse width	T3	150 nsec	
Dwell time per state	T4	75 nsec	
Index setup and hold	T5	0 nsec	
Reset low pulse width	T6	1.0 μ sec	
Device ready/outputs initialized	T7		1 μ sec
Clock High	T8	40 nsec	
Data Hold	T9	50 nsec	
Data Setup	T10	0 nsec	
Clock Rise/Fall	T11		10 nsec
Clock Period	T12	100 nsec	

- a. Performance figures and timing information valid at $F_{clk} = 10.0$ MHz only. For timing information and performance parameters at $F_{clk} < 10.0$ MHz, contact PMD.
- b. The clock low/high split has an allowable range of 40–60%.

3.4 Timing Diagrams

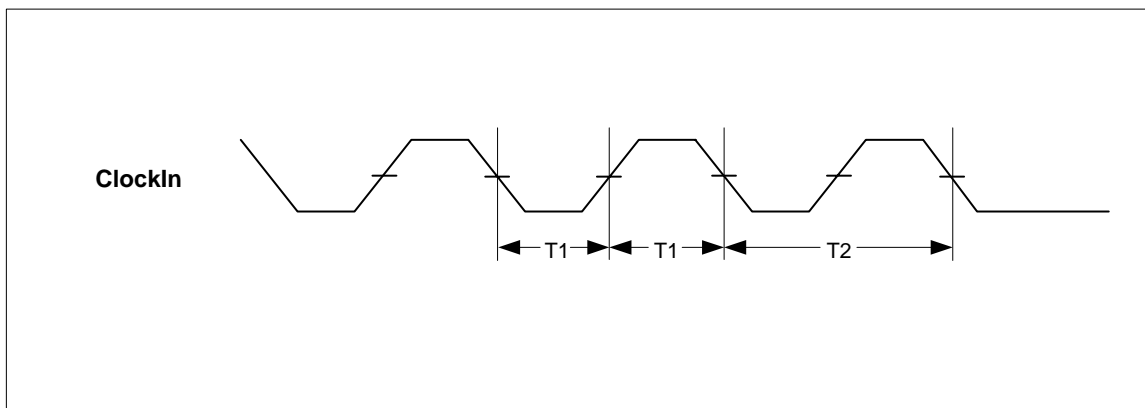


Figure 3-1:
MC73110
Clock timing
diagram

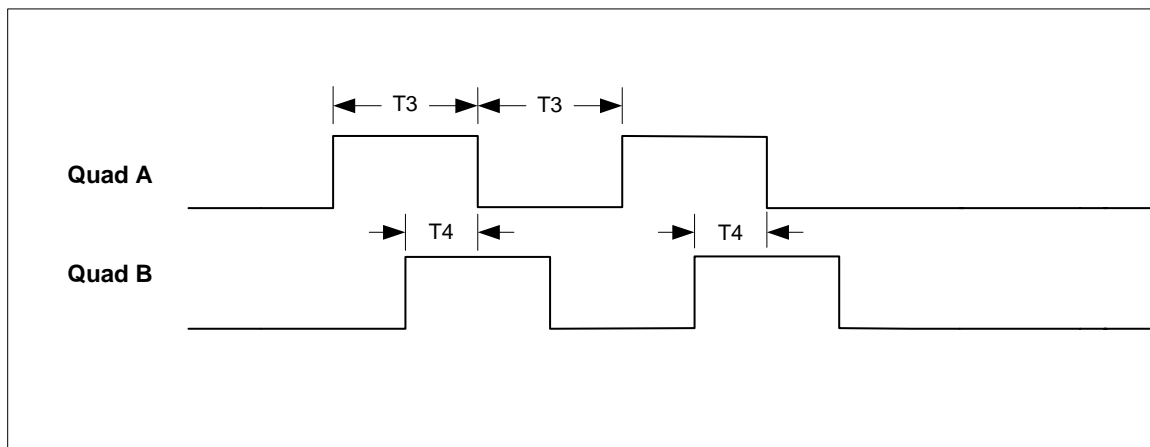


Figure 3-2:
MC73110 Quad
encoder timing
diagram

Figure 3-3:
MC73110
Reset timing
diagram

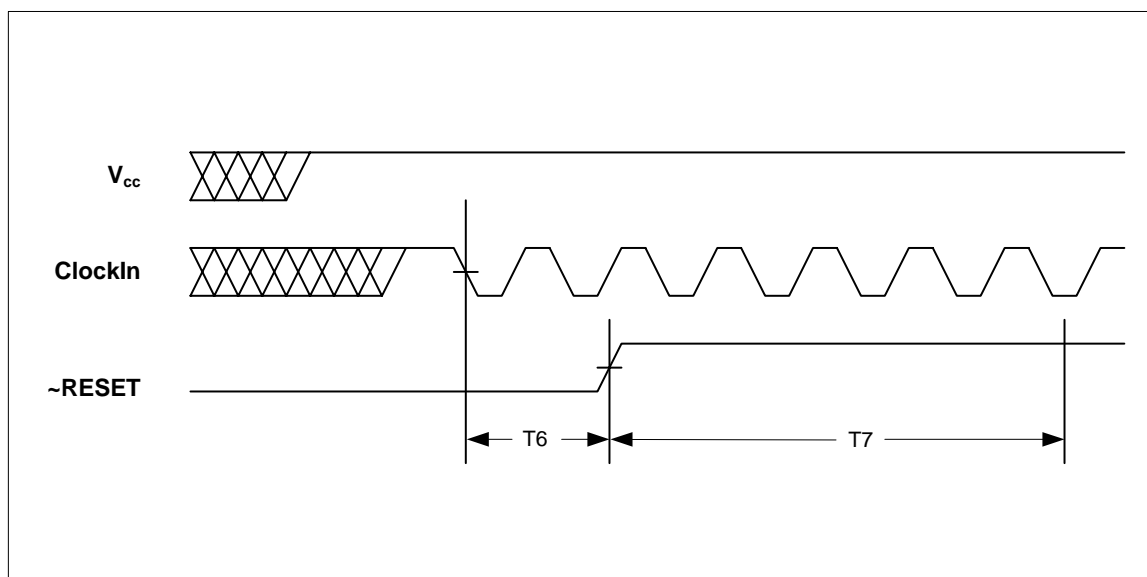
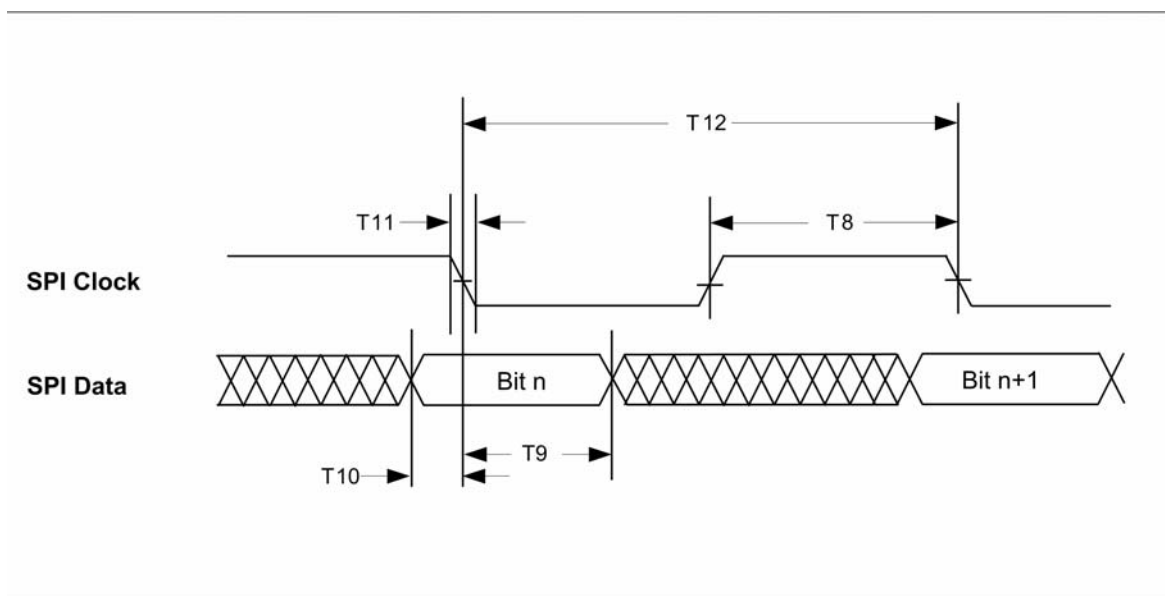


Figure 3-4:
MC73110 SPI
timing diagram



3.5 Pin Descriptions

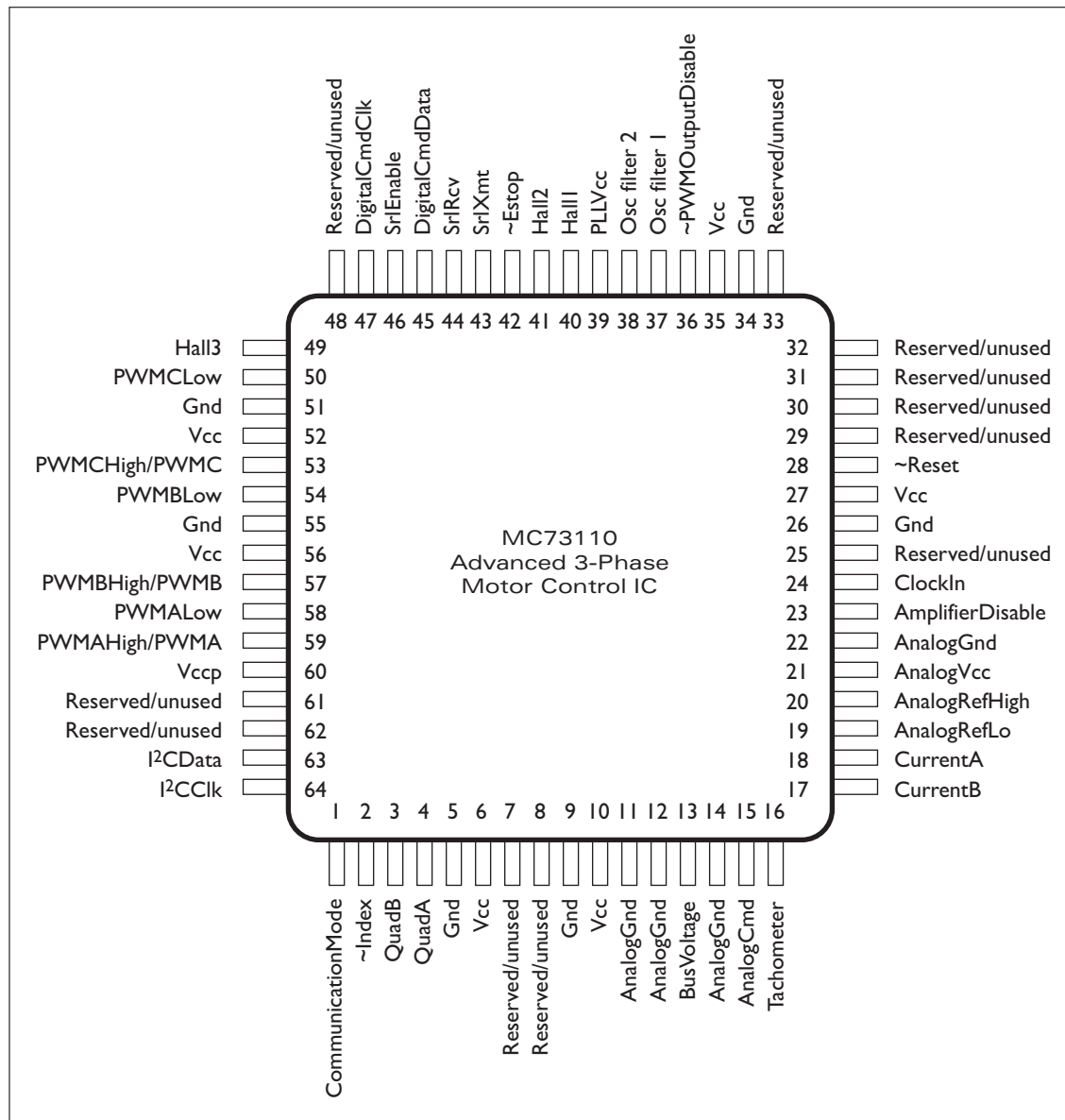


Figure 3-5:
MC73110 chip
pin layout and
descriptions

The functions of the MC73110's pins are defined as follows:

Pin Name	Pin Number	Direction	Description
QuadA	4	Input	These digital signals provide the A and B quadrature input from the QuadB incremental encoder. When the axis is moving in the positive (forward) direction, signal A leads signal B by 90°. NOTE: Many encoders require a pull-up resistor on these signals to establish a proper high signal. Check your encoder's electrical specifications. If not used, these pins may be left unconnected.
QuadB	3		
~Index	2	Input	This digital signal provides the Index signal from the incremental encoder. NOTE: Many encoders require a pullup resistor on this signal to establish a proper high signal. Check your encoder's electrical specifications. If not used, this pin may be left unconnected.
PWMAHigh/ PWMA	59 58	Output	These digital signals provide the Pulse Width Modulated output for each phase to the motor. In 6-signal mode, all 6 signals are used. In 3-signal mode, PWMAHigh, PWMBHigh, and PWMCHigh are used. If not used, these pins may be left unconnected.
PWMALow	57		
PWMBHigh/ PWMB	54 53		
PWMBLow	50		
PWMCHigh/ PWMC			
PWMLow			
Hall1	40	Input	These digital signals provide Hall sensor inputs.
Hall2	41		
Hall3	49		
~Estop	42	Input	This digital signal provides an emergency stop signal that may be used to stop motor output. Unless the default interpretation is changed, an emergency stop condition occurs when this signal is brought low. If not used, this pin may be left unconnected.
Tachometer	16	Input	This analog signal provides optional analog feedback for the motor velocity. After conditioning, this signal is commonly connected to the motor's tachometer. The allowed voltage range is AnalogRefLow to AnalogRefHigh. If not used, this pin should be tied to AnalogGND.
CurrentA	18	Input	These analog signals provide the instantaneous current flowing through coils A and B of the motor. These signals, after conditioning, are commonly connected to the A&B motor coils through a dropping resistor or Hall sensor. The allowed voltage range is AnalogRefLow to AnalogRefHigh. If not used, this pin should be tied to AnalogGND.
CurrentB	17		
AnalogCmd	15	Input	This analog signal provides a command value for either the desired voltage, torque or velocity, depending on how the chip has been programmed. The allowed voltage range is AnalogRefLow to AnalogRefHigh. If not used, this pin should be tied to AnalogGND.
Bus Voltage	13	Input	This analog signal provides the ability to monitor the Bus Voltage. The allowed voltage range is AnalogRefLow to AnalogRefHigh. If not used, this pin should be tied to AnalogGND.
Communication- Mode	1	Input	This digital signal should be tied low at all times through a 10K resistor to the digital ground.
SrIEnable	46	Output	This digital signal sets the serial port enable line. SerialEnable is always high for the point-to-point communication mode, and is strobed high during transmission for the multi-drop protocol.
I ² CData	63	Bidirectional	This digital signal and the I ² CCLK signal comprise an I ² C bus used for inputting amplifier temperature from an I ² C compatible device, and/or an I ² C-compatible serial EEPROM device. If not used, these signals may be left unconnected.

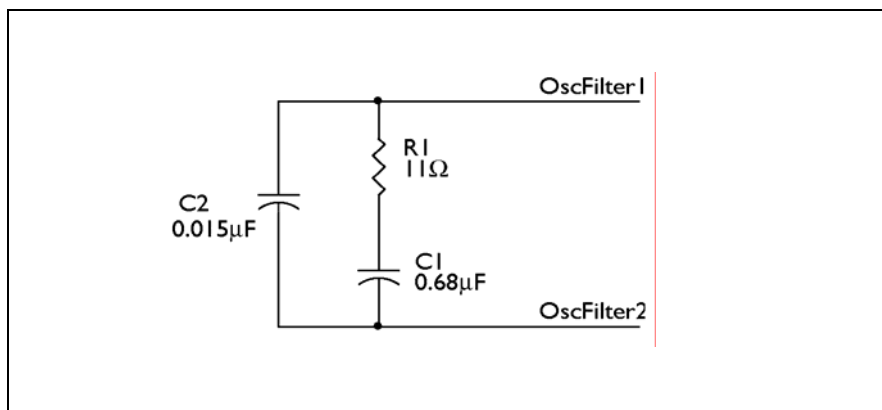
Pin Name	Pin Number	Direction	Description
I ² CCLK	64	Output	This digital signal and the I ² CData signal comprise an I ² C bus used for inputting amplifier temperature from an I ² C compatible device, and/or an I ² C compatible serial EEPROM device. If not used, these signals may be left unconnected.
DigitalCmdClk	47	Input	These digital signals encode a 16-bit digital command containing the desired voltage, torque or velocity, depending on how the chip has been programmed. These signals are encoded using an SPI word format, with one line providing clock information (DigitalCmdClk), and the other providing data (DigitalCmdData). If not used, these signals may remain unconnected.
DigitalCmdData	45		
SrIXmt	43	Output	This digital signal transmits serial data to the asynchronous serial port.
SrIRcv	44	Input	This digital signal inputs serial data from the asynchronous serial port. If not used, this signal may remain unconnected.
AmplifierDisable	23	Output	This digital signal provides a general purpose output which can be programmed for a variety of internal conditions of the chip. It is most commonly used to control external amplifier circuitry in the event that a condition such as overtemperature or a motion error occurs. The sense of this signal is active high. That is, this signal is normally low, and transitions high upon the occurrence of the programmed special event. NOTE: This signal must not be pulled down. It should be either connected to a high-impedance input or pulled up to 3.3V with a 10K resistor.
~PWMOutput-Disable	36	Input	This digital signal directly controls the PWM output circuitry. When this signal is high the PWM output of the chip is enabled. When it is low, PWM output is disabled, and all PWM output signals are tri-stated. If not used, this signal may remain unconnected.
AnalogRefHigh	20	Input	These analog signals provide the high voltage reference and the low voltage reference value used to define the allowed range of voltage for the pins Velocity, CurrentA, CurrentB and AnalogCmd. The recommended value of AnalogRefHigh is between 2.0V and AnalogV _{cc} . The recommended value of AnalogRefLow is AnalogGND. The voltage change on both pins shall be smaller than half of the LSB of the target resolution.
AnalogRefLow	19		
AnalogV _{cc}	21	Input	This signal provides power to the analog portion of the chip's circuitry. It should be connected to a 3.3V supply. It is recommended that this (analog) power supply be isolated from digital power supply V _{cc} in order to ensure noise immunity and meet the specified A/D performance. The recommended operating range is from 3.0V to 3.6V, with a nominal value of 3.3V. The difference between AnalogV _{cc} and V _{cc} should not exceed 0.3V.
AnalogGND	11, 12, 14, 22	Input	These signals provide the return for the analog portion of the chip's circuitry. It should be connected to the analog return. It is recommended that this (analog) power return be isolated from digital power return in order to ensure noise immunity, and to meet the specified A/D performance.
ClockIn	24	Input	This is the master clock signal for the chip. It is nominally driven at 10 MHz.
~Reset	28	Input	This digital signal is used to reset the chip. When brought low, this pin resets the chip to its initial conditions. This pin must be high for normal operation. Refer to Figure 3-3 on page 16 for timing requirements.
V _{cc}	6, 10, 27, 35, 52, 56	—	These signals provide power to the digital portion of the chip's circuitry. They should be connected to a 3.3V supply.
GND	5, 9, 26, 34, 51, 55	—	These signals provide the return for the digital portion of the chip's circuitry. They should be connected to the digital return.

Pin Name	Pin Number	Direction	Description
V _{ccp}	60	—	This signal provides 5V to the internal Flash programming circuitry of the chip. If it is desired that the chips' startup configuration be stored in the chip's internal Flash memory, 5V must be provided at this pin. Otherwise, this pin must be connected to the digital return.
— (Reserved/ unused)	7, 8, 25, 29, 30, 31, 32, 33, 48, 61, 62	—	These pins should remain unconnected.
Osc filter1	37	—	These signals form a PLL (phase lock loop) circuit. See Section 3.6, "Phase Lock Loop (PLL)," on page 20 for more information.
Osc filter2	38	—	
PLL _{V_{cc}}	39	—	This signal provides the V _{cc} for the phase locked loop circuit. It should be connected to a 3.3V supply.

3.6 Phase Lock Loop (PLL)

The circuit in Figure 3-6 shows the recommended configuration and suggested values for the filter that must be connected to the OscFilter1 and OscFilter2 pins of the chip. The resistor tolerance is $\pm 5\%$, and the capacitor tolerance is $\pm 20\%$. Unpolarized capacitors must be used.

Figure 3-6:
PLL circuitry
design



4. Theory of Operations

In This Chapter

- ▶ Functional Overview
- ▶ Internal Block Diagram
- ▶ Connection Summary
- ▶ Control Loop Overview
- ▶ Motor Output and Signal Generation
- ▶ Current Loop
- ▶ Commutation
- ▶ Field Oriented Control (FOC)
- ▶ Velocity Loop
- ▶ Velocity Integrator
- ▶ Profile Generation
- ▶ Loop Rate
- ▶ Status Words
- ▶ Programmable Conditions
- ▶ Temperature Sensor
- ▶ Bus Voltage Sensor
- ▶ Serial Port
- ▶ Incremental Encoder Input
- ▶ Serial EEPROM
- ▶ Synchronous Serial Input (SPI Port)
- ▶ Analog Signal Processing
- ▶ GetLoop Commands and Variables

4.1 Functional Overview

The MC73110 Motor Control IC is a single-axis device for velocity, torque, or voltage-mode control of three-phase brushless DC motors. It can perform a number of functions including three-phase PWM signal generation, commutation, current loop, velocity loop, profile generation, Hall sensor input, quadrature encoder input, emergency stop processing, serial port command I/O, synchronous serial SPI data input, direct analog signal input, I²C temperature sensor input, and automatic configuration upload via serial EEPROM.

At power-up or reset, the MC73110 checks for the presence of a serial EEPROM at the I²C interface. If a serial EEPROM is present, the stored configuration commands are read into the chip, providing parameter information that will be used during operation. See Section 4.19, “Serial EEPROM,” on page 57 for more information on serial EEPROM processing. Alternatively, configuration information may be stored in the MC73110’s Flash memory. If no initial configuration is stored in Flash or is provided by serial EEPROM, then default values are used, and information will then be sent by serial port commands from a host device such as a microprocessor or PC. See Section 4.17, “Serial Port,” on page 51 for more information on serial port command processing. Depending on how the control loop has

been configured, an external analog signal may serve as the velocity or torque set point values. Alternatively, a synchronous serial (SPI) data stream may also be used for this command value, or the internal profile generator can be used.

Current loop control is performed via direct input of two analog signals representing the instantaneous current through the A and B motor coils. These signals are typically derived from external dropping resistors or Hall sensors at the amplifier circuitry. This analog current information is then combined with the desired current for each phase to generate symmetric 6-signal or 3-signal PWM signals. See Section 4.5, “Motor Output and Signal Generation,” on page 27 for more information on motor output.

To create a complete motion controller, the MC73110 is connected to three half-bridge amplifiers, typically MOSFET or IGBT-based. A programmable dead time function ensures that adequate off-time is provided during state switching of each motor coil.

A number of safety features are incorporated into the MC73110 including direct Estop (emergency stop) signal input, PWM output disable, and an amplifier disable output signal which can be used to enable and disable the external amplifier circuitry. See Section 4.14, “Programmable Conditions,” on page 47 for more information on emergency stop and related functions.

4.2 Internal Block Diagram

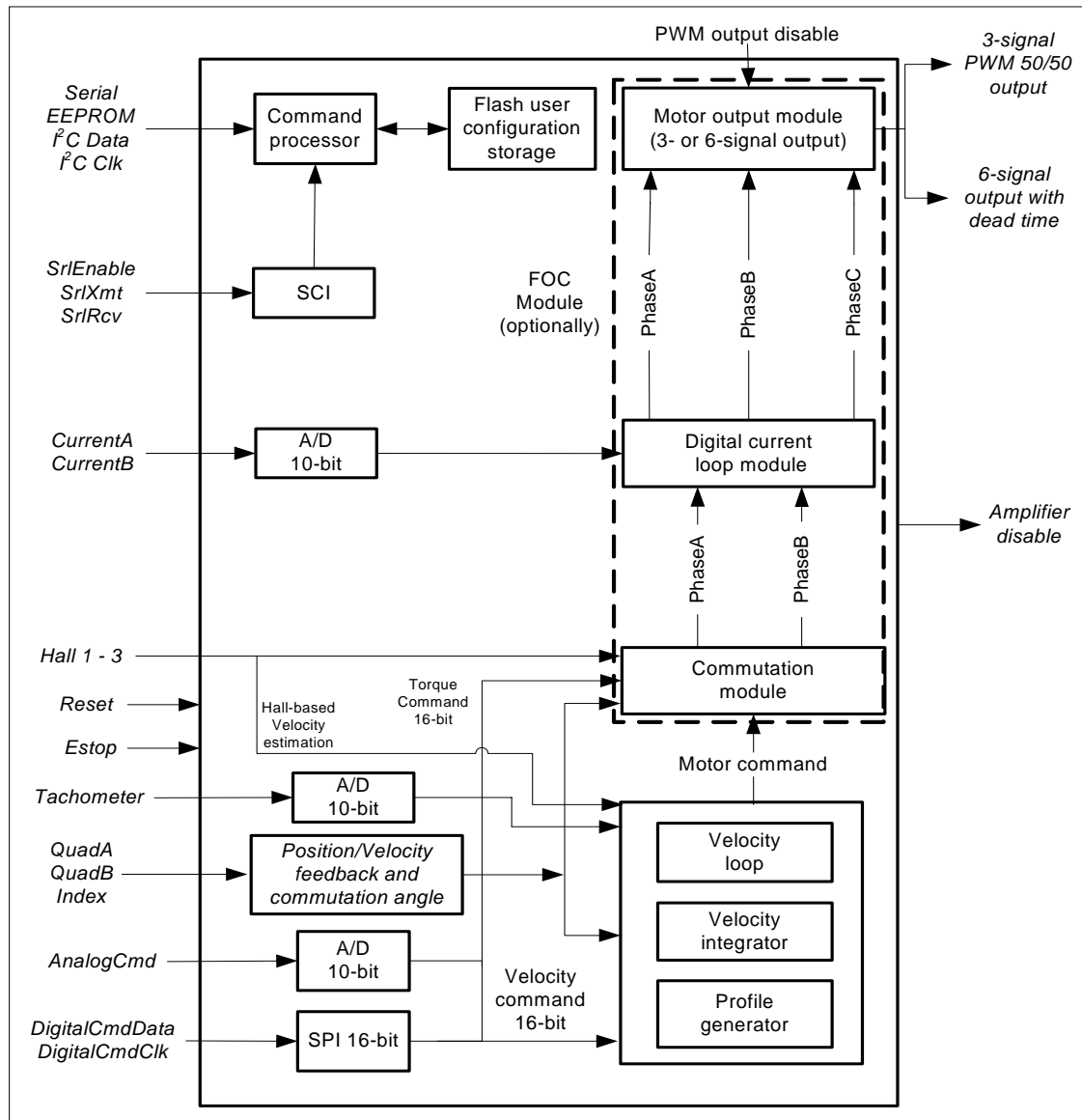


Figure 4-1:
MC73110
internal block
diagram

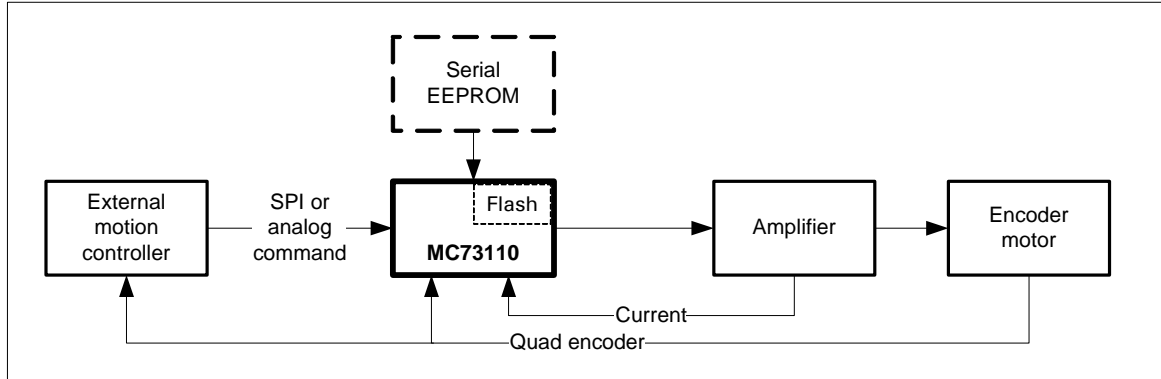
4.3 Connection Summary

The MC73110 can be used in one of two connection modes: either as a dedicated motion controller used in conjunction with an external controller such as a PMD positioning motion processor or motion control card, or as a complete intelligent motion controller driven by serial port commands. This is illustrated in Figure 4-2 on page 24 and Figure 4-2 on page 24.

When using the MC73110 in conjunction with an external motion controller, a continuous torque or velocity command in either analog or 16-bit SPI format is provided by the external motion controller. The MC73110 provides current control, commutation, and velocity loop functions to control the motor per the command provided by the external controller. In this connection configuration, operational information such as gain factors and other values

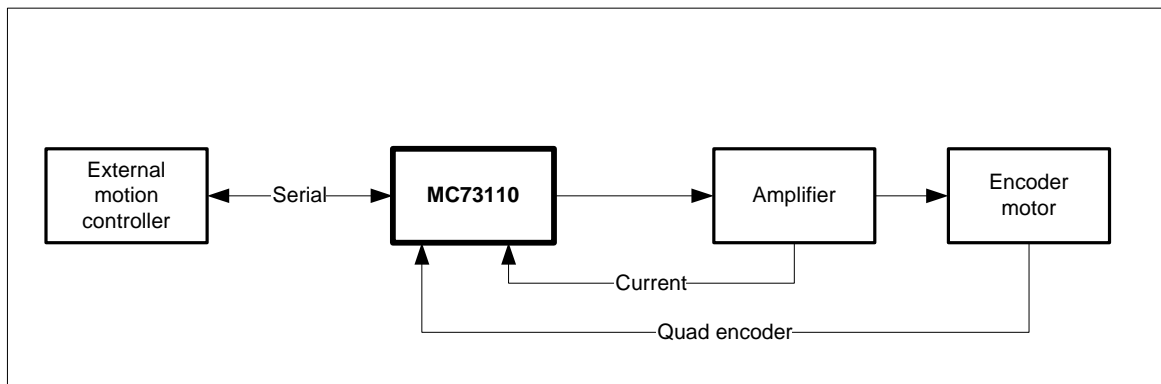
required by the chip are stored in an external serial EEPROM which is automatically loaded upon reset, or in the Flash memory of the MC73110.

Figure 4-2:
With an external
motion
controller



Using the MC73310 as a complete intelligent motion controller, a PC or host microprocessor sends profile commands via the serial port, and the MC73110 responds by moving the axis along the desired profile, using the programmed velocity and acceleration. In this application, gain factors and other values required by the chip are usually sent by the host via the serial port after power-up.

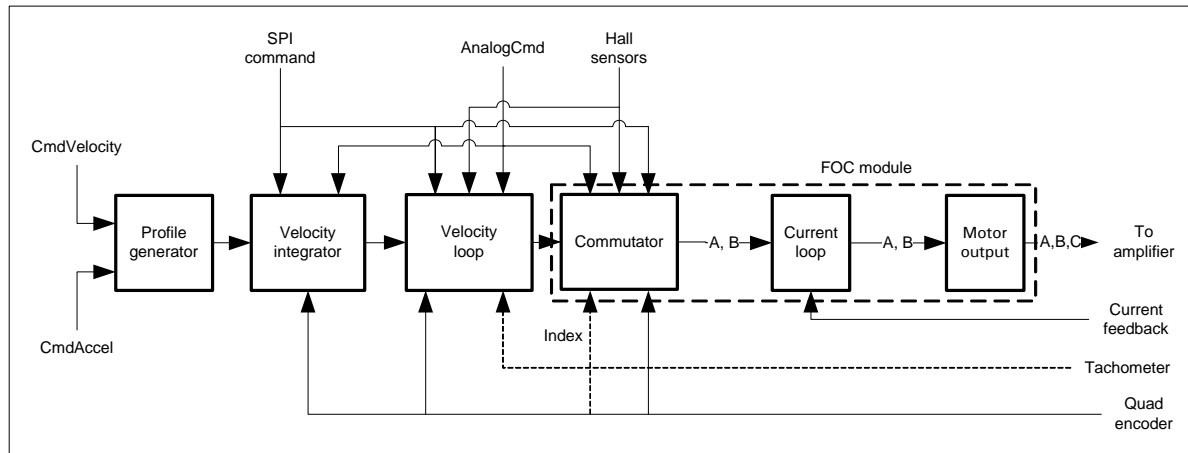
Figure 4-3:
As a complete
intelligent
motion
controller



4.4 Control Loop Overview

This section summarizes the elements in the main control loop, beginning with those elements closest to the motor. Figure 4-4 on page 25 shows the overall control loop flow of the MC73110.

Figure 4-4:
Control loop
flow



Motor Output Module—This module inputs the desired voltage for each of the three motor coils, and generates the correct PWM signal in either 3-signal mode (one signal per phase), or 6-signal mode (high & low signal for each phase). The output signals are symmetric in waveform, and synchronized to the master PWM output which occurs at 20kHz frequency. The output signals are presented on PWMAHigh, PWMALow, PWMBHigh, PWMBLow, PWMCHigh, and PWMCLow.

Current Loop Module—This module inputs the desired current for each of the two motor coils, and uses two analog feedback signals (CurrentA, CurrentB) to develop a PWM output value for each motor connection. The current loop module may be disabled, in which case the MC73110 will drive the motor in voltage mode. See Section 4.5, “Motor Output and Signal Generation,” on page 27 for more information on the motor output logic. See Section 4.6, “Current Loop,” on page 30 for more information on the current loop module.

Commutator Module—This module accepts a single-phase desired torque or voltage command (depending on whether the Current Loop Module is enabled), and vectorizes this command into three phased commands, one for each motor connection. Two commutation methods are supported: Hall-based, and sinusoidal. If Hall-based is selected, then the Hall signals must be connected through Hall1, Hall2, and Hall3. If sinusoidal is selected, in addition to the Hall signals, quadrature encoder data must be connected through the signals QuadA and QuadB. This module cannot be disabled. See Section 4.7, “Commutation,” on page 32 for more information on commutation.

FOC Module—Optionally, the commutation, digital current control, and motor output modules can be replaced with FOC (Field Oriented Control). This provides Current Control in the de-referenced (D,Q) frame, and utilizes space vector PWM for the motor output.

The Velocity Loop Module—This module accepts a desired velocity command, and combines this with the instantaneous velocity of the motor axis to determine a desired torque or voltage command. The desired velocity command can come from either the profile generator module, the velocity integrator module, an external analog signal (AnalogCmd), or a synchronous serial (SPI) digital 16-bit word data stream, which is encoded on two signals: DigitalCmdClk, and DigitalCmdData. The instantaneous velocity can come from an analog signal through the velocity pin of MC73110, or via the quadrature encoder or Hall sensors. This module may be disabled, in which case the chip operates in torque or voltage mode, depending on the state of the current control module. See Section 4.8, “Field Oriented Control (FOC),” on page 35 for more information on the velocity loop.

The Velocity Integrator Module—This module accepts a desired velocity from the profile generator, an external signal (AnalogCmd), or the SPI data stream (DigitalCmdClk, DigitalCmdData), and integrates this value into an instantaneous desired position, which is compared with the actual position from the encoder to develop an output desired velocity, torque, or voltage. This module may be disabled, in which case the desired velocity from the profile generator will be fed directly into the velocity loop. See Section 4.10, “Velocity Integrator,” on page 40 for more information on the velocity integrator module.

The Profile Generator Module—This module uses serial port commands to generate a velocity and acceleration-bounded profile. The instantaneous desired velocity is output to the velocity integrator or the velocity loop, depending on which of these modules are enabled. See Section 4.11, “Profile Generation,” on page 42 for more information on the profile generator module.

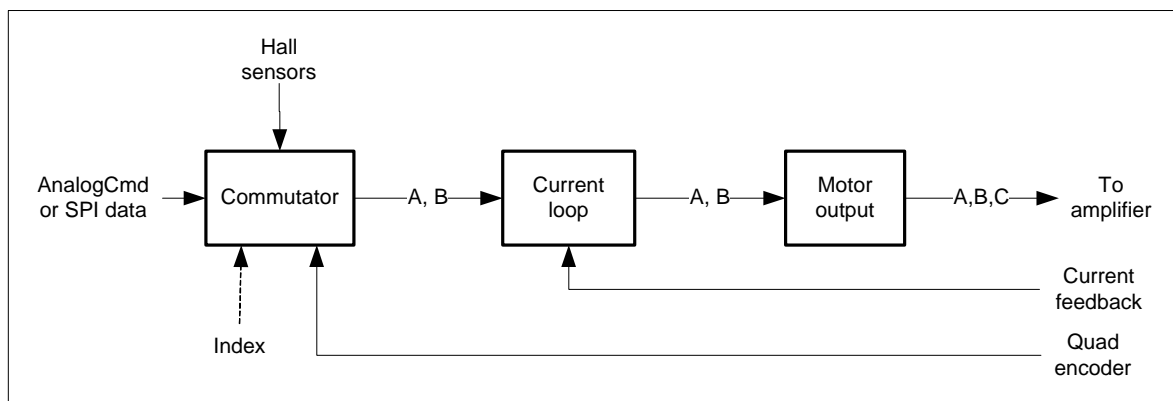
4.4.1 Typical Control Applications

Although the MC73110 control loop structure is very flexible, and may be programmed in a number of ways, most applications fall into one of three standard loop configurations. These are summarized in the following table.

Name	Modules Enabled	Minimal Connections	Comments
Torque-mode amplifier	Current loop	Hall I–3 CurrentA/B QuadA/B (if commutating sinusoidally) DigitalCmdClk/Data or AnalogCmd	This configuration is typical of a number of applications including a torque-mode amplifier used in conjunction with an external position controller. The MC73110 accepts a continually changing torque command, represented as an analog signal or as an SPI data stream, commutates this signal into 3 phases, and drives the motor at those torque values using analog current signals from the motor.
Velocity-mode amplifier	Current loop Velocity loop	Hall I–3 CurrentA/B QuadA/B (if commutating sinusoidally) DigitalCmdClk/Data or AnalogCmd	Similar to a torque-mode amplifier, this configuration adds a velocity loop, so that the continually changing command is a velocity command rather than a torque command. The MC73110 inputs this command, performs a velocity loop on this command using either an analog signal, encoder data stream, or Hall sensors for velocity feedback. The signal is then commutated into three phases, and drives the motor at those torque values using analog current signals from the motor.
Intelligent motion controller	Current loop Velocity integrator Profile generator	Hall I–3 CurrentA/B QuadA/B SrlXmt/Rcv	This control loop configuration is common when the chip will be used as an intelligent programmable motion controller via serial port commands.

Figure 4-5 and Figure 4-6 on page 27 and Figure 4-7 on page 27 illustrate the three control application configurations.

Figure 4-5:
Configuration
for a torque-
mode amplifier



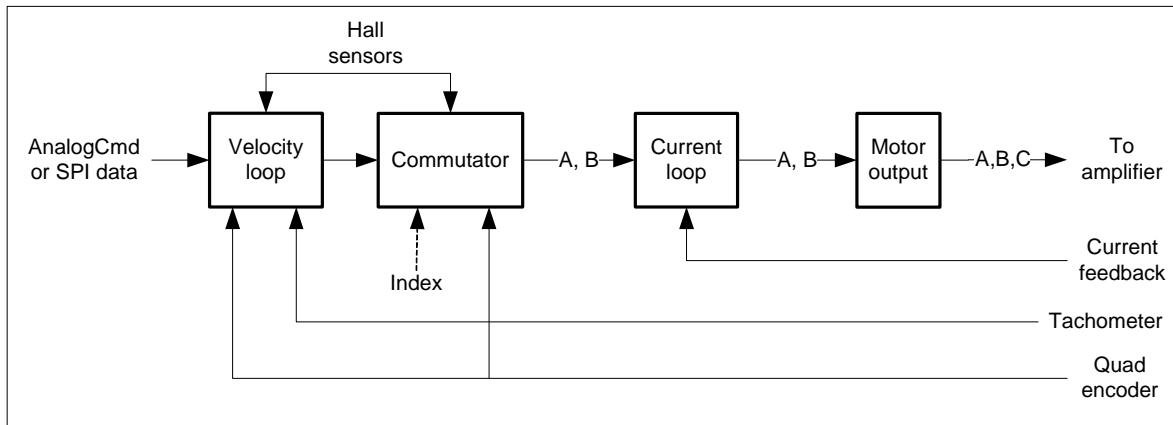


Figure 4-6:
Configuration
for a velocity-
mode amplifier

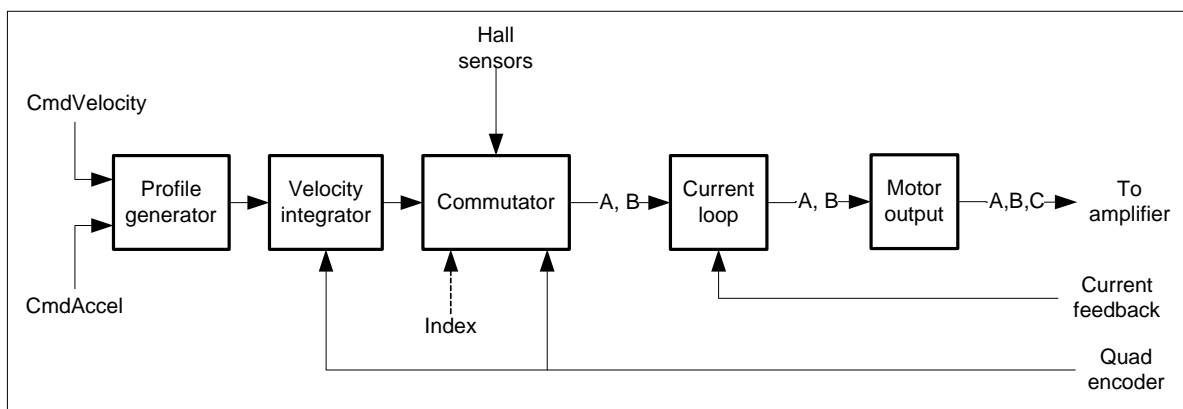
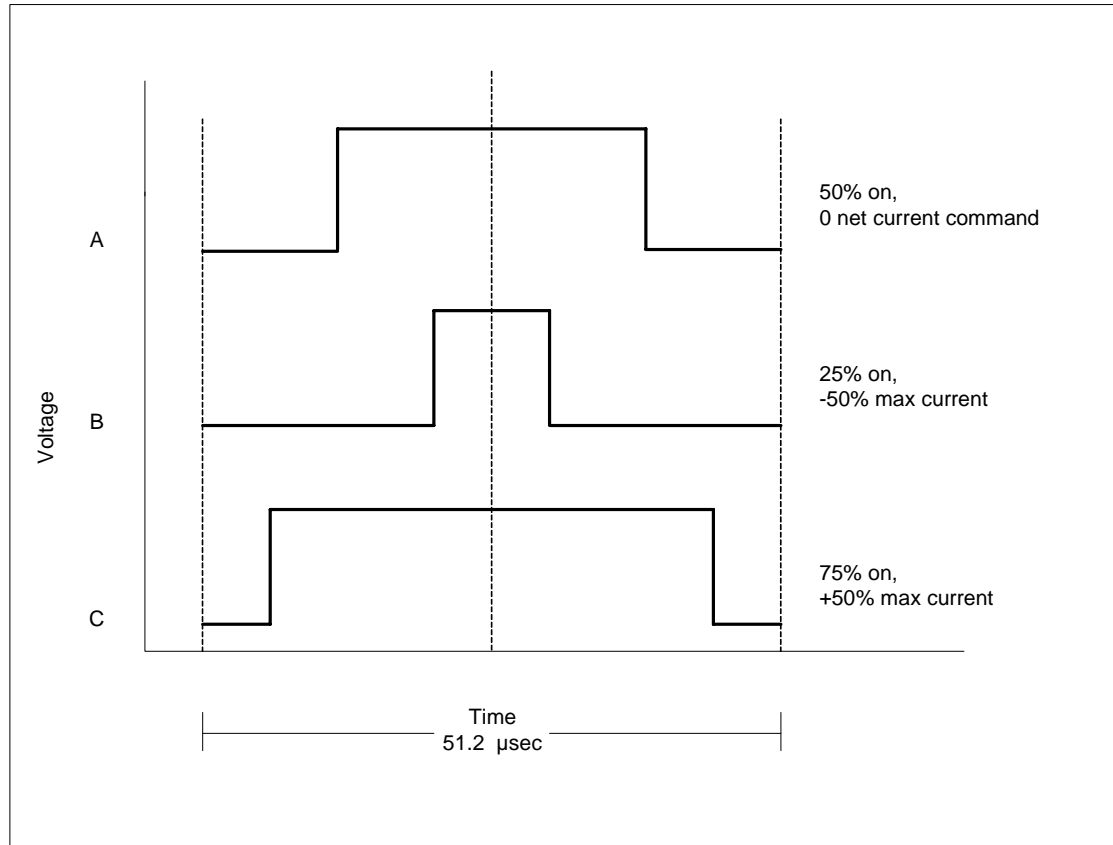


Figure 4-7:
Intelligent
motion
controller
configuration

4.5 Motor Output and Signal Generation

The MC73110's motor output logic accepts three 16-bit motor coil voltage commands (one for each coil), and generates symmetric synchronized PWM signals output on dedicated hardware pins. Two signal generation modes are supported: six-signal output, and three-signal output mode. The command **SetPWMOutputMode** controls which of these two modes are used. Figure 4-8 on page 28 shows typical PWM waveforms.

Figure 4-8:
PWM
waveforms and
currents



During normal operations, a new desired coil voltage is determined at each cycle of the PWM update frequency. These new output values are applied in synchrony at the start of each PWM cycle. If the current loop is active, these voltage values are determined after the analog current has been input, the current control filter has been calculated, and the new value generated. If the current loop is not active, the values are derived directly from the output of the commutator.

4.5.1 Signal Representation

To read the output PWM signal for each phase, the command **GetPWMCommand** is used. The returned value is a 16-bit signed value. The actual PWM generator uses the top ten (for 20kHz PWM) or nine (for 40kHz PWM) bits of this value level-shifted to a 50/50 PWM output value to create the final waveform. For example, a desired output value of zero results in a waveform that is active 50% of the time, and inactive 50% of the time. A desired output value of $+1/2$ total output is active 75% of the time, and inactive 25% of the time, etc. Note that all of these waveforms are symmetric to minimize torque ripple during switching. Selection of 20kHz or 40kHz PWM output frequency is done using the **SetPWMFrequency** command.

4.5.2 Six-Signal Mode

The output of six-signal mode provides separate high/low bridge drive signals for each of the three phases, six signals in all. The PWM signals are output on the pins: PWMAHigh, PWMALow, PWMBHigh, PWMBLow, PWMCHigh, and PWMCLow. When operating in this mode, a programmable dead time is active. This feature allows the user to program an interval between successive high/low or low/high turn-on sequences for a given phase. This is often an important requirement to avoid excessive current flow between the upper and lower switching elements of the amplifier. To determine the correct minimum dead time, consult the specifications for your switching IC or circuit. Six-signal and 3-signal modes are illustrated in Figure 4-9 on page 29.

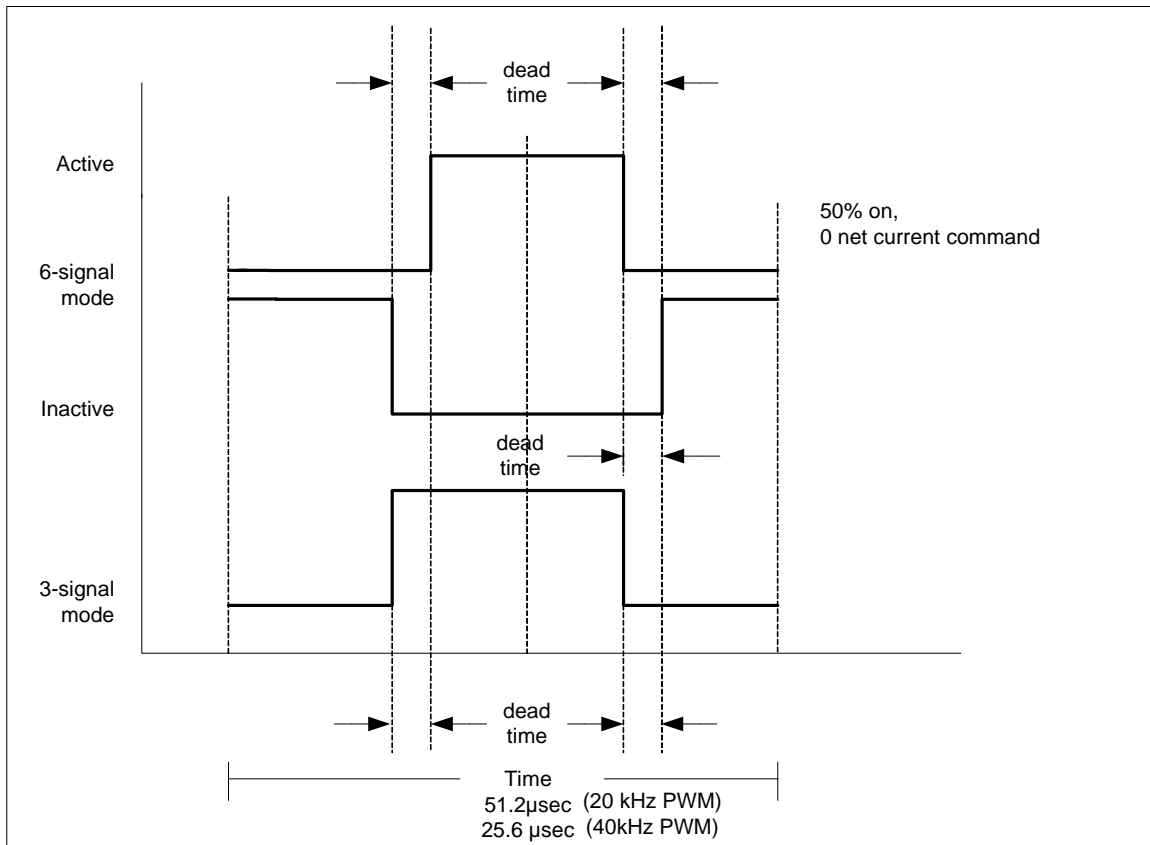


Figure 4-9:
Six-signal
mode with
dead time delay

The programmed dead time delay affects all phases. The dead time delay is programmed using the command **SetPWMDeadTime**, and can be read back using the command **GetPWMDeadTime**.

A special six-signal mode is supported when using Hall-based commutation (not FOC). In this mode, one phase of the motor (based on the Hall states) is always left floating by keeping both the upper and lower drive signals in the off state. This mode can result in improved efficiency and performance in some applications. This mode is selected using **SetPWMOutputMode**, and can be used with or without a digital current loop.

4.5.3 Three-Signal Mode

The three-signal output mode provides one signal per phase. The dead time timer does not function when the chip is set to this mode. Each of these three signals is encoded such that a high value means the high side of the half bridge should be turned on, and a low signal means the low side should be turned on. Note that when using this mode, if shoot-through protection off-times are required, this must be arranged using external circuitry provided by the user.

In this mode, the PWM signals are output on the following pins: PWMAHigh, PWMBHigh, and PWMCHigh.

4.5.4 Maximum On-Time

Many amplifiers have a requirement for a minimum off-time to allow bootstrap capacitors to recharge or for other reasons related to the specific amplifier circuit chosen. To accommodate this, the maximum output value for each phase can be programmed using the command **SetPWMLimit**. This value can be read using the command **GetPWMLimit**.

The limit value specified is a 16-bit number representing the maximum value that can be output to the PWM generation circuitry. For example, if the limit is specified as +31,500, positive values will be clipped should they exceed

+31,500 and negative values will be clipped should they be less than -31,500. Note that this represents approximately 96% ($31,500/32,767$) maximum on time, or 4% guaranteed minimum off-time.

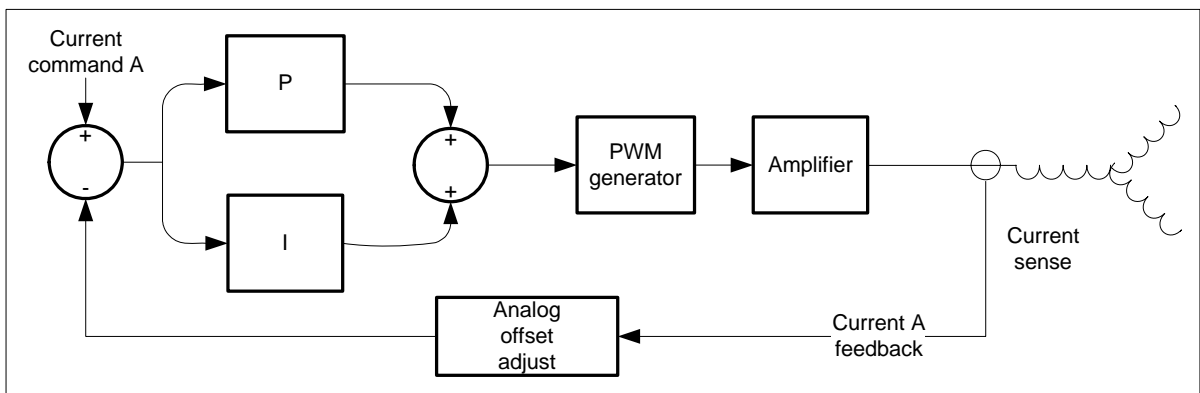
4.6 Current Loop

The MC73110 includes a sophisticated digital current controller that utilizes analog feedback signals to match the actual motor currents with the desired current through each coil. The two desired motor current values are provided by the commutator module. The actual motor currents are provided on the signals CurrentA and CurrentB.

To read the current values for each phase, the command **GetLoopCommand** is used. The returned value is a signed 16-bit number encoded such that +32,767 represents a request for maximum positive current output, and -32,768 represents maximum negative current output.

Figure 4-10 shows a typical setup for sensing of the motor currents.

Figure 4-10:
Motor current
sensing



For the current loop to function correctly, two analog signals, CurrentA and CurrentB, are expected to be provided to the chip. These signals are positive voltage referenced; that is, negative, current values at the coil are input to the chip as a positive voltage. See Section 4.21, “Analog Signal Processing,” on page 60 for details.

The current loop operates in synchrony with the PWM output generator. Utilizing high speed on-chip A/D converters, measurements of the current are taken at each PWM cycle. The current loop then performs a PI (proportional, integral) filter calculation to determine the desired voltages, which are then fed to the PWM circuitry. The current loop is calculated for coils A and B, with C being calculated from $C = -(A+B)$.



The current loop can be disabled using the command **SetLoopMode**. This value can be read using **GetLoopMode**. If the current loop is turned off, then the values from the commutator for desired A and B commands are passed through unmodified to the PWM generator output logic.

4.6.1 Current Loop Filter

Three values must be set by the user: $K_{p_current}$, $K_{i_current}$, and $IntegrationLimit_{current}$. These parameters are set and read using the commands **SetLoopGain** and **GetLoopGain**, respectively.

The result of this calculation is a 16-bit number for each of the A and B phases. These values are then output to the PWM generator circuitry. The exact filter equation for each phase is as follows.

$$CE = \text{CurrentDesired} - (\text{CurrentActual} + \text{Offset})$$

$$\text{CurrentOutput}_n = CE_n \times \text{CurrentKp}/64 + \sum_{j=0}^n CE_j \times \text{CurrentKi}/256$$

where:

CE	is the current error term
CurrentDesired	is the output of the commutator for either phase A or phase B
CurrentActual	is the value input at CurrentA or CurrentB
Offset	is the offset parameters stored using SetAnalogOffset
CurrentOutput	is the output from the current loop
CurrentKp	is the current proportional gain
CurrentKi	is the current integral gain

When the current loop is disabled, or when the motor mode is off, then the integrator remains at zero (0).

4.6.2 Current Signal Input

The analog current signals are input on the CurrentA and CurrentB signals of the MC73110 IC. To read these values, the command **GetAnalog** is used. The returned value is a signed 16-bit number representing the current represented by the voltage presented on the analog input pins. See Section 4.21, “Analog Signal Processing,” on page 60 for details on converting analog values to numeric values and vice-versa.

4.6.3 Current Signal Offset Bias

A special feature of the MC73110 is that a software offset bias can be introduced after the A/D conversion of the analog input signals. This may be useful to zero-reference the analog circuitry without the need for manually adjusted potentiometers. To add a bias to the CurrentA and CurrentB values read by the chip, the command **SetAnalogOffset** is used. To read this value, the command **GetAnalogOffset** is used.

The offset value is added to the value read by the chip to determine the value used during current loop calculations. A simple way of determining the current offset of each signal is to disable the amplifier output, and read the analog signals using **GetAnalog**. Since the amplifier is turned off, the read value should indicate zero current. This value can then be negated and entered as the offset to correctly zero-reference that analog input. For example, if the value read is -75 , the offset value should be set as $+75$.

Due to temperature changes and other factors, the ideal offset value may change during operation and over the lifetime of product usage. It is the responsibility of the designer to ensure that the MC73110 and any associated amplifier circuitry is operated within safe limits.



4.6.4 Feedback Signal Scaling Considerations

The MC73110 provides general-purpose current control features that can be used with a wide variety of amplifiers and over a wide range of power ratings. It is important to insure that the overall range of expected currents driven in the motor coils matches the overall operational range for which the feedback circuit is scaled.

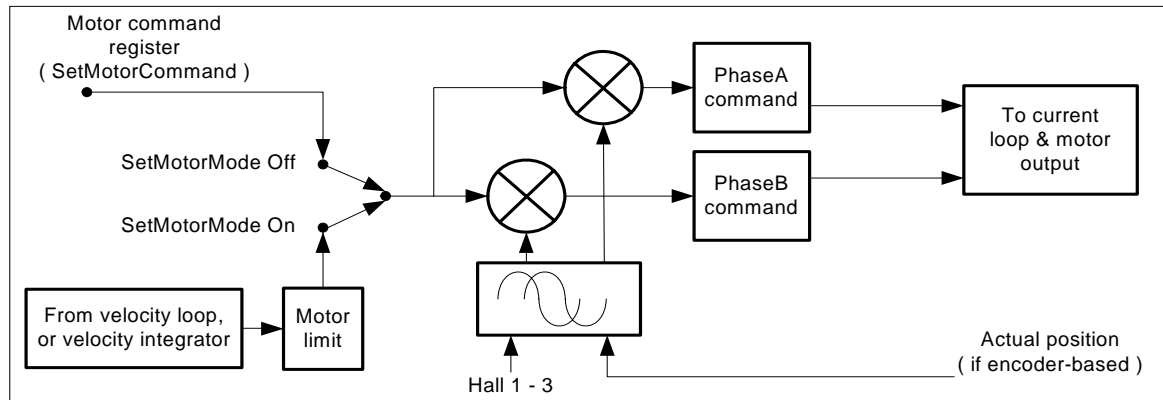
When determining overall sensitivity, some over-range current capacity beyond the steady-state operating values should be included. In most applications, 30–70% is advised. For example, if the maximum steady state current of the motor is designed to be 10 amps, it is advisable to scale the feedback circuit to allow a total range of $\sim \pm 15$ amps. In

this application, this would mean that the current sense circuitry would have a sensitivity of maximum current/(3.3V/2) or 9.1 A/V (amps per volt).

4.7 Commutation

Figure 4-11 shows an overview of the control flow of the sinusoidal commutation portion of the MC73110 IC.

Figure 4-11:
Sinusoidal
commutation



For input, the commutation portion of the IC uses the command value from the velocity loop. The first source is the velocity integrator loop, or profile generator (depending on which of these modules are enabled). The second source is a manually-set register known as the Motor Command register. The MC73110 accepts the command value from the velocity loop and/or upstream components when the motor mode parameter is set to **On**. Conversely, the manual register, which is loaded by the user, is used as motor command input when the motor mode is set to **Off**.

4.7.1 Motor Command Register

To set the motor mode, the command **SetMotorMode** is used. To read this value, the command **GetMotorMode** is used. To set the Motor Command register, which allows the user to write output commands directly to the motor, the command **SetMotorCommand** is used. Operating the IC in manual mode can be useful for calibrating the amplifier circuitry. For example, by setting the motor mode off, write commands to the motor register can be used to set an exact value of 5000, or 10,000 to the PWM output logic. This can be useful for running the motor at a fixed voltage value, or to measure the controlled response of the system to changes in the motor command.

The motor mode is also important during recovery from a motion error, if this facility is used. The MC73110 has the ability to compare the velocity or position error against a user-defined window, and enter a motion error condition if this value is exceeded. When this error occurs (assuming **AutoStopMode** is enabled), motor mode is set to **Off**. This feature is described in more detail in Section 4.9.5, “Motion Error Detection,” on page 39.

4.7.2 Motor Command Limiting

The MC73110 allows the maximum value accepted by the commutator to be set. This motor limit value is set using the command **SetMotorLimit**. It can be read using the command **GetMotorLimit**.

The specified motor limit affects the value input into the commutator such that if the magnitude of the input motor command exceeds the motor limit, then the output value is maintained at the motor limit value. For example, if the

value of the motor command from the velocity loop is –32,500, and the value of the motor limit has been set to 30,000, then the value used by the commutator will be –30,000.

The motor limit is only applied when the motor is “On,” that is, when the value set using **SetMotorMode** is “On.” If the motor mode is off, the value set using **SetMotorCommand** will be applied regardless of whether it exceeds the motor limit.



4.7.3 Sinusoidal Commutation

The pre-commutated motor command is vectorized into three phased signals using either a sinusoidal lookup technique, or a Hall-based technique. To set the commutation mode, the command **SetCommutationMode** is used. To read this value, the command **GetCommutationMode** is used.

If sinusoidal commutation is selected, in addition to Hall sensors, an encoder must be connected to the MC73110. The Hall sensors are required for initializing sinusoidal commutation to the correct angle. It is necessary to specify the number of encoder counts per electrical cycle. To determine this value, the number of magnetic poles on the motor, along with the number of encoder counts per motor revolution, must be known. Knowing these two quantities, the number of encoder counts per electrical cycle is produced using the following equation.

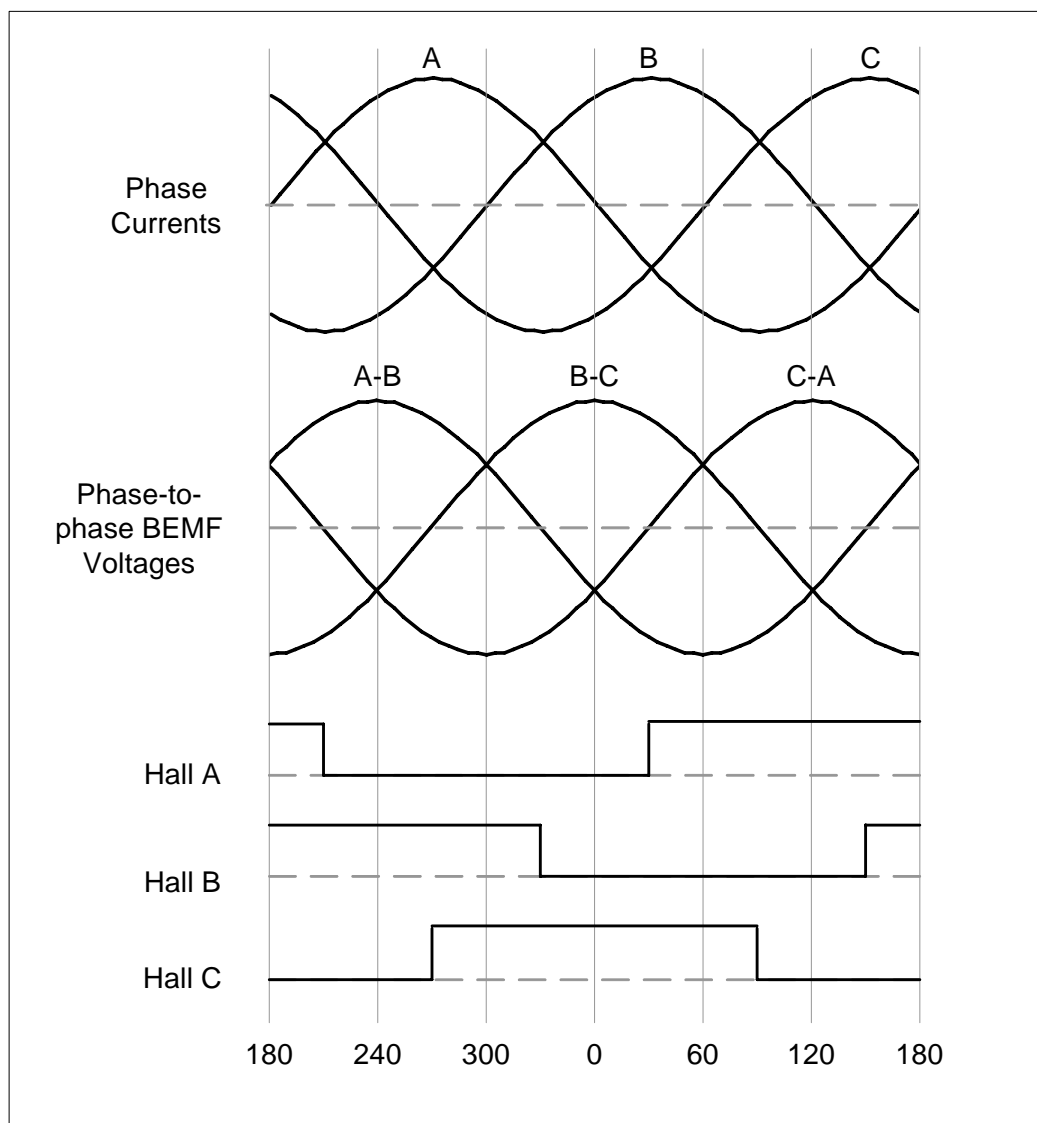
$$\text{Counts_per_cycle} = 2 * \text{Counts_per_rot} / N_poles$$

where:

Counts_per_rot	is the number of encoder counts per motor rotation
N_poles	is the number of motor poles

Figure 4-12 on page 34 shows the relationship between the state of the three Hall sensor inputs, the sinusoidally commutated phase currents, and motor phase-to-phase back EMF waveforms during forward motion of the motor (positive motor command).

Figure 4-12:
Motor
command
phasing vs.
Hall sensors



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

4.7.4 Hall-based Commutation

If Hall-based commutation is specified, the Hall sensors are used to commutate the motor using a six-step technique, and no encoder data is used for commutation.

Figure 4-12 on page 34 shows the relationship between the state of the three Hall sensor inputs and the output waveforms when Hall-based commutation is selected during forward motion of the motor. The following table details the expected Hall states for forward rotation of the motor. In order for correct motor operation, the Hall states must match the states in the table:

Hall1	Hall2	Hall3	Phase A Output	Phase B Output
1	0	0	- Motor Command	0
1	1	0	0	- Motor Command
0	1	0	+ Motor Command	- Motor Command

Hall1	Hall2	Hall3	Phase A Output	Phase B Output
0	1	1	+ Motor Command	0
0	0	1	0	+ Motor Command
1	0	1	- Motor Command	+ Motor Command

4.7.5 Automatic Phase Correction

To enhance long-term commutation reliability when operating in sinusoidal commutation mode, the MC73110 provides automatic phase correction using Hall sensors (the default value), or an index pulse, if one is available. By using an index pulse during the phase calculations, any long term loss of quadrature encoder counts which might otherwise affect the accuracy of the commutation are automatically eliminated.

If an index pulse is available, then it should be used to perform automatic phase correction. To set the phase correction mode, the command **SetPhaseCorrectionMode** is used. To read this value, the command **GetPhaseCorrectionMode** is used.

4.8 Field Oriented Control (FOC)

The MC73110 supports the use of Field Oriented Control (FOC), which provides improved performance compared to phase current, especially at fast motor speeds. The performance benefits are realized because, in FOC, the current loops are run in a de-rotated frame of reference (D/Q frame), compared to the rotating motor frame of reference (phase A/B frame). Thus the torque producing (Q) and magnetizing (D) currents are controlled directly, rather than controlling the sinusoidal A/B phase currents.

FOC current control is enabled using the **SetCommutationMode** command. FOC can be used with either Hall-based or Encoder-based (sinusoidal) phasing. When enabled, FOC replaces the commutation, digital current loop, and motor output blocks. However, all of the API commands that normally control the operation of these blocks are still used to control the operation of the equivalent functionality in FOC.

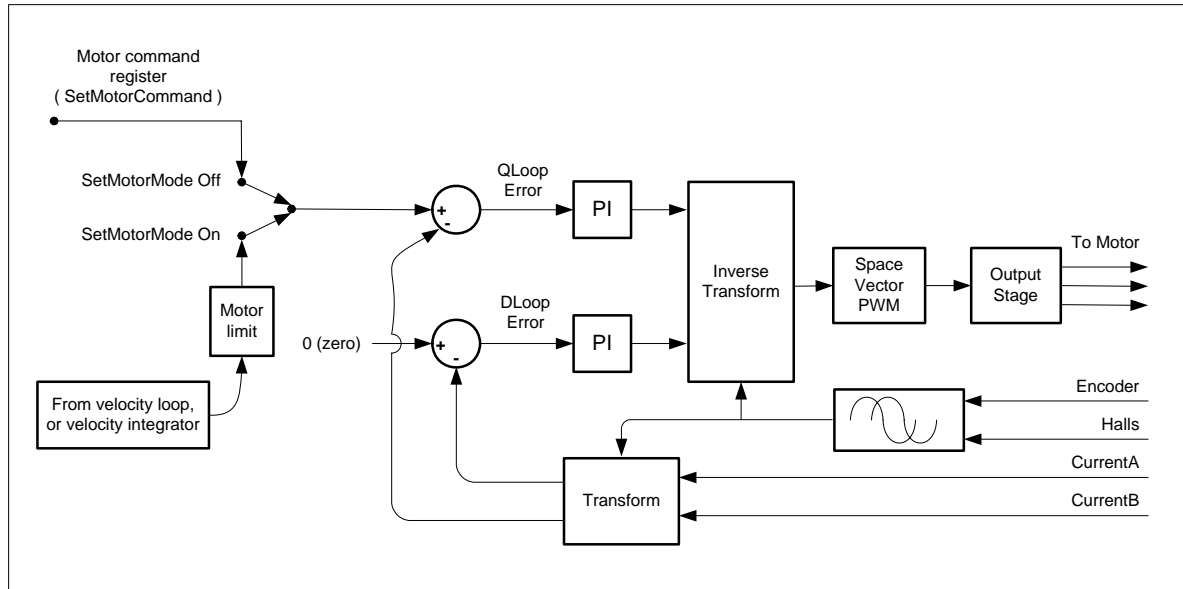
Sinusoidal phasing requires both encoder and Hall sensors.



Figure 4-13 on page 36 shows the FOC current control flow. The commutation module is replaced with transforms that convert the 3-phase rotating (A,B,C) frame to/from a 2-phase de-rotated frame (D/Q). These transform operations require the motor phase, which is initialized and commutated using the same methods and commands used for normal Hall-based or sinusoidal commutation.

The Phase A/B digital current loops are replaced with a D and a Q current loop. The same commands are used to set parameters and read values for the current loops, with the D loop replacing the A loop, and Q loop replacing the B loop. This applies to **SetLoopGain**, **GetLoopCommand**, **GetLoopError**, and **GetLoopIntegral**. The feedback currents (in A/B domain) are still available using **GetAnalog**.

Figure 4-13:
FOC current
control flow



For motor output, the phase-specified PWM generation is replaced with space-vector PWM. However, the PWM output mode (6-signal or 3-signal), dead time, and PWM limit are still configurable using the same commands as would be used in the A/B domain operation. The actual PWM duty cycle sent to each phase is still available using **GetPWMCommand**. The only restriction when using FOC is that the PWM output mode which floats the third leg is not available, regardless of whether Halls are used for phasing or not.

4.9 Velocity Loop

The velocity loop accepts a desired velocity command from one of four sources: the profile generator, the velocity integrator, the analog signal input AnalogCmd, or the 16-bit synchronous serial SPI-port at pins DigitalCmdClk and DigitalCmdClk. In all cases, the input value is a signed 16-bit word representing the desired velocity. If the velocity loop is disabled, then this value is passed through to the commutation module, effectively putting the chip in torque or voltage mode. To disable and enable the velocity loop, the command **SetLoopMode** is used, and the command **GetLoopMode** reads this value. To read the instantaneous value of the selected velocity command source, the command **GetLoopCommand** is used.

For more information on scaling and related issues for analog signal input, see Section 4.21, “Analog Signal Processing,” on page 60. See Section 4.20, “Synchronous Serial Input (SPI Port),” on page 59 for more information on the format and timing of the SPI port.

4.9.1 Velocity Feedback

There are three possible sources for the instantaneous value of the motor velocity, which is required for the velocity loop to operate properly. The first is an analog input connected to a tachometer, the second is the encoder data stream, and the third are the Hall sensors. To select the velocity source, the command **SetVelocityFeedbackSource** is used. This value can be read using **GetVelocityFeedbackSource**.

If the analog source is selected, the voltage from the tachometer is read at the velocity loop update rate, converted internally using an A/D, and then used for subsequent velocity loop filter calculations. If the encoder or Halls is used, the velocity is calculated using an estimator model which observes the encoder or Hall sensor data stream.

If the velocity source is selected as tachometer, then the velocity value, input through the tachometer signal, is a signed 16-bit number encoded in the same way as other analog input signals (see Section 4.21, “Analog Signal Processing,”

on page 60 for more information). However, if the velocity source is selected as encoder or Halls, the velocity value must be re-scaled from its native representation, which is counts per cycle (either encoder counts or Hall states).

This is accomplished using a velocity scalar, set using the command **SetVelocityScalar**, and read using the command **GetVelocityScalar**. The scalar operates within a range of 3 to 32,767. After a reset or at power-up, the default value is 3. In addition to converting counts per cycle representations to 16-bit velocity representation, the velocity scalar may also be used to convert in the opposite direction. See Section 4.10, “Velocity Integrator,” on page 40 for details. To set the cycle time of the IC, which is normally 102.4 μsec (9.76 kHz cycle rate), the command **SetSampleTime** is used. See Section 4.12, “Loop Rate,” on page 43 for more information.

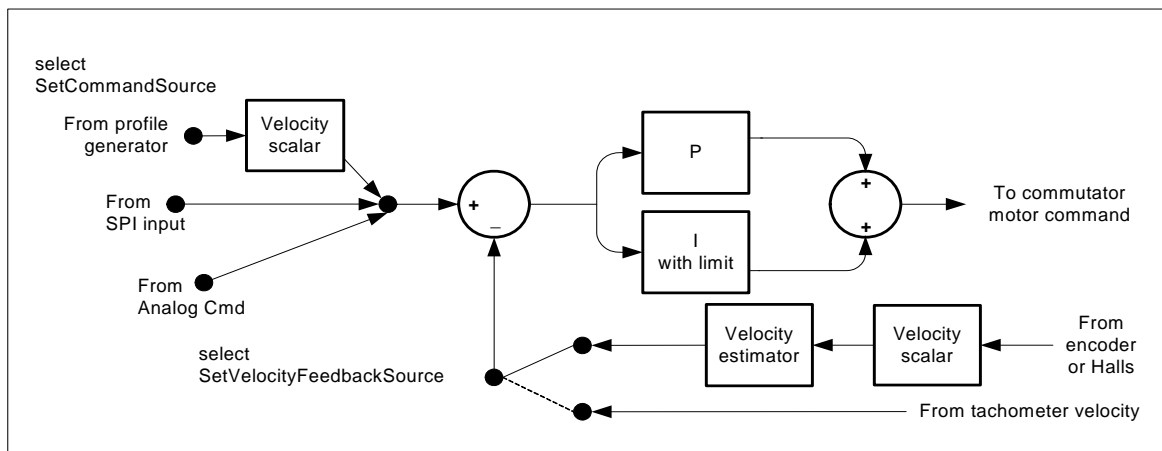
To convert a counts per cycle representation to 16-bit velocity representation, the specified velocity scalar is simply multiplied by the velocity in counts per cycle. For example, if measured velocity using the encoder is 7 counts in a particular cycle, and the velocity scalar has been set to 1,000, then the 16-bit velocity representation of that velocity will be 7,000. The velocity signal is filtered using a second order IIR filter and then input to the velocity loop as a 16-bit quantity. Because the velocity is subject to quantization error it is good practice to choose the velocity scalar as large as possible while still avoiding overflow (see the warning below). See Section 4.9.3, “Velocity Loop Filter,” on page 38 for more information.

If the Halls are used for velocity feedback, performance will be limited by resolution of the Hall sensors. In order to achieve reasonable performance using Hall-based velocity estimation, it is recommended that at least one Hall crossing occur every 5 cycles. In most cases, this must be done by significantly increasing the sample time from the 102.4 μsec default. When using Halls, the same rules apply regarding setting the velocity scalar as in the encoder case.

When selecting the velocity scalar, be sure to specify a range appropriate for the expected encoder rate and cycle time which has been selected. For example, if the instantaneous encoder rate experienced by the MC73110 and the velocity scalar programmed by the user results in a 16-bit velocity value exceeding 32,767, or less than -32,768, then the value will become “clipped.” This will result in an inaccurate velocity value, and may potentially cause unstable motion.



Figure 4-14:
Velocity
feedback and
scaling



4.9.2 Velocity Scalar

In this section, two examples which illustrate the functionality of the velocity scalar will be provided.

Velocity Scalar Example 1: Velocity Loop

In this example, the desired result will be using the velocity loop to command a motor to move at seven counts per cycle.

First, the expected absolute maximum encoder or Hall counts per cycle must be determined. Anticipating a 50% velocity overshoot transient results in a maximum encoder rate of 10.5 counts per cycle. The result of multiplying the encoder or Hall feedback times the velocity scalar must fit in a 16-bit 2's complemented word. Hence, the velocity scalar is determined by dividing 32767 by 10.5:

$$32,767/10.5 = 3121 \text{ (rounded up)}$$

Therefore, the command **SetVelocityScalar** 3121 would be sent to the motion control processor, along with other startup configuration settings. The velocity has been set at seven counts per cycle. So the 16-bit value that represents the velocity command in Figure 4-14 on page 37 would be:

$$3121 \times 7 = 21,847$$

If the command source is set to SPI, then the 16-bit value received on the SPI interface is 21847 (5557h). If a velocity of -7 counts per cycle is desired, then the 16-bit value on the SPI interface should be -21847 (AAA9h). If the command source is set to AnalogCmd, then the voltage on the analog input would be calculated as follows:

$$1.65V + (21,847 \times 1.65/32,767) = 2.75V$$

If the command source is set to profile generator, then the velocity scalar is determined in the same way. The value used in the **SetVelocity** command should not include the velocity scalar.

Velocity Scalar Example 2: Velocity Integrator Loop

In the following example, the desired goal is to use the velocity integrator loop to command a motor to move at 7 counts per cycle. The logic behind determining an appropriate velocity scalar when using the velocity integrator loop is different than the logic when using a velocity loop. In the velocity integrator loop, the velocity scalar is applied to the SPI or analog input instead of the encoder feedback. In the context of the velocity integrator, the velocity scalar allows for increased resolution of the reference velocity. For instance, if a Velocity Scalar was not used, then a fractional reference velocity could not be represented at the SPI or analog input. Therefore, the velocity scalar is really only useful when the desired value of counts per cycle is non-integer.

When using the SPI or analog command source input, the value read at the input results in a 16-bit number. Only after this number has been divided by the velocity scalar and the result transformed to a 16.16 format is the number used for integration. As in the previous example, a velocity scalar of 3121 will be chosen. (However, a larger velocity scalar could have been chosen, such as 4681 (32,767/7)). Using a velocity scalar value of 3121, the 16-bit SPI value would still be 21,847, and the analog voltage input would be 2.75V. The equations for determining these values remain the same. When using the profile generator as the command source, the velocity scalar becomes irrelevant in the velocity integrator loop.

4.9.3 Velocity Loop Filter

Three values must be set by the user: VelocityKp, VelocityKi, and VelocityILimit. These parameters are set and read using the commands **SetLoopGain**, and **GetLoopGain**, respectively.

The result of the velocity loop calculation is a signed 16-bit number. This value is then input to the commutation module. To read this value the command **GetMotorCommand** is used. To read the instantaneous velocity loop error value, the command **GetLoopError** is used. The velocity loop is calculated as follows:

$$VE_n = \text{VelocityDesired}_n - \text{VelocityActual}_n$$

$$\text{VelocityOutput}_n = VE_n \times \text{VelocityKp}/64 + \sum_{j=0}^n VE_j \times \text{VelocityKi}/256$$

where:

VE is the velocity error term

VelocityDesired	is the velocity command from the velocity integrator, profile generator, SPI, or analog source
VelocityActual	is the measured velocity from the encoder or tachometer source
VelocityOutput	is the output from the velocity filter
VelocityKp	is the velocity proportional gain
VelocityKi	is the velocity integral gain
VelocityKout	is a velocity output scalar

When the velocity loop is disabled, or when the motor mode is off, then the integrator remains at zero (0).

4.9.4 Tachometer Signal Offset Bias

As was the case for the digital current feedback inputs, the MC73110 has the ability to store a software offset bias which is combined with the A/D value for the AnalogCmd signal and/or the tachometer signal. This may be useful for zero-referencing the analog circuitry without the need for external adjusting circuitry such as potentiometers.

The AnalogCmd and Tachometer signals can be read using the command **GetAnalog**. The returned value is a signed 16-bit number representing the velocity command or analog feedback value on the analog input pin. See Section 4.21, “Analog Signal Processing,” on page 60 for details on converting analog values to numeric values and vice-versa.

To add a bias offset to the AnalogCmd or Tachometer signals, the command **SetAnalogOffset** is used. To read this value, the command **GetAnalogOffset** is used.

The offset value is added to the value read by the chip to determine the value used during velocity loop calculations. A simple way of determining the offset of this signal is to lock the motor position, and read the analog signal using **GetAnalog**. Since the external voltage sources are disconnected, the read value should indicate zero voltage. This value can then be negated and entered as the offset to correctly zero-reference that analog input. For example, if the value read is -75 , the offset value should be set as $+75$.

Due to temperature changes and other factors, the ideal offset value may change during operation and over the lifetime of product usage. It is the responsibility of the designer to insure that the MC73110 and any associated amplifier circuitry is operated within safe limits.



4.9.5 Motion Error Detection

Under certain circumstances, the actual axis velocity may differ from the desired velocity by an excessive amount. Such an excessive velocity error often indicates a potentially dangerous condition such as motor or encoder failure, or excessive mechanical friction. To detect this condition, the MC73110 includes a programmable maximum error feature.

The maximum error is set using the command **SetMotionErrorLimit**, and read using the command **GetMotionErrorLimit**. If the maximum velocity error value is exceeded, then the motor is said to be in motion error. When this occurs, the Motion Error bit in the axis Event Status word is set. At this time, the axis motor may be turned off (equivalent of **SetMotorMode Off** command), depending on the state of the automatic motor shutdown mode (see **SetAutoStopMode** command description). See Section 4.13, “Status Words,” on page 44 for more information on the Event Status word.

Motion error detection is still performed when using tachometer feedback or an analog or SPI command reference input. When defining the value for **SetMotionErrorLimit**, 16.16 scaling and the velocity scalar should be taken into consideration. That is, the value used for the parameter to **SetMotionErrorLimit** should be in a 16.16 format and then divided by the velocity scalar before being sent to the chip.

4.9.6 Recovering from a Motion Error

If the **AutoStopMode** is set active and a motion error occurs, the motor mode will automatically be set to off. In addition, at the time this happens, the Motor Command register, normally set using the command **SetMotorCommand**, will have a value of zero written to it. This will result in the motor coming to a halt, because the motor command at that point is zero.

To return to normal operation, the command **SetMotorMode** is used with a value of On. This will set the IC to automatic control using the velocity loop to continuously drive the commutator and downstream modules. In addition, it may be desirable to clear the Motion Error bit in the Event Status word. See Section 4.13, “Status Words,” on page 44 for more information on the Event Status word.



After a motion error, it is not recommended that operation of the IC be restarted before the axis has come to a full stop, and the cause of the motion error has been determined.

4.9.7 Auto Stop Mode

The command **SetAutoStopMode** determines whether or not a motion error causes the motor mode to be automatically set to off, which will relinquish control of the motor output to the manually-set Motor Command register. The value of this register can be read using **GetAutoStopMode**.

If this command is set to **Enable**, then upon a motion error the motor mode will be set off, the Motor Command register will be set to zero, and thus the axis will come to a stop. If the **SetAutoStopMode** is set to **Disable**, then even if a motion error occurs, the motor mode will not be affected.

4.10 Velocity Integrator

The MC73110 includes a special feature known as a velocity integrator which allows 32-bit velocity profile resolutions to be generated from 16-bit velocity values. If enabled, the velocity integrator can utilize velocity values from the profile generator, the SPI data stream, or the AnalogCmd signal. To select the source for the velocity integrator, the command **SetCommandSource** is used. To read this value, the command **GetCommandSource** is used.

To convert 16-bit velocity representations provided by the SPI data stream or AnalogCmd into the 16.16 velocity format which is integrated to create a 32-bit position, the VelocityScalar register is used. The conversion is performed by multiplying the 16-bit value by the reciprocal of the scalar value. For example, if the value read through AnalogCmd is 3,500, and the scalar is 1,000, the velocity used by the integrator is 3.5 counts per cycle, which in 16.16 coding is a value of 3 in the high word, and a value of 8000h in the low word. See Section 4.9.1, “Velocity Feedback,” on page 36 for more information on the VelocityScalar. To enable and disable this module, the command **SetLoopMode** is used. To read this setting, the command **GetLoopMode** is used.

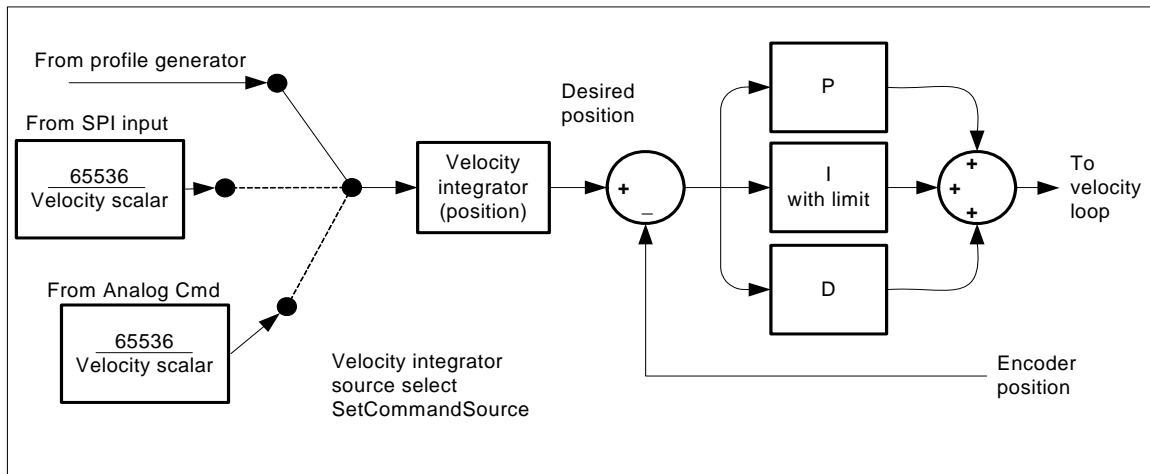


Figure 4-15:
Velocity
integrator
source select

If this module is enabled, at each servo loop update cycle, the output of the profile generator is accumulated in a 32-bit Target Position register, which is then directly compared with the 32-bit actual position to form a 16-bit position error. This position error is then put through a complete PID filter, and the output of this filter calculation is made available to the downstream components. This will be either the velocity loop or the commutator, depending on whether the velocity loop has been enabled.

4.10.1 Velocity Integrator Loop Filter

Figure 4-15 shows a flow diagram for the integrated velocity loop filter incorporated in the MC73110. Four values can be set by the user: $K_{p_{\text{velocityintegrator}}}$, $K_{i_{\text{velocityintegrator}}}$, $K_{d_{\text{velocityintegrator}}}$, and $\text{IntegrationLimit}_{\text{velocityintegrator}}$. These parameters are set using the command **SetLoopGain**, and the parameters are read using the command **GetLoopGain**. To read the current integral value, the command **GetLoopIntegral** is used.

The equation for calculating the output of the velocity integrator filter is as follows.

$$E_t = \text{PositionDesired} - \text{PositionActual}$$

$$\text{Output}_n = \left[K_p E_n + K_d (E_n - E_{n-1}) + \sum_{j=0}^n E_j \times K_i \right]$$

where:

PositionDesired	is the 32-bit desired position from the velocity integrator
PositionActual	is the 32-bit position from the encoder
E_n	is the position error at time n
E_{n-1}	is the position error at time $n-1$
K_p	is the proportional gain
K_d	is the derivative gain
K_i	is the integral gain

When the velocity integrator loop is disabled, or when the motor mode is off, then the integrator remains at zero(0).

The tachometer or Hall-based velocity estimation can not be used with the velocity integrator loop.



4.10.2 Motion Error Detection

Similar to the velocity loop, under certain circumstances, the actual axis position may differ from the integrated position by an excessive amount. Such an excessive position error often indicates a potentially dangerous condition such as motor or encoder failure, or excessive mechanical friction. To detect this condition, the MC73110 automatically utilizes the motion error facility. See Section 4.9.5, “Motion Error Detection,” on page 39 for more information. If this module is enabled, it will monitor the position error of the velocity integrator.

If the velocity integrator loop is enabled, the maximum error set using the **SetMotionErrorLimit** applies to the position error of the velocity integrator, rather than the velocity error. If both the velocity integrator and the velocity loop are active, the velocity integrator has precedence, and no velocity error monitoring is performed. In other respects, the motion error processing is identical.

To return to normal operation, one additional step is required. The position error should be set to zero to avoid a motion jump upon re-enabling the motor mode to on. This is accomplished with the **ClearPositionError** command. Executing this command will reset the velocity integrator target position equal to the current actual position, thereby insuring that there is no motion discontinuity upon setting the motor mode on.

4.10.3 Recovering from a Motion Error

Recovering from a position-based motion error is similar to recovering from a velocity-based motion error. If **AutoStopMode** is set active and a motion error occurs, the motor mode will automatically be set to off. In addition, at the time this happens, the Motor Command register, normally set using the command **SetMotorCommand** will have a value of zero written to it. This will generally result in the motor coming to a halt because the motor command at that point is zero.

To recover, the command **SetMotorMode** is then used with a value of on. This will set the IC to restore operation through the velocity integrator; continuously driving the velocity loop (if enabled) and/or commutator and downstream modules. In addition, it may be desirable to clear the Motion Error bit in the Event Status word. See Section 4.13, “Status Words,” on page 44 for more information on the Event Status word.

4.11 Profile Generation

The internal profile generator can be used to generate acceleration and velocity-bounded profiles to directly drive the motor without continuous external signals from the AnalogCmd or DigitalCmdClk and DigitalCmdData signals.

To enable the profile generator, the command **SetCommandSource** is used. To read this value, the command **GetCommandSource** is used. Note that effective use of the profile generator is tied to the processing of serial port commands; if the serial port is not being used, the profile generator cannot be effectively used.

The following table summarizes the host-specified profile parameters for the profile generator:

Profile Parameter	Format	Word Size	Range
Maximum 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 .

To operate the profile generator, the host specifies two parameters: the commanded acceleration, and the maximum velocity. The trajectory is executed by continuously accelerating the axis at the commanded rate until the maximum velocity is reached, or until a new acceleration or velocity command is given. To set the maximum velocity, the command **SetVelocity** is used. This value can be read using **GetVelocity**. To set the acceleration, the command **SetAcceleration** is used. To read this value, the command **GetAcceleration** is used.

The acceleration value must always be positive. Motion direction is controlled using the velocity value. Positive velocity values result in positive motion, and negative velocity values result in negative motion. There are no restrictions to on-the-fly profile parameter changes.

Figure 4-16 details a typical velocity profile using this mode.

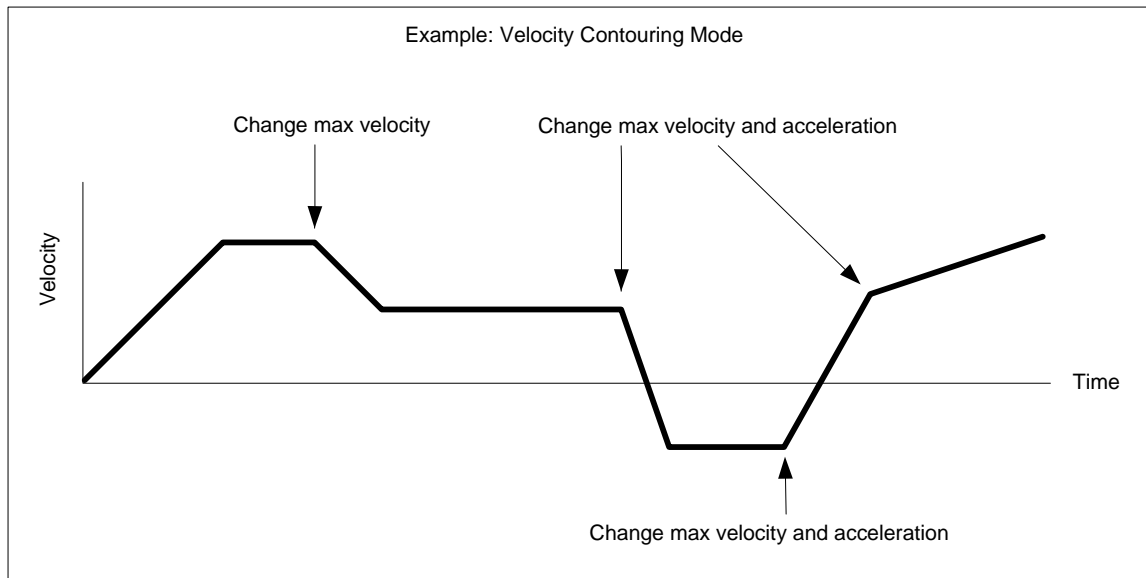


Figure 4-16:
Typical velocity
profile

To read the instantaneous velocity of the profile generator, the command **GetLoopCommand** using the argument *Velocity/Integrator* is used. The returned value is a 32-bit number with $1/2^{16}$ scaling.

4.11.1 Motor Mode

If the motor mode is set to off, either by a host command, or automatically because of a motion error, the trajectory generator is automatically halted. The instantaneous velocity, read using the command **GetLoopCommand**, will be zero. Once the **SetMotorMode** command is set to “on,” the trajectory will resume. The instantaneous velocity will be zero at that point, and any subsequent motion will be based on the trajectory profile commands in place at that time.

4.12 Loop Rate

The overall servo loop rate, also known as the the cycle time, can be changed by the user using the command **SetSampleTime**. This value can be retrieved using the command **GetSampleTime**.

Although it is generally not necessary to change the sample time, for some applications this may improve performance. Sample time affects the rates of the following modules: profile generator, velocity integrator, and velocity loop. It does not affect the commutation rate, the current loop rate or the PWM output rate.

The loop rate is specified to the chip set as an integer number of microseconds. The minimum value allowed is 102 μsec . This is also the default value. The sample time can be increased by multiples of 51.2 μsec , rounding to the nearest integer value. For example, 154, 205, 256, etc.

The sample time should not be changed while axes are in motion.



4.13 Status Words

The MC73110 can monitor various aspects of the motion of the axis. Various numerical registers can be queried to determine the current state of the axis, such as the actual position (**GetActualPosition** command), or the current motor command (**GetMotorCommand**), etc.

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

The host may query these three registers, or the contents of these registers may be used in some operations to trigger other events such as **AmplifierDisable** signal output, or **PWMOutputDisable**. See Section 4.14, “Programmable Conditions,” on page 47 for more information on these functions.

4.13.1 Event Status Register

The Event Status register is designed to record events which do not continuously change in value, but tend to occur once upon some specific event. As such, each of the bits in this register are set by the chip and cleared by the host.

The Event Status register is defined in the following table.

Bit	Name	Description
0	—	Reserved
1	Wrap-around	Set (1) when the actual (encoder) position has wrapped from maximum allowed position to minimum, or vice versa.
2	—	Reserved
3	Capture Received	Set (1) when a position capture has occurred.
4	Motion Error	Set (1) when a motion error has occurred.
5–7	—	Reserved
8	Amplifier Error	Set when the programmable amplifier error condition becomes true.
9–14	—	Reserved
15	User Command Error	Set (1) if error occurred processing user commands from EEPROM or Flash after chip reset.

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

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 command **ResetEventStatus**.

4.13.2 Activity Status Register

Like the Event Status register, the Activity Status register tracks various chip registers. Activity Status register bits are not latched; they are continuously set and reset by the chip to indicate the states of the corresponding conditions.

The Activity Status register is defined in the following table:

Bit	Name	Description
0–5	—	Reserved
6	Overtemperature	Set (1) when the overtemperature condition is active, cleared (0) when the overtemperature condition is not active. The overtemperature condition is controlled by the maximum temperature register and the command SetTemperatureLimit .
7	PWMDisable	Set (1) when PWMDisable condition mask evaluates to True.

Bit	Name	Description
8	Motor Mode	Set (1) when the motor is on, cleared (0) when the motor is off. When the motor is on, the chip can perform profile generation, and the velocity integrator and velocity loop, if enabled, will be active. When the motor is off, profile generation cannot be performed, the velocity integrator and velocity loop are not active (regardless of whether they are enabled or disabled), and the motor command is derived from the Motor Control register.
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.
10	Overvoltage	Set (1) when Bus voltage exceeds the Overvoltage limit.
11	Undervoltage	Set (1) when Bus voltage exceeds the Undervoltage limit.
12–15	—	Reserved

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

The bits in this register are set by the chip and cannot be directly reset by the host, as is possible with the Event Status word. Some host commands will affect the state of these bits, such as altering the maximum temperature value, or reading a position capture.

4.13.3 Signal Status

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

Bit	Name	Description
0	QuadA	QuadratureA encoder input
1	QuadB	QuadratureB encoder input
2	Index	Index input
3–6	—	Reserved
7	Hall1	Hall effect sensor input number 1
8	Hall2	Hall effect sensor input number 2
9	Hall3	Hall effect sensor input number 3
10–12	—	Reserved
13	Estop	Emergency stop input signal
14	PWMOutputDisable	PWMOutputDisable input signal
15	AmplifierDisable	AmplifierDisable output signal level

The command **GetSignalStatus** returns the contents of the Signal Status register.

All Signal Status register bits are inputs, except bit 15 (AmplifierDisable), which is an output.

The **SetSignalSense** command will effect the values returned by the **GetSignalStatus** command. The default state of the signal sense mask is 0000h. In this state, **GetSignalStatus** will return 1 for any input at 3.3V, and 0 for any input at 0V. The results of **GetSignalStatus** returning [010x xx10 1xxx x011]b are shown in the following table.

Input	Voltage
AmplifierDisable	0
PWMOutputDisable	3.3
Estop	0
Hall3	3.3
Hall2	0
Hall1	3.3
Index	0
EncoderB	3.3

Input	Voltage
EncoderA	3.3

If the **SetSignalSense** command is used to change the signal sense mask from the default value (0000h), then the interpretation of the voltage states will change. For example, if the command **SetSignalSense 2380h** is used, and the **GetSignalStatus** returns the same value as in the previous example ([010x xx10 1xxx x011]b), then the interpretation of the voltage state of the various inputs are changed as shown in the following table.

Input	Voltage
AmplifierDisable	0
PWMOutputDisable	3.3
Estop	3.3
Hall3	0
Hall2	3.3
Hall1	0
Index	0
EncoderB	3.3
EncoderA	3.3

Using the **SetSignalSense** command changes the values returned by **GetSignalStatus**. However, **SetSignalSense** will not affect the logical interpretation of the Signal Status values. For example, if **GetSignalStatus** returns a zero for the Estop bit, then the Estop is considered to be inactive, regardless of whether or not **SetSignalSense** was used to translate the true voltage level at the Estop pin. This is because the MC73110 will interpret the state of a bit (active or inactive) only after taking the signal sense mask into consideration.

The bits in the Signal Status register represent the logical level (not the hardware level) of each of these signals. These two values may be different if a value other than 0 is programmed in the signal sense mask. See Section 4.13.4, “Signal Sense Mask,” on page 46 for more information.

4.13.4 Signal Sense Mask

The bits in the Signal Status register represent the state of various signal pins on the chip. These bits can be inverted. This is useful for changing the interpretation of input or output signals to match the signal interpretation of the user's hardware.

The Signal Sense Mask register is defined in the following table.

Bit	Name	Description
0, 1, 3–6	—	Reserved
2	Index	Set (1) is Low to High transition resulting in a capture. Set (0) is High to Low transition resulting in a capture.
7	Hall1	Set (1) to invert Hall1 signal. Clear (0) for no inversion.
8	Hall2	Set (1) to invert Hall2 signal. Clear (0) for no inversion.
9	Hall3	Set (1) to invert Hall3 signal. Clear (0) for no inversion.
10–12	—	Reserved
13	Estop	Set (1) for active high interpretation. Clear (0) for active low interpretation.
14–15	—	Reserved

The command **SetSignalSense** sets the signal sense mask value. The command **GetSignalSense** retrieves the current signal sense mask.

4.14 Programmable Conditions

The MC73110 supports the ability to monitor a number of condition bits in either the Event Status, Activity Status, or Signal Status words. This condition may be used to trigger various chip events.

The chip events that can be programmed are the setting of the AmplifierError (bit 8 in the Event Status word), the setting of the AmplifierDisable signal, and the control of PWM output. A wide variety of conditions can be programmed, so that behaviors such as “disable PWM output when an overtemperature condition occurs,” or “define an amplifier error as AmplifierDisable being low” may be defined.

The following table defines the condition-select bit mask for all programmable events:

Bit	Name	Register Location	Source Bit
0–2	— (Reserved)	—	—
3	Capture Received	Event Status	3
4	Motion Error	Event Status	4
5	— (Reserved)	—	—
6	Overtemperature	Activity Status	6
7	— (Reserved)	—	—
8	Amplifier Error*	Event Status	8
9	— (Reserved)	—	—
10	Overvoltage	Activity Status	10
11	Undervoltage	Activity Status	11
12	— (Reserved)	—	—
13	Estop	Signal Status	13
14	PWMOutputDisable	Signal Status	14
15	— (Reserved)	—	—

*The Amplifier error bit is available only for the AmplifierDisable and PWMDisable events, and not for the Amplifier error event.

4.14.1 Amplifier Error Bit

Bit 8 of the Event Status is designed to indicate when an amplifier error has occurred. However, the definition of an amplifier error is user-programmable.

The command **SetConditionMask** is used to set the condition mask, and the command **GetConditionMask** is used to read it. A 1 in the bit mask means that that if an active condition occurs in the corresponding source register bit, the condition is satisfied. If more than one bit-field is programmed with a 1, a logical OR condition is applied. For example, to define an amplifier error condition as the occurrence of an overtemperature or an Estop going active, the condition mask would be set to 2040h.

4.14.2 AmplifierDisable Output Pin

The AmplifierDisable signal of the MC73110 may be programmed in a manner similar to the amplifier error condition. The table in Section 4.14, “Programmable Conditions,” on page 47 is used to determine the correct mask value. The command that is used to program this mask is **SetConditionMask**. This value can be read using the command **GetConditionMask**.

In this case, if the defined condition becomes true, then the AmplifierDisable signal is driven active. If conditions change such that it is no longer true, then the signal is driven inactive.



The AmplifierDisable pin is high (active) after power-up. Once a command that defines the control mask for this pin has been sent (**SetConditionMask**), normal processing of the AmplifierDisable pin condition begins, which usually (if none of the conditions are present) will result in the signal becoming inactive (low). This special processing of the AmplifierDisable pin insures that the AmplifierDisable pin is not brought inactive until the chip has completed its power-up sequence, or if using the serial port, until the user has sent the **SetConditionMask** command for the AmplifierDisable pin mask.

The AmplifierDisable output will also be driven active in case of an error processing user commands from EEPROM/Flash. Regardless of the condition mask result, it will remain active until the User Command Error bit in the Event Status register is cleared.



When the AmplifierDisable pin is active, the MC73110 is held in a dormant state equivalent to **SetLoopMode** = 0 (all loops disabled).

4.14.3 Disabling PWM Output

The final condition that can be programmed using a mask is PWM output. The table in Section 4.14, “Programmable Conditions,” on page 47 is used to determine the correct mask value. The command that is used to program this mask is **SetConditionMask**. This value can be read using the command **GetConditionMask**.

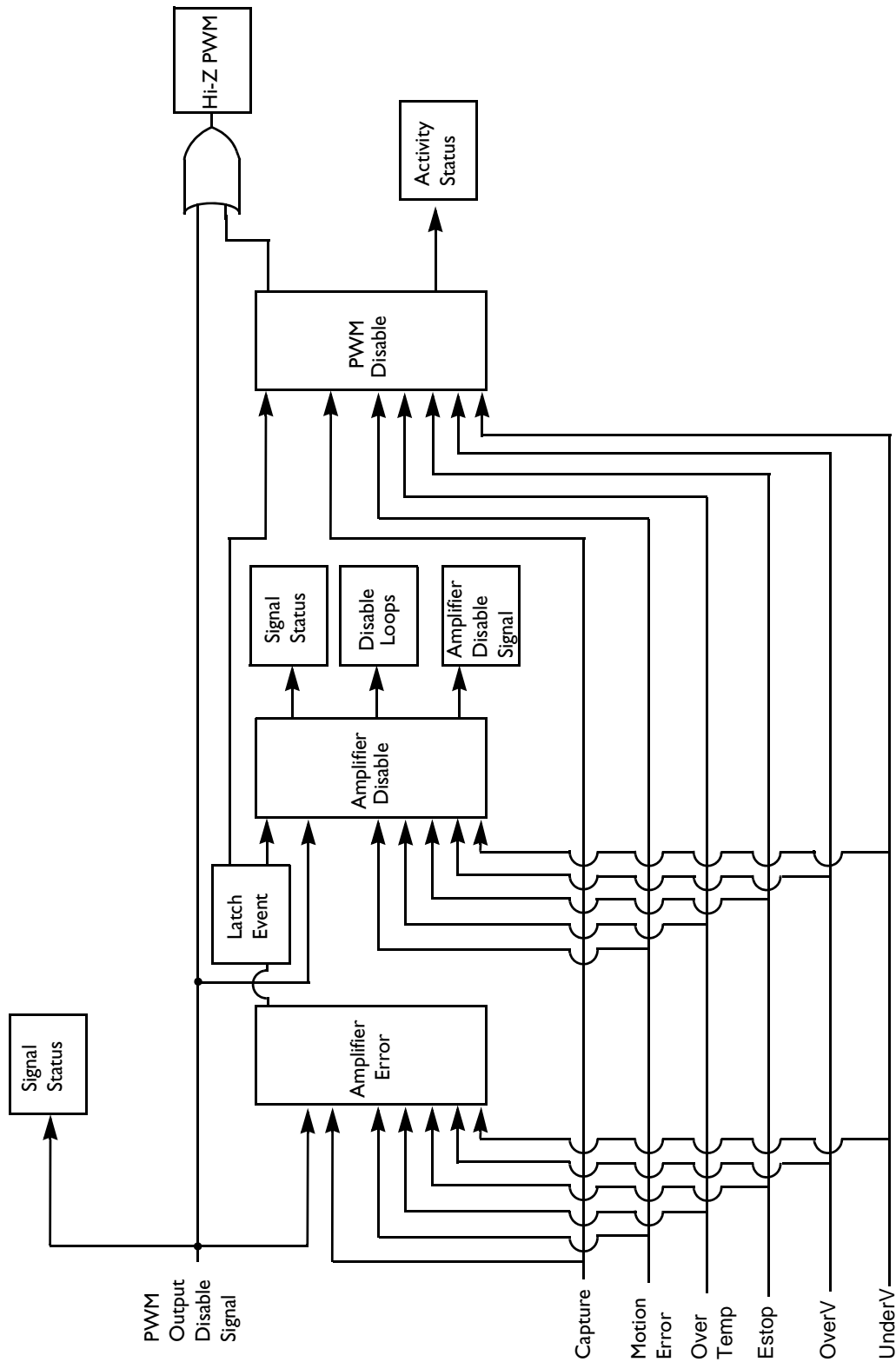
In this case, if the defined condition becomes true, then PWM output will be put in high impedance. If conditions change such that it is no longer true, PWM output will be enabled. The PWMDisable condition mask defaults to 2000h (Estop) and as a result Estop will disable this output if the condition mask remains unchanged.



The PWM outputs will be tri-stated if either the PWMOutputDisable input pin is active or the PWMDisable condition is active.

4.14.4 Condition Mask Diagram

The following diagram provides an overview of the programmable triggers for the condition masks and the subsequent effect.



4.15 Temperature Sensor

The MC73110 supports the ability to input temperature data from an I²C compatible temperature sensor. These are available from a number of vendors. Typically, such a sensor would be placed on or near critical amplifier components such as MOSFETs or IGBTs to guard against overtemperature conditions.

Although several sensors are supported, the MC73110 is certified to work with the Burr-Brown TMP100 digital temperature sensor with I²C interface.

The sensor is connected to the MC73110's I²C bus (pins I²CCLK and I²CData). The temperature sensor's address must be set to 48h. The following table summarizes the I²C addresses used by the MC73110.

MC73110-Attached Device	I ² C Device Address
Temperature Sensor	48h
Serial EEPROM	50h

After reset, the MC73110 will use the I²C bus to configure the sensor for 12-bit resolution.



For future product versions: currently, the MC73110 is certified to work only with the parts and addresses shown in the preceding table. Additional commands could be required if different parts and addresses are used. For details, contact PMD

4.15.1 Overtemperature Detection

The temperature sensor is sampled approximately every 50 cycle times, and compared against an overtemperature set point. To read the current value from the temperature sensor, the command **GetTemperature** is used. To set the overtemperature value, the command **SetTemperatureLimit** is used. To read this value, the command **GetTemperatureLimit** is used.

The value returned by the I²C sensor is expected to be a two's complemented signed 16-bit word. It is the responsibility of the user to determine the scaling of the temperature sensor and the correct comparison value to load into the Temperature Limit register. The limit value comparison will be performed after the raw value from the sensor is shifted right by four.

The default value of the temperature limit is 7FFFh. If the overtemperature set point is exceeded, or if there is an error reading the data from the I²C sensor, then the Overtemperature bit in the Activity Status register will be set. This bit can be used as a trigger to disable the amplifier, or to inhibit PWM generation. See Section 4.14, "Programmable Conditions," on page 47 for more information.



If no temperature sensor is present, then the Overtemperature bit in the Activity Status register is always set .

4.16 Bus Voltage Sensor

The MC73110 supports the ability to monitor the bus voltage using an analog input, which feeds an internal 10-bit A/D converter. In order to use this functionality, not only must the bus voltage be connected to the Bus Voltage input, but also AnalogVCC, AnalogGND, AnalogRefHigh, and AnalogRefLow must be connected.

The bus voltage is monitored using the **GetBusVoltage** command. The value returned is an unsigned 16-bit value, representing the voltage on the BusVoltage pin. The scaling is:

$$\text{ReadValue} = 65535 * (\text{BusVoltage} - \text{AnalogRefLow}) / (\text{AnalogRefHigh} - \text{AnalogRefLow})$$

Assuming AnalogRefHigh is 3.3V and AnalogRefLow is 0V, this is simply:

$$\text{ReadValue} = 65535 * \text{BusVoltage pin} / 3.3\text{V}$$

To relate this value to the actual bus voltage, the gain of the bus voltage sensor must be taken into account. This gain should be chosen so that the maximum operating voltage results in a BusVoltage pin value of about 50% between AnalogRefHigh and AnalogRefLow. This will allow room for overvoltage and undervoltage detection, and provide good resolution in bus voltage readings.

For example, assuming a maximum nominal operating voltage of 48 volts and AnalogRefHigh=3.3V and AnalogRefLow=0V, the gain should be:

$$\text{SensorGain} = 0.5 * (3.3\text{V} - 0\text{V}) / 48\text{V} = 0.034375 \text{ V/V}$$

With the SensorGain taken into account, the bus voltage reading, in terms of actual bus voltage, is (assuming AnalogRefHigh=3.3V and AnalogRefLow=0V):

$$\text{ReadValue} = 65535 * \text{Bus voltage} * \text{SensorGain} / 3.3\text{V}$$

or

$$\text{Bus voltage} = (\text{ReadValue} * 3.3\text{V}) / (65535 * \text{SensorGain})$$

Note that, unlike the other analog inputs supported by the MC73110, the BusVoltage input is not bipolar, so no bias should be applied. Also, unlike the other analog inputs, the offset cannot be adjusted.

4.16.1 Over- and Undervoltage Detection

The bus voltage is sampled every 51.2us, passed through a low-pass filter, and compared against overvoltage and undervoltage limits. If the bus voltage exceeds the overvoltage limit, the Overvoltage bit in ActivityStatus is set. If the bus voltage reading is less than the undervoltage limit, the Undervoltage bit in ActivityStatus is set. Both of these bits in ActivityStatus can be used for any of the condition masks. For example, they can trigger disabling of the amplifier or PWM generation.

To configure the overvoltage and undervoltage limits, the **SetBusVoltageLimits** command is used. The range and scaling of these settings is identical to the value read using **GetBusVoltage**. The defaults for the overvoltage and undervoltage limits are 65535 and 0, respectively. If the bus voltage monitoring functionality is not used, leaving these limits at their defaults will guarantee that the over- and undervoltage conditions will never be detected.

4.17 Serial Port

The MC73110 can communicate with a host microprocessor, PC, or other controller through an asynchronous serial port. During development prototyping, it is very convenient to use the serial port, along with PC-based programs provided by PMD to exercise the MC73110, experiment with gain values, and measure system performance. During

actual product operation, the chip may still be controlled via the serial port. Alternatively, the boot serial EEPROM or internal Flash facility can be used to load parameter information, and no serial port is needed.

Whether the serial port will be used depends on the usage of the MC73110's internal profile generator, and on whether additional information regarding the motor's operating state is needed. For all features other than the profile mode, the chip may be operated without serial port communication. However, using these features in no way eliminates the ability to use the serial port for monitoring or other purposes.

4.17.1 Command Packets

The chip accepts commands from the host in a packet format. By sending sequences of commands, the host can control the behavior of the motion system as desired, and monitor the status of the chip and the motor.

A packet is a sequence of transfers to and/or from the host, which results in a chip action or data transfer. Packets can consist of a command with no data, a command with associated data that is written to the chip, or a command with associated data that is read from the chip.

All commands with associated data (read or write) have one, two, or three words of data. 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 command format used to communicate between the host and chip consists of a command packet sent by the host processor followed by a response packet sent by the chip.

Command packets sent by the host contain the following fields.

Field	Byte No.	Description
Address	1	One byte identifying the chip 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 integrity. See Section 4.17.3, "Checksums," on page 53 for details.
— (Reserved)	3	Always contains 0.
Instruction code	4	A one-byte instruction. See Chapter 5, "Instruction Reference," on page 67 for more information.
Data (optional)	5–8	Zero to four 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 chip will respond with a packet of the following format.

Field	Byte No.	Description
Status	1	Zero if the command was completed correctly; otherwise an error code specifying the nature of the error. See Section 4.17.2, "Host I/O Errors," on page 53 for a description of these errors.
Checksum	2	One byte value used to validate the packet's integrity. See Section 4.17.3, "Checksums," on page 53 for a description.
Data (optional)	3–6	Zero to four 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 chip was responding. Data is always sent most significant byte (MSB) first.

4.17.2 Host I/O Errors

There are a number of checks that the chip makes on the command sent to the chip. These checks improve safety of the motion system by eliminating some obviously incorrect command data values. All checks associated with host I/O commands are referred to as host I/O errors. All unspecified codes are reserved.

Code	Indication	Cause
0	No error	No error condition.
1	73110 reset	Default value of error code on reset or power-up.
2	Invalid instruction	Instruction is not valid in the current context, or an illegal instruction code has been detected.
4	Invalid parameter	The parameter value sent to the motion processor was out of its acceptable range.
9	Bad checksum	The checksum compiled and returned by MC73110 does not match that sent by the host.

Serial Packet Example: **GetLoopIntegral** command

The use of asynchronous serial communication implies that each transmission byte will be wrapped in a frame containing extra bits for the start bit, parity bit and stop bit. Since these extra bits are always present in every frame, they will not be shown in the following example.

When using the **GetLoopIntegral** (velocity loop) command, the host must send six (6) frames to the MC73110, as shown in the following table:

Frame Number	Data
1	00h (address byte)
2	C2h (checksum)
3	00h
4	3Dh (op code)
5	00h
6	01h (velocity loop)

The host will then receive a response from the MC73110, which also contains six frames. The six response frames are detailed in the following table.

Frame Number	Data
1	00h (status)
2	36h (checksum)
3	02h
4	4Eh
5	7Ah
6	00h

These frames can be combined to form a single 32-bit value (024E7A00h). Chapter 5, "Instruction Reference," on page 67 shows that the value returned by this command is scaled by $1/2^8$. Therefore, the returned decimal value is 151,162 counts.

4.17.3 Checksums

Both command and response packets contain a checksum byte. The checksum is used to detect transmission errors, and allows the chip to identify and reject packets which have been corrupted during transmission, or which were not properly formed.

Any command packets sent to the chip containing invalid checksums will not be processed. This will result in a data packet being returned which contains 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). If the lower eight bits of the result are zero, then the checksum is valid.

For example, if a command packet is sent to chip address 3, containing instruction code 77h (**SetMotorCommand**) with the one word data value 1234h, then the checksum will be calculated by summing all bytes of the command packet ($03h + 00h + 77h + 12h + 34h = C0h$) and negating this to find the checksum value (40h). On receipt, the chip will sum all bytes of the packet and if the lower eight bits of the result are zero, then it will accept the packet ($03h + 40h + 00h + 77h + 12h + 34h = 100h$).

4.17.4 Baud Rate and Protocol

The MC73110 may be configured to operate at baud rates ranging from 1200 baud to 460,800 baud. In addition, it may be operated in point-to-point serial mode, or multi-drop idle-line mode. To set the serial port configuration, the command **SetSerialPortMode** is used. To read this value, the command **GetSerialPortMode** is used.

The following table shows the values of various parameters set using the **SetSerialPortMode** command.

Parameter	Encoding	
Transmission	0	1200 bits per second
Rate Selector	1	2400 bps
	2	9600 bps
	3	19,200 bps
	4	57,600 bps
	5	115,200 bps
	6	230,400 bps
	7	460,800 bps
Parity Selector	0	None
	1	Odd parity
	2	Even parity
	3	Reserved (do not use)
Number of Stop Bits	0	1 stop bit
	1	2 stop bits
Protocol Type	0	Point-to-point
	1	Reserved (do not use)
	2	Reserved (do not use)
	3	Multi-drop (idle line mode)
— (Reserved)	-	
Multi-drop	0	Address 0
Address Selector. Should be zero in point-to-point mode.	1	Address 1

	31	Address 31

The default configuration of the serial port is point-to-point, 57,600 baud, 1 stop bit, no parity. To change this, a **SetSerialPortMode** command can be sent while communicating by the serial port (which means the communication parameters of the host's serial port must also be changed to further communicate). This information may also be loaded during the power-up procedure by using either the serial EEPROM or the Flash mechanism.



Multi-drop support requires unique addresses for each mode on the network. The serial EEPROM mechanism is the best way to store these addresses.

4.17.5 Control Signals

Three signals, SerialXmt, SerialRcv, and SerialEnable mediate serial transfers. SerialXmt encodes data transmitted from the chip to the host processor. SerialRcv receives data sent from the host to the chip. SerialEnable output goes active (high) when data is being transmitted in multi-drop mode. This signal may be connected to the output enable pin of a serial buffer IC to allow tri-stating of the transmit line of a serial bus when not in use. In point-to-point mode, the SerialEnable pin is always active (high).

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

- One start bit.
- Eight data bits.
- An optional even/odd parity bit.
- One or two stop bits.

This data frame format is shown in Figure 4-17.

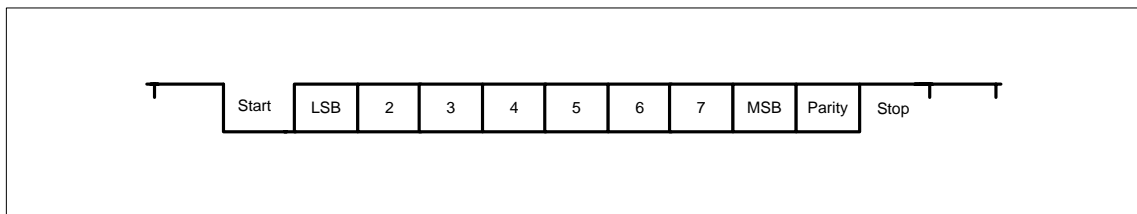


Figure 4-17:
Typical data
frame format

4.17.6 Transmission Protocols

Two serial transmission protocols are supported 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 latter method supports more than one chip on a serial bus, thereby allowing a chain, or network of chips to communicate on the same serial hardware signals. The following sections describe these transmission protocols.

4.17.7 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 chip. In this mode, the address byte is not used by the chip (except in the calculation of the checksum), and the chip 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 instruction code in the packet. Each instruction code has a specific amount of data associated with it. When the chip receives a packet, it waits for all data bytes to be received before processing the packet. The amount of data returned from any command is also determined by the instruction code. After processing a command, the chip will return a data packet of the necessary length.

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

and 1 stop bit. Therefore, one frame takes just over 1 millisecond, and the recommended minimum timeout is 2 milliseconds.

If the timeout period elapses between frames 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 chip. To resynchronize, the host should send a frame 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 chip, at which point the two processors will be synchronized.

4.17.8 Multi-drop Protocol

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

Because the address that starts a command packet is used to enable or disable the response from a chip in multi-drop mode, the multi-drop protocol must include a method to avoid losing synchronization between the host and the chip, because it would be difficult to regain synchronization in this environment. The method supported by the MC73110 is Idle Line mode.

Note that the multi-drop protocol may also be used when the host and chip 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. It will also follow any additional rules for the selected protocol. This mode of operation allows the host to be sure that it will remain synchronized with the chip without implementing the timeout and re-synch procedure previously outlined.

When the idle-line multi-drop protocol is used, the chip imposes tight timing requirements on the data sent as part of a command packet. In this mode, the chip 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 (roughly 1 millisecond at 9600 baud for example). 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.

4.18 Incremental Encoder Input

Incremental encoder input is supported by the MC73110 along with an index pulse and associated high speed capture system. Two square-wave signals: QuadA and QuadB are expected to be offset from each other by 90°, as shown in Figure 4-18 on page 57.

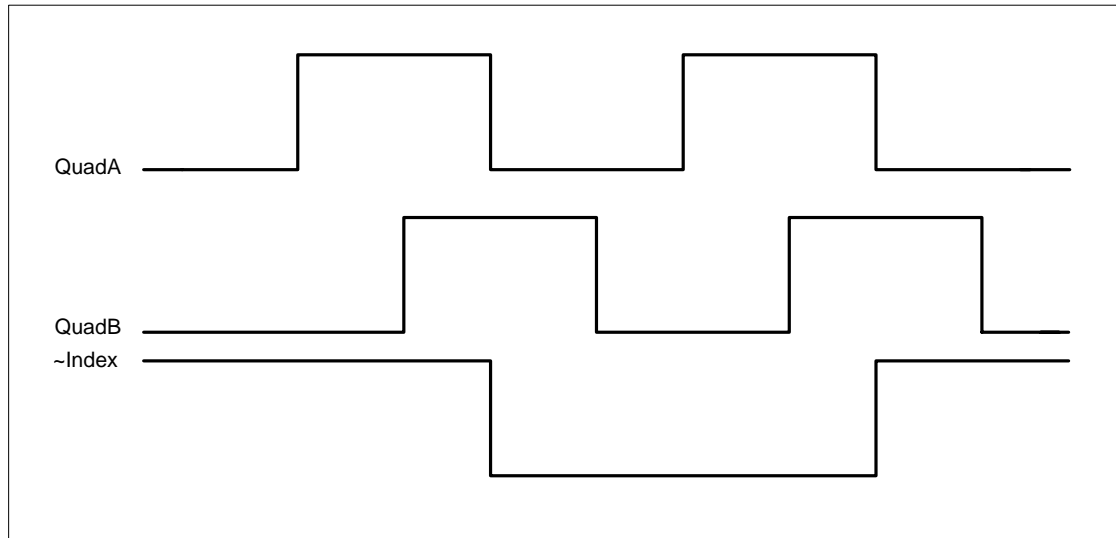


Figure 4-18:
QuadA, QuadB,
and Index
signals

When the motor moves in the positive direction, QuadA should lead QuadB. This allows the quadrature incremental position to be properly registered by the chip. When the motor moves in the negative direction, QuadB should lead QuadA. Because of the 90° offset, four resolved quadrature counts occur for one full phase of each A and B channel.

4.18.1 Actual Position Register

The chip continually monitors the position feedback signals, and accumulates a 32-bit position value called the Actual Position. Upon power-up, the default Actual Position is zero. The Actual Position can be explicitly set using the command **SetActualPosition**, and can be retrieved using the command **GetActualPosition**.

4.18.2 High Speed Position Capture

The MC73110 supports a high-speed position capture function that allows the instantaneous axis location to be captured, triggered by the index signal. A capture will be triggered when the index signal transitions to a high or low state (defined by the Signal Sense register using the **SetSignalSense** command). See Section 4.13.4, “Signal Sense Mask,” on page 46 for a description of the functionality of the **SetSignalSense** function.

The default value for the index sense mask is 0, meaning this signal is active low. In this condition, a capture will be recognized when index transitions low. Any change to the sense mask from active low interpretation to active high interpretation will cause a corresponding change in recognition of the trigger condition. When a capture is triggered, the contents of the actual position registers are transferred to the Position Capture register, and the capture-received indicator (Bit 3 of the Event Status register) is set. See Section 4.13, “Status Words,” on page 44 for more information on the Event Status register. 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, meaning that subsequent captures can occur. To clear the position capture indicator for all Event Status register bits, the command **ResetEventStatus** is used.

4.19 Serial EEPROM

To facilitate use of the MC73110 as a dedicated amplifier chip without need for communication from a host, a serial EEPROM is supported which can be used to load operational parameters into the chip before operation. Typically, after prototyping of your product with the MC73110, you will use one of the PMD software utilities to automatically build a serial EEPROM file, which can then be programmed and installed on the production product card.

A single generic I²C bus serial EEPROM device is supported. This device must be located at address 50h. See the table in Section 4.15, “Temperature Sensor,” on page 50 for more information on I²C.

After power-up or reset, the MC73110 interrogates the I²C bus to determine if a serial EEPROM is present. If an EEPROM is present, the MC73110 will begin reading data from the EEPROM. The data is interpreted as command packets in a similar format to the serial port command packets. After this process is complete, and the entire EEPROM has been read, normal chip processing will begin.

As each command is read from the EEPROM, the checksum for the command is validated prior to executing the command. If any command is corrupt, or if processing the command results in an error, the process is aborted. If this happens, the User Command Error bit is set in Event Status register, and the amplifier disable output is driven active. If the MC73110 is connected to a host using the serial port, the host can proceed by manually configuring the MC73110 and then clearing the User Command Error bit. Otherwise, the corrupt EEPROM or MC73110 Flash memory must be corrected before the MC73110 can be used.

4.19.1 EEPROM Command Format

The host commands stored in the EEPROM are expected to have a byte-oriented packet format very similar to the commands sent by the serial port. The only difference is that the address byte is always zero. In addition, there is no response packet in the Serial EEPROM initialization mode. Finally, to end the packet sequence, the last command should be the **NoOperation** command. This sequence indicates to the MC73110 that the packet sequence is completed.

For your convenience, PMD has a feature in Pro-Motor to convert MC73110 commands into a programming file suitable for a serial EEPROM. Refer to the *MC73110 Developer's Kit Manual*.

The following table shows the expected format in the serial EEPROM.

Field	Byte No.	Description
Address	1	Should always be zero.
Checksum	2	One byte value used to validate packet data.
Instruction number	3–4	Two byte instruction, sent upper byte (axis number) first. The axis number should always be set to zero.
Data (optional)	5–8	Zero to four bytes of data, sent most significant byte (MSB) first. See the individual command descriptions for details on data required for each command.

4.19.2 Storing User Commands to FLASH

In applications where a host processor is used for communications with the MC73110, an alternate method may be used to load operational parameters to replace the reset defaults. This method uses a region of the on-chip Flash memory of the MC73110 to store user commands to set the parameters. Similar to the method using the external EEPROM, the commands are stored in the Flash in command packet format. When the MC73110 boots up, it processes the command packets from the flash, modifying the indicated parameters from their reset defaults. A checksum is computed over each command packet, and the validity of the opcode and number and range of parameters checked. If any command packet fails these tests then flash command processing terminates and the User Command Error bit of the Event Status word is set. See Section 4.13, “Status Words,” on page 44 for more information on the Event Status word.

Approximately 8000 bytes of flash are available, enough to store all the commands required for setting every parameter on the chip.

To configure the Flash with user commands, the **StoreUserData** command is used. This command can only be sent when AmplifierDisable output is active.

After this command is issued, the host streams the desired command packets over the serial port to the MC73110. Prior to the first command, the host sends a word containing the total number of bytes that will follow. The final command in the sequence is required to be the **NoOperation** command. The format for each command packet is the same as that which is used for the EEPROM method. See the table in Section 4.19.1, “EEPROM Command Format,” on page 58 for details.

Upon receiving the **StoreUserData** command, and after receiving each byte of data that follows, the MC73110 responds by sending a null byte over the serial port. The timeout that the host uses for this handshake should be fairly long (for example, 10ms), as the **StoreUserData** command involves erasing and writing the Flash memory in the chip. Upon receiving the last byte of data, the MC73110 does not respond. Instead, it resets, so care should be taken in re-establishing serial communications after this sequence.

For example, to configure the MC73110 to boot with a sample time of 256 μ s and a LoopMode of 1 (current loop only), the following sequences would be used (all data in hex).

Bytes Sent	MC73110 Response
00 (ADR—can be non-zero if mult dropout)	No response
8F (Checksum of command packet)	No response
00	No response
71 (StoreUserData command)	00 (if AmplifierDisable active)
00 (number of bytes to follow high)	00
10 (number of bytes to follow low)	00
00	00
C7 (Checksum)	00
00	00
38 (SetSampleTime)	00
01 (256 high byte)	00
00 (256 low byte)	00
00	00
90 (Checksum)	00
00	00
6F (SetLoopMode)	00
00 (I high byte)	00
01 (I low byte)	00
00	00
00 (Checksum)	00
00	00
00 (NoOperation command)	No response; performs reset.

4.20 Synchronous Serial Input (SPI Port)

There are two options for providing a continuous hardware-level velocity or torque command to the MC73110. The first is an analog voltage present at pin AnalogCmd. The other is the SPI port located at pins DigitalCmdData and DigitalCmdClk. Because the SPI port is digital, and because it has a 16-bit resolution, versus the internal MC73110's A/D 10-bit resolution, this is the preferred method when using the MC73110 in this mode.

The velocity commands input by the SPI port consists of a sequence of offset binary (0 = $-\text{max}$, 8000h = 0, FFFFh = $+\text{max}$) 16-bit numbers without further protocol layering. The data should be sent MSB (most significant bit) first. At the maximum SPI input rate of 10 MHz, a complete 16-bit digital word can be transferred in 1.6 μ Sec. See Figure 3-4 on page 16 for more information.

Most motion systems which can output an SPI data stream will do so at a regular rate. Usually, this is the servo loop rate, which is once per 100 μ sec or slower. If the hardware system will be sending out SPI data at a higher rate, the

total rate should be limited to one update every 25 μsec . Note that the default sample time for the MC73110 is 102.4 μsec . Sending SPI data at a higher frequency than 102.4 μsec will have no effect, as 102.4 μsec is the maximum update rate.

Once the 16-bit command is input at the SPI port, it is made available to either the velocity loop, commutator module, or velocity integrator, depending on the loop configuration.

4.20.1 SPI Command Synchronization

Since the MC73110 does not have a chip select for its SPI input, care should be taken to insure the integrity of the SPI clock. In normal SPI mode, extra or missing SPI clocks can cause permanent loss of synchronization in the SPI data stream, and hence erroneous commands being received by the MC73110. These extra or missing clocks can be caused by noise or excessive capacitive loading on the SPI clock signal. The system design should take all precautions to guarantee a clean SPI clock.

To enhance SPI integrity, a special mode can be enabled that allows the MC73110 to recover synchronization in the SPI data stream in the case of missing or extra SPI clocks. This mode is enabled using the **SetSPISyncMode** command. This mode should only be used if the SPI data stream meets the following timing requirements:

- 1 SPI clock rate at least 1MHz
- 1 SPI command rate is 10 kHz or less
- 1 SPI commands are issued at constant rate

While there can be some jitter in the SPI command rate, the critical requirement is that there is always at least 51.2 μsec between the end of the previous SPI command and the beginning of the next one.

4.21 Analog Signal Processing

Four direct analog signals may be input to the MC73110. These signals are summarized in the following table.

Signal Name	Function
CurrentA	Phase A instantaneous current through motor coil.
CurrentB	Phase B instantaneous current through motor coil.
AnalogCmd	Desired velocity or torque command input to velocity loop.
Tachometer	Instantaneous motor velocity, generally derived from a tachometer

Each of these signals are converted internally in the MC73110 using a 10-bit A/D. To read these signals, the command **GetAnalog** is used. To offset these values, the commands **SetAnalogOffset** and **GetAnalogOffset** are used.

If one or more of these analog signals are being used, a number of additional signals must also be connected. These signals are summarized in the following table.

Signal Name	Function
AnalogVcc	Provides 3.3V power to on-chip analog circuitry.
AnalogGnd	Provides return path for on-chip analog circuitry.
AnalogRefHigh	Provides high voltage level for A/D circuitry. Usually connected to 3.3V supply.
AnalogRefLow	Provides low voltage level for A/D circuitry. Usually connected to analog return (0.0 V).

The A/D output is monotonic from $-32,736$ (8020h) to $32,736$ (7FE0h) when the input signal is between the analog low and high voltage reference. When the input signal is higher than analog high reference voltage, the A/D output is $32,736$ (7FE0h); when the input signal is lower than the analog low reference voltage, the A/D output is $-32,736$ (8020h).

Variations in the analog high and analog low reference must be less than half LSB of the target resolution (A/D conversion resolution) during the conversion time in order to ensure the specified performance.

4.21.1 Analog Signal Range & Representation

The voltages presented at the four analog input pins (CurrentA, CurrentB, AnalogCmd, Tachometer) must always be positive, ranging at their lowest value from AnalogRefLow, to their highest value AnalogRefHigh. These analog signals are assumed to represent bipolar, symmetric values (centered around zero volts), so the voltages present at these pins are interpreted as follows.

A voltage of AnalogRefLow is the smallest possible negative voltage. A voltage of AnalogRefHigh is the largest possible positive voltage. A value of $(\text{AnalogRefHigh} - \text{AnalogRefLow}) / 2$ represents a value of zero volts. When presenting bipolar signals, such as the current through each motor coil to the MC73110 input pins corresponding to those signals, the appropriate external circuitry should be installed to shift and rescale those signals.

4.21.2 Converting Voltages to Values

Assuming that AnalogRefHigh is 3.3V, and AnalogRefLow is 0.0V, to determine the numerical value that will be read by the motion processor given a specific voltage at the MC73110's input pin, the following formula is used.

$$\text{ReadValue} = (32,736 * \text{AnalogVoltage} / 1.65\text{V}) - 32,736$$

For example, if the analog voltage is 3.3V, the read value will be 32736 (7FE0h). If the analog voltage is 0.0V, the read value will be -32736 (8020h), and if the analog voltage is 1.65V, the read value will be 0 (zero).

Conversely, given a read value, the voltage at the connection is calculated as:

$$\text{AnalogVoltage} = 1.65\text{V} * (\text{ReadValue} + 32,736) / 32,736$$

4.21.3 Current Sensing

A special instance of analog input occurs for the CurrentA and CurrentB signals. While the input voltage range is the same (from AnalogRefLow to AnalogRefHigh), the sense of the provided current signals must be correct for current sensing to function properly. This sense is non-reversed, meaning that a positive voltage output at the PWM coil (greater than 50% duty cycle) should result in a positive current signal being fed back to the chips. Depending on the type of op amps used in the circuit, it is not unusual for this sense to be reversed. Care should be taken to ensure that the relationship between the voltage output and current feedback is correct. A positive voltage potential should be generated on the respective motor winding when GetPWMCommand is negative.

4.21.4 PWMOutputDisable Signal

Under certain error conditions, it may be desirable to immediately disable the PWM output. A dedicated input is provided for this function. The hardware designer must include circuitry external to the MC73110 for determining if PWM output should be disabled. ~PWMOutputDisable is an active low signal, and when brought low will immediately tri-state all PWM output signals. When the signal is in the high state, PWM output is enabled and operates normally. In designs where this signal is not used, it may remain unconnected.

4.22 GetLoop Commands and Variables

4.22.1 Read Loop Variables

The MC73110's flexible control loop structure may be configured in a number of ways. When it is enabled, the control loop filter will calculate the output based on variables such as instantaneous error and error integral. These loop variables may be read at any time using the commands **GetLoopCommand**, **GetLoopError**, and **GetLoopIntegral**. Each control loop is assigned a unique loop ID so that these commands may be applied to all loops.

4.22.2 Read Variables in the Current Loop

There are two current loops: CurrentA regulates the current in motor coil A, and CurrentB regulates the current in motor coil B. When FOC is used, the current loops are CurrentD, which regulates the direct, or magnetization current, and CurrentQ, which regulates the Quadrature, or torque producing current.

Loop ID	Loop
2	CurrentA Current D (with FOC)
3	CurrentB CurrentQ (with FOC)

Figure 4-19 shows how the current loop variables can be read.

Figure 4-19:
The current loop

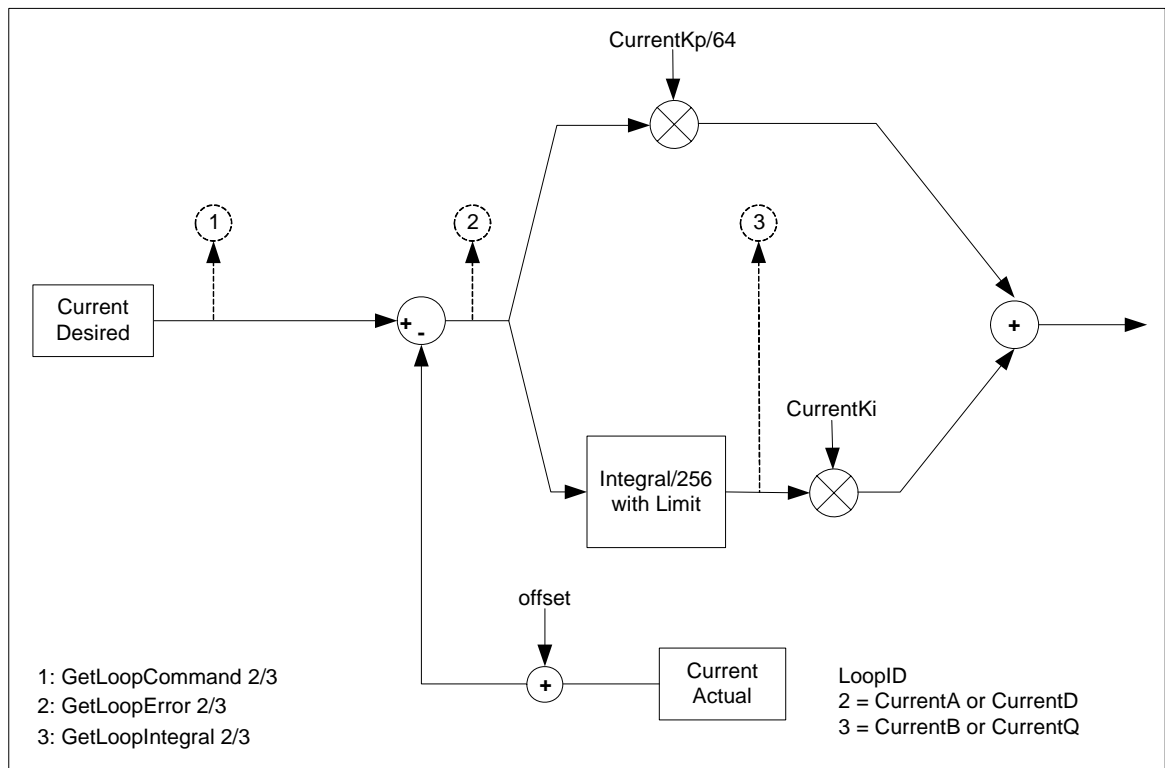


Figure 4-19 represents the filter equation described in Section 4.6.1, “Current Loop Filter,” on page 30 and the dashed arrow lines show the location to which the returned value corresponds in the filter calculation. For example, **GetLoopCommand 2** returns the loop command of loop CurrentA or CurrentD. **GetLoopError 3** returns the loop error of loop CurrentB or CurrentQ.

GetLoopCommand 2 and **GetLoopCommand 3** return the desired current for CurrentA or CurrentD and CurrentB or CurrentQ respectively, which are also the output values of the commutation portion. See Section 4.7, “Commutation,” on page 32 for detailed information. The returned value is a signed 16-bit number.

GetLoopError 2 and **GetLoopError 3** return current error for CurrentA or CurrentD and CurrentB or CurrentQ, respectively. The returned value is a signed 16-bit number.

GetLoopIntegral 2 and **GetLoopIntegral 3** return the integrated current error with a scaling of $1/2^8$, which is:

$$\frac{\sum_{j=0}^n CE_j}{256}$$

for CurrentA or CurrentD and CurrentB or CurrentQ, respectively. The returned value is a signed 16-bit number.

4.22.3 Read Variables in the Velocity Loop

The loop ID of the velocity loop is 1. With loop ID 1 specified, the velocity loop variables may be read, as shown in Figure 4-20.

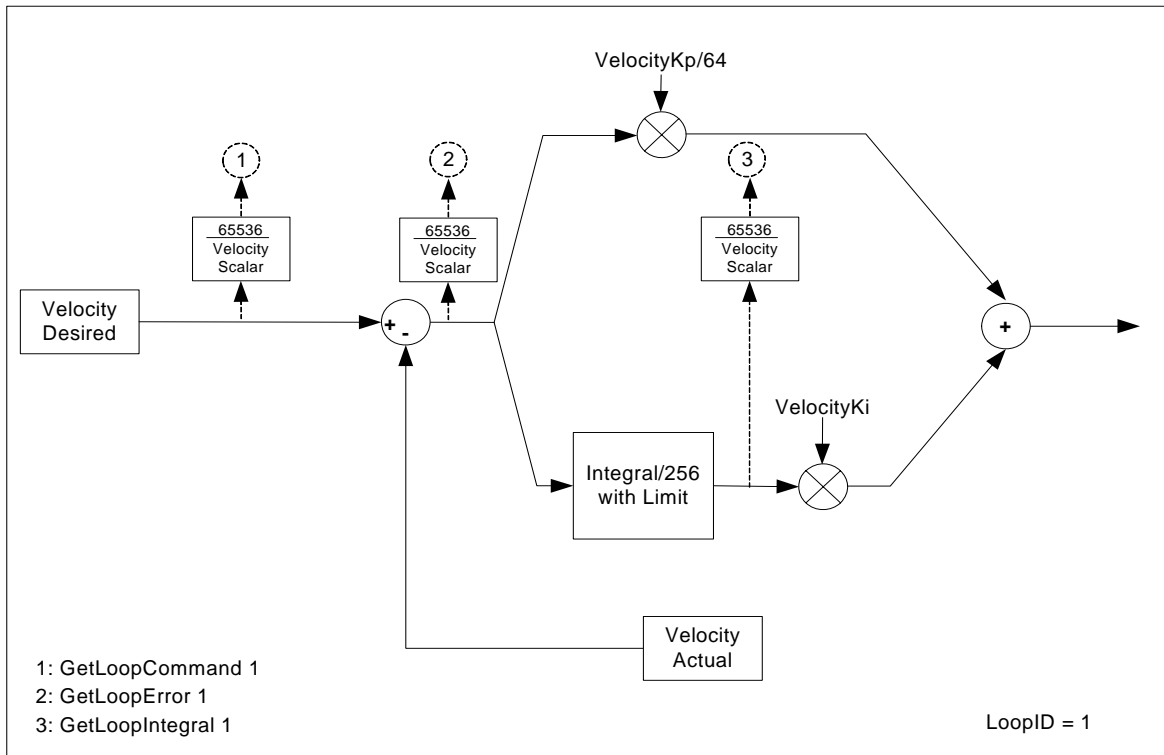


Figure 4-20:
The velocity
loop

Figure 4-20 represents the filter equation in Section 4.9.3, “Velocity Loop Filter,” on page 38 and the dashed arrow lines shows the location to which the returned value corresponds in the filter calculation. For example,

GetLoopCommand 1 returns the desired velocity of the velocity loop.

When the **CommandSource** is set to ProfileGenerator, **GetLoopCommand I** returns the desired velocity of the velocity loop without velocity scalar. The returned value is a signed 32-bit number in 16.16 format. The result does not include the velocity scalar; therefore, it has the same unit as the feedback signal. If the feedback signal is from the encoder, the returned value is the desired velocity in counts per cycle. For example, a velocity command of 123,863 corresponds to a velocity of 123,863/65,536, or 1.89 counts/cycle. If the feedback signal is from the tachometer, the returned value is the desired A/D value of the tachometer signal, divided by the velocity scalar.

When the **CommandSource** is set to AnalogCmd or SPI, **GetLoopCommand I** returns the value as a signed 32-bit number in 16.16 format with the velocity scalar divided out.

When the **VelocityFeedbackSource** is set to encoder then **GetLoopError I** returns the velocity error without velocity scalar. The returned value is a signed 32-bit number in 16.16 format. The result does not include the velocity scalar; therefore, it has the same unit as the feedback signal. If the feedback signal is from the encoder, the returned value is the velocity error in counts per cycle. For example, a velocity error of 234,618 corresponds to a velocity error of 234,618/65,536, or 3.58 counts/cycle. If the feedback signal is from the tachometer, the velocity error is in the unit of the digitalized tachometer signal, divided by the scalar.

GetLoopIntegral I returns the integrated velocity error with a scaling of $1/2^8$, which is:

$$\frac{\sum_{j=0}^n VE_j}{256}$$

with the velocity scalar. The returned value is a signed 32-bit number.

GetActualVelocity returns the velocity being measured by the velocity feedback device in a signed 16.16 format.

When the **VelocityFeedbackSource** is set to Encoder the returned value is “counts/cycle” in a 16.16 format. When the **VelocityFeedbackSource** is set to Tachometer the returned value is a signed 16.16 format divided by the Velocity Scalar.

4.22.4 Read Variables in the Velocity Integrator Loop

The loop ID of the velocity integrator loop is 0. With loop ID 0 specified, the velocity integrator loop variables may be read, as shown in Figure 4-21 on page 65.

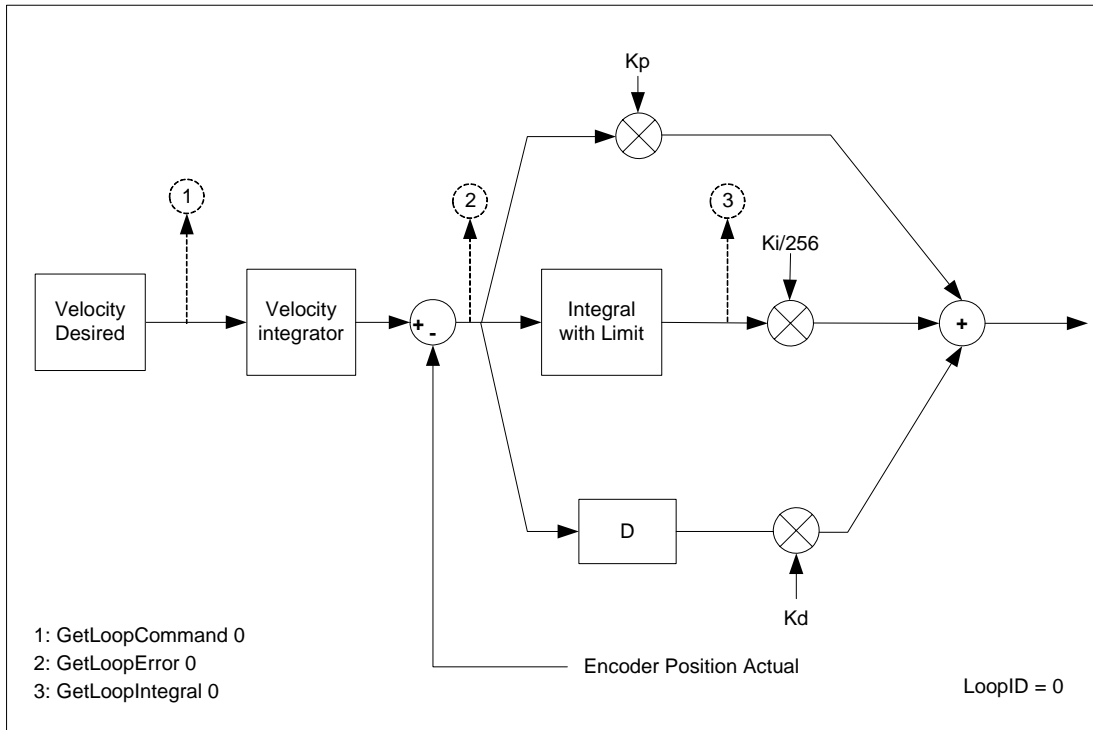


Figure 4-21:
The velocity
integrator loop

Figure 4-21 represents the filter equation in Section 4.10.1, “Velocity Integrator Loop Filter,” on page 41 with the velocity integrator, and the dashed arrow lines shows the location to which the returned value corresponds in the filter calculation. For example, **GetLoopCommand 0** returns the input to the velocity integrator.

GetLoopCommand 0 returns the velocity command being input to the velocity integrator without the velocity scalar. The returned value is a signed 32-bit number in 16.16 format. The result does not include the velocity scalar, and it has a unit of count/cycle.

GetLoopError 0 returns the position error. The returned value is a signed 32-bit number in counts.

GetLoopIntegral 0 returns the integrated position error, which is:

$$\sum_{j=0}^n PE_j$$

This page intentionally left blank.

5. Instruction Reference

5

5.1 How to Use This Reference

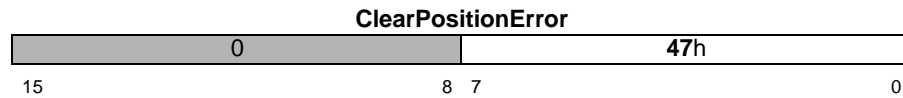
In this section, instructions are arranged alphabetically; except that all “Set/Get” pairs (for example, **SetVelocity** and **GetVelocity**) are described together. Each description begins on a new page; most occupy no more than a single page. Each page is organized as described in the following table.

Name	The instruction mnemonic is shown at the left, its hexadecimal code at the right.
Syntax	The instruction mnemonic and its required arguments are shown with all arguments separated by spaces.
Arguments	<p>There are two types of arguments: encoded-field and numeric.</p> <p>Encoded-field arguments are packed into a single 16-bit data word. Bits 8–15 of the instruction word are always 0. The name of the argument (in <i>italic</i>) is the name shown in the generic syntax. Instance (in <i>italic</i>) is the mnemonic used to represent the data value. Encoding is the value assigned to the field for that instance.</p> <p>For numeric arguments, the parameter value, the type (signed or unsigned integer), and range of acceptable values are given. Numeric arguments may require one or two data words. For 32-bit arguments, the high-order part is transmitted first.</p>
Packet Structure	<p>This is a graphic representation of the 16-bit words transmitted in the packet: the instruction, which is identified by its name, followed by one, two, or three data words. Bit numbers are shown directly below each word. For each field in a word, only the high and low bits are shown. For 32-bit numeric data, the high-order bits are numbered from 16 to 31, the low-order bits from 0 to 15. A packet consists of two serial frames.</p> <p>The hex code of the instruction is shown in boldface.</p> <p>Argument names are shown in their respective words or fields.</p> <p>For data words, the direction of transfer—read or write— is shown at the left of the word's diagram.</p> <p>Unused bits are shaded. All unused bits must be 0 in data words and instructions sent (written) to the motion IC.</p>
Description	Describes what the instruction does, and includes any special information relating to the instruction.
Restrictions	Describes the circumstances in which the instruction is not valid; that is, when it should not be issued.
see	Refers to related instructions, information, or topics in this manual.

Syntax **ClearPositionError**

Arguments None

**Packet
Structure**



Description

Executing this command will reset the velocity integrator target position equal to the current position. This command can be used when the axis is at rest, or when it is moving.

Restrictions

ClearPositionError should only be used when the velocity integrator loop is enabled. If this loop is not enabled, then this command has no effect.

see

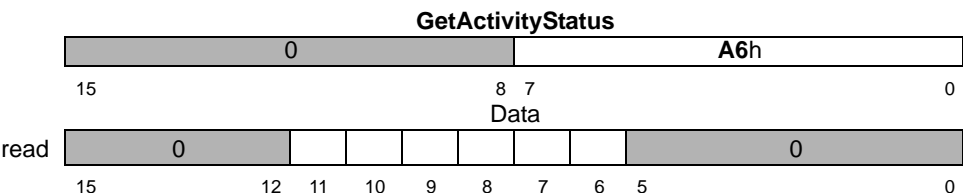
GetLoopCommand ([p. 76](#)), **GetMotionErrorLimit** ([p. 97](#))

Syntax **GetActivityStatus**

Arguments None

Returned data **Name** **Type**
 see below unsigned 16-bit

Packet Structure



Description **GetActivityStatus** reads the 16-bit Activity Status register. Each of the bits in this register continuously indicate the state of the motion IC without any action on the part of the host. Because the bits in this register are controlled by the chip, there is no direct way to set or clear the state of these bits.

The following table shows the encoding of the data returned by this command.

Name	Bit(s)	Description
—	0–5	Reserved
Overtemperature	6	Set (1) when the overtemperature condition is active, cleared (0) when the overtemperature condition is not active. The overtemperature condition is controlled by the maximum temperature register and the command SetTemperatureLimit . This bit is also Set(1) if the temperature sensor is not present.
PWMDisable	7	Set (1) when PWMDisable condition mask evaluates to True.
Motor Mode	8	Set (1) when motor mode is on; 0 when off.
Position Capture	9	Set (1) when a value has been captured by the high speed position capture hardware, but has not yet been read.
Overvoltage	10	Set (1) when bus voltage exceeds the overvoltage limit.
Undervoltage	11	Set (1) when bus voltage is less than the undervoltage limit.
—	12–15	Reserved

Restrictions

see **GetEventStatus** (p. 74) **GetSignalStatus** (p. 80)

Syntax

GetAnalog *portID*

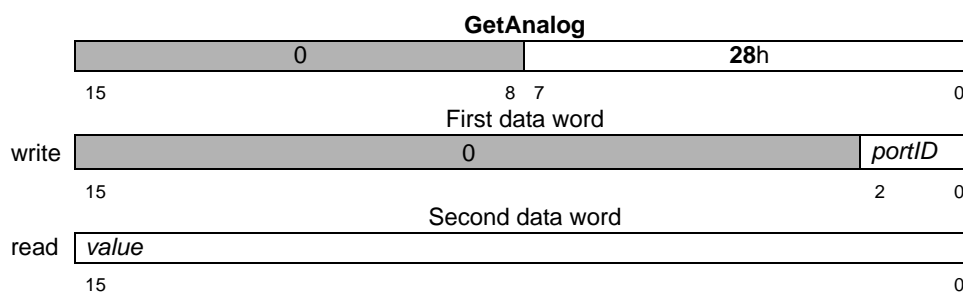
Arguments

Name	Instance	Encoding
<i>portID</i>	<i>AnalogCmd</i>	0
	<i>Tachometer</i>	1
	<i>CurrentA</i>	2
	<i>CurrentB</i>	3

Returned data

Type	Range	Scaling
signed 16-bit	-2^{15} to $2^{15}-1$	unity

Packet Structure



Description

GetAnalog returns a 16-bit value representing the voltage (read by an on-chip 10-bit A/D) presented to the specified analog input port. The value returned is the result of summing the raw value with the analog offset which was set using **SetAnalogOffset**.

Restrictions

see

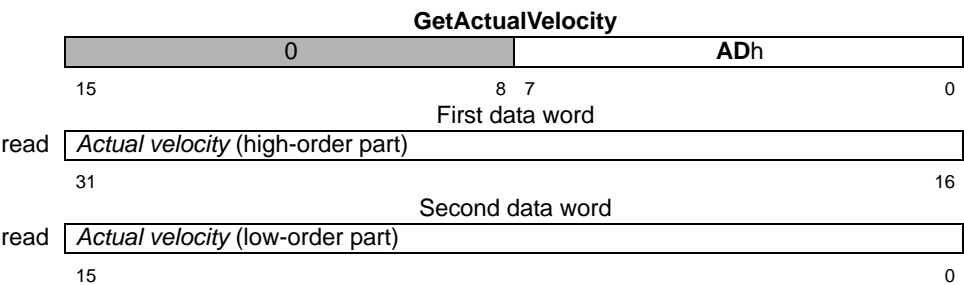
SetAnalogOffset/GetAnalogOffset ([p. 88](#))

Syntax **GetActualVelocity**

Arguments None

Returned data	Name	Type	Range	Scaling	Units
	<i>Actual velocity</i>	signed 32-bit	-2^{31} to $2^{31}-1$	$1/2^{16}$	counts/cycle

Packet Structure



Description **GetActualVelocity** reads the velocity. Actual velocity is determined using either the analog velocity input, the incremental encoder input, or the Hall sensor inputs. Scaling example: if a value of 1,703,936 is retrieved by the **GetActualVelocity** command (high word: 01Ah, low word: 0h), this corresponds to a velocity of 1,703,936/65,536; or 26 counts/cycle.

When the velocity feedback source is set to Tachometer, the returned value will represent the velocity in a 16.16 format after the user multiplies the returned value by the velocity scalar.

Restrictions

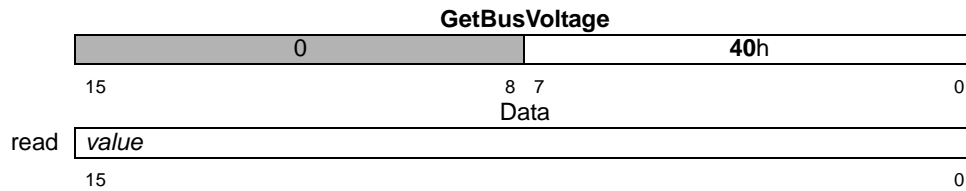
see **SetVelocityFeedbackSource/GetVelocityFeedbackSource** (p. 117)

Syntax **GetBusVoltage**

Arguments None

Returned data	Name	Type	Range
	<i>value</i>	unsigned 16-bits	0 to $2^{16}-1$

**Packet
Structure**



Description **GetBusVoltage** gets the most recent Bus voltage reading. The scaling of the returned value depends on the scaling used in Bus voltage sensor.

Restrictions **GetBusVoltage** will only return valid information if analog Bus voltage signal is connected.

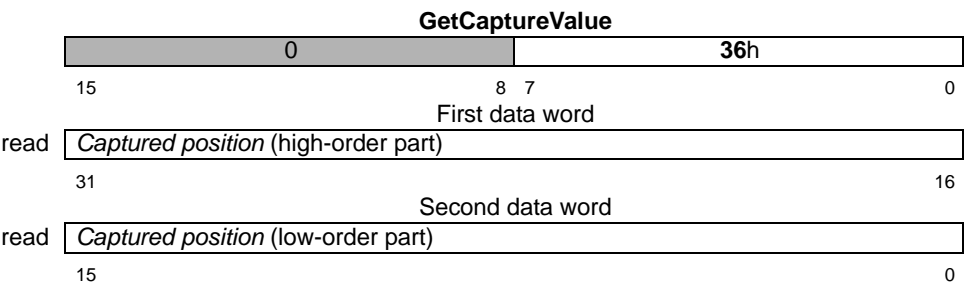
see

Syntax **GetCaptureValue**

Arguments None

Returned data	Name	Type	Range	Scaling	Units
	<i>Captured position</i>	signed 32-bit	-2^{31} to $2^{31}-1$	unity	counts

Packet Structure



Description **GetCaptureValue** returns the contents of the Position Capture register, and clears the position capture bit in the Activity Status register. This command also resets the capture hardware, allowing another capture to occur.

Restrictions An encoder with index must be present.

see

Syntax

GetEventStatus

Arguments

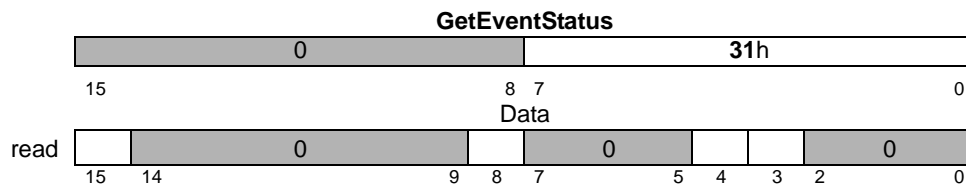
None

Returned data

Name
see below

Type
unsigned 16-bit

Packet Structure



Description

GetEventStatus reads the Event Status register. The encoding of the data returned by this command is outlined in the following table.

Name	Bit(s)	Description
—	0–2	Reserved
Wrap-around	1	Set(1) when the actual (encoder) position has wrapped from maximum allowed position to minimum, or vice versa.
Capture Received	3	Set(1) when a position capture has occurred.
Motion Error	4	Set(1) when a motion error has occurred.
—	5–7	Reserved
Amplifier Error	8	Set when the programmable amplifier error condition becomes true.
—	9–14	Reserved
User Command Error	15	Set (1) if error occurred processing user commands from EEPROM or Flash after chip reset.

Restrictions

All of the bits in this status word are set by the chip and cleared by the host. To clear these bits, use the **ResetEventStatus** command.

see

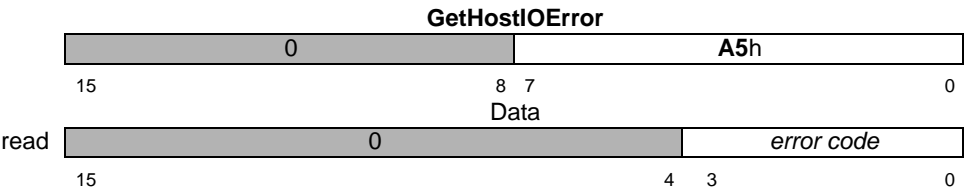
GetActivityStatus (p. 69), **GetSignalStatus** (p. 80)

Syntax **GetHostIOError**

Arguments None

Returned data	Name	Type	Range
	see below	unsigned 16-bit	0–15

Packet Structure



Description **GetHostIOError** returns the code for the last host I/O error, and then resets the error to zero. The error codes are encoded as defined in the following table.

Error Code	Encoding
No error	0
73110 Reset	1
Invalid instruction	2
— (Reserved)	3
Invalid parameter	4
— (Reserved)	5–8
Bad checksum	9
— (Reserved)	10–15

Restrictions

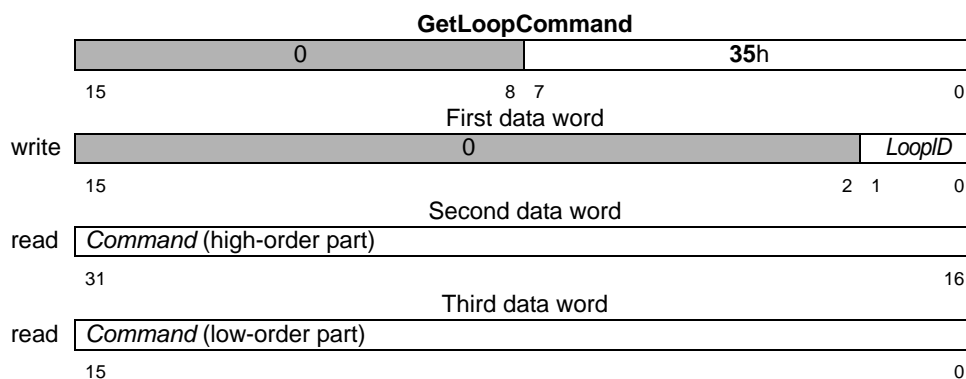
see **GetEventStatus** (p. 74)

Syntax**GetLoopCommand** *loopID***Arguments**

Name	Type	Instance	Encoding (hex)
<i>LoopID</i>	unsigned 16-bit	<i>Velocity Integrator Loop</i>	0
		<i>Velocity Loop</i>	1
		<i>CurrentA or D Loop</i>	2
		<i>CurrentB or Q Loop</i>	3

Returned data

Command	Type	Range	Scaling	Units
	signed 32-bit	-2^{15} to $2^{15}-1$	see below	see below

Packet Structure**Description**

GetLoopCommand returns the input command for the selected *LoopID*. The following table shows the values which are returned for each *LoopID*.

LoopID	Format	Range	Description
<i>Velocity Integrator Loop</i>	16.16	-2^{15} to $2^{15}-1$	Commanded velocity.
<i>Velocity Loop</i>	16.16	-2^{15} to $2^{15}-1$	Commanded velocity.
<i>CurrentA or D Loop</i>	32.0	-2^{15} to $2^{15}-1$	Commanded current for winding #1 (when in A/B mode) or Commanded Direct (D) current (when in FOC mode).
<i>CurrentB or Q Loop</i>	32.0	-2^{15} to $2^{15}-1$	Commanded current for winding #2 (when in A/B mode) or Commanded Quadrature (Q) current (when in FOC mode).

When the loop command source is set to Analog or SPI, the returned value will represent the velocity reference command in a 16.16 format after the user multiplies the returned value by the velocity scalar.

Restrictions

If the selected *LoopID* is disabled (**SetLoopMode**), this command will return zero.

see

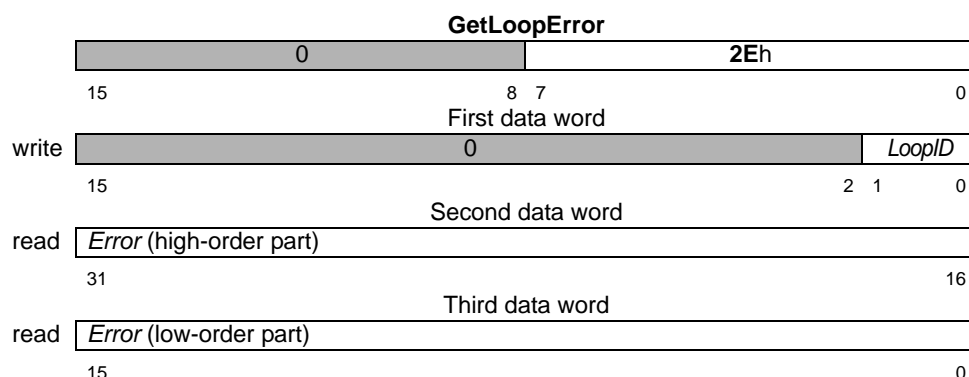
GetLoopError (p. 77)

Syntax **GetLoopError** *loopID*

Arguments	Name	Type	Instance	Encoding (hex)
	<i>LoopID</i>	unsigned 16-bit	<i>Velocity Integrator Loop0</i>	
			<i>Velocity Loop</i>	1
			<i>CurrentA or D Loop</i>	2
			<i>CurrentB or Q Loop</i>	3

Returned data	Error	Type	Range	Scaling	Units
		signed 32-bit	see below	see below	see below

Packet Structure



Description

GetLoopError returns the error value for the selected *LoopID*. The following table shows the values which are returned for each *LoopID*.

LoopID	Format	Range	Description
<i>Velocity Integrator loop</i>	32.0	-2^{31} to $2^{31}-1$	Position error.
<i>Velocity Loop</i>	16.16	-2^{15} to $2^{15}-1$	Velocity error.
<i>CurrentA or D Loop</i>	32.0	-2^{15} to $2^{15}-1$	Motor current error for winding #1 (when in A/B mode) or Direct (D) current error (when in FOC mode).
<i>CurrentB or Q Loop</i>	32.0	-2^{15} to $2^{15}-1$	Motor current error for winding #2 (when in A/B mode) or Quadrature (Q) current error (when in FOC mode).

When the velocity feedback source is set to Tachometer, the returned value will represent the velocity error in a 16.16 format after the user multiplies the returned value by the velocity scalar.

Restrictions

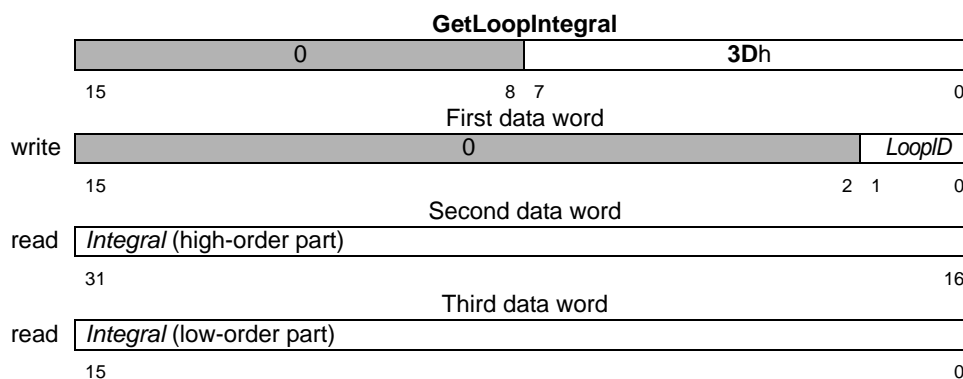
If the selected *LoopID* is disabled (**SetLoopMode**), this command will return zero.

see

GetLoopCommand (p. 76)

Syntax	GetLoopIntegral <i>loopID</i>				
Arguments	Name	Type	Instance	Encoding (hex)	
	<i>LoopID</i>	unsigned 16-bit	<i>Velocity Integrator Loop</i>	0	
			<i>Velocity Loop</i>	1	
			<i>CurrentA or D Loop</i>	2	
			<i>CurrentB or Q Loop</i>	3	
Returned data		Type	Range	Scaling	Units
	<i>Integral</i>	signed 32-bit	-2^{15} to $2^{15}-1$	see below	see below

Packet Structure



Description

GetLoopIntegral returns the integrated error value for the selected *LoopID*. When the *LoopID* is set to velocity integrator loop, the value returned is a full 32-bit value. For all other *LoopIDs*, the value returned is a sign-extended 16-bit value scaled by 1/256. The same scaling factors are applied to the integrator limit values used by **Get/SetLoopGain**.

GetLoopIntegral can be used to monitor loading on the axis, because changes in the axis loading can be reflected in the values returned by this command.

Scaling example: if a constant velocity error of 100 counts is present for 256 cycles, then the total accumulated integral value will be 100 (100*256/256). A returned value of 1,000 indicates a total stored value of 256,000 counts (1,000*256). The following table shows the values which are returned for each *LoopID*.

LoopID	Format	Range	Scaling	Description
<i>Velocity Integrator Loop</i>	32.0	-2^{31} to $2^{31}-1$	unity	Integrated position error.
<i>Velocity Loop</i>	16.16	-2^{15} to $2^{15}-1$	1/256	Integrated velocity error divided by the velocity scalar.
<i>CurrentA or D Loop</i>	32.0	-2^{15} to $2^{15}-1$	1/256	Integrated motor current error for winding #1 (when in A/B mode) or integrated Direct (D) current error (when in FOC mode).
<i>CurrentB or Q Loop</i>	32.0	-2^{15} to $2^{15}-1$	1/256	Integrated motor current error for winding #2 (when in A/B mode) or integrated Quadrature (Q) current error (when in FOC mode).

Restrictions

If the selected *LoopID* is disabled (**SetLoopMode**), this command will return zero.

see

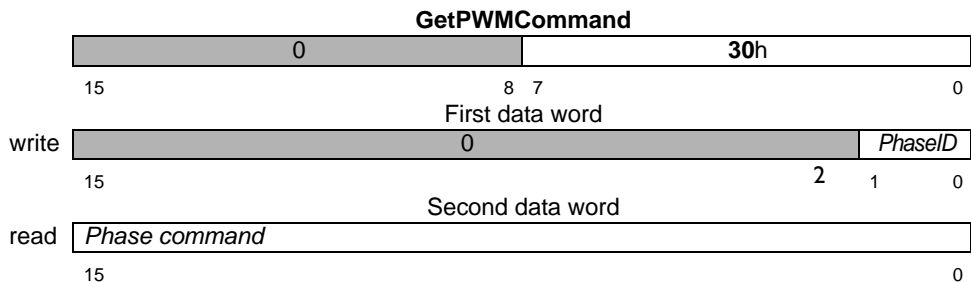
Get/SetLoopGain (p. 94)

Syntax **GetPWMCommand** *PhaseID*

Arguments	Name	Type	Instance	Encoding
	<i>PhaseID</i>	unsigned 16-bit	<i>PhaseA</i>	0
			<i>PhaseB</i>	1
			<i>PhaseC</i>	2

Returned data	<i>Phase command</i> signed 32-bit	Range	Scaling	Units
		-2^{15} to $2^{15}-1$	$100/2^{15}$	%output

Packet Structure



Description

GetPWMCommand returns the value of the motor output command for phase A, B, or C. These are the phase values directly output to the motor, regardless of commutation mode, motor mode, or loop mode.

Scaling example: if a value of $-4,489$ is retrieved (EE77h) for a given phase, then this corresponds to $-4,489 \times 100 / 32,768 = -13.7\%$ of full-scale output. The sign relationship between **GetPWMCommand** and the voltage polarity applied to the motor winding is reversed. For example, a positive voltage should be present on motor terminal phase A when **GetPWMCommand** 0 returns a negative value. This relationship is required for proper operation of the current loop.

Restrictions

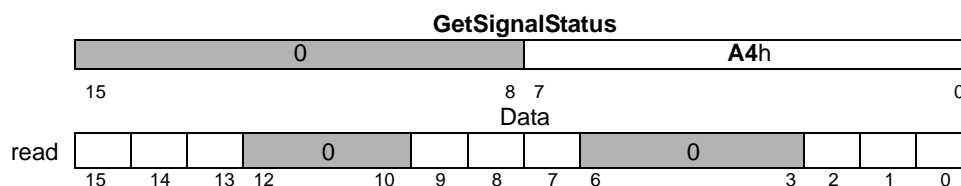
see

Syntax GetSignalStatus

Arguments None

Returned data see below **Type** unsigned 16-bit

Packet Structure



Description

GetSignalStatus returns the contents of the Signal Status register. The Signal Status register contains the value of the various hardware signals connected to the motion processor. The value read is combined with the Signal Sense register (see **SetSignalSense**), and then returned to the user. For each bit in the Signal Sense register that is set to 1, the corresponding bit in the **GetSignalStatus** command will be inverted. Therefore, a low signal will be read as 1, and a high signal will be read as 0. Conversely, for each bit in the Signal Sense register that is set to 0, the corresponding bit in the **GetSignalStatus** command is not inverted. Therefore, a low signal will be read as 0, and a high signal will be read as 1.

The bit definitions are as follows.

Description	Bit Number
QuadA	0
QuadB	1
Index	2
— (Reserved)	3–6
Hall1	7
Hall2	8
Hall3	9
— (Reserved)	10–12
Estop	13
PWMOutputDisable	14
AmplifierDisable	15

Restrictions

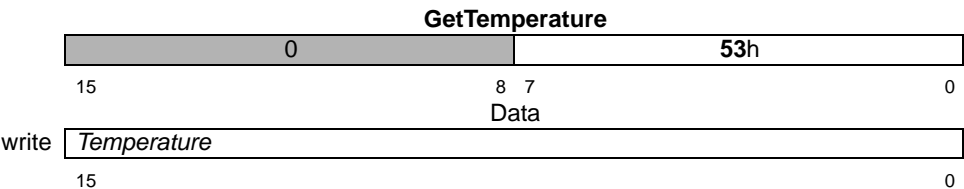
see **GetActivityStatus** (p. 69), **GetEventStatus** (p. 74), **SetSignalSense/GetSignalSense** (p. 113)

Syntax **GetTemperature**

Arguments None

Returned data	Name	Type	Range
	<i>Temperature</i>	signed 16-bit	-2^{15} to $2^{15}-1$

**Packet
Structure**

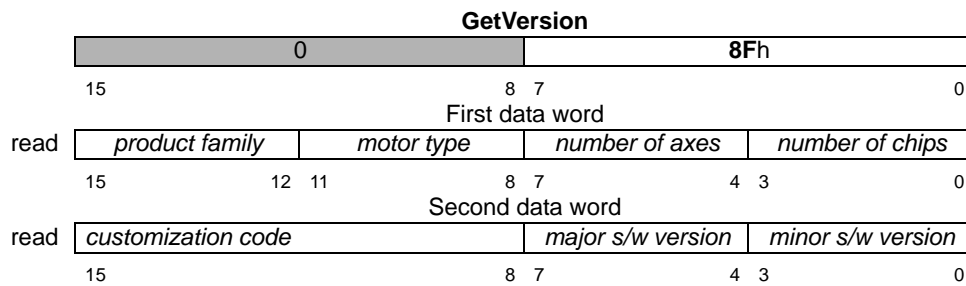


Description **GetTemperature** returns the value read from the temperature sensor right shifted by 4.

Restrictions If no temperature sensor is present, this command returns 0. The overtemperature bit in the Activity Status register will be Set(1).

see

Packet Structure



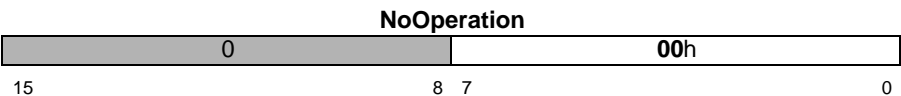
Restrictions

see

Syntax **NoOperation**

Arguments None

**Packet
Structure**



Description The **NoOperation** command has no effect on the IC. This command may be used to resynchronize communications with the chip.

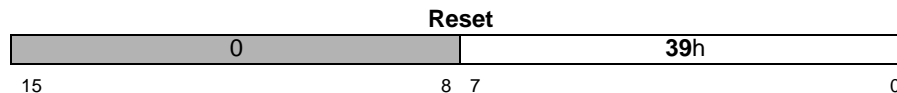
Restrictions

see

Syntax **Reset**

Arguments None

**Packet
Structure**



Description

Reset restores the chip to its initial condition, setting all chip variables to their default values. The table below lists the default values for all internal registers. If an external EEPROM or internal Flash is used to store additional startup commands, the values shown in the following table will be overwritten immediately following reset.

Variable Name	Default Value
<i>Acceleration</i>	0
<i>ActualPosition</i>	0
<i>AmplifierDisable</i>	0
<i>AmplifierError</i>	0
<i>AnalogOffset(s)</i>	0
<i>AutoStopMode</i>	1
<i>CommandSource</i>	0
<i>CommutationMode</i>	1 (Hall)
<i>LoopGain(s)</i>	0 (All cleared)
<i>MotionErrorLimit</i>	$2^{31} - 1$
<i>MotorCommand</i>	0
<i>MotorLimit</i>	32767
<i>MotorMode</i>	1 (On)
<i>OverVoltageLimit</i>	65535
<i>PhaseAngle</i>	0
<i>PhaseCounts</i>	1
<i>PhasePrescale</i>	0 (Off)
<i>PWMDeadTime</i>	55 (12μs)
<i>PWMLimit</i>	31784 (97%)
<i>PWMOutputDisable</i>	default 2000h
<i>PWMOutputMode</i>	0 (Six signal)
<i>PWMSense</i>	0 (All active low)
<i>SampleTime</i>	102
<i>SerialPortMode</i>	04h (57600 baud, NoParity, 1 stop bit)
<i>SignalSense</i>	0
<i>SPISyncMode</i>	0 (off)
<i>Velocity</i>	0
<i>VelocityFeedbackSource</i>	0 (Encoder)
<i>VelocityScalar</i>	3
<i>TemperatureLimit</i>	32767
<i>UnderVoltageLimit</i>	0

Restrictions

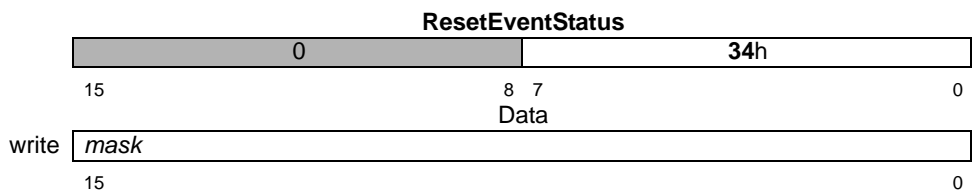
The command response is issued prior to reset. Further communications should not be attempted until after the reset time has elapsed.

see Section 3.4, “Timing Diagrams,” on page 15.

Syntax **ResetEventStatus** *mask*

Arguments	Name	Instance	Encoding	Bit number
	<i>mask</i>	<i>Capture Received</i>	0008h	3
		<i>Motion Error</i>	0010h	4
		<i>Amplifier Error</i>	0100h	8
		<i>UserCommand Error</i>	8000h	15

Packet Structure



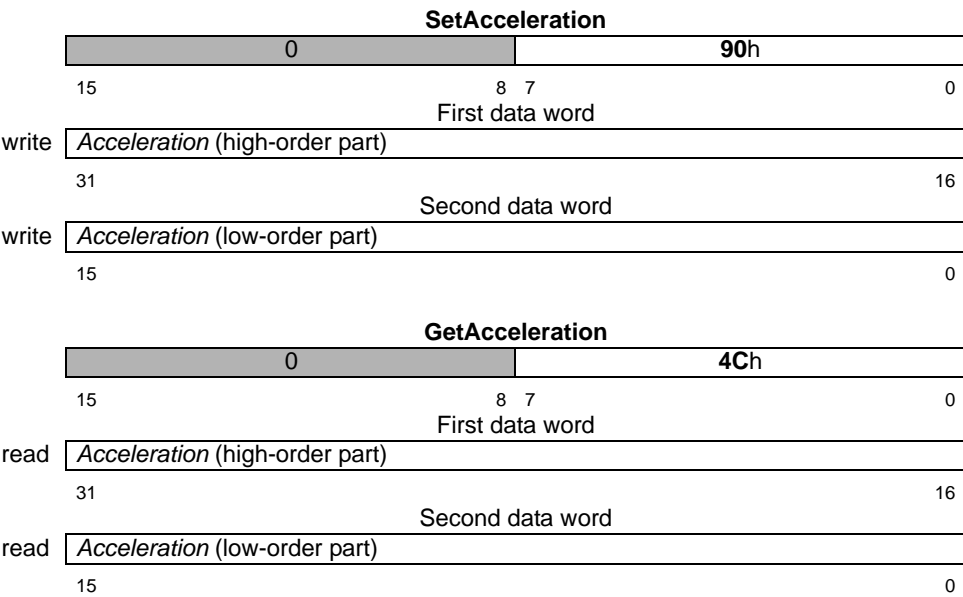
Description **ResetEventStatus** clears (sets to 0) each bit in the Event Status register that has a value of 0 in the mask sent with this command. All other Event Status register bits (bits which have a mask value of 1) are unaffected. For example, setting the mask to 0018h will clear the Amplifier error and UserCommand error bits, but will leave the capture received and motion error bit unchanged.

Restrictions

see **GetEventStatus** ([p. 74](#))

Syntax	SetAcceleration <i>Acceleration</i> GetAcceleration				
Arguments	Name	Type	Range	Scaling	Units
	<i>Acceleration</i>	unsigned 32-bit	0 to 2 ³¹ –1	1/2 ¹⁶	counts/cycle ²

Packet Structure



Description

SetAcceleration loads the maximum acceleration buffer register. This command is used when the command source is set to profile generator.

GetAcceleration reads the maximum acceleration buffer register.

Scaling example: to load a value of 1.750 counts/cycle², multiply by 65,536 (giving 114,688), and load the resultant number as a 32-bit number; giving 0001 in the high word and C000h in the low word. Values returned by **GetAcceleration** must be divided by 65,536 to convert to units of counts/cycle².

Restrictions

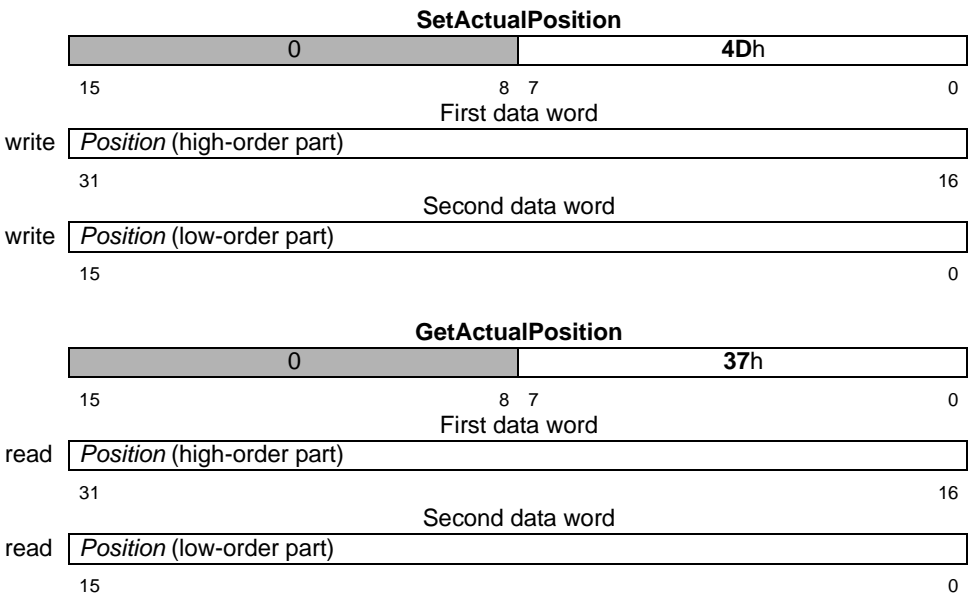
see

SetVelocity/GetVelocity (p. 116), SetCommandSource/GetCommandSource (p. 91)

Syntax **SetActualPosition** *position*
GetActualPosition

Arguments	Name	Type	Range	Scaling	Units
	<i>position</i>	signed 32-bit	-2^{31} to $2^{31}-1$	unity	counts

**Packet
Structure**



Description

SetActualPosition loads the position register (encoder position). This instruction establishes a new reference position from which subsequent positions can be referenced. It is used to set a known reference position after a homing procedure.

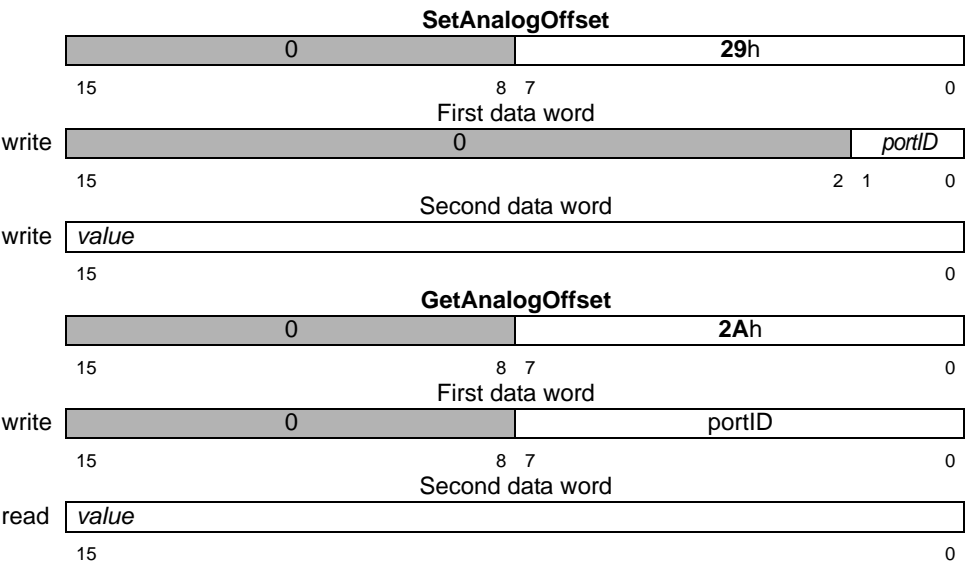
GetActualPosition reads the contents of the encoder's position register.

Restrictions

see **SetVelocity/GetVelocity** (p. 116), **SetCommandSource/GetCommandSource** (p. 91)

Syntax	SetAnalogOffset <i>portID</i> <i>value</i> GetAnalogOffset <i>portID</i>			
Arguments	Name	Type	Instance	Encoding
	<i>portID</i>	unsigned 16-bit	AnalogCmd Tachometer CurrentA CurrentB	0 1 2 3
	<i>value</i>	signed 16-bit	Range −2 ¹⁵ to 2 ¹⁵ −1	Scaling unity

Packet
Structure



Description

SetAnalogOffset sets the offset that is summed with the value at the desired *portID* prior to use in chip calculations.

GetAnalogOffset returns the value of the offset for the specified port.

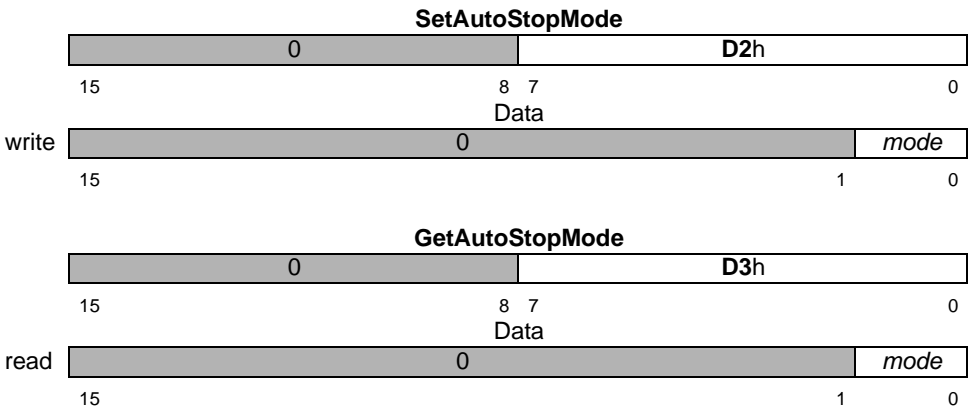
Restrictions

see

Syntax **SetAutoStopMode** *mode*
 GetAutoStopMode

Arguments	Name	Type	Instance	Encoding
	<i>mode</i>	unsigned 16-bit	<i>Disable</i>	0
			<i>Enable</i>	1

**Packet
Structure**



Description **SetAutoStopMode** determines the behavior when a motion error occurs. When auto stop is enabled (**SetAutoStopMode Enable**), the axis goes into open-loop mode when a motion error occurs. When auto stop is disabled (**SetAutoStopMode Disable**), the axis is not affected by a motion error.

GetAutoStopMode returns the current state of the auto stop mode.

Restrictions .

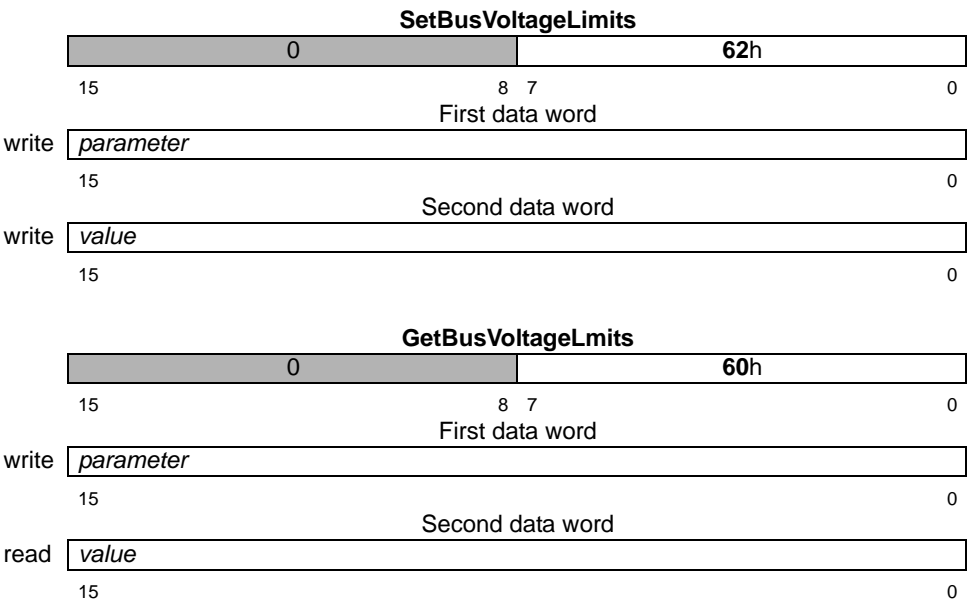
see **GetEventStatus** (p. 74), **SetMotionErrorLimit /GetMotionErrorLimit** (p. 97)

Syntax

SetBusVoltageLimits *parameter value*
GetBusVoltageLimits *parameter*

Arguments	Name	Instance	Encoding
	<i>parameter</i>	<i>OverVoltageLimit</i> <i>UnderVoltageLimit</i>	0 1
	<i>value</i>	Type unsigned 16 bits	Range 0 to 2 ¹⁶ −1

Packet
Structure



Description

SetBusVoltageLimits sets the thresholds for determination of overvoltage and undervoltage conditions. If the Bus voltage exceeds the *OverVoltageLimit* value, an overvoltage condition occurs. If the Bus voltage is less than the *UnderVoltageLimit* value, an undervoltage condition occurs. Both the *OverVoltageLimit* and *UnderVoltageLimit* have ranges of 0 to 2¹⁶−1, with scaling dependent on Bus voltage sensor scaling.

GetBusVoltageLimits reads the indicated limit.

Restrictions

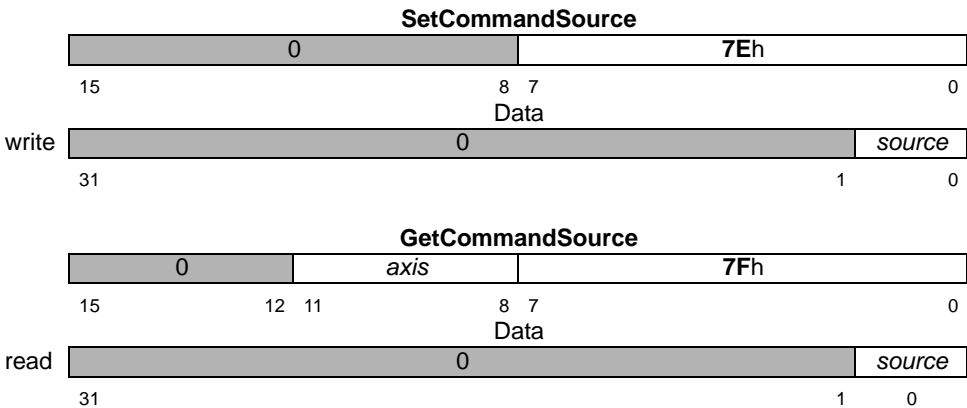
see

GetBusVoltage (p. 72), GetActivityStatus (p. 69)

Syntax **SetCommandSource** *source*
GetCommandSource

Arguments	Name	Type	Instance	Encoding (hex)
	<i>source</i>	unsigned 16-bit	<i>AnalogCmd</i>	0
			<i>SPI</i>	1
			<i>ProfileGenerator</i>	2

**Packet
Structure**



Description **SetCommandSource** determines the source of input for the chip control loops. When set to *AnalogCmd*, the command for the velocity or current loops originates from the analog input signal. When set to *SPI*, the command for the velocity or current loops is a 16-bit value read from the incoming SPI data stream. When set to *ProfileGenerator*, the command is internally generated, based on trajectory parameters set by the host.

GetCommandSource returns the command source.

Restrictions

see **SetLoopMode /GetLoopMode** (p. 96)

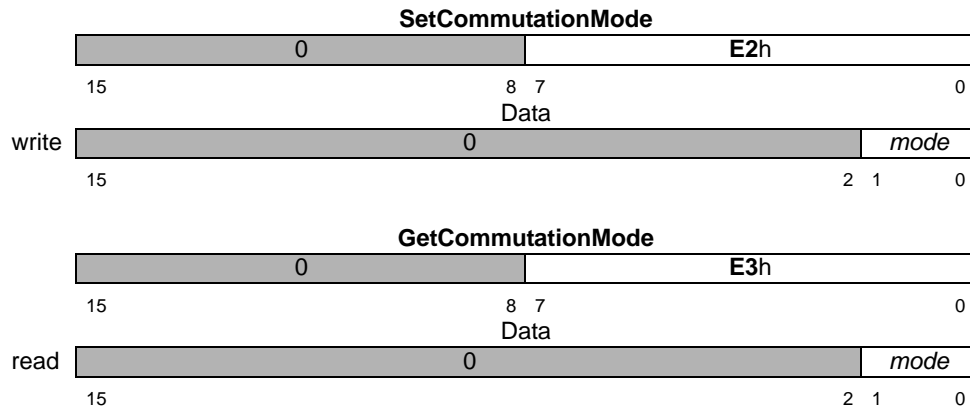
Syntax

SetCommutationMode *mode*
GetCommutationMode

Arguments

Name	Type	Instance	Encoding
<i>mode</i>	unsigned 16-bit	<i>Sinusoidal Commutation</i>	0
		<i>Hall-Based Commutation</i>	1
		<i>Sinusoidal FOC</i>	2
		<i>Hall-Based FOC</i>	3

Packet Structure



Description

SetCommutationMode sets the phase commutation mode.

When set to sinusoidal, as the motor turns, the encoder input signals are used to calculate the phase angle. This angle is in turn used to generate sinusoidally varying outputs to each motor winding.

When set to Hall based, the Hall effect sensor inputs are used to commutate the motor windings using a trapezoidal (six-step) waveform method.

In the *Sinusoidal Commutation* and *Hall-Based Commutation* modes, commutation is performed to create the commands for the motor phases. If current loops are enabled in either of these modes, they will be run independently for each motor phase (phase A/B current loops).

In the *Sinusoidal FOC* and *Hall-Based FOC* modes, Field Oriented Control (FOC) is used, in conjunction with Space Vector PWM, to determine the motor phase commands. If current loops are enabled in either of these modes, they will be run as part of the FOC algorithm (FOC D/Q loops).

GetCommutationMode returns the value of the commutation mode.

Restrictions

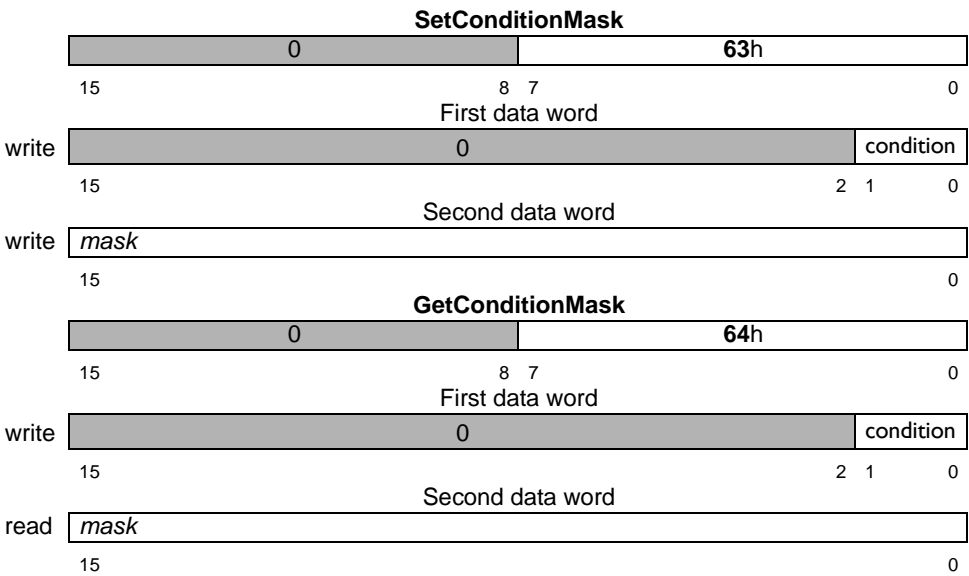
see

SetPhaseCounts/GetPhaseCounts (p. 103)

Syntax **SetConditionMask** *condition mask*
 GetConditionMask *condition*

Arguments	Name	Type	Instance	Encoding
	<i>condition</i>	unsigned 16-bit	<i>AmplifierError</i> <i>AmplifierDisable</i> <i>PWMDisable</i>	0 1 2
	<i>mask</i>	unsigned 16-bit	Range 0 to 2 ¹⁶ –1	

Packet Structure



Description

SetConditionMask sets the mask that is compared to a combined status register to activate the specified state. A 1 in the bit mask means that that if an active condition occurs in the corresponding source register bit, the condition is satisfied. If more than one bit-field is programmed with a 1, a logical OR of the conditions is applied.

For example, to define an amplifier error condition as the occurrence of an overtemperature or an Estop going active, the condition mask would be set to 2040 (hex).

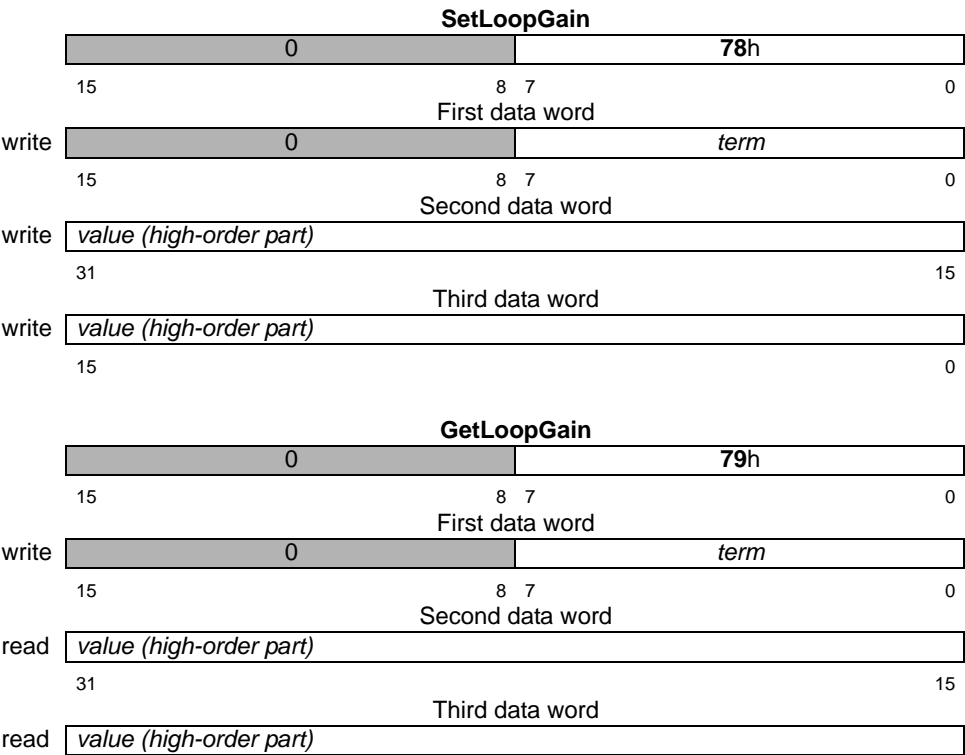
GetConditionMask returns the mask for the specified condition.

Restrictions

see Section 4.14, “Programmable Conditions,” on page 47.

Syntax	SetLoopGain <i>term value</i> GetLoopGain <i>term</i>			
Arguments	Name	Type	Instance	Encoding
	<i>term</i>	unsigned 16-bit	<i>CurrentKp</i> <i>CurrentKi</i> <i>CurrentLimit</i> <i>VelocityKp</i> <i>VelocityKi</i> <i>VelocityLimit</i> <i>VelocityIntegratorKp</i> <i>VelocityIntegratorKi</i> <i>VelocityIntegratorLimit</i> <i>VelocityIntegratorKd</i>	00h 10h 20h 01h 11h 21h 02h 12h 22h 32h
	<i>value</i>	unsigned 32-bit	Range	Scaling
			0 to 2 ¹⁵ –1 0 to 2 ³¹ –1	unity

Packet
Structure



Description

SetLoopGain assigns the given value to the specified loop gain.

GetLoopGain returns the value of the specified loop gain.

CurrentKp, *CurrentKi*, and *CurrentLimit* apply to the current loops, regardless of whether the loops are being run in A/B mode or FOC (D/Q loops).

Restrictions

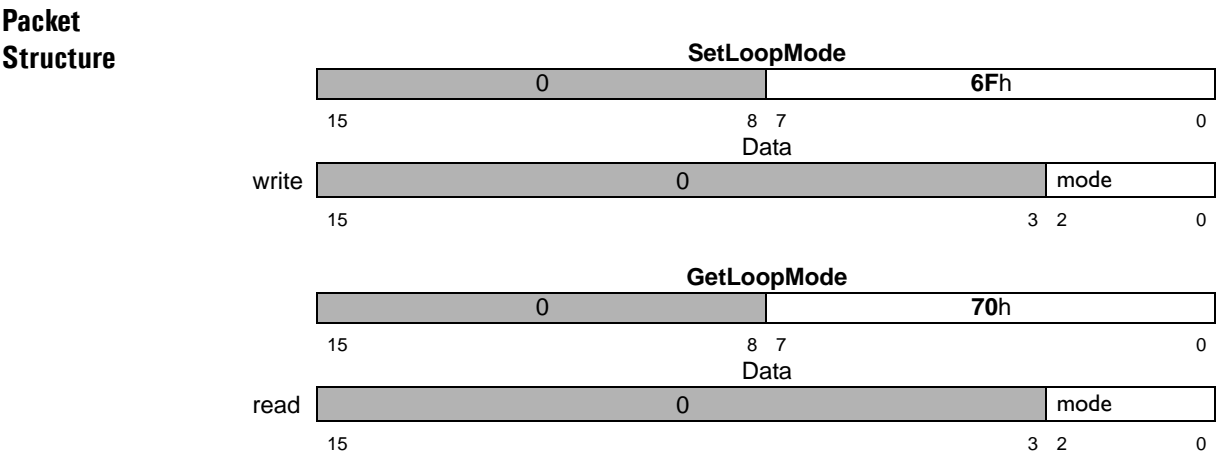
When the term is set to *VelocityIntegratorLimit*, the value set and returned is a full 32-bit value.

For all other terms, the value set and returned is a 16-bit value in the low-order part of value. When the term is set to *CurrentLimit* or *VelocityLimit* the value set and returned is scaled by 1/256.

see

GetLoopIntegral ([p. 78](#))

Syntax	SetLoopMode <i>mode</i> GetLoopMode				
Arguments	Name <i>mode</i>	Type unsigned 16-bit	Instance <i>Current Loop</i> <i>Velocity Loop</i> <i>Velocity Integrator</i>	Encoding 0001h 0002h 0004h	Bit number 0 1 2



Description

SetLoopMode is used to enable and disable the chip current, velocity and velocity integrator loops. Setting the corresponding bit to 1 enables the loop. Setting the bit to 0 disables the loop.

GetLoopMode returns the loop mode.

The current loops can be enabled/disabled regardless of whether in A/B or FOC current control.

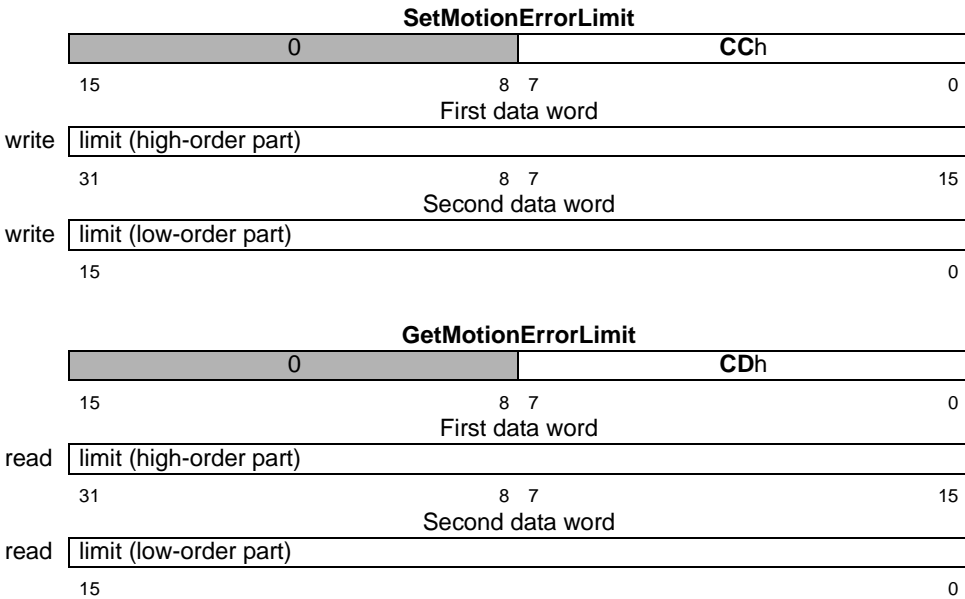
Restrictions

see

Syntax **SetMotionErrorLimit** *limit*
GetMotionErrorLimit

Arguments	Name	Type	Range	Scaling	Units
	<i>limit</i>	unsigned 32-bit	0 to $2^{31}-1$	unity	counts counts/cycle

**Packet
Structure**



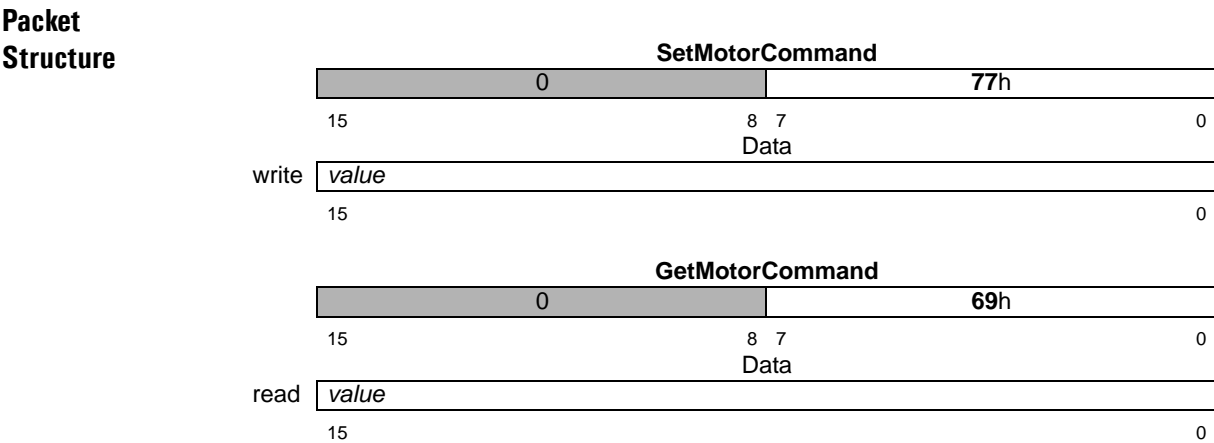
Description **SetMotionErrorLimit** sets the value of the maximum motion error allowable by the chip. If the velocity integrator loop is enabled, it will compare this value with its error term (**GetLoopError**). If this value is exceeded, then a motion error occurs. Such a motion error may or may not cause the axis to stop moving, depending on the value set using the **SetAutoStopMode** command. If the highest enabled loop is the velocity integrator, this value represents a position error in counts in 32.0 format. If the highest enabled loop is the velocity loop, this value represents a velocity error in counts/cycle in 16.16 format. If the highest enabled loop is the current loop, this value is not used.

GetMotionErrorLimit returns the motion error limit value.

Restrictions

see **SetActualPosition/GetActualPosition** (p. 87), **SetAutoStopMode/GetAutoStopMode** (p. 89)

Syntax	SetMotorCommand <i>value</i> GetMotorCommand				
Arguments	Name <i>value</i>	Type signed 16-bit	Range -2^{15} to $2^{15}-1$	Scaling $100/2^{15}$	Units % output



Description

SetMotorCommand loads the Motor Command register.

GetMotorCommand returns the motor output command. In open-loop mode, it returns the contents of the motor output command register. In closed-loop mode the value returned is meaningless.

Scaling example: If it is desired that a motor command value of 13.7 % of full scale be output to the motor, then this register should be loaded with a value of $13.7 * 32,768 / 100 = 4,489$ (decimal). This corresponds to a hexadecimal value of 1189h.

Restrictions

SetMotorCommand and GetMotorCommand are valid only when the motor mode is set to off.

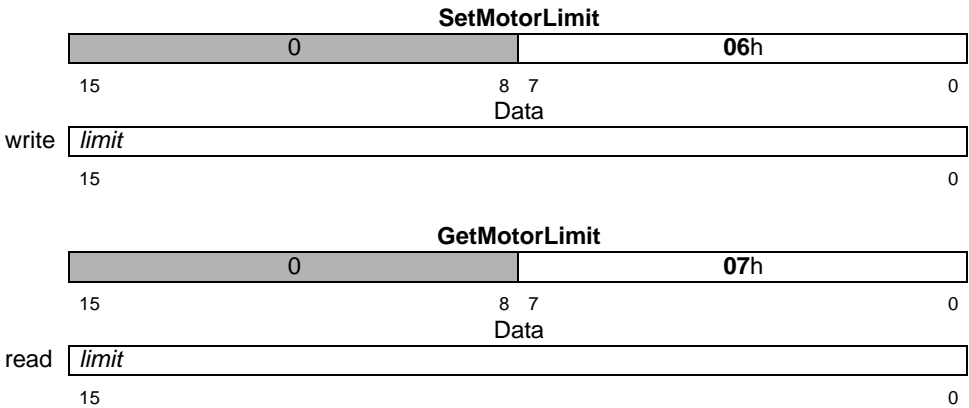
see

SetPWMLimit/GetPWMLimit (p. 108), SetMotorMode/GetMotorMode (p. 100)

Syntax **SetMotorLimit** *limit*
GetMotorLimit

Arguments	Name	Type	Range	Scaling	Units
	<i>limit</i>	unsigned 16-bit	0 to $2^{15}-1$	$100/2^{15}$	% output

Packet Structure



Description **SetMotorLimit** sets the maximum value for the motor output command allowed by the digital servo filter. Motor command values beyond this value will be clipped to the specified motor command limit. For example, if the motor limit was set to 1,000, and the servo filter determined that the current motor output value should be 1,100, then the actual output value would become 1,000. If the output value were -1,100, then it would be clipped to -1,000. This command is useful for protecting amplifiers, motors, or system mechanisms in situations wherein a motor command exceeding a specific value will cause damage.

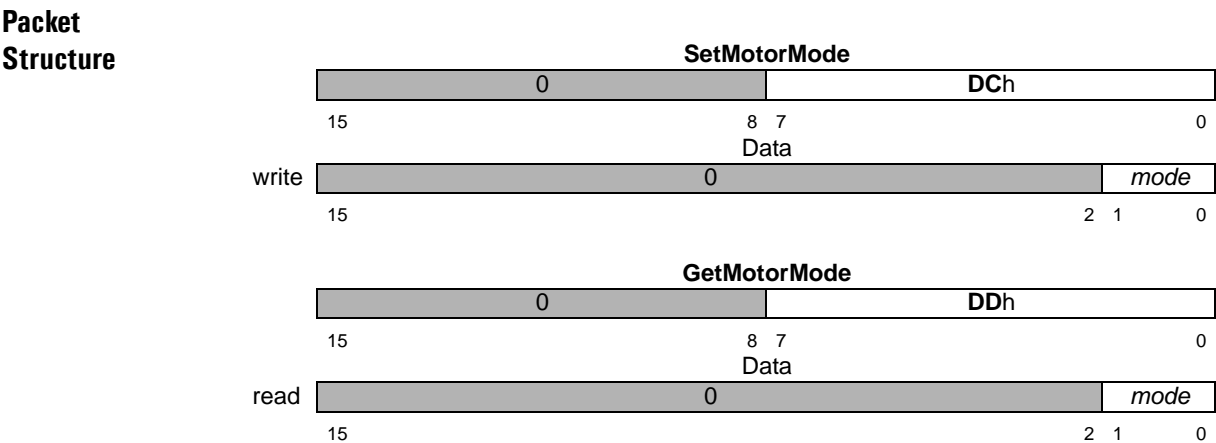
GetMotorLimit reads the motor limit value.

Scaling example: if it is desired that a motor limit of 75% of full scale be established, then this register should be loaded with a value of $75.0 * 32,768 / 100 = 24,576$ (decimal). This corresponds to a hexadecimal value of 06000h.

Restrictions This command only affects the motor output when the axis motor mode is on (**SetMotorMode**). When the motion IC is in open loop mode, this command has no affect.

see **SetMotorCommand/GetMotorCommand** (p. 98)

Syntax	SetMotorMode <i>mode</i> GetMotorMode			
Arguments	Name <i>mode</i>	Type unsigned 16-bit	Instance <i>Off</i> <i>On</i>	Encoding 0 1



Description

SetMotorMode determines the mode of motor operation. When set to *On*, the axis is placed in closed-loop mode, and is controlled by the output of the current, velocity integrator filter, or velocity filter. When the motor mode is set to *Off*, the axis is in open-loop mode, and is controlled by commands placed directly into the motor output register by the host.

GetMotorMode returns the motor mode.

Restrictions

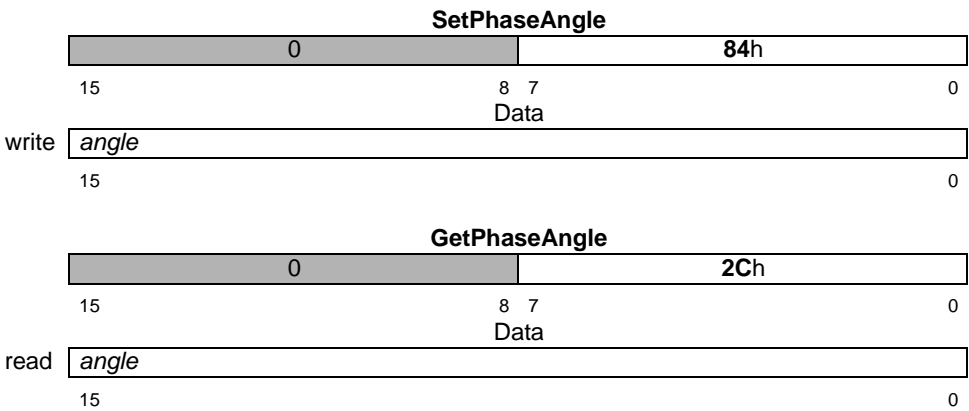
see

GetActivityStatus (p. 69), SetMotorCommand/GetMotorCommand (p. 98)

Syntax **SetPhaseAngle** *angle*
GetPhaseAngle

Arguments	Name	Type	Range	Scaling	Units
	<i>angle</i>	unsigned 16-bit	0 to $2^{15}-1$	unity	counts

**Packet
Structure**



Description **SetPhaseAngle** sets the instantaneous commutation angle.

GetPhaseAngle returns the value of the phase angle. To convert counts to an actual phase angle, divide by the number of encoder counts per electrical cycle and multiply by 360. For example, if a value of 500 is retrieved using **GetPhaseAngle**, and the counts per electrical cycle value has been set to 2,000 (**SetPhaseCounts** command), this corresponds to an angle of $(500/2,000)*360 = 90$ degrees phase angle position.

Restrictions The specified angle must not exceed the number of counts per electrical cycle set by the **SetPhaseCounts** command. **SetPhaseAngle** is only valid when using sinusoidal commutation.

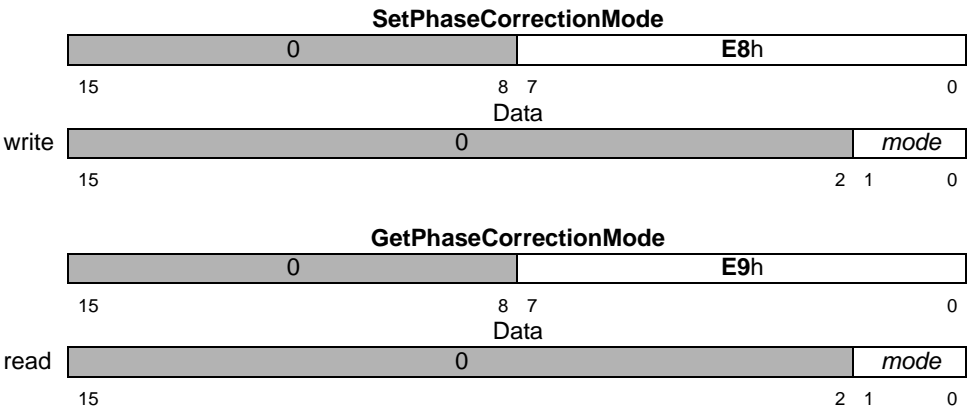
see **SetPhaseCounts/GetPhaseCounts** ([p. 103](#))

Syntax

SetPhaseCorrectionMode *mode*
GetPhaseCorrectionMode

Arguments	Name	Type	Instance	Encoding
	<i>mode</i>	unsigned 16-bit	— (Reserved)	0
			<i>Index</i>	1
			<i>Hall</i>	2

Packet
Structure



Description

SetPhaseCorrectionMode sets the phase correction mode for to either *Index* or *Hall*. When phase correction is set to *Index*, the encoder index signal is used to synchronize the commutation phase angle for each motor revolution. This ensures that the commutation angle will remain correct even if some encoder counts are lost due to electrical noise, or due to the number of encoder counts/electrical phase not being an integer. When phase correction is set to *Hall*, the Hall sensor signals are used to synchronize the commutation phase angle once per electrical cycle.

GetPhaseCorrectionMode returns the phase correction mode.

Restrictions

see

SetPhaseCounts/GetPhaseCounts (p. 103)

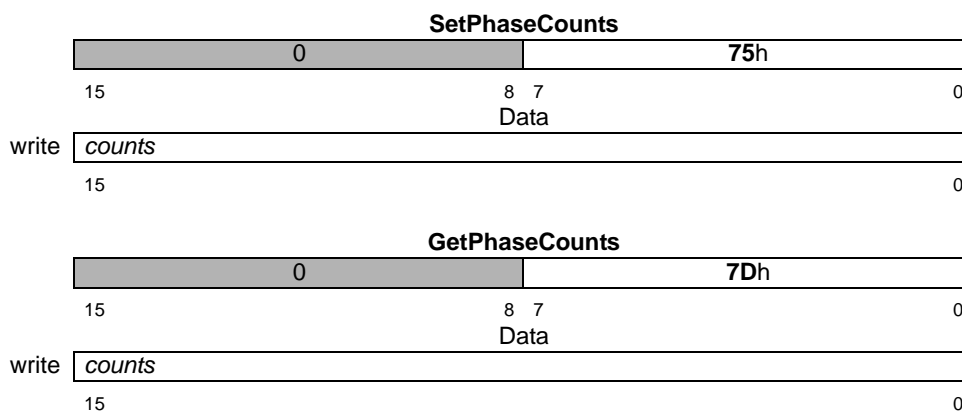
Syntax

SetPhaseCounts *counts*
GetPhaseCounts

Arguments

Name	Type	Range	Scaling	Units
<i>counts</i>	unsigned 16-bit	1 to $2^{15}-1$	unity	counts

Packet Structure



Description

SetPhaseCounts sets the number of encoder counts per electrical cycle of the motor. If this value is not an integer, then the closest integer value should be used. Automatic phase correction (**Set/GetPhaseCorrectionMode**) should be used to correct the phase angle. The number of electrical cycles is equal to 1/2 the number of motor poles. If the number of encoder counts per electrical cycle exceeds 2^{15} , see **SetPhasePrescale**.

GetPhaseCounts returns the number of counts per electrical cycle.

Restrictions

If an encoder is being used for commutation then **SetCommutationMode** must be called after calling **SetPhaseCounts** and before moving the motor, in order to ensure that the motor phase is correctly set.

see

SetCommutationMode (p. 92)

Syntax	SetPhasePrescale <i>scale</i> GetPhasePrescale			
Arguments	Name <i>scale</i>	Type unsigned 16-bit	Instance <i>Off</i> <i>On</i>	Encoding 0 1
Packet Structure	<div> <div>SetPhasePrescale</div> <div> <div>0</div> <div>E6h</div> </div> <div> 15870 </div> <div>Data</div> </div> <div> <div>write</div> <div> <div>0</div> <div><i>scale</i></div> </div> <div> 15210 </div> </div> <div> <div>GetPhasePrescale</div> <div> <div>0</div> <div>E7h</div> </div> <div> 15870 </div> <div>Data</div> </div> <div> <div>read</div> <div> <div>0</div> <div><i>scale</i></div> </div> <div> 15210 </div> </div>			
Description	<p>SetPhasePrescale <i>On</i> causes the number of encoder counts to be scaled by a factor of 1/64 before being used to calculate a commutation angle. When operated in the prescale mode, the chip can commute motors with a high number of counts per electrical cycle, such as motors with very high accuracy encoders.</p> <p>SetPhasePrescale <i>Off</i> removes the scale factor.</p> <p>GetPhasePrescale returns the scaling mode.</p>			
Restrictions	This command should only be used if the encoder count per electrical cycle of the motor exceeds 32767.			
see				

Syntax

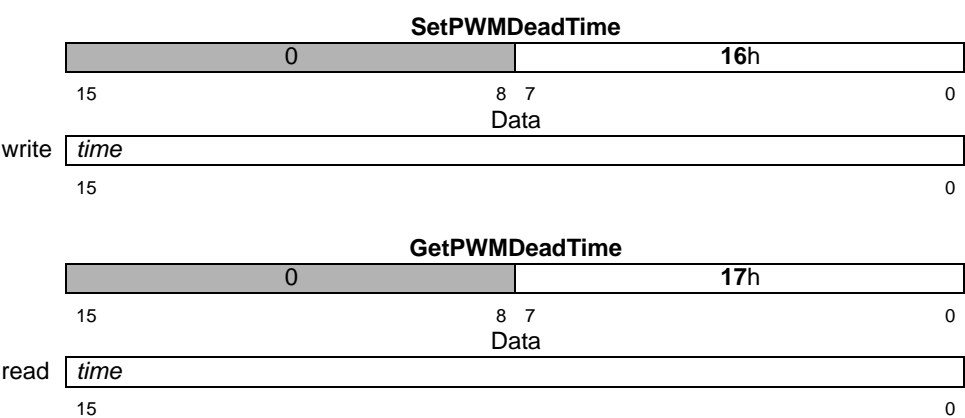
SetPWMDeadTime *time*

GetPWMDeadTime

Arguments

Name	Instance (in μ s)	Encoding	Instance (in μ s)	Encoding
time	0	0	1.2	28
	0.025	1	1.3	29
	0.05	2	1.4	30
	0.075	3	1.5	31
	0.1	4	1.6	32
	0.125	5	1.8	33
	0.15	6	2	34
	0.175	7	2.2	35
	0.2	8	2.4	36
	0.225	9	2.6	37
	0.25	10	2.8	38
	0.275	11	3	39
	0.3	12	3.2	40
	0.325	13	3.6	41
	0.35	14	4	42
	0.375	15	4.4	43
	0.4	16	4.8	44
	0.45	17	5.2	45
	0.5	18	5.6	46
	0.55	19	6	47
	0.6	20	6.4	48
	0.65	21	7.2	49
	0.7	22	8	50
	0.75	23	8.8	51
	0.8	24	9.6	52
	0.9	25	10.4	53
	1	26	11.2	54
	1.1	27	12	55

Packet Structure



Description

SetPWMDeadTime sets the time between successive high/low or low/high turn-on sequences for a given phase. This is often an important requirement to avoid excessive current flow between the upper and lower switching elements of the amplifier. To determine the correct minimum delay time, consult the specifications for the switching IC or circuit. The programmed dead time delay affects all phases.

GetPWMDeadTime returns the code for the dead time.

Restrictions

Dead time generation is only active when the chip is operating in six-signal PWM mode.

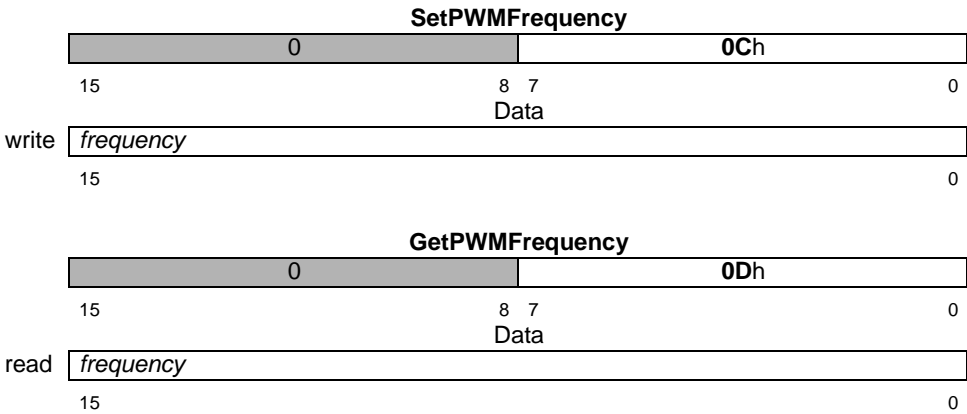
see

SetPWMOutputMode/GetPWMOutputMode (p. 109),
SetPWMSense/GetPWMSense (p. 110)

Syntax **SetPWMFrequency** *frequency*
GetPWMFrequency

Arguments	Name	Type	Range	Scaling	Units
	<i>frequency</i>	unsigned 16 bits	0 to 2 ¹⁶ −1	1/2 ⁸	kHz

**Packet
Structure**



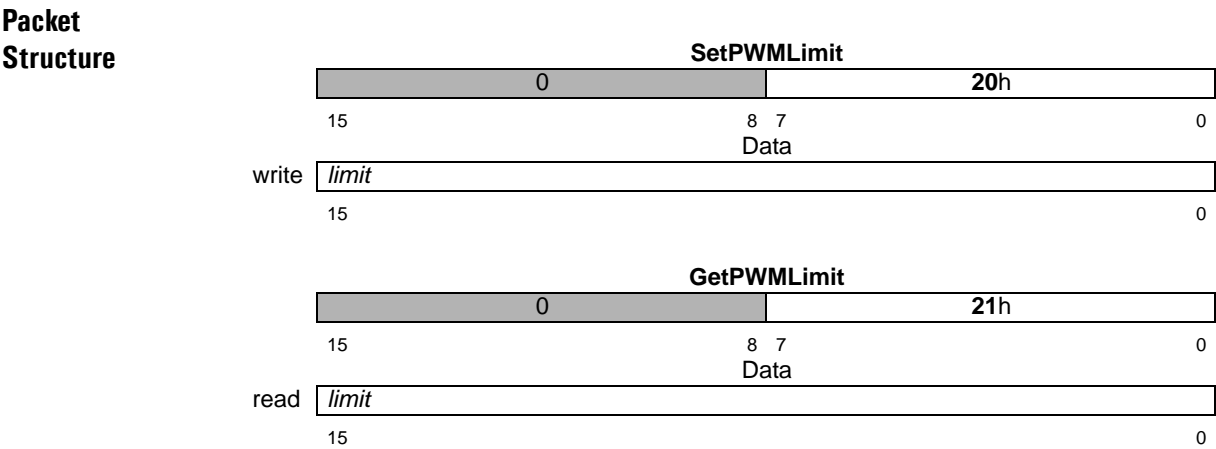
Description **SetPWMFrequency** sets the PWM output frequency (in kHz). Only two frequencies are supported by the MC73110; these are shown in the table below. To select one of the supported frequencies, pass the value listed in the **SetPWMFrequency** value column as the *frequency* argument to this command.

Approximate Frequency	PWM bit Resolution	Actual Frequency	SetPWMFrequency Value
20 kHz	10	19.531 kHz	5,000
40 kHz	9	39.062 kHz	10,000

Restrictions

see **SetPWMOutputMode/GetPWMOutputMode** (p. 109)

Syntax	SetPWMLimit <i>limit</i> GetPWMLimit				
Arguments	Name	Type	Range	Scaling	Units
	<i>limit</i>	unsigned 16-bit	0 to 2 ¹⁵ –1	100/2 ¹⁵	% output



Description

SetPWMLimit sets the output PWM duty cycle limit.

GetPWMLimit reads the current value of the output PWM duty cycle limit. Scaling example: if a limit of 97% of full scale is required, then this register should be loaded with a value of 97.0 *32,768/100 = 31,784 (decimal). This corresponds to a hexadecimal value of 7C28h.

Restrictions

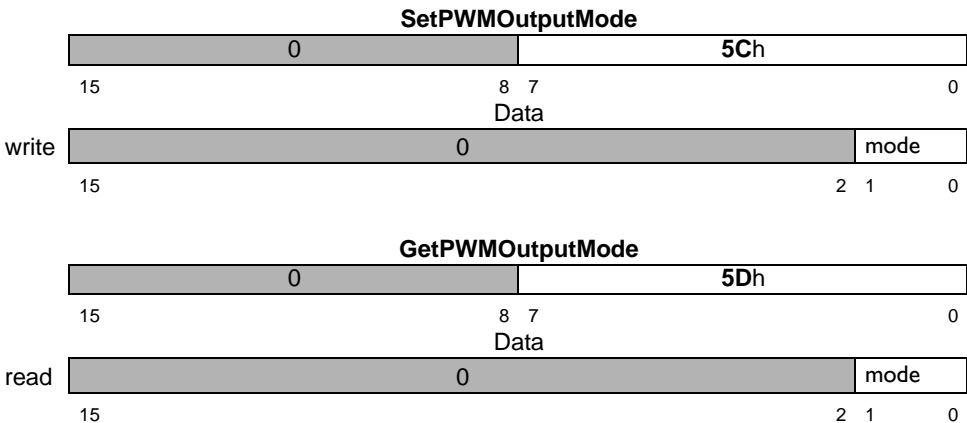
see

SetMotorCommand/GetMotorCommand (p. 98), SetMotorMode/GetMotorMode (p. 100), SetPWMOutputMode/GetPWMOutputMode (p. 109)

Syntax **SetPWMOutputMode** *mode*
 GetPWMOutputMode

Arguments	Name	Type	Instance	Encoding
	<i>mode</i>	unsigned 16-bit	6-signal with dead time 3-signal 6-signal with dead time, 3rd Leg floating	0 1 2

**Packet
Structure**



Description **SetPWMOutputMode** determines the form of the motor output signal.

GetPWMOutputMode returns the code for the motor output mode.

Restrictions The 6-signal mode which floats the 3rd leg’s drive is only available if using Hall-Based commutation (not FOC).

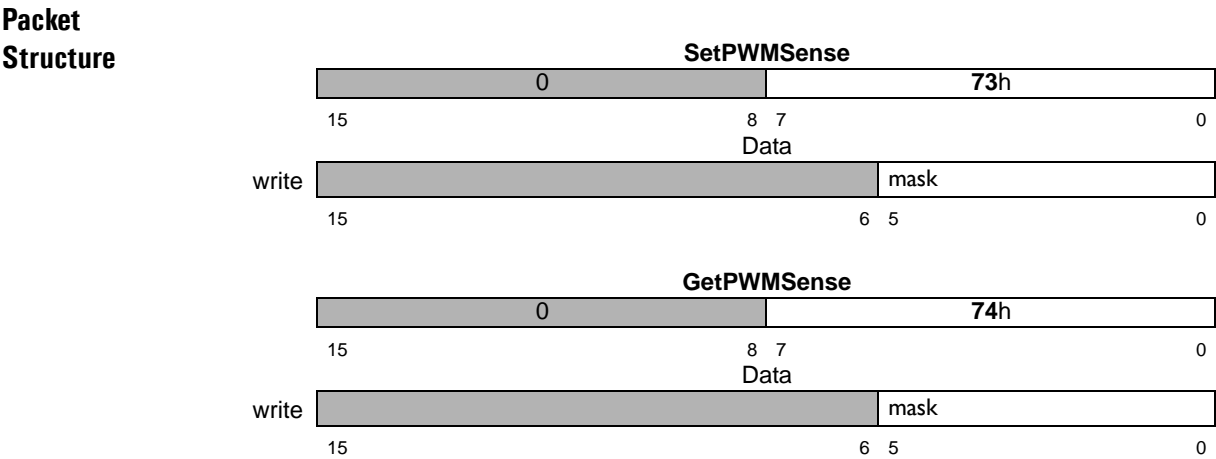
see **SetPWMSense/GetPWMSense** (p. 110)

Syntax

SetPWMSense *mask*
GetPWMSense

Arguments

Name	Instance	Encoding	Bit number
<i>mask</i>	<i>PWMAHigh/PWMA</i>	0001h	0
	<i>PWMALow</i>	0002h	1
	<i>PWMBHigh/PWMB</i>	0004h	2
	<i>PWMBLow</i>	0008h	3
	<i>PWMCHigh/PWMC</i>	0010h	4
	<i>PWMCLow</i>	0020h	5



Description

SetPWMSense establishes the sense of the PWM output signals from the chip by using a bitwise mask.

For each sense bit that is 0, the output is active low.

For each sense bit that is 1, the output is active high.

When the chip is operating in six-signal output mode (see **SetPWMOutputMode**), all six bits of the mask are valid. When the chip is operating in three-signal output mode, only bits 0, 2, and 4 of the mask are used.

GetPWMSense returns the PWM sense mask.

Restrictions

Warning: Incorrect settings in this register may damage the output circuitry or motor.

see

SetPWMOutputMode/GetPWMOutputMode (p. 109)

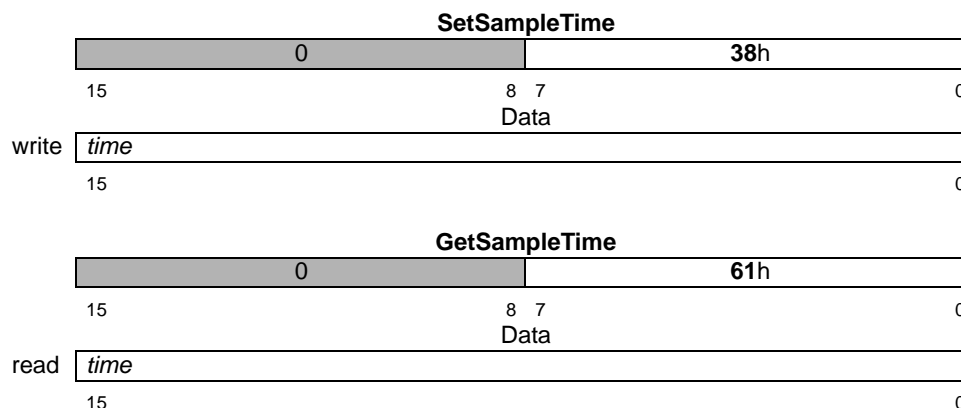
Syntax

SetSampleTime *time*
GetSampleTime

Arguments

Name	Type	Range	Units
<i>time</i>	unsigned 16-bit	102 to 2000	µsec/cycle

Packet Structure



Description

SetSampleTime sets the time basis for the motion processor. This time basis determines the trajectory update and the velocity servo loop calculation rate. It does not, however, determine the commutation rate or the current loop rate. The time value is expressed in microseconds (μsec). The minimum value allowed is 102 μsec . The motion processor hardware can adjust the cycle time only in increments of 51.2 μsec (rounded up). For example, 154, 205, 256, etc. The time value passed to this command will be rounded up to the nearest increment of this base value.

GetSampleTime returns the current sample time value.

Restrictions

see

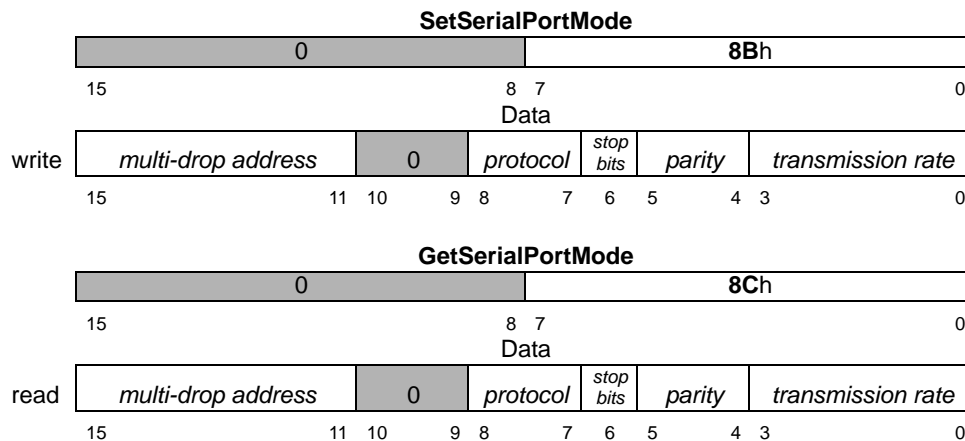
Syntax

SetSerialPortMode *mode*
GetSerialPortMode

Arguments

Name	Type	Encoding
<i>mode</i>	unsigned 16-bit	see below

Packet Structure



Description

SetSerialPortMode sets the configuration for the asynchronous serial port.

GetSerialPortMode returns the configuration for the asynchronous serial port.

The following table shows the encoding of the data used by this command.

Bit Number	Name	Instance	Encoding
0–3	Transmission Rate	1200 baud	0
		2400 baud	1
		9600 baud	2
		19200 baud	3
		57600 baud	4
		115200 baud	5
		230400 baud	6
		460800 baud	7
4–5	Parity	None	0
		Odd	1
		Even	2
6	Stop Bits	1	0
		2	1
7–8	Protocol	Point-to-point	0
		— (Reserved)	1–2
		Multi-drop using idle-line detection	3
11–15	Multi-drop Address	Address 0	0
		Address 1	1
	
		Address 31	31

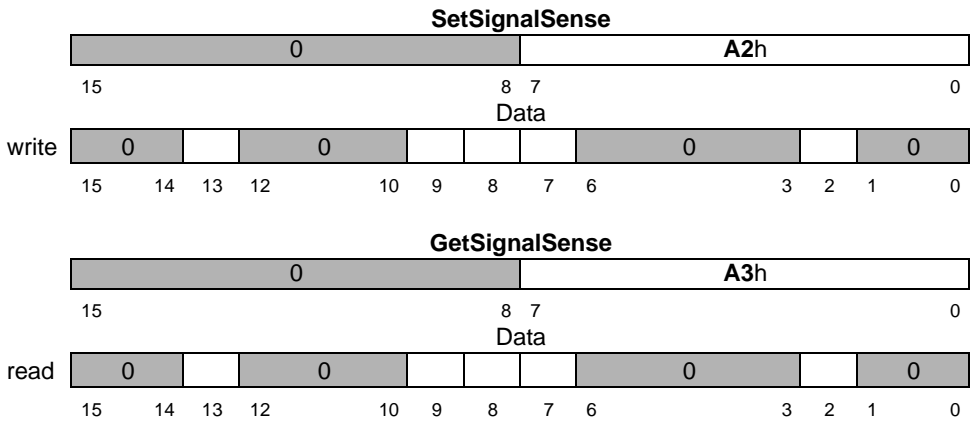
Restrictions

see

Syntax **SetSignalSense** *mask*
GetSignalSense

Arguments	Name	Instance	Encoding	Bit Number
	<i>mask</i>	— (Reserved)	—	0-1
		<i>Index</i>	0004h	2
		— (Reserved)	—	3-6
		<i>HallA</i>	0080h	7
		<i>HallB</i>	0100h	8
		<i>HallC</i>	0200h	9
		— (Reserved)	—	10-12
		<i>Estop</i>	2000h	13
		— (Reserved)	—	14-15

Packet Structure



Description

SetSignalSense establishes the sense of the corresponding bits of the Signal Status register. For all input signals, the input is inverted if the corresponding sense bit is one; otherwise it is not inverted. For encoder index/home: if the sense bit is 1, a capture will occur on a low-to-high signal transition. Otherwise, a capture will occur on a high-to-low transition.

GetSignalSense returns the signal sense mask.

Restrictions

Calling **SetSignalSense** in a tight loop may interfere with motor commutation if a real change is being made. Calling **SetSignalSense** in a tight loop with the same argument is safe.

If an encoder is used for commutation and the sense of the Hall sensors or the index sensor is changed, then **SetCommutationMode** must be called before moving the motor to ensure that the motor phase is set correctly.

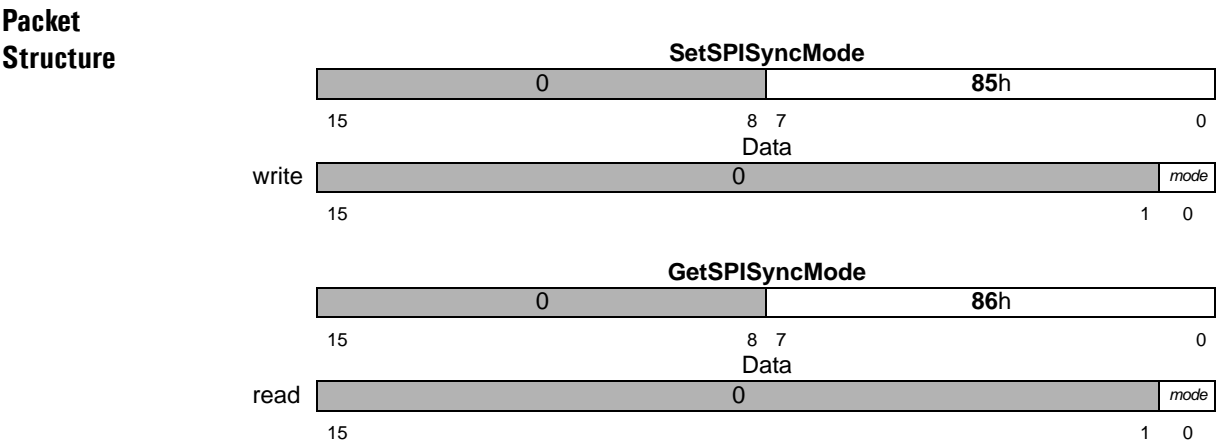
see **GetSignalStatus** (p. 80), **SetCommutationMode** (p. 92)

Syntax

SetSPISyncMode *mode*
GetSPISyncMode

Arguments

Name	Instance	Encoding
<i>mode</i>	<i>Off</i>	0
	<i>On</i>	1



Description

SetSPISyncMode enables or disables the special SPI mode which enables the recovery of SPI synchronization under some conditions. GetSPISyncMode returns whether this mode is enabled or not.

Restrictions

The SPISyncMode should only be set to *On* if certain timing requirements on the SPI packets are met.

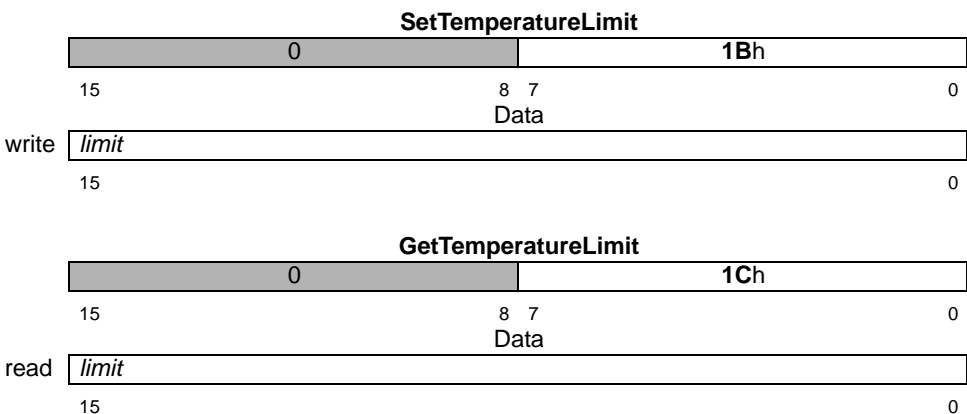
see

Section 4.20, “Synchronous Serial Input (SPI Port),” on page 59.

Syntax **SetTemperatureLimit** *limit*
GetTemperatureLimit

Arguments	Name	Type	Range
	<i>limit</i>	signed 16-bit	-2^{15} to $2^{15}-1$

Packet Structure



Description **SetTemperatureLimit** sets the limit value used to determine if an overtemperature condition has occurred. If an overtemperature condition occurs, bit 6 of the Activity Status register will be set to 1.

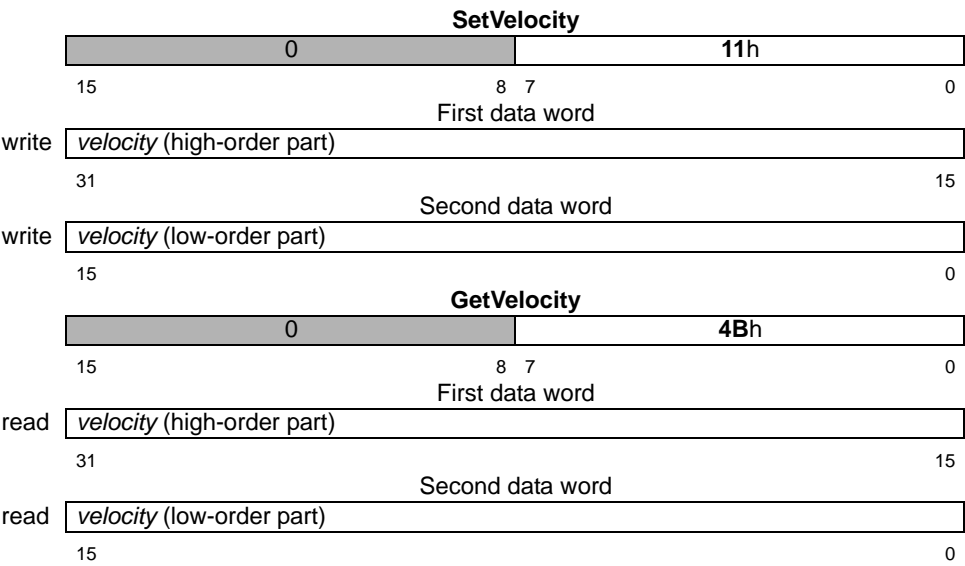
GetTemperatureLimit returns the overtemperature limit value.

Restrictions

see **GetActivityStatus** (p. 69), **GetTemperature** (p. 81)

Syntax	SetVelocity <i>velocity</i> GetVelocity				
Arguments	Name	Type	Range	Scaling	Units
	<i>velocity</i>	signed 32-bit	-2^{31} to $2^{31}-1$	$1/2^{16}$	counts/cycle

Packet
Structure



Description

SetVelocity loads the maximum velocity buffer register. This command is used when the command source is set to profile generator.

GetVelocity returns the maximum velocity buffer register.

Scaling example:

To load a velocity value of 1.750 counts/cycle, multiply by 65,536 (giving 114,688) and load the resultant number as a 32-bit number; giving 0001 in the high word and C000h in the low word. Retrieved numbers (**GetVelocity**) must be divided by 65,536 to convert to units of counts/cycle.

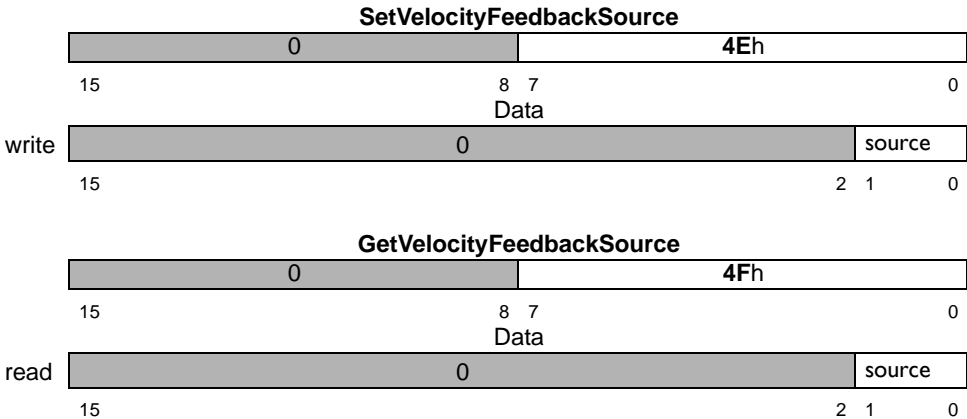
Restrictions

see **SetAcceleration/GetAcceleration** (p. 86), **SetCommandSource/GetCommandSource** (p. 91)

Syntax **SetVelocityFeedbackSource** *source*
GetVelocityFeedbackSource

Arguments	Name	Type	Instance	Encoding (hex)
	<i>source</i>	unsigned 16-bits	<i>Encoder</i>	0
			<i>Tachometer</i>	1
			<i>Hall Sensors</i>	2

Packet Structure



Description **SetVelocityFeedbackSource** sets the source of input for the actual velocity used by the velocity loop. When set to *Tachometer*, the source of velocity information is provided on the tachometer input signal. When set to *Encoder*, the source of velocity information is derived from position information provided by the quadrature A/B signals. When set to *Hall Sensors*, the velocity is derived from position information provided by the Hall sensors.

GetVelocityFeedbackSource returns the feedback source.

Restrictions **SetVelocityFeedbackSource** is not valid with Velocity Integrator enabled.

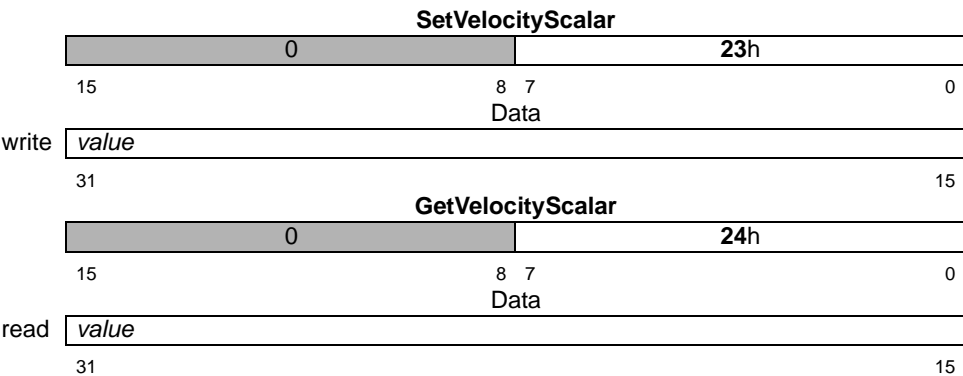
see

Syntax

SetVelocityScalar *value*
GetVelocityScalar

Arguments	Name	Type	Range
	<i>value</i>	unsigned 16-bit	3 to 2 ¹⁵ –1

Packet
Structure



Description

SetVelocityScalar sets the multiplier used to convert units of encoder or Hall sensor counts per cycle into a 16-bit fixed value which is used in the velocity filter equation to represent actual velocity.

The correct value to be used is determined by first calculating the maximum desired motor speed range in chip units. For example, if the maximum desired speed is 3,000 RPMs, and the encoder has 4000 counts per revolution, then:

$$\begin{aligned}
 \text{speed} &= 4000 \times 3000 \\
 &= 12,000,000 \text{ counts per minute} \\
 &= 12,000,000 / 60 = 200,000 \text{ counts per second} \\
 &\text{assuming a sample time of } 102.4 \text{ }\mu\text{sec (SetSampleTime 102),} \\
 \text{speed} &= 200,000 / (10^6 / 102.4) = 20.48 \text{ counts per cycle} \\
 \text{scalar} &= 32767 / 20.48 = 1600
 \end{aligned}$$

GetVelocityScalar returns the velocity scalar.

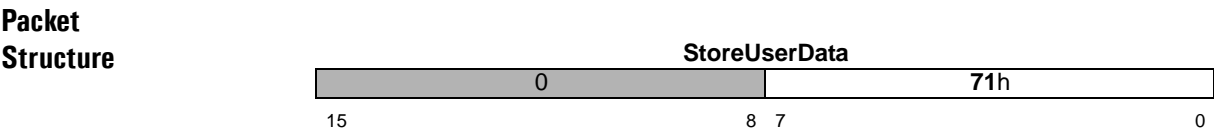
Restrictions

see

Section 4.9.2, “Velocity Scalar,” on page 37.

Syntax **StoreUserData**

Arguments Variable



Description **StoreUserData** initiates a special mode to store commands into on-chip Flash memory.

Restrictions This command can only be sent when the *AmplifierDisable* output is active.

see Section 4.19.2, “Storing User Commands to FLASH,” on page 58.

This page intentionally left blank.

Appendix: List of Commands

Get/Set instructions pairs are shown together on the same line of the table.



ClearPositionError	68	Set/GetConditionMask	93
GetActivityStatus	69	Set/GetLoopGain	94
GetAnalog	70	Set/GetLoopMode	96
GetActualVelocity	71	Set/GetMotionErrorLimit	97
GetBusVoltage	72	Set/GetMotorCommand	98
GetCaptureValue	73	Set/GetMotorLimit	99
GetEventStatus	74	Set/GetMotorMode	100
GetHostIOError	75	Set/GetPhaseAngle	101
GetLoopCommand	76	Set/GetPhaseCorrectionMode	102
GetLoopError	77	Set/GetPhaseCounts	103
GetLoopIntegral	78	Set/GetPhasePrescale	104
GetPWMCommand	79	Set/GetPWMDeadTime	105
GetSignalStatus	80	Set/GetPWMFrequency	107
GetTemperature	81	Set/GetPWMLimit	108
GetVersion	82	Set/GetPWMOutputMode	109
NoOperation	83	Set/GetPWMSense	110
Reset	84	Set/GetSampleTime	111
ResetEventStatus	85	Set/GetSerialPortMode	112
Set/GetAcceleration	86	Set/GetSignalSense	113
Set/GetActualPosition	87	Set/GetSPISyncMode	114
Set/GetAnalogOffset	88	Set/GetTemperatureLimit	115
Set/GetAutoStopMode	89	Set/GetVelocity	116
Set/GetBusVoltageLimits	90	Set/GetVelocityFeedbackSource	117
Set/GetCommandSource	91	Set/GetVelocityScalar	118
Set/GetCommutationMode	92	StoreUserData	119

This page intentionally left blank.

Index

Numerics

- 16-bit motor coil voltage commands 27
- 16-bit registers 44
- 16-bit synchronous serial SPI-port 36
- 16-bit velocity representation 37
- 3-signal mode 25
- 50/50 PWM output 28
- 6-signal mode 25

A

- absolute maximum ratings 13
- AC characteristics 14
- acceleration profiles 42
- active condition 47
- activity status 44, 47
 - register 44
- actual motor currents 30
- Actual Position 57
- actual position registers 57
- address byte 58
- amplifier
 - disable 22
 - error 47
 - error condition 47
- analog
 - current signals 31
 - input voltage 38
- analog feedback
 - signals 25, 30
 - value 39
- analog input
 - pin 39, 61
 - specifications 14
- analog signal
 - input 36
 - range 61
- argument
 - numeric 67
 - required 67
- associated data words 52
- asynchronous
 - frame 55

- serial communication 53
- serial port 51
- auto stop mode 40
- automatic
 - motor shutdown mode 39
 - phase correction 35
- axis
 - position 42
 - velocity 39

B

- baud rates 54
- bias offset 39
- bi-polar signals 61
- bit
 - field 47
 - mask 47
 - oriented status registers 44
- boot 59
 - serial EEPROM 52
- byte-oriented packet format 58

C

- calibrating amplifier circuitry 32
- capture
 - received indicator 57
 - register 57
- checksum 53, 54
 - byte 53, 54
 - invalid 54
 - serial 54
 - valid 54
- chip
 - programming events 47
 - set action 52
- coil voltage 28
- command
 - associated data 52
 - desired velocity 25
 - packet 53
- command packet 52, 54, 56, 58
 - format 52

- commanded acceleration 42
- commutation 21
 - methods 25
 - module 36, 38
 - rate 43
 - reliability 35
- commutator 28, 30, 40
 - module 25, 30, 60
- complete intelligent motion controller 23
- condition
 - bits 47
 - mask 47
- control
 - application configurations 26
 - signals 55
- control loop 22
 - structure 26
- converting voltages to values 61
- counts per cycle 38
- current
 - feedback 61
 - integral value 41
 - sensing 61
 - signal 61
- current loop 21, 30
 - calculations 31
 - control 22
 - filter 30
 - module 25
- cycle time 43

D

- data
 - frame format 55
 - packet 54
 - transfer 52
 - words 67
- dedicated
 - amplifier chip 57
 - motion controller 23
- default sample time 60
- desired
 - current 30
 - velocity 39
 - velocity command 25, 36
- digital current controller 30
- direct analog signal 60
 - input 21
- disabling PWM output 48

- downstream modules 40

E

- EEPROM 21, 58
 - command format 58
- emergency stop 22
 - processing 21
- encoded-field argument 67
- encoder 33
 - counts 33
 - data stream 36
 - feedback 38
- Estop 22, 47
 - bit 46
 - pin 46
- event status 44, 47
 - register 44
 - register definition 44
 - word 40, 42
- external
 - adjusting circuitry 39
 - controller 23
 - dropping resistors 22
 - serial EEPROM 24
 - signal 25
 - voltage sources 39

F

- feedback circuit 31
- fixed torque value 32
- flash
 - memory 21, 24, 58
 - memory, erasing 59
 - memory, writing 59

G

- gain factors 23
- general-purpose current control 31

H

- half-bridge amplifiers 22
- Hall
 - sensor 22, 33
 - sensor input 21, 34
 - states 34
- Hall-based
 - commutation 34
 - technique 33
- hexadecimal code 67

- high impedance 48
- high speed
 - capture 56
 - position capture function 57
- host
 - device 21
 - I/O commands 53
 - I/O errors 53
 - microprocessor 24
- I**
- I2C
 - bus 58
 - interface 21
 - temperature sensor input 21
- IC cycle time 37
- idle-line protocol 56
- IGBT 22
- incremental encoder input 56
- index
 - pulse 35, 56
 - sense mask 57
 - signal 57
- input
 - voltage range 61
 - voltages 14
- instantaneous
 - axis location 57
 - velocity 25, 43
 - velocity loop error 38
- instruction
 - code 55
 - mnemonic 67
 - syntax 67
- integrated position 42
- integration limit 38
- intelligent motion controller 26
- internal
 - flash facility 52
 - profile generator 22, 42, 52
- invalid checksums 54
- inverted bits, signal status register 46

K

- Kicurrent 38
- Kpcurrent 38

L

- latched bits 44

- limit value 29
- load operational parameters 58
- loop
 - configuration 60
 - rate 43

M

- main control loop 24
- manual mode 32
- maximum
 - counts per cycle 38
 - error 39
 - negative current output 30
 - on time 30
 - output value 29
 - positive current output 30
 - velocity 42
 - velocity error 39
- MC73110 physical characteristics 12
- measured velocity 37
- mechanical friction 42
- minimum
 - off-time 29, 30
 - shoot-through time 28
 - timeout period 55
- mode
 - three-signal output 29
 - torque 25, 36
 - voltage 25, 36
- module
 - motor output 26
 - profile generator 26
 - velocity integrator 25
- monitoring chip status 52
- MOSFET 22
- motion control card 23
- motion error 32, 39, 42
 - bit 40
 - condition 32
 - facility 42
 - processing 42
- motor
 - coils 22
 - command limit 32
 - command register 32, 40, 42
 - currents 30
 - mode 43
 - output logic 27
 - output module 26

mounting dimensions 12
MSB 59
multi-drop
 idle-line mode 54, 55
 mode 55
 protocol 56

N

negative velocity values 43
numeric argument 67
numerical registers 44

O

offset
 bias 31
 value 39
output
 enable pin 55
 values 28
 voltages 14
over temperature 47
 bit 50
overall control loop flow 24
over-range current 31
over-temperature
 set point 50
 value 50

P

packet
 command 53
 response 53
 structure 67
parity bit 55
phase
 calculations 35
 correction mode 35
PI filter 30
PID filter 41
pin
 descriptions 17
 functionality 18
point-to-point 55
 configuration 56
 mode 55
 serial mode 54, 55
position
 based motion error 42
 capture 45

 capture, indicator 57
 capture, register 57
 error 42
 feedback signals 57
positioning motion processor 23
positive
 velocity values 43
 voltage referenced 30
potentiometers 39
power-up 21
pre-commutated motor command 33
profile generation 21
profile generator 25, 32, 36, 38, 40, 42, 43
 module 26
 velocity 43
programmable
 maximum error 39
 shoot-through function 22
 shoot-through timer 28
programmed shoot-through delay 29
proportional integral 30
protocol layering 59
prototyping 51, 58
PWM
 coil 61
 cycle 28
 generator 28, 30
 output 25, 47, 48
 output disable 22
 output generator 30
 output rate 43
 signal 25
 synchronized signals 27
 update frequency 28
 waveforms 27

Q

quadrature
 counts 57
 encoder 25
 incremental position 57
quadrature encoder
 counts 35
 data 25
 input 21

R

real-time signal levels 45
recommended

- minimum timeout 56
- operating conditions 13
- register, position capture 57
- required arguments 67
- reset 21
 - defaults 58
- response
 - frames 53
 - packet 52, 53
 - packet format 52
- resynchronizing with the chip 56

S

- sample time 43
- scaling 36
- serial
 - buffer IC 55
 - checksum 54
 - data transfer 55
 - EEPROM 21, 57
 - EEPROM file 58
 - hardware signals 55
 - mode, point-to-point 55
 - transfers 55
- serial port 51, 52
 - command I/O 21
 - command packet 58
 - commands 23
 - communication 52
 - configuration 54
 - default configuration 54
- servo loop
 - rate 43, 60
 - update cycle 41
- shoot-through
 - delay 29
 - programmable 22
 - protection off-times 29
 - timer 29
- signal pins 46
- signal sense mask
 - default 45
 - register 46
 - value 46
- signal sense register 57
- signal status 44, 47
 - register 45, 46
 - register definition 45
- single host processor 56
- single-axis device 21
- single-phase 25
- sinusoidal
 - commutation 32, 33
 - commutation mode 35
 - lookup technique 33
- six-signal
 - mode 28
 - output 27
- software offset bias 31
- SPI
 - data 60
 - data stream 25, 40, 60
 - interface 38
 - port 59
 - value 38
- square-wave signals 56
- standard loop configurations 26
- status
 - event word 40, 58
 - words 44
- stop bits 55
- supported temperature sensors 50
- symmetric
 - values 61
 - waveforms 28
- synchronized PWM signals 27
- synchronous serial
 - (SPI) data stream 22
 - SPI data input 21
- syntax, instruction 67

T

- tachometer 36
 - signal 37, 39
 - velocity source 37
- temperature
 - data 50
 - limit, default value 50
 - limit, register 50
 - sensing devices 50
 - sensor 50
- three-phase
 - brushless DC motors 21
 - PWM signal generation 21
- three-signal
 - output 27
 - output mode 29
- timeout period 56

- torque 21
 - mode 25, 36
 - mode, amplifier 26
 - ripple 28
 - set point value 22
- trajectory
 - generator 43
 - profile commands 43
- transfer
 - read 67
 - write 67
- transmission
 - byte 53
 - errors 53
 - protocols. 55
- transmit line 55
- trigger condition 57
- triggering chip events 47
- tri-stating 55

U

- unused bits 67

V

- valid checksum 54
- velocity
 - based, motion error 42
 - bounded, profiles 42
 - command 39
 - control 21
 - error 39, 42
 - error monitoring 42
 - feedback 36
 - integrator 40, 42, 60
 - mode, amplifier 26
 - overshoot transient 38
 - pin 25
 - profile 40
 - scalar 37, 38
 - scalar register 40
 - source 36, 37
 - values 40
- velocity integrator 36, 43
 - filter 41
 - loop 32, 38
 - module 25
- velocity loop 21, 36, 42, 43, 60
 - calculation 38
 - filter 36, 41

- module 25
- update rate 36
- velocity values
 - negative 43
 - positive 43
- voltage
 - mode 25, 36
 - mode, control 21
 - output 61
 - states 46
 - value 32

Z

- zero
 - current 31
 - data packet 56
 - reference 31, 39

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>



Performance Motion Devices
80 Central Street
Boxborough, MA 01719

