SAIA-Burgess Electronics

SWITCHES • MOTORS • CONTROLLERS

PCD2.H320

Motion Library xx7

Programming Guide

26/778 E1

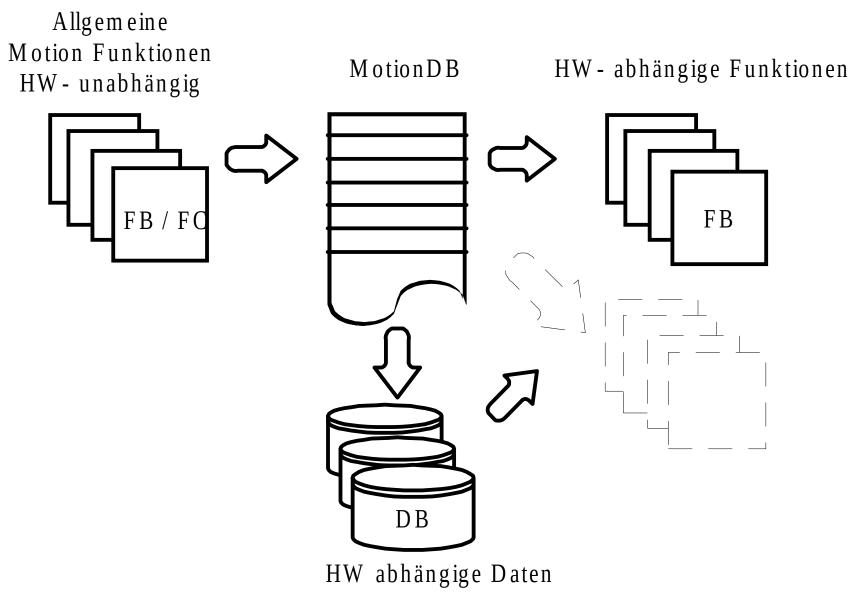
Table of Contents

1 Introduction

In Chapter 1 is a general overview of the motion Concept. Chapter 2 describes the main function blocks for basic functions. Chapter 4 to 9 describes how to use the functions for the H320 module.

1.1 *Structure of the motion control concept*

The concept comprises three parts: The first part consists of HW-independent function and data-blocks. These describe the functions that are to be executed (initialization, positioning, etc.). They also provide the user with a uniform, HW-independent interface. The second part consists of HW-dependent function and data-blocks. These execute the functions on the projected HW. These functions are not visible to the user, unless they have to be loaded. The DB parameter represents an exception. In this DB the user must enter the various HW parameters (such as module addresses, control parameters, mechanical factors, etc.).
The third part is the link between the first and second parts. It consists of a DB containing stored information on which FBs and DBs are valid for the chosen HW. (In the following diagram it is called the MotionDB).



2 Basic functions

This chapter describes the basic functions. At present 6 functions exist for controlling the axes:

FC 20	Initialization
FB 21	Homing
FC 22	Motion in velocity mode
FC 23	Positioning mode without breakpoint
FB 24	Positioning mode with table without breakpoint
FB 25	Teach in for tables without breakpoint
FC 26	Current position and status
FC 27	Positioning mode with breakpoint
FB 28	Positioning mode with table with breakpoint
FC 29	write new Parameter
FB 30	Gearing Mode
FC 31	SetBreakpoint
UDT 110	Structure of the motionDB
UDT 300	Structure of the Init DB
UDT 301	Structure of the Position table
UDT 303	Structure of the Position table with breakpoints

Each function has at least three parameters: MotionDB, Error and Enable/Start. The “MotionDB” input is an INT value. This value corresponds to the number of the DB that stores which FBs and DBs should be used for the relevant HW. The precise structure of this MotionDB is described in chapter 3.

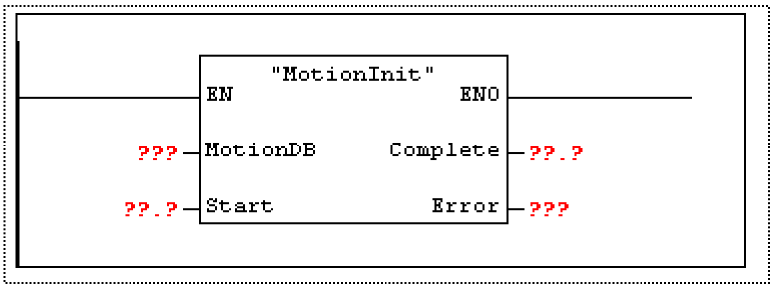
The “Error” output is one BYTE. If the function has been executed successfully, this value is set at zero. If an error occurs during execution, this output is set at a value not equal to zero.

Neither of these parameters is further described or listed with the individual functions.

The same function must never be active repeatedly for one axis, as otherwise movements will not be correctly executed.

2.1 *Initialization*

Call block FC20:



Parameters:

Name	**Type**	**I/O**	**Description**
MotionDB	INT	IN	Number of motion DB
Start	BOOL	IN	Initialization is started with a positive edge
Complete	BOOL	OUT	When initialization is complete, this bit becomes high
Error	BYTE	OUT	0: The initialization is executed successfully 1: The Filter Parameters (PID- Parameters) are out of range 5: Trace is running B: Invalid negativ value 92: DB could not be produced because the function compress memory is activ A1: Error on DB Nr. The DB Nr. has to be between 1 and 1023. (Value in MotionDB at Adress 4) B1: There is not enough memory to produce a DB B2: There is not enough memory to produce a DB B3: There is not enough memory to produce a DB (you should compress the memory)

Function:

This block calls the FB specified in MotionDB at address 6. The DB number at address 0 is given as the ParameterDB. A precise description of this ParameterDB is given in chapter 3. In the initialization routine the parameters are first checked and converted into suitable values for the module. These modified parameters are filed in a special DB (DW 4 in the MotionDB). Finally, the H320 module is initialized.

Initialization must be called once. The functions described in the following can fulfil their proper function only after the successful execution of the block.

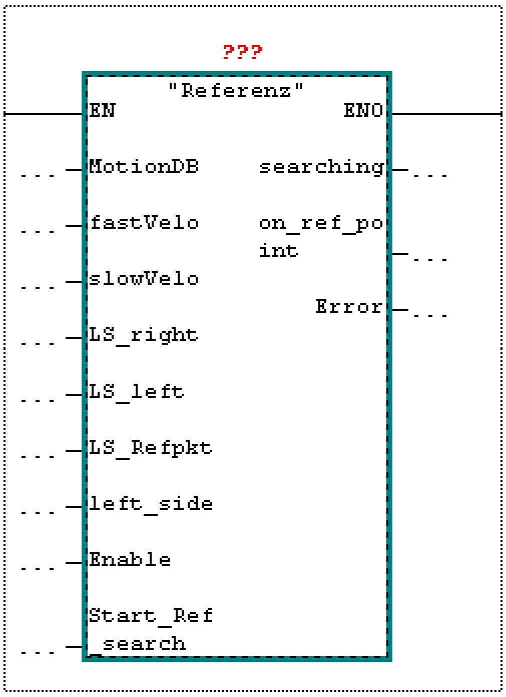
When a positive edge arrives at the Start input, initialization commences. The Start input must remain at true at least until the Complete bit has been set.

Initialization cannot be executed in OB100, but must be called in OB1.

The Initialization has to be called for both axis on the module. For each axis it is nessesary to use a different Motion DB.

2.2 Home position

Call block FB 21:



Parameters:

Name	**Type**	**I/O**	**Description**
fastVelo	real	in	Velocity of search for reference point
slowVelo	real	in	Velocity of travel down from reference point
LS_right	bool	in	Right-hand limit switch input is not used, because the H320 Module has its own limit switch input.
LS_left	bool	in	Left-hand limit switch input is not used, because the H320 Module has its own limit switch input.
LS_refPoint	bool	in	Reference point input is not used. The H320 has its own Reference input. If the Reference Input on the H320 becomes 0, direction is changed and travel continues at SlowVelo speed until the input becomes 1 again.

left_side	bool	in	If this input is TRUE, travel takes place first to the left. The reference point must be approached equally from the left for it to be set. If this input is FALSE, travel takes place first to the right. The reference point must be approached equally from the right for it to be set.
Enable	bool	in	Axis enable. If this input is FALSE, the axis remains immobile.
searching	bool	out	Searching for the reference point, the axis is in motion.
on_ref_point	bool	out	Reference point has been found and set.
Error	byte	out	0: no error 1: a position Function is activ 4: invalid parameter 6: Both limit switches are reached without reaching the Reference switch B: invalid negative value C: invalid parameter change
Start_Ref_search	bool	in_out	On receipt of a positive edge, parameters are adopted and homing commences. When the reference point has been reached, this function resets the bit.

Description:

This function is used to search for the reference point. The “left_side” input defines the first search direction and the side from which the reference switch must be approached. When the reference point has been reached, travel continues at the second velocity from the reference switch. Only then is the reference point set. See Figure 1.

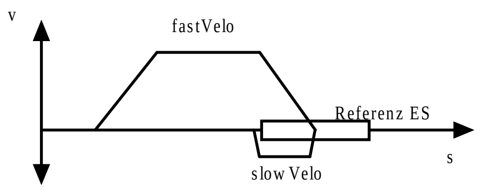
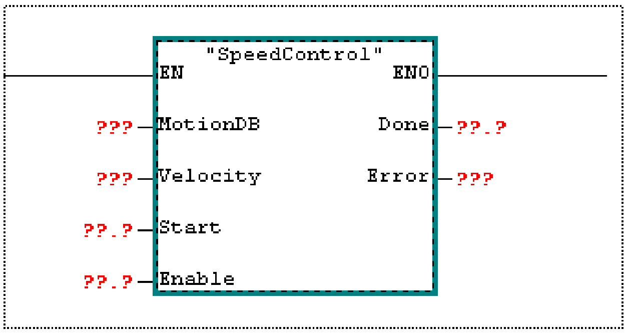


Figure 1 Search for reference point

Zero offset compensation can be specified in the ParameterDB for initialization. When the reference point is set, this zero offset compensation is also taken into account.
When both limit switches “LS_right” and “LS_left” have each been reached once, the “Error” output is set and the axis is stopped.
In the first Version, the Index Input is not used for finding the reference point.

2.3 Speed Control

Call block FC22:



Parameters:

Name	**Type**	**IO**	**Description**
Velocity	real	in	Velocity at which the drive runs. The preceding sign indicates direction.
Start	bool	in	Start axis.
Enable	bool	in	Enable axis.
Done	bool	out	The axis moves.
Error	byte	out	If the axis cannot be moved, this byte is set not equal to 0. Error = 1: a motion control function is already active. 4: invalid Parameter B: invalid negativ value C: invalid parameter change D: invalid move after limit condition E: invalid move into limit 10: motion error 12: motion error + Wrap around 20: positiv limit switch reached 22: positiv limit switch and wrap around 40: negativ limit switch 42: negativ limit switch and wrap around

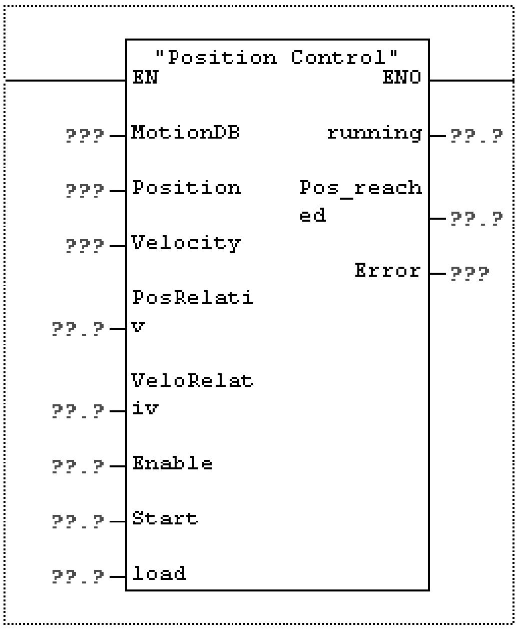
Description:

This function is used to move the axis at the prescribed velocity. The preceding sign for velocity determines the direction of motion. If the value of “Velocity” changes, the new velocity is adopted.

If the enable signal is zeroed during motion, a fast stop will be executed at the maximum breaking ramp.

2.4 Position_Control (without breakpoint)

Call block FC23:



Parameters:

Name	Type	IO	Description
Position	real	in	Position to be approached. The direction is indicated by the preceding sign.
Velocity	real	in	Max. velocity at which the above position should be approached. The preceding sign has no significance here.
PosRelativ	bool	in	True: The position is relative False: The positon is absolute
VeloRelativ	bool	in	True: Velocity is relative to the current one False: Velocity is absolute
Enable	bool	in	Enable axis.
running	bool	out	Message returned when axis is travelling to the position
Pos_reached	bool	out	Message returned when the axis is located at the position
Error	byte	out	1: other motion function is active 4: Invalid Parameter 6: Motion Error B: Invalid negativ value C: Invalid parameter change D: Invalid move after limit condition

			E: Invalid move into limit
Start	bool	in_ out	On receipt of a positive edge, parameters are loaded and effective. As soon as parameters are effective, the function resets this bit to 0 and the axis is started.
Load	bool	in_ out	On receipt of a positive edge, parameters are just loaded. The parameters get effective with the next update from an other function or after some breakpoints conditions.

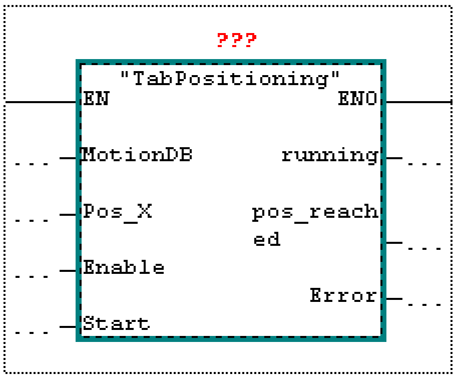
Description:

This function is required for positioning an axis.

The drive travels at the prescribed velocity to the specified position. During travel it is possible to specify a new position / velocity. As soon as the start command is set, the new position / velocity will be adopted, regardless of whether the old position has been reached. When the specified position has been reached, the drive stops. The message returned indicates the status of positioning. If it has not been possible to approach the position, one or more of the Bits in the error message is set.

2.5 *Approach previously defined position*

Call block FB24:



Parameters:

Name	Type	IO	Description
Pos_X	int	in	Position number to be reached. A table (MotionDB address 2) contains the absolute position and the velocity at which the position should be approached.
Enable	bool	in	Enable axis
running	bool	out	Message returned when motion control is running
pos_reached	bool	out	Message returned when the axis is located on the position

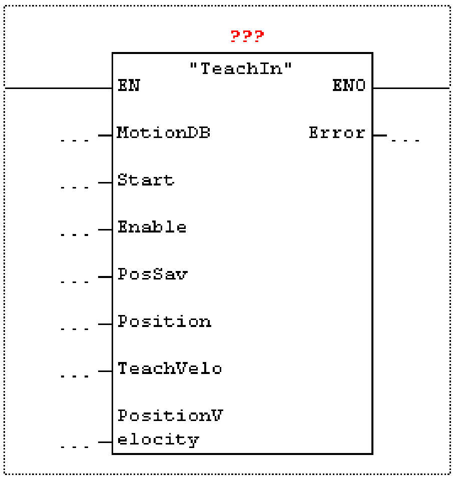
Error	byte	out	1: other motion function is active 2: invalid instruction 4: invalid parameter 6: Motion error B: invalid negativ value C: invalid parameter change D: invalid move after limit condition E: invalid move into limit
Start	bool	in_ out	On receipt of a positive edge, parameters are adopted. This bit is reset to 0 by the function.

Description:

A table (DBNo. in the Motion DB under address 2) is used to store positions and the appropriate velocities (entered in the table manually or via a Teach-In function). Using the position number, the function locates the corresponding position and velocity in the table. That position is then approached. If a positive edge is set at the "Start" input during motion control, the new position is adopted and approached, regardless of whether the old position has been reached.

2.6 Teach in

Call block FB25:



Parameters:

Name	Type	IO	Description
Start	bool	in	As long as this input is at TRUE, the axis moves at the velocity of "TeachVelo".
Enable	bool	in	Enable axis. Without this enable, values cannot be saved.

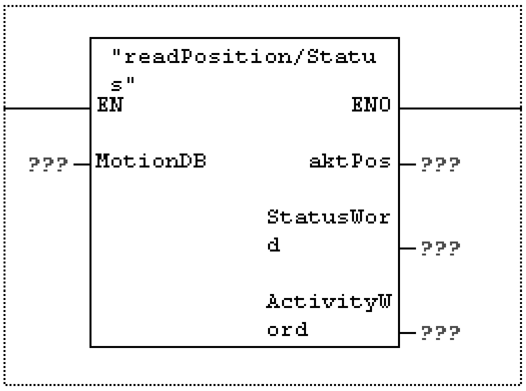
PosSav	bool	in	The current position is entered in the table
Position	int	in	The current position will be approached again later with this number
TeachVelo	real	in	The velocity at which the axis will move during the Teach-In. The preceding sign determines the direction.
PositionVelo	real	in	The position saved will be approached at this speed. (The preceding sign has no significance).
Error	byte	out	1: The axis cannot be moved 2: The value at input “Position” is invalid 0: No error

Description:

The “Start” input moves the axis at the velocity of “TeachVelo”. The sign preceding the velocity produces the direction in which the axis moves. When there is a positive edge at the “PosSav” input, the current position is entered in a table together with the velocity at the “PositionVelo” input. If the “Enable” input is at zero, the axis is immobilized after a fast stop. Likewise, no positions may be stored.

2.7 *Read current position and status*

Call block FC26:



Parameters:

Name	Type	IO	Description
Position	real	out	The current position at which the axis is located at the moment of calling.
StatusWord	word	out	See Table below.
ActivityWord	word	out	See Table below.

Description:

Current position can be queried in user units with this function.

The Event Status register and the Activity register from the H320 can be read with this function.

The following Table shows the meaning of the two registers.

Status Word

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

Activity Word

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.

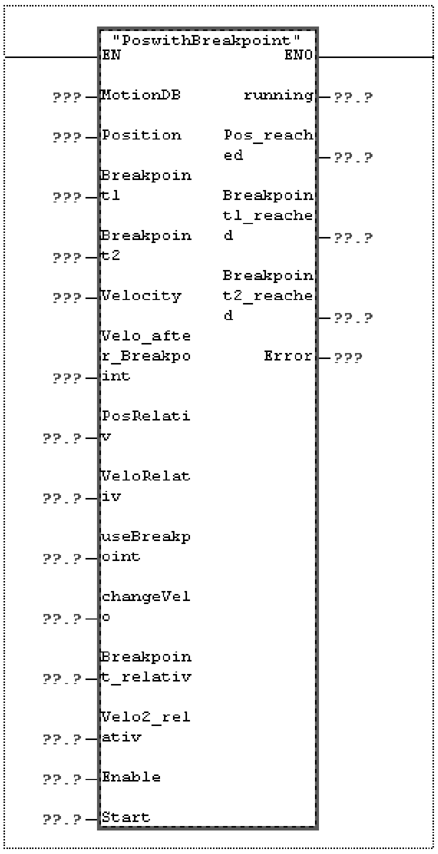
The following table shows the encoding of the data returned by this command.

Name	***Bit Number***	***Description***
Phasing initialized	0	Set to 1 if phasing is initialized (MC2300 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
In negative limit	12	Set to 1 when the negative limit switch is active
Profile segment	13-15	Only used during S-curve profile mode. Contains value of 0 when the profile is at rest. Contains phase number 1-7 when profile is in motion.

2.8 Position Control with Break Point

Call block FC27:



Parameters:

Name	Type	IO	Description
Position	real	in	Position to be approached. The direction is indicated by the preceding sign.
Breakpoint	real	in	At this Position the Breakpoint Flag will be set.
Velocity	real	in	Max. velocity at which the above position should be approached. The preceding sign has no significance here.
Velo_after_Breakpoint	real	in	When the Breakpoint Flag is set, the speed of the axis will change to this speed.
PosRelativ	bool	in	True: The position is relative False: The positon is absolute
VeloRelativ	bool	in	True: Velocity is relative to the current one False: Velocity is absolute
useBreakpoint	bool	in	True: The Breakpoint will be loaded and the breakpoint flag2 be set and the output OUT is set.

			False: Thre Breakpoint has no affect.
changeVelo	bool	in	True: When the breakpoint is reached, the velocity will change to Velo_after_Breakpoint. Also the brakpoint flag1 be set. False: The Value Velo_afterBreakpoint has no affect.
Breakpoint_relativ	bool	in	True: The breakpoint is relativ False: The breakpoint is absolut
Velo2_relativ	bool	in	True: the second Velocity is relative to the current one False: the second Velocity is absolute
Enable	bool	in	Enable axis.
running	bool	out	Message returned when axis is travelling to the position
Pos_reached	bool	out	Message returned when the axis is located at the position
Breakpoint_reached	bool	out	True: the breakpoint is reached.
Error	byte	out	1: other motion function is active 2: invalid instruction 4: invalid parameter 6: Motion error B: invalid negativ value C: invalid parameter change D: invalid move after limit condition E: invalid move into limit
Start	bool	in_out	On receipt of a positive edge, parameters are loaded and effective. As soon as parameters are effective, the function resets this bit to 0 and the axis is started.

Description:

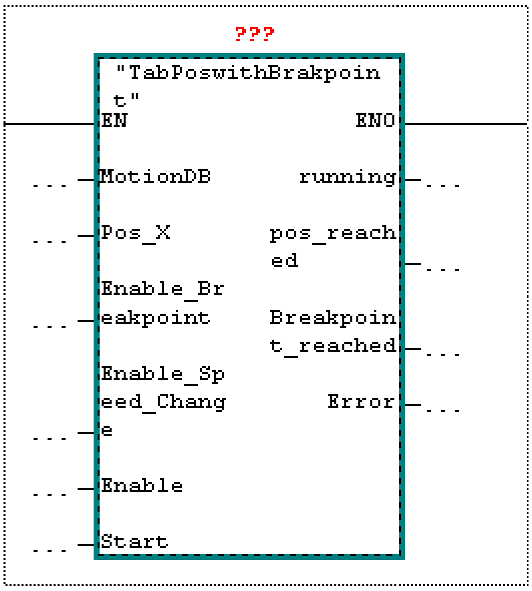
This function is required for positioning an axis.

The drive travels at the prescribed velocity to the specified position. During travel, it is possible to specify a new position / velocity. As soon as the start command is set, the new position / velocity will be adopted, regardless of whether the old position has been reached or not. When the specified position has been reached, the drive stops. The message returned indicates the status of positioning. If it has not been possible to approach the position, an error message is set.

In this function it is possible to set two breakpoints. One to change the velocity at the breakpoint, and the other to set the Output OUT.

2.9 *Approach previously defined position with Break Point*

Call block FB28:



Parameters:

Name	Type	IO	Description
Pos_X	int	in	Position number to be reached. A table (MotionDB address 34) contains the absolute position and the velocity at which the position should be approached.
Enable_Breakpoint	bool	in	True: The Breakpoint will be loaded and the breakpoint flag2 be set. The output OUT will be set. False: The Breakpoint has no affect.
Enable_Speed_Change	bool	in	True: When the breakpoint is reached, the velocity will change to Velo_after_Breakpoint False: The Value Velo_afterBreakpoint has no affect.
Enable	bool	in	Enable axis
running	bool	out	Message returned when motion control is running
pos_reached	bool	out	Message returned when the axis is located on the position
Breakpoint_reached	bool	out	True: the breakpoint is reached.
Error	byte	out	1: other motion function is active 2: invalid instruction 4: invalid parameter 6: Motion error B: invalid negativ value C: invalid parameter change D: invalid move after limit condition

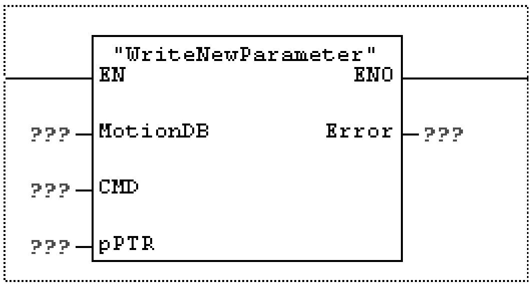
			E: invalid move into limit
Start	bool	in_ out	On receipt of a positive edge, parameters are adopted. This bit is reset to 0 by the function.

Description:

A table (DBNo. in the Motion DB under address 34) is used to store positions, the appropriate velocities, the breakpoint and the second velocities (entered in the table manually). Using the position number, the function locates the corresponding position, breakpoint and velocity in the table. That position is then approached. If the breakpoint is reached, the flag Breakpoint reched is set to True and the Velocity will change to the second Velocity. If a positive edge is set at the "Start" input during motion control, the new position is adopted and approached, regardless of whether the old position has been reached.

2.10 Write Parameters to the Module

Call block FC29:



Parameters:

Name	**Type**	**IO**	**Description**
CMD	int	in	Command Nummer 0 to 41. see next Table
pPTR	pointer	in	The pointer shows to the value. Ex. P#M10.0. If the value is a byte then MB10 is readed. If the value is a word then MW10 is readed. If the value is a real or double word then MD10 is readed.
Error	word	out	0: No Error 1: Processor Reset 2: Invalid instruction 3: Invalid axis 4: Invalid parameter 5: Trace running

			6: reserved 7: Block out of bounds 8: Trace buffer zero 9: Bad serial checksum A: Not primary port B: Invalid negativ value C: Invalid parameter change D: Invalid move after limit condition E: Invalid move into limit F: Wrong command Number

Description:

The function change parameters. The right use of these parameters see this dokumentation.

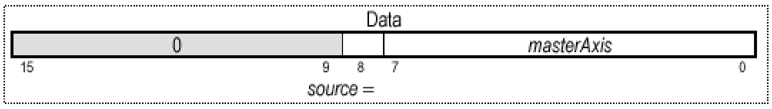
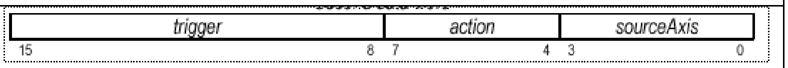
CMD Nr	**Name**	**Type**	**Description**
0	LimitSwitch Mode	byte	SetLimitSwitchMode enables (On) or disables (Off) limit-switch sensing for the specified *axi*s. When the mode is enabled, the axis will cause the corresponding limit-switch bits in the Event Status register and Activity Status register 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.
1	AxisMode	byte	SetAxisMode enables (On) or disables (Off) the specified *axi*s. A disabled axis will not respond to profile or other motion commands.
2	ProfilMode	byte	SetProfileMode sets the profile mode, selecting Trapezoidal (0), Velocity Contouring (1), S-curve (2), or Electronic gear (3) for the specified *axi*s.
3	Stop	byte	Stop stops the specified axis. The available stop modes are AbruptStop (1), which instantly (without any deceleration phase) stops the axis, SmoothStop (2) which uses the programmed deceleration value and profile shape for the current profile mode to stop the axis, or NoStop (0) 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. **Restrictions** SmoothStop mode is not valid in the

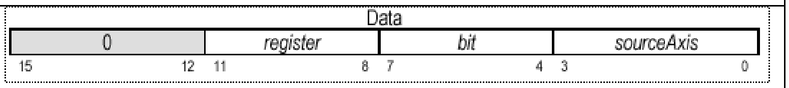
			electronic-gearing profile. SetStop is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.
4	MotorMode	byte	MotorMode determines the mode of motor operation; open loop or closed loop for the specified *axi*s. When set to On, the axis is 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. When 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.
5	AutoStopMode	byte	AutoStopMode determines the behavior of the specified *axis* when a motion error occurs. When auto stop is enabled (AutoStopMode = 1), the axis goes into open-loop mode when a motion error occurs. When Auto-Stop is disabled (AutoStopMode = 0), the axis is not affected by a motion error.
6	CaptureSource	byte	SetCaptureSource determines which of two encoder signals, Index (0) or Home (1), is used to trigger the high-speed capture of the actual axis position for the specified *axi*s.
7	MotionComplite Mode	byte	MotionCompleteMode establishes the source for the comparison which determines the motion-complete status for the specified *axi*s. When set to commanded (0) 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 (1) 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.
8	Update	byte	Update causes all buffered data parameters are copied into the corresponding run-time registers on the specified *axi*s. 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, ntegrationLimit, Kaff, Kd, Ki, Kp, and Kvff. The following Motor Command parameters is buffered: MotorCommand.
9	ClearPosition Error	byte	ClearPositionError sets the current profile's commanded position equal to the actual position (encoder input), thereby clearing the position error for the specified *axi*s. 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.
10	Multiuptdate	byte	***Instance*** ***Encoding*** None 0 Axis1mask 1 Axis2mask 2 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, Integration Limit, Kaff, Kd, Ki, Kp, and Kvff. The following Motor Command parameters is buffered: MotorCommand
11	MotorLimit	real -100.0 to +100.0 %	SetMotorLimit sets the maximum value for the motor output command allowed by the digital servo filter of the specified *axi*s. 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.

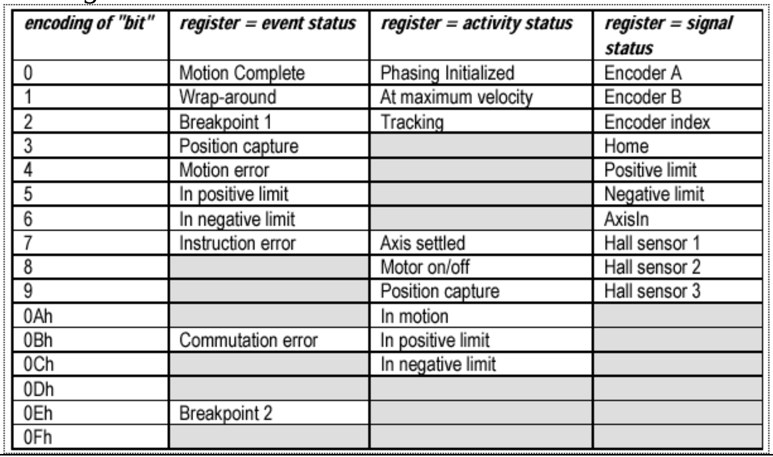
			Restrictions This command only affects the motor ouput when in closed loop mode. When the chipset is in open loop mode this command has no affect.
12	MotorBias	real -100.0 to +100.0 %	SetMotorBias sets the bias voltage of the digital servo filter for the specified axis.
13	KP	int 0 *to* 2^{15} -1	SetKp sets the proportional gain of the digital servo filter for the specified *axi*s. **Restrictions** SetKp is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.
14	KI	int 0 *to* 2^{15} -1	SetKi sets the integral gain of the digital servo filter for the specified *axi*s. **Restrictions** This is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.
15	KD	int 0 *to* 2^{15} -1	SetKd sets the derivative gain of the digital servo filter for the specified axis. **Restrictions** SetKd is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.
16	Kvff	int 0 *to* 2^{15} -1	SetKvff sets the velocity feedforward gain of the digital servo filter for the specified *axi*s. **Restrictions** SetKvff is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.
17	InterruptMaske	word	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 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.

			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).
18	MotorCommand	real -100.0 to +100.0 %	SetMotorCommand loads the motor-command buffer register of the specified *axi*s. **Restrictions** SetMotorCommand is valid only when the motor is "off". SetMotorCommand is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.
19	Kaff	int 0 *to* 2^{15} -1	SetKaff sets the acceleration feedforward gain of the digital servo filter for the specified *axi*s. **Restrictions** *SetKaff* is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.
20	DerivativeTime	real >= 0.0	SetDerivativeTime sets the sampling time, in number of servo cycles, for the servo filter to use in calculating the derivative term for the specified *axi*s. **Restrictions** This command is NOT buffered. The new sampling time value will take effect immediately after the command is sent to the chipset. 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.
21	Kout	real 0.0 to +100.0 %	SetKout sets the output scale factor of the digital servo filter for the specified axis. The default value is 100.0 % **Restrictions** This command is NOT buffered. It will take affect immediately after it is sent.
22	SignalSense	word	Encoder A 0001h Encoder B 0002h Encoder Index 0004h Encoder Home 0008h Positive limit 0010h Negative limit 0020h AxisIn 0040h AxisOut 0400h 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 *axi*s. 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.

23	Tracking Window	real >= 0.0	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.
24	SettleTime	real >= 0.0	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.
25	GearMaster	word	SetGearMaster establishes the slave *(axis*) and master *(masterAxi*s) 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. **Restrictions** For electronic gear mode to operate properly the master axis must be enabled.  Data 0 masterAxis 15 9 8 7 0 *source* = master Axis: Axis1 0 Axis2 1 *source* Actual 0 Commanded 1
26	SettleWindow	real >= 0.0	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.
27	Breakpoint1	word	 *trigger* *action* *sourceAxis* 15 8 7 4 3 0 *sourceAxis* Axis1 0 Axis2 1 *action* (none) 0 Update 1 AbruptStop 2 SmoothStop 3 MotorOff 4 *trigger* (none) 0 PositiveCommandedPosition 1 NegativeCommandedPosition 2 PositiveActualPosition 3 NegativeActualPosition 4

			CommandedPositionCrossed 5 ActualPositionCrossed 6 Time 7 EventStatus 8 ActivityStatus 9 Signal Ah SetBreakpoint establishes a breakpoint for the specified *axis* to be triggered by a condition or event on *sourceAxi*s, which may be the same as or different from *axi*s. Up to two concurrent breakpoints can be set for each axis. The six Position breakpoints and the Time breakpoint are *threshold-triggere*d; the breakpoint occurs when the indicated value reaches or crosses a threshold. The Status breakpoints are *level-triggere*d; 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 — SetStop *axi*s, AbruptStop Update *axis* SmoothStop — SetStop *axi*s, SmoothStop Update *axis* MotorOff — SetMotorMode *axi*s, Off Update *axis* *axis* is the axis for which the breakpoint has been set. 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.
28	Breakpoint2	word	see Breakpoint 1
29	AxisOutSource	word	 Data 0 register bit sourceAxis 15 12 11 8 7 4 3 0 *axis* Axis1 0 Axis2 1 *sourceAxis* Axis1 0 Axis2 1 *bit see below* 0 to 15

			register (none) 0 EventStatus 1 ActivityStatus 2 SignalStatus 3 SetAxisOutSource maps the specified *bit* of the specified status *register* of *axisn* to the AxisOut pin for the specified *axi*s. The state of the AxisOut pin will thereafter track the state of *bi*t. If *register* is absent (encoding of 0), *bit* is ignored, and the specified AxisOut pin is, in effect, turned off (inactive). The table below shows the source for combinations of *bit* and *registe*r.



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			

30	ResetEvent Status		ResetEventStatus clears (sets to 0) , for the specified *axi*s, 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. 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
31	Position	real in User units	SetPosition specifies the trajectory destination of the specified *axi*s. It is used in the Trapezoidal and S-curve profile modes. **Restrictions** SetPosition is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.
32	Velocity	real in User units	SetVelocity loads the Maximum Velocity buffer register for the specified *axi*s. **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.
33	Jerk	real in User units	SetJerk loads the jerk register in the parameter buffer for the specified *axi*s. 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.
34	GearRation	real	SetGearRatio sets the ratio between the master and slave axes for the electronic gearing profile for the current *axi*s. Positive ratios cause the slave to move in the same direction as the master, negative ratios in the opposite direction. **Restrictions** This is a buffered command. The new value set will not take effect until the next Update or MultiUpdate instruction is entered.
35	Actual Positions	real in User units	SetActualPosition loads the actual position register (encoder position) for the specified *axi*s. 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.
36	Acceleration	real in User units	SetAcceleration loads the maximum acceleration buffer register for the specified *axi*s. This command is used with the Trapezoidal, Velocity Contouring, and S-curve profiling modes. **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.
37	Deceleration	real in User units	SetDeceleration loads the maximum deceleration buffer register for the specified *axi*s. This command sets the magnitude of the deceleration register, which always has a negative sign. **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.
38	Integration Limit	real length * time	SetIntegrationLimit loads the integration-limit register of the digital servo filter for the specified *axi*s. **Restrictions** This is a buffered command. The value set using this command will not take effect until the next Update or MultiUpdate instruction.
39	PositionError Limit	real in User units	SetPositionErrorLimit sets the absolute value of the maximum position error allowable by the chipset for the specified *axi*s. 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.
40	Breakpoint Value1	dword	PositiveCommandedPosition signed 32 bit NegativeCommandedPosition signed 32 bit PositiveActualPosition signed 32 bit NegativeActualPosition signed 32 bit CommandedPositionCrossed signed 32 bit ActualPositionCrossed signed 32 bit Time unsigned 32 bit EventStatus 2 word mask* ActivityStatus 2 word mask* SignalStatus 2 word mask* * see description section below for more details on mask format SetBreakpointValue sets the breakpoint comparison value for the specified *axi*s. 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, and 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. **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.
41	Breakpoint Value2		see BreakpointValue1

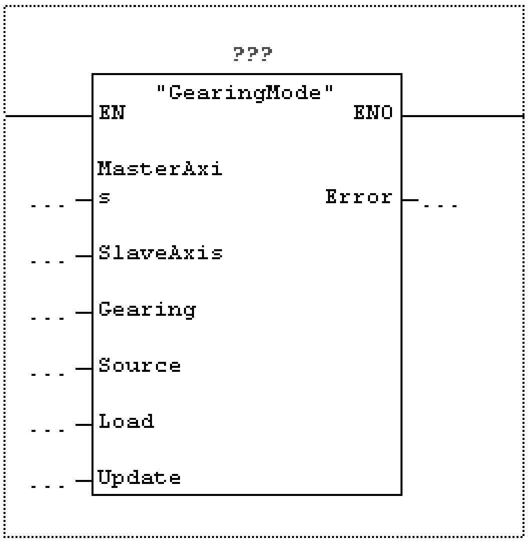
Description:

With this function it is possible to write all possible parameters. All parameters are in User units and the function callculate it in Modul units.

For setting breakpoint parameters there exists other Functions which do it more easyly. (FC27, FB28, FC31).

2.11 Gearing Mode

Call block FB30:



Parameters:

Name	**Type**	**IO**	**Description**
MasterAxis	int	in	MotionDB Number of the Master Axis
SlaveAxis	int	in	MotionDB Number of the Slave Axis
Gearing	real	in	ratio Master / Slave Impulse
Source	byte	in	0: Actual Position 1: Commanded Position
Load	bool	in	Load the Value in the Module. The Values are not activ until a update instruction followed. (By a this module or by a breakpoint or an other function).
Update	bool	in	Update the loaded parameter.
Error	byte	out	0 No error 1 Processor Reset 2 Invalid instruction 3 Invalid axis 4 Invalid parameter 5 Trace running 6 *reserved* 7 Block out of bounds 8 Trace buffer zero 9 Bad serial checksum Ah Not primary port Bh Invalid negative value Ch Invalid parameter change Dh Invalid move after limit condition Eh Invalid move into limit

Description:

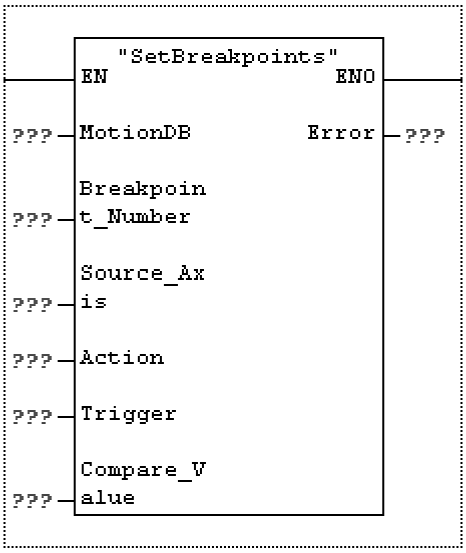
In this profile, the host specifies three parameters. The first is the 'master' axis number which is the axis that will be the source of position information used to drive the 'slave' axis, which is the axis in gear mode. The second is the gear source, which is either actual (the encoder position of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

A positive gear ratio value means that when the master axis actual or commanded position is increasing the slave commanded position will also increase. A negative gear ratio value has the opposite effect; increasing master position will result in decreasing slave axis commanded position.

The slave Axis is in Gearing Mode until a position or velocity Function is called.

2.12 *SetBreakpoint*

Call block FC 31:



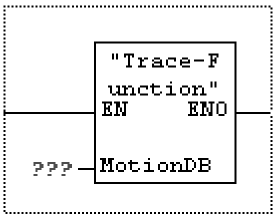
Parameters:

Name	**Type**	**IO**	**Description**
BreakpointNumber	int	in	0: Breakpoint 1 1: Breakpoint 2
Source_Axis	int	in	0: Axis 1 1: Axis 2
Action	int	in	(none) 0 Update 1 AbruptStop 2 SmoothStop 3 MotorOff 4
Trigger	int	in	(none) 0 PositiveCommandedPosition 1 NegativeCommandedPosition 2 PositiveActualPosition 3 NegativeActualPosition 4 CommandedPositionCrossed 5 ActualPositionCrossed 6 Time 7 EventStatus 8 ActivityStatus 9 Signal Ah
Compare_Value	dword	in	If the compare Value is a Position or Time, this value is in User units. The function calculates it in Module unit. If the Compare_value is a Register the value is a double word mask. See section below.

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

2.13 Commisioning Tool

Call block FC32:



Description:

If the commisioning Tool is in use, this FC 32 is to call once in each cycle. For each H320 Module this FC32 is to call with the motion DB Number from the first axis.

3 Datablocks and Parameters

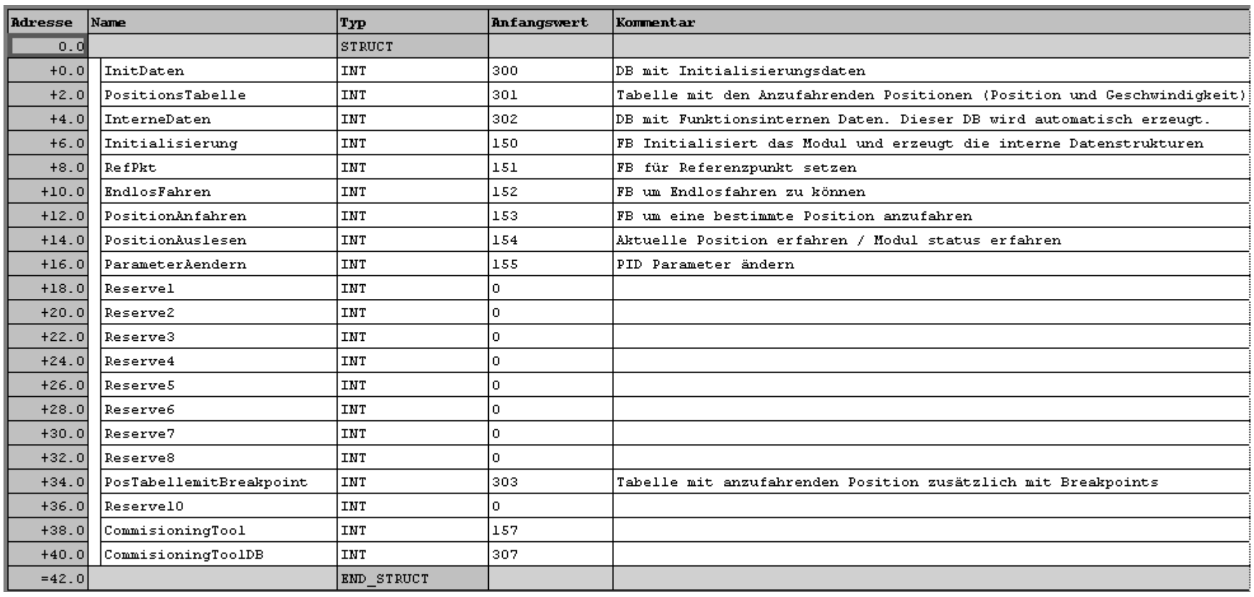
The following chapter describes the parameter Datablocks for the H320 module. For a precise functional description of modules, consult the relevant HW descriptions.

For all necessary DBs, there are UDTs that exist to generate these blocks.
DBs required internally (with which the user has nothing to do) are generated automatically if they are not already present.

3.1 Motion DB

The Motion DB serves to inform Standard Interfaces which HW functions must be called and where they can obtain the data.
If FBs 150 – 155 do not have to be renamed and if the data-block range from DB 300 - 317 has been reserved for these functions, the Motion DB can be generated from UDT110 and the specified values can be adopted. For both axis an individual Motion DB must be generated. If more than one H320 module is used, an individual Motion DB must be generated for each module and each axis. Likewise, a new DB range must be reserved for each axis and entered accordingly in the Motion DB.

The Motion DB is structured as follows:



Adresse	Name	Typ	Anfangswert	Kommentar
0.0		STRUCT		
+0.0	InitDaten	INT	300	DB mit Initialisierungsdaten
+2.0	PositionsTabelle	INT	301	Tabelle mit den Anzufahrenden Positionen (Position und Geschwindigkeit)
+4.0	InterneDaten	INT	302	DB mit Funktionsinternen Daten. Dieser DB wird automatisch erzeugt.
+6.0	Initialisierung	INT	150	FB Initialisiert das Modul und erzeugt die interne Datenstrukturen
+8.0	RefPkt	INT	151	FB für Referenzpunkt setzen
+10.0	EndlosFahren	INT	152	FB um Endlosfahren zu können
+12.0	PositionAnfahren	INT	153	FB um eine bestimmte Position anzufahren
+14.0	PositionAuslesen	INT	154	Aktuelle Position erfahren / Modul status erfahren
+16.0	ParameterAendern	INT	155	PID Parameter ändern
+18.0	Reserve1	INT	0	
+20.0	Reserve2	INT	0	
+22.0	Reserve3	INT	0	
+24.0	Reserve4	INT	0	
+26.0	Reserve5	INT	0	
+28.0	Reserve6	INT	0	
+30.0	Reserve7	INT	0	
+32.0	Reserve8	INT	0	
+34.0	PosTabellemitBreakpoint	INT	303	Tabelle mit anzufahrenden Position zusätzlich mit Breakpoints
+36.0	Reserve10	INT	0	
+38.0	CommisioningTool	INT	157	
+40.0	CommisioningToolDB	INT	307	
=42.0		END_STRUCT		

The first three values indicate the numbers of the following datablocks:

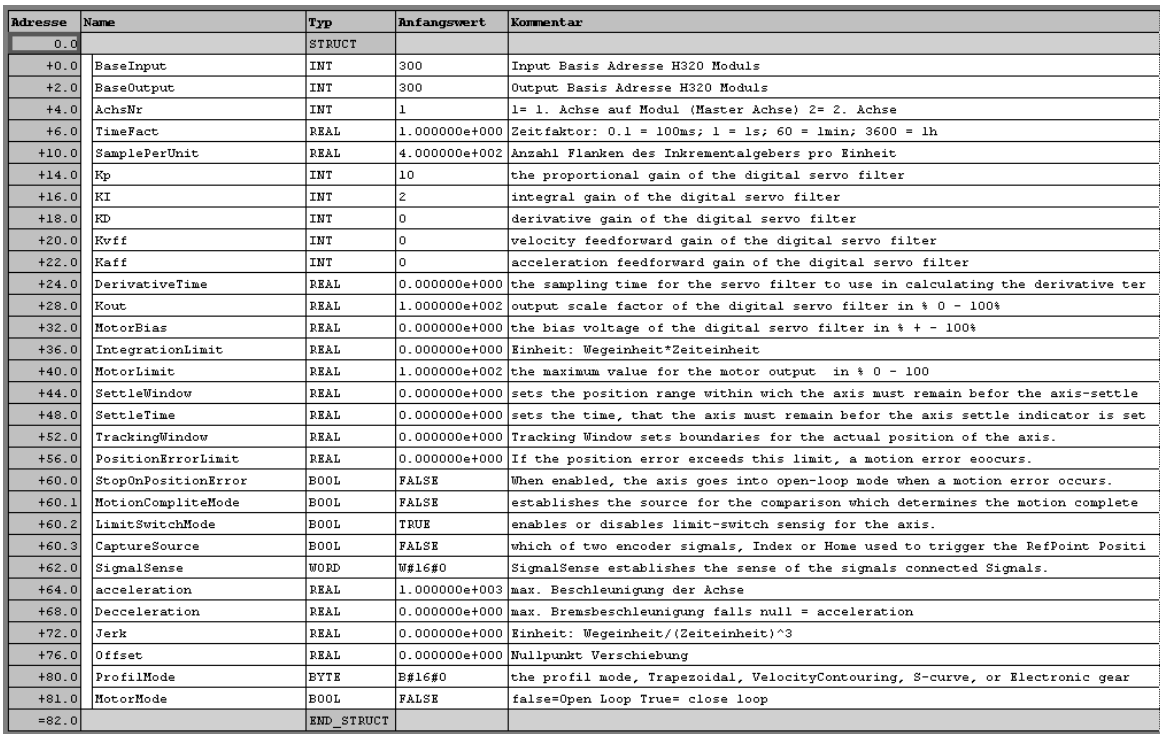
- InitData is a DB in which the user stores all data required for initialization. The precise structure can be consulted in the appropriate description.
- In the second DB, positions to be approached are entered.
- Interne Daten is the DB required for overall motion control. This DB is generated automatically upon initialization.
- In the DB at adress 34, are positions to be approached with the breakpoints

- In the DB at adress 40, are values for the commisioning tool. This DB is generatet automatical by the commisioning tool function.

The other 6 values are the HW-specific functions called by the functions presented in chapter 2. These FBs must be copied to the corresponding project. The user has nothing further to do with these functions.

3.2 InitDaten

The user stores the H320 module's HW data in this data-block. The data are read during initialization and the module is initialized. This block can be generated with the help of UDT300. The structure of this DB appears as follows:



Adresse	Name	Typ	Anfangswert	Kommentar
0.0		STRUCT		
+0.0	BaseInput	INT	300	Input Basis Adresse H320 Moduls
+2.0	BaseOutput	INT	300	Output Basis Adresse H320 Moduls
+4.0	AchsNr	INT	1	1= 1. Achse auf Modul (Master Achse) 2= 2. Achse
+6.0	TimeFact	REAL	1.000000e+000	Zeitfaktor: 0.1 = 100ms; 1 = 1s; 60 = 1min; 3600 = 1h
+10.0	SamplePerUnit	REAL	4.000000e+002	Anzahl Flanken des Inkrementalgebers pro Einheit
+14.0	Kp	INT	10	the proportional gain of the digital servo filter
+16.0	KI	INT	2	integral gain of the digital servo filter
+18.0	KD	INT	0	derivative gain of the digital servo filter
+20.0	Kvff	INT	0	velocity feedforward gain of the digital servo filter
+22.0	Kaff	INT	0	acceleration feedforward gain of the digital servo filter
+24.0	DerivativeTime	REAL	0.000000e+000	the sampling time for the servo filter to use in calculating the derivative ter
+28.0	Kout	REAL	1.000000e+002	output scale factor of the digital servo filter in % 0 - 100%
+32.0	MotorBias	REAL	0.000000e+000	the bias voltage of the digital servo filter in % + - 100%
+36.0	IntegrationLimit	REAL	0.000000e+000	Einheit: Wegeinheit*Zeiteinheit
+40.0	MotorLimit	REAL	1.000000e+002	the maximum value for the motor output in % 0 - 100
+44.0	SettleWindow	REAL	0.000000e+000	sets the position range within wich the axis must remain befor the axis-settle
+48.0	SettleTime	REAL	0.000000e+000	sets the time, that the axis must remain befor the axis settle indicator is set
+52.0	TrackingWindow	REAL	0.000000e+000	Tracking Window sets boundaries for the actual position of the axis.
+56.0	PositionErrorLimit	REAL	0.000000e+000	If the position error exceeds this limit, a motion error eoocurs.
+60.0	StopOnPositionError	BOOL	FALSE	When enabled, the axis goes into open-loop mode when a motion error occurs.
+60.1	MotionCompliteMode	BOOL	FALSE	establishes the source for the comparison which determines the motion complete
+60.2	LimitSwitchMode	BOOL	TRUE	enables or disables limit-switch sensig for the axis.
+60.3	CaptureSource	BOOL	FALSE	which of two encoder signals, Index or Home used to trigger the RefPoint Positi
+62.0	SignalSense	WORD	W#16#0	SignalSense establishes the sense of the signals connected Signals.
+64.0	acceleration	REAL	1.000000e+003	max. Beschleunigung der Achse
+68.0	Decceleration	REAL	0.000000e+000	max. Bremsbeschleunigung falls null = acceleration
+72.0	Jerk	REAL	0.000000e+000	Einheit: Wegeinheit/(Zeiteinheit)^3
+76.0	Offset	REAL	0.000000e+000	Nullpunkt Verschiebung
+80.0	ProfilMode	BYTE	B#16#0	the profil mode, Trapezoidal, VelocityContouring, S-curve, or Electronic gear
+81.0	MotorMode	BOOL	FALSE	false=Open Loop True= close loop
=82.0		END_STRUCT		

Parameters have the following meanings:
- BaseInput / BaseOutput: Input and output address of the H320 module. These addresses correspond to the values set with the SAIA HW configurator. These values are for both axis the same
- Axis Number: The number of the Axis on the module for whitch thes values are valid
- Sample per Unit: Number of positive and negative edges of incremental shaft encoder per user unit. How to determine this figure is described in the HW documentation.
- Time Factor: Velocity and acceleration are calculated according to this setting.
  For example: Enter 400.0 against "Sample per Unit" and 60 against "TimeFact". At velocity 1, the module adjusts the velocity so that 400 pulses are emitted per minute.
- KP is the proportional gain of the digital servo filter for the specified *axi*s.
- KI is the integral gain of the digital servo filter for the specified *axi*s.
- KD is the derivative gain of the digital servo filter for the specified axis.
- Kvff is the velocity feedforward gain of the digital servo filter for the specified *axi*s.
- Kaff is the acceleration feedforward gain of the digital servo filter for the specified *axi*s.

- DerivativeTime: the sampling time for the servo filter to use in calculating the derivative term for the specified *axi*s.
- Kout: the output scale factor of the digital servo filter for the specified axis. The default value of Kout is 100.0%.
- MotorBias: the bias voltage of the digital servo filter for the specified axis.
- Integration Limit: the integration-limit register of the digital servo filter for the specified *axi*s.
- MotorLimit: the maximum value for the motor output command allowed by the digital servo filter of the specified *axi*s. 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.
- SettleWindow: 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.
- SettleTime: 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.
- TrackingWindow: 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.
- PositionErrorLimit: the absolute value of the maximum position error allowable by the chipset for the specified *axi*s. 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.
- StopOnPositionError: 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.
- MotionCompliteMode: establishes the source for the comparison which determines the motion-complete status for the specified *axi*s. When set to commanded (0) 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 (1) 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.
- LimitSwitchMode: enables (On) or disables (Off) limit-switch sensing for the specified *axi*s. 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.

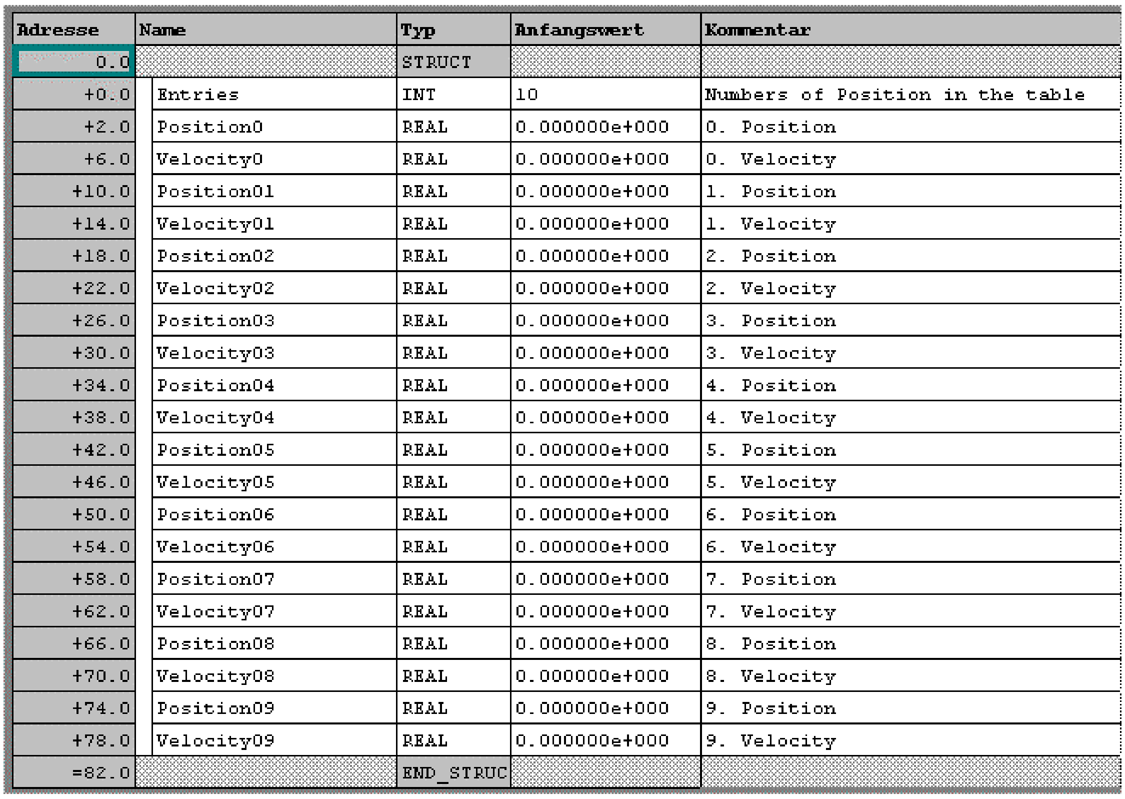
- Capture Source: determines which of two encoder signals, Index or Home, is used to trigger the high-speed capture of the actual axis position for the specified *axi*s.
- SignalSense: establishes the sense of the signals connected to the Signal Sense register by using a bitwise mask that corresponds to the bits of the Signal Status register, for the specified *axi*s. 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.

mask Encoder A	0001h
Encoder B	0002h
Encoder Index	0004h
Encoder Home	0008h
Positive limit	0010h
Negative limit	0020h
AxisIn	0040h
AxisOut	0400h

- Acceleration: Maximum acceleration of the axis. The unit is the result of Sample per Unit and TimeFact.
- Deceleration: the maximum acceleration buffer register for the specified *axi*s. This command is used with the Trapezoidal, Velocity Contouring, and S-curve profiling modes.
- Jerk: loads the jerk register in the parameter buffer for the specified *axi*s.
- Offset: If the zero point of the installation is not at the reference switch, this difference can be entered as an offset. The unit is derived from Sample per Unit.
- ProfilMode: sets the profile mode, selecting Trapezoidal (0), Velocity Contouring (1), S-curve (2), or Electronic gear (3) for the specified *axi*s.
- MotorMode: determines the mode of motor operation; open loop or closed loop for the specified *axi*s. When set to On, the axis is in *closed-loop* mode, and is controlled by the output of the servo filter. When 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.

3.3 *Approach position tables*

The positions in the table DB “PosTabDB” (DB301) can be entered via a Teach-In or manually. This DB must be created manually (with the help of UDT301). The structure of “PosTabDB” appears as follows:

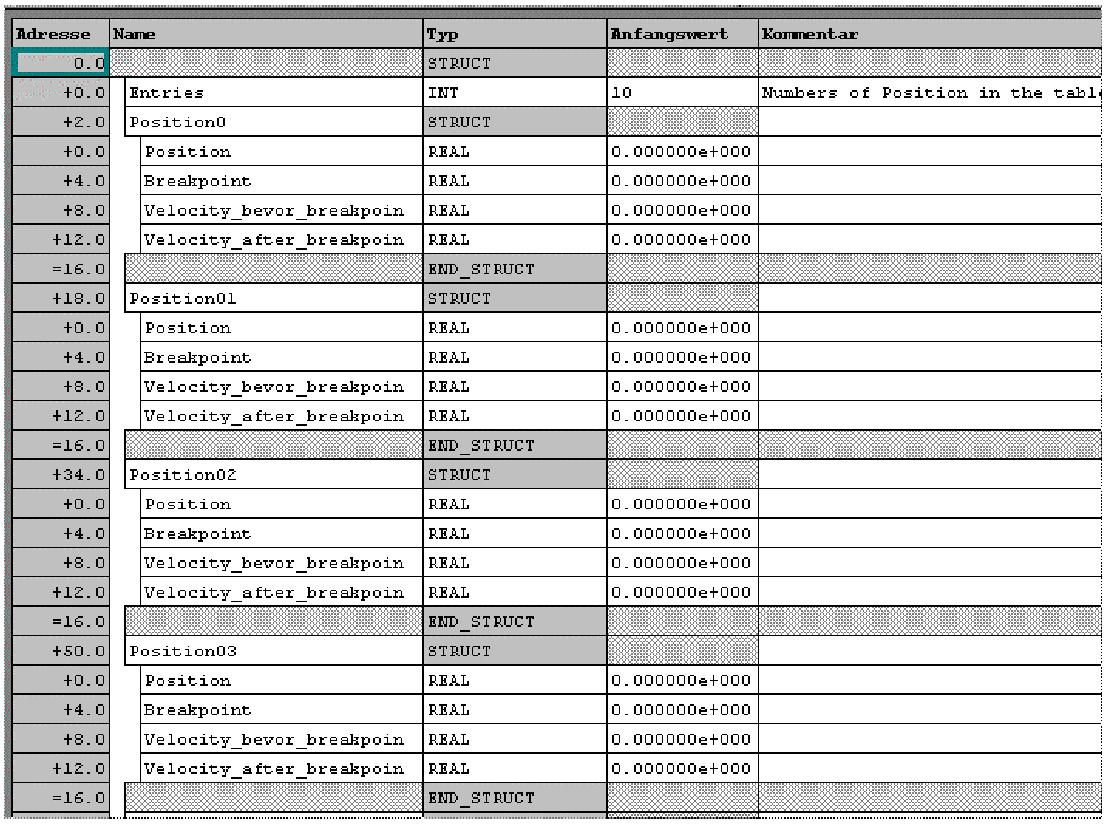


Adresse	Name	Typ	Anfangswert	Kommentar
0.0		STRUCT		
+0.0	Entries	INT	10	Numbers of Position in the table
+2.0	Position0	REAL	0.000000e+000	0. Position
+6.0	Velocity0	REAL	0.000000e+000	0. Velocity
+10.0	Position01	REAL	0.000000e+000	1. Position
+14.0	Velocity01	REAL	0.000000e+000	1. Velocity
+18.0	Position02	REAL	0.000000e+000	2. Position
+22.0	Velocity02	REAL	0.000000e+000	2. Velocity
+26.0	Position03	REAL	0.000000e+000	3. Position
+30.0	Velocity03	REAL	0.000000e+000	3. Velocity
+34.0	Position04	REAL	0.000000e+000	4. Position
+38.0	Velocity04	REAL	0.000000e+000	4. Velocity
+42.0	Position05	REAL	0.000000e+000	5. Position
+46.0	Velocity05	REAL	0.000000e+000	5. Velocity
+50.0	Position06	REAL	0.000000e+000	6. Position
+54.0	Velocity06	REAL	0.000000e+000	6. Velocity
+58.0	Position07	REAL	0.000000e+000	7. Position
+62.0	Velocity07	REAL	0.000000e+000	7. Velocity
+66.0	Position08	REAL	0.000000e+000	8. Position
+70.0	Velocity08	REAL	0.000000e+000	8. Velocity
+74.0	Position09	REAL	0.000000e+000	9. Position
+78.0	Velocity09	REAL	0.000000e+000	9. Velocity
=82.0		END_STRUC		

If more positions are required, the list can be enlarged according to the same pattern. The total number of entries must in this case be supplemented manually.
Entries are read by the function FB24 “PosTabAnfahren” and entered with FB25 “TeachIn”. These two functions check that the access number is lower than the total number of entries.

3.4 Approach position tables with breakpoint and two velocitys

The positions in the table DB "PosTabDB" (DB303) can be entered manually. This DB must be created manually (with the help of UDT303). The structure of "PosTabDB" appears as follows:



Adresse	Name	Typ	Anfangswert	Kommentar
0.0		STRUCT		
+0.0	Entries	INT	10	Numbers of Position in the tabl
+2.0	Position0	STRUCT		
+0.0	Position	REAL	0.000000e+000	
+4.0	Breakpoint	REAL	0.000000e+000	
+8.0	Velocity_bevor_breakpoin	REAL	0.000000e+000	
+12.0	Velocity_after_breakpoin	REAL	0.000000e+000	
=16.0		END_STRUCT		
+18.0	Position01	STRUCT		
+0.0	Position	REAL	0.000000e+000	
+4.0	Breakpoint	REAL	0.000000e+000	
+8.0	Velocity_bevor_breakpoin	REAL	0.000000e+000	
+12.0	Velocity_after_breakpoin	REAL	0.000000e+000	
=16.0		END_STRUCT		
+34.0	Position02	STRUCT		
+0.0	Position	REAL	0.000000e+000	
+4.0	Breakpoint	REAL	0.000000e+000	
+8.0	Velocity_bevor_breakpoin	REAL	0.000000e+000	
+12.0	Velocity_after_breakpoin	REAL	0.000000e+000	
=16.0		END_STRUCT		
+50.0	Position03	STRUCT		
+0.0	Position	REAL	0.000000e+000	
+4.0	Breakpoint	REAL	0.000000e+000	
+8.0	Velocity_bevor_breakpoin	REAL	0.000000e+000	
+12.0	Velocity_after_breakpoin	REAL	0.000000e+000	
=16.0		END_STRUCT		

If more positions are required, the list can be enlarged according to the same pattern. The total number of entries must in this case be supplemented manually.
Entries are read by the function FB28 "PosTabAnfahren". These function check that the access number is lower than the total number of entries.

4 Trajectory Generation

4.1 Trajectories, profiles, and parameters

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

Throughout this manual various command mnemonics will be shown to clarify chipset usage or provide specific examples. See the previous sections for more information on host commands, nomenclature, and syntax.

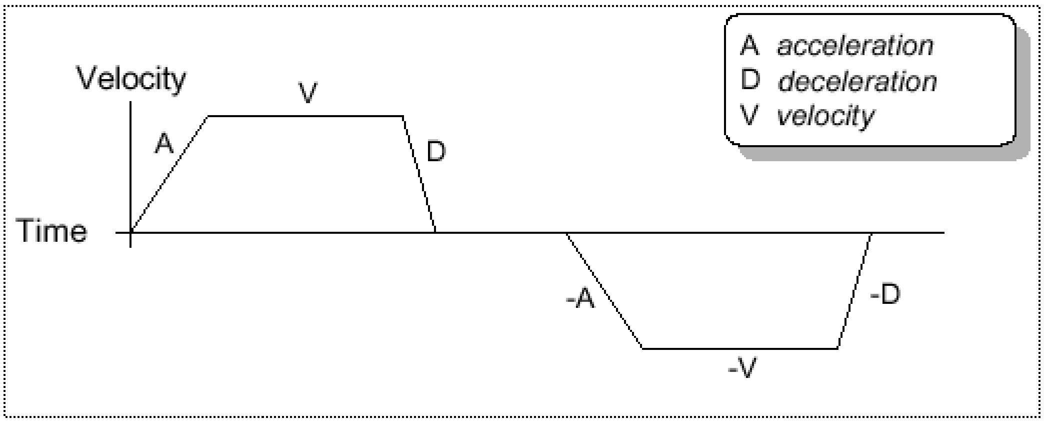
The specific profile that is created by the Navigator depends on several factors including the presently selected profile mode, the presently selected profile parameters, and other system conditions such as whether a motion stop has been requested. Four trajectory profile modes are supported: S-curve point to point, Trapezoidal point to point, Velocity contouring and Electronic gearing. The operation of these profile modes will be explained in detail in subsequent sections. The commands used to select the profile mode are FC's for Motion in velocity Mode, FC for Motion in position Mode and FC for Gearing Mode. In the Initial DB (Datenbyte 80 Profil Mode) it is possible to choose the profile for the positioning Mode (S-curve or Trapezoidal).

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

With one exception, Navigator motion processors can switch from one profile to another while an axis is in motion. The exception: when switching to the S-curve point-to-point profile from any other profile, the axis must be at rest.

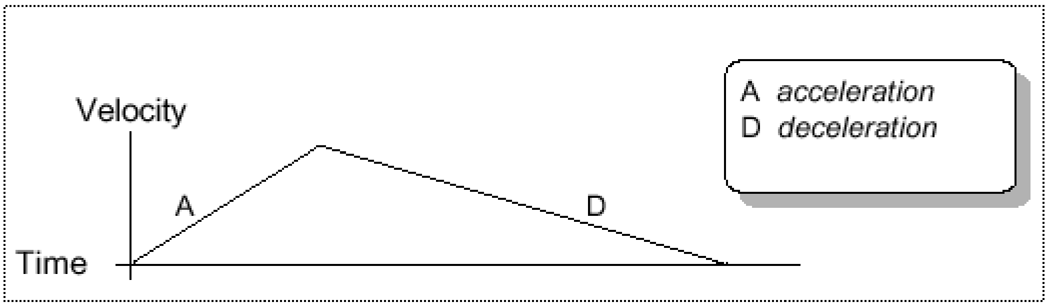
4.2 Trapezoidal point-to-point profile

For this profile, the host specifies an initial acceleration and deceleration (InitialDB DBd64 and DBD 68), a velocity, and a destination position. The profile gets its name from the resulting curve: the axis accelerates linearly (at the programmed acceleration value) until it reaches the programmed velocity. It continues in motion at that velocity, then decelerates linearly (using the deceleration value) until it stops at the specified position.



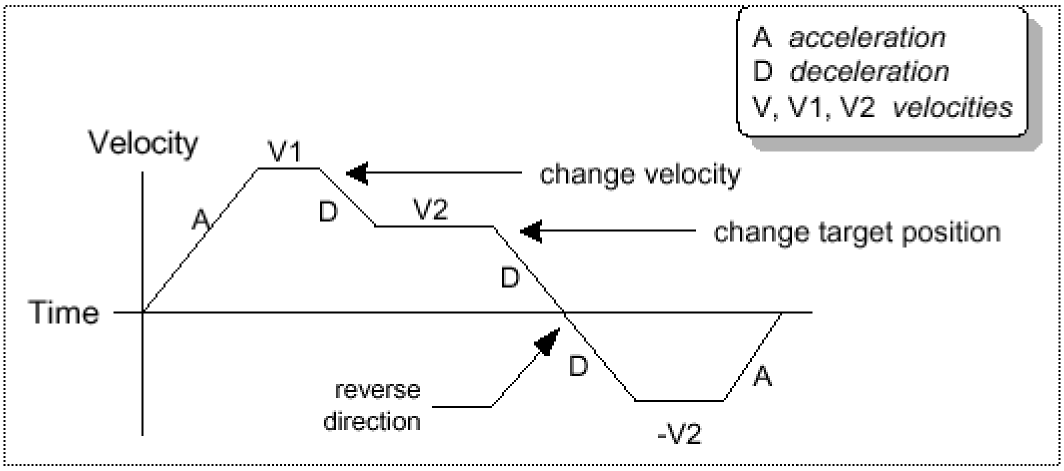
Simple trapezoidal point-to-point profil 1

If deceleration must begin before the axis reaches the programmed velocity, the profile will have no constant velocity portion, and the trapezoid becomes a triangle.



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

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



Complex trapezoidal profile, showing parameter changes

It is acceptable to change any of the profile parameters while the axis is moving in this profile mode. The profile generator will always attempt to remain within the legal bounds of motion specified by the parameters. If, during the motion, the destination position is changed in such a way that an overshoot is unavoidable, the profile generator will decelerate until stopped, then reverse direction to move to the specified position. Note that since the direction of acceleration/deceleration is fixed at the start of the move, the deceleration value will be used when ramping up velocity for the final move to the destination position. This is shown in Figure abouve.

If a deceleration value of 0 (zero) is programmed (or no value is programmed leaving the chipset's default value of zero), then the value specified for acceleration will automatically be used to set the magnitude of deceleration.

4.3 S-curve point-to-point profile

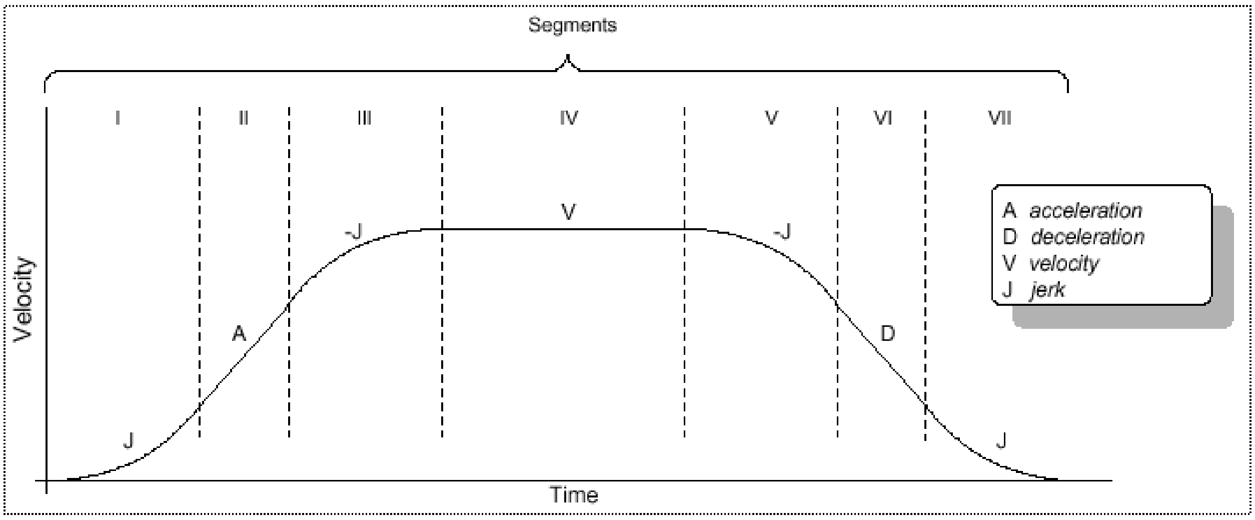
The following table summarizes the host specified profile parameters for the S-Curve point to point profile mode:

Position	Set with Positioning FC's
Velocity	Set with Positioning FC's
Acceleration	Set during Initialisation or with the Change Parameter FC
Deceleration	Set during Initialisation or with the Change Parameter FC
Jerk	Set during Initialisation or with the Change Parameter FC

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

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

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

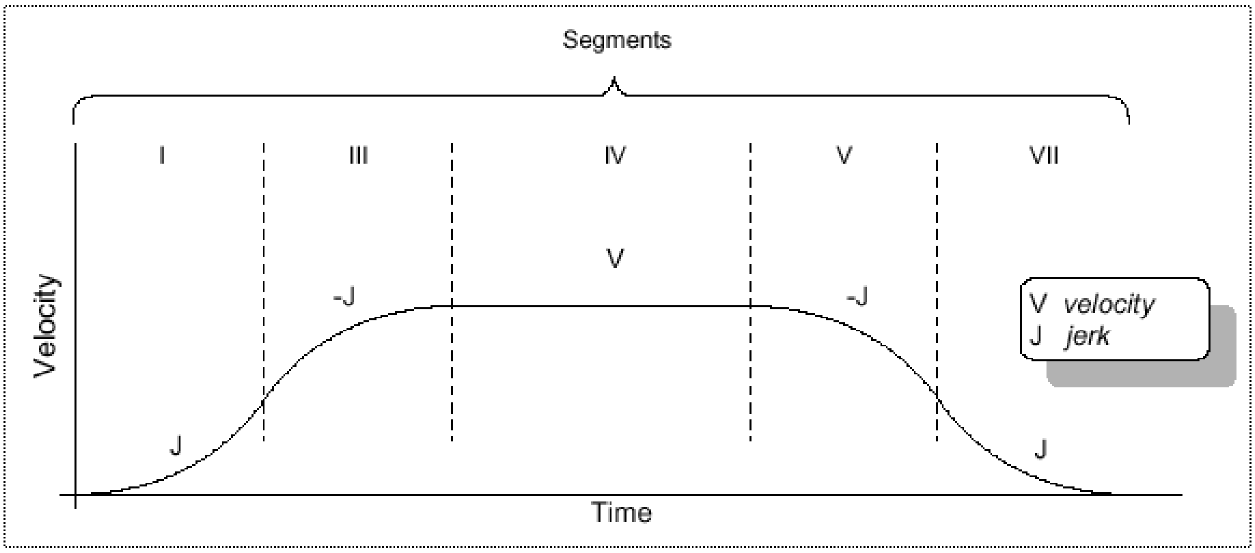


S-curve profile

Figure above shows a typical S-curve profile. In Segment I, the S-curve profile drives the axis at the specified jerk (J) until the maximum acceleration (A) is reached. The axis continues to accelerate linearly (jerk = 0) through Segment II. The profile then applies the negative value of the jerk to reduce acceleration to 0 during Segment III. The axis is now at maximum velocity

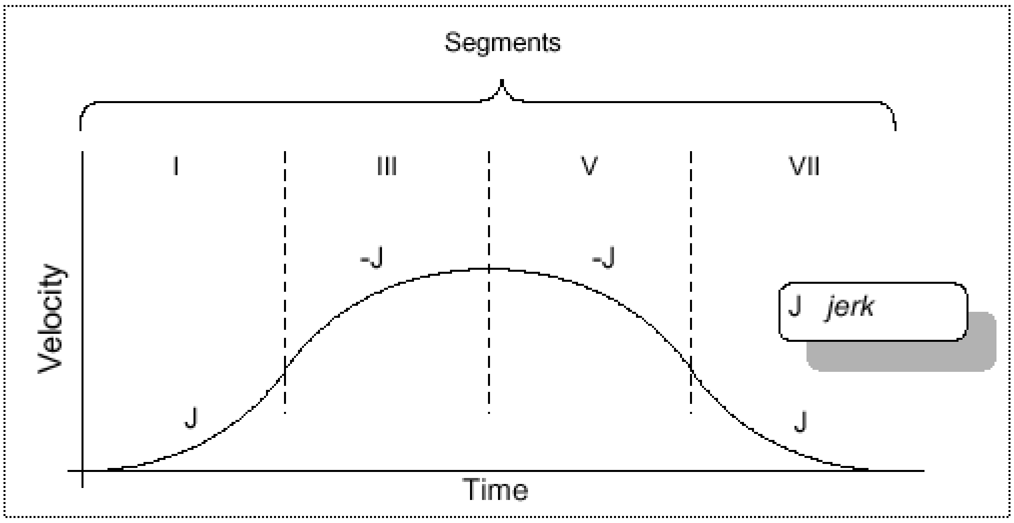
(V), at which it continues through Segment IV. The profile will then decelerate in a manner similar to the acceleration stage, using the jerk value first to reach the maximum deceleration (D), and then to bring the axis to a halt at the destination.

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



S-curve that doesn't reach maximum acceleration

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



S-curve with no maximum-velocity segment

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

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

4.4 Velocity-contouring profile

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

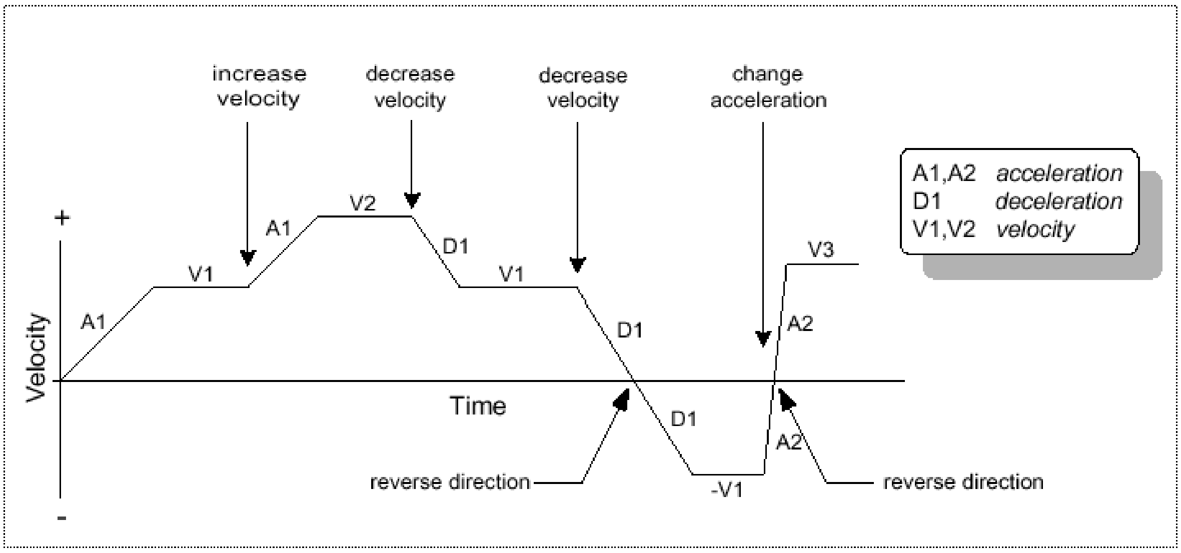
Velocity	Set with Positioning FC's
Acceleration	Set during Initialisation or with the Change Parameter FC
Deceleration	Set during Initialisation or with the Change Parameter FC

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

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

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

The trajectory is executed by continuously accelerating the axis at the specified rate until the velocity is reached. The axis starts decelerating when a new velocity is specified which has a smaller value (in magnitude) than the present velocity, or has a sign that is opposite to the present direction of travel.



Velocity-contouring profile

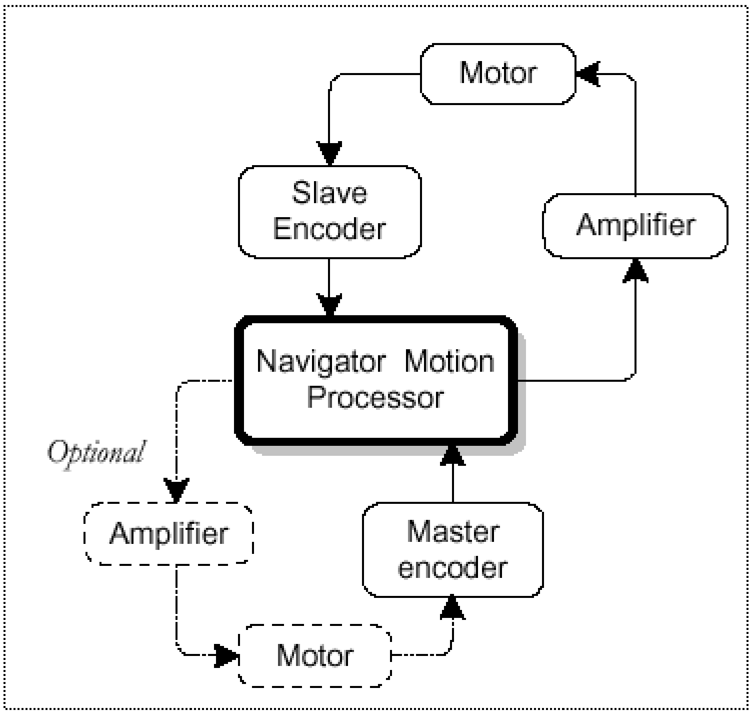
A simple velocity-contouring profile looks just like a simple trapezoidal point-to-point profile.

Figure above illustrates a more complicated profile, in which both the velocity and the direction of motion change twice.

4.5 *Electronic-gear profile*

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

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



Axes set up for electronic-gear profile

A positive gear ratio value means that when the master axis actual or commanded position is increasing the slave commanded position will also increase. A negative gear ratio value has the opposite effect; increasing master position will result in decreasing slave axis commanded position.

For example, let us assume the slave axis is axis #0 (axes are counted 0, 1) and the master axis are set to axis #1. Also assume the source will be 'actual' with a gear ratio of -1/2. Then for each positive encoder count of axis 1, axis 0 commanded position will decrease in value by 1/2 count, and for each negative encoder count of axis 1, axis 0 commanded position will increase in value by 1/2 count.

The electronic gear profile requires both two axes to be enabled.

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

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

4.6 Motor Mode

The Module chipsets support a programmable motor mode that can enable and disable the profile generator, and can set the chipsets to open loop mode or closed loop mode.

If the motor mode is set to on then the trajectory generator is active. If the motor mode is set to off then the profile generator is disabled. In addition, if the motor mode is set off then the chipset enters open loop mode which means the servo filter is disabled and the motor command (the current output level requested by the chipset) is determined manually by the host using the change parameter (MotorCommand). If the motor mode is set on then the motor command is determined by the servo loop.

The most common use of the motor mode in anything other than the standard "on" state is after a motion error. In the case of a motion error (and if auto stop is enabled) then the chipset will set the motor mode off automatically, thereby placing it in a safe state where no further motion can occur until the host explicitly restores the motor mode to the on condition. For more information on motion errors see section 7.1.

It may also be useful to set the motor mode to off for purposes of amplifier calibration.

5 The Servo Loop

5.1 *Overview*

A servo loop is used as part of the basic method of determining the motor command output. The function of the servo loop is to match as closely as possible the commanded position, which comes from the trajectory generator, and the actual motor position.

To accomplish this the profile generator *commanded value* is combined with the actual encoder position to create a position error, which is then passed through a digital PID-type servo filter. The scaled result of the filter calculation is the *motor comman*d, which is output to the motor amplifier.

5.1.1 PID loop algorithm

The servo filter used is a proportional-integral-derivative (PID) algorithm, with velocity and acceleration feed-forward terms and an output scale factor. An integration limit provides an upper bound for the accumulated error. An optional bias value can be added to the filter calculation to produce the final motor output command. A limiting value for the filter output provides additional constraint.

The PID + V_{ff}+A_{ff} formula, including the scale factor and bias terms, is as follows:

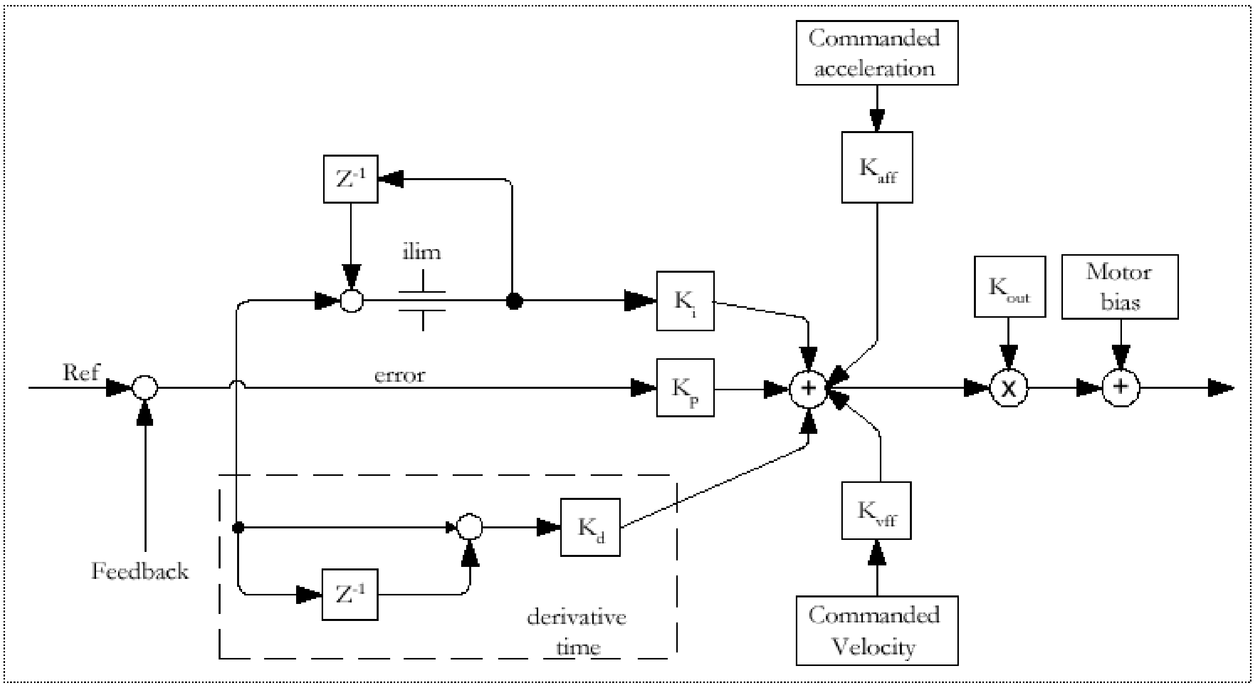
$$Output_n = \left[K_P E_n + K_d \left(E_K - E_{(K-1)} \right) + K_i \times \sum_{J=0}^{n} Ej + K_{vff}(CmdVel) + K_{Aff}(CmdAccel) \right] \times K_{out} + Bias$$

where:

I_{lim}	is the Integration Limit
E_n	are the accumulated error terms
K_I	is the Integral Gain
K_d	is the Derivative Gain
K_p	is the Proportional Gain
K_{aff}	is the Acceleration feed-forward
K_{vff}	is the Velocity feed-forward
Bias	is the DC motor offset
K_{out}	is the scale factor for the output command.

All filter parameters, the motor output command limit, and the motor bias are programmable, so that the filter may be fine-tuned to any application.

The structure of the digital filter is shown in Figure below.



5.1.2 Motor bias

When an axis is subject to a net external force in one direction (such as a vertical axis pulled downward by gravity), the servo filter can compensate for it by adding a constant DC bias to the filter output.

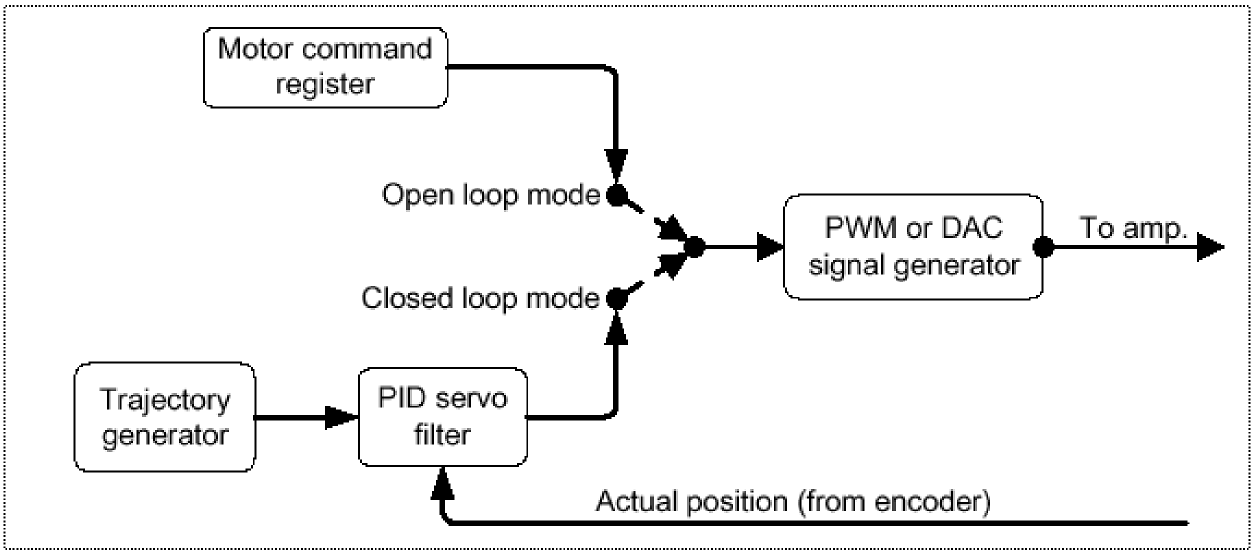
5.1.3 Output limit

The motor output limit prevents the filter output from exceeding a boundary magnitude in either direction. If the filter produces a value greater than the limit, the motor command takes the limiting value.

The motor limit applies only in closed-loop mode. It does not affect the motor command value set by the host in open-loop mode (see next section for more information on open and closed loop operations).

5.2 Closed-loop and open-loop control modes

In a previous section motor mode is discussed. For all Navigator chipsets setting the motor offhas the effect of disabling the trajectory generator. In addition however, turning the motor off, or having the motor be turned off automatically by the chipset via a motion error, places the chipset into what is known as 'open loop' mode. In open loop mode the servo filter does not operate and the motor command output value is set manually by the host changing the MotorCommand value. With the motor 'on' the chipset is in 'closed loop' mode and the motor command value is controlled automatically by the servo filter. Figure below shows the control flow for open and closed loop operation.



Closed-loop mode is the normal operating mode. Open-loop mode is typically used when one or more axes require torque control only, or when the amplifier must be calibrated.

Limit switches do not function in open-loop mode.

5.2.1 Motor bias in open-loop mode

The motor bias applies at all times when operating in closed-loop mode. If the axis is switched to open-loop mode, the bias value continues to be output to the motor, to prevent the axis from suddenly lurching in the direction of the external force. Once the host issues a new motor command, however, its value supersedes the bias output, which no longer has any effect. As soon as the axis returns to closed-loop mode, the previous bias value is reinstated.

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

6 Parameter update and breakpoints

6.1 *Parameter buffering*

Various parameters must be specified to the chipset for an axis to be controlled correctly. In some cases it may be desired that a set of parameters take effect at the exact same time to facilitate precise synchronized motion.

To support this all profile parameters and several other types of parameters such as servo parameters

are loaded into the chipset using a buffered scheme. These buffered commands are loaded into an area of the chipset that does not affect the actual chipset behavior until a special event known as an Update occurs. An Update causes the buffered registers to be copied to the active registers, thereby causing the chipset to act on the new parameters.

For example if you use the FC 23 and set only the load flag, Function loads values in but they will not become active (take effect) until an Update is given:

After this Function is completed the buffered registers for these parameters are loaded into the chipset but the trajectory generator module still operates on whatever the previous trajectory profile mode and parameters were. Only when an Update occurs will the trajectory profile mode actually be changed to trapezoidal and the specified parameters loaded into the trajectory generator, causing the trajectory generator to start the programmed motion.

6.1.1 Updates

There are three different ways that an Update can occur. They are listed below:

1) Update command - The simplest way is to give an Update command. This causes the parameters for the programmed axis to be updated immediately. In the motion Functions the Update command occur when the start flag is set.

2) MultiUpdate command - The multiple axis instantaneous update, which is specified using the MultiUpdate command, causes multiple axes to be updated simultaneously. This can be useful when synchronized multi-axis profiling is desired. This command takes a 1-word argument that consists of a bit mask, with 1 bit assigned to each axis. Executing this command has the same effect as sending a set of Update commands to each of the individual axes selected in the MultiUpdate command mask.

3) Breakpoints - There is a very useful facility supported by the chipset that can be programmed to generate an Update command automatically when a pre-programmed condition becomes true. This feature is known as the breakpoint facility, and it is useful for performing operations such as "automatically change the velocity when a particular position is reached", or "stop the axis abruptly when a particular external signal goes active." Breakpoints are discussed in more detail in section 6.2.

Whichever Update method is used, at the time the update occurs, all of the buffered registers and commands will be copied to the active registers. However, depending on which calculations have already been performed in the servo loop, these values may not be used until the next cycle. Before the Update occurs, sending buffered commands will have no effect on the system behavior. In addition to profile generation most servo parameter commands are buffered, and some other commands are buffered. Following is a complete list of buffered values and commands.

6.2 *Breakpoints*

Breakpoints are a convenient way of programming a chipset event upon some specific condition. Depending on the breakpoint instruction's arguments, a breakpoint can cause an update; an abrupt stop followed by an update; a smooth stop followed by an update; a motor-off followed by an update (more on this function in a later section); or no action whatsoever.

Each Navigator axis has two breakpoints that may be programmed for it. So two completely separate conditions may be monitored and acted upon. These two breakpoints are known as breakpoint 1 and breakpoint 2.

6.2.1 Defining a breakpoint, Overview

Each breakpoint has five components: the breakpoint axis, the source axis for the triggering event, the event itself, the action to be taken and the comparison value.

The *breakpoint axis* is the axis on which the specified action is to be taken. The *source axis* is the axis on which the triggering event is located. It can be the same as or different than the breakpoint axis. Any number of breakpoints may use the same axis as a *source axi*s.

The *trigger* is the event that causes the breakpoint. The *action* is the sequence of operations executed by the chipset when the breakpoint is triggered. After a breakpoint is triggered the *action* is performed on the breakpoint axis. The *comparison value* is used in conjunction with the action to define the breakpoint event. Altogether these parameters provide great flexibility in setting breakpoint conditions. By combining these components, almost any event on any axis can cause a breakpoint.

The command used to send the breakpoint axis, the trigger, the source axis and the action is SetBreakpoint. To retrieve these same values the command GetBreakpoint is used. To set the comparison value the command SetBreakpointValue is used. This comparison value can be retrieved using the command GetBreakpointValue. For each of these commands the breakpoint number (1 or 2) must be specified.

6.2.2 Breakpoint triggers

The Navigator motion processors support the following breakpoint trigger conditions:

Trigger Condition	**Level or Threshold**	**Description**
GreaterOrEqual CommandedPosition	threshold	Is satisfied when the current commanded position is equal to or greater than the programmed compare value.
LesserOrEqua lCommandedPosition	threshold	Is satisfied when the current commanded position is equal to or less than the programmed compare value.
GreaterOrEqual ActualPosition	threshold	Is satisfied when the current actual position is equal to or greater than the programmed compare value.
LesserOrEqual ActualPosition	threshold	Is satisfied when the current actual position is equal to or less than the programmed compare value.
CommandedPosition Crossed	threshold	Is satisfied when the current commanded position crosses (is equal to) the programmed compare value.
ActualPositionCrossed	threshold	Is satisfied when the current actual position crosses (is equal to) the programmed compare value.

Time	threshold	Is satisfied when the current chipset time (in number of cycles since power-up) is equal to the programmed compare value.
EventStatus	level	Is satisfied when the EventStatus register matches bit mask and high/low pattern in programmed compare value.
ActivityStatus	level	Is satisfied when the ActivityStatus register matches bit mask and high/low pattern in programmed compare value.
SignalStatus	level	Is satisfied when the SignalStatus register matches bit mask and high/low pattern set in programmed compare value.
none		Disables any previously set breakpoint.

If "none" is selected for the breakpoint trigger then this effectively means that that breakpoint is inactive. Only one of the above triggers can be selected at a given time. For a description of level triggered breakpoints refer to section 6.2.4.

6.2.3 Threshold-triggered breakpoints

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

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

6.2.4 Level-triggered breakpoints

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

The trigger mask determines which bits of the selected status register are enabled for the breakpoint. A 1 in any position of the trigger mask enables the corresponding status register bit to trigger a breakpoint, a 0 in the trigger mask disables the corresponding status register bit. If more then one bit is selected, then the breakpoint will be triggered when any selected bit enters the specified state.

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

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

6.2.5 Breakpoint actions

Once a breakpoint has been triggered, the chipset can be programmed to perform one of the following instruction sequences:

Action	**Command Sequence Executed**
None	No commands executed.
Update	Update *axis*.
Abrupt Stop	SetStop *axis*, AbruptStop Update *axis*
SmoothStop	SetStop *axis*, SmoothStop Update *axis*
MotorOff	SetMotorMode *axi*s, Off Update *axis*

Regardless of the host’s action, once a breakpoint condition has been satisfied, the Event Status bit corresponding to the breakpoint is set and the breakpoint is deactivated.

6.2.6 Breakpoint Examples

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

7 Status Registers

7.1 *Overview*

The Navigator Motion Processor can monitor almost every aspect of the motion of an axis. There are numerous numerical registers that can be queried to determine the current state of the chipset, such as the current actual position, the current commanded position, etc. In addition to these numerical registers there are three bit-oriented status registers which provide a continuous report on the state of a particular axis. These status registers conveniently combine a number of separate bit-oriented fields for the specified axis. These three 16-bit registers are Event Status, Activity Status, and Signal Status.

The host may query these three registers, or the contents of these registers may be used in breakpoint operations to define a triggering event such as "trigger when bit 8 in the signal status register goes low." These registers are also the source of data for the AxisOut (see Section 9.2) mechanism, which allows any bit within these three registers to be output as a hardware signal.

7.1.1 Event Status register

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

The EventStatus register is defined in the table below:

Bit	**Name**	**Description**
0	Motion complete	Set when a trajectory profile completes. The motion being considered complete may be based on the commanded position or the actual encoder position. See section 7.3 for more details.
1	Position wraparound	Set when the actual motor position exceeds 7FFFFFFFh (the most positive position) and wraps to 80000000h (the most negative position), or vice versa.
2	Breakpoint 1	Set when breakpoint #1 is triggered.
3	Capture received	Set when the high-speed position capture hardware acquires a new position value.
4	Motion error	Set when the actual position differs from the commanded position by an amount more then the specified maximum position error.
5	Positive limit	Set when a positive limit switch event occurs.
6	Negative limit	Set when a negative limit switch event occurs.
7	Instruction error	Set when an instruction error occurs.
8-13	*Reserved*	*May contain 1 or 0.*
14	Breakpoint 2	Set when breakpoint #2 is triggered.
15	*Reserved*	*May contain 0 or 1.*

The command GetEventStatus (or FC 26 read Position and Status) returns the contents of the Event Status register for the specified axis.

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

7.1.2 Instruction error

Bit 7 of the event status register indicates an instruction error. Such an error occurs if an otherwise valid instruction or instruction sequence is sent when the Navigator's current operating state makes the instructions invalid. Instruction errors occur at the time of an update.

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

7.1.3 Activity Status register

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

The ActivityStatus register is defined in the table below:

Bit	**Name**	**Description**
0	*Reserved*	*May contain 0 or 1.*
1	At maximum velocity	Set (1) when the commanded velocity is equal to the maximum velocity specified by the host. Cleared (0) if it is not. This bit functions only when the profile mode is trapezoidal, velocity contouring, or S-curve. It will not function when the chipset is in electronic gearing mode.
2	Position tracking	Set (1) when the servo is keeping the axis within the Tracking Window. Cleared (0) when it is not. See Section 6.2.
3-5	Current profile mode	These bits indicate the profile mode currently in effect, which might be different than the value set using the SetProfileMode command if an Update command has not yet been issued. The 3 bits define the current profile mode as follows: bit 5 / bit 4 / bit 3 / Profile Mode 0 / 0 / 0 / trapezoidal 0 / 0 / 1 / velocity c ontouring 0 / 1 / 0 / S-curve 0 / 1 / 1 / electronic g ear.
6	Axis settled	Set (1) when the axis has remained within the Settle Window for a specified period of time. Cleared (0) if it has not. See Section 8.5.
7	*Reserved*	*May contain 0 or 1.*
8	Motor mode	Set (1) when the motor is "on", cleared (0) when the motor is off."

		When the motor is on this means that the chipset can perform trajectory operations, and the chipset is in closed loop mode and the servo loop is operating. When the motor is off this means trajectory operations cannot be performed and the chipset is in open loop mode and the servo loop is disabled. The SetMotorMode command is normally used to select the mode of the motor, however the chipset will reset the mode to 0 (turn the motor off) if a motion error occurs.
9	Position capture	Set (1) when a new position value is available to read from the high speed capture hardware. Cleared (0) when a new value is not yet been captured. While this bit is set, no new values will be captured. The command GetCaptureValue retrieves a captured position value and clears this bit, allowing additional captures to occur.
10	In-motion indicator	Set (1) when the trajectory profile commanded position is changing. Cleared (0) when the commanded position is not changing. The value of this bit may or may not correspond to the value of the motion complete bit of the event status register depending on whether the motion complete mode has been set to commanded or actual.
11	In positive limit	Set (1) when the motor is in a positive limit condition. Cleared (0) when it is not.
12	In negative limit	Set (1) when the motor is in a negative limit condition. Cleared (0) when it is not.
13-15	S-curve segment	Indicates the S-curve segment number using values 1-7 to indicate S-curve phases 1-7 as shown in the S-curve trajectory section of this manual. A value of 0 in this field indicates the trajectory is not in motion. This field is undefined for profile modes other than S-curve, and may contain 0's or 1's.

The command GetActivityStatus returns the contents of the Activity Status register for the specified axis. Or The Motion Function FC26 Get Position and Status give the Activity Status back.

7.1.4 Signal Status

The signal status register provides real time signal levels for various chipset I/O pins. The SignalStatus register is defined in the table below:

Bit	**Name**	**Description**
0	A encoder	A encoder A signal of quadrature encoder input.
1	B encoder	B signal of quadrature encoder input.
2	Index encoder	Index signal of quadrature encoder input.
3	Home	Home position capture input.

4	Positive limit	Positive limit switch input
5	Negative limit	Negative limit switch input.
6	AxisIn	Generic axis input signal
7-9	*Reserved*	
10	AxisOut	Programmable axis output signal.
11-15	Reserved	

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

The bits in the signal status register always represent the actual hardware levels on the corresponding pins. A 1 in this register represents an electrically high value on the pin; a 0 indicates an electrically low level. The state of the signal sense mask affects the value read using GetSignalStatus (see the next section for more information on the signal sense mask).

7.1.5 Signal Sense Mask

The bits in the signal status register represent the high/low state of various signal pins on the chipset. How these signal pins are interpreted by the chipset may be controlled using a signal sense mask. This is useful for changing the interpretation of input signals to match the signal interpretation of the user's hardware.

The SignalSense mask register is defined in the table below:

Bit	**Name**	**Interpretation**
0	A encoder	Set (1) to invert quadrature A input signal. Clear (0) for no inversion.
1	B encoder	Set (1) to invert quadrature B input signal. Clear (0) for no inversion.
2	Index encoder	Set (1) to invert Index signal. Clear (0) for no inversion.
3	Home	Set (1) to invert home signal. Clear (0) for no inversion.
4	Positive limit	Set (1) for active high interpretation of positive limit switch, meaning positive limit occurs when signal is high. Clear (0) for active low.
5	Negative limit	Set (1) for active high interpretation of negative limit switch, meaning negative limit occurs when signal is high. Clear (0) for active low.
6	AxisIn	Set (1) to invert AxisIn signal. Clear (0) for no inversion.
7-9	*Reserved*	
10	AxisOut	Set (1) to invert AxisOut signal. Clear (0) for no inversion.
11-15	Reserved	

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

8 Monitoring Motion Performance

8.1 Motion Error

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

To detect this condition, thereby increasing safety and equipment longevity, the Navigator chipsets include a programmable maximum position error.

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

At the moment a motion error occurs, several events occur simultaneously. The Motion Error bit of the event status word is set. If automatic stop on motion error is enabled the motor is set off, which has the effect of disabling the trajectory generator. It has the additional effect of disabling the servo loop and placing the chipset into open loop mode. This is the equivalent to a SetMotorMode *axi*s, Off command.

To recover from a motion error that results in the motor being turned off, the cause of motion error should be determined and the problem corrected (this may require human intervention). The host should then issue a SetMotorMode *axi*s, On command.

After the above sequence, the axis will be at rest, with the motor on.

If automatic stop on motion error is not set then only the motion error status bit is set, the motor is not stopped and no recovery sequence is required to continue operating the chipset. Nevertheless, for safety reasons, the user may still want to manually shut down the motion and explore the cause of the motion error.

8.1.1 Automatic Stop On Motion Error

Because a motion error may indicate a serious problem it is useful to have the axis automatically stop until the problem can be addressed and rectified. This feature is known as automatic stop on motion error.

The command SetAutoStopMode controls the action that will be taken if a motion error occurs. The options for this command are disable and enable.

If autostop is enabled, when a motion error occurs a SetMotorMode Off command is generated which has the effect of instantaneously halting the trajectory generator and (for servo chipsets) putting the chipset into open loop.

For stepping chipsets the motor will stop moving immediately (same as an abrupt stop). For servo chipsets the trajectory will stop instantly but because motor off means open loop mode the motor will coast to a stop not under servo control in an amount of time determined by the velocity at the time of motion error and the inertia of the system.

For the servo chipsets, transitioning to open loop mode can be dangerous if the axis is oriented vertically, because the axis may fall downward due to gravity if not supported by the servo feedback. This problem can be rectified by use of the motor bias value, which is discussed in Section 6.1.2.

The motor bias is a fixed, open-loop command to the motor, which is added to the PID filter output. Upon a motion error with automatic stop enabled, the motor bias will be output even while the chipset is in open loop mode. Thus, with a properly set motor bias, if the axis experiences a motion error and transitions to open loop mode, the motor bias can prevent the axis from falling.

Caution: Because the motor bias value is applied 'open-loop' to the axis, care should be taken when setting its value.

8.2 Tracking window

The Navigator chipsets provide a programmable *tracking window* that can be used to monitor servo performance outside the context of a motion error. The tracking window functions similarly to the motion error, in that there is a programmable position error limit within which the axis must stay. Unlike the motion error facility however, if the axis moves outside of the tracking window the axis is not stopped. The tracking window is useful when external processes depend on the motor tracking the desired trajectory within some range. Alternatively the tracking window can be used as an early warning for performance problems that do not yet qualify as a motion error.

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

When the position error is less than or equal to the window value the tracking bit in the activity status register is set. When the position error exceeds the tracking window value, the tracking bit is cleared. See Figure The Tracking Window.

8.3 Motion Complete Indicator

In many cases it is useful to have the chipset signal that a given motion profile is complete. This function is available in the motion complete indicator.

The motion complete indicator appears in bit 0 of the event status register. As are all bits in the event status register, the motion complete bit is set by the chipset and cleared by the host. When a motion has been completed, the chipset sets the motion complete bit on. The host can examine this bit by polling the event status register or the host can program some automatic follow-on function using a breakpoint, a host interrupt, or an AxisOut signal. In either case, once the host has recognized that the motion has been completed the host should clear the motion complete bit, enabling the bit to indicate the end of motion for the next move.

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

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

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

- The profile generator (commanded) motion is complete.
- The difference between the actual position and the commanded position is less than or equal to the value of the settle window. The settle window is set using the command SetSettleWindow. This same value may be read back using the command GetSettleWindow.
- The above two conditions have been met continuously for the last N cycles, where N is the programmed settle time. The settle time is set using the command SetSettleTime. This same value may be read back using the command GetSettleTime.

At the end of the trajectory profile the cycle timer for the actual-based motion complete mechanism is cleared, so there will always be at least an N cycle delay (where N is the settle time) between the profile generator being complete and the motion complete bit being set.

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

The motion complete bit functions in the S-curve point-to-point, Trapezoidal point-to-point, and Velocity Contouring profile modes only. It does not function when the profile mode is set to Electronic Gearing.

8.4 *In-motion indicator*

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

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

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

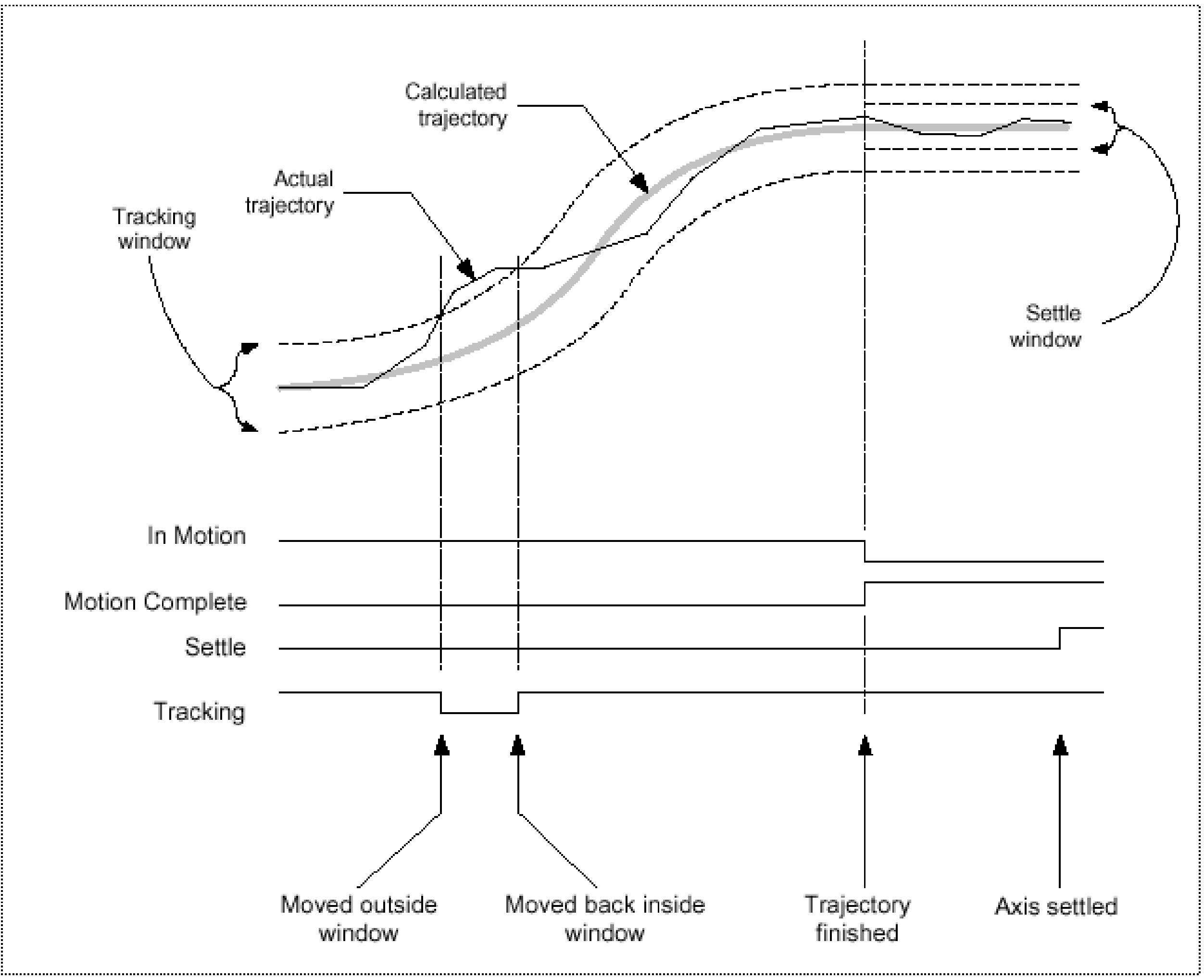
8.5 *Settled indicator*

The chipset can also continuously indicate whether or not the axis has 'settled'.

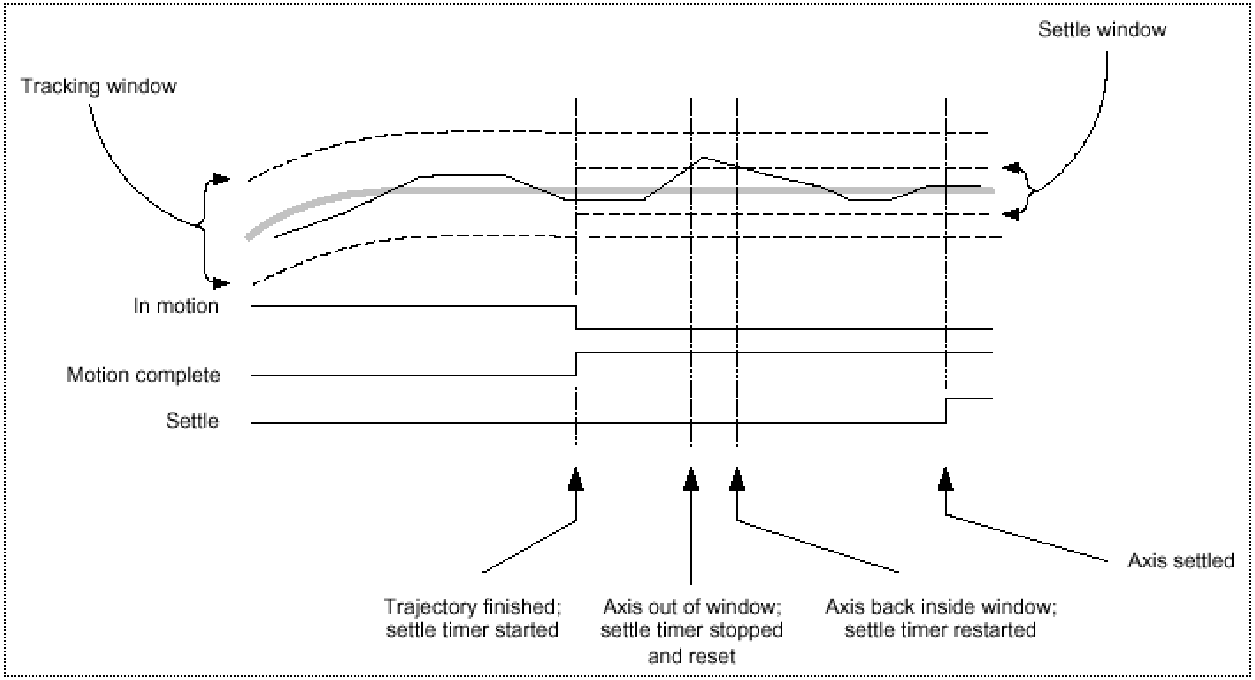
The settled indicator appears in bit 6 of the activity status register. The settled indicator is similar to the motion complete bit when the motion complete mode is set to actual. The differences are that the settled indicator continuously indicates its status (cannot be set or cleared) and also that it indicates regardless of whether or not the motion complete mode is set to actual.

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

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



The tracking window


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

The settle window

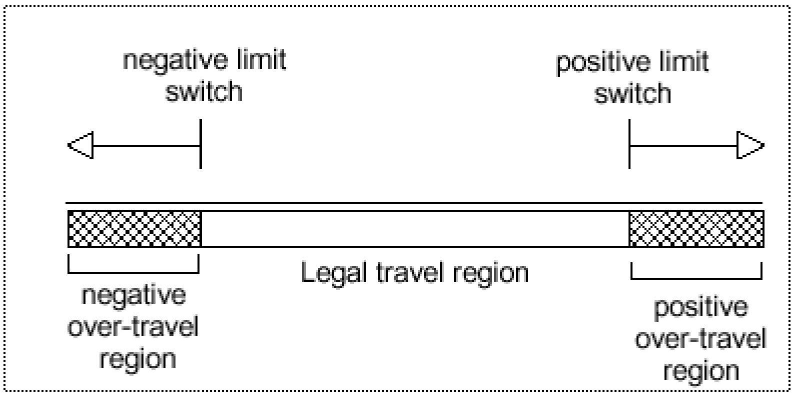
9 Hardware Signals

There are a number of signals that appear on each axis of the Navigator chipsets, which can be used to coordinate chipset activity with events outside the chipset. In this section we will discuss these signals. They are the bidirectional travel limit switches, the AxisIn pin, and the AxisOut pin. These signals appear on each axis of the chipset.

9.1 Travel-limit switches

The Navigator chipsets support motion travel limit switches that can be used to automatically recognize an "end of travel" condition. This is an important safety feature for systems that have a defined range of motion.

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



Directional limit switch operation

The positive and negative switches are connected to CP chip inputs LS1 and LS2, to detect overtravel in the positive and negative directions, respectively.

There are two primary functions that the Navigator provides in connection with the over-travel limit switch inputs. The host can be automatically notified that an axis has entered an over-travel condition, allowing the host to take appropriate special action to manage the over-travel condition.

Upon entering an over-travel condition, the trajectory generator can be automatically halted, so that the motor does not travel further into the over travel region.

Limit switch processing may be enabled or disabled for a given axis through the command SetLimitSwitchMode. This same register can be read using the command GetLimitSwitchMode.

If limit processing is enabled then the chipset will constantly monitor the limit switch input pins looking for a limit switch event. A limit switch event occurs when a limit switch goes active while the axis commanded position is moving in that limit switch’s direction. If the axis is not moving, is in open-loop mode, or is moving in the opposite direction, then a limit switch event will not occur. For example a positive limit switch will occur when the axis commanded position is moving in the positive direction and the positive limit switch goes active. However it will not occur if the axis commanded position is moving in the negative direction or is stationary.

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

When a limit event occurs, the chipset will generate an abrupt stop thereby halting the motion. In addition, the bit in the event status register corresponding to the active limit switch will be set. Finally, the appropriate bit in the activity status register will be set. Once an axis has entered a limit switch condition the following steps should be taken to clear the limit switch event:

- Unless limit switch events can occur during normal machine operation the cause of the event should be investigated and appropriate safety corrections made.
- The limit switch bit(s) in the event status register should be cleared by issuing the ResetEventStatus command. No motion is possible in any direction while either of the limit switch bits in the event status register is set.
- A move should be made in the direction opposite to the direction that caused the limit switch event. This can be any profile move that 'backs' the axis out of the limit. If the host instead attempts to move the axis further into the limit, a new limit event will occur and an instruction error will be generated. (See section 7.1.2 on instruction errors for more information).

If the limit switches are wired to separate switches it should not be possible for both limit switches to be active at the same time. However, if this does occur (presumably due to a special wiring arrangement) then both limit switch bits in the activity status register would be set, thus disabling moves in either direction. In this case, the SetLimitMode command should be used to temporarily disable limit switch processing while the motor is moved off of the switches.

NOTE: Limit switches do not function when the chipset is in 'motor off', also known as 'open-loop' mode.

9.2 The AxisOut pin

Each axis has a general purpose axis output pin which can be programmed to track the state of any of the assigned bits in the Event Status, Activity Status, or Signal Status registers. The tracked bit in one of these three registers may be in the same axis or in a different axis as the axis of the AxisOut pin itself. This function is useful for outputting hardware signals to trigger external peripherals.

The chipset command SetAxisOutSource may be used to configure the axis output pin. This command takes a single word as an argument. The value of this parameter is interpreted as follows:

Bit	**Name**	**Description**
0-3	Source axis	Specifies the axis to be used as a source for the axis output signal. The axis output pin will follow the specified register bit of the source axis. Writing a zero indicates axis 1, writing a one indicates axis 2, etc.
4-7	Bit number	Indicates which bit in the selected status register will be followed by the axis output pin. Bits are numbered from 0 to 15 where bit 0 indicates the least significant bit.
8-11	Status register	Indicates which register will be used as the source for the axis

		output. The encoding is as follows: **ID** **Register** 0 None, the axis output pin will always be inactive 1 Event status register 2 Activity status register 3 Signal status register 4-15 Reserved, do not use
12-15	reserved	Reserved for future use. Should be written as zeros.

Note that the AxisOut pin may be configured to be active low or active high using the SetSignalSense command.

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

9.3 The AxisIn pin

Each axis has an input pin (AxisIn) which can be used as a general purpose input, readable using the GetSignalStatus command, as well as to trigger automatic events such as performing a motion change (stop, start, change of velocity, etc.) upon a signal transition using breakpoints.

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

10 Direct Access to the H320 Controller

10.1 Introduction

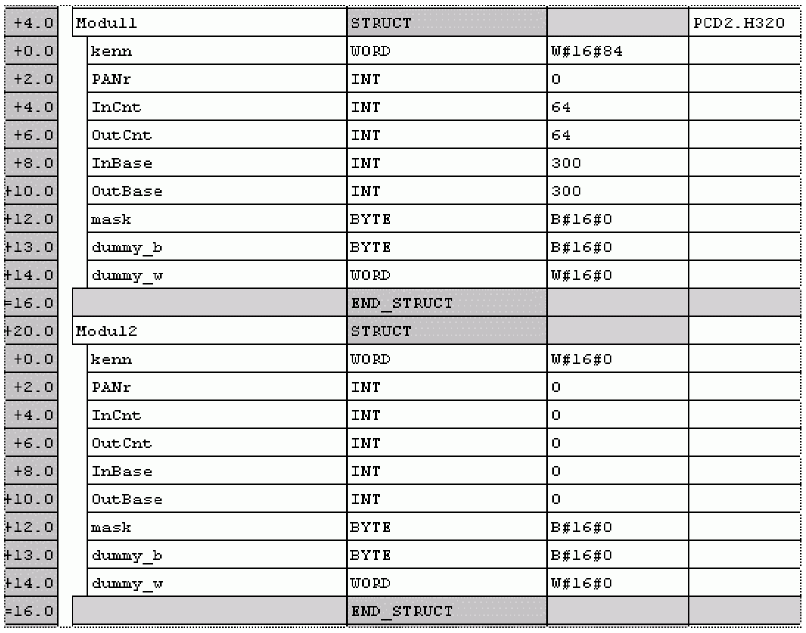
This document describes the configuration and the adressing Modes of the PCD2.H320 with the xx7. With direct peripherie access it is possible to adress the H320 Module. The Module needs a peripherie field of 64 Byte for input and 64 Byte for output data.

It is not possible to adress the H320 Modul in the Start OB (OB100, OB101).

10.2 Definition in the Peripherie DB

The definition of the H320 Modul is done in the periferie definition DB (DB1 or DB511). The H320 needs two places in the PCD. In the definition DB it needs two module entrys.

The Modul definition has the following structure:



+4.0	Modul1	STRUCT		PCD2.H320
+0.0	kenn	WORD	W#16#84	
+2.0	PANr	INT	0	
+4.0	InCnt	INT	64	
+6.0	OutCnt	INT	64	
+8.0	InBase	INT	300	
+10.0	OutBase	INT	300	
+12.0	mask	BYTE	B#16#0	
+13.0	dummy_b	BYTE	B#16#0	
+14.0	dummy_w	WORD	W#16#0	
=16.0		END_STRUCT		
+20.0	Modul2	STRUCT		
+0.0	kenn	WORD	W#16#0	
+2.0	PANr	INT	0	
+4.0	InCnt	INT	0	
+6.0	OutCnt	INT	0	
+8.0	InBase	INT	0	
+10.0	OutBase	INT	0	
+12.0	mask	BYTE	B#16#0	
+13.0	dummy_b	BYTE	B#16#0	
+14.0	dummy_w	WORD	W#16#0	
=16.0		END_STRUCT		

The following table summarizes the entrys in the definition:

Modul 1:

Name	Format	Description
kenn	word	Kennung of the Moduls: **84h**
PaNr	int	has no meening. Allways 0
InCnt	int	Number of Input bytes: **64**
OutCnt	int	Number of Output bytes: **64**
InBase	int	Basis Adresse of the Moduls in the Peripherie- Input field
OutBase	int	Basis Adresse of the Moduls in the Peripherie- Output field
mask	byte	Encoder configuration of the X- Achse:
dummy_b	byte	has no meening. Allways 0
dummy_w	word	SSI Konfiguration der X- Achse:

Modul 2:

Name	Format	Description
kenn	word	has no meening. Allways 0
PaNr	int	has no meening. Allways 0
InCnt	int	has no meening. Allways 0
OutCnt	int	has no meening. Allways 0
InBase	int	has no meening. Allways 0
OutBase	int	has no meening. Allways 0
mask	byte	Encoder Konfiguration der Y- Achse:
dummy_b	byte	has no meening. Allways 0
dummy_w	word	SSI Konfiguration der Y- Achse:

Attention:

In the actuall FW- Version thes byte Mask and the word dummy_w are not implemented. Please set this values to zero!

10.3 Modul access

There are two types of Modul access:

1. read and write to the FPGA (this functions are not implementet)
2. read and write to the Controller on the Module

The peripherie field is devided into three parts.

1. read and write values and commands to Axis one (Address 0 to 24)
2. read and write values and commands to Axis two (Adress 25 to 49)
3. read and write values and commands in general (Adress 50 to 59)

10.3.1 Access to the FPGA

Read and write to the FPGA is in the actuall FW version not possible.

10.3.2 Access to the Controller

There are two ways to acess to the controller.

1. For nearly each command exists a direct peripherie adress (see table below).
2. It existe one peripherie Adress, to write the command Nr. to the Module and a command to wirte or read datas from to the controller. If you use these commands pleas look and unlook the interrupts with sfc41 and sfc42.

In the table below are all commands mapped on the peripherie adress range. The real Adress is the basis Adress (defined in DB1 or 511) pluss the Adress Offset in the table. The value for the command has to be in Akku 1 or will be in Akku 1. CMD is the Command Number for the Controller, it is only interesting as a link to the controller documentation.

Adresse	Transfer Byte	CMD	Transfer Word	CMD	Transfer Doppelword	CMD	Load Byte	CMD	Load Word	CMD	Load Doppelword	CMD
0	SetLimitSwitch Mode	80	SetMotorLimit	6	SetPostion	10	GetLimitSwitch Mode	81	GetMotorLimit	7	GetPostion	4A
1	SetAxisMode	87	SetMotorBias	0F	SetVelocity	11	GetAxisMode	88	GetMotorBias	2D	GetVelocity	4B
2	SetProfilMode	A0	SetKP	25	SetJerk	13	GetProfilMode	A1	GetKP	50	GetJerk	58
3	SetStop	D0	SetKI	26	SetGear Ration	14	GetStop	D1	GetKI	51	GetGearRatio n	59
4	SetMotorMode	DC	SetKD	27	SetActual Position	4D	GetMotorMode	DD	GetKD	52	GetActualPosit ion	37
5	SetAutoStop Mode	D2	SetKvff	2B	Set Acceleration	90	GetAutoStopM ode	D3	GetKvff	54	GetAcceleratio n	4C
6	SetCapture Source	D8	SetInterrupt Maske	2F	Se tDeceleration	91	GetCaptureSou rce	D9	SetInterruptMa sk	56	GetDeceleratio n	92
7	SetMotion CompliteMode	EB	SetMotor Command	77	Set Integration Limit	95	GetMotion CompliteMode	EC	GetMotor Command	69	Get Integration Limit	96
8	Update	1A	SetKaff	93	SetPosition ErrorLimit	97	GetEncoder Source	DB	GetKaff	94	GetPostion ErrorLimit	98

9	ClearPosition Error	47	SetDerivative Time	9C	SetBreakpoint Value1	D6	Read HW Inputs X		Get Derivative Time	9D	Get Breakpoint Value1	D7
10			SetKout	9E	Set Breakpoint Value2	D6	Read SSI Status X		GetKout	9F	Get Breakpoint Value2	D7
11			SetSignal Sense	A2					GetSignal Sense	A3	Get Commanded Positon	1D
12			SetTracking Window	A8					GetTracking Window	A9	Get Commanded Velocity	1E
13			SetSettleTime	AA					GetSettle Time	AB	GetCapture Value	36
14			SetGear Master	AE					GetGearMaster	AF	GetPosition Error	99
15			SetSettle Window	BC					GetSettle Window	BD	GetIntegral	9A
16			SetBreakpoint 1	D4					GetBreakpoint 1	D5	Get Commanded Acceleration	A7
17			SetBreakpoint 2	D4					GetBreakpoint 2	D5	GetActual Velocity	AD
18			SetAxisOut Source	ED					GetAxis Out Source	EE		
19			ResetEvent Status	34					GetEvent Status	31		
20									GetEncoder Modulus	8D		
21									GetCurrentMot orCommand	3A		
22									GetDerivative	9B		
23									GetSignalStatu s	A4		
24									GetActivity	A6		

25	SetLimitSwitch Mode	80	SetMotorLimit	6	SetPostion	10	GetLimitSwitch Mode	81	GetMotorLimit	7	GetPostion	4A
26	SetAxisMode	87	SetMotorBias	0F	SetVelocity	11	GetAxisMode	88	GetMotorBias	2D	GetVelocity	4B
27	SetProfilMode	A0	SetKP	25	SetJerk	13	GetProfilMode	A1	GetKP	50	GetJerk	58
28	SetStop	D0	SetKI	26	SetGearRation	14	GetStop	D1	GetKI	51	GetGear Ration	59
29	SetMotorMode	DC	SetKD	27	SetActual Position	4D	GetMotorMode	DD	GetKD	52	GetActual Position	37
30	SetAutoStop Mode	D2	SetKvff	2B	Set Acceleration	90	GetAutoStop Mode	D3	GetKvff	54	Get Acceleration	4C
31	SetCapture Source	D8	SetInterrup tMaske	2F	Set Deceleration	91	GetCapture Source	D9	SetInterrupt Mask	56	Get Deceleration	92
32	SetMotion CompliteMode	EB	SetMotor Command	77	SetIntegration Limit	95	GetMotion CompliteMode	EC	GetMotor Command	69	GetIntegration Limit	96
33	Update	1A	SetKaff	93	SetPosition ErrorLimit	97	GetEncoder Source	DB	GetKaff	94	GetPostion ErrorLimit	98
34	ClearPosition Error	47	SetDerivative Time	9C	SetBreakpoint Value1	D6	Read HW Inputs Y		GetDerivative Time	9D	GetBreakpoint Value1	D7
35			SetKout	9E	SetBreakpoint Value2	D6	Read SSI Status Y		GetKout	9F	GetBreakpoint Value2	D7
36			SetSignal Sense	A2					GetSignal Sense	A3	Get Commanded Positon	1D
37			SetTracking Window	A8					GetTracking Window	A9	Get Commanded Velocity	1E
38			SetSettleTime	AA					GetSettleTime	AB	GetCapture Value	36
39			SetGear Master	AE					GetGearMaster	AF	GetPosition Error	99
40			SetSettle Window	BC					GetSettle Window	BD	GetIntegral	9A
41			SetBreakpoint 1	D4					GetBreakpoint 1	D5	Get Commanded Acceleration	A7
42			SetBreakpoint 2	D4					GetBreakpoint 2	D5	GetActual Velocity	AD
43			SetAxisOut Source	ED					GetAxisOut Source	EE		
44			ResetEvent Status	34					GetEvent Status	31		
45									GetEncoder Modulus	8D		
46									GetCurrent Motor Command	3A		
47									GetDerivative	9B		
48									GetSignal Status	A4		
49									GetActivity	A6		

50	Reset	39	Write 16Bit		Write32 Bit		GetInterrupt Axis	E1	Read 16 Bit		Read 32 Bit	
51	MultiUpdate	5B	Write CMD		*SetBufferStart	C0	GetHostIOError	A5	Read Signature		*GetBuffer Start	C1
52	ClearInterrupt	AC	SetSample Time	38	*SetBuffer Length	C2	GetModul Status		GetSample Time	61	*GetBuffer Length	C3
53	SetTraceMode	B0	SetTraceStart	B2	*SetBuffer WriteIndex	C4	GetTraceMode	B1	GetTraceStart	B3	*GetBuffer ReadIndex	C7
54	NoOperation	0	SetTraceStop	B4	*SetBuffer ReadIndex	C6	GetTraceStatus	BA	GetTraceStop	B5	*ReadTrace Buffer	C9
55			SetTrace Variable1	B6	*WriteTrace Buffer	C8			GetTrace Variable1	B7	GetTime	3E
56			SetTrace Variable2	B6					GetTrace Variable2	B7	GetTrace Count	BB
57			SetTrace Variable3	B6					GetTrace Variable3	B7		
58			SetTrace Variable4	B6					GetTrace Variable4	B7		
59			SetTrace Period	B8					GetTrace Period	B9		
60												

* the Buffer ID for the tracebuffer is 0. The other Buffers can be written only with the "Write CMD" command.

Read the FPGA Data. This functions are not implementet in the actual FW Version.(V2.09)

For a detailed description of all the commands see in the "Navigator Motion Processor User Guide" and "Navigator Motion Processor Programmer's Reference" from PMD.