Navigator® Motion Processor

Programmer's Reference

P M D

Performance Motion Devices, Inc.
55 Old Bedford Road
Lincoln, MA 01773

Revision 1.7, November 2003

NOTICE

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

Navigator Motion Processor User's Guide (MC2000UG)

How to set up and use all members of the Navigator Motion Processor family.

Navigator Motion Processor Programmer's Reference (MC2000PR)

Descriptions of all Navigator Motion Processor commands, with coding syntax and examples, listed alphabetically for quick reference.

Navigator Motion Processor Technical Specifications

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

MC2100 Series, for brushed servo motion control (MC2100TS);
MC2300 Series, for brushless servo motion control (MC2300TS);
MC2400 Series, for microstepping motion control (MC2400TS);
MC2500 Series, for stepping motion control (MC2500TS);
MC2800 Series, for brushed servo and brushless servo motion control (MC2800TS).

Navigator Motion Processor Developer's Kit Manual (DK2000M)

How to install and configure the DK2000 developer's kit PC board.

Table of Contents

1 The Navigator Family

	MC2100 Series	MC2300 Series	MC2400 Series	MC2500 Series	MC2800 Series
# of axes	4, 2, or 1	4, 2 or 1	4, 2 or 1	4, 2, or 1	4 or 2
Motor type supported	Brushed servo	Brushless servo	Stepping	Stepping	Brushed servo + brushless servo
Output format	Brushed servo (single phase)	Commutated (6-step or sinusoidal)	Microstepping	Pulse and direction	Brushed servo (single phase) + commutated (6-step sinusoidal)
Incremental encoder input	√	√	√	√	√
Parallel word device input	√	√	√	√	√
Parallel communication	√	√	√	√	√
Serial communication	√	√	√	√	√
Diagnostic port	√	√	√	√	√
S-curve profiling	√	√	√	√	√
Electronic gearing	√	√	√	√	√
On-the-fly changes	√	√	√	√	√
Directional limit switches	√	√	√	√	√
Programmable bit output	√	√	√	√	√
Software-invertable signals	√	√	√	√	√
PID servo control	√	√	-	-	√
Feedforward (accel & vel)	√	√	-	-	√
Derivative sampling time	√	√	-	-	√
Data trace/diagnostics	√	√	√	√	√
PWM output	√	√	√	-	√
Motion error detection	√	√	√ (with encoder)	√ (with encoder)	√
Axis settled indicator	√	√	√ (with encoder)	√ (with encoder)	√
DAC-compatible output	√	√	√	-	√
Pulse & direction output	-	-	-	√	-
Index & Home signals	√	√	√	√	√
Position capture	√	√	√	√	√
Analog input	√	√	√	√	√
User-defined I/O	√	√	√	√	√
External RAM support	√	√	√	√	√
Chipset part numbers	MC2140 (4 axes) MC2120 (2 axes) MC2110 (1 axis)	MC2340 (4 axes) MC2320 (2 axes) MC2310 (1 axis)	MC2440 (4 axes) MC2420 (2 axes) MC2410 (1 axis)	MC2540 (4 axes) MC2520 (2 axes) MC2510 (1 axis)	MC2840 (4 axes) MC2820 (2 axes)
Developer's Kit p/n's:	DK2100	DK2300	DK2400	DK2500	DK2800

Introduction

This manual describes the format of instructions supported by the Navigator family of Motion Processors from PMD. These devices are members of PMD's second-generation motion processor family, which consists of 12 separate products organized into 4 series.

Each of these devices are a complete chip-based motion processors. They provide trajectory generation and related motion control functions. Depending on the type of motor controlled they provide servo loop closure, on-board commutation for brushless motors, and high speed pulse and direction outputs. Together these products provide a software-compatible family of dedicated motion processors that can handle a large variety of system configurations.

Each of these chips utilize a similar architecture, consisting of a high-speed DSP (Digital Signal Processor) computation unit, along with an ASIC (Application Specific Integrated Circuit). The computation unit contains special on-board hardware that makes it well suited for the task of motion control.

Along with similar hardware architecture these chips also share most software commands, so that software written for one chipset may be re-used with another, even though the type of motor may be different.

Each chipset consists of two PQFP (Plastic Quad Flat Pack) ICs: a 100-pin Input/Output (I/O) chip, and a 132-pin Command Processor (CP) chip.

The four different series in the Navigator family are designed for a particular type of motor or control scheme. Here is a summary description of each series.

Family Summary

MC2100 Series (MC2140, MC2120, MC2110) – This series outputs motor commands in either Sign/Magnitude PWM or DAC-compatible format for use with brushed servo motors, or with brushless servo motors having external commutation.

MC2300 Series (MC2340, MC2320, MC2310) – This series outputs sinusoidally commutated motor signals appropriate for driving brushless motors. Depending on the motor type, the output is a two-phase or three-phase signal in either PWM or DAC-compatible format.

MC2400 Series (MC2440, MC2420, MC2410) – This series provides microstepping signals for stepping motors. Two phased signals per axis are generated in either PWM or DAC-compatible format.

MC2500 Series (MC2540, MC2520, MC2510) – These chipsets provide high-speed pulse and direction signals for stepping motor systems.

MC2800 Series (MC2840, MC2820) – This series outputs sinusoidally or 6-step commutated motor signals appropriate for driving brushless servo motors as well as PWM or DAC- compatible outputs for driving brushed servo motors.

2 Instruction Reference

2.1 How to use this reference

This document is in two parts: first, a detailed description of all host instructions, and second, a set of summary tables listing the instructions by functional group, alphabetically by instruction mnemonic, and numerically by hexadecimal code.

In the reference 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 page. The page is organized as follows:

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 There are two types of arguments: encoded-field and numeric.

Encoded-field arguments are packed into a single 16-bit data word, except for axis, which occupies bits 11-8 of the instruction word. The ***Name*** of the argument is that shown in the generic syntax. ***Instance*** mnenomic used to represent the data value. ***Encoding*** is the value assigned to the field for that instance.

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.

Buffered Certain parameters and other data written to the chipset are buffered, that is, they are not acted upon until the next Update or MultiUpdate command is executed. These parameters are identified by the word **buffered** in the instruction heading.

Packet structure This is a graphic representation of the 16-bit words transmitted in the packet: the instruction, which is identified by its name, followed by 1, 2, or 3 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 31 to 16, the low-order bits from 15 to 0.

The hex code of the instruction is shown in boldface.

Argument names are shown in their respective words or fields.

For data words, the direction of transfer—read or write—is shown at the left of the word's diagram.

Unused bits are shaded. **In data words and instructions sent (written) to the motion processor, all unused bits must be 0**.

Description Describes what the instruction does and 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. For example, velocity, acceleration, deceleration, and jerk parameters may not be issued while an S-curve profile is being executed.

see Refers to related instructions.

AdjustActualPosition F5h

Syntax

AdjustActualPosition *axis position*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	***Range***	***Scaling***	***Units***
position	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts|steps

Packet structure

AdjustActualPosition

0	*axis*	**F5**h
15 … 12	11 … 8	7 … 0

First data word

write	*position* (high-order part)
	31 … 16

Second data word

write	*position* (low-order part)
	15 … 0

Description

The *position* specified as the parameter to AdjustActualPosition is summed with the actual position register (encoder position) for the specified *axis.* This has the effect of adding or subtracting an offset to the current actual position. At the same time, the current commanded position is replaced by the new actual position value minus the current actual position error. This prevents a servo "bump" when the new axis position is established. The destination position (see SetPosition) is also modified by this amount so that no trajectory motion will occur when the update instruction is issued. In effect, this instruction establishes a new reference position from which subsequent positions can be calculated. It is commonly used to set a known reference position after a homing procedure.

Note: On the MC2400 and MC2500 series, the current actual position error is zeroed.

AdjustActualPosition takes effect immediately, it is not buffered.

Restrictions

see

GetPositionError; GetActualVelocity, Set/GetActualPositionUnits, Set/GetActualPosition

ClearInterrupt ACh

Syntax ClearInterrupt

Arguments none

Packet structure

ClearInterrupt	
0	ACh
15 8	7 0

Description ClearInterrupt resets the HostInterrupt signal to its inactive state. If interrupts are still pending, the HostInterrupt line will return to its active state within one cycle. It is used after an interrupt has been recognized and processed by the host. This command does not affect the Event Status Register. If this command is executed when no interrupts are pending it has no effect.

Restrictions

see GetInterruptAxis, Set/GetInterruptMask

ClearPositionError buffered 47h

Syntax

ClearPositionError *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet structure

ClearPositionError

15–12	11–8	7–0
0	*axis*	**47**h

Description

ClearPositionError sets the current profile's commanded position equal to the actual position (encoder input), thereby clearing the position error for the specified *axis*. This command can be used when the axis is at rest, or when it is moving. If it is used when the axis is moving the host should be aware that the trajectory destination position (used in trapezoidal and s-curve modes) is not changed by this command.

Restrictions

ClearPositionError is a buffered command. The new value set will not take effect until the next Update or MultiUpdate instruction is entered.

This command cannot be executed while the chip is performing an s-curve profile.

see

GetPositionError, MultiUpdate, Set/GetPositionErrorLimit, Update

GetActivityStatus A6h

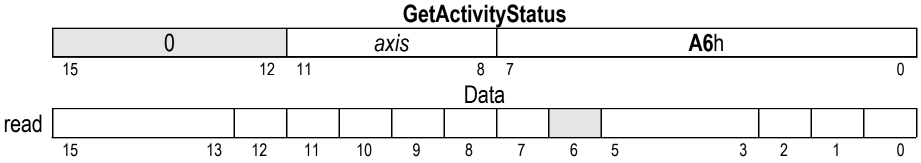
Syntax GetActivityStatus *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data *status* *see below*

Packet structure



Description

GetActivityStatus reads the 16 bit activity status register for the specified *axis*. Each of the bits in this register continuously indicate the state of the chipset without any action on the part of the host. There is no direct way to set or clear the state of these bits, since they are controlled by the chip set.

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

Name	*Bit Number*	*Description*
Phasing initialized	0	Set to 1 if phasing is initialized (MC2300/MC2800 series only)
At maximum velocity	1	Set to 1 when the trajectory is at maximum velocity. This bit is determined by the trajectory generator, not the actual encoder position.
Tracking	2	Set to 1 when the axis is within the tracking window
Current profile mode	3-5	Contains trajectory mode encoded as follows: bit 5 bit 4 bit 3 Profile Mode 0 0 0 trapezoidal 0 0 1 velocity contouring 0 1 0 s-curve 0 1 1 electronic gear
reserved	6	not used, may be 0 or 1
Axis settled	7	Set to 1 when the axis is settled
Motor on/off	8	Set to 1 when motor mode is on, 0 when off.
Position capture	9	Set to 1 when a value has been captured by the high speed position capture hardware but has not yet been read. The GetCaptureValue command must be executed before another capture can occur.
In-motion	10	Set to 1 when the trajectory generator is executing a profile on the axis.
In positive limit	11	Set to 1 when the positive limit switch is active

Name	*Bit Number*	*Description*
In negative limit	12	Set to 1 when the negative limit switch is active
Profile segment	13-15	When the profile mode is S-curve it contains the profile segment number 1-7 while profile is in motion and contains a value of 0 when the profile is at rest. When the External profile mode is used it contains a 1 while the trajectory generator is processing data and 0 otherwise. This field is undefined when using the Trapezoidal and Velocity Contouring profile modes.

Restrictions

see GetEventStatus, GetSignalStatus

GetActualVelocity ADh

Syntax

GetActualVelocity *axis*

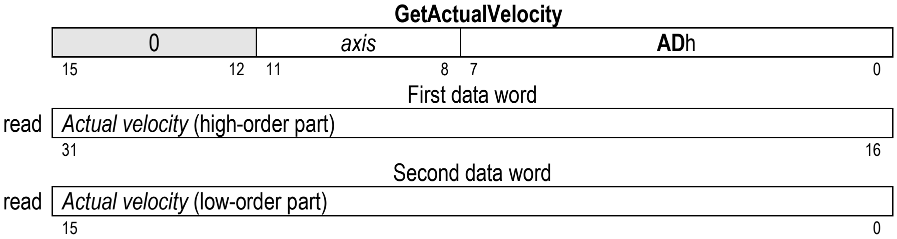
Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	*Range*	*Scaling*	*Units*
velocity	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	counts/cycle

Packet structure



Description

GetActualVelocity reads the current actual velocity for the specified *axis*. This value is the result of the last encoder input, so it will be accurate to within one cycle.

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.

Restrictions

The actual velocity is derived by subtracting the actual postion during the previous chip cycle from the actual position for this chip cycle. The result of this subtraction will always be integer because position is always integer. As a result the value returned by GetActualVelocity will always be a multiple of 65536 since this represents a value of one in the 16.16 number format. The low word is always zero.

see

GetCommandedVelocity

GetCaptureValue 36h

Syntax GetCaptureValue *axis*

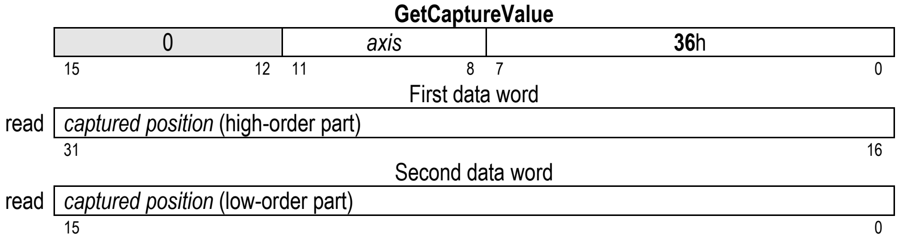
Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	*Range*	*Scaling*	*Units*
captured position	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts

Packet structure



Description GetCaptureValue returns the contents of the Position Capture Register for the specified *axis*. This command also resets the capture hardware to allow another capture to occur.

Restrictions

see Set/GetCaptureSource

GetChecksum F8h

Syntax GetChecksum

Returned data

	Type
checksum	unsigned 32 bits

Packet structure

GetChecksum

0	**F8**h
15 8	7 0

First data word

read	*Checksum* (high-order part)
	31 16

Second data word

read	*Checksum* (low-order part)
	15 0

Description GetChecksum reads the chips internal 32-bit checksum value. The value should be 12345678 (hex) for a correctly manufactured chipset.

Restrictions

see

GetCommandedAcceleration A7h

Syntax GetCommandedAcceleration *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	*Range*	*Scaling*	*Units*
acceleration	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	counts/cycle^2

Packet structure

GetCommandedAcceleration

0	*axis*	**A7**h
15 12	11 8	7 0

First data word

read	*acceleration* (high-order part)
	31 16

Second data word

read	*acceleration* (low-order part)
	15 0

Description GetCommandedAcceleration returns the current commanded acceleration value for the specified *axis*. Commanded acceleration is the instantaneous acceleration value output by the trajectory generator.

Scaling example: If a value of 114,688 is retrieved using this command then this corresponds to 114,688/65,536 = 1.750 counts/cycle^2 acceleration value.

Restrictions This command functions when the profile mode is set to Trapezoidal, S-curve, or Velocity Contouring. It does not function when the profile mode is set to electronic gearing.

see GetCommandedPosition, GetCommandedVelocity

GetCommandedPosition 1Dh

Syntax GetCommandedPosition *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	*Range*	*Scaling*	*Units*
position	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts

Packet structure

GetCommandedPosition

0	*axis*	**1D**h
15 12	11 8	7 0

First data word

read	*position* (high-order part)
	31 16

Second data word

read	*position* (low-order part)
	15 0

Description GetCommandedPosition returns the current commanded position for the specified *axis*. Commanded position is the instantaneous position value output by the trajectory generator.

This command functions in all profile modes.

Restrictions

see GetCommandedAcceleration, GetCommandedVelocity

GetCommandedVelocity 1Eh

Syntax GetCommandedVelocity *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	***Range***	***Scaling***	***Units***
velocity	signed integer	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	counts/cycle

Packet structure

GetCommandedVelocity

0	*axis*	**1E**h
15 12	11 8	7 0

First data word

read	*velocity* (high-order part)
	31 16

Second data word

read	*velocity* (low-order part)
	15 0

Description GetCommandedVelocity returns the current commanded velocity value for the specified *axis*. Commanded velocity is the instantaneous velocity value output by the trajectory generator.

Scaling example: If a value of -1,234,567 is retrieved using this command (FFEDh in high word, 2979h in low word) then this corresponds to -1,234,567/65,536 = -18.8380 counts/cycle velocity value.

This command functions in all profile modes.

Restrictions

see GetCommandedAcceleration, GetCommandedPosition

GetCurrentMotorCommand 3Ah

Syntax

GetCurrentMotorCommand *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	*Range*	*Scaling*	*Units*
motor output command	signed 16 bits	-2^{15} *to* $2^{15}-1$	$100/2^{15}$	% output

Packet structure

GetCurrentMotorCommand

0	*axis*	**3A**h
15 12	11 8	7 0

First data word

read	*motor output command*
	15 0

Description

GetCurrentMotorCommand returns the current motor output command for the specified *axis*. In closed-loop mode, this is the output of the servo filter; in open-loop mode it is the contents of the motor output command register.

Scaling example: To convert the retrieved value to units of % of full scale motor output multiply by 100/32,768. For example if the value -123 is retrieved by the GetCurrentMotorCommand, this represents -123*100/32,768 or -.3754 % of full scale output.

Restrictions

This command is not available on the MC2500 chipset.

see

Set/GetMotorCommand

GetDerivative *(Servo products only)* 9Bh

Syntax GetDerivative *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	*Range*	*Scaling*	*Units*
derivative	signed 16 bits	-2^{15} *to* $2^{15}-1$	unity	counts/cycle

Packet structure

GetDerivative

0	*axis*	**9B**h
15 12	11 8	7 0

Data

read	*derivative*
	15 0

Description GetDerivative returns the derivative of the current position error as calculated by the servo filter. The derivative value is defined as the previous position error subtracted from the current position error.

See SetDerivativeTime for details on setting the derivative sampling time.

Restrictions This value is available only when the chipset is in closed-loop operation.

This command is not valid on the MC2400 and MC2500.

see GetIntegral, Set/GetDerivativeTime

GetEventStatus 31h

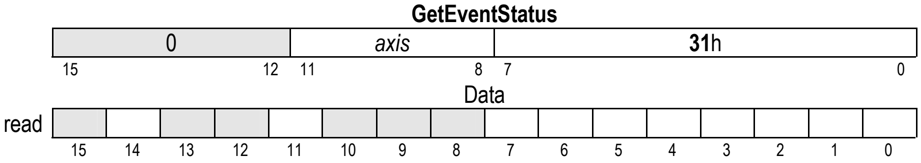
Syntax GetEventStatus *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data see below

Packet structure



Description GetEventStatus reads the event register for the specified *axis*.

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

Name	*Bit(s)*	*Description*
Motion complete	0	Set to 1 when motion is completed. SetMotionCompleteMode determines if this bit is based on the trajectory generator position or the encoder position.
Wrap-around	1	Set to 1 when the actual (encoder) position wraps from maximum allowed position to minimum or vice versa
Breakpoint 1	2	Set to 1 when breakpoint 1 is triggered
Capture received	3	Set to 1 when a position capture occurs
Motion error	4	Set to 1 when a motion error occurs
In positive limit	5	Set to 1 when the axis enters a positive limit switch condition
In negative limit	6	Set to 1 when the axis enters a negative limit switch condition
Instruction error	7	Set to 1 when instruction error occurs
reserved	8-10	Not used, may be 0 or 1.
Commutation error	11	Set to 1 when a commutation error occurs
reserved	12-13	Not used, may be 0 or 1.
Breakpoint 2	14	Set to 1 when breakpoint 2 is triggered
reserved	15	Not used, may be 0 or 1.

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

see GetActivityStatus, GetSignalStatus

GetHostIOError A5h

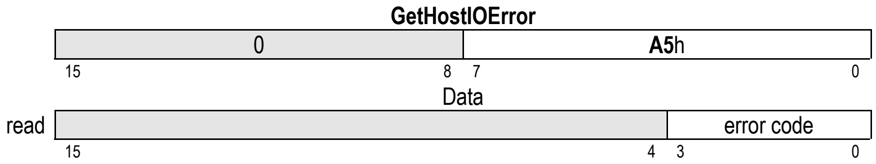
Syntax GetHostIOError

Arguments none

Returned data

Name	*Instance*	*Encoding*
error code	No error	0
	Processor Reset	1
	Invalid instruction	2
	Invalid axis	3
	Invalid parameter	4
	Trace running	5
	reserved	6
	Block out of bounds	7
	Trace buffer zero	8
	Bad serial checksum	9
	Not primary port	Ah
	Invalid negative value	Bh
	Invalid parameter change	Ch
	Invalid move after limit condition	Dh
	Invalid move into limit	Eh

Packet structure



Description GetHostIOError returns the code for the last Host I/O error, then resets to 0 both the *error* and the Host I/O bit in the Status-Read word. Generally this command is issued only after the Host I/O error bit in the Status-read word indicates there was an I/O error.

Restrictions

see GetEventStatus

GetIntegral *(Servo products only)* 9Ah

Syntax GetIntegral *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	*Range*	*Scaling*	*Units*
integral	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^8$	count*cycles

Packet structure

GetIntegral

0	*axis*	**9A**h
15 12	11 8	7 0

First data word

read	*Integrated position error* (high-order part)
	31 16

Second data word

read	*Integrated position error* (low-order part)
	15 0

Description GetIntegral returns the current integrated position error of the servo filter for the specified *axis*. GetIntegral can be used to monitor loading on the axis, because changes in the axis loading can be reflected in the value of the integration limit.

Scaling example:

If a constant position error of 100 counts is present for 256 cycles than the total accumulated integral value will be 100 (100*256/256). Alternatively a returned value of 1,000 indicates a total stored value of 256,000 count*cycles (1,000*256).

Restrictions The integrated position error is available only when the chipset is in closed-loop mode (SetMotorMode command).

This command is not valid on the MC2400 and MC2500.

see GetDerivative, Set/GetIntegrationLimit

GetInterruptAxis E1h

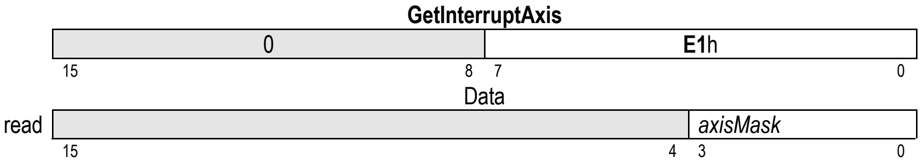
Syntax GetInterruptAxis

Arguments none

Returned data

Name	*Instance*	*Encoding*
axisMask	Axis1	1
	Axis2	2
	Axis3	4
	Axis4	8

Packet structure



Description GetInterruptAxis returns a field which identifies all axes with pending interrupts. Axis numbers are assigned to the low-order four bits of the returned word; bits corresponding to interrupting axes are set to 1. If the host interrupt signal has not been set, the returned word is 0.

Restrictions

see ClearInterrupt, Set/GetInterruptMask

GetPhaseCommand *(MC2300, MC2400 and MC2800 only)* EAh

Syntax GetPhaseCommand *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
phase	PhaseA	0
	PhaseB	1
	PhaseC	2

Returned data

	Type	*Range*	*Scaling*	*Units*
motor command	signed 16 bit	-2^{15} *to* $2^{15}-1$	$100/2^{15}$	% output

Packet structure

GetPhaseCommand

0	*axis*	**EA**h
15 12	11 8	7 0

First data word

write	0	*phase*
	15 3	2 0

Second data word

read	*motor command*
	15 0

Description

GetPhaseCommand returns the value of the current motor output command for phase A, B, or C of the specified axis. These are the phase values directly output to the motor after commutation.

Scaling example:

If a value of -4,489 is retrieved (EE77h) for a given axis and phase then this corresponds to -4,489*100/32,768 = -13.7 % of full-scale output.

Restrictions

see InitializePhase, Set/GetNumberPhases

GetPositionError 99h

Syntax GetPositionError *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Type	***Range***	***Scaling***	***Units***
position error	signed 32 bit	-2^{31} *to* $2^{31}-1$	unity	counts|steps

Packet structure

GetPositionError

0	*axis*	**99**h
15 12	11 8	7 0

First data word

read	*position error* (high-order part)
	31 16

Second data word

read	*position error* (low-order part)
	15 0

Description GetPositionError returns the current position error of the specified *axis*. The error is the difference between the actual position (encoder position) and the commanded position (instantaneous output of the trajectory generator). Refer to the User's Guide for more information on this command when it is used with the stepping motor chipsets.

Restrictions

see Set/GetPosition, Set/GetPositionErrorLimit

GetSignalStatus A4h

Syntax GetSignalStatus *axis*

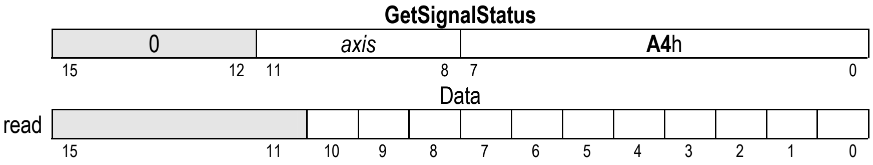
Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Returned data

	Description	*Bit Number*
status	Encoder A	0
	Encoder B	1
	Encoder Index	2
	Encoder Home	3
	Positive limit	4
	Negative limit	5
	AxisIn	6
	Hall A	7
	Hall B	8
	Hall C	9
	AxisOut	10
	reserved	11-15

Packet structure



Description

GetSignalStatus returns the contents of the signal status register for the specified *axis*. The signal status register contains the current value of the various hardware signals connected to each axis of the chipset. The value read is combined with the signal sense register (SetSignalSense command) 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, so that a low signal will be read as 1 and a high signal will be read as a 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, so that a low signal will be read as 0 and a high signal will be read as a 1.

All of the bits in the GetSignalStatus command are inputs except for AxisOut. The value read for this bit is equal to the current value output by the axis out mechanism. See SetAxisOutSource command for more details.

Restrictions

see GetActivityStatus, GetEventStatus

GetTime 3Eh

Syntax GetTime

Arguments none

Returned data

Name	*Type*	*Range*	*Scaling*	*Units*
current chipset time	unsigned 32 bit	0 *to* 2^{32}-1	unity	cycles

Packet structure

GetTime

0	**3E**h
15 8	7 0

First data word

read	*current chipset time* (high-order part)	
	31	16

Second data word

read	*current chipset time* (low-order part)	
	15	0

Description Returns the number of cycles that have occurred since the processor was last initialized or reset.

Restrictions

see

GetTraceCount BBh

Syntax GetTraceCount

Arguments none

Returned data

Value	*Type*	*Range*	*Scaling*	*Units*
trace count	unsigned 32 bit	0 *to* 2^{32}-1	unity	samples

Packet structure

GetTraceCount

0	**BB**h
15 8	7 0

First data word

read *trace count* (high-order part)
31 16

Second data word

read *trace count* (low-order part)
15 0

Description GetTraceCount returns the number of points (variable values) stored in the trace buffer since the beginning of the trace.

Restrictions

see ReadBuffer, Set/GetTraceStart, Set/GetTraceStop

GetTraceStatus BAh

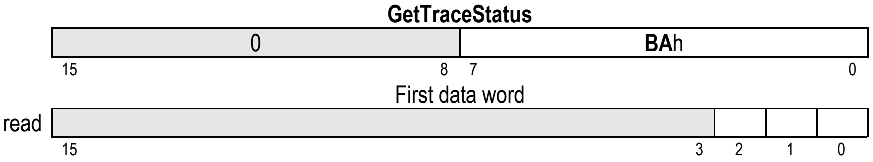
Syntax GetTraceStatus

Arguments none

Returned data

Name	*Bit*	*Instance*	*Description*
mask	0	Mode	Set to 0 when trace is in one-time mode, 1 when in rolling mode.
	1	Activity	Set to 1 when trace is active (currently tracing) , 0 if trace not active
	2	Data wrap	Set to 1 when trace has wrapped, 0 if it has not wrapped. If 0, the buffer has not yet been filled and all recorded data are intact. If 1, the trace has wrapped to the beginning of the buffer; any previous data may have been overwritten if not explicitly retrieved by the host using the ReadBuffer command while the trace is active.

Packet structure



Description GetTraceStatus returns the current trace status.

Restrictions

see Set/GetTraceStart, Set/GetTraceMode

GetVersion 8Fh

Syntax GetVersion

Arguments None

Returned data

Product information		*Encoding*
product family	Navigator	2
motor type	Servo	1
	Brushless	3
	Microstepping	4
	Pulse & Direction	5
	Multiple Motor	8
axes supported		1, 2, *or* 4
special attributes		0 *to* 15
customization code	none	0
	other	1 *to* 255
major s/w version		0 *to* 15
minor s/w version		0 *to* 15

Packet structure

GetVersion

	0	8Fh
	15 … 8	7 … 0

First data word

read	*product family*	*motor type*	*number of axes*	*special attributes*
	15 … 12	11 … 8	7 … 4	3 … 0

Second data word

read	*customization code*	*major s/w version*	*minor s/w version*
	15 … 8	7 … 4	3 … 0

Description GetVersion returns product information encoded as shown above.

Restrictions

see

InitializePhase *(MC2300 and MC2800 only)* 7Ah

Syntax InitializePhase *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet structure

InitializePhase

0	*axis*	**7A**h
15 12	11 8	7 0

Description

InitializePhase initializes the phase angle for the specified axis using the mode (Hall-based or Algorithmic) specified by the SetPhaseInitializationMode command.

Restrictions

Warning: If the phase initialization mode has been set to algorithmic then after this command is sent the motor can move suddenly in an uncontrolled manner.

This command is only applicable in the sinusoidal Commutation Mode. (see SetCommutationMode)

see GetPhaseCommand, Set/GetNumberPhases

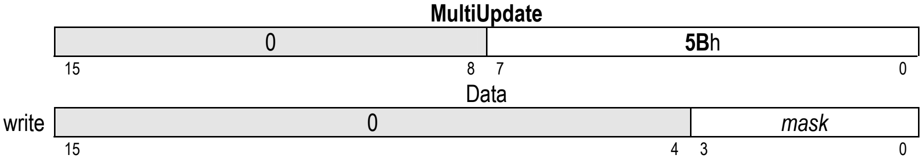
MultiUpdate 5Bh

Syntax MultiUpdate *mask*

Arguments

Name	*Instance*	*Encoding*
mask	None	0
	Axis1mask	1
	Axis2mask	2
	Axis3mask	4
	Axis4mask	8

Packet structure



Description

MultiUpdate causes an Update to occur on all axes whose corresponding bit is set to 1 in the mask argument. After this command is executed, and for those axes which are selected using the mask, all buffered data parameters are copied into the corresponding run-time registers.

The following instruction is buffered: ClearPositionError.

The following trajectory parameters are buffered: Acceleration, Deceleration, GearRatio, Jerk, Position, ProfileMode, StartVelocity, StopMode, and Velocity.

The following PID filter parameters are buffered: DerivativeTime, IntegrationLimit, Kaff, Kd, Ki, Kp, and Kvff.

The following Motor Command parameter is buffered: MotorCommand

Restrictions

see Update

NoOperation 00h

Syntax NoOperation

Arguments none

Packet structure

NoOperation

0		00h	
15	8	7	0

Description The NoOperation command has no affect on the chipset. It is useful as a “null” operation to verify communications with the Motion Processor.

Restrictions

see

ReadAnalog EFh

Syntax ReadAnalog *portID*

Arguments

	Name	*Type*	*Range*	*Scaling*	*Units*
	portID	unsigned 16 bit	0 to 7	unity	-
Returned data	*value*	unsigned 16 bit	0 *to* 2^{16}-1	$1/2^{16}$	% input

Packet structure

ReadAnalog

0	EFh
15 8	7 0

First data word

write	0	*portID*
	15	0

Second data word

read	*value*
	15 0

Description ReadAnalog returns a 16-bit value representing the voltage (read by an on-chip 10 bit A/D) presented to the specified analog input. See User's Guide for more information on analog input and scaling. The value returned is the result of shifting the 10-bit value 6 bits left.

Restrictions

see

ReadBuffer C9h

Syntax ReadBuffer *bufferID*

	Name	Type	Range	Scaling	Units
Arguments	*bufferID*	unsigned 16 bit	0 to 31	unity	-
Returned data	*value*	signed 32 bit	-2^{31} *to* $2^{31}-1$	unity	-

Packet structure

ReadBuffer

	15 … 8	7 … 0
	0	**C9**h

First data word

	15 … 4	3 … 0
write	0	*bufferID*

Second data word

	31 … 16
read	*buffer contents* (high-order part)

Third data word

	15 … 0
read	*buffer contents* (low-order part)

Description

ReadBuffer returns the 32-bit contents of the current location in the specified buffer. The current location is determined by adding the base address of the buffer (set by SetBufferStart), to the buffer's Read Index (set by SetBufferReadIndex). After the contents have been read, the Read Index is incremented by 1; if the result is equal to the buffer length (set by SetBufferLength), the Index is reset to 0.

Some commands automatically change the read index such as at the completion of a trace when in rolling mode. Refer to Section 7.6.4 of the User's Guide for details.

Restrictions

see Set/GetBufferReadIndex, WriteBuffer

ReadIO 83h

Syntax ReadIO *address*

Arguments

	Name	*Type*	*Range*	*Scaling*	*Units*
Arguments	*address*	unsigned 8 bit	0 to 255	unity	-
Returned data	*value*	unsigned 16 bit	0 *to* 2^{16}-1	unity	-

Packet structure

ReadIO

	0	**83**h
	15 12 11 8	7 0

First data word

write	0	*address*
	15 8	7 0

Second data word

read	*data*
	15 0

Description ReadIO reads one 16-bit word of data from the device whose address is calculated by adding 1000h to *address*. (*address* is an offset from the base address, 1000h, of the MC2000's memory-mapped I/O space.)

The format and interpretation of the 16-bit data word are dependent on the user-defined device being addressed. User-defined I/O can be used to implement a number of features including additional parallel I/O, flash memory for non-volatile configuration information storage, or display devices such as LED arrays.

Restrictions

see WriteIO

Reset 39h

Syntax

Reset

Arguments

none

Packet structure

Reset

0	39h
15 … 8	7 … 0

Description

Reset restores the chipset to its initial condition, setting all chipset variables to their default values. These default values are shown in the following table:

Acceleration	0	MotorBias	0
ActualPosition	0	MotorCommand	0
AutoStopMode	1	MotorLimit	32767
AxisMode	1	MotorMode	1
AxisOutSource	0	NumberPhases	*see note 1*
Breakpoint 1	0	OutputMode	*see note 2*
Breakpoint 2	0	PhaseAngle	65535
BreakpointValue 1	0	PhaseCorrectionMode	1
BreakpointValue 2	0	PhaseCounts	1
BufferLength	0	PhaseInitializeMode	0
BufferReadIndex	0	PhaseInitializeTime	0
BufferStart	200h	PhaseOffset	65535
BufferWriteIndex	0	PhasePrescale	0
CaptureSource	0	Position	0
CommutationMode	0	PositionErrorLimit	2^{31}-1
Deceleration	0	ProfileMode	0
DerivativeTime	1	SampleTime	*see note 3*
EncoderModulus	0	SettleTime	0
EncoderSource	0	SettleWindow	0
GearMaster	0	SignalSense	0
GearRatio	0	Stop	0
GetActualPositionUnits	0	TraceMode	0
IntegrationLimit	0	TracePeriod	1
InterruptMask	0	TraceStart	0
Jerk	0	TraceStop	0
Kaff	0	TraceVariable 1	0
Kd	0	TraceVariable 2	0
Ki	65535	TraceVariable 3	0
Kout	0	TraceVariable 4	0
Kp	0	TrackingWindow	0
Kvff	1	Velocity	0
LimitMode	0		
MotionCompleteMode			

Notes:

1. The reset value for the number of phases is dependent on the Motion Processor series, as follows:

MC2100	1
MC2300	3
MC2400	2
MC2800	3

2. The reset value for the output mode is dependent on the Motion Processor series, as follows:

MC2100	1
MC2300	2
MC2400	1
MC2800	2

3. The reset value for SampleTime depends on the number of axes and the motion processor series, as follows:

MC2100	102 x number of axes
MC2300	154 x number of axes
MC2400	154 x number of axes
MC2500	102 x number of axes
MC2800	154 x number of axes

All axes supported by the motion processor are enabled at reset.

Profile, servo filter, and other axis-specific parameters are reset on all axes. External-memory buffer parameters are reset for all buffers. BufferStart is reset to (200h), the lowest user-accessible address.
Axis-specific conditions are reset on all axes. External-memory buffer conditions are reset on all 32 memory buffers.

Restrictions

For the MC2400/MC2500:

AutoPositionUnits Counts

AutoStopMode Off

EncoderSource None

For the MC2500:

StepRange 1

see

ResetEventStatus

Syntax ResetEventStatus *axis mask*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mask	Motion complete	0001h
	Wrap-around	0002h
	Breakpoint 1	0004h
	Capture received	0008h
	Motion error	0010h
	In positive limit	0020h
	In negative limit	0040h
	Instruction error	0080h
	Commutation error	0800h
	Breakpoint 2	4000h

Packet structure

ResetEventStatus

0	*axis*	**34**h
15 12	11 8	7 0

Data

write	0		0	0		0	0	0	*mask*
		14			11				7 0

Description ResetEventStatus clears (sets to 0) , for the specified *axis*, 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.

Restrictions

see GetEventStatus

SetAcceleration buffered 90h
GetAcceleration 4Ch

Syntax

SetAcceleration *axis acceleration*
GetAcceleration *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
acceleration	unsigned 32 bit	0 *to* 2^{31}-1	1/2^{16}	counts/cycle^2

Packet structure

SetAcceleration

0	*axis*	**90**h
15 12	11 8	7 0

First data word

write	*acceleration* (high-order part)
	31 16

Second data word

write	*acceleration* (low-order part)
	15 0

GetAcceleration

0	*axis*	**4C**h
15 12	11 8	7 0

First data word

read	*acceleration* (high-order part)
	31 16

Second data word

read	*acceleration* (low-order part)
	15 0

Description

SetAcceleration loads the maximum acceleration buffer register for the specified *axis*. This command is used with the Trapezoidal, Velocity Contouring, and S-curve profiling modes.

GetAcceleration reads the maximum acceleration buffer register set by the previous SetAcceleration command.

Scaling example: To load a value of 1.750 counts/cycle^2 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 correspondingly be divided by 65,536 to convert to units of counts/cycle^2.

Restrictions

SetAcceleration may not be issued while an axis is in motion with the S-curve profile.

SetAcceleration is not valid in Electronic Gearing profile mode.

SetAcceleration is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.

see

Set/GetDeceleration, Set/GetJerk, Set/GetPosition, Set/GetVelocity, MultiUpdate, Update

SetActualPosition 4Dh
GetActualPosition 37h

Syntax

SetActualPosition *axis position*
GetActualPosition *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	***Range***	***Scaling***	***Units***
position	signed 32 bits	-2^{31} *to* $2^{31}-1$	unity	counts|steps

Packet structure

SetActualPosition

0	*axis*	**4D**h
15 — 12	11 — 8	7 — 0

First data word

write	*position* (high-order part)
	31 — 16

Second data word

write	*position* (low-order part)
	15 — 0

GetActualPosition

0	*axis*	**37**h
15 — 12	11 — 8	7 — 0

First data word

read	*position* (high-order part)
	31 — 16

Second data word

read	*position* (low-order part)
	15 — 0

Description

SetActualPosition loads the actual position register (encoder position) for the specified *axis*. At the same time, the current commanded position is replaced by the loaded value minus the current actual position error. This prevents a servo "bump" when the new axis position is established. The destination position (see SetPosition) is also modified by this amount so that no trajectory motion will occur when the update instruction is issued. In effect, this instruction establishes a new reference position from which subsequent positions can be calculated. It is commonly used to set a known reference position after a homing procedure.

Note: On the MC2400 and MC2500 series, the position error is zeroed.

SetActualPosition takes effect immediately, it is not buffered.

GetActualPosition reads the contents of the encoder's actual position register. This value will be the result of the last encoder input, which will be accurate to within one cycle (as determined by Set/GetSampleTime).

Restrictions

see

GetPositionError; GetActualVelocity, Set/GetActualPositionUnits, AdjustActualPosition

SetActualPositionUnits *(MC2400 and MC 2500 only)* BEh
GetActualPositionUnits *(MC2400 and MC 2500 only)* BFh

Syntax

SetActualPositionUnits *axis mode*
GetActualPositionUnits *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	Counts	0
	Steps	1

Packet structure

SetActualPositionUnits

	0	*axis*	**BE**h
	15 … 12	11 … 8	7 … 0

Data

	0	*mode*
write	15 … 1	0

GetActualPositionUnits

	0	*axis*	**BF**h
	15 … 12	11 … 8	7 … 0

Data

		mode
read	15 … 1	0

Description

SetActualPositionUnits determines the units used by the Set/GetActualPosition, AdjustActualPosition and GetCaptureValue for the specified *axis*. When set to *Counts* position units are in encoder counts. When set to *Steps* GetActualPosition position units are in steps.

GetActualPositionUnits returns the mode for the specified *axis*.

Restrictions

This command is only available on the MC2400 and MC2500 series.

see

Set/GetActualPosition, Set/GetEncoderToStepRatio, AdjustActualPosition, GetCaptureValue

SetAutoStopMode D2h
GetAutoStopMode D3h

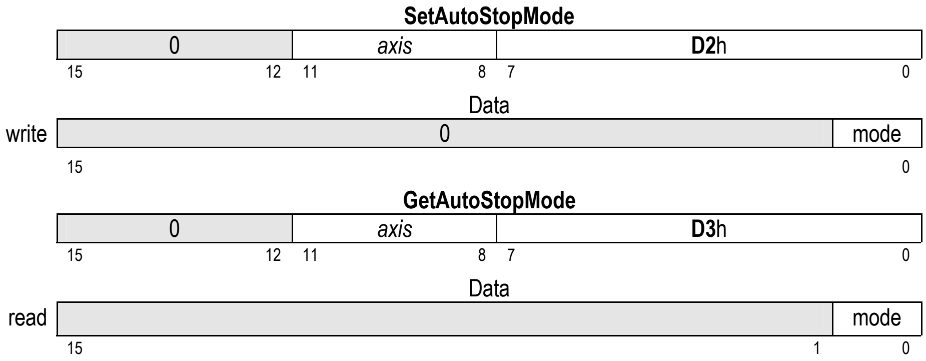
Syntax

SetAutoStopMode *axis mode*
GetAutoStopMode *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	Disable	0
	Enable	1

Packet structure



Description

SetAutoStopMode determines the behavior of the specified *axis* 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

When the encoder source is set to none (SetEncoderSource None), setting the auto stop mode to Enable will not stop motion in the event that the position error limit is exceeded.

see

GetEventStatus, SetPositionErrorLimit

SetAxisMode 87h
GetAxisMode 88h

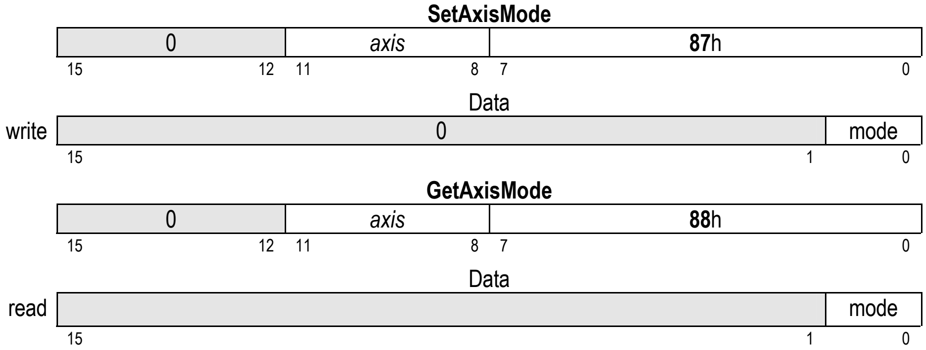
Syntax

SetAxisMode *axis mode*
GetAxisMode *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	off	0
	on	1

Packet structure



Description

SetAxisMode enables (On) or disables (Off) the specified *axis*. A disabled axis will not respond to profile or other motion commands.

GetAxisMode returns the current status of the specified axis.

Restrictions

Disabled axes do not provide encoder feedback. If it is desired that an axis provide encoder feedback even though no profiling or servo control is to be used, that axis must be left enabled.

see

SetAxisOutSource EDh
GetAxisOutSource EEh

Syntax

SetAxisOutSource *axis sourceAxis bit register*
GetAxisOutSource *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
sourceAxis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
bit	*see below*	0 to 15
register	(none)	0
	EventStatus	1
	ActivityStatus	2
	SignalStatus	3

Packet structure

SetAxisOutSource

0	*axis*	**ED**h
15 12	11 8	7 0

Data

write	0	*register*	*bit*	*sourceAxis*
	15 12	11 8	7 4	3 0

GetAxisOutSource

0	*axis*	**EE**h
15 12	11 8	7 0

Data

read		*register*	*bit*	*sourceAxis*
	15 12	11 8	7 4	3 0

Description

SetAxisOutSource maps the specified *bit* of the specified status *register* of *axisn* to the AxisOut pin for the specified *axis*. The state of the AxisOut pin will thereafter track the state of *bit*. If *register* is absent (encoding of 0), *bit* is ignored, and the specified AxisOut pin is, in effect, turned off (inactive).

GetAxisOutSource reads the mapping of the AxisOut pin of *axis*.

The table below shows the corresponding value for combinations of *bit* and *register*.

encoding of "bit"	***register = event status***	***register = activity status***	***register = signal status***
0	Motion Complete	Phasing Initialized	Encoder A
1	Wrap-around	At maximum velocity	Encoder B
2	Breakpoint 1	Tracking	Encoder index
3	Position capture		Home
4	Motion error		Positive limit
5	In positive limit		Negative limit
6	In negative limit		AxisIn
7	Instruction error	Axis settled	Hall sensor 1
8		Motor on/off	Hall sensor 2
9		Position capture	Hall sensor 3
0Ah		In motion	
0Bh	Commutation error	In positive limit	
0Ch		In negative limit	
0Dh			
0Eh	Breakpoint 2		
0Fh			

Restrictions

see SetSignalSense

SetBreakpoint D4h
GetBreakpoint D5h

Syntax

SetBreakpoint *axis breakpoint sourceAxis action trigger*
GetBreakpoint *axis breakpoint*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
breakpoint	Breakpoint1	0
	Breakpoint2	1
sourceAxis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
action	(none)	0
	Update	1
	AbruptStop	2
	SmoothStop	3
	MotorOff	4
trigger	(none)	0
	GreaterOrEqualCommandedPosition	1
	LesserOrEqualCommandedPosition	2
	GreaterOrEqualActualPosition	3
	LesserOrEqualActualPosition	4
	CommandedPositionCrossed	5
	ActualPositionCrossed	6
	Time	7
	EventStatus	8
	ActivityStatus	9
	SignalStatus	Ah

Packet structure

SetBreakpoint

	0	*axis*	**D4**h
	15 12	11 8	7 0

First data word

write	0	*breakpoint*
	15 1	0

Second data word

write	*trigger*	*action*	*sourceAxis*
	15 8	7 4	3 0

GetBreakpoint

	0	*axis*	**D5**h
	15 12	11 8	7 0

First data word

write	0	*breakpoint*
	15 1	0

Second data word

read	*trigger*	*action*	*sourceAxis*
	15 8	7 4	3 0

Description

SetBreakpoint establishes a breakpoint for the specified *axis* to be triggered by a condition or event on *sourceAxis*, which may be the same as or different from *axis.* Up to two concurrent breakpoints can be set for each axis.

The six Position breakpoints and the Time breakpoint are *threshold-triggered*; the breakpoint occurs when the indicated value reaches or crosses a threshold. The Status breakpoints are *level-triggered*; the breakpoint occurs when a specific bit or combination of bits in the indicated status register changes state. Thresholds and bit specifications are both set by the SetBreakpointValue instruction.

action determines what the Navigator does when the breakpoint occurs, as follows:

Action	**Resultant command sequence**
none	no action
Update	Update *axis*
AbruptStop	The profile executes an abrupt stop
SmoothStop	The profile executes a smooth stop
MotorOff	SetMotorMode *axis*, Off

axis is the axis for which the breakpoint has been set.

GetBreakpoint returns the trigger, action, and axis for the specified breakpoint (1 or 2) of the indicated axis. When a breakpoint occurs the trigger value will be reset to none. The CommandedPositionCrossed and the ActualPositionCrossed triggers are converted to one of the Position trigger types 1-4 depending on the current position when the command is issued.

Two completely separate breakpoints are supported, each of which may have its own breakpoint type and comparison value. The ***breakpoint*** field specifies which breakpoint the SetBreakpoint and GetBreakpoint commands will address.

Restrictions

Before setting a new breakpoint condition (SetBreakpoint command) ALWAYS load the comparison value first (SetBreakpointValue command). This is because as soon as the breakpoint condition is set the chipset will start using the breakpoint value register, and if it is not yet defined the breakpoint will not behave as expected.

see

Set/GetBreakpointValue

SetBreakpointValue D6h
GetBreakpointValue D7h

Syntax

SetBreakpointValue *axis breakpoint value*
GetBreakpointValue *axis breakpoint*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
breakpoint	Breakpoint1	0		
	Breakpoint2	1		
		Type	***Range***	***Units***
value	GreaterOrEqualCommandedPosition	signed 32 bit	-2^{31} *to* $2^{31}-1$	counts
	LesserOrEqualCommandedPosition	signed 32 bit	-2^{31} *to* $2^{31}-1$	counts
	GreaterOrEqualActualPosition	signed 32 bit	-2^{31} *to* $2^{31}-1$	counts
	LesserOrEqualActualPosition	signed 32 bit	-2^{31} *to* $2^{31}-1$	counts
	CommandedPositionCrossed	signed 32 bit	-2^{31} *to* $2^{31}-1$	counts
	ActualPositionCrossed	signed 32 bit	-2^{31} *to* $2^{31}-1$	counts
	Time	unsigned 32 bit	0 *to* $2^{32}-1$	cycles
	EventStatus	2 word mask*	-	-
	ActivityStatus	2 word mask*	-	-
	SignalStatus	2 word mask*	-	-

* see description section below for more details on mask format

Packet structure

SetBreakpointValue

	0	*axis*	**D6**h
	15 – 12	11 – 8	7 – 0

First data word

	0	*breakpoint*
write	15 – 1	0

Second data word

	value (high-order part)
write	31 – 16

Third data word

	value (low-order part)
write	15 – 0

GetBreakpointValue

	0	*axis*	**D7**h
	15 – 12	11 – 8	7 – 0

First data word

	0	*breakpoint*
write	15 – 1	0

Second data word

	value (high-order part)
read	31 – 16

Third data word

	value (low-order part)
read	15 – 0

Description

SetBreakpointValue sets the breakpoint comparison value for the specified ***axis***. For the position and time breakpoints this is a threshold comparison value.

For level-triggered breakpoints, the high-order part of ***value*** is the selection mask, and the low-order word is the sense mask. For each selection bit that is set to 1, the corresponding bit of the specified status register is conditioned to cause a breakpoint when it changes state. The sense-mask bit determines which state causes the break. If it is 1, the corresponding status-register bit will cause a break when it is set to 1. If it is 0, the status-register bit will cause a break when it is set to 0.

For example assume it is desired that the breakpoint type will be set to "EventStatus" and that a breakpoint should be recognized whenever the motion complete bit (bit 0 of event status register) is set to 1, or the commutation error bit (bit 11 of event status register) is set to 0. In this situation the high and low words for ***value*** would be high word: 0x801 (hex) and low word: 1.

GetBreakpointValue returns the current breakpoint value for the specified breakpoint.

Two completely separate breakpoints are supported, each of which may have its own breakpoint type and comparison value. The ***breakpoint*** field specifies which breakpoint the SetBreakpointValue and GetBreakpointValue commands will address.

Restrictions

Before setting a new breakpoint condition (SetBreakpoint command) ALWAYS load the comparison value first (SetBreakpointValue command). This is because as soon as the breakpoint condition is set the chipset will start using the breakpoint value register, and if it is not yet defined the breakpoint will not behave as expected.

see

Set/GetBreakpoint

SetBufferFunction CAh
GetBufferFunction CBh

Syntax

SetBufferFunction *axis function bufferID*
GetBufferFunction *axis function*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
function	Position	0
	Velocity	1
	Acceleration	2
	Jerk	3
	Time	4

Name	***Type***	***Range***	***Scaling***	***Units***
bufferID	signed 16 bits	-1 *to* 31	unity	-

Packet structure

SetBufferFunction

	0	*axis*	**CA**h
	15 12	11 8	7 0

First data word

write	*function*
	15 0

Second data word

write	*bufferID*
	15 0

GetBufferFunction

	0	*axis*	**CB**h
	15 12	11 8	7 0

First data word

write	*function*
	15 0

Second data word

read	*bufferID*
	15 0

Description

SetBufferFunction sets the interpretation for data stored in a buffer when an axis is in External Profile mode. A function will have no associated buffer if the bufferID parameter is set to -1. This is useful for disabling a function.

GetBufferFunction returns the bufferID for the specified function. If a function has not been assigned a buffer, the return value is –1.

Restrictions

see Set/GetProfileMode

SetBufferLength C2h
GetBufferLength C3h

Syntax

SetBufferLength *bufferID length*
GetBufferLength *bufferID*

Arguments

Name	*Type*	*Range*	*Scaling*	*Units*
bufferID	unsigned 16 bits	0 to 31	unity	-
length	unsigned 32 bits	1 to 2^{30}-1	unity	-

Packet structure

SetBufferLength

	0	**C2**h
	15 8	7 0

First data word

write	0	*bufferID*
	15 5	4 0

Second data word

write	*length* (high-order part)
	31 16

Third data word

write	*length* (low-order part)
	15 0

GetBufferLength

	0	**C3**h
	15 8	7 0

First data word

write	0	*bufferID*
	15 5	4 0

Second data word

read	*length* (high-order part)
	31 16

Third data word

read	*length* (low-order part)
	15 0

Description

SetBufferLength sets the length, in number of 32-bit elements, of the buffer in the memory block identified by *bufferID*.

Note: SetBufferLength resets the buffers read and write indexes to 0.

GetBufferLength returns the length of the specified buffer.

Restrictions

If the specified length extends beyond the end of addressable memory, SetBufferLength is not executed, and returns host-I/O error code 7, *buffer bound exceeded.*

Note: **Setting the buffer length beyond the end of physical memory could cause the chip set to unexpectedly reset during operation.**

see

Set/GetBufferReadIndex; Set/GetBufferStart; Set/GetBufferWriteIndex

SetBufferReadIndex C6h
GetBufferReadIndex C7h

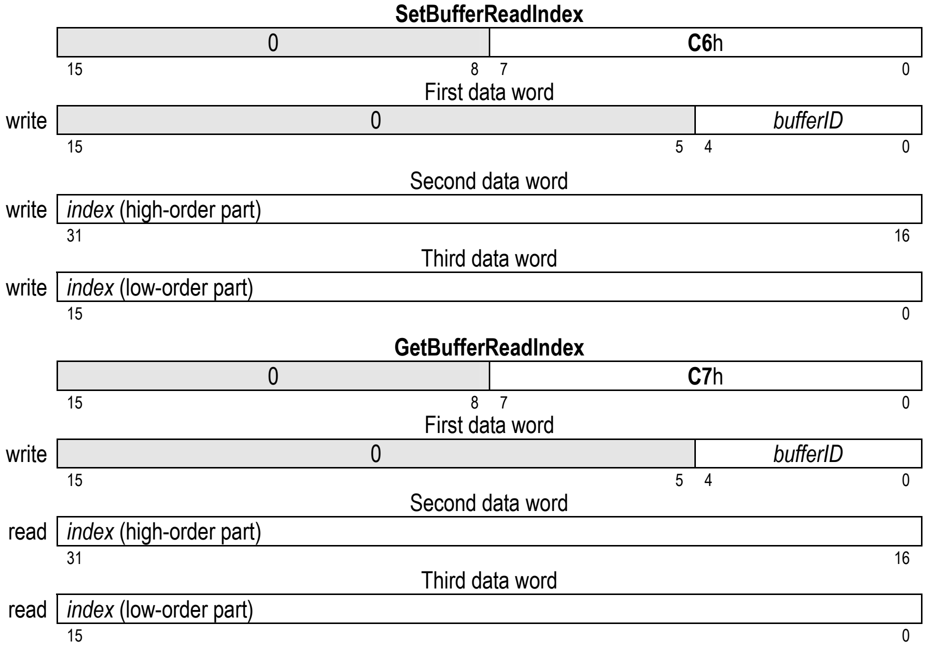
Syntax

SetBufferReadIndex *bufferID index*
GetBufferReadIndex *bufferID*

Arguments

Name	***Type***	***Range***	***Scaling***	***Units***
bufferID	unsigned 16 bits	0 *to* 31	unity	-
index	unsigned 32 bits	0 to buffer length-1	unity	double words (32 bit)

Packet structure



Description

SetBufferReadIndex sets the address of the Read Index for the specified buffer. If the read index is set to an address beyond the length of the buffer, the command will not be executed and will return an error.

GetBufferReadIndex returns the current Read Index for the specified buffer.

Restrictions

see Set/GetBufferLength, Set/GetBufferStart, Set/GetBufferWriteIndex

SetBufferStart C0h
GetBufferStart C1h

Syntax

SetBufferStart *bufferID address*
GetBufferStart *bufferID*

Arguments

Name	*Type*	*Range*	*Scaling*	*Units*
bufferID	unsigned 16 bit	0 *to* 31	unity	-
address	unsigned 32 bit	2^9 *to* 2^{31}-1	unity	double words (32 bit)

Packet structure

SetBufferStart

	15–8	7–0
	0	**C0**h

First data word

	15–5	4–0
write	0	*bufferID*

Second data word

	31–16
write	*address* (high-order part)

Third data word

	15–0
write	*address* (low-order part)

GetBufferStart

	15–8	7–0
	0	**C1**h

First data word

	15–5	4–0
write	0	*bufferID*

Second data word

	31–16
read	*address* (high-order part)

Third data word

	15–0
read	*address* (low-order part)

Description

SetBufferStart sets the starting address for the specified buffer**. The buffer start address must be 200h or greater**.

Note: SetBufferStart **resets the buffers read and write indexes to 0.**

GetBufferStart returns the starting address for the specified buffer.

Restrictions

If the specified length extends beyond the end of addressable memory, SetBufferStart is not executed, and returns host-I/O error code 7, *buffer bound exceeded.*

Note: **Setting the buffer start beyond the end of physical memory could cause the chip set to unexpectedly reset during operation.**

see

Set/GetBufferLength, Set/GetReadIndex, Set/GetBufferWriteIndex

SetBufferWriteIndex C4h
GetBufferWriteIndex C5h

Syntax

SetBufferWriteIndex *bufferID index*
GetBufferWriteIndex *bufferID*

Arguments

Name	*Type*	*Range*	*Scaling*	*Units*
bufferID	unsigned 16 bit	0 *to* 31	unity	-
index	unsigned 32 bit	0 *to* buffer length-1	unity	long words (32 bits)

Packet structure

SetBufferWriteIndex

	15 … 8	7 … 0
	0	**C4**h

First data word

	15 … 5	4 … 0
write	0	*bufferID*

Second data word

	31 … 16
write	*index* (high-order part)

Third data word

	15 … 0
write	*index* (low-order part)

GetBufferWriteIndex

	15 … 8	7 … 0
	0	**C5**h

First data word

	15 … 5	4 … 0
write	0	*bufferID*

Second data word

	31 … 16
read	*index* (high-order part)

Third data word

	15 … 0
read	*index* (low-order part)

Description

SetBufferWriteIndex sets the address of the write index for the specified buffer. If the write index is set to an address beyond the length of the buffer, the command will not be executed and will return an error.

GetBufferWriteIndex returns the current write index for the specified buffer.

Restrictions

see Set/GetBufferLength, Set/GetBufferReadIndex, Set/GetBufferStart

SetCaptureSource D8h
GetCaptureSource D9h

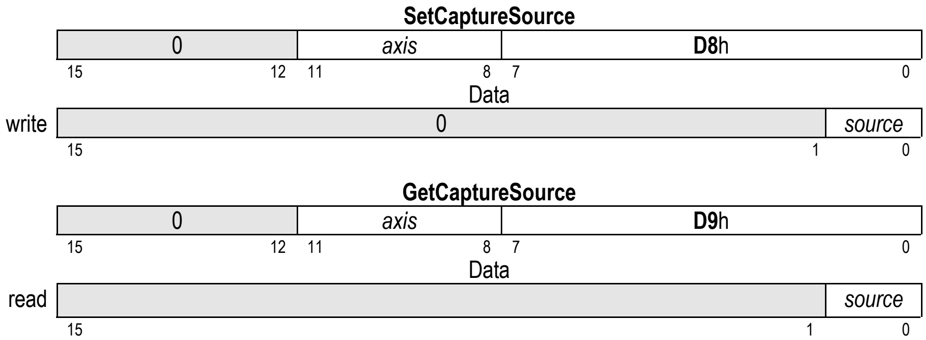
Syntax

SetCaptureSource *axis source*
GetCaptureSource *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
source	Index	0
	Home	1

Packet structure



Description

SetCaptureSource determines which of two encoder signals, Index or Home, is used to trigger the high-speed capture of the actual axis position for the specified *axis*.

GetCaptureSource returns the capture signal *source* for the selected *axis*.

Restrictions

see GetCaptureValue

SetCommutationMode *(MC2300 and MC2800 only)* E2h
GetCommutationMode *(MC2300 and MC2800 only)* E3h

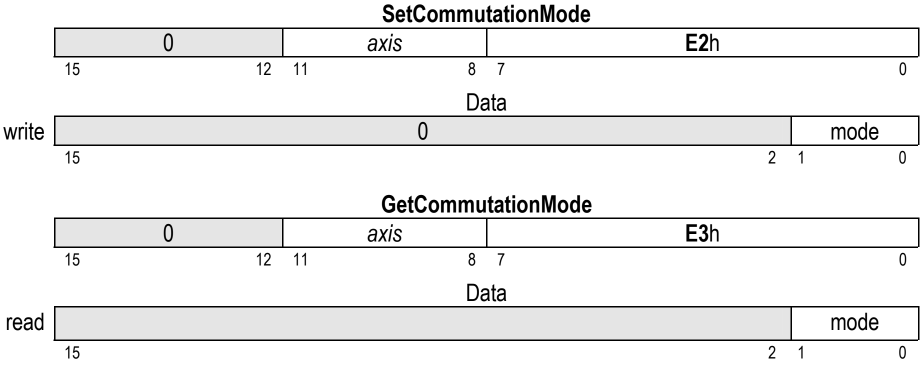
Syntax

SetCommutationMode *axis mode*
GetCommutationMode *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	Sinusoidal	0
	Hall-Based	1
	Microstepping	2

Packet structure



Description

SetCommutationMode sets the phase commutation mode for the specified *axis*.

When set to sinusoidal, as the motor turns, the encoder input signal is 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 "six-step" or "trapezoidal" waveform method.

When set to microstepping the output of the trajectory generator is used to calculate the phase angle. This angle is in turn used to generate sinusoidally varying outputs to each motor phase.

GetCommutationMode returns the current commutation mode.

When operating with brushless servo motors either sinusoidal or Hall-based are typically used for motor commutation.

Microstepping is sometimes used with brushless motors to "manually" move the motor before phase initialization has occurred. Alternatively, Microstepping can be used with step motors or with AC induction motors where frequency synthesis is all that is required to rotate the motor.

Restrictions

see

Set/GetCommutationPrescale, Set/GetCommutationCounts, Set/GetPhase commands

SetDeceleration buffered 91h
GetDeceleration 92h

Syntax

SetDeceleration *axis deceleration*
GetDeceleration *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
deceleration	unsigned 32 bits	0 *to* 2^{31}-1	$1/2^{16}$	counts/cycle^2

Packet structure

SetDeceleration

	0	*axis*	**91**h
	15 12	11 8	7 0

First data word

write	*deceleration* (high-order part)
	31 16

Second data word

write	*deceleration* (low-order part)
	15 0

GetDeceleration

	0	*axis*	**92**h
	15 12	11 8	7 0

First data word

read	*deceleration* (high-order part)
	31 16

Second data word

read	*deceleration* (low-order part)
	15 0

Description

SetDeceleration loads the maximum deceleration buffer register for the specified *axis*. This command sets the magnitude of the deceleration register, which always has a negative sign.

GetDeceleration reads the Maximum Deceleration buffer.

Scaling example: To load a value of 1.750 counts/cycle^2 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 (GetDeceleration) must correspondingly be divided by 65,536 to convert to units of counts/cycle^2

Restrictions

This is a buffered command. The new value set will not take effect until the next Update or MultiUpdate instruction is entered.

These commands are used with the Trapezoidal, S-curve, and Velocity contouring profile modes. They are not used with the electronic gearing profile mode.

Note: If deceleration is set to zero, then the value specified for acceleration (SetAcceleration) will automatically be used to set the magnitude of deceleration.

see

Set/GetAcceleration, Set/GetJerk, Set/GetPosition, Set/GetVelocity, MultiUpdate, Update

SetDerivativeTime *(Servo products only)* buffered 9Ch
GetDerivativeTime *(Servo products only)* 9Dh

Syntax

SetDerivativeTime *axis time*
GetDerivativeTime *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
time	unsigned 16 bits	0 *to* 2^{15}-1	unity	cycles

Packet structure

SetDerivativeTime

0	*axis*	**9C**h
15 12	11 8	7 0

Data

write	*time*
	15 0

GetDerivativeTime

0	*axis*	**9D**h
15 12	11 8	7 0

Data

read	*time*
	15 0

Description

SetDerivativeTime sets the sampling time, in number of servo cycles, for the servo filter to use in calculating the derivative term for the specified *axis*.

GetDerivativeTime returns the derivative sampling time.

Restrictions

This command does not affect the overall cycle time of the chipset, only the derivative sampling time. The overall cycle time of the chipset is set using the command SetSampleTime.

see

GetDerivative, GetIntegral, MultiUpdate, Update

SetDiagnosticPortMode 89h
GetDiagnosticPortMode 8Ah

Syntax

SetDiagnosticPortMode *mode*
GetDiagnosticPortMode

Arguments

Name	*Instance*	*Encoding*
mode	Limited	0
	Full	1

Packet structure

SetDiagnosticPortMode

	0	**89**h
	15 8	7 0

Data

write	0	mode
	15 1	0

GetDiagnosticPortMode

	0	**8A**h
	15 8	7 0

Data

read		mode
	15 1	0

Description

SetDiagnosticPortMode determines the instruction set that can be executed through the diagnostic (serial) port. When set to Limited, only the following instructions may be executed:

all Get instructions

The SetBufferReadIndex instruction

When set to Full, all instructions may be executed.

GetDiagnosticPortMode returns the current mode of the diagnostic port.

Restrictions

See

Set/GetSerialPortMode

SetEncoderModulus 8Dh
GetEncoderModulus 8Eh

Syntax

SetEncoderModulus *axis modulus*
GetEncoderModulus *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
modulus	unsigned 16 bit	1 *to* 2^{16}-1	unity	counts

Packet structure

SetEncoderModulus

0	*axis*	**8D**h
15 12	11 8	7 0

Data

write	*modulus*
	15 0

GetEncoderModulus

0	*axis*	**8E**h
15 12	11 8	7 0

Data

read	*modulus*
	15 0

Description

SetEncoderModulus sets the parallel word range for the specified *axis* when parallel-word feedback is used. *Modulus* determines the range of the connected device. The value provided should be one-half of the actual *modulus* of the axis. For example if the parallel-word input is used with a linear potentiometer connected to an external A/D (Analog to Digital converter) which has 12 bits of resolution, then the total range is 4,096 and a value of 2,048 should be loaded with this command.

GetEncoderModulus returns the current encoder modulus.

Restrictions

These commands are only used if parallel-word feedback is used. If incremental encoder feedback is used then these commands are not required.

see Set/GetEncoderSource

SetEncoderSource DAh
GetEncoderSource DBh

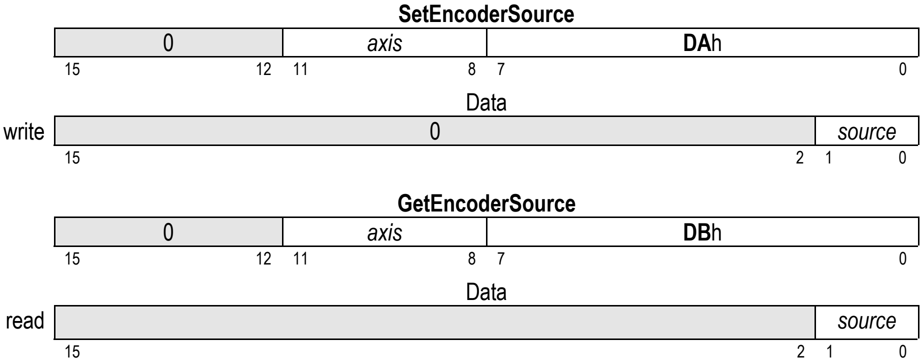
Syntax

SetEncoderSource *axis source*
GetEncoderSource *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
source	Incremental	0
	Parallel	1
	None	2

Packet structure



Description

SetEncoderSource sets the type of feedback (incremental quadrature encoder or parallel-word) for the specified *axis*. When incremental quadrature is selected the chip set expects A and B quadrature signals to be input at the I/O chip. When parallel-word is selected the chipset expects user-defined external circuitry connected to the chip set's external bus to load a 16-bit word containing the current position value for each axis. External feedback devices with less than 16 bits may be used but the unused bits must be sign extended or 'zeroed'.

GetEncoderSource returns the code for the current type of feedback.

Restrictions

see Set/GetEncoderModulus

SetEncoderToStepRatio *(MC2400 and MC2500 only)* DEh
GetEncoderToStepRatio *(MC2400 and MC2500 only)* DFh

Syntax

SetEncoderToStepRatio *axis counts steps*
GetEncoderToStepRatio *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
counts	unsigned 16 bit	0 *to* 2^{15}-1	unity	encoder counts
steps	unsigned 16 bit	0 *to* 2^{15}-1	unity	steps

Packet structure

SetEncoderToStepRatio

15 … 12	11 … 8	7 … 0
0	*axis*	**DE**h

First data word

	31 … 16
write	*counts*

Second data word

	15 … 0
write	*steps*

GetEncoderToStepRatio

15 … 12	11 … 8	7 … 0
0	*axis*	**DF**h

First data word

	31 … 16
read	*counts*

Second data word

	15 … 0
read	*steps*

Description

SetEncoderToStepRatio sets the ratio of number of encoder counts to the number of output steps per motor rotation used by the motion processor to convert encoder counts into steps/microsteps. *Counts* is the number of encoder counts per full rotation of the motor. *Steps* is the number of steps/microsteps output by the motion processor per full rotation of the motor. Since this command sets a ratio, the parameters do not have to be for a full rotation as long as they correctly represent the encoder count to step ratio.

GetEncoderToStepRatio gets the ratio of number of encoder counts to the number of output steps per motor rotation.

Restrictions

see Set/GetActualPositionUnits

SetGearMaster AEh
GetGearMaster AFh

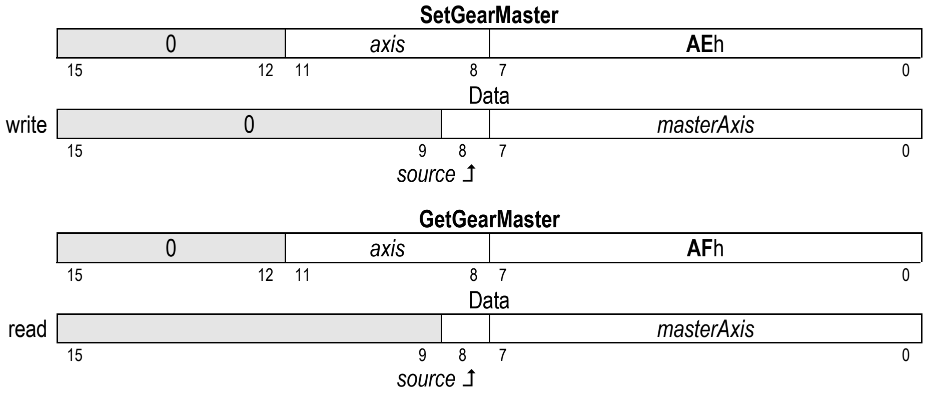
Syntax

SetGearMaster *axis masterAxis source*
GetGearMaster *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
masterAxis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
source	Actual	0
	Commanded	1

Packet structure



Description

SetGearMaster establishes the slave (*axis*) and master (*masterAxis*) axes for the electronic-gearing profile, and sets the source, Actual or Commanded, of the master axis position data to be used.

The masterAxis determines what axis will drive the slave axis. Both the slave and the master axes must be enabled (SetAxisMode command). The source determines whether the master axis' commanded position as determined by the trajectory generator will be used to drive the slave axis, or whether the master axis' encoder position will be used to drive the slave.

GetGearMaster returns the codes for the geared axes and position source.

Restrictions

For electronic gear mode to operate properly the master axis must be enabled.

see

Set/GetGearRatio

SetGearRatio GetGearRatio

buffered 14h 59h

Syntax

SetGearRatio *slaveAxis ratio*
GetGearRatio

Arguments

Name	***Instance***	***Encoding***		
slaveAxis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
ratio	signed 32 bits	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	SlaveCounts/ MasterCounts

Packet structure

SetGearRatio

0	*slaveAxis*	**14**h
15 12	11 8	7 0

First data word

write	*ratio* (high-order part)
	31 16

Second data word

write	*ratio* (low-order part)
	15 0

GetGearRatio

0	*slaveAxis*	**59**h
15 12	11 8	7 0

First data word

read	*ratio* (high-order part)
	31 16

Second data word

read	*ratio* (low-order part)
	15 0

Description

SetGearRatio sets the ratio between the master and slave axes for the electronic gearing profile for the current *axis*. Positive ratios cause the slave to move in the same direction as the master, negative ratios in the opposite direction. The specified ratio has a unity scaling of 65,536.

GetGearRatio returns the gear ratio set for the specified slave axis.

Scaling examples:

ratio value	**resultant ratio**
-32,768	.5 negative slave counts for each positive master count
1,000,000	15.259 positive slave counts for each positive master count
123	.0018 positive slave counts for each positive master count

Restrictions

This is a buffered command. The new value set will not take effect until the next Update or MultiUpdate instruction is entered.

See

Set/GetGearMaster, MultiUpdate, Update

SetIntegrationLimit *(Servo products only)* buffered 95h
GetIntegrationLimit *(Servo products only)* 96h

Syntax

SetIntegrationLimit *axis limit*
GetIntegrationLimit *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	***Range***	***Scaling***	***Units***
limit	unsigned 32 bits	0 *to* 2^{31}-1	1/2^8	count*cycles

Packet structure

SetIntegrationLimit

	0	*axis*	**95**h
	15 12	11 8	7 0

First data word

write	*limit* (high-order part)
	31 16

Second data word

write	*limit* (low-order part)
	15 0

GetIntegrationLimit

	0	*axis*	**96**h
	15 12	11 8	7 0

First data word

read	*limit* (high-order part)
	31 16

Second data word

read	*limit* (low-order part)
	15 0

Description

SetIntegrationLimit loads the integration-limit register of the digital servo filter for the specified *axis*.

GetIntegrationLimit returns the value of the current integration limit.

Scaling example: The scaling is the same as for the GetIntegral command, namely that (for example) a constant position error of 100 counts which is present for 256 cycles will result in an integral value of 100 (100*256/256) , and therefore an IntegrationLimit value of 100 will limit the total accumulated integration error to 25,600 count*cycles.

Restrictions

This is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.

This command is not valid on the MC2400 and MC2500.

see

GetIntegral, GetDerivative, Set/GetDerivativeTime, MultiUpdate, Update

SetInterruptMask 2Fh
GetInterruptMask 56h

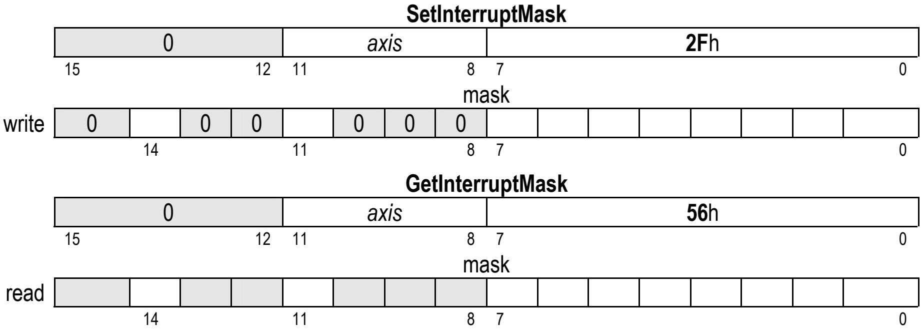
Syntax

SetInterruptMask *axis interruptMask*
GetInterruptMask *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
interruptMask	Motion complete	0001h
	Wrap-around	0002h
	Breakpoint 1	0004h
	Capture received	0008h
	Motion error	0010h
	In positive limit	0020h
	In negative limit	0040h
	Instruction error	0080h
	Commutation error	0800h
	Breakpoint 2	4000h

Packet structure



Description

SetInterruptMask determines which bits in the Event Status register of the specified axis will cause a host interrupt. For each interrupt mask bit that is set to 1, the corresponding Event Status register bit will cause an interrupt when that status register bit goes active (is set to 1). Interrupt mask bits set to 0 will not generate interrupts.

GetInterruptMask returns the current mask for the specified axis.

Example: The interrupt mask value 28h will generate an interrupt when either the "in positive limit" bit or the "capture received" bit of the event status register goes active (set to 1).

Restrictions

see ClearInterrupt, GetInterruptAxis

SetJerk buffered 13h
GetJerk 58h

Syntax

SetJerk *axis jerk*
GetJerk *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
jerk	unsigned 32 bits	0 *to* 2^{31}-1	$1/2^{32}$	counts/cycle^3

Packet structure

SetJerk

	0	*axis*	**13**h
	15 12	11 8	7 0

First data word

write	*jerk* (high-order part)
	31 16

Second data word

write	*jerk* (low-order part)
	15 0

GetJerk

	0	*axis*	**58**h
	15 12	11 8	7 0

First data word

read	*jerk* (high-order part)
	31 16

Second data word

read	*jerk* (low-order part)
	15 0

Description

SetJerk loads the jerk register in the parameter buffer for the specified *axis*.

GetJerk reads the contents of the Jerk register.

Scaling example: To load a jerk value (time rate of change of acceleration) of .012345 counts/cycle^3 multiply by 2^{32} or 4,294,967,296. In this example this gives a value to load of 53,021,371 (decimal) which corresponds to a high word of 0329h and a low word of 0ABBh when loading each word in hexadecimal.

Restrictions

SetJerk is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.

This command is used only with the S-curve profile mode. It is not used with the trapezoidal, velocity contouring, or electronic gear profile modes.

see

Set/GetAcceleration, Set/GetDeceleration, Set/GetPosition, Set/GetVelocity, MultiUpdate, Update

SetKaff *(Servo products only)* buffered 93h
GetKaff *(Servo products only)* 94h

Syntax

SetKaff *axis Kaff*
GetKaff *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
Kaff	unsigned 16 bit	0 *to* 2^{15}-1	unity	-

Packet structure

SetKaff

	0	*axis*	**93**h
	15 12	11 8	7 0

Data

write	*Kaff*
	15 0

GetKaff

	0	*axis*	**94**h
	15 12	11 8	7 0

Data

read	*Kaff*
	15 0

Description

SetKaff sets the acceleration feedforward gain of the digital servo filter for the specified *axis*.

GetKaff reads the current value of the acceleration feedforward gain.

Restrictions

SetKaff is a buffered command. . The value set using this command will not take effect until the next Update or MultiUpdate instruction.

This command is not valid on the MC2400 and MC2500.

see

Set/GetKd, Set/GetKi, Set/GetKout, Set/GetKp, Set/GetKvff, MultiUpdate, Update

SetKd *(Servo products only)* buffered 27h
GetKd *(Servo products only)* 52h

Syntax

SetKd *axis Kd*
GetKd *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	***Range***	***Scaling***	***Units***
Kd	unsigned 16 bit	0 *to* 2^{15}-1	unity	-

Packet structure

SetKd

0	*axis*	**27**h
15 12	11 8	7 0

Data

write	*Kd*
	15 0

GetKd

0	*axis*	**52**h
15 12	11 8	7 0

Data

read	*Kd*
	15 0

Description

SetKd sets the derivative gain of the digital servo filter for the specified axis.

GetKd reads the current value of the derivative gain.

Restrictions

SetKd is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.

This command is not valid on the MC2400 and MC2500.

see

Set/GetKaff, Set/GetKi, Set/GetKout, Set/GetKp, Set/GetKvff, MultiUpdate, Update

SetKi *(Servo products only)* buffered 26h
GetKi *(Servo products only)* 51h

Syntax

SetKi *axis Ki*
GetKi *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
Ki	unsigned 16 bit	0 *to* 2^{15}-1	unity	-

Packet structure

SetKi

15–12	11–8	7–0
0	*axis*	**26**h

Data

	15–0
write	*Ki*

GetKi

15–12	11–8	7–0
0	*axis*	**51**h

Data

	15–0
read	*Ki*

Description

SetKi sets the integral gain of the digital servo filter for the specified *axis*.

GetKi reads the current value of the integral gain.

Restrictions

This is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.

This command is not valid on the MC2400 and MC2500.

see

Set/GetKaff, Set/GetKd, Set/GetKout, Set/GetKp, Set/GetKvff, MultiUpdate, Update

SetKout *(Servo products only)* 9Eh
GetKout *(Servo products only)* 9Fh

Syntax

SetKout *axis Kout*
GetKout *axis*

Arguments

Name	*Instance*	*Encoding*		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	*Range*	*Scaling*	*Units*
Kout	unsigned 16 bit	0 *to* 2^{16}-1	100/2^{16}	% output

Packet structure

SetKout

0	*axis*	**9E**h
15 12	11 8	7 0

Data

write	*Kout*
	15 0

GetKout

0	*axis*	**9F**h
15 12	11 8	7 0

Data

read	*Kout*
	15 0

Description

SetKout sets the output scale factor of the digital servo filter for the specified axis. The default value of Kout is 65535.

GetKout reads the current value of the output scale factor.

Example:

To set the output scaling of the servo filter to half, set the Kout register to 32767.

Restrictions

This command is NOT buffered. It will take affect immediately after it is sent.

This command is not valid on the MC2400 and MC2500.

see

Set/GetKaff, Set/GetKd, Set/GetKi, Set/GetKp, Set/GetKvff

SetKp *(Servo products only)* buffered 25h
GetKp *(Servo products only)* 50h

Syntax

SetKp *axis Kp*
GetKp *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
Kp	unsigned 16 bit	0 *to* 2^{15}-1	unity	-

Packet structure

SetKp

15–12	11–8	7–0
0	*axis*	**25**h

Data

	15–0
write	*Kp*

GetKp

15–12	11–8	7–0
0	*axis*	**50**h

Data

	15–0
read	*Kp*

Description

SetKp sets the proportional gain of the digital servo filter for the specified *axis*.

GetKp reads the current value of the proportional gain.

Restrictions

SetKp is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.

This command is not valid on the MC2400 and MC2500.

see

Set/GetKaff, Set/GetKd, Set/GetKi, Set/GetKout, Set/GetKvff, MultiUpdate, Update

SetKvff *(Servo products only)* buffered 2Bh
GetKvff *(Servo products only)* 54h

Syntax

SetKvff *axis Kvff*
GetKvff *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
Kvff	unsigned 16 bit	0 *to* 2^{15}-1	unity	-

Packet structure

SetKvff

	0	*axis*	**2B**h
	15 12	11 8	7 0

Data

write	*Kvff*
	15 0

GetKvff

	0	*axis*	**54**h
	15 12	11 8	7 0

Data

read	*Kvff*
	15 0

Description

SetKvff sets the velocity feedforward gain of the digital servo filter for the specified *axis*.

GetKvff reads the current value of the velocity feedforward gain.

Restrictions

SetKvff is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.

This command is not valid on the MC2400 and MC2500.

see

Set/GetKaff, Set/GetKd, Set/GetKi, Set/GetKout, Set/GetKp, MultiUpdate, Update

SetLimitSwitchMode 80h
GetLimitSwitchMode 81h

Syntax

SetLimitSwitchMode *axis mode*
GetLimitSwitchMode *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	off	0
	on	1

Packet structure

SetLimitSwitchMode

15 – 12	11 – 8	7 – 0
0	*axis*	**80**h

Data

	15 – 1	0
write	0	mode

GetLimitSwitchMode

15 – 12	11 – 8	7 – 0
0	*axis*	**81**h

Data

	15 – 1	0
read		mode

Description

SetLimitSwitchMode enables (On) or disables (Off) limit-switch sensing for the specified *axis*. When the mode is enabled, the axis will cause the corresponding limit-switch bits in the Event Status register and Activity Status register to be set when it enters either the positive or negative limit switches and the axis will be immediately stopped. When it is disabled these bits are not set, regardless of whether the axis is in a limit switch or not.

GetLimitSwitchMode returns the code for the current state of the limit-sensing mode.

Restrictions

see GetActivityStatus, GetEventStatus

SetMotionCompleteMode EBh
GetMotionCompleteMode ECh

Syntax

SetMotionCompleteMode *axis mode*
GetMotionCompleteMode *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	commanded	0
	actual	1

Packet structure

SetMotionCompleteMode

15–12	11–8	7–0
0	*axis*	**EB**h

Data

	15–1	0
write	0	*mode*

GetMotionCompleteMode

15–12	11–8	7–0
0	*axis*	**EC**h

Data

	15–1	0
read		*mode*

Description

SetMotionCompleteMode establishes the source for the comparison which determines the motion-complete status for the specified *axis*. When set to commanded mode the motion is considered complete when the profile velocity reaches zero and no further motion will occur without an additional host command. This mode is unaffected by the actual encoder location.

When set to actual mode the motion complete bit will be set when the above condition is true AND the actual encoder position has been within the Settle Window (SetSettleWindow command) for the number of servo loops specified by the SetSettleTime command. The settle "timer" is started at zero at the end of the trajectory profile motion so at a minimum a delay of SettleTime cycles will occur after the trajectory profile motion is complete.

GetMotionCompleteMode returns the current motion-complete mode.

Restrictions

see Set/GetSettleTime, Set/GetSettleWindow

SetMotorBias *(Servo products only)* 0Fh
GetMotorBias *(Servo products only)* 2Dh

Syntax

SetMotorBias *axis bias*
GetMotorBias *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	***Range***	***Scaling***	***Units***
bias	signed 16 bit	-2^{15} *to* $2^{15}-1$	$100/2^{15}$	% output

Packet structure

SetMotorBias

0	*axis*	**0F**h
15 12	11 8	7 0

Data

write	*bias*
	15 0

GetMotorBias

0	*axis*	**2D**h
15 12	11 8	7 0

Data

read	*bias*
	15 0

Description

SetMotorBias sets the bias voltage of the digital servo filter for the specified axis.

GetMotorBias reads the current bias voltage of the digital servo filter.

Scaling example:

If it is desired that a motor bias value of -2.5 % of full scale be placed on the servo filter output than this register should be loaded with a value of -2.5*32,768/100 = -819 (decimal). This corresponds to a loaded hexadecimal value of 0FCCDh.

Restrictions

This command is not valid on the MC2400 and MC2500.

see

Set/GetMotorCommand, Set/GetMotorLimit

SetMotorCommand buffered 77h
GetMotorCommand 69h

Syntax

SetMotorCommand *axis value*
GetMotorCommand *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
value	signed 16 bit	-2^{15} *to* $2^{15}-1$	$100/2^{15}$	% output

Packet structure

SetMotorCommand

	0	*axis*	77h
	15 12	11 8	7 0

Data

write	*value*
	15 0

GetMotorCommand

	0	*axis*	69h
	15 12	11 8	7 0

Data

read	*value*
	15 0

Description

SetMotorCommand loads the motor-command buffer register of the specified *axis*. For the MC2400 series, this command is used to control the magnitude of the output waveform.

GetMotorCommand reads the contents of the motor-command buffer register.

Scaling example:

If it is desired that a motor command value of 13.7 % of full scale be output to the motor than 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 is valid only when the motor is "off" for the MC2100 and MC2300 series.

SetMotorCommand is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.

This command is not available on the MC2500 series.

see

Set/GetMotorBias, Set/GetMotorLimit, Set/GetMotorMode, MultiUpdate, Update

SetMotorLimit *(Servo products only)* 06h
GetMotorLimit *(Servo products only)* 07h

Syntax

SetMotorLimit *axis limit*
GetMotorLimit *axis limit*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
limit	unsigned 16 bit	0 *to* 2^{15}-1	100/2^{15}	% output

Packet structure

SetMotorLimit

0	*axis*	**06**h
15 12	11 8	7 0

Data

write	*limit*
	15 0

GetMotorLimit

0	*axis*	**07**h
15 12	11 8	7 0

Data

read	*limit*
	15 0

Description

SetMotorLimit sets the maximum value for the motor output command allowed by the digital servo filter of the specified *axis*. 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 ouput value should be 1,100 the actual output value would be 1,000. Conversely 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 when it is known that a motor command exceeding a certain value will cause damage.

GetMotorLimit reads the current motor limit value.

Scaling example:

If it is desired that a motor limit of 75 % of full scale be established than 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 ouput when in closed loop mode. When the chipset is in open loop mode this command has no affect.

This command is not valid on the MC2400 and MC2500.

see

Set/GetMotorBias, Set/GetMotorCommand

SetMotorMode DCh
GetMotorMode DDh

Syntax

SetMotorMode *axis mode*
GetMotorMode *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	Off	0
	On	1

Packet structure

SetMotorMode

0	*axis*	**DC**h
15 12	11 8	7 0

Data

write	0	mode
	15 1	0

GetMotorMode

0	*axis*	**DD**h
15 12	11 8	7 0

Data

read		mode
	15 1	0

Description

SetMotorMode determines the mode of motor operation. When set to On, several events take place. For servo products, the axis is placed in *closed-loop* mode, and is controlled by the output of the servo filter. On the MC2400 series and MC2500 series, the trajectory generator controls the motor output. For all products, when the encoder source (Set/GetEncoderSource) is set to incremental or parallel, the position error is cleared; equivalent to a ClearPositionError command.

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. Setting the motor mode to Off also resets the trajectory generator, bringing any active motion to an abrupt stop. In additiona, the maximum velocity (Set/GetVelocity) is set to zero. On the MC2400 series and MC2500 series the step generator is switched off when the motor mode is set to Off.

GetMotorMode retusns the current motor mode.

Restrictions

see GetActivityStatus, Set/GetMotorCommand

SetNumberPhases *(MC2300, MC2400 and MC2800 only)* 85h
GetNumberPhases *(MC2300, MC2400 and MC2800 only)* 86h

Syntax

SetNumberPhases *axis phases*
GetNumberPhases *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
phases	1Phase	1
	2Phases	2
	3Phases	3

Packet structure

SetNumberPhases

	0	*axis*	**85**h
	15 12	11 8	7 0

Data

write	0	*phases*
	15 2	0

GetNumberPhases

	0	*axis*	**86**h
	15 12	11 8	7 0

Data

read		*phases*
	15 2	0

Description

SetNumberPhases establishes the number of phases, 1, 2 or 3, for commutation of the specified *axis*.

GetNumberPhases returns the number of phases set for the *axis*.

Restrictions

In PWM Sign/Magnitude output mode, the number of phases can be set to 1 or 2.

In PWM 5050 output mode, the number of phases can be set to 1,2 or 3.

For MC2300 & MC2400, the number of phases cannot be set to 1 (an "invalid parameter" error occurs).

see

GetPhaseCommand, InitializePhase, Set/GetPhase Set/GetOutputMode commands

SetOutputMode GetOutputMode

E0h
6Eh

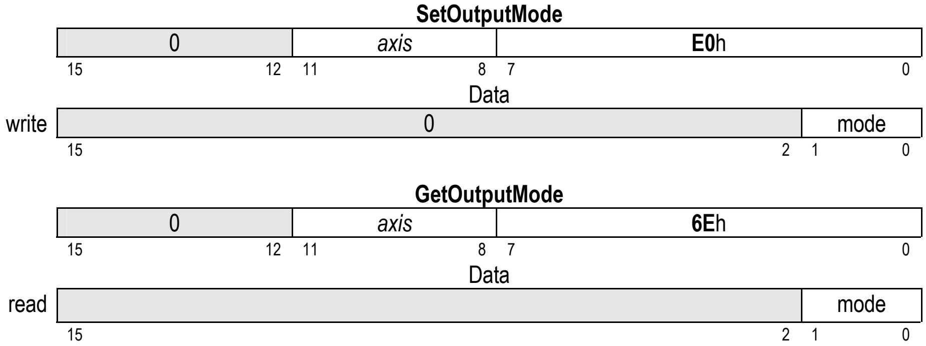
Syntax

SetOutputMode *axis mode*
GetOutputMode *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	DAC	0
	PWMSignMagnitude	1
	PWM5050Magnitude	2

Packet structure



Description

SetOutputMode determines the form of the motor output signal of the specified *axis*.

GetOutputMode returns the code for the current motor output mode.

Restrictions

This command is not available on the MC2500.

If the number of phases is set to 3, PWM Sign/Magnitude output mode is not available.

see

SetPhaseAngle *(MC2300 and MC2800 only)* 84h
GetPhaseAngle *(MC2300 and MC2800 only)* 2Ch

Syntax

SetPhaseAngle *axis angle*
GetPhaseAngle *axis*

Arguments

Name	*Instance*	*encoding*		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	*Range*	*Scaling*	*Units*
angle	unsigned integer	0 *to* 2^{15}-1	unity	counts

Packet structure

SetPhaseAngle

0	*axis*	**84**h
15 12	11 8	7 0

Data

write	*angle*
	15 0

GetPhaseAngle

0	*axis*	**2C**h
15 12	11 8	7 0

Data

read	*angle*
	15 0

Description

SetPhaseAngle sets the instantaneous commutation angle for the specified *axis*.

GetPhaseAngle returns the value of the current 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 current phase angle position.

Restrictions

The specified angle must not exceed the number of counts per electrical cycle set by the SetPhaseCounts command.

see

GetPhaseCommand, InitializePhase, Set/GetNumberPhases

SetPhaseCorrectionMode *(MC2300 and MC2800 only)* E8h
GetPhaseCorrectionMode *(MC2300 and MC2800 only)* E9h

Syntax

SetPhaseCorrectionMode *axis mode*
GetPhaseCorrectionMode *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	Disabled	0
	Enabled	1

Packet structure

SetPhaseCorrectionMode

	0	*axis*	**E8**h
	15 12	11 8	7 0

Data

write	0	mode
	15 1	0

GetPhaseCorrectionMode

	0	*axis*	**E9**h
	15 12	11 8	7 0

Data

read		mode
	15 1	0

Description

SetPhaseCorrectionMode sets the phase correction mode for the specified *axis* to either 0 (disabled) or 1(enabled). When phase correction is enabled, the encoder index signal is used to update the commutation phase angle 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.

GetPhaseCorrectionMode returns the current phase correction mode.

Restrictions

see

GetPhaseCommand, InitializePhase, Set/GetNumberPhases, Set/GetPhaseCounts

SetPhaseCounts *(MC2300, MC2400 and MC2800 only)* 75h
GetPhaseCounts *(MC2300, MC2400 and MC2800 only)* 7Dh

Syntax

SetPhaseCounts *axis counts*
GetPhaseCounts *axis*

Arguments

Name	*Instance*	*encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	*Range*	*Scaling*	*Units*
counts	unsigned 16 bit	0 *to* 2^{15}-1	unity	counts

Packet structure

SetPhaseCounts

	0	*axis*	**75h**
	15 12	11 8	7 0

Data

write	*count*
	15 0

GetPhaseCounts

	0	*axis*	**7Dh**
	15 12	11 8	7 0

Data

read	*count*
	15 0

Description

SetPhaseCounts sets the number of encoder count per electrical phase of the motor. If this value is not an integer then the closest integer value should be used, and phase correction mode should be enabled (See SetPhaseCorrectionMode command). The number of electrical cycles is equal to 1/2 the number of motor poles.

GetPhaseCounts returns the number of counts per electrical cycle.

Restrictions

For MC2400:

The number of microsteps per full step is set using the command SetPhaseCounts. The parameter used for this command represents the number of microsteps per electrical cycle (4 times the desired number of microsteps). So for example, to set 64 microsteps per full step, the command SetPhaseCounts 256 should be used. The maximum number of microsteps that can be generated per full step is 256, giving a maximum parameter for this command of 1024.

see

GetPhaseCommand, InitializePhase, Set/GetNumberPhases

SetPhaseInitializeMode *(MC2300 and MC2800 only)* E4h
GetPhaseInitializeMode *(MC2300 and MC2800 only)* E5h

Syntax

SetPhaseInitializeMode *axis mode*
GetPhaseInitializeMode *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	Algorithmic	0
	Hall-based	1

Packet structure

SetPhaseInitializeMode

0	*axis*	**E4**h
15 12	11 8	7 0

Data

	0	mode
write	15 1	0

GetPhaseInitializeMode

0	*axis*	**E5**h
15 12	11 8	7 0

Data

		mode
read	15 1	0

Description

SetPhaseInitializeMode establishes the mode in which the specified axis is to be initialized for commutation. The options are Algorithmic and Hall-based. In algorithmic mode the chipset briefly stimulates the motor windings and sets the initial phasing based on the observed motor response. In Hall-based initialization mode the 3 Hall sensor signals are used to determine the motor phasing.

GetPhaseInitializeMode returns the current initialization mode.

Restrictions

Algorithmic mode should only be selected if it is known that the axis is free to move in both directions, and that a brief uncontrolled move can be tolerated by the motor, mechanism, and load.

see

GetPhaseCommand, InitializePhase, Set/GetNumberPhases

SetPhaseInitializeTime *(MC2300 and MC2800 only)* 72h
GetPhaseInitializeTime *(MC2300 and MC2800 only)* 7Ch

Syntax

SetPhaseInitializeTime *axis time*
GetPhaseInitializeTime *axis*

Arguments

Name	***Instance***	***encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
time	unsigned 16 bit	0 *to* 2^{15}-1	unity	cycles

Packet structure

SetPhaseInitializeTime

0	*axis*	**72**h
15 … 12	11 … 8	7 … 0

Data

write	*time*
	15 … 0

GetPhaseInitializeTime

0	*axis*	**7C**h
15 … 12	11 … 8	7 … 0

Data

read	*time*
	15 … 0

Description

SetPhaseInitializeTime sets the time value (in cycles) to be used during the algorithmic phase initialization procedure. This value determines the duration of each of the four segments in the phase initialization algorithm. See the User's guide for more information on algorithmic initialization.

GetPhaseInitializeTime returns the current phase initialization time.

Restrictions

see

GetPhaseCommand, InitializePhase, Set/GetNumberPhases

SetPhaseOffset *(MC2300 and MC2800 only)* 76h
GetPhaseOffset *(MC2300 and MC2800 only)* 7Bh

Syntax

SetPhaseOffset *axis offset*
GetPhaseOffset *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
offset	unsigned 16 bit	0 *to* 2^{15}-1	unity	counts

Packet structure

SetPhaseOffset

0	*axis*	**76**h
15 12	11 8	7 0

Data

write	*offset*
	15 0

GetPhaseOffset

0	*axis*	**7B**h
15 12	11 8	7 0

Data

read	*offset*
	15 0

Description

SetPhaseOffset sets the offset from the index mark of the specified axis to the maximum output value of phase A. This command will have no immediate effect on the commutation angle but will have an affect once the index pulse is encountered.

GetPhaseOffset returns the current value of the phase offset.

To convert counts to a phase angle in degrees, divide by the number of encoder counts per electrical cycles and multiply by 360. For example if a value of 500 is specified using SetPhaseOffset 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 at the index mark.

Restrictions

see GetPhaseCommand, InitializePhase, Set/GetNumberPhases

SetPhasePrescale *(MC2300 and MC2800 only)* E6h
GetPhasePrescale *(MC2300 and MC2800 only)* E7h

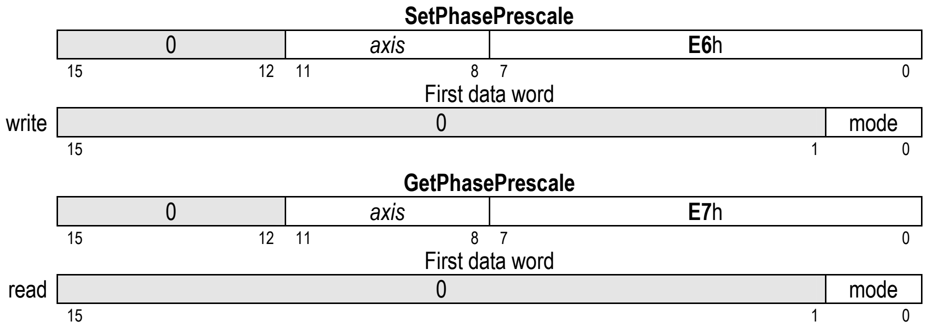
Syntax

SetPhasePrescale *axis scale*
GetPhasePrescale *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	Off	0
	64	1
	128	2
	256	3

Packet structure



Description

SetPhasePrescale On causes the number of encoder counts to be scaled by a factor of $\frac{1}{64}$ before being used to calculate a commutation angle for the specified axis. When operated in the prescale mode the chipset can commutate motors with a high number of counts per electrical cycle, such as motors with very high accuracy encoders.

SetPhasePrescale Off removes the scale factor.

GetPhasePrescale returns the current scaling mode.

Restrictions

see GetPhaseCommand, InitializePhase, Set/GetNumberPhases

SetPosition buffered 10h
GetPosition 4Ah

Syntax

SetPosition *axis position*
GetPosition *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
position	signed 32 bit	-2^{31} *to* $2^{31}-1$	unity	counts

Packet structure

SetPosition

	15–12	11–8	7–0
	0	*axis*	**10**h

First data word

	31–16
write	*position* (high-order part)

Second data word

	15–0
write	*position* (low-order part)

GetPosition

	15–12	11–8	7–0
	0	*axis*	**4A**h

First data word

	31–16
read	*position* (high-order part)

Second data word

	15–0
read	*position* (low-order part)

Description

SetPosition specifies the trajectory destination of the specified *axis*. It is used in the Trapezoidal and S-curve profile modes.

GetPosition reads the contents of the buffered position register.

Restrictions

SetPosition is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.

see

Set/GetAcceleration, Set/GetDeceleration, Set/GetJerk, Set/GetVelocity, GetPositionError, Set/GetPositionErrorLimit, MultiUpdate, Update

SetPositionErrorLimit 97h
GetPositionErrorLimit 98h

Syntax

SetPositionErrorLimit *axis limit*
GetPositionErrorLimit *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	*Range*	*Scaling*	*Units*
limit	unsigned 32 bit	0 *to* 2^{31}-1	unity	counts

Packet structure

SetPositionErrorLimit

0	*axis*	**97**h
15 12	11 8	7 0

First data word

write	*limit* (high-order part)
	31 16

Second data word

write	*limit* (low-order part)
	15 0

GetPositionErrorLimit

0	*axis*	**98**h
15 12	11 8	7 0

First data word

read	*limit* (high-order part)
	31 16

Second data word

read	*limit* (low-order part)
	15 0

Description

SetPositionErrorLimit sets the absolute value of the maximum position error allowable by the chipset for the specified *axis*. If the position error exceeds this limit, 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.

GetPositionErrorLimit returns the current position error limit value.

Restrictions

see GetPositionError, GetActualPosition, Set/GetPosition

SetProfileMode buffered A0h
GetProfileMode A1h

Syntax

SetProfileMode *axis profile*
GetProfileMode *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
profile	Trapezoidal	0
	Velocity contouring	1
	S-curve	2
	Electronic gear	3
	External	4

Packet structure

SetProfileMode

	0	*axis*	**A0**h
	15 12	11 8	7 0

Data

write	0	profile
	15 3	2 0

GetProfileMode

	0	*axis*	**A1**h
	15 12	11 8	7 0

Data

read		profile
	15 3	2 0

Description

SetProfileMode sets the profile mode, selecting Trapezoidal, Velocity Contouring, S-curve, Electronic gear or External for the specified *axis*.

GetProfileMode returns the contents of the buffered profile-mode register for the specified axis.

Restrictions

SetProfileMode is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.

see

Set/GetGearMaster, Set/GetGearRatio, Set/GetBufferFunction, MultiUpdate, Update

SetSampleTime 38h
GetSampleTime 61h

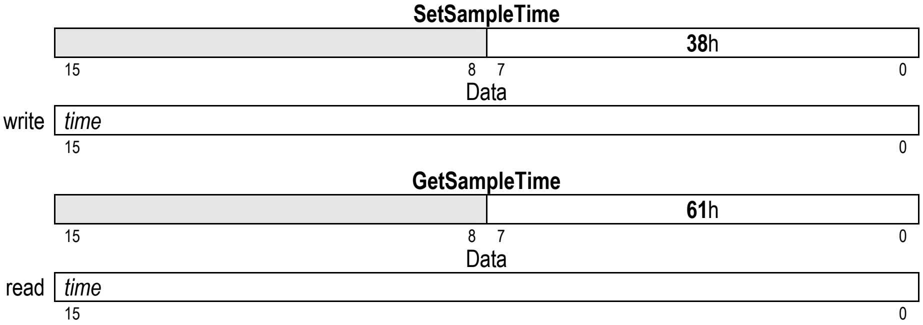
Syntax

SetSampleTime *time*
GetSampleTime

Arguments

Name	*Type*	*Range*	*Scaling*	*Units*
time	unsigned 16 bit	1 *to* 2^{15}-1	unity	µsec/cycle

Packet structure



Description

SetSampleTime sets the cycle time for the chipset. This is the time between servo loop updates and trajectory calculations. The value is expressed in microseconds. Only certain values are allowed as follows:

Product	**Allowed values**
MC2100 series	multiples of 51.2 and at least 102 µsec per enabled axis
MC2300 series	multiples of 51.2 and at least 154 µsec per enabled axis
MC2400 series	multiples of 51.2 and at least 154 µsec per enabled axis

GetSampleTime returns the current sample time value.

Result of invalid sample time arguments

The PMD device does not return an error when an invalid sample time is attempted. If the value is less than the required minimum (based on the number of enabled axes), the sample time will be set to the minimum value. If the value is not an increment of 51.2 µsec then the sample time will be set to the closest valid increment to that value.

Restrictions

This command affects the cycle time for all axes.

see

SetSerialPortMode 8Bh
GetSerialPortMode 8Ch

Syntax

SetSerialPort *mask*
GetSerialPort

Arguments

Name	***Instance***	***Encoding***
mask		*see below*

Packet structure

SetSerialPortMode

0	8Bh
15 … 8	7 … 0

Data

	multi-drop address	0	*protocol*		*parity*	*transmission rate*
write	15 … 11	10 … 9	8 … 7	6	5 … 4	3 … 0
				↳ *stop bits*		

GetSerialPortMode

0	8Ch
15 … 8	7 … 0

Data

	multi-drop address		*protocol*		*parity*	*transmission rate*
read	15 … 11	10 … 9	8 … 7	6	5 … 4	3 … 0
				↳ *stop bits*		

Description

SetSerialPortMode sets the configuration for the asynchronous serial port.

Note: It is recommended that two stop bits be used for baud rates greater than 19200bps.

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 2400 9600 19200 57600 115200 250000 416667	0 1 2 3 4 5 6 7
4-5	parity	none odd even	0 1 2
6	stop bits	1 2	0 1
7-8	protocol	Point-to-point Multi-drop using address bit Multi-drop using idle-line detection	0 2 3
11-15	multi-drop address	Address 0 Address 1 … Address 31	0 1 … 31

Restrictions

see Set/GetDiagnosticPortMode

SetSettleTime AAh
GetSettleTime ABh

Syntax

SetSettleTime *axis time*
GetSettleTime *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

	Type	*Range*	*Scaling*	*Units*
time	unsigned 16 bit	0 *to* 2^{15}-1	unity	cycles

Packet structure

SetSettleTime

0	*axis*	**AA**h
15 12	11 8	7 0

Data

write	*time*
	15 0

GetSettleTime

0	*axis*	**AB**h
15 12	11 8	7 0

Data

read	*time*
	15 0

Description

SetSettleTime sets the time, in number of cycles, that the specified *axis* must remain within the settle window before the axis-settled indicator (in the activity status register) is set.

GetSettleTime returns the current settle time for the specified *axis*.

Restrictions

see Set/GetMotionCompleteMode, Set/GetSettleWindow, GetActivityStatus

SetSettleWindow BCh
GetSettleWindow BDh

Syntax

SetSettleWindow *axis window*
GetSettleWindow *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
window	unsigned 16 bit	0 *to* 2^{15}-1	unity	counts

Packet structure

SetSettleWindow

0	*axis*	**BC**h
15 12	11 8	7 0

Data

write	*window*
	15 0

GetSettleWindow

0	*axis*	**BD**h
15 12	11 8	7 0

Data

read	*window*
	15 0

Description

SetSettleWindow sets the position range within which the specified *axis* must remain for the duration specified by SetSettleTime before the axis-settled indicator (in the activity status register) is set.

GetSettleWindow returns the current value of the settle window.

Restrictions

see Set/GetMotionCompleteMode, Set/GetSettleTime, GetActivityStatus

SetSignalSense A2h
GetSignalSense A3h

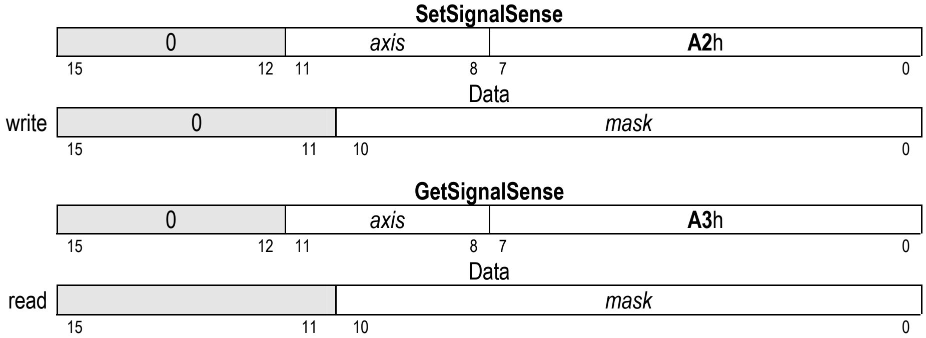
Syntax

SetSignalSense *axis mask*
GetSignalSense *axis*

Arguments

Name	***Instance***	***Encoding***	
axis	Axis1	0	
	Axis2	1	
	Axis3	2	
	Axis4	3	
	Indicator		***Bit Number***
mask	Encoder A	0001h	0
	Encoder B	0002h	1
	Encoder Index	0004h	2
	Encoder Home	0008h	3
	Positive limit	0010h	4
	Negative limit	0020h	5
	AxisIn	0040h	6
	Hall A	0080h	7
	Hall B	0100h	8
	Hall C	0200h	9
	AxisOut	0400h	10
	StepOutput	0800h	11
	MotorOutput	1000h	12
	reserved		13 - 15

Packet structure



Description

SetSignalSense establishes the sense of the signals connected to the Signal Sense register by using a bitwise mask that corresponds to the bits of the Signal Status register, for the specified *axis*.

For each sense bit that is 0, the input is active low, or not inverted.

For each sense bit that is 1, the input is active high, or inverted.

Inverting the MotorOutput has the effect of reversing the direction of motion when a positive or negative motor command is given.

When the StepOutput bit is set to 1 a step will be generated by the MC2500 with a LOW to HIGH transition on the Pulse signal. The default condition is a HIGH to LOW transition. Refer to the User's Guide MC2500 section for more information.

GetSignalSense returns the current signal sense mask.

Restrictions

Inverting ther encoder A,B, or index may prevent the index capture mechanism from operating correctly. Refer to the Navigator Technical Specifications for the index capture electrical requirements.

see

GetSignalStatus

SetStartMode CCh
GetStartMode CDh

Syntax

SetStartMode *axis mode*
GetStartMode *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	None	0
	Immediate	1
	Update	2

Packet structure

SetStopMode

	0	*axis*	**CC**h
	15 12	11 8	7 0

Data

write	0	*mode*
	15 2	1 0

GetStopMode

	0	*axis*	**CD**h
	15 12	11 8	7 0

Data

read		*mode*
	15 2	1 0

Description

SetStartMode starts motion on the specified axis when that axis is external profile mode. The available start modes are Immediate, which instantly starts the external profile on the specified axis, Update, which will start the profile when the next update command is issued, or None which can be used to turn off a previously issued SetStartMode Update command.

Note: After the profile has started, the start mode will reset to the None condition. In other words if the command SetStartMode Update is followed by an Update command and then by a GetStartMode command, the retrieved start mode will be None.

GetStopMode returns the start mode set using SetStartMode.

Restrictions

This command should only be used in external profile mode. It has no effect when the chip is in other profile modes.

This command should not be executed while an external profile is executing.

see

Update, SetProfileMode

SetStartVelocity *(MC2400 and MC2500 only)* buffered 6Ah
GetStartVelocity *(MC2400 and MC2500 only)* 6Bh

Syntax

SetStartVelocity *axis velocity*
GetStartVelocity *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
velocity	unsigned 32 bit	0 *to* 2^{31}-1	1/2^{16}	counts/cycle

Packet structure

SetStartVelocity

15 – 12	11 – 8	7 – 0
0	*axis*	**6A**h

First data word

	31 – 16
write	*velocity* (high-order part)

Second data word

	15 – 0
write	*velocity* (low-order part)

GetStartVelocity

15 – 12	11 – 8	7 – 0
0	*axis*	**6B**h

First data word

	31 – 16
read	*velocity* (high-order part)

Second data word

	15 – 0
read	*velocity* (low-order part)

Description

SetStartVelocity loads the starting velocity buffer register for the specified axis.

GetStartVelocity reads the starting velocity buffer register.

Scaling example: To load a starting 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 (GetStartingVelocity) must correspondingly be divided by 65,536 to convert to units of counts/cycle.

Restrictions

SetStartVelocity has no effect when the chip is in S-curve profile mode.

SetVelocity is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.

see

Set/GetAcceleration, Set/GetDeceleration, Set/GetPosition

SetStepRange *(MC2500 only)* CFh
GetStepRange *(MC2500 only)* CEh

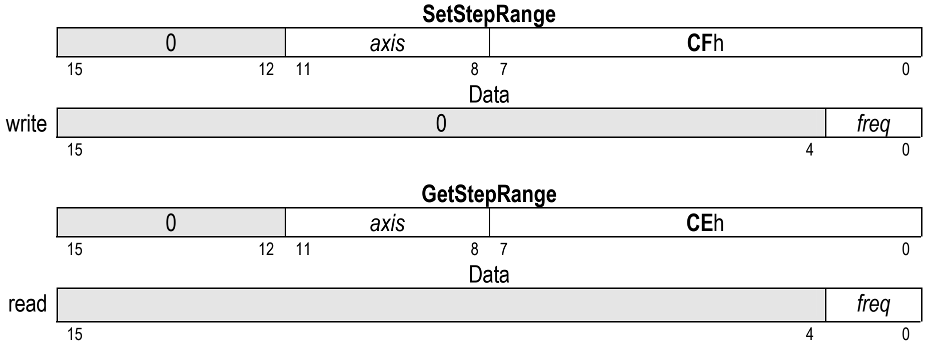
Syntax

SetStepRange *axis frequency*
GetStepRange *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
frequency	5 MHz	1
	625 kHz	4
	156.25 kHz	6
	39.062 kHz	8

Packet structure



Description

SetStepRange set the maximum pulse rate frequency for the specified *axis*. For example, if the desired maximum pulse rate is 200,000 pulses/second, the command SetStepRange 4 should be issued.

GetMaxStepRange returns the maximum pulse rate frequency for the specified *axis*.

Restrictions

This command is only available on the MC2500 series.

SetStopMode buffered D0h
GetStopMode D1h

Syntax

SetStopMode *axis mode*
GetStopMode *axis*

Arguments

Name	***Instance***	***Encoding***
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
mode	NoStop	0
	AbruptStop	1
	SmoothStop	2

Packet structure

SetStopMode

0	*axis*	**D0**h
15 12	11 8	7 0

Data

write	0	*mode*
	15 2	1 0

GetStopMode

0	*axis*	**D1**h
15 12	11 8	7 0

Data

read		*mode*
	15 2	1 0

Description

SetStopMode stops the specified axis. The available stop modes are AbruptStop, which instantly (without any deceleration phase) stops the axis, SmoothStop which uses the programmed deceleration value and profile shape for the current profile mode to stop the axis, or NoStop which is generally used to turn off a previously set stop command.

Note: After an Update a buffered stop command (SetStopMode command) will reset to the NoStop condition. In other words if the command SetStopMode is followed by an Update command and then by a GetStopMode command, the retrieved stop mode will be NoStop.

GetStopMode returns the stop mode set using SetStopMode.

Restrictions

SmoothStop mode is not available in the electronic-gearing profile.

SetStopMode is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.

see

MultiUpdate, Update

SetSynchronizationMode *(MC2xx3 only)* F2h
GetSynchronizationMode *(MC2xx3 only)* F3h

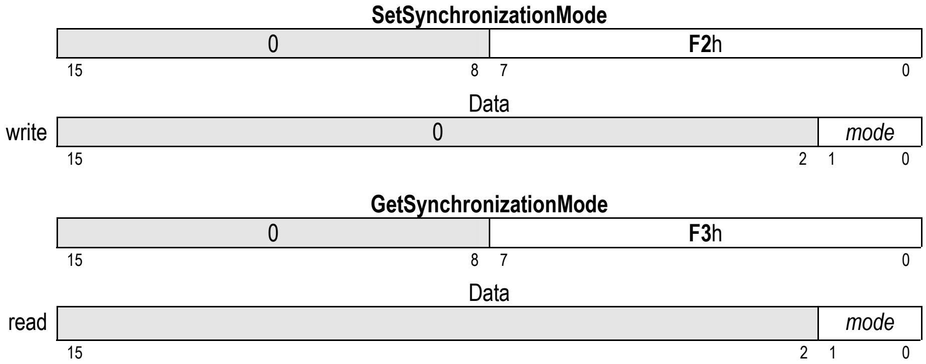
Syntax

SetSynchronizationMode *mode*
GetSynchronizationMode

Arguments

Name	***Instance***	***Encoding***
Mode	Disabled	0
	Master	1
	Slave	2

Packet structure



Description

SetSynchronizationMode sets the mode of the pin used for the synchcronization of the internal timer across multiple multiple PMD motion processors. In the disabled mode, the pin is configured as an input and is not used. In the master mode, the pin outputs a synchcronization pulse that can be used by slave nodes or other devices to synchronize with the internal chip cycle of the master node. In the slave mode, the pin is configured as an input and a pulse on the pin synchronizes the internal chip cycle.

GetEncoderSource returns the code for the current synchronization mode.

Restrictions

This command is only available on chipsets with the synchronization feature enabled.

see

Set/GetSampleTime, Set/GetBreakpoint, Set/GetBreakpointValue

SetTraceMode B0h
GetTraceMode B1h

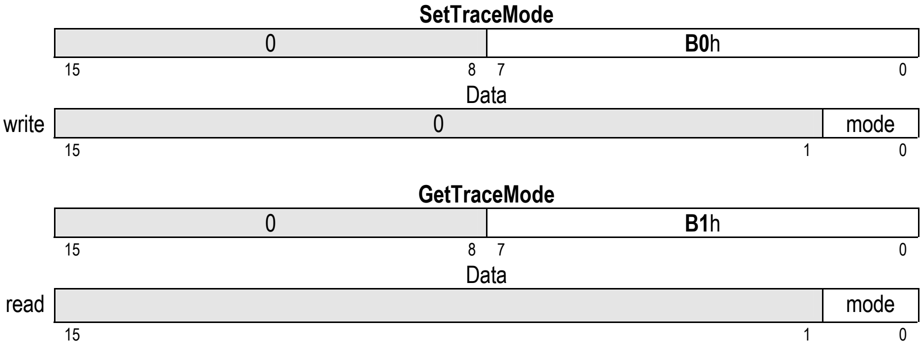
Syntax

SetTraceMode *mode*
GetTraceMode

Arguments

Name	*Instance*	*Encoding*
mode	OneTime	0
	RollingBuffer	1

Packet structure



Description

SetTraceMode sets the buffer usage for the next trace. In OneTime mode, the trace continues until the buffer is filled, then stops. In Rolling mode, the trace continues from the beginning of the buffer after the end is reached. Values stored when in the rolling mode are lost if they are not read before being overwritten by the wrapped data being traced and stored.

GetTraceMode returns the code for the current buffer mode.

Restrictions

see GetTraceStatus

SetTracePeriod B8h
GetTracePeriod B9h

Syntax

SetTracePeriod *period*
GetTracePeriod

Arguments

Name	***Type***	***Range***	***Scaling***	***Units***
period	unsigned 16 bit	1 *to* 2^{15}-1	unity	cycles

Packet structure

SetTracePeriod

	15 … 8	7 … 0
	0	**B8**h

Data

	15 … 0
write	*period*

GetTracePeriod

	15 … 8	7 … 0
	0	**B9**h

Data

	15 … 0
read	*period*

Description

SetTracePeriod sets the time period, expressed in number of cycles, between successive trace points.

GetTracePeriod returns the current trace period.

Restrictions

see Set/GetSampleTime, Set/GetTraceStart, Set/GetTraceStop

SetTraceStart B2h
GetTraceStart B3h

Syntax

SetTraceStart *triggerAxis condition triggerBit triggerState*
GetTraceStart

Arguments

Name	***Instance***	***Encoding***
triggerAxis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
	Description	
condition	Immediate	0
	Next update	1
	Event Status register bit	2
	Activity Status register bit	3
	Signal Status register bit	4
triggerBit	Status register bit	0 *to* 15
triggerState	Triggering state of the bit	0 (value = 0) 1 (value = 1)

Packet structure

SetTraceStart

15–8	7–0
0	**B2**h

Data

	15–13	12	11–8	7–4	3–0
write	0	state	*triggerBit*	*condition*	*triggerAxis*

GetTraceStart

15–8	7–0
0	**B3**h

Data

	15–13	12	11–8	7–4	3–0
read		state	*triggerBit*	*condition*	*triggerAxis*

Description

SetTraceStart sets the condition for starting the trace. The *Immediate* condition requires no axis to be specified and the trace will begin upon execution of this instruction. The other four conditions require an axis to be specified; and when the condition for that axis is attained, the trace will begin.

When a status register bit is the trigger, the bit number and state must be included in the argument. The trace is started when the indicated bit reaches the specified state (0 or 1).

Once a trace has started, the trace-start indicator is reset and the SetTraceStart instruction must be reentered before another trace can be started.

GetTraceStart returns the the current trace-start condition.

Examples:

If it is desired that the trace begin on the next Update for axis 3, then a "1" is set for the condition, a "2" is set for the axis number, and bit number and state can be loaded with zeroes since they are not used.

If it is desired that the trace begin when bit 7 of the Activity Status register for axis 2 goes to 0 then the trace start is loaded as follows: A "3" is loaded for condition, a "1" is loaded for axis number, a "7" is loaded for bit number, and a "0" is loaded for state.

The table below shows the corresponding value for combinations of *triggerBit* and *register*.

encoding of "triggerBit"	***register = event status***	***register = activity status***	***register = signal status***
0	Motion Complete	Phasing Initialized	Encoder A
1	Wrap-around	At maximum velocity	Encoder B
2	Breakpoint 1	Tracking	Encoder index
3	Position capture		Home
4	Motion error		Positive limit
5	In positive limit		Negative limit
6	In negative limit		AxisIn
7	Instruction error	Axis settled	Hall sensor 1
8		Motor on/off	Hall sensor 2
9		Position capture	Hall sensor 3
0Ah		In motion	
0Bh	Commutation error	In positive limit	
0Ch		In negative limit	
0Dh			
0Eh	Breakpoint 2		
0Fh			

Restrictions

see Set/GetBufferLength, GetTraceCount, Set/GetTraceMode, Set/GetTracePeriod, Set/GetTraceStop

SetTraceStop B4h
GetTraceStop B5h

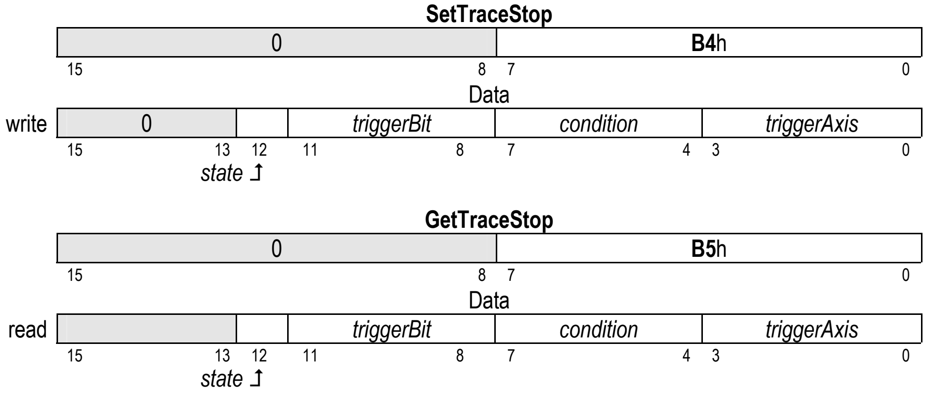
Syntax

SetTraceStop *triggerAxis condition triggerBit triggerState*
GetTraceStop

Arguments

Name	***Instance***	***Encoding***
triggerAxis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
	Description	
condition	Immediate	0
	Next update	1
	Event Status register bit	2
	Activity Status register bit	3
	Signal register bit	4
triggerBit	Status register bit	0 *to* 15
triggerState	Triggering state of the bit	0 (value = 0) 1 (value = 1)

Packet structure



Description

SetTraceStop sets the condition for stopping the trace. The *Immediate* condition requires no axis to be specified and the trace will stop upon execution of this instruction. The other four conditions require an axis to be specified; and when the condition for that axis is attained, the trace will stop.

When a status register bit is the trigger, the bit number and state must be included in the argument. The trace stops when the indicated bit reaches the specified state (0 or 1).

Once a trace has stopped, the trace-stop indicator is reset and the SetTraceStop instruction must be reentered before another trace can be stopped.

GetTraceStop returns the code for the current trace-stop condition.

Examples:

If it is desired that the trace stop on the next Update for axis 3, then a "1" is set for the condition, a "2" is set for the axis number, and bit number and state can be loaded with zeroes since they are not used.

If it is desired that the trace stop when bit 7 of the Activity status for axis 2 goes to 0 then the trace stop is loaded as follows: A "3" is loaded for condition, a "1" is loaded for axis number, a "7" is loaded for bit number, and a "0" is loaded for state.

The table below shows the corresponding value for combinations of *triggerBit* and *register*.

encoding of "triggerBit"	***register = event status***	***register = activity status***	***register = signal status***
0	Motion Complete	Phasing Initialized	Encoder A
1	Wrap-around	At maximum velocity	Encoder B
2	Breakpoint 1	Tracking	Encoder index
3	Position capture		Home
4	Motion error		Positive limit
5	In positive limit		Negative limit
6	In negative limit		AxisIn
7	Instruction error	Axis settled	Hall sensor 1
8		Motor on/off	Hall sensor 2
9		Position capture	Hall sensor 3
0Ah		In motion	
0Bh	Commutation error	In positive limit	
0Ch		In negative limit	
0Dh			
0Eh	Breakpoint 2		
0Fh			

Restrictions

see Set/GetTraceCount, Set/GetTraceStart, Set/GetTraceStatus

SetTraceVariable B6h
GetTraceVariable B7h

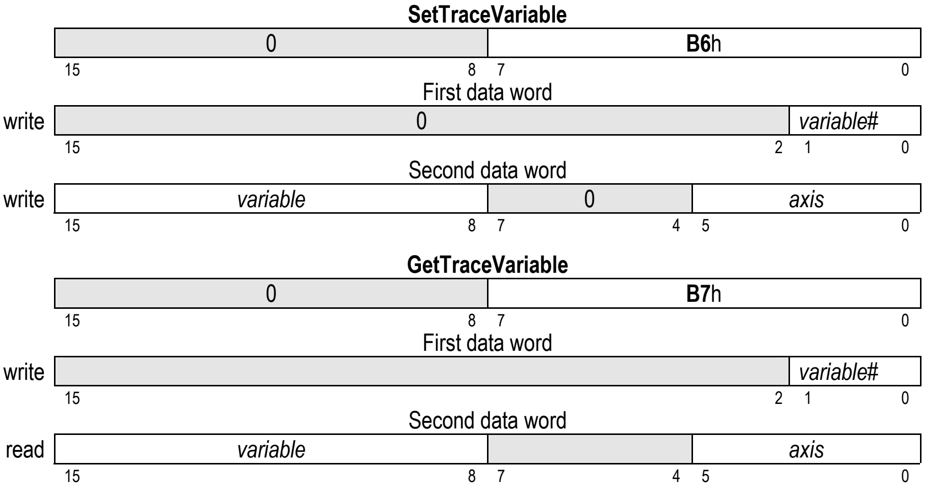
Syntax

SetTraceVariable *variableNumber traceAxis variable*
GetTraceVariable *variableNumber*

Arguments

Name	***Instance***	***Encoding***
variableNumber	Variable1	0
	Variable2	1
	Variable3	2
	Variable4	3
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3
variable	None (disable the variable)	0
	Position error (32 bits)	1
	Commanded position (32 bits)	2
	Commanded velocity (32 bits)	3
	Commanded acceleration (32 bits)	4
	Actual position (32 bits)	5
	Actual velocity (32 bits)	6
	Motor command (16 bits)	7
	Chipset time (32 bits)	8
	Capture register (32 bits)	9
	Integral (32 bits)	Ah
	Derivative (16 bits)	Bh
	Event Status register (16 bits)	Ch
	Activity Status register (16 bits)	Dh
	Signal Status register (16 bits)	Eh
	Phase angle (16 bits)	Fh
	Phase offset (16 bits)	10h
	Phase A (16 bits)	11h
	Phase B (16 bits)	12h
	Phase C (16 bits)	13h
	Analog input 1 (16 bits)	14h
	Analog input 2 (16 bits)	15h
	Analog input 3 (16 bits)	16h
	Analog input 4 (16 bits)	17h
	Analog input 5 (16 bits)	18h
	Analog input 6 (16 bits)	19h
	Analog input 7 (16 bits)	1Ah
	Analog input 8 (16 bits)	1Bh
	PID Servo Error	1Ch

Packet structure



Description

SetTraceVariable assigns the given variable to the specified *variableNumber* location in the trace buffer. The variable will always occupy a 32-bit buffer location. 16-bit values are sign extended to 32 bits. Up to four variables may be traced at one time. All combinations of axis numbers and trace variables are supported.

All variable assignments must be contiguous starting with *variableNumber* = 0.

GetTraceVariable returns the variable and axis of the specified *variableNumber*.

Example: To set up a 3 variable trace capturing the commanded acceleration for axis 1, the actual position for axis 1, and the event status word for axis 3 the following sequence of commands would be used. First a SetTraceVariable command with traceId of "0", axis of "0", and variable of "4" would be sent. Then a SetTraceVariable command with traceId of "1", axis of "0", and variable of "5" would be sent. Finally a SetTraceVariable command with traceId of "2", axis of "2" and variable of "0Ch" would be sent.

Restrictions

see Set/GetTrace commands

SetTrackingWindow A8h
GetTrackingWindow A9h

Syntax

SetTrackingWindow *axis window*
GetTrackingWindow *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
window	unsigned 16 bit	0 *to* 2^{15}-1	unity	counts

Packet structure

SetTrackingWindow

	0	*axis*	**A8**h
	15 12	11 8	7 0

Data

write	*window*	
	15	0

GetTrackingWindow

	0	*axis*	**A9**h
	15 12	11 8	7 0

Data

read	*window*	
	15	0

Description

SetTrackingWindow sets boundaries for the actual position of the specified axis. If the axis crosses the window boundary in either direction, the Tracking indicator (bit 2 of the activity Status register) is set to 0. When the axis returns to within the window, the tracking indicator is set to 1.

GetTrackingWindow returns the value of the current tracking window.

Restrictions

see GetActivityStatus, GetActualPosition

SetVelocity buffered 11h
GetVelocity 4Bh

Syntax

SetVelocity *axis velocity*
GetVelocity *axis*

Arguments

Name	***Instance***	***Encoding***		
axis	Axis1	0		
	Axis2	1		
	Axis3	2		
	Axis4	3		
	Type	***Range***	***Scaling***	***Units***
velocity	signed 32 bit	-2^{31} *to* $2^{31}-1$	$1/2^{16}$	counts/cycle

Packet structure

SetVelocity

	15–12	11–8	7–0
	0	*axis*	**11**h

First data word

	31–16
write	*velocity* (high-order part)

Second data word

	15–0
write	*velocity* (low-order part)

GetVelocity

	15–12	11–8	7–0
	0	*axis*	**4B**h

First data word

	31–16
read	*velocity* (high-order part)

Second data word

	15–0
read	*velocity* (low-order part)

Description

SetVelocity loads the Maximum Velocity buffer register for the specified *axis*.

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 correspondingly be divided by 65,536 to convert to units of counts/cycle.

Restrictions

SetVelocity may not be issued while an axis is in motion with the S-curve profile.

SetVelocity is not valid in Electronic Gearing profile mode.

The velocity must not be < 0 except in the Velocity-Contouring profile mode.

SetVelocity is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.

see

Set/GetAcceleration, Set/GetDeceleration, Set/GetJerk, Set/GetPosition, MultiUpdate, Update

Update 1Ah

Syntax Update *axis*

Arguments

Name	*Instance*	*Encoding*
axis	Axis1	0
	Axis2	1
	Axis3	2
	Axis4	3

Packet structure

Update

0	*axis*	**1A**h
15 12	11 8	7 0

Description

Update causes all buffered data parameters are copied into the corresponding run-time registers on the specified *axis*.

The following instruction is buffered: ClearPositionError.

The following trajectory parameters are buffered: Acceleration, Deceleration, GearRatio, Jerk, Position, ProfileMode, StartVelocity, Stop, and Velocity.

The following PID filter parameters are buffered: DerivativeTime, IntegrationLimit, Kaff, Kd, Ki, Kp, and Kvff.

The following Motor Command parameters is buffered: MotorCommand.

Restrictions

see MultiUpdate

WriteBuffer C8h

Syntax WriteBuffer *bufferID value*

Arguments

Name	*Type*	*Range*	*Scaling*	*Units*
bufferID	unsigned 16 bit	0 *to* 15	unity	-
value	signed 32 bit	-2^{31} *to* $2^{31}-1$	unity	-

Packet structure

WriteBuffer

	0	**C8**h
	15 8	7 0

First data word

write	0	*bufferID*
	15 4	3 0

Second data word

write	*value* (high-order part)
	31 16

Third data word

write	*value* (low-order part)
	15 0

Description

WriteBuffer writes the 32-bit *value* into the current location in the specified buffer. The current location is determined by adding the base address of the buffer (set by SetBufferStart), to the buffer's Write Index (set by SetBufferWriteIndex). After the contents have been read, the Write Index is incremented by 1; if the result is equal to the buffer length (set by SetBufferLength), the Index is reset to 0.

Some chipset operations automatically change the write index such as during a trace. See the User's Guide for more details.

Restrictions

see ReadBuffer, Set/GetBufferWriteIndex

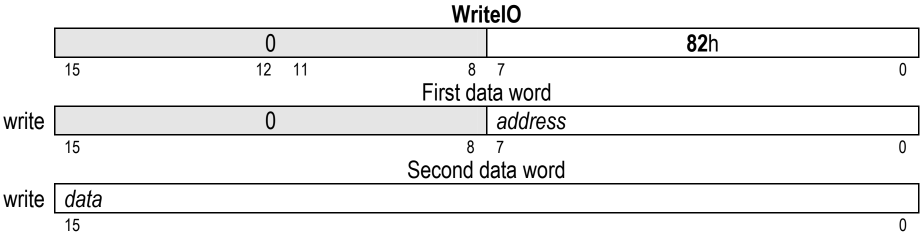
WriteIO 82h

Syntax WriteIO *address data*

Arguments

Name	*Type*	*Range*	*Scaling*	*Units*
address	unsigned 8 bit	0 *to* 255	unity	-
data	unsigned 16 bit	0 *to* 2^{16}-1	unity	-

Packet structure



Description

WriteIO writes one 16-bit word of data to the device whose address is calculated by adding 1000h to *address*. (*address* is an offset from the base address, 1000h, of the Navigator's memory-mapped I/O space.)

The format and interpretation of the 16-bit data word are dependent on the user-defined device being addressed. User-defined I/O can be used to implement a variety of features such as additional parallel I/O, flash memory for non-volatile configuration information storage, or display devices such as LED arrays.

Restrictions

see ReadIO

3 Instruction Summary Tables

3.1 Descriptions by Functional Category

Breakpoints and Interrupts

ClearInterrupt	Reset interrupt line
GetBreakpoint	Get breakpoint type
GetBreakpointValue	Get breakpoint comparison value
GetInterruptAxis	Get the axes with pending interrupts
GetInterruptMask	Get interrupt mask
SetBreakpoint	Set breakpoint type
SetBreakpointValue	Set breakpoint comparison value
SetInterruptMask	Set interrupt mask

Commutation

GetCommutationMode	Get the commutation mode
GetNumberPhases	Get the number of phases
GetPhaseAngle	Get current commutation phase angle
GetPhaseCommand	Get the motor output command for a given phase A, B, or C
GetPhaseCorrectionMode	Get phase correction mode
GetPhaseCounts	Get number of encoder counts per commutation cycle
GetPhaseInitializeMode	Get phase initialization mode
GetPhaseInitializeTime	Get the time parameters for algorithmic phase initialization
GetPhaseOffset	Get phase offset value
GetPhasePrescale	Get phasing prescaler
InitializePhase	Perform phase initialization procedure
SetCommutationMode	Set the commutation mode (Hall-based, sinusoidal, or microstepping)
SetNumberPhases	Set the number of phases (1, 2, or 3)
SetPhaseAngle	Set current commutation phase angle
SetPhaseCorrectionMode	Set phase correction mode (on or off)
SetPhaseCounts	Set number of encoder counts per commutation cycle
SetPhaseInitializeMode	Set phase initialization method (hall-based or algorithmic)
SetPhaseInitializeTime	Set the time parameters for algorithmic phase initialization
SetPhaseOffset	Set phase offset value
SetPhasePrescale	Set commutation prescaler mode (enable or disable)

Digital Servo Filter

ClearPositionError	Set position error to 0
GetAutoStopMode	Get auto stop mode
GetDerivative	Get the derivative of the error signal
GetDerivativeTime	Get derivative sampling time
GetIntegral	Get integrated position error value
GetIntegrationLimit	Get integration limit
GetKaff	Get acceleration feedforward gain
GetKd	Get derivative gain
GetKi	Get integral gain
GetKout	Get servo filter output scaler
GetKp	Get proportional gain
GetKvff	Get velocity feedforward gain
GetMotorBias	Get motor output bias
GetMotorLimit	Get motor output limit

GetPositionError	Get actual position error
GetPositionErrorLimit	Get position error limit
SetAutoStopMode	Set auto stop on position error (on or off)
SetDerivativeTime	Set derivative sampling time
SetIntegrationLimit	Set integration limit
SetKaff	Set acceleration feedforward gain
SetKd	Set derivative gain
SetKi	Set integral gain
SetKout	Set servo filter output scaler
SetKp	Set proportional gain
SetKvff	Set velocity feedforward gain
SetMotorBias	Set motor output bias
SetMotorLimit	Set motor output limit
SetPositionErrorLimit	Set maximum position error limit

Encoder

AdjustActualPosition	Sums the specified offset with the actual encoder position
GetActualPosition	Get the actual encoder position
GetActualPositionUnits	Get the unit type returned for the actual encoder position
GetActualVelocity	Get the actual encoder velocity
GetCaptureSource	Get capture source
GetCaptureValue	Get current axis position capture value and reset the capture
GetEncoderModulus	Get the full scale range of the parallel-word encoder
GetEncoderSource	Get encoder type
GetEncoderToStepRatio	Get encoder count to step ratio
SetActualPosition	Set the actual encoder position
SetActualPositionUnits	Set the unit type returned for the actual encoder position
SetCaptureSource	Set capture source (home or index)
SetEncoderModulus	Set the full scale range of the parallel-word encoder
SetEncoderSource	Set encoder type (incremental or 16-bit parallel word)
SetEncoderToStepRatio	Set encoder count to step ratio

External RAM

GetBufferFunction	Returns the buffer ID for a specified function
GetBufferLength	Get the length of a memory buffer
GetBufferReadIndex	Get the buffer read pointer for a particular buffer
GetBufferStart	Get the start location of a memory buffer
GetBufferWriteIndex	Get the buffer write pointer for a particular buffer
ReadBuffer	Read a long word value from a buffer memory location
SetBufferFunction	Assigns a buffer to the specified function
SetBufferLength	Set the length of a memory buffer
SetBufferReadIndex	Set the buffer read pointer for a particular buffer
SetBufferStart	Set the start location of a memory buffer
SetBufferWriteIndex	Set the buffer write pointer for a particular buffer
WriteBuffer	Write a long word value to a buffer memory location

Motor Output

GetCurrentMotorCommand	Read the current motor command value
GetMotorCommand	Read buffered motor output command
GetMotorMode	Get motor loop mode
GetOutputMode	Get output mode
SetStepRange	Sets the allowable range (in KHz) for step output generation
SetMotorCommand	Set direct value to motor output register
SetMotorMode	Set motor loop mode (on or off)
SetOutputMode	Set motor output mode (PWM sign-magnitude, PWM 50%, or DAC)

Profile generation

GetAcceleration	Get acceleration limit
GetCommandedAcceleration	Get commanded (instantaneous desired) acceleration
GetCommandedPosition	Get commanded (instantaneous desired) position
GetCommandedVelocity	Get commanded (instantaneous desired) velocity
GetDeceleration	Get deceleration limit
GetGearMaster	Get the electronic gear mode master axis and source
GetGearRatio	Get commanded electronic gear ratio
GetJerk	Get jerk limit
GetPosition	Get destination position
GetProfileMode	Get current profile mode set using SetProfileMode
GetStartVelocity	Get start velocity
GetStop	Get stop command; abrupt, smooth, or none
GetVelocity	Get velocity limit
MultiUpdate	Multiple axis immediate parameter update
SetAcceleration	Set acceleration limit
SetDeceleration	Set deceleration limit
SetGearMaster	Set the master axis and source (actual or target-based)
SetGearRatio	Set command electronic gear ratio
SetJerk	Set jerk limit
SetPosition	Set position limit
SetProfileMode	Set profile mode (S-curve, trapezoidal, velocity-contouring, or electronic gear)
SetStartVelocity	Set start velocity
SetStop	Set stop command. (abrupt stop, smooth stop, or none)
SetVelocity	Set velocity limit
Update	Immediate parameter update

Servo loop control

GetAxisMode	Get axis mode
GetLimitSwitchMode	Get limit switch mode
GetMotionCompleteMode	Get the motion complete mode
GetSampleTime	Get servo loop sample time
GetSettleTime	Get the axis-settled time
GetSettleWindow	Get the settle-window boundary value
GetTime	Get current chip set time (number of servo loops)
GetTrackingWindow	Get the tracking window boundary value
SetAxisMode	Set axis operation mode (enabled or disabled)
SetLimitSwitchMode	Set limit switching (on or off)
SetMotionCompleteMode	Set the motion complete mode (target-based or actual)
SetSampleTime	Set servo loop sample time
SetSettleTime	Set the axis-settled time
SetSettleWindow	Set the settle-window boundary
SetTrackingWindow	Set the tracking window boundary

Status Registers and AxisOut Indicator

GetActivityStatus	Get Activity Status
GetAxisOutSource	Get axis out signal monitor source
GetEventStatus	Get event status word
GetSignalStatus	Get the current axis Signal Status register
GetSignalSense	Get the interpretation of the Signal Status bits
ResetEventStatus	Reset bits in event status word
SetAxisOutSource	Set axis out monitor signal source
SetSignalSense	Set the interpretation of the Signal Status bits

Traces

GetTraceCount	Get the number of traced data points
GetTraceMode	Get the trace mode

GetTracePeriod	Get the trace period
GetTraceStart	Get the trace start condition
GetTraceStatus	Get the trace status word
GetTraceStop	Get the trace stop condition
GetTraceVariable	Get a trace variable setting
SetTraceMode	Set the trace mode (rolling or one-time)
SetTracePeriod	Set the trace period
SetTraceStart	Start the trace
SetTraceStop	Stop the trace
SetTraceVariable	Set variable (i.e., data) to be traced

Miscellaneous

GetChecksum	Reads the internal chip checksum
GetDiagnosticPortMode	Get the diagnostic port valid instruction mode
GetHostIOError	Get the most recent I/O error code
GetSerialPortMode	Read serial-port configuration data
GetSynchronizationMode	Get the synchronization mode
GetVersion	Get chipset software version information
NoOperation	Perform no operation, used to verify communications
ReadIO	Read user defined I/O value
Reset	Reset chipset
SetDiagnosticPortMode	Set the diagnostic port valid instruction mode (limited or full)
SetSerialPortMode	Set serial-port configuration data
SetSynchronizationMode	Set the synchronization mode to (master or slave)
WriteIO	Write user-defined I/O value

3.2 Alphabetical Listing

Note: Get/Set instruction pairs are shown together on the same line of the table

Instruction	Code	Instruction	Code
AdjustActualPosition	F5		
ClearInterrupt	AC		
ClearPositionError	47		
GetAcceleration	4C	SetAcceleration	90
GetActivityStatus	A6		
GetActualPosition	37	SetActualPosition	4D
GetActualPositionUnits	BF	SetActualPositionUnits	BE
GetActualVelocity	AD		
GetAutoStopMode	D3	SetAutoStopMode	D2
GetAxisMode	88	SetAxisMode	87
GetAxisOutSource	EE	SetAxisOutSource	ED
GetBreakpoint	D5	SetBreakpoint	D4
GetBreakpointValue	D7	SetBreakpointValue	D6
GetBufferFunction	CB	SetBufferFunction	CA
GetBufferLength	C3	SetBufferLength	C2
GetBufferReadIndex	C7	SetBufferReadIndex	C6
GetBufferStart	C1	SetBufferStart	C0
GetBufferWriteIndex	C5	SetBufferWriteIndex	C4
GetCaptureSource	D9	SetCaptureSource	D8
GetChecksum	F8		
GetCaptureValue	36		
GetCommandedAcceleration	A7		
GetCommandedPosition	1D		
GetCommandedVelocity	1E		
GetCommutationMode	E3	SetCommutationMode	E2
GetCurrentMotorCommand	3A		
GetDeceleration	92	SetDeceleration	91
GetDerivative	9B		
GetDerivativeTime	9D	SetDerivativeTime	9C
GetDiagnosticPortMode	8A	SetDiagnosticPortMode	89
GetEncoderModulus	8E	SetEncoderModulus	8D
GetEncoderSource	DB	SetEncoderSource	DA
GetEncoderToStepRatio	DF	SetEncoderToStepRatio	DE
GetEventStatus	31		
GetGearMaster	AF	SetGearMaster	AE
GetGearRatio	59	SetGearRatio	14
GetHostIOError	A5		
GetIntegral	9A		
GetIntegrationLimit	96	SetIntegrationLimit	95
GetInterruptAxis	E1		
GetInterruptMask	56	SetInterruptMask	2F
GetJerk	58	SetJerk	13
GetKaff	94	SetKaff	93
GetKd	52	SetKd	27
GetKi	51	SetKi	26
GetKout	9F	SetKout	9E
GetKp	50	SetKp	25
GetKvff	54	SetKvff	2B
GetLimitSwitchMode	81	SetLimitSwitchMode	80
GetMotionCompleteMode	EC	SetMotionCompleteMode	EB
GetMotorBias	2D	SetMotorBias	0F
GetMotorCommand	69	SetMotorCommand	77
GetMotorLimit	07	SetMotorLimit	06

Instruction	Code	Instruction	Code
GetMotorMode	DD	SetMotorMode	DC
GetNumberPhases	86	SetNumberPhases	85
GetOutputMode	6E	SetOutputMode	E0
GetPhaseAngle	2C	SetPhaseAngle	84
GetPhaseCommand	EA		
GetPhaseCorrectionMode	E9	SetPhaseCorrectionMode	E8
GetPhaseCounts	7D	SetPhaseCounts	75
GetPhaseInitializeMode	E5	SetPhaseInitializeMode	E4
GetPhaseInitializeTime	7C	SetPhaseInitializeTime	72
GetPhaseOffset	7B	SetPhaseOffset	76
GetPhasePrescale	E7	SetPhasePrescale	E6
GetPosition	4A	SetPosition	10
GetPositionError	99		
GetPositionErrorLimit	98	SetPositionErrorLimit	97
GetProfileMode	A1	SetProfileMode	A0
GetSampleTime	61	SetSampleTime	38
GetSerialPortMode	8C	SetSerialPortMode	8B
GetSettleTime	AB	SetSettleTime	AA
GetSettleWindow	BD	SetSettleWindow	BC
GetSignalStatus	A4		
GetSignalSense	A3	SetSignalSense	A2
GetStartVelocity	6B	SetStartVelocity	6A
GetStepRange	CE	SetStepRange	CF
GetStopMode	D1	SetStopMode	D0
GetSynchronizationMode	F3	SetSynchronizationMode	F2
GetTime	3E		
GetTraceCount	BB		
GetTraceMode	B1	SetTraceMode	B0
GetTracePeriod	B9	SetTracePeriod	B8
GetTraceStart	B3	SetTraceStart	B2
GetTraceStatus	BA		
GetTraceStop	B5	SetTraceStop	B4
GetTraceVariable	B7	SetTraceVariable	B6
GetTrackingWindow	A9	SetTrackingWindow	A8
GetVelocity	4B	SetVelocity	11
GetVersion	8F		
InitializePhase	7A		
MultiUpdate	5B		
NoOperation	00		
ReadAnalog	EF		
ReadBuffer	C9		
ReadIO	83		
Reset	39		
ResetEventStatus	34		
Update	1A		
WriteBuffer	C8		
WriteIO	82		

3.3 Numeric Listing

Code	Instruction
00	NoOperation
06	SetMotorLimit
07	GetMotorLimit
0F	SetMotorBias
10	SetPosition
11	SetVelocity
13	SetJerk
14	SetGearRatio
1A	Update
1D	GetCommandedPosition
1E	GetCommandedVelocity
25	SetKp
26	SetKi
27	SetKd
2B	SetKvff
2C	GetPhaseAngle
2D	GetMotorBias
2F	SetInterruptMask
31	GetEventStatus
34	ResetEventStatus
36	GetCaptureValue
37	GetActualPosition
38	SetSampleTime
39	Reset
3A	GetCurrentMotorCommand
3E	GetTime
47	ClearPositionError
4A	GetPosition
4B	GetVelocity
4C	GetAcceleration
4D	SetActualPosition
50	GetKp
51	GetKi
52	GetKd
54	GetKvff
56	GetInterruptMask
58	GetJerk
59	GetGearRatio
5B	MultiUpdate
61	GetSampleTime
69	GetMotorCommand
6A	SetStartVelocity
6B	GetStartVelocity
6E	GetOutputMode
72	SetPhaseInitializeTime
75	SetPhaseCounts
76	SetPhaseOffset
77	SetMotorCommand
7A	InitializePhase
7B	GetPhaseOffset
7C	GetPhaseInitializeTime
7D	GetPhaseCounts
80	SetLimitSwitchMode
81	GetLimitSwitchMode
82	WriteIO
83	ReadIO
84	SetPhaseAngle
85	SetNumberPhases
86	GetNumberPhases
87	SetAxisMode
88	GetAxisMode
89	SetDiagnosticPortMode
8A	GetDiagnosticPortMode
8B	SetSerialPortMode
8C	GetSerialPortMode
8D	SetEncoderModulus
8E	GetEncoderModulus
8F	GetVersion
90	SetAcceleration
91	SetDeceleration
92	GetDeceleration
93	SetKaff
94	GetKaff
95	SetIntegrationLimit
96	GetIntegrationLimit
97	SetPositionErrorLimit
98	GetPositionErrorLimit
99	GetPositionError
9A	GetIntegral
9B	GetDerivative
9C	SetDerivativeTime
9D	GetDerivativeTime
9E	SetKout
9F	GetKout
A0	SetProfileMode
A1	GetProfileMode
A2	SetSignalSense
A3	GetSignalSense
A4	GetSignalStatus
A5	GetHostIOError
A6	GetActivityStatus
A7	GetCommandedAcceleration
A8	SetTrackingWindow
A9	GetTrackingWindow
AA	SetSettleTime
AB	GetSettleTime
AC	ClearInterrupt
AD	GetActualVelocity
AE	SetGearMaster
AF	GetGearMaster
B0	SetTraceMode
B1	GetTraceMode
B2	SetTraceStart
B3	GetTraceStart
B4	SetTraceStop
B5	GetTraceStop
B6	SetTraceVariable
B7	GetTraceVariable
B8	SetTracePeriod
B9	GetTracePeriod
BA	GetTraceStatus
BB	GetTraceCount
BC	SetSettleWindow
BD	GetSettleWindow
BE	SetActualPositionUnits
BF	GetActualPositionUnits
C0	SetBufferStart
C1	GetBufferStart
C2	SetBufferLength
C3	GetBufferLength
C4	SetBufferWriteIndex
C5	GetBufferWriteIndex
C6	SetBufferReadIndex
C7	GetBufferReadIndex
C8	WriteBuffer
C9	ReadBuffer
CA	SetBufferFunction
CB	GetBufferFunction
CE	GetStepRange
CF	SetStepRange
D0	SetStopMode
D1	GetStopMode
D2	SetAutoStopMode
D3	GetAutoStopMode
D4	SetBreakpoint
D5	GetBreakpoint
D6	SetBreakpointValue
D7	GetBreakpointValue
D8	SetCaptureSource
D9	GetCaptureSource
DA	SetEncoderSource
DB	GetEncoderSource
DC	SetMotorMode
DD	GetMotorMode
DE	SetEncoderToStepRatio
DF	GetEncoderToStepRatio
E0	SetOutputMode
E1	GetInterruptAxis
E2	SetCommutationMode
E3	GetCommutationMode
E4	SetPhaseInitializeMode
E5	GetPhaseInitializeMode
E6	SetPhasePrescale
E7	GetPhasePrescale
E8	SetPhaseCorrectionMode
E9	GetPhaseCorrectionMode
EA	GetPhaseCommand
EB	SetMotionCompleteMode
EC	GetMotionCompleteMode
ED	SetAxisOutSource
EE	GetAxisOutSource
EF	ReadAnalog
F2	SetSynchronizationMode
F3	GetSynchronizationMode
F5	AdjustActualPosition
F8	GetChecksum