LOGOSOL

Four-Axis DC Servo Controller With Built-In Power Amplifiers

LS-421

Software Reference

Document No 715000010 / Rev. 1.1, April 1999

The information in this data book has been carefully checked and is believed to be reliable, however, no responsibility can be assumed for inaccuracies. All information in this manual is subject to change without notice and does not represent a commitment on the part of the vendor.

FlexWare ™ is trademark of Logosol, Inc.

Trademarks are used throughout the manual only as reference to existing common products. If a registered trademark is used without being indicated as such, this does not imply that the name is free.

Logosol, Inc.

1155 Tasman Drive, Sunnyvale, CA 94089

Phone: (408) 744-0974 Fax: (408) 744-0977

LS-421 Software Reference

April 1999

Document No 715000010

Preface

Thank you for choosing Logosol Motion Controller for addressing your automation needs. Logosol is committed to providing products with exceptional value and functionality.

LS-421 Motion Control Board is an integrated product which combines three major functions in one board: *Motion Control, I/O Control and Power Amplification*.

The Software Reference presents a summarized description of the board low-level software features and gives an example motion control application.

We sincerely hope this manual will help you quickly and efficiently integrate the board into your system.

Table of Contents

I. How To Use This Manual

This manual is a brief reference on the low-level software features of the LS-421 motion control board. It summarizes the functions controlled through LS-421 port registers, stripped of the intricacy of the hardware implementation. Its goals are twofold:

- First, it is intended to serve as a concise reference for software developers. Some of the topics copy respective parts of the LS-421 Technical Reference and the user is encouraged to turn to those parts whenever he or she needs to verify how things work on the hardware level.
- Second, it presents some aspects of the Motion Processor (MP) programming, which are specific to LS-421 motion controller. For reference purposes it presents a list of MP commands and status words but it is not a substitute for the MC1401A Reference Manual.

The material in this publication is organized as follows:

- *Software model* introduces the concepts of the Motion Processor Interface and the System and Peripheral Interface.
- *Motion Processor Interface* presents MP command set, MP statuses and MP interrupts.
- *System and Peripheral Interface* presents in a table format the controller functions, which are performed by the board itself and not by the MP, such as amplifier control, control of digital inputs and outputs, setting of current limits, etc.
- *Basic Control Procedures* presents as flowcharts the procedures for board initialization, identification, enabling and disabling of amplifiers and controller diagnostics.
- *Programming Examples* presents an example motion control program written in C programming language, which is intended to serve as a guideline for system developers.

II. Architecture Overview

LS-421 is designed as a standard ISA bus peripheral device. The host system communicates with the LS-421 motion controller using designated port addresses. The port address space is organized in bit-registers, having fixed offsets with respect to the board base address. Eight base addresses are available, depending on J1 jumper setting – 0x280, 0x1280, 0x2280, 0x3280, 0x2A0, 0x12A0, 0x22A0, 0x32A0.

To save I/O space, LS-421 uses only 16 bytes from the lowest 1024 I/O addresses range. The rest of LS-421 registers occupy the space located at offsets 0x400, 0x800 and 0xC00. For instance, if LS-421 base address is set to 0x280, the following I/O address are in use – 0x280÷0x28F, 0x680÷0x68F, 0xA80÷0xA8F, 0xE80÷0xE8F.

In this manual and in other reference sources, registers are named consistently with three to six-letter abbreviations, which correspond to their function. For a quick reference on register names, offsets and function turn to “Memory Map” in Appendix A on page 60.

Broadly speaking, LS-421 motion controller presents two software interfaces to the host:

- Motion Processor Interface
- System and Peripheral Interface

This description of the MP interface, starting on page 8 covers the following topics:

- Register addresses and host to MP synchronization
- MP Command Set
- MP Status
- Interrupt generation by MP
- Position Capture

> ***Note***
>
> *This manual is not intended to be the complete software reference for MP commands and programming techniques. Software developers are advised to use the MC1401 Reference Manual, available from Precision Motion Devices, Inc., as the ultimate source of information for MP programming.*

The System and Peripheral Interface consists of a set of registers for controlling the following functions:

- Board identification
- Interrupt control
- Amplifier control
- Control of digital I/O lines
- Control of system and motor power
- Control of outbound power sources
- Encoder Interface
- System diagnostics

A summarized description of the above functions and the registers, which control them is given in section "System and Peripheral Interface", starting on page 21.

NOTATION

Throughout this chapter, offsets and access type for each register are presented in table format as shown below. The first column represents the function of the register, the second column represents the offset in bytes from the base address. The third column presents the abbreviated name of the bits within a register. The fourth column shows the type of access to the register: *Read (R)*, *Write (W)* or *Read/Write (R/W)*. The fifth column provides details on the usage of the respective bit or register.

Bits are numbered bit 7 – MSB and bit 0 – LSB. Some of the bits have overlapping functions, i.e. may support different registers in *Read* and *Write* modes. In the following example the READY bit (Base Addr + 1, bit 7) in *Read* mode overlaps bit 7 of the MP-CMD register in *Write* mode.

Function	Address	Data bits - Name	Access	Comment
MOTION PROCESSOR REGISTERS	Base Addr + 0x00	Bit 7-0 - MP-DAT	R/W	MC1401A Data Register.
	Base Addr + 0x01	Bit 7-0 - MP-CMD	W	MC1401A Command Register.
	Base Addr + 0x01	Bit 7 – READY	R	MC1401A Ready flag

III. Motion Processor Interface

In LS-421 motion controller a DSP chipset (MC1401A) performs the functions of a motion control processor. It consists of a pair of integrated circuits, which function together as one integrated motion processor.

MP communicates with its environment through the following three interfaces:

- Encoder Inteface
- Amplifier Interface
- Host Processor Interface

Within the LS-421 board architecture, the amplifier interface is connected to the Base Module, the encoder interface is connected to the Input Board and MP *Host Processor Interface* is represented in the MP-CMD and MP-DAT registers.

MP performs the following functions:

- Reads quadrature encoder signals and accumulates current encoder position.
- Generates PWM motor output signals, which are fed to the motor power amplifiers.
- Closes the servo loop, based on Proportional-Integral-Derivative (PID) + Velocity Feed Forward (Vff) + DC bias digital servo filtering. Open-loop motor control is also an option.
- Performs high-speed position capture (latching), triggered by external signals.
- Generates the trajectory profile. The following types are supported: S-curve, trapezoidal, velocity contouring, electronic gearing.
- Checks motor torque limits.
- Detects and recovers from excessive motion error, when motion is stopped abruptly by a protection circuit.
- Interprets host commands sent to MP.

- Sets host interrupts.
- Maintains MP status information.

The following features of the MC1401 chipset are not supported in the LS-421 motion controller:

- The control lines for travel limit switches are not connected. Thus MP axis status will never report travel limit exceeded.
- *Index* and *Home* high-speed position capture lines of the MP are connected together. Thus, position latching signals from LS-421 (index pulses or inputs designated for position capture) will activate both *Index* and *Home* lines. The host software may choose which of one of them to use for position capturing.

This description of the MP interface includes the following subjects:

- Register addresses and host to MP synchronization
- MP Command Set
- MP Status
- Interrupt generation by MP
- Position Capture

The following aspects of MP operation are not covered:

- Reading of the current positions for each of the motion axes
- Setting target position and motion execution
- Velocity Profiles
- Setting and reading of MP parameters (PID coefficients, velocity profile parameters, etc.)

The user is referred to the MC1401 Reference Manual for additional information on these topics.

MP to Host Communication and Synchronization

The interface from the host system to the Motion Processor is through single-byte commands written to the MP Command Register (MP-CMD: 8 bit) and 16-bit words, coded as pairs of bytes that are read or written to the MP Data Register (MP-DAT: 8bit). When MP is available to receive the next command or portion of data it sets the READY bit. The READY bit (Base Addr + 1, D7) in *Read* mode overlaps the 7-th bit (D7) of the MP-CMD register in *Write* mode.

Function	Address	Data bits - Name	Access	Comment
MOTION PROCESSOR REGISTERS	Base Addr + 0x00	Bit 7-0 - MP-DAT	R/W	MC1401A Data Register.
	Base Addr + 0x01	Bit 7-0 - MP-CMD	W	MC1401A Command Register.
	Base Addr + 0x01	Bit 7 - READY	R	MC1401A Ready flag

MP command set is made of single-byte commands. Data is organized in 16-bit words. All data is encoded "high to low", i.e. each 16-bit word is encoded high byte first, low byte second, and two word data values are encoded high word first, low word second. See MC1401 Reference Manual for details on coding of negative values and internal scaling.

Commands could be of one of the following three types:

- Dataless command
- Write command
- Read command

All commands associated with data (read or write) have either 1 or 2 words of data.

The host system must comply with the following requirements when sending a command and reading or writing data to the MP:

- To send a command, the host must write the command code to the MP-CMD register first, then write or read the words of data to the MP-DAT register, if any.
- Each write operation to the MP-CMD register must be preceded by poling (reading repeatedly) the READY register to confirm that MP is available to receive the command.
- Each writing / reading of the high byte of each word to / from the MP-DAT register must be preceded by poling the READY register to confirm that MP is available to receive data or that the data contained in MP-DAT is valid for reading. This poling is not necessary between the bytes of each word (before reading / writing the second byte).

MOTION PROCESSOR RESET

After hardware reset or after system power up, MP is reset. To remove the MP reset, RST bit should be set LOW (0). Setting the same bit HIGH (1) will reset MP.

Function	Address	Data bits - Name	Access	Comment
POWER AND SERVO CONTROL	Base Addr + 0x0a	Bit6 – RST	R/W	Reset MC1401A. Active state HIGH.

MP Command Summary

A list of MP commands, including command codes, parameters and a short description is given in the following table:

Command Mnemonic	Code Hex	Available On	Axes Acted on	Data words /direction	Description
Axis control					
SET_1	01	All axes	Set by	1/read	Set current axis number to 1
SET_2	02	All axes	Set by	1/read	Set current axis number to 2
SET_3	03	All axes	Set by	1/read	Set current axis number to 3
SET_4	04	All axes	Set by	1/read	Set current axis number to 4
SET_I	08	All axes	Interrupting	1/read	Set current axis number to the interrupting axis
Profile Generation					
SET_PRFL_S_CRV	0b	All axes	Current	0	Set profile mode to S-curve
SET_PRFL_TRAP	09	All axes	Current	0	Set profile mode to trapezoidal point to point
SET_PRFL_VEL	0a	All axes	Current	0	Set profile mode to velocity-contouring
SET_PRFL_GEAR	0c	1,2	Current	0	Set profile mode to electronic gear
SET_POS	10	All axes	Current	2/write	Set command position
SET_VEL	11	All axes	Current	2/write	Set command velocity
SET_ACC	12	All axes	Current	2/write	Set command acceleration
SET_MAX_ACC	15	All axes	Current	1/write	Set max acceleration (S-curve profile only)
SET_JERK	13	All axes	Current	2/write	Set command jerk
SET_RATIO	14	1,2	Current	2/write	Set command electronic gear ratio
STOP/CLR_PRFL	46	All axes	Current	0	Abruptly stop current axis trajectory motion

Command Mnemonic	Code Hex	Available On	Axes Acted on	Data words /direction	Description
SMOOTH_STOP	4e	All axes	Current	0	Smoothly stop current axis trajectory motion
SYNCH_PRFL	47	All axes	Current	0	Set target position equal to actual position
ZERO_POS	3f	All axes	Current	0	Zero actual axis position and target position
GET_POS	4a	All axes	Current	2/read	Get command position
GET_VEL	4b	All axes	Current	2/read	Get command velocity
GET_ACC	4c	All axes	Current	2/read	Get command acceleration
GET_MAX_ACC	4f	All axes	Current	1/read	Get max. acceleration (S-curve profile only)
GET_JERK	58	All axes	Current	2/read	Get command jerk
GET_RATIO	59	1,2	Current	2/read	Get command electronic gear rate
GET_TRGT_POS	1d	All axes	Current	2/read	Get current target position
GET_TRGT_VEL	1e	All axes	Current	2/read	Get current target velocity
Digital Filter					
SET_KP	25	All axes	Current	1/write	Set proportional gain
SET_KD	27	All axes	Current	1/write	Set derivative gain
SET_KI	26	All axes	Current	1/write	Set integral gain
SET_KVFF	2b	All axes	Current	1/write	Set feed-forward gain
SET_I_LM	28	All axes	Current	1/write	Set integration limit
SET_MTR_LMT	06	All axes	Current	1/write	Set motor output limit
SET_MTR_BIAS	0f	All axes	Current	1/write	Set motor output bias
SET_POS_ERR	29	All axes	Current	1/write	Set maximum position error limit
GET_KP	50	All axes	Current	1/read	Get proportional gain

Command Mnemonic	Code Hex	Available On	Axes Acted on	Data words /direction	Description
GET_KD	52	All axes	Current	1/read	Get derivative gain
GET_KI	51	All axes	Current	1/read	Get integral gain
GET_KVFF	54	All axes	Current	1/read	Get velocity feed-forward gain
GET_I_LM	53	All axes	Current	1/read	Get integration limit
GET_MTR_LMT	07	All axes	Current	1/read	Get motor output limit
GET_MTR_BIAS	2d	All axes	Current	1/read	Get motor output bias
GET_POS_ERR	55	All axes	Current	1/read	Get position error
GET_INTGR	2e	All axes	Current	1/read	Get integrated position error value
GET_ACTL_POS_ERR	60	All axes	Current	1/read	Get actual position error
SET_AUTO_STOP_ON	45	All axes	Current	0	Set auto stop on motion error mode on
SET_AUTO_STOP_OFF	44	All axes	Current	0	Set auto stop on motion error mode off
Parameter Update					
SET_TIME_BRK	17	All axes	Current	0	Set breakpont mode to time
SET_POS_BRK	18	All axes	Current	0	Set breakpont mode to pos. target position
SET_NEG_BRK	19	All axes	Current	0	Set breakpont mode to neg. target position
SET_ACTL_POS_BRK	1b	All axes	Current	0	Set breakpont mode to pos. actual position
SET_ACTL_NEG_BRK	1c	All axes	Current	0	Set breakpont mode to neg. actual position
SET_MTN_CMPLT_BRK	35	All axes	Current	0	Set breakpont mode to motion complete
SET_EXT_BRK	5e	All axes	Current	0	Set breakpont mode to external
SET_BRK_OFF	6d	All axes	Current	0	Set breakpont mode off
SET_BRK_PNT	16	All axes	Current	2/write	Set breakpoint comparison value

Command Mnemonic	Code Hex	Available On	Axes Acted on	Data words /direction	Description
UPDATE	1a	All axes	Current	0	Immediate parameter update
MULTI_UPDATE	5b	All axes	Set by mask	1/write	Multiple axis immediate parameter update
SET_AUTO_UPDATE_ON	5c	All axes	Current	0	Set automatic profile update on
SET_AUTO_UPDATE OFF	5d	All axes	Current	0	Set automatic profile update off
GET_BRK_PNT	57	All axes	Current	2/read	Get breakpoint comparison value
Interrupt processing					
SET_INTRPT_MASK	2f	All axes	Current	1/write	Set interrupt mask
GET_INTRPT	30	All axes	Interrupting	1/read	Get status of interrupting axis
RST_INTRPT	32	All axes	Interrupting	1/write	Reset interrupting events
GET_INTRPT_MASK	56	All axes	Current	1/read	Get interrupt mask
Status/Mode					
CLR_STATUS	33	All axes	Current	0	Reset status of current axis
RST_STATUS	34	All axes	Current	1/write	Reset events for current axis
GET_STATUS	31	All axes	Current	1/read	Get axis status word
GET_MODE	48	All axes	Current	1/read	Get axis mode word
Encoder					
SET_CAPT_INDEX	64	All axes	Current	0	Set index signal as position trigger
SET_CAPT_HOME	65	All axes	Current	0	Set home signal as position trigger
GET_CAPT	36	All axes	Current	2/read	Get current axis position capture location
Motor					
SET_OUTPUT_PWM	3c	All axes	Global	0	Set motor output mode to PWM

Command Mnemonic	Code Hex	Available On	Axes Acted on	Data words /direction	Description
MTR_ON	43	All axes	Current	0	Enable motor output
MTR_OFF	42	All axes	Current	0	Disable motor output
SET_MTR_CMD	62	All axes	Current	1/write	Write direct value to motor output
GET_MTR_CMD	3a	All axes	Current	1/read	Read motor output command
GET_OUTPUT_MODE	6e	All axes	Global	1/read	Get current output mode
Miscellaneous					
AXIS_ON	41	All axes	Current	0	Enable axis
AXIS_OFF	40	All axes	Current	0	Disable axis
SET_ACTL_POS	4d	All axes	Current	2/write	Set current actual axis location
GET_ACTL_POS	37	All axes	Current	2/read	Get current actual axis location
SET_LMT_SENSE	66	All axes	Global	1/write	Set limit switch bit sense
GET_LMT_SWTCH	67	All axes	Global	1/read	Get state of limit switches
LMTS_ON	70	All axes	Global	0	Set limit switch sensing on
LMTS_OFF	71	All axes	Global	0	Set limit switch sensing off
SET_SMPL_TIME	38	All axes	Global	1/write	Set servo loop sample time
GET_SMPL_TIME	61	All axes	Global	1/read	Get servo loop sample time
RESET	39	All axes	Global	0	Reset chipset
GET_VRSN	6c	All axes	Global	1/read	Get chipset software version information
GET_TIME	3e	All axes	Global	2/read	Get current chipset time (num. of servo loops)

For additional information on MP commands, see MC1401 Reference Guide, available from Precision Motion Devices, Inc.

MP Status Information

MP supports two types of statuses:

- *Axis status*, which is a16-bit word, containing information about the state of the axis. The axis status is reported by GET_STATUS command.
- *Miscellaneous mode status,* which is a 16-bit word, containing information about various mode settings or conditions, for example trajectory generator mode. It is reported by GET_MODE command.

The user should be able to identify what status information is available in the MP status registers and what in the port registers of the LS-421 motion controller. To distinguish between the two, the second will be referred to as *board status*.

MP MOTION AXIS STATUS

The following table contains a description of the 16-bit axis status word, reported by the MP in response to the GET_STATUS command:

Bit #	Description
0	Motion complete flag. This bit is set (1) when the axis trajectory is completed. This flag is only valid for the S-curve, trapezoidal and velocity contouring modes.
1	Wrap-around condition flag. This bit is set (1) when the axis has reached the end of its travel range. Specially, when travelling in a positive direction past the position +1,073,741,823, the axis will wrap to position –1,073,741,824, and vice versa.
2	Breakpoint reached flag. This bit is set (1) when one of the breakpoint conditions has occurred.
3	Index pulse received flag. This bit is set (1) when an index pulse has been received.
4	Motion error flag. This bit is set (1) when the position error is exceeded. This bit can only be reset when the axis is no longer in a motion error condition.
5	Positive limit switch flag. This bit is set (1) when the positive limit switch goes active.[1]
6	Negative limit switch flag. This bit is set (1) when the negative limit switch goes active.
7	Command error flag. This bit is set (1) when a command error has occurred.

[1] Positive and negative travel limit switch lines are not connected to MP in the LS-421 implementation

Bit #	Description
8	Motor on/off status (1 indicates the motor is on, 0 indicates the motor is off).
9	Axis on/off status. (1 indicates on, 0 indicates off)
10	In motion flag. This bit continuously indicates whether or not the axis trajectory is in motion. This bit is set (1) when the axis is in motion, and cleared (0) when the axis is not in motion.
11	Reserved (may contain 0 or 1).
12, 13	Current axis # (13 bit = high bit, 12 bit = low bit). Therefore axis encoding is as follows: Bit 13 Bit 12 Axis 0 0 1 0 1 2 1 0 3 1 1 4
14, 15	Reserved (may contain 0 or 1).

Bits 8-10 and 12-13 indicate continuous status information, and do not need to be reset by the host.

Bits 0-7 of the status word indicate various status flags that can also generate host interrupts. These flags are set by the MP, and must be reset by the host system. They will not be cleared by the MP.

MISCELLANEOUS MODE STATUS

The following table contains a description of the 16-bit status word, reported by the MP in response to the GET_MODE command. This status word reports various mode settings or conditions.

Bit #	Description
0-6	Used internally by chipset.
7	Stop on motion error mode flag. This bit indicates the state of the stop on motion error mode, set by the commands SET_AUTO_STOP_ON and SET_AUTO_STOP_OFF. This bit is set (1) when the auto stop is on.
8-9	Used internally by chipset. Contains no host – usable information.
10	Auto update flag. This bit indicates the state of the auto update mode, set by the following commands: SET_AUTO_UPDATE_ON and SET_AUTO_UPDATE_OFF. This bit is set (1) when the auto update is disabled.

Bit #	Description
11	Trajectory generator mode. This bit indicates the mode of the trajectory generator, set by the following commands: SET_PRFL_S_CRV, SET_PRFL_TRAP, SET_PRFL_VEL and SET_PRFL_GEAR. The encoding is as follows: Bit 12 Bit 11 Profile Mode 0 0 trapezoidal 0 1 velocity contouring 1 0 s-curve 1 1 electronic gear
13-15	Phase #. These bits indicate the current phase # of the S-curve profile (only valid if the current profile mode is S-curve). A 0 indicates that the profile has not started yet, and phases 1-7 indicate the phase number corresponding to the phases described in the S-curve profiling mode. The 3-bit phase number word is encoded bit 15 MSB and bit 13 LSB.

Host Interrupts

In many situations, during axis motion or at other times, it is useful to have the MP signal to the host that a special condition has occurred. Indeed, the MP is able to initiate an interrupt signal. In the MC1401 Reference manual this signal is referred to as *host interrupt*.

In the LS-421 architecture the MP *host interrupt* is connected to the board IRQ line. LS-421 may use IRQ7, IRQ10, IRQ12, IRQ15, depending on the J2 jumper block setting. In addition to setting HIGH the IRQ line the MP *host interrupt* signal sets HIGH (1) the MPI flag in the LS-421 Interrupt Status register. MPI can not be masked off using LS-421 Interrupt Mask register. See “Interrupt Processing” on page 24. As explained further, an internal MP mask register, accessible with MP commands can mask MPI.

Several MP conditions may occur that can result in a generation of host interrupts. Whether these conditions in fact interrupt the host is controllable for each condition and for each axis. The mechanism used to control each condition is an internal MP mask register.

The interrupt conditions correspond to bits 0-7 of the MP Axis Status register (the axis event flags), described in “MP Status” on page 16. These conditions are summarized below:

- *Motion Complete*. Occurs when the profile is complete.
- *Wrap-around condition*. Occurs when the axis position wraps.
- *Break Point Reached*. Occurs when a breakpoint condition has been satisfied.
- *Position Capture Received*. Occurs when the index pulse or home pulse has been captured.
- *Motion Error*. Occurs when the maximum position has been exceeded.
- *Negative Limit Switch*. Occurs when the negative over-travel limit switch is active.

- *Positive Limit Switch*. Occurs when the positive over travel limit switch is active.

- *Command Error*. Occurs when a command error condition is detected.

When one of these interrupt conditions occur for a particular axis, the *host interrupt* line is made active. At this point the host can respond to the interrupt (although the current I/O operation should be completed), but it is not required to do so. When the host has completed processing the interrupt, it sends a RST_INTRPT command to clear the interrupt. This command includes a clearing mask as an argument, which allows one interrupt to be cleared at a time.

Interrupts occur for a particular axis. When an interrupt occurs, it is the host's responsibility to change the current axis number. If more than one axis interrupt becomes active at the same time, then the axis with the lowest number will generate the interrupt first.

The following MP commands are used in managing interrupts:

- SET_INTRPT_MASK. Sets the interrupt conditions mask.

- GET_INTRPT. Returns the status of the interrupting axis (including the interrupting axis number). The current axis number is not altered by this command.

- SET_I. Changes the current axis number to the interrupting axis. This is a "time saver" command which performs the dual operations of getting the interrupting axis # and switching to that axis in one command.

- RST_INTRPT. Clears particular conditions for the interrupting axis. The current axis number is not altered by this command.

For details on MP interrupt handling see MC1401 Reference Manual, available from Precision Motion Devices, Inc.

Position Capture

The Motion Processor is able to latch the current motor position using one of two strobe signal lines:

- *Index Pulse*

- *Home Signal*

These two signal lines are part of the MP encoder interface. In the LS-421 hardware *Index Pulse* and *Home Signal* high-speed position capture lines of the MP are connected together. Thus, position latching signals from LS-421 (index pulses or inputs designated for position capture) will activate both *Index* and *Home* lines. The host software may choose which of one of them to use for position capturing.

MP commands SET_CAPT_INDEX and SET_CAPT_HOME could be alternatively used to select, which input signal to trigger the capture. MP command GET_CAPT is used to read the captured position.

MP recognizes that an index has occurred (and will capture current position) when the index signal transitions low, followed by the A, B encoder channels transitioning low. MP will capture current position based on the *Home* signal regardless of the state of A and B encoder lines.

After current position has been captured by the MP, the host must read the captured position before another capture can occur. In addition, if the index signal is being used as the trigger, the index signal, along with A and B quadrature signals must transition high before another index pulse can be registered.

The above MP position capture interface is enhanced by the LS-421 board, which connects one of 6 designated general purpose inputs or the encoder index lines to the coupled MP *Index Pulse/Home Signal* strobe lines. Writing the corresponding value to Position Capture Select (PCS) register selects one of the inputs: IN2, IN5, IN6, IN7, IN8, IN9 or the indexes to serve as strobes.

Position capturing based on one of the inputs may be useful in sensor devices or measuring equipment for latching position based on external events. Position capture based on an index signal is usually used for homing.

IV. System and Peripheral Interface

This section covers the interface to the following hardware features of the LS-421 board:

- Servo Control and Diagnostics
- Amplifiers Current Limit
- Digital Outputs
- Digital Inputs
- Board ID
- Absolute Position Encoder Interface

Servo Control and Diagnostics

The Servo Control and Diagnostics circuit (SCD) allows the host to set up the controller and to keep track of the current LS-421 status.

The SCD circuit controls the amplifier enable bit (AE) and Motion Processor reset bit (RST) based on several internal and external exception signals.

The following exception signals are monitored:

- Motor Power Failure (MPF)
- Output Overload / Output Power Failure (OFP)
- Emergency Input (EMG)
- Stop Input (STP)
- Amplifier Enable (AE)

- Servo Chip Reset (RST)
- Motor Current Limit Overload (CL0 to CL3)

Conditions that trigger each one of the above signals and the registers reflecting their state are described in the hardware reference and summarized in the following table:

Function	Address	Data bits - Name	Access	Comment
POWER AND SERVO CONTROL	Base Addr + 0x0a	Bit7 – AE	R/W	Amplifier Enable Control. Active state HIGH. Double buffered logic.
		Bit6 – RST	R/W	Reset MC1401A. Active state HIGH.
		Bit5 – OPT	R/W	Optional Relay output OPT. Active state HIGH.
		Bit4 – SPS	R/W	System Power Supply. Active state HIGH. Double buffered logic.
		Bit3 – CL3	R	Power Amplifiers Current Limit Status. This status is latched when the amplifiers are disabled. HIGH indicates overload.
		Bit2 – CL2	R	
		Bit1 – CL1	R	
		Bit0 – CL0	R	
POWER AND SERVO LOOP DIAGNOSTICS	Base Addr + 0x0b	Bit7 – EMG	R	Emergency Stop Input. HIGH indicates that Emergency stop is activated (open contact). Always HIGH when no External power is connected to the controller.
		Bit6 – STP	R	Stop Inputs (option). HIGH indicates that one or more of selected for this function input combinations are activated.
		Bit5 – MPF	R	Motor Power Failure. HIGH indicates that the power is out of range, or there is a leakage to other power sources. Always HIGH if SPS is not enabled.
		Bit4 – OPF	R	Output Power Failure. HIGH indicates that Operating power (24V) is out of range, or one or more of the digital outputs are overloaded. Always HIGH if SPS is not enabled.
		Bit2 – STPL Bit1 – MPFL Bit0 – OPFL		These status bits have the same meaning as above three and are latched at the moment, the amplifiers are disabled (Falling edge of AE).

Digital Outputs and Outbound Power Sources

Digital outputs are controlled by the control bits designated OUT0 to OUT7. The outbound power supplies: OPS, SPS and OPT are controlled by control bits having the same names. For details on the schematics of the digital outputs and the outbound power supplies, see "Digital Outputs and Outbound Power" in the LS-421 Technical Reference.

Function	Address	Data bits - Name	Access	Comment
DIGITAL OUTPUTS	Base Addr + 0x0d	Bit7 – OUT7	R/W	General Purpose Digital Outputs. Logic one activates the outputs. Reading returns the current outputs state.
		Bit6 – OUT6		
		Bit5 – OUT5		
		Bit4 – OUT4		
		Bit3 – OUT3		
		Bit2 – OUT2		
		Bit1 – OUT1		
		Bit0 – OUT0		

Short-circuit protection for the digital outputs is discussed in section Output Overload / Operating Power Failure" in the LS-421 Technical Reference.

> ***Note***
>
> *Activating the emergency stop will turn off SPS and will disable all other power sources and outputs. The encoders and the digital inputs will remain powered.*
>
> *For safety reasons and to prevent an accidental turn on after an emergency, SPS is equipped with a double buffered logic. In order to restore SPS after being shut off, SPS must be cycled off-on, i.e. SPS control bit should be first set LOW (0), then HIGH (1).*

Setting of Current Limits

The current limit of the amplifiers can be set individually for each axis in the range of 2÷10 A with 4-bit precision. Only the four MSB are used. Writing 0x00 to the corresponding register sets the current limit to the lowest level (less than 1A). Writing 0xF0 sets the current limit to the highest level - 10A.

Function	Address	Data bits – Name	Access	Comment
AMPLIFIERS CURRENT LIMIT CONTROL	Base Addr + 0x408	Bit7 – Bit4	W	Current limit for channel #0.
	Base Addr + 0x409	Bit7 – Bit4	W	Current limit for channel #1.
	Base Addr + 0x40a	Bit7 – Bit4	W	Current limit for channel #2.
	Base Addr + 0x40b	Bit7 – Bit4	W	Current limit for channel #3.

Interrupt Processing

LS-421 motion controller may generate hardware interrupts in two cases:

- when amplifiers are disabled (falling edge of AE), regardless of the cause that disabled the amplifiers
- when MP generates an interrupt, which may happen under various circumstances. See “Host Interrupts on page 18.

To distinguish among the two, the host system has to poll the Amplifier Disable Interrupt (ADI) and the Motion Processor Interrupt (MPI) bits in the Interrupt Status register. ADI interrupt may be masked by a control bit (ENADI). MP interrupts may be masked by an internal MP interrupt mask register, which can be set with appropriate commands to MP.

Function	Address	Data bits – Name	Access	Comment
INTERRUPT STATUS AND CONTROL	Base Addr + 0x0c	Bit7 – ADI	R	Amplifier Disable Interrupt. HIGH if power amplifiers are disabled.
		Bit7 – ENADI	W	ADI interrupt mask. LOW (0) after power up or reset. HIGH (1) enables ADI interrupt
		Bit1 – MPI	R	HIGH if there is IRQ from MP. MP-CMD register should be used to handle the interrupt.

Board Identifier

LS-421 features an 8-bit ID number, available to the host in two four-bit read-only registers. Bit 3 is the MSB in each nibble.

The ID for the standard LS-421 configuration is 0xB1.

Function	Address	Data bits – Name	Access	Comment
BOARD INDETIFICATION	Base Addr + 0x807	Bit3 – Bit0	R	Low ID nibble
	Base Addr + 0xc07	Bit3 – Bit0	R	High ID nibble

Digital Optoisolated Inputs

LS-421 provides 16 general-purpose inputs IN0 to IN15. Optionally, IN0 and IN1 may be used as inputs for two multi-turn absolute position encoders. In this case, they are labeled APE2 and APE3, respectively and are not available as general-purpose inputs.

The active state of the inputs is LOW (0), i.e. when a current flows through the sensor circuit the corresponding input register bit is set to logic zero.

Function	Address	Data bits – Name	Access	Comment
INPUTS	Base Addr + 0x08	Bit7 – IN7	R	General Purpose optoisolated inputs. Active state LOW. IN0 and IN1 are not available when multi-turn absolute encoders APE2 and APE3 are used.
		Bit6 – IN6		
		Bit5 – IN5		
		Bit4 – IN4		
		Bit3 – IN3		
		Bit2 – IN2		
		Bit1 – IN1		
		Bit0 – IN0		
	Base Addr + 0x09	Bit7 – IN15	R	General Purpose optoisolated inputs. Active state LOW.
		Bit6 – IN14		
		Bit5 – IN13		
		Bit4 – IN12		
		Bit3 – IN11		
		Bit2 – IN10		
		Bit1 – IN9		
		Bit0 – IN8		

Position Capture Select

LS-421 architecture permits one of 6 designated general purpose inputs or the encoder index lines to be used as strobes. Writing the corresponding value to Position Capture Select (PCS) register selects one of the inputs: IN2, IN5, IN6, IN7, IN8, IN9 or the indexes to serve as strobes.

Function	Address	Data bits – Name	Access	Comment
HIGH SPEED POSITION CAPTURE AND MULTI-TURN ABSOLUTE POSITION ENCODER (APE) RESET	Base Addr + 0x0e	Bit7 – Bit4	W	Position Capture Select. For coding of the input line number and active state, see next table.
		Bit0	W	Multi-turn Absolute Position Encoders Reset. Setting this bit to HIGH for at least 5 sec will reset all absolute encoders.

#	PCS3	PCS2	PCS1	PCS0	Channel#0	Channel#1	Channel#2	Channel#3	Active level
0	0	0	0	0	IN6	IN6	IN6	IN6	Low
1	0	0	0	1	IN6	IN6	IN6	IN6	High
2	0	0	1	0	IN7	IN7	IN7	IN7	Low
3	0	0	1	1	IN7	IN7	IN7	IN7	High
4	0	1	0	0	Reserved	Reserved	Reserved	Reserved	-
5	0	1	0	1	Not used	Not used	Not used	Not used	-
6	0	1	1	0	IN2	IN2	IN2	IN2	Low
7	0	1	1	1	IN2	IN2	IN2	IN2	High
8	1	0	0	0	IN5	IN5	IN5	IN5	Low
9	1	0	0	1	IN5	IN5	IN5	IN5	High
10	1	0	1	0	IN8	IN8	IN8	IN8	Low
11	1	0	1	1	IN8	IN8	IN8	IN8	High
12	1	1	0	0	IN9	IN9	IN9	IN9	Low
13	1	1	0	1	IN9	IN9	IN9	IN9	High
14	1	1	1	0	Index#0	Index#1	Index#2	Index#3	Low
15	1	1	1	1	Index#0	Index#1	Index#2	Index#3	Low

Multi-turn Absolute Position Encoders

In LS-421 motion controller multi-turn absolute position encoders (APE) are used to report to the host the absolute position in case of power failure or other exceptional event. The incremental module of the supported APE[1] is a standard incremental encoder The absolute module constantly transmits serial data consisting of 24-bit absolute position and 6-bit status over the corresponding input line (APE0, APE1, IN0/APE2, IN1/APE3).

To obtain absolute position, the host software must select one of the encoder channels for position reading by performing a write operation to one of the APE REQ0 – APE REQ3 registers. Received data is stored in a 24-bit APE DATA and a 6-bit APE STATUS registers. APE RDY data ready flag is HIGH if data is valid. The supported APE are reset by setting HIGH (1) the RES bit and holding it in that state for at least 4÷5 sec. To enable the absolute position encoders, RES should be set LOW (0).

1 Models SA35-11/24bit-LPS-5V and SA56-11/24bit-LPS-5V, manufactured by Tamagawa Seiki Co are supported

Function	Address	Data bits – Name	Access	Comment
APE DATA	Base Addr + 0x404	Bit7 – APED7	R	Multi-turn absolute position encoder data. The data is valid only if APE READY bit is HIGH.
		Bit6 – APED6		
		Bit5 – APED5		
		Bit4 – APED4		
		Bit3 – APED3		
		Bit2 – APED2		
		Bit1 – APED1		
		Bit0 – APED0		
	Base Addr + 0x405	Bit7 – APED15		
		Bit6 – APED14		
		Bit5 – APED13		
		Bit4 – APED12		
		Bit3 – APED11		
		Bit2 – APED10		
		Bit1 – APED9		
		Bit0 – APED8		
	Base Addr + 0x406	Bit7 – APED23		
		Bit6 – APED22		
		Bit5 – APED21		
		Bit4 – APED20		
		Bit3 – APED19		
		Bit2 – APED18		
		Bit1 – APED17		
		Bit0 – APED16		
MULTI-TURN APE STATUS	Base Addr + 0x407	Bit7 – APE READY	R	Ready flag. HIGH indicates APE data and status are valid.
		Bit5 – APE BE/OS	R	Multi-turn absolute position encoder status. For more information refer to the following table.
		Bit4 – APE OF		
		Bit3 – APE OS		
		Bit2 – APE BA		
		Bit1 – APE PS		
		Bit0 – APE CE		
APE SELECT	Base Addr + 0x404	Any write operation	W	Select APE #0.
	Base Addr + 0x405	Any write operation	W	Select APE #1.
	Base Addr + 0x406	Any write operation	W	Select APE #2.
	Base Addr + 0x407	Any write operation	W	Select APE #3.
HIGH SPEED POSITION CAPTURE AND MULTI-TURN ABSOLUTE POSITION ENCODER (APE) RESET	Base Addr + 0x0e	Bit7 – Bit4 Bit0	W W	Position Capture Select. Multi-turn Absolute Position Encoders Reset. Setting this bit to HIGH for at least 5 sec will reset all absolute encoders.

The following table explains the abbreviations used to describe absolute position encoder status.

APE Status Bits	Description
APE RDY	APE ready flag. If RDY=1, data and status are valid. If RDY=0, data acquiring is in progress.
APE D23 ÷ D0	Multi-turn absolute position encoder data
CE	Counter error status
PS	Pre-load status
BA	Battery alarm
OS	Over-speed
OF	Overflow
BE	Battery error

V. PROGRAMMING EXAMPLES

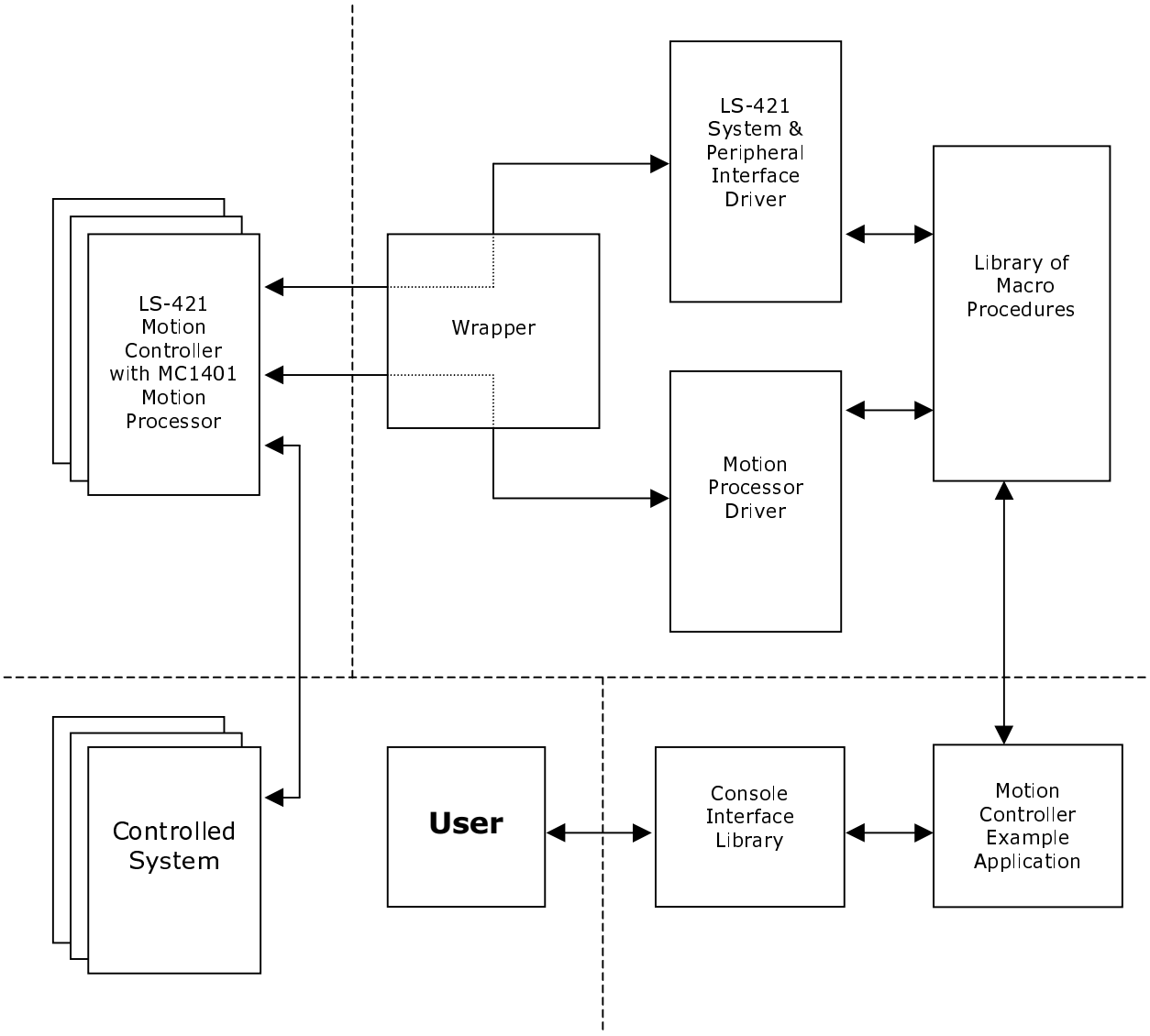
This chapter presents an example motion control program written in C programming language, which is intended to serve as a guideline for system developers. The approach when writing the example was to make it as clear and simple as possible, sacrificing (unfortunately) features, performance and efficiency.

The source code and the compiled executables may be found in file ls421sdk.zip.

The program is build up of the following parts:

- *Wrapper* (wrapper.c). This layer wraps the platform dependent features into C-functions, thus making the rest of the applications less platform dependent.
- *LS-421 System and Peripheral Interface Driver* (ls421.c). This module provides a set of functions, which support the System and Peripheral Interface of LS-421 motion control board.
- *Motion Processor Driver* (pmd1401.c). This module provides a set of functions, which support the MP Interface. The set includes motion primitives, MP parameter setting functions (PID coefficients, speeds, motion profiles, etc.) and other MP related functions.
- *Library of Macro Procedures* (mcp.c). The library contains functions performing common basic operations with the controller, like initialization, homing and recovery after an exception.
- *Motion Controller Example Application* (ls421t.exe). This sample software serves as a command-line interpreter of user commands transforming them into calls to the lower-level functions, mentioned above.

The illustration below gives a general idea of the design of example motion control program, as well as of the components that is part of it:



LS-421 System and Peripheral Interface Driver.

LS-421 System and Peripheral Interface Driver (ls421.c) provides a set of functions, which support the System and Peripheral Interface of LS-421 motion control board.

Function summary is given in the following table:

Function name and parameters	Description
byte mb_init (word io_a, void **mb_h)	Initialize LS-421 board
byte mb_down (void *mb_h)	Reclaim system resources allocated to the board
byte mb_power_on (void *mb_h)	Turn ON the System Power Supply (SPS)
byte mb_power_off (void *mb_h)	Turn OFF the System Power Supply (SPS)
byte mb_set_limit (void *mb_h, byte a_no, byte lim)	Set current limits
byte mb_get_limit (void *mb_h, byte a_no, byte *lim)	Read current limits
byte mb_set_opt (void *mb_h, byte opt)	Turn ON or OFF the Optional Power Supply (OPT)
byte mb_get_opt (void *mb_h, byte *opt)	Read the state of the Optional Power Supply (OPT)
byte mb_set_out (void *mb_h, byte o_no, byte out)	Sets digital output
byte mb_get_out (void *mb_h, byte o_no, byte *out)	Reads a digital output
byte mb_get_inp (void *mb_h, byte i_no, byte *inp)	Reads a digital input
byte mb_set_pcs (void *mb_h, byte pcs)	Set position capture input
byte mb_get_pcs (void *mb_h, byte *pcs)	Get position capture input
byte mb_check (void *mb_h)	Check LS-421 board status
byte mb_servo_on (void *mb_h)	Enable power amplifiers
byte mb_servo_off (void *mb_h)	Disable power amplifiers

Function Descriptions

This section provides short descriptions of the functions contained in the LS-421 System and Peripheral Interface Driver.

byte **mb_init** (word ***io_a***, void ****mb_h***);

This function initializes the a LS-421 board installed at address ***io_a***. The function checks if the hardware is available and allocates the needed resources for the functioning of the driver. The ****mb_h*** parameter returns a pointer to the board identifier (of type *void)*.

byte **mb_down** (void *mb_h);

This function releases the resources allocated by the driver. The ***mb_h*** is a pointer to the board identifier.

byte **mb_power_on** (void ***mb_h***);

This function turns ON the *System Power Supply* (SPS). It *does not enable* the power amplifiers. The ***mb_h*** parameter is the board identifier.

byte **mb_power_off** (void ***mb_h***);

This command turns OFF the *System Power Supply* (SPS). It automatically *disables* the power amplifiers. The ***mb_h*** parameter is the board identifier.

byte **mb_set_limit** (void ***mb_h***, byte ***a_no***, byte ***lim***);

Sets the current limit of the amplifier for axis ***a_no***. The ***lim*** parameter represents limit. It can take integer values between 0 to 15, corresponding to the lowest (1A) and the highest (10 A) current limits, respectively.

byte **mb_get_limit** (void ***mb_h***, byte ***a_no***, byte ****lim***);

Reads the current limit of the amplifier for axis ***a_no***. ***lim*** is a pointer to the obtained value. The current limit can take integer values between 0 to 15, corresponding to the lowest (1A) and the highest (10 A) current limits, respectively.

byte **mb_set_opt** (void ***mb_h***, byte ***opt***);

Turns ON or OFF the *Optional Power Supply* (OPT) relay output by setting the ***opt*** parameter to 1 or 0, respectively. The ***mb_h*** parameter specifies the board identifier.

byte **mb_get_opt** (void ***mb_h***, byte ****opt***);

Reads the current state of the Optional Power Supply (OPT) relay output. The ***mb_h*** parameter specifies the board identifier.

byte **mb_set_out** (void ***mb_h***, byte ***o_no***, byte ***out***);

Sets the state of the digital output ***o_no***. The ***mb_h*** parameter specifies the board identifier.

byte **mb_get_out** (void ***mb_h**, byte ***o_no**, byte ***out**);

Reads the current state of the digital output ***o_no***. The ***mb_h** parameter specifies the board identifier.

byte **mb_get_inp** (void ***mb_h**, byte ***i_no**, byte ***inp**);

Reads the current state of the digital input ***i_no***. The ***mb_h** parameter specifies the board identifier.

byte **mb_set_pcs** (void ***mb_h**, ***byte pcs**);

Sets an input line as a trigger signal for fast position capture. The ***mb_h** parameter specifies the board identifier. The ***pcs*** parameter indicates the input line and is given by one of the following constants:

Constant name	Strobe source	Active state
LS_PCS_IN10L	Digital input 10	Logical zero
LS_PCS_IN10H	Digital input 10	Logical one
LS_PCS_IN11L	Digital input 11	Logical zero
LS_PCS_IN11H	Digital input 11	Logical one
LS_PCS_IN06L	Digital input 6	Logical zero
LS_PCS_IN06H	Digital input 6	Logical one
LS_PCS_IN09L	Digital input 9	Logical zero
LS_PCS_IN09H	Digital input 9	Logical one
LS_PCS_IN12L	Digital input 12	Logical zero
LS_PCS_IN12H	Digital input 12	Logical one
LS_PCS_IN13L	Digital input 13	Logical zero
LS_PCS_IN13H	Digital input 13	Logical one
LS_PCS_INX	Axis encoder index pulse	N/A

byte **mb_get_pcs** (void ***mb_h**, byte ***pcs**);

Reads the identifier of the input line selected as a trigger signal for fast position capture. The ***mb_h** parameter specifies the board identifier. ***pcs*** is a pointer to the identifier of the selected input line.

byte **mb_check** (void ***mb_h**);

Checks the status of the LS-421 board. The status is returned in the following format:

Error	Description
MB_GOOD	Servo loop is closed
MB_ERR_OVLD0	Channel 0 overload
MB_ERR_OVLD1	Channel 1 overload
MB_ERR_OVLD2	Channel 2 overload
MB_ERR_OVLD3	Channel 3 overload
MB_ERR_EMGSTOP	Emergency stop activated
MB_ERR_EMGINP	Emergency input activated
MB_ERR_MPF	Motor power failure
MB_ERR_OPF	Operating power failure
MB_ERR_NOSPS	No power
MB_DISABLED	Servo is disabled

byte **mb_servo_on** (void ***mb_h**);

This function enables the power amplifiers. The ***mb_h** parameter specifies the board identifier.

byte **mb_servo_off** (void ***mb_h**);

This function disables the power amplifiers for all axes. The ***mb_h** parameter specifies the board identifier.

MOTION PROCESSOR DRIVER

The Motion Processor Driver (pmd1401.c) provides a set of functions supporting the MP Interface. The set includes motion primitives, MP parameter setting functions (PID coefficients, speeds, motion profiles, etc.) and other MP related functions.

Function summary is given in the following table:

Function name and parameter	Description
byte mp_init (word io_a, void **mp_h)	Initialize MP
byte mp_down (void *mp_h)	Shut down MP and free system resources
byte mp_update (void *mp_h, byte a_no)	Load PID and trajectory coefficients into MP
byte mp_kp_set (void *mp_h, byte a_no, word kp)	Set new KP value
byte mp_kp_get (void *mp_h, byte a_no, word *kp)	Read current KP value
byte mp_kd_set (void *mp_h, byte a_no, word kd)	Set new KD value
byte mp_kd_get (void *mp_h, byte a_no, word *kd)	Read current KD value
byte mp_ki_set (void *mp_h, byte a_no, word ki)	Set new KI value
byte mp_ki_get (void *mp_h, byte a_no, word *ki)	Read current KI value
byte mp_il_set (void *mp_h, byte a_no, word il)	Set new integration limit (IL) value
byte mp_il_get (void *mp_h, byte a_no, word *il)	Read current integration limit (IL) value
byte mp_profile_set (void *mp_h, byte a_no, byte p_se)	Set velocity profile
byte mp_profile_get (void mp_h, byte a_no, byte *p_se)	Get velocity profile code
byte mp_vel_set (void *mp_h, byte a_no, dword vel)	Set maximum velocity
byte mp_vel_get (void *mp_h, byte a_no, dword *vel)	Read maximum velocity
byte mp_tacc_set (void *mp_h, byte a_no, dword acc)	Set acceleration for trapezoidal profiles

Function name and parameter	Description
byte mp_tacc_get (void *mp_h, byte a_no, dword *acc)	Read acceleration for trapezoidal profiles
byte mp_macc_set (void *mp_h, byte a_no, dword acc)	Set maximum acceleration for S-curve profile
byte mp_macc_get (void mp_h, byte a_no, dword *acc)	Read maximum acceleration for S-curve profile
byte mp_jerk_set (void *mp_h, byte a_no, dword jerk)	Set jerk parameter for S- curve profile
byte mp_jerk_get (void *mp_h, byte a_no, dword *jerk)	Read jerk parameter for S- curve profile
byte mp_ratio_set (void *mp_h, byte a_no, dword ratio)	Set the gear ratio for the axis a_no
byte mp_ratio_get (void *mp_h, byte a_no, dword *ratio)	Reads the gear ratio for the axis a_no
byte mp_dpos_set (void *mp_h, byte a_no, dword pos)	Set desired position
byte mp_apos_set (void *mp_h, byte a_no, dword pos)	Set current position
byte mp_apos_get (void *mp_h, byte a_no, dword *pos)	Read current position
byte mp_tpos_get (void *mp_h, byte a_no, dword *pos)	Read target position
byte mp_stop (void *mp_h, byte a_no)	Stop motion with maximum acceleration
byte mp_zero (void *mp_h, byte a_no)	Make current position equal to zero
byte mp_synch (void *mp_h, byte a_no)	Synchronize current and target positions
byte mp_capt_get (void *mp_h, byte a_no, dword *val)	Read captured (latched) position
byte mp_stat_get (void *mp_h, byte a_no, word *val)	Read axis status
byte mp_mtr_off (void *mp_h, byte a_no)	MP servo off
byte mp_mtr_on (void *mp_h, byte a_no)	MP servo on
byte mp_err_get (void *mp_h, byte a_no, word *err)	Read position error
byte mp_merr_set (void *mp_h, byte a_no, word merr)	Set position error limit
byte mp_merr_get (void *mp_h, byte a_no, word *merr)	Read position error limit

Function Descriptions

This section provides short descriptions of the functions contained in the LS-421 Motion Processor Driver.

byte **mp_init** (word ***io_a***, void ****mp_h***);

Initializes the MP on a board installed on a base address ***io_a***. This function checks whether MP is available and allocates the necessary resources for the functioning of the driver. A pointer to the MP identifier (of type *void*) will be returned in the ****mp_h*** parameter.

byte **mp_down** (void ***mp_h***);

This function shuts down MP and releases the system resources allocated to the driver. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_update** (void ***mp_h***, byte ***a_no***);

Loads the PID filter coefficients and trajectory parameters into MP for axis ***a_no***. Prior to the activation of this function the newly set values are stored in a buffer and are not used by the MP. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_kp_set** (void ***mp_h***, byte ***a_no***, word ***kp***);

This function sets a new value for the KP coefficient of the PID filter for the axis specified by the ***a_no*** parameter. In order to load the new value in the MP, **mp_update** function must be called. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_kp_get** (void ****mp_h***, byte ***a_no***, word ****kp***);

Reads the current value of the KP PID filter coefficient for the axis specified by the ***a_no*** parameter. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_kd_set** (void ***mp_h***, byte ***a_no***, word ***kd***);

This function sets a new value for the KD coefficient of the PID filter for the axis specified by the ***a_no*** parameter. In order to load the new value in the MP, **mp_update** function must be called. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_kd_get** (void ****mp_h***, byte ***a_no***, word ****kd***);

Reads the current value of the KD PID filter coefficient for the axis specified by the ***a_no*** parameter. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_ki_set** (void ***mp_h***, byte ***a_no***, word ***ki***);

This function sets a new value for the KI coefficient of the PID filter for the axis specified by the ***a_no*** parameter. In order to load the new value in the MP, **mp_update** function must be called. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_ki_get** (void ***mp_h**, byte ***a_no**, word ***ki**);

Reads the current value of the KI PID filter coefficient for the axis specified by the ***a_no*** parameter. The ***mp_h** parameter specifies the MP identifier.

byte **mp_il_set** (void ***mp_h**, byte ***a_no**, word ***il**);

This function sets a new value for the integration limit of the PID filter for the axis specified by the ***a_no*** parameter. In order to load the new value in the MP, **mp_update** function must be called. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_il_get** (void ***mp_h**, byte ***a_no**, word ***il**);

Reads the current value of the integration limit of the PID filter for the axis specified by the ***a_no*** parameter. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_profile_set** (void ***mp_h**, byte ***a_no**, byte ***p_se**);

Sets a new velocity profile, specified by the ***p_se*** parameter, for the axis ***a_no***. The ***mp_h*** parameter specifies the MP identifier. The ***p_se*** parameter should take one of the values given by the following constants:

- PMD_T_PROF – trapezoidal trajectory motion;
- PMD_S_PROF – s-curve trajectory motion;
- PMD_V_PROF – velocity contour;
- PMD_G_PROF – e-gear motion;

byte **mp_profile_get** (void ***mp_h**, byte ***a_no**, byte ***p_se**);

Reads the velocity profile code for the axis ***a_no***. The value is returned in the location pointed by ***p_se*** parameter. The ***mp_h*** parameter specifies the MP identifier. The following constants give the correspondence between the profile type and the coded value:

- PMD_T_PROF – trapezoidal trajectory motion;
- PMD_S_PROF – s-curve trajectory motion;
- PMD_V_PROF – velocity contour;
- PMD_G_PROF – e-gear motion;

byte **mp_vel_set** (void ***mp_h**, byte ***a_no**, dword ***vel**);

Sets the maximum velocity for the axis ***a_no***. In order to load the new value in the MP the ***mp_update*** function must be called. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_vel_get** (void ***mp_h***, byte ***a_no***, dword ***vel***);

This function reads the maximum velocity to which the axis ***a_no*** could accelerate. The value is returned in the location pointed by the ***vel*** parameter. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_tacc_set** (void ***mp_h***, byte ***a_no***, dword ***acc***);

This function sets the acceleration during the acceleration/deceleration phase of the velocity profile for the ***a_no*** axis. In order to load the new value in the MP the ***mp_update*** function must be called. The ***mp_h*** parameter specifies the MP identifier. This function can only be used for axes with current velocity profile PMD_T_PROF or PMD_V_PROF.

byte **mp_tacc_get** (void ***mp_h***, byte ***a_no***, dword ***acc***);

This function reads the acceleration during acceleration/deceleration phase of the velocity profile for the ***a_no*** axis. The value is returned in the location pointed by ***acc***. The ***mp_h*** parameter specifies the MP identifier. This function can only be used for axes with current velocity profile PMD_T_PROF or PMD_V_PROF.

byte **mp_macc_set** (void ***mp_h***, byte ***a_no***, dword ***acc***);

This function sets the maximum acceleration for the ***a_no*** axis. In order to load the new value in the MP the ***mp_update*** function must be called. The ***mp_h*** parameter specifies the MP identifier. This function can only be used for axes with current velocity profile PMD_S_PROF.

byte **mp_macc_get** (void ***mp_h***, byte ***a_no***, dword ***acc***);

This function reads the maximum acceleration for the ***a_no*** axis. The value is returned in the location pointed by ***acc***. The ***mp_h*** parameter specifies the MP identifier. This function can only be used for axes with current velocity profile PMD_S_PROF.

byte **mp_jerk_set** (void ***mp_h***, byte ***a_no***, dword ***jerk***);

This function sets the jerk motion parameter (derivative of acceleration) for the ***a_no*** axis. In order to load the new value in the MP the ***mp_update*** function must be called. The ***mp_h*** parameter specifies the MP identifier. This function can only be used for axes with current velocity profile PMD_S_PROF.

byte **mp_jerk_get** (void ***mp_h***, byte ***a_no***, dword ***jerk***);

This function reads the maximum acceleration for the ***a_no*** axis. The value is returned in the location pointed by ***jerk***. The ***mp_h*** parameter specifies the MP identifier. This function can only be used for axes with current velocity profile PMD_S_PROF.

byte **mp_ratio_set** (void ***mp_h***, byte ***a_no***, dword ***ratio***);

This function sets the gear ratio between the leading axis and the axis ***a_no***. In order to load the new value in the MP the ***mp_update*** function must be called. The ***mp_h*** parameter specifies the MP identifier. This function can only be used for axes with current velocity profile PMD_G_PROF. The combinations of

master/slave axes are fixed due to MP hardware limitations. Axis #1 can be a slave to axis #3 and axis #2 can be a slave to axis #4.

byte **mp_ratio_get** (void ***mp_h***, byte ***a_no***, dword ***ratio***);

This function reads the gear ratio between the leading axis and the axis ***a_no***. The value is returned in the location pointed by ***ratio***. The ***mp_h*** parameter specifies the MP identifier. This function can only be used for axes with current velocity profile PMD_G_PROF.

byte **mp_dpos_set** (void ***mp_h***, byte ***a_no***, dword ***pos***);

This function sets the desired position for the ***a_no*** axis. In order to load the new desired position value in the MP the ***mp_update*** function must be called. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_apos_set** (void ***mp_h***, byte ***a_no***, dword ***pos***);

This function sets the current position of the ***a_no*** axis. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_apos_get** (void ***mp_h***, byte ***a_no***, dword ***pos***);

Reads the current position of the ***a_no*** axis. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_tpos_get** (void ***mp_h***, byte ***a_no***, dword ***pos***);

Reads the target position of the ***a_no*** axis for the next MP cycle. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_stop** (void ***mp_h***, byte ***a_no***);

Stops the motion along the ***a_no*** axis with maximum acceleration. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_zero** (void ***mp_h***, byte ***a_no***);

Sets the current position of the ***a_no*** axis to zero. In order to load the new current position value in the MP the ***mp_update*** function must be called.

byte **mp_synch** (void ***mp_h***, byte ***a_no***);

Synchronizes the positions. The desired position of the a_no axis is made equal to the current one. In order to complete the synchronization it is necessary to call the mp_update function. The *mp_h is the coprocessor's identifier.

byte **mp_capt_get** (void ***mp_h***, byte ***a_no***, dword ***val***);

Reads the register for fast position capturing of the axis ***a_no*** and resets the capture logic of the MP. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_stat_get** (void ***mp_h***, byte ***a_no***, word ***val***);

Reads the current status of the ***a_no*** axis and resets MP axis status register. See MP Status Information" on page 16. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_mtr_off** (void ***mp_h***, byte ***a_no***);

This function switches off the MP servo loop for axis ***a_no***. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_mtr_on** (void ***mp_h***, byte ***a_no***);

This function switches on the MP servo loop for axis ***a_no***. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_err_get** (void ***mp_h***, byte ***a_no***, word ***err***);

This function reads the position error for the ***a_no*** axis. Position error is the difference between the target and current positions. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_merr_set** (void ***mp_h***, byte ***a_no***, word ***merr***);

This function sets the maximum position error tolerated for the ***a_no*** axis. In case when the absolute value of the position error exceeds the specified ***merr*** limit, the MP automatically deactivates the servo channel for that axis. The ***mp_h*** parameter specifies the MP identifier.

byte **mp_merr_get** (void ***mp_h***, byte ***a_no***, word ***merr***);

This function reads the maximum position error for the ***a_no*** axis. The ***mp_h*** parameter specifies the MP identifier.

LIBRARY OF MACRO CONTROL PROCEDURES

The Library of Macro Control Procedures (mcp.c) is a set of functions performing control sequences for the following tasks:

- Initialization of the controller
- Homing
- Axis motion check
- Diagnostics

Function name and parameter	Description
char *mcp_error (byte e_no)	Return Text Error Message
byte mcp_init (word io_a, void **mb_h, void **mp_h)	Initialize Controller
byte mcp_shutdown (void *mb_h, void *mp_h)	Shutdown Controller
byte mcp_servo_on (void *mb_h, void *mp_h)	Servo ON
byte mcp_servo_off (void *mb_h, void *mp_h);	Servo OFF
byte mcp_synch (void *mp_h, byte a_no)	Synchronize target and actual position
int cp_wait_axis (void *mb_h, void *mp_h, byte a_no)	Wait motion to stop
int mcp_stop_axis (void *mb_h, void *mp_h, byte a_no)	Stop axis motion
int mcp_home_axis (void *mb_h, void *mp_h, byte a_no, byte i_no)	Home the axis

Function Descriptions

This section provides short descriptions of the functions contained in the LS-421 Library of Macro Control Procedures.

char *mcp_error (byte e_no);

This function transforms the error code ***e_no*** generated by the Motion Processor driver and the System and Peripheral Interface driver into a text error message.

byte mcp_init (word *io_a*, void ***mb_h*, void ***mp_h*);

This function initializes the servo controller at base address ***io_a***. It checks if the hardware of the servo controller is available and allocates all necessary resources for the operation of the other software modules. The ***mb_h* and ***mp_h* parameters return pointers to the board identifier and the Motion Processor identifier, respectively.

byte mcp_shutdown (void **mb_h*, void **mp_h*);

This procedure reclaims the system resources allocated to the driver software. The ***mb_h* and ***mp_h* parameters are pointers to the identifiers of the board and the Motion Processor, respectively.

byte mcp_servo_on (void **mb_h*, void **mp_h*);

This function switches on the servo loop. This includes activation of all axes supported by the MP and enabling the amplifiers The ***mb_h* and ***mp_h* parameters are pointers to the identifiers of the board and the Motion Processor, respectively.

byte mcp_servo_off (void **mb_h*, void **mp_h*);

This procedure disables the motor power amplifiers. The ***mb_h* and ***mp_h* parameters are pointers to the identifiers of the board and the Motion Processor, respectively.

byte mcp_synch (void **mp_h*, byte *a_no*);

This function synchronizes the target and current position and sets the position error to zero along the ***a_no*** axis. This has to be performed each time the system has stopped accidentally (e.g. after the emergency button is pressed). The ***mb_h*** parameter is a pointer to the board identifier.

int mcp_wait_axis (void **mb_h*, void **mp_h*, byte *a_no*);

This function causes the program flow to waits until motion along ***a_no*** axis stops. The ***mb_h* and ***mp_h*** parameters are pointers to the identifiers of board and the Motion Processor, respectively.

int **mcp_stop_axis** (void ***mb_h***, void ***mp_h***, byte ***a_no***);

This function stops the motion along axis ***a_no*** with maximum acceleration and waits until the MP confirms that the axis has stopped. The ***mb_h*** and ***mp_h*** parameters are pointers to the identifiers of the board and the Motion Processor, respectively.

int **mcp_home_axis** (void ***mb_h***, void ***mp_h***, byte ***a_no***, byte ***i_no***);

This function performs a homing procedure for the ***a_no*** axis. The *i_no* parameter designates the input, which is used as a home switch for the ***a_no*** axis. The ***mb_h*** and ***mp_h*** parameters are pointers to the identifiers of the board and the Motion Processor, respectively.

TEST PROGRAM FOR LS-421 SERVO CONTROLLER

The example program (ls421t.exe) has two major objectives. In the first place, it has to provide means for initial testing and acquaintance with servo systems based on the LS-421 controller. The program supports all basic commands necessary for setting the parameters of the PID filter and execution of simple movements. Second, the program can be used as an illustration to the software model of LS-421 servo controller. The sources for all software components are provided on the accompanying diskette.

The program may be used to control a servo system by interpreting text commands for setting parameters and for movement along the different axes. The command syntax has the following format: a keyword identifying the respective command is given first, followed by arguments (if any), which are space delimited. In case the keyword cannot be identified due wrong syntax, omitted or extra arguments, the interpreter prints an error message.

The program is capable of handling both decimal and hex values of the arguments. Numeric arguments designating controller resources, e.g. axes, digital inputs or outputs are numbered starting from zero. For instance, the controller axes, four in total, are labeled zero to three.

The program does not convert the units of measure used by the MC1401 motion processor and by the controller hardware. For example, velocity is measured in encoder's counts per MP cycle, and not in radians per second. Also, some trajectory parameters are given in the so-called 16/16 format, meaning that in a 32-bits value the upper 16 bits represent the integer part, and the lower 16 bits - the fractional part. For further details about units of measure and data presentation see the MC1401 Reference Manual.

When the program is loaded, it executes its configuration file, called *t.ini,* unless otherwise stated in the command line. In case the file cannot be opened, the program prints an error message and switches to interactive mode.

The configuration file is in ASCII text format. Essentially, this is a script file, which may contain all commands recognized by the interpreter (described below in section 'Test program commands'). In case an error occurs during the execution of a command, an error message is displayed and the program switches to interactive mode immediately afterwards.

The distribution package includes a sample configuration file t.ini. It reflects a particular configuration of a demo servo system and may require modifications when used with a different setup. The PID digital filter and the trajectory generating parameters are the most likely to need adjustments when this file is used with a real system.

Servo Controller Status Visualization

During operation, the status of the servo controller is displayed in the upper half of the screen. This information is constantly updated during the keyboard idle loop. Until the controller is fully initialized a reminder that the system is not yet operational is displayed on screen. After initialization the screen looks as follows:

The top line displays the current status of the servo controller. This line displays error messages and/or one of the following statuses:

- *The system power is off*. The controller has been initialized but the system power supply is still off.
- *The amplifiers are disabled*. The system power supply is on but the motor power amplifiers are not enabled.
- *The amplifiers are enabled*. The amplifiers are enabled and the servo-loop is closed.

The next few lines display four columns of data, one per axis, reflecting the current state of each servo axis of the controller. The columns have the following fields:

astat: <num>/<stat>

This is the identifier of the axis followed by the current state (active or inactive) separated by the symbol ”/”.

prof: <type>

This field contains information about the type of the velocity profile chosen for this axis. The possible types are: trapezoidal, s-curve, velocity and electronic gear.

vel: <value>

This field contains the velocity, to which the axis is accelerated, while moving along a set trajectory.

acc: <value>

Displays the acceleration for the respective axis. This field has a double function: when a s-curve profile is selected, it reflects the so-called maximum acceleration. For all the other profiles the field reflects the constant acceleration.

jerk: <value>

This field is displayed on the screen only when a s-curve profile has been selected for the respective axis. It reflects the jerk for reaching maximum acceleration.

apos: <value>

Displays the present position of the axis. It reflects the contents of the register for the absolute position of the MC1401 motion processor.

tpos: <value>

Displays the target position of the axis. It shows the desired position of the axis, towards the end of the current time interval (time slice) of the MP.

err: <value>

Displays the position error of the axis, which is the difference between the target position and the actual position.

Test Program Commands

This section lists the command recognized by the sample application program.

Quit

Exit the program.

Abbreviation: qui

Parameters: no

Description: The program automatically stops the amplifiers and switches off the servo system power supply. After that, the sample application program is terminated and control is given back to the operating system.

Echo <state>

Controls the local echo.

Abbreviation: none

Parameters: <state> new value of the flag, logical zero or one

Description: This command is usually applied during the execution of configuration files. When the *echo flag* is active, the program prints all commands being interpreted. Initially, the *echo* flag is off.

Example: echo 0

Init <address>

Initial identification and initialization of the servo controller.

Abbreviation: none

Parameters: <address> base input/output address of the controller

Description: This command sets the input/output address of the controller board. The program performs a number of checks in order to determine the functionality of the controller. At this stage all necessary resources are reserved. **Init** command does not switch on servo control or the system power supply.

> ***Note***
>
> *The only commands that can be executed before the initialization of the controller are **quit**, **echo** and **help**. **Init** command cannot be issued twice within one working session of the program.*

Example: init 0x280

Power

Switches on the system power supply .

Abbreviation: pow

Parameters: no

Description: This command activates the relay, controlling the system power supply (SPS) and verifies the switching of the relay by checking the corresponding bit. In case of failure, it displays an error message.

Nopower

Switches off the power supply of the servo system.

Abbreviation: nop

Parameters: no

Description: This command deactivates the relay, which controls the system power supply (SPS) and checks the corresponding bit. MP servo loop is automatically switched off.

Servo [<axis>]

Closes the MP servo loop and enables motor power amplifiers.

Abbreviation: ser

Parameters: <axis> axis number, optional

Description: When the command has been used without parameters the program checks the status of the controller and displays an error in case a protection circuit is activated. Otherwise, the servo control loop is closed for all MP axes and the motor power amplifiers are enabled (switched on).

When the axis parameter is specified, it is interpreted as an axis number (0 to 3) and the servo loop is closed for the corresponding MP axis. The state (enabled or disabled) of the motor power amplifiers *is not changed*.

Examples: servo

servo 1

Noservo [<axis>]

Switches off the MP servo loop and disables the motor power amplifiers.

Abbreviation: nos

Parameters: <axis> axis number, optional

Description: When the command has been used without parameters it disables the built-in power amplifiers and deactivates (opens) the servo control loop for all axes.

If the axis parameter is specified, it is interpreted as an axis number and the servo channel of the motion processor for this axis is deactivated (opened). The state of the motor power amplifiers *is not changed*. If amplifiers are enabled, other servo axes are still controllable by the motion processor.

Examples: noservo

noservo 0

Limit [<axis>] [<value>]

Sets or prints the value of the current limits.

Abbreviation: lim

Parameters: <axis> axis number, optional

<value> new value from zero to fifteen, optional

Description: When used without parameters, this command prints the values of the current limits for all controller axes.

If the first parameter is specified, but the second parameter is omitted, it is interpreted as an axis number and the value of the current limit for that axis is displayed on screen.

If both parameters are specified, the value of the second parameter is set as current limit for the axis specified by the first parameter. The values are dimensionless quantities in the range zero to fifteen. For additional information about the correspondence of these values to the actual motor current, see section “Setting Current Limits” on page 23.

Examples: limit

limit 0

limit 1 14

In [<input>]

Displays the state of the digital inputs.

Abbreviation: no

Parameters: <input> input number, optional

Description: When used without parameters, this command displays the state of all digital inputs. The values are given in four columns, with references to the number of the input and its current state.

If the input parameter is specified, it is interpreted as an input number and the state of that input is displayed.

Examples: in

in 11

Out [<output>] [<state>]

Sets or displays the state of the digital outputs.

Abbreviation: no

Parameters: <output> output number, optional

<state> new value of the output, logical zero or one, optional

Description: When used without parameters, this command displays the state of all outputs of the controller. The values are given in 4 columns, with references to the number of the output and its current state.

When the first parameter is specified but the second parameter is omitted, it is interpreted as an output number and the current state of that output is displayed.

If both parameters are specified, the state of the digital output referenced by the first parameter is set to the value of the second parameter.

Examples: out

out 1

out 7 1

Opt [<state>]

Sets or displays the condition of the relay output OPT.

Abbreviation: no

Parameters: <state> new value of the output, logical zero or one, optional

Description: When used without parameters, the command displays the current state of the relay output OPT.

When the state parameter is specified, the state of the OPT the relay output changes according to that value.

Examples: opt

opt 1

KP [<axis>] [<value>]

Sets or displays the values of the KP coefficient used by the motion processor PID filter.

Abbreviation: no

Parameters: <axis> axis number, optional

<value> new value in MP measuring units, optional

Description: When used without parameters, this command displays the values of the KP coefficient for all servo axes.

When the first parameter is specified but the second parameter is omitted, it is interpreted as axis number and the current value of the KP coefficient for that axis is displayed.

If both parameters are specified, the value of KP coefficient for the axis referenced by the first parameter is set to the value of the second parameter.

Examples: kp

kp 0

kp 3 100

KD [<axis>] [<value>]

Sets or displays the values of the KD coefficient used by the motion processor PID filter.

Abbreviation: no

Parameters: <axis> axis number, optional

<value> new value in MP measuring units, optional

Description: When used without parameters, this command displays the values of the KD coefficients for all servo axes.

When the first parameter is specified but the second parameter is omitted, it is interpreted as axis number and the current value of the KD coefficient for that axis is displayed.

If both parameters are specified, the value of KD coefficient for the axis referenced by the first parameter is set to the value of the second parameter.

Examples: kd

kd 0

kd 3 2000

KI [<axis>] [<value>]

Sets or displays the values of the KI coefficient used by the motion processor PID filter.

Abbreviation: no

Parameters: <axis> axis number, optional

<value> new value in MP measuring units, optional

Description: When used without parameters, this command displays the values of the KI coefficients for all servo axes.

When the first parameter is specified but the second parameter is omitted, it is interpreted as axis number and the current value of the KI coefficient for that axis is displayed.

If both parameters are specified, the value of KI coefficient for the axis referenced by the first parameter is set to the value of the second parameter.

Examples: ki

ki 0

ki 3 100

IL [<axis>] [<value>]

Sets or displays the values of the integral limit coefficient used by the motion processor PID filter.

Abbreviation: no

Parameters: <axis> axis number, optional

<value> new value in the MC1401 measuring units, optional

Description: When used without parameters, this command displays the values of the integral limits for all servo axes.

When the first parameter is specified but the second parameter is omitted, it is interpreted as axis number and the current value of the IL coefficient for that axis is displayed.

If both parameters are specified, the value of IL coefficient for the axis referenced by the first parameter is set to the value of the second parameter.

Examples: il

il 0

il 3 100

Velocity [<axis>] [<value>]

Sets or displays the maximum velocity to which the servo axes are accelerated.

Abbreviation: vel

Parameters: <axis> axis number, optional

<value> new value in MP measuring units, optional

Description: When used without parameters, this command displays the maximum velocities of all servo axes.

When the first parameter is specified but the second parameter is omitted, it is interpreted as axis number and the current value of the maximum velocity for that axis is displayed.

If both parameters are specified, the maximum velocity for the axis referenced by the first parameter is set to the value of the second parameter.

Examples: vel

vel 0

vel 2 200000

Acceleration [<axis>] [<value>]

Sets or displays the acceleration used during acceleration/deceleration phases of the velocity profile.

Abbreviation: acc

Parameters: <axis> axis number, optional

<value> new value in MP measuring units, optional

Description: When used without parameters, this command displays the acceleration setting for all servo axes.

When the first parameter is specified but the second parameter is omitted, it is interpreted as axis number and the current value of the acceleration for that axis is displayed.

If both parameters are specified, the acceleration for the axis referenced by the first parameter is set to the value of the second parameter.

Examples: acc

acc 0

acc 2 2000

Jerk [<axis>] [<value>]

Sets or displays the jerk (derivative of acceleration) for an axis with a s-curve velocity profile.

Abbreviation: jer

Parameters: <axis> axis number, optional

<value> new value in MP measuring units, optional

Description: When typed without parameters, this command displays the jerk of all servo axes.

When the first parameter is specified but the second parameter is omitted, it is interpreted as axis number and the current value of the jerk for that axis is displayed.

If both parameters are specified, the jerk of the axis referenced by the first parameter is set to the value of the second parameter.

Examples: jer

jer 0

jer 0 20000

Profile [<axis>] [<type>]

Sets or displays the velocity profile used for a given axis.

Abbreviation: pro

Parameters: <axis> axis number, optional

<value> new profile code [0,1,2,3], optional

Description: When typed without parameters, this command displays the current profile code of all axes.

When the first parameter is specified but the second parameter is omitted, it is interpreted as axis number and the current velocity profile code for that axis is displayed.

If both parameters are specified, the velocity profile code for the axis referenced by the first parameter is set to the value of the second parameter.

The profiles are given by the following codes:

0 – trapezoidal
1 – s-curve
2 – velocity contouring
3 – e-gearing

Although all types of profiles may be selected, the program provides support for trapezoidal and s-curve profiles only. After initialization all axis profiles are trapezoidal.

Examples: pro

pro 0

pro 2 1

Position [<axis>] [<value>]

Sets or displays the current position of one or all axes.

Abbreviation: pos

Parameters: <axis> axis number, optional

<value> new value, number of encoder counts, optional

Description: When used without parameters, this command displays the current absolute positions for all servo axes.

When the first parameter is specified but the second parameter is omitted, it is interpreted as axis number and the current position for that axis is displayed.

If both parameters are specified, the position of the axis referenced by the first parameter is set to the value of the second parameter.

Examples: pos

pos 0

pos 1 40000

Absolute <axis> <value>

Sets an axis into motion until it reaches the designated absolute position.

Abbreviation: abs

Parameters: <axis> axis number

<value> new absolute position in encoder counts

Description: This command starts motion of a given axis to the specified desired position. Previously set velocity, acceleration and profile are used for that motion.

The first parameter is axis number that will be moved. The second parameter is the desired absolute position of this axis with respect to the zero position of this axis.

Example: abs 2 160000

Relative <axis> <value>

Sets an axis into motion until it reaches a desired position, relative to the current one.

Abbreviation: rel

Parameters: <axis> axis number

<value> new relative position in encoder counts

Description: This command starts motion of a given axis to the specified desired position relative to the current position. Previously set velocity, acceleration and profile are used for that motion.

The first parameter is axis number that will be moved. The second parameter is the desired travel of this axis with respect to its current position.

Example: rel 2 10000

Error [<axis>] [<value>]

Sets or displays the maximum admissible position error.

Abbreviation: err

Parameters: <axis> axis number, optional

<value> new value in MP measuring units, optional

Description: When used without parameters, this command displays the values of the maximum position errors for all servo axes.

When the first parameter is specified but the second parameter is omitted, it is interpreted as axis number and the current value of the maximum admissible position for that axis is displayed.

If both parameters are specified, the maximum admissible position error for the axis referenced by the first parameter is set to the value of the second parameter.

Example: err

err 0

err 2 500

Wait [<axis>]

Waits until the motion of one or more axes has stopped.

Abbreviation: wai

Parameters: <axis> axis number, optional

Description: When the command is entered without parameters, the program waits until the motion of all axes supported by the controller stops.

When the command is followed by a parameter, it is interpreted as an axis number. In this case the program waits until motion along that axis stops.

Example: wai

wai 1

Stop [<axis>]

Stops the motion on one or more axes with maximum acceleration.

Abbreviation: sto, stp

Parameters: <axis> axis number, optional

Description: When the command is entered without parameters, the program gives consecutive commands to the controller and waits until the motion of all axes stops. Stopping is performed with the maximum possible acceleration.

When the axis parameter is specified, then motion is stopped along that axis only. The program waits until motion along the specified axis stops. Stopping is with the maximum possible acceleration.

Example: stp

sto 2

Home <axis> <input>

Positions a specific axis in its zero position in accordance with state transition a given input.

Abbreviation: no

Parameters: <axis> axis number

<input> input number to be used as a home switch

Description: When this command is entered the program executes a homing procedure for the selected axis using the following program sequence:

1. A direction for searching the home switch state transition is selected. If the selected input is not active, the direction of the motion is negative (towards the greatest negative coordinate allowed by the MP). If the input is active, the direction is positive, i.e. towards the greatest positive coordinate.
2. The axis starts moving in the selected direction at low velocity and high acceleration. In the meantime, the state of the selected input is being monitored. The motion is stopped with maximum acceleration when the input state changes.
3. If the motion was in a negative direction, the program moves the axis back to a position where the state of the input will be again 1. Thus, it is guaranteed that the direction when finding the input is always the same and the input itself in not active at the initial state of the axis.
4. If homing will be performed without latching the encoder index signal, go to step 8.
5. The program selects the encoder index signal as a source for fast position capturing. The MP position capture register is reset.
6. The axis starts moving in a positive direction at low velocity and high acceleration. The motion is stopped with maximum acceleration at the moment when the motion processor registers the index pulse (latches current position based on the encoder index signal).
7. The axis moves to the latched position.
8. The actual position of the axis is set to zero.

Example: home 1 15

VI. Appendix A: Memory Map

LS-421 is designed as a standard ISA bus peripheral device. To save I/O space, LS-421 uses only 16 bytes from the lowest 1024 I/O addresses range. The rest of LS-421 registers occupy the space located at offset 0x400, 0x800 and 0xC00. For instance, if LS-421 base address is set to 0x280, the following I/O address are in use – 0x280÷0x28F, 0x680÷0x68F, 0xA80÷0xA8F, 0xE80÷0xE8F. Eight base addresses are available, depending on J1 setting – 0x280, 0x1280, 0x2280, 0x3280, 0x2A0, 0x12A0, 0x22A0, 0x32A0. The table below shows LS-421 register location relatively to controller base address. The offsets are in hexadecimal notation.

	READ									WRITE								
	D7	D6	D5	D4	D3	D2	D1	D0		D7	D6	D5	D4	D3	D2	D1	D0	
0	MC1401A DATA REGISTER								0	MC1401A DATA REGISTER								0
1	READY								1	MC1401A COMMAND REGISTER								1
2									2									2
3									3									3
4									4									4
5									5									5
6									6									6
7									7									7
8	IN7	IN6	IN5	IN4	IN3	IN2	IN1	IN0	8									8
9	IN15	IN14	IN13	IN12	IN11	IN10	IN9	IN8	9									9
A	AE	RST	OPT	SPS	CL3	CL2	CL1	CL0	A	AE	RST	OPT	SPS					A
B	EMG	STP	MPF	OPF	EMG (L)	STP (L)	MPF (L)	OPF (L)	B									B
C	ADI						MPI		C	EN ADI								C
D	OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0	D	OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0	D
E									E	PCS3	PCS2	PCS1	PCS0				APE RES	E
F									F									F
400									400									400
401									401									401
402									402									402
403									403									403
404	APE D7	APE D6	APE D5	APE D4	APE D3	APE D2	APE D1	APE D0	404	REQ APE#0								404
405	APE D15	APE D14	APE D13	APE D12	APE D11	APE D10	APE D9	APE D8	405	REQ APE#1								405
406	APE D23	APE D22	APE D21	APE D20	APE D19	APE D18	APE D17	APE D16	406	REQ APE#2								406
407	APE RDY		BE	OF	OS	BA	PS	CE	407	REQ APE#3								407
408									408	CURRENT LIMIT #0 (0x0 - 0xF)								408
409									409	CURRENT LIMIT #1 (0x0 - 0xF)								409
40A									40A	CURRENT LIMIT #2 (0x0 - 0xF)								40A
40B									40B	CURRENT LIMIT #3 (0x0 - 0xF)								40B
40C									40C									40C
40D									40D									40D
40E									40E									40E
40F									40F									40F
807					ID3 = 0	ID2 = 0	ID1 = 0	ID0 = 1	807									807
C07					ID7 = 1	ID6 = 0	ID5 = 1	ID4 = 1	C07									C07

Table 1: LS-421 Memory Map

Logosol, Inc.
1155 Tasman Drive, Sunnyvale, CA 94089
USA
Phone: (408) 744-0974 Fax: (408) 744-0977
http://www.logosolinc.com

Document No 715000010

Printed in USA