MoCon

User's Guide

Document Number:	MPIA-MoCon-UG
Issue Number:	Issue 1.00
Issue Date:	20.08.2009
Prepared by:	MPIA Heidelberg

MoCon

Motion Controller for eight Axes

Max-Planck-Institute for Astronomy
Koenigstuhl 17
69117 Heidelberg, Germany
Tel.: +49 6221/528-0
Fax: +49 6221/528-246

Table of Content

Table of Content (continued)

List of Figures

List of Tables

1 Introduction

1.1 Important Notes

Always follow the safety and warning notes in this publication!

Hazardous situation
Possible consequence: Slight or minor injuries

Harmful situation
Possible consequence: Damage to the unit and the environment

To ensure trouble-free operation and fulfillment of any rights you must adhere to this technical manual.

1.2 Purpose / Intended Use

This user's manual contains information and references, necessary for safe operation and maintenance of the MoCon board.
Prior to using the unit (commissioning / assembly) the user is kindly requested to thoroughly read the instruction manual and comply with it in all sections.
Failure to read the instruction manual or to comply with the warnings and references contained herein can result in serious bodily injury or instrument damage.

Target group:
This user manual is directed to trained personnel.
The contents of this manual should be made available to these personnel and put into practice by them.

1.3 Abbreviations and Acronyms

Abbr.	
CAN	Controller Area Network
EEPROM	Electrically Erasable Programmable Read-Only Memory
EPLD	Erasable Programmable Logic Device
ESD	Electro Static Discharge
FPGA	Field Programmable Gate Array
I^2C	Inter-integrated circuit
IP	Internet Protocol
MoCon	Motion Controller
MPIA	Max-Planck-Institute for Astronomy
N.C.	Not Connected
NEGLIM	negative limit switch
PCB	Printed Circuit Board
POSLIM	positive limit switch
RJ	Registered Jack
SCL	Serial Clock Line
SDA	Serial Data Line
SSI	Synchronous Serial Interface
TTL	Transistor-transistor logic

Table 1: Abbreviations and Acronyms

1.4 Reference Documents

[1] MoCon Programmers Guide
[2] MPIA Makro Guide

2 Safety Notes

2.1 Personnel Qualifications

Design, installation and operation of systems using the MoCon may only be performed by qualified and trained personnel. These persons should be able to recognize and handle risks emerging from electrical, mechanical or electronical system parts.

WARNING !
By persons without the proper training and qualification damages to devices and persons might result!

2.2 Appropriate use

The motion controller MoCon is a PCB hardware for controlling up to eight motors and was developed for scientific instrumentation applications. For this purpose the unit is foreseen to be installed into 19" systems, which are specified according to IEC 60297. Depending on the range of application, the user may have to take care of the country-specific safety standards and accident prevention rules.

The MoCon may not be used:
- If the PCB shows visible damage
- with damaged connectors or cables
- if it no longer works properly

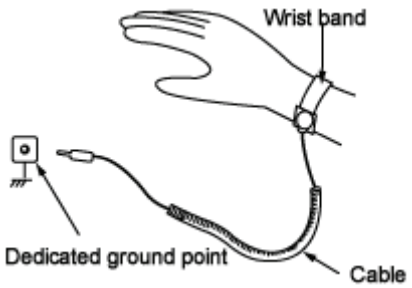
In such cases, the MoCon hardware has to be shut down and secured against unintentional use.

2.3 Temperature and Cooling Air Flow

The heat production of the MoCon is up to 8 W of thermal power. If mounting several MoCon devices in a system, ensure a sufficient cooling.

2.4 Electrostatic Discharge Precautions

Electro Static Discharge (ESD) is the leading cause of electronic component failure during and after the manufacturing process. High frequency and highly minimized active components are especially prone to damage by ESD. The Persons involved in handling the MoCon printed circuit board should always wear a grounded wrist strap. Transportation and storage of the electronics board should always be done with ESD protective packaging materials.



2.5 Warning about misuse

Inappropriate or incorrect use of the MoCon can give rise to application-specific hazards or damage to system components.

3 MoCon Hardware Design

3.1 MOCON Design

Figure 1: Top view of MOCON board

3.2 General

The MPIA motion controller printed circuit board MoCon is an intelligent, compact and user-friendly control unit for up to eight motion axis. The MoCon hardware is a multi-functional device which is able to control a huge variety of motors. Due to the modular concept, the MoCon electronics is capable to drive stepper motors as well as servo loop and DC motors. Core of the hardware is a 16 bit Infineon Controller which contains the firmware managing the communication and command functions. For motion controlling, the electronics is equipped with two chipset modules. Each chipset module can manage either 4 stepper or 4 servo motors. For each chipset module a RAM of 1MByte is provided to store the traces and external motion profiles. The two chipset modules have to be of the same kind.
The flexible design and configuration options allow it to built system with several MoCon boards in a chain. In addition, the MoCon hardware is capable to provide the following functionality:

- Motion profiles include S-curve, trapezoidal, velocity contouring, and electronic gearing
- Asymmetric acceleration and deceleration to custom program a trapezoidal motion profile
- Velocity and acceleration changes on-the-fly for trapezoidal and velocity-contouring profiles
- Incremental encoder quadrature input and parallel input for absolute encoder or resolver for on-the-fly motor stall detection
- Trace capabilities for system performance checks, maintenance and diagnostics
- Advanced breakpoint capability allows precise sequencing of events
- Two-directional limit switches, index input, and home indicator per axis
- Serial Interface (RS232), CAN Bus, Ethernet interface

By means of several MPIA amplifier board, the MoCon is capable to drive a huge variety of motors. Table 2 and 3 give an overview of the available MPIA stepper and servo amplifier boards.

Board	No. of channels	Max. output Current	Microsteps	Type	closed loop control
SMD8_V2	8	2,1 A (peak)	1/256	Chopper	no
SMD8CL_V1	8	2,1 A (peak)	1/256	Chopper	yes
SMD3_V3	3	5A (peak)	1/256	Chopper	no
LAMP_V2	1	10A (peak, max 100W)	1/2000	linear	no

Table 2: MPIA stepper motor amplifiers

Board	No. of channels	Output Current	Type	Closed loop control
DMD8_V2	8	3 A (55V)	PWM	yes
LAMP_V2	3	10A (max 100W)	linear	yes
SIGAMP_V1	8	TTL	PWM	yes

Table 3: MPIA servo motor amplifiers

In combination with the MPIA amplifiers, the MoCon can read the amplifier card and amplifier type information via an internal I^2C bus. This feature allows the MoCon to adapt the control signals and to restrict the functionality depending on the used amplifier.

3.3 Technical Characteristics

Technical Characteristics	
Supply Voltage	Regulated, filtered DC voltage Admissible voltage range: +4,8…+5,2 V DC Reinforced or double insulation between mains and secondary circuit is required
Max. run frequency	up to 5 M-pulses/sec
Step resolution	1/1 to 1/2000 depending on amplifier Board
Interfaces	RS232 / RS422, Ethernet and CAN
Connectors	X-1: Backplane connector (HARD METRIC 2mm) X-2 / X-3: CAN bus connectors (Mini DIN 4S) X-4: Serial Interface (Mini DIN 4S) X-5: Ethernet (RJ45) X-6 / X-7: digital outports (FTSH-20P) X-9 / X-10: digital inports (FTSH-20P) X-12 / X-14: hall sensor inputs (FTSH-26P)
Fuse	3,0 A picofuse
Ambient Temperature	Operation: 0...50°C Storage: -10...60°C Transport: -10...60°C
Dimensions WxHxD	250x100x26 (mm)
Weight	Approx. 300 g, depending on installed modules
Mounting	Inside standardized 19” systems according to IEC 60297

Table 4: Technical Characteristics

3.4 Functional Description

As shown in the block diagram (figure 2), the MOCON board consists of the following main parts:

- Phytec PhyCore Module with Infineon XC161 microprocessor
- Altera FPGA (PHY-EPLD)
- Altera FPGA for motor chipset control (PMD-EPLD)
- PMD Chipset module(s) for stepper or DC motors
- RAM for trace and external profiles
- registers for digital inputs and outputs
- Transceivers for bidirectional bus transfer
- Differential line drivers for incremental encoder inputs
- Protection circuits for limit switches

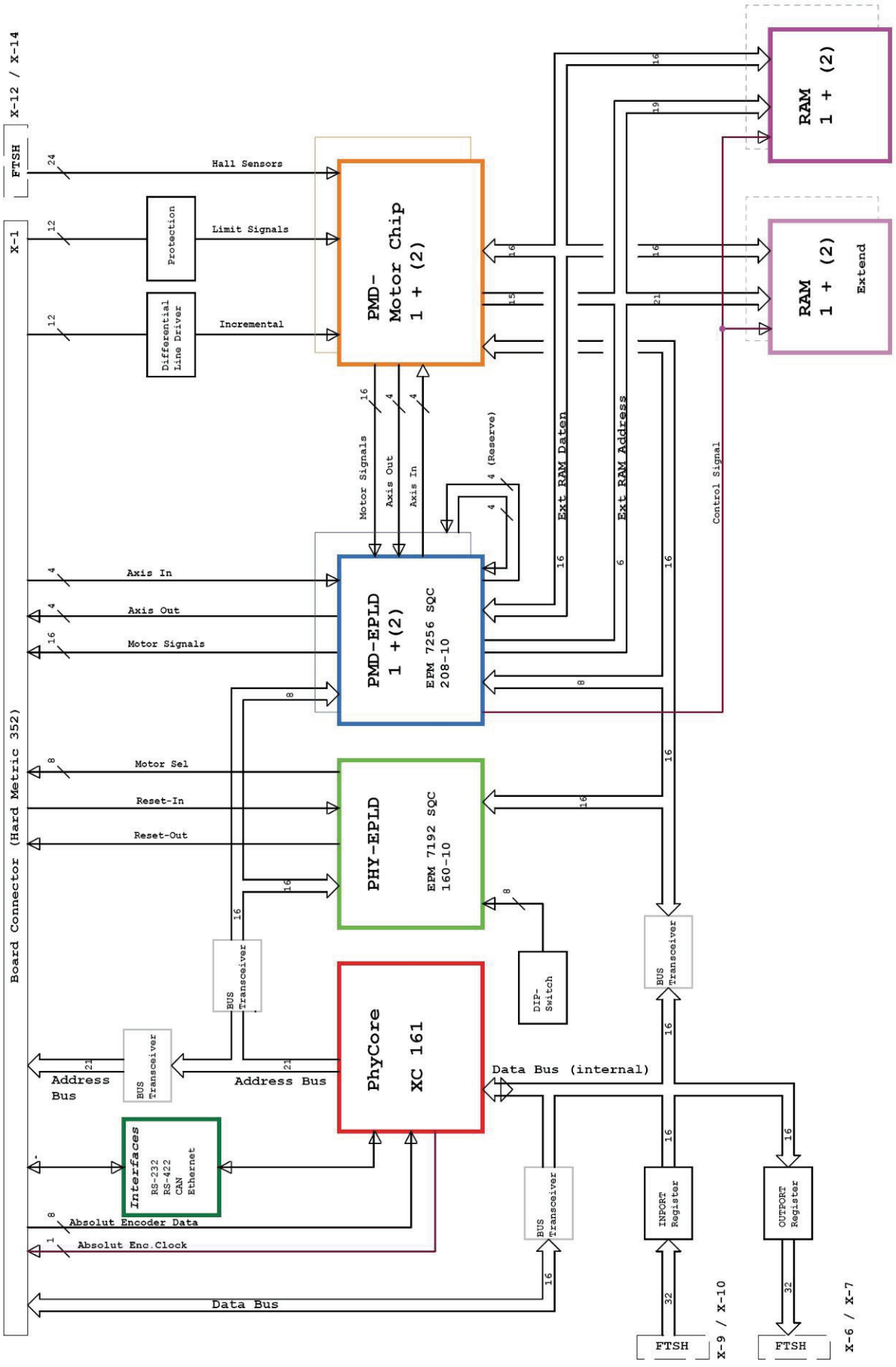


Figure 2: MOCON block diagram

3.5 Location on the printed circuit board

Figure 3 and 4 show the location of the main MoCon components and connectors at the top and bottom of the printed circuit board.

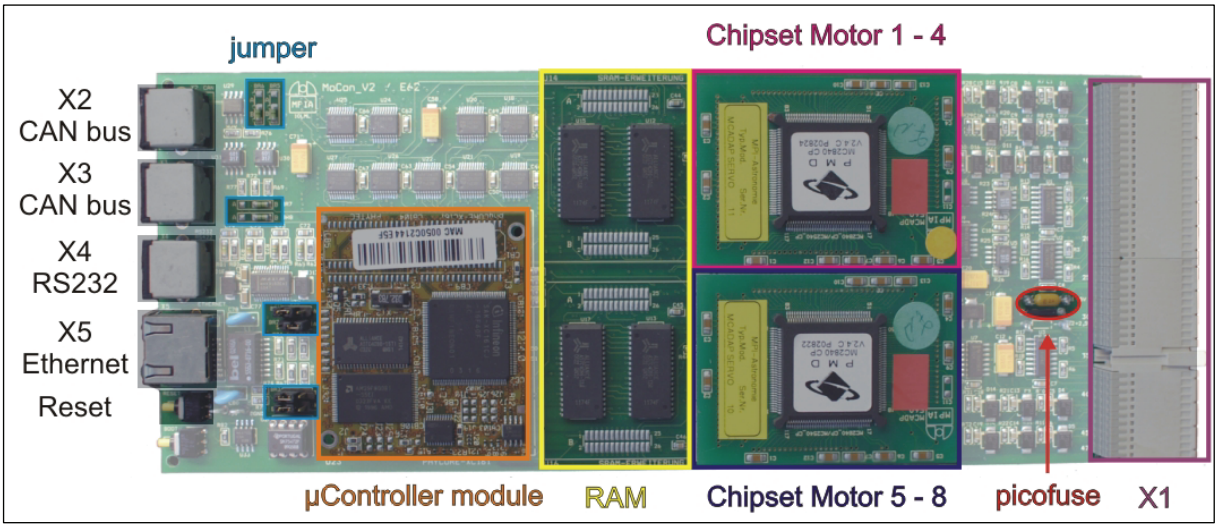


Figure 3: Location of components and connectors at top of MoCon pcb

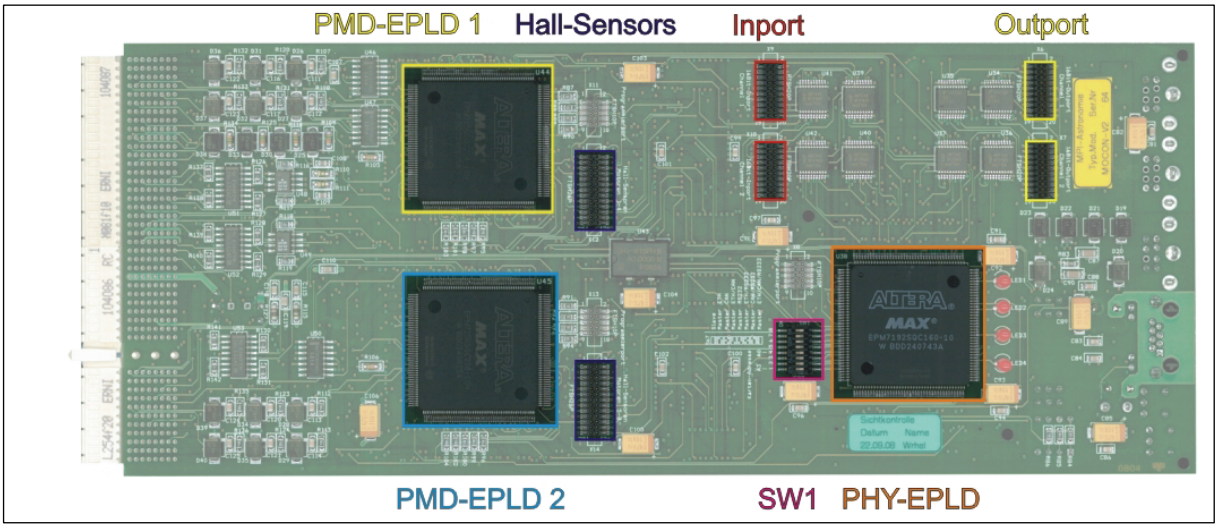


Figure 4: Location of components and connectors at bottom of MoCon pcb

3.6 Master / Slave Mode

Master/slave is a model of communication where one device or process has unidirectional control over one or more other devices. By using the master / slave communication via CAN bus it is possible to built up system joining several controllers a physical unit.

The MoCon supports both communication modes. The communication mode is configured via jumper seetings of SW1. Refer to chapter 4.1 for more details.

3.7 Limit and reference Switch Inputs

The signals for the limit and reference signals for each motor channel are routed via the main backplane connector X1. These signals are low-active and internally connected to a pull-up resistor and to a suppressor diode and a capacitor for overvoltage and reversal voltage protection. In addition, a Schmitt Trigger inverter is added for signal conditioning. Figure 5 shows the reference signal for channels 1 - 4 as an example. Table 5 gives a summary of all limit and reference signals available at the backplane connector X1.

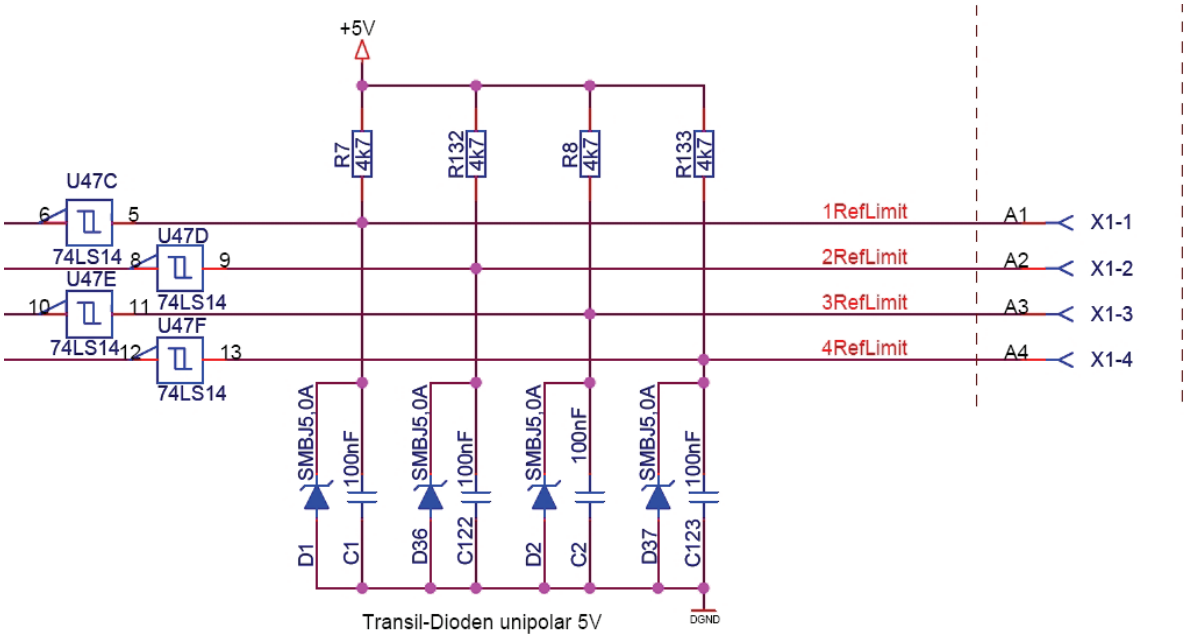


Figure 5: limit and reference switch interface

Channel	Signal	X1 contact
1	Ref Limit	1 a
1	Neg Limit	1 b
1	Pos Limit	1 c
2	Ref Limit	2 a
2	Neg Limit	2 b
2	Pos Limit	2 c
3	Ref Limit	3 a
3	Neg Limit	3 b
3	Pos Limit	3 c
4	Ref Limit	4 a
4	Neg Limit	4 b
4	Pos Limit	4 c

Channel	Signal	X1 contact
5	Ref Limit	39 a
5	Neg Limit	39 b
5	Pos Limit	39 c
6	Ref Limit	40 a
6	Neg Limit	40 b
6	Pos Limit	40 c
7	Ref Limit	41 a
7	Neg Limit	41 b
7	Pos Limit	41 c
8	Ref Limit	42 a
8	Neg Limit	42 b
8	Pos Limit	42 c

Table 5: Inputs for reference and limit switches at connector X-1

3.8 Absolute Encoder Interface

The MoCon board offers the option to connect absolute encoders having the standardized synchronous serial interface (SSI). Both the clock and input signals are controlled via a parallel port of the microcontroller.

3.8.1 Clock output for absolute encoder

The differential absolute encoder clock signal is generated by the microcontroller, adjusted to differential signal via a 74ALS86 and buffered by a MAX627 MOSFET driver. The output clock signal will have a frequency of up to 1.5 MHz. The clock signal is only active during the readout of data. External cables for connecting an encoder must be twisted pair and screened.

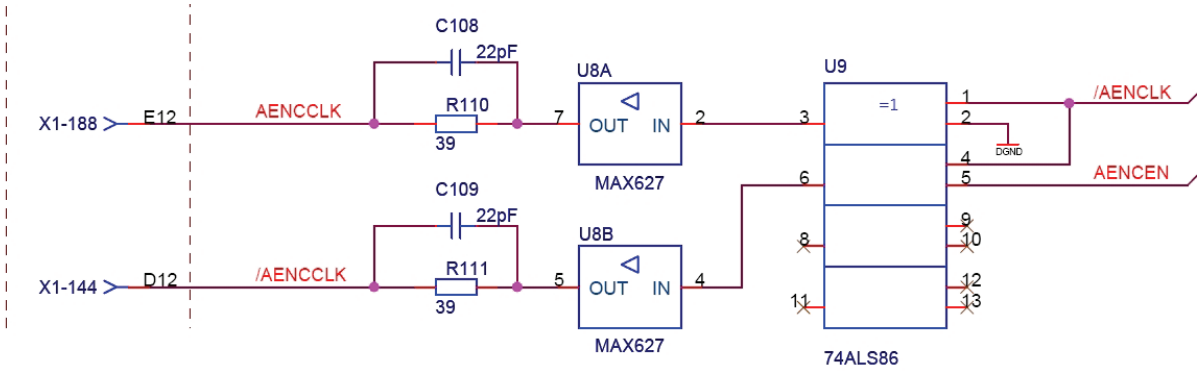


Figure 6: Clock Output for absolute encoder

3.8.2 Absolute encoder Inputs

The differential input signals for the absolute encoders are routed via the main backplane connector X1. A suppressor diode parallel to the signals works as a voltage protection. An optical coupler HCPL-0631 will isolate the external signals to internal supply and ground. Figure 6 shows one channel of the absolute encoder signal as an example. The external cables should be twisted pair and screened for reducing the noise. The microcontroller will capture all eight absolute encoder signals simultaneously.

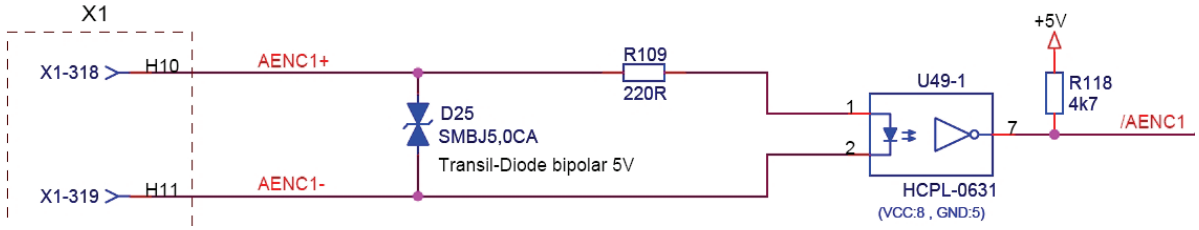


Figure 7: Absolute Encoder Input

Channel	Signal	X1 contact
1	AENC1+	10 h
1	AENC1-	11 h
2	AENC2+	10 g
2	AENC2-	11 g
3	AENC3+	10 f
3	AENC3-	11 f
4	AENC4+	10 e
4	AENC4-	11 e

Channel	Signal	X1 contact
5	AENC5+	10 d
5	AENC5-	11 d
6	AENC6+	10 c
6	AENC6-	11 c
7	AENC7+	10 b
7	AENC7-	11 b
8	AENC8+	10 a
8	AENC8-	11 a

Table 6: Inputs for absolute encoders

Refer to [1], command “GetAEncActualPosition” for more details how to read the absolute encoder inputs via the firmware functions.

3.9 Incremental Encoder Interface

The signals for the incremental encoders are routed via the main backplane connector X1 to differential line receivers 26LS32. This chip will adapt the differential encoder signal to a single ended one.

Refer to [1] for more details how to configure and read the incremental encoder inputs via the firmware functions.

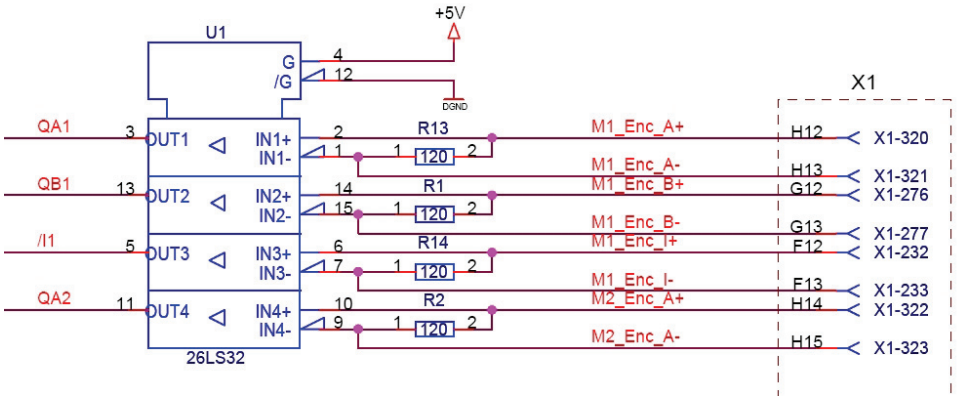


Figure 8: Incremental Encoder Interface

Channel	Signal	X1 contact
1	M1_Enc_A+	12 h
1	M1_Enc_A-	13 h
1	M1_Enc_B+	12 g
1	M1_Enc_B-	13 g
1	M1_Enc_I+	12 f
1	M1_Enc_I-	13 f
2	M2_Enc_A+	14 h
2	M2_Enc_A-	15 h
2	M2_Enc_B+	14 g
2	M2_Enc_B-	15 g
2	M2_Enc_I+	14 f
2	M2_Enc_I-	15 f
3	M3_Enc_A+	16 h
3	M3_Enc_A-	17 h
3	M3_Enc_B+	16 g
3	M3_Enc_B-	17 g
3	M3_Enc_I+	16 f
3	M3_Enc_I-	17 f
4	M4_Enc_A+	18 h
4	M4_Enc_A-	19 h
4	M4_Enc_B+	18 g
4	M4_Enc_B-	19 g
4	M4_Enc_I+	18 f
4	M4_Enc_I-	19 f

Channel	Signal	X1 contact
5	M5_Enc_A+	20 h
5	M5_Enc_A-	21 h
5	M5_Enc_B+	20 g
5	M5_Enc_B-	21 g
5	M5_Enc_I+	20 f
5	M5_Enc_I-	21 f
6	M6_Enc_A+	22 h
6	M6_Enc_A-	23 h
6	M6_Enc_B+	22 g
6	M6_Enc_B-	23 g
6	M6_Enc_I+	22 f
6	M6_Enc_I-	23 f
7	M7_Enc_A+	24 h
7	M7_Enc_A-	25 h
7	M7_Enc_B+	24 g
7	M7_Enc_B-	25 g
7	M7_Enc_I+	24 f
7	M7_Enc_I-	25 f
8	M8_Enc_A+	37 h
8	M8_Enc_A-	38 h
8	M8_Enc_B+	37 g
8	M8_Enc_B-	38 g
8	M8_Enc_I+	37 f
8	M8_Enc_I-	38 f

Table 7: Inputs for incremental encoders

3.10 Digital Inputs

There is 32 bit input port available on the MoCon board. This input port is divided to the connectors X9 and X10. X9 carries the Bits 0 to 15 and X10 bit 16 to 31. These input signals are routed directly to bi-directional transceivers 74ABT245. Via an enable signal, the microcontroller is capable to read the input information. These signals may be used for binary applications such as position detection and status control.

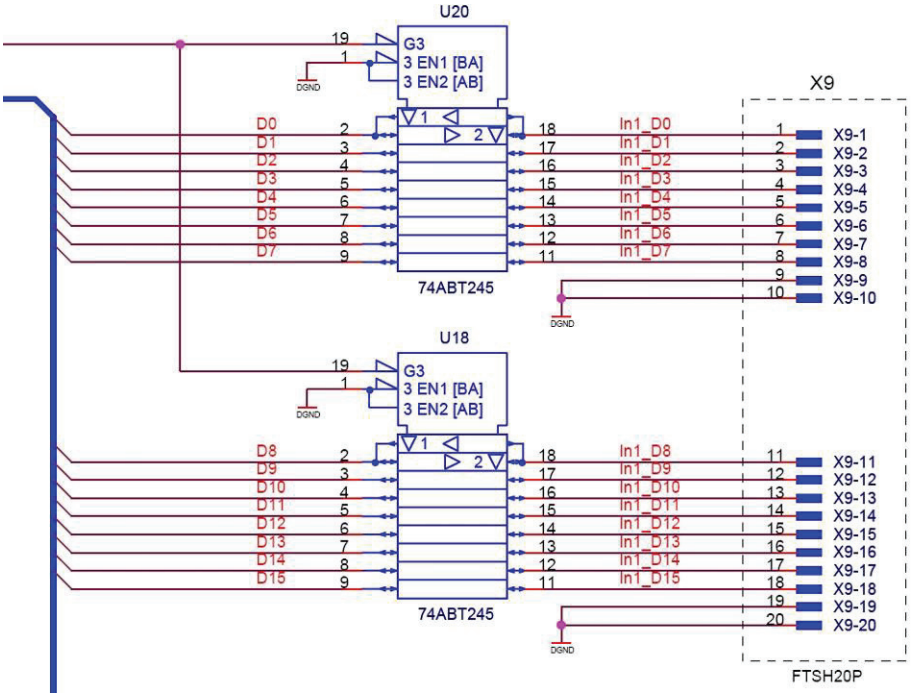


Figure 9: Digital Inputs

Since the inputs are not protected against over-voltages it is recommended to connect a suitable external protection circuit.

Refer to [1] for more details how to read the digital inputs via the firmware functions.

Bit	Signal name	X9 contact
D0	IN1_D0	1
D1	IN1_D1	2
D2	IN1_D2	3
D3	IN1_D3	4
D4	IN1_D4	5
D5	IN1_D5	6
D6	IN1_D6	7
D7	IN1_D7	8
D8	IN1_D8	11
D9	IN1_D9	12
D10	IN1_D10	13
D11	IN1_D11	14
D12	IN1_D12	15
D13	IN1_D13	16
D14	IN1_D14	17
D15	IN1_D15	18
all	digital ground	9,10,19,20

Bit	Signal name	X10 contact
D16	IN2_D0	1
D17	IN2_D1	2
D18	IN2_D2	3
D19	IN2_D3	4
D20	IN2_D4	5
D21	IN2_D5	6
D22	IN2_D6	7
D23	IN2_D7	8
D24	IN2_D8	11
D25	IN2_D9	12
D26	IN2_D10	13
D27	IN2_D11	14
D28	IN2_D12	15
D29	IN2_D13	16
D30	IN2_D14	17
D31	IN2_D15	18
all	digital ground	9,10,19,20

Table 8: Digital input signals

3.11 Digital Outputs

There is 32 bit output port, divided to the connectors X6 and X7. X6 carries the Bits 0 to 15, X7 bit 16 to 31. These outputs are routed via edge-triggers D-type flip-flops 74ALS273 and may be used for controlling binary functions such as shutters or breaks. Please note, that the output current of the signals is limited to 2.6 mA (high) resp. 24 mA (low).

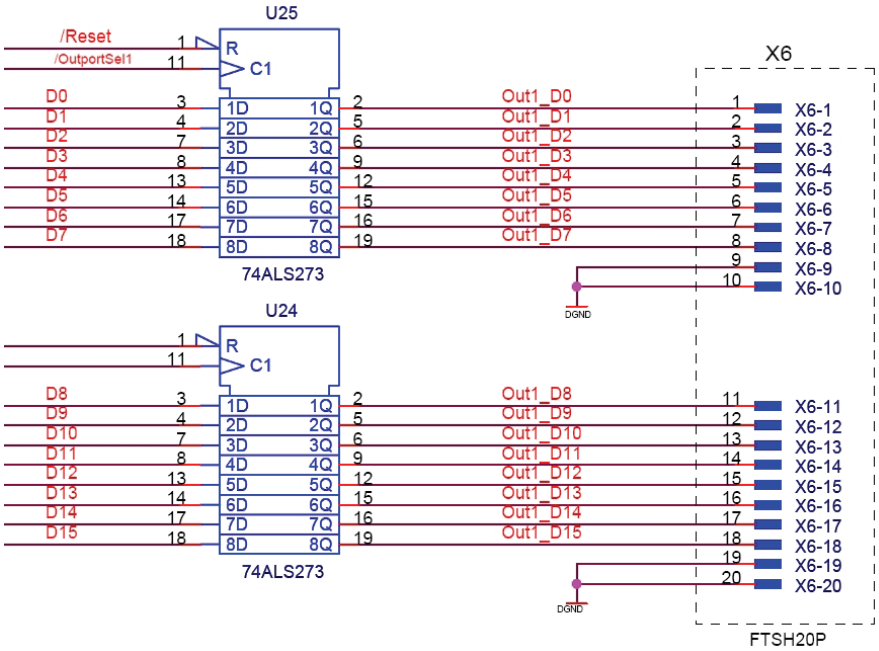


Figure 10: Digital Outputs

Refer to [1] for more details how to set the digital outputs via the firmware functions.

Bit	Signal name	X6 contact
D0	OUT1_D0	1
D1	OUT 1_D1	2
D2	OUT 1_D2	3
D3	OUT 1_D3	4
D4	OUT 1_D4	5
D5	OUT 1_D5	6
D6	OUT 1_D6	7
D7	OUT 1_D7	8
D8	OUT 1_D8	11
D9	OUT 1_D9	12
D10	OUT 1_D10	13
D11	OUT 1_D11	14
D12	OUT 1_D12	15
D13	OUT 1_D13	16
D14	OUT 1_D14	17
D15	OUT 1_D15	18
all	digital ground	9,10,19,20

Bit	Signal name	X7 contact
D16	OUT 2_D0	1
D17	OUT 2_D1	2
D18	OUT 2_D2	3
D19	OUT 2_D3	4
D20	OUT 2_D4	5
D21	OUT 2_D5	6
D22	OUT 2_D6	7
D23	OUT 2_D7	8
D24	OUT 2_D8	11
D25	OUT 2_D9	12
D26	OUT 2_D10	13
D27	OUT 2_D11	14
D28	OUT 2_D12	15
D29	OUT 2_D13	16
D30	OUT 2_D14	17
D31	OUT 2_D15	18
all	digital ground	9,10,19,20

Table 9: Digital output signals

3.12 Hall sensor inputs

For 2- or 3-phase brushless motors, the MoCon provides sinusoidal commutation; initialization is achieved using hall-based sensors. The inputs for the TTL hall signals are routed via the connectors X12 and X14. For each motor up to three hall signals and a common digital +5V power supply is provided.

channel	Signal name	X12 contact
1	MC1_Hall1A	1
1	MC1_Hall1B	2
1	MC1_Hall1C	3
2	MC1_Hall2A	7
2	MC1_Hall2B	8
2	MC1_Hall2C	9
3	MC1_Hall3A	13
3	MC1_Hall3B	14
3	MC1_Hall3C	15
4	MC1_Hall4A	19
4	MC1_Hall4B	20
4	MC1_Hall4C	21
all	+5V supply	5,11,17,23
all	digital ground	9,10,19,20

Channel	Signal name	X14 contact
5	MC2_Hall1A	1
5	MC2_Hall1B	2
5	MC2_Hall1C	3
6	MC2_Hall2A	7
6	MC2_Hall2B	8
6	MC2_Hall2C	9
7	MC2_Hall3A	13
7	MC2_Hall3B	14
7	MC2_Hall3C	15
8	MC2_Hall4A	19
8	MC2_Hall4B	20
8	MC2_Hall4C	21
all	+5V supply	5,11,17,23
all	digital ground	9,10,19,20

Table 10: Hall sensor Inputs

3.13 Serial interface

For the serial interface with the signals TXD0 und RXD0 it is possible to select either a RS232 or RS422 connection. The signal adaption is done with a MAX3162 circuit and the signals for the RS232 and RS422 connection are routed to the main connector X1 and in parallel to a mini-din connector X5. Via jumper BR1 the receive signal for the interface is either routed to the RS485/RS422 or to the RS232 section. The interface uses 8 data bits, no parity, and 1 stop bit. The Baud rate of the serial interface could be configured via the firmware and allows transmission rates up to 115200 baud. Via a firmware command it is possible to enable/disable the XOn/XOff protocol for a software flow control.

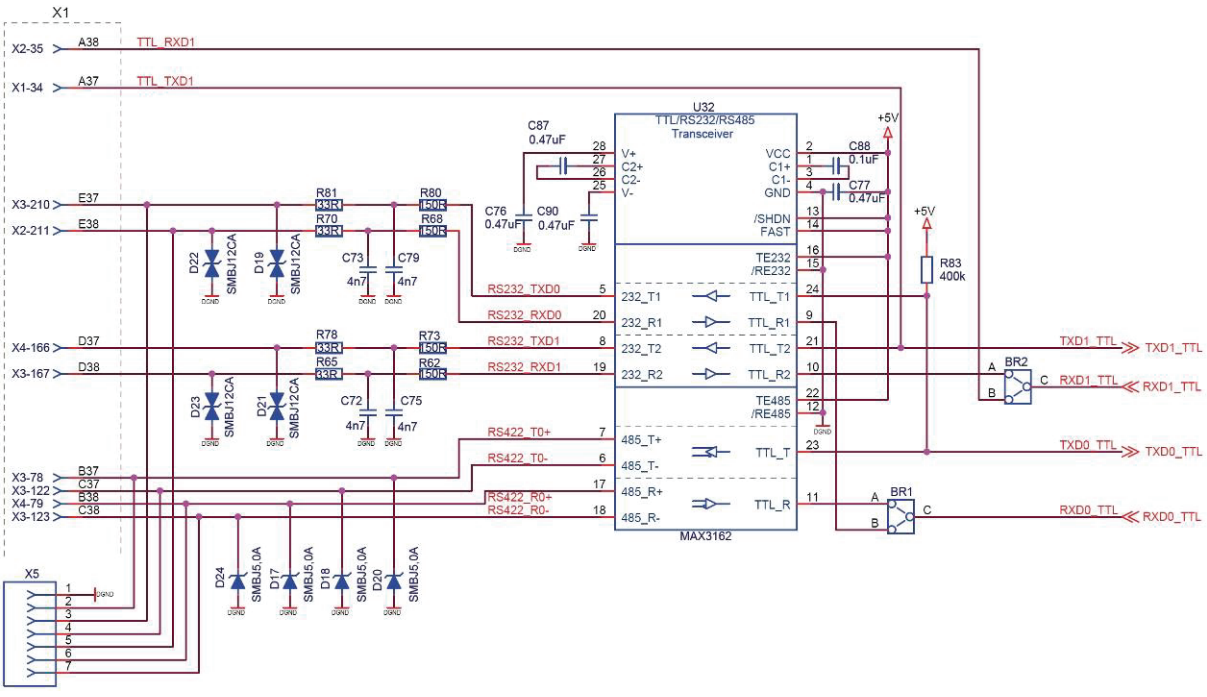


Figure 11: RS232 Interface

The second RS232 port with TXD1 and RXD1 signals is foreseen as extension and is presently unused.
Refer to chapter 4.2 for more details about the BR1 and BR2 jumper configuration

3.14 Ethernet interface

The MoCon hardware offers a standardized Ethernet communication interface. Via the jumpers BR3 and BR4 it is possible to route the differential receive signal for this interface either to the onboard RJ45 connector X5 or to an external connection via the main connector X1. The interface allows a 10Mbit connection. Refer to chapter 4.3 for more details about the BR3 and BR4 jumper configuration.

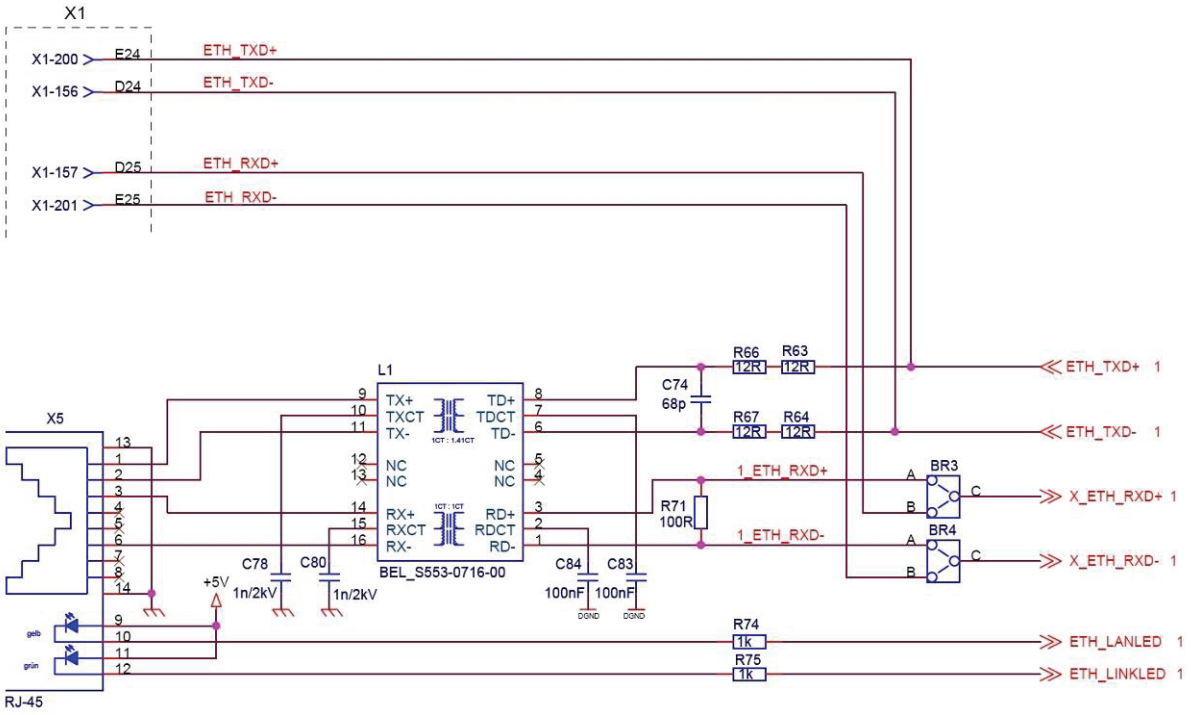


Figure 12: Ethernet Interface

3.15 CAN bus interface

For building up systems of controllers, the MoCon provides a CAN bus interface. This bus is routed parallel to the main connector X1 as well as to the two connectors X2 and X3 at the front of the board. Via the CAN bus it is possible to join further MPIA CAN hardware to a multiple controller system.
Note, that the protocol running via the CAN bus is MPIA specified and not compatible to the CANOpen standard. For jumper configuration details for the CAN bus interface, refer to chapter 4.4 of this manual.

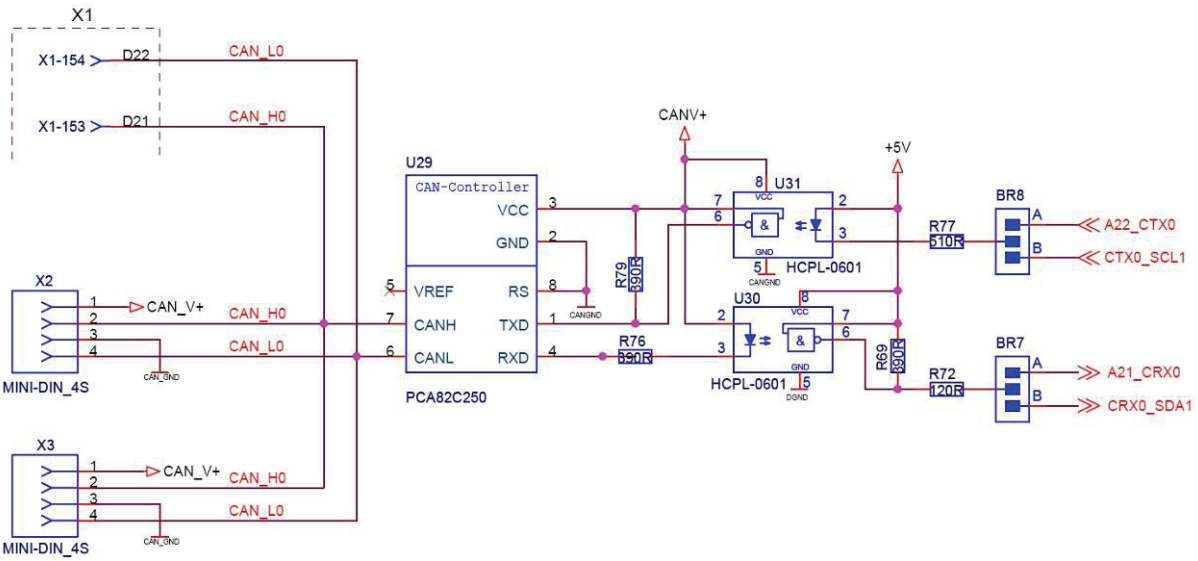


Figure 13: CAN bus interface

3.16 Reset logic and manual reset button

A Max700 circuit is used to monitor the power supply. When the +5 V_{CC} power supply on the MoCon board falls to 4.65V, the active-low RESET goes low.

During power up, a MAX700 circuit will guaranty a correct system reset and startup of the microcontroller.

In addition, a manual reset switch is available at the front end of the MoCon board.

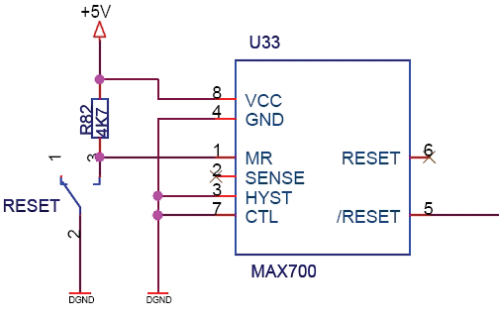


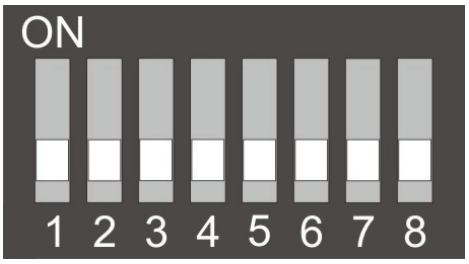
Figure 14: Reset circuit

4 Hardware setup and configuration

For setting up the hardware configuration, the MoCon hardware carries a few jumpers and switches.

4.1 Communication mode and card address

The configuration of the MoCon communication mode and card address is done via the onboard switch SW1.



1	2	3	4	5	6	7	8
A3	A2	A1	A0	Master / Slave Configuration			Default Interface Selection
Card - Address							

4.1.1 Default interface settings

SW 1 Settings	Configuration
	Default Interface settings: RS232: 9600 Baud CAN: 115000 Baud
	interface settings by firmware

Table 11: Interface settings

4.1.2 Master/Slave selection

SW 1 Settings	Configuration
	Slave
	Master Ethernet
	Master CAN
	Master Ethernet / CAN
	Master RS232
	Master Ethernet / RS232
	Master CAN / RS232
	Master Ethernet / CAN / RS232

Table 12: Master/Slave Selection

4.1.3 Card address selection

Running several controllers in a CAN bus wired structure, each card has to have a unique card number. The range of the MoCon card address is 1 to 16.

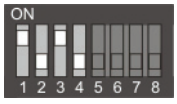
SW 1 Settings	Configuration	SW 1 Settings	Configuration
	MoCon card address = “1”		MoCon card address = “9”
	MoCon card address = “2”		MoCon card address = “10”
	MoCon card address = “3”		MoCon card address = “11”
	MoCon card address = “4”		MoCon card address = “12”
	MoCon card address = “5”		MoCon card address = “13”
	MoCon card address = “6”		MoCon card address = “14”
	MoCon card address = “7”		MoCon card address = “15”
	MoCon card address = “8”		MoCon card address = “16”

Table 13: Card address selection

4.2 Serial interface

Since the MoCon hardware has the option to select either a RS232 or RS422 connection, it is necessary to set the jumper BR1 into the correct position. BR2 is associated to the second RS232 port with TXD1 and RXD1 signals which is foreseen as extension and is presently unused.

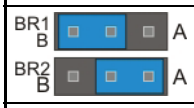
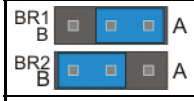
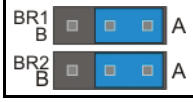
BR1 and BR2 setting	Configuration
	jumper BR1 in position “B”: RXD0 - RS232 (default position) jumper BR2 in position “B”: RXD1 - TTL (default position)
	jumper BR1 in position “B”: RXD0 - RS232 (default position) jumper BR2 in position “A”: RXD1 - RS232
	jumper BR1 in position “A”: RXD0 - RS422 jumper BR2 in position “B”: RXD1 - TTL (default position)
	jumper BR1 in position “A”: RXD0 - RS422 jumper BR2 in position “A”: RXD1 - RS232

Table 14: Configurations for serial interface

4.3 Ethernet interface

The MoCon hardware offers a standardized Ethernet interface. Via the jumpers BR3 and BR4 it is possible to route the signal for this interface either to the onboard RJ45 connector X5 or to an external connection via the main connector X1.

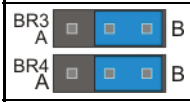
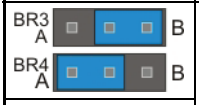
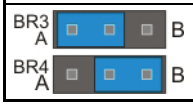
BR3 and BR4 setting	Configuration
	Jumpers in position “A”: Ethernet via onboard RJ45 connector
	Jumpers in position “B”: Ethernet via external connector X-1
	Improper settings, not permitted
	

Table 15: Ethernet Interface settings

4.4 CAN bus interface

Via two zero ohm resistors it is possible to switch the CAN bus power supply either to the internal +5V source (default) or to an external source. The as-delivered condition of the setting is position “B” which selects the internal CAN bus power supply. Table 16 gives an overview about the possible configuration. Any other combinations are permitted.

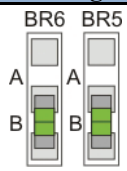
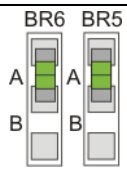
BR5 and BR6 setting	Configuration
	zero ohm resistors in position “B”: CAN bus power supply internal (default position)
	zero ohm resistors in position “A”: external CAN bus power supply

Table 16: Configurations for CAN bus power supply

By means of another two zero ohm resistors, it is possible to change the CAN bus signal routing to the microcontroller. The as-delivered condition of the setting is position “B” and should not be changed.

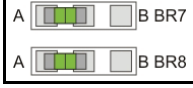
BR7 and BR8 setting	Configuration
	CAN bus routed to microcontroller SCL1/P9.3 and SDA/P9.2 (default)
	CAN bus routed to microcontroller P4.5 and P4.6 (not supported jet)

Table 17: Configuration of CAN bus signals to the microcontroller

5 Host Communication

5.1 Serial and Ethernet communication

5.1.1 Serial and Ethernet Data Protocol

A data package contains a card address (destination address), command number, message identification (only at a response massage) and the parameters.

The syntax for sending an order message to a card:
<card no.> <command no.> <module no.> [<parameter1>… <parameter n>]$_{opt}$ <\r\n>

The syntax for sending a response message from a card
(slave->host, slave->master, master->host)
<card no.> <command no.> <module no.> <message id> [<parameter1>… <parameter n>]$_{opt}$ <\r\n>

5.1.1.1 Card number <card no.>:

Order message: the card number specifies the destination of the hardware
Response message: the card number specifies the hardware which is sending the message

The master card compares the card number in the message with its own number. If the number is not corresponding with its own number, then the message will be sent to the slave card(s).

Running several controllers in a CAN bus wired structure, each hardware card has to have a unique card number. As shown in table 18, each MPIA card type has a defined number area.

MPIA Hardware	Address Range
MoCon	**1 - 16**
UNIMOD	17 - 32
RoCon	33 - 48
MCCLBLEV	49 - 64
CoCon	65 - 80

Table 18: Address range of MPIA microcontroller hardware

5.1.1.2 Command number <command no.>:

The command number indicates to which command function of the microcontroller the message goes to.

A summary of all commands and their descriptions are given in [1].

5.1.1.3 Module number <module no.>:

A hardware card can have several modules (e.g. motors) and the modules can have the same command function, therefore the module number is used to identify the message clearly.
Doesn't hardware have modules or does a command go to the basic hardware on the board, the module number is 0. If a module has a sub-module, this will be separated from the module number with a point (e.g. module 1 and sub-module 3: 1.3).
The MoCon board uses 9 module numbers, one for each motor and one for the basic functions. Table 19 shows a summary of these module numbers.

Module number	Assignment	Command example
0	MoCon basis functions	software reset, digital I/O
1	Motor 1	move N steps, get position
2	Motor 2	
3	Motor 3	
4	Motor 4	
5	Motor 5	
6	Motor 6	
7	Motor 7	
8	Motor 8	

Table 19: MoCon module numbers

5.1.1.4 Message Identification <message id> (response messages only)

The message ID gives information about the kind of response message.

message id	Assignment	Example
4	Information response message (test modus)	
3	Event message	motion complete
2	Data message	
1	Acknowledge message	
-1	Card address not exists	
-2	Command message not exists	
-3	Illegal module number	
-4	Parameter error message	wrong syntax, out of range
-5	Two slave cards with same card ID	
-6	Command buffer overrun	
-7	CAN buffer overrun	
-8	Serial buffer overrun	
-9	Dynamic memory full	
-13	Maximum of Ethernet users exceeded	
-14	Wrong password	
-15	User not logged in	
-16	User is disabled by the administrator	
-17	Port in use	
-18	Command already in use	
-20	Function error	Motor already in motion

Table 20: MoCon Message identification

5.1.1.5 Parameters <parameter1>

The number of parameters is arbitrary and depends on the number of parameters needed by the command function. The message string is limited to 75 signs including blanks.

5.1.1.6 End of command <\r\n>

The message string is completed by a carriage return and linefeed sign. Receiving this sign, the command will go to the microcontroller command queue and will be processed.

5.1.2 Serial interface settings

5.1.2.1 Software configuration of baud rate

Using the firmware command “SetSerialRate”, it is possible to adapt the baud rate setting of the serial interface. The new value is valid after a software reset. If the bit 8 of the onboard configuration switch SW1 is in OFF position, the default setting (9600 baud) is dominant.

5.1.2.2 Software configuration of serial flow control

The software flow control flag for the serial interaface is configured by using the firmware command “SetXOnXOff”.
It is suggested not to use the XOn/XOff protocol and to implement a host based flow control interpreting the command acknowledges.

5.1.3 Setup of Ethernet Interface

Table 21 gives a summary of all firmware commands which are needed to setup an Ethernet connection. Refer to [1] for a summary of all firmware functions, their command numbers and a detailed description of the command.

Firmware command	description	note
SetIPAddress	Set the IP address of the MoCon card. The new value is not valid until a software reset.	
SetGateAddress	Set the network gateway for the Ethernet connection. The changing is operative after a software reset.	
SetMaskAddress	Set the IP mask address of the MoCon card The changing is operative after a software reset.	
TCPLogin	This command is the first step to link the MoCon via an Ethernet connection	user name =”user”
TCPPassword	After the command “TCPLogin” this command is needed. If username and password are correct, the login procedure is finished	the login password is “mpia”
TCPReserveModule	More than one host can connect with the MoCon. To have exclusive access to a module, each host has to reserve a module before using it.	Up to eight simultaneously host connections are possible
TCPFreeModule	Reserved module numbers can be released with this command. After releasing, the module number is free to use for other connections	
TCPLogout	This command will close the ethernet connection.	This command sends no command acknowledge

Table 21: Firmware commands for setup of Ethernet interface

5.1.4 Host connection with HyperTerminal

5.1.4.1 RS232 host connection

After power on, the MoCon will send the version string of the firmware.
Figure 15 shows a screenshot for an example for a serial connection, the setup and the received data.

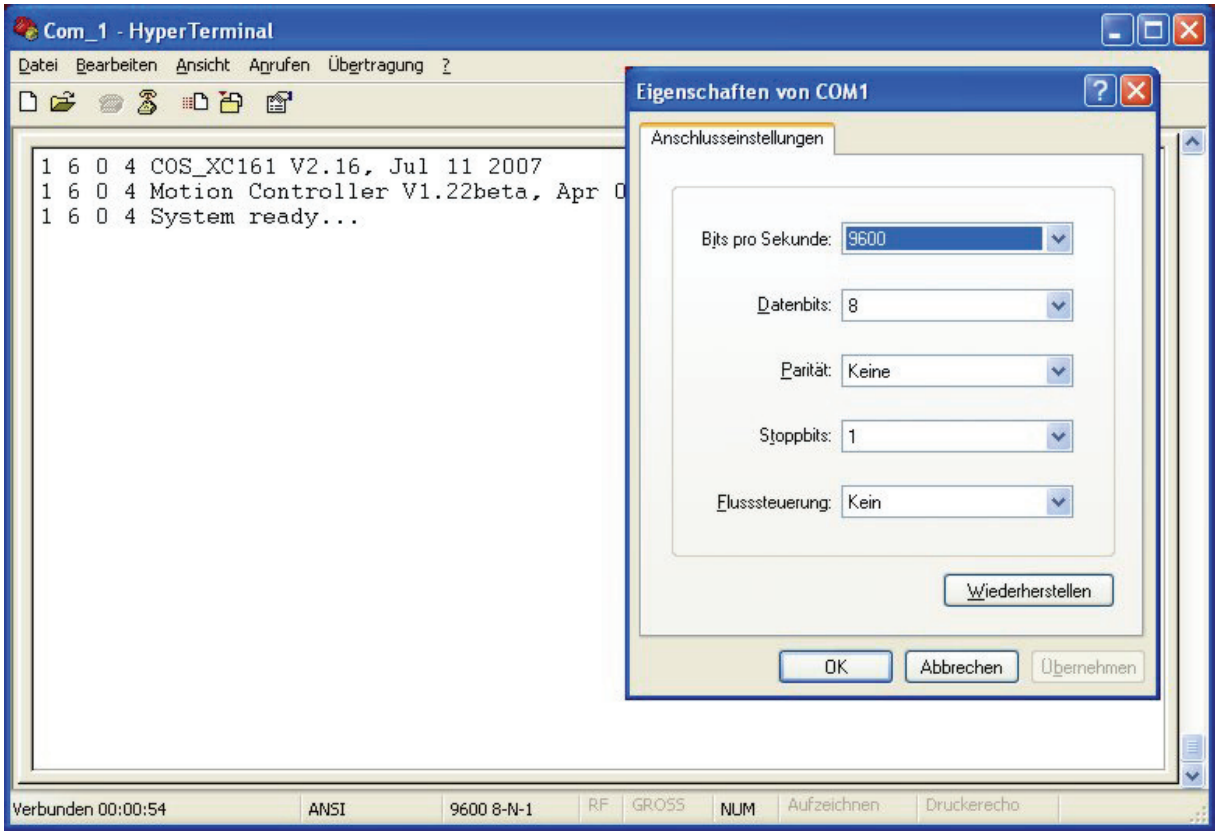


Figure 15: Serial Connection via HyperTerminal

5.1.4.1 Ethernet host connection

Connect the MoCon to the ethernet network, start a hyperterminal and connect to the MoCons IP address and Port 4000.

Figure 16 shows a screenshot with the terminal settings, the commands for login and the received data.

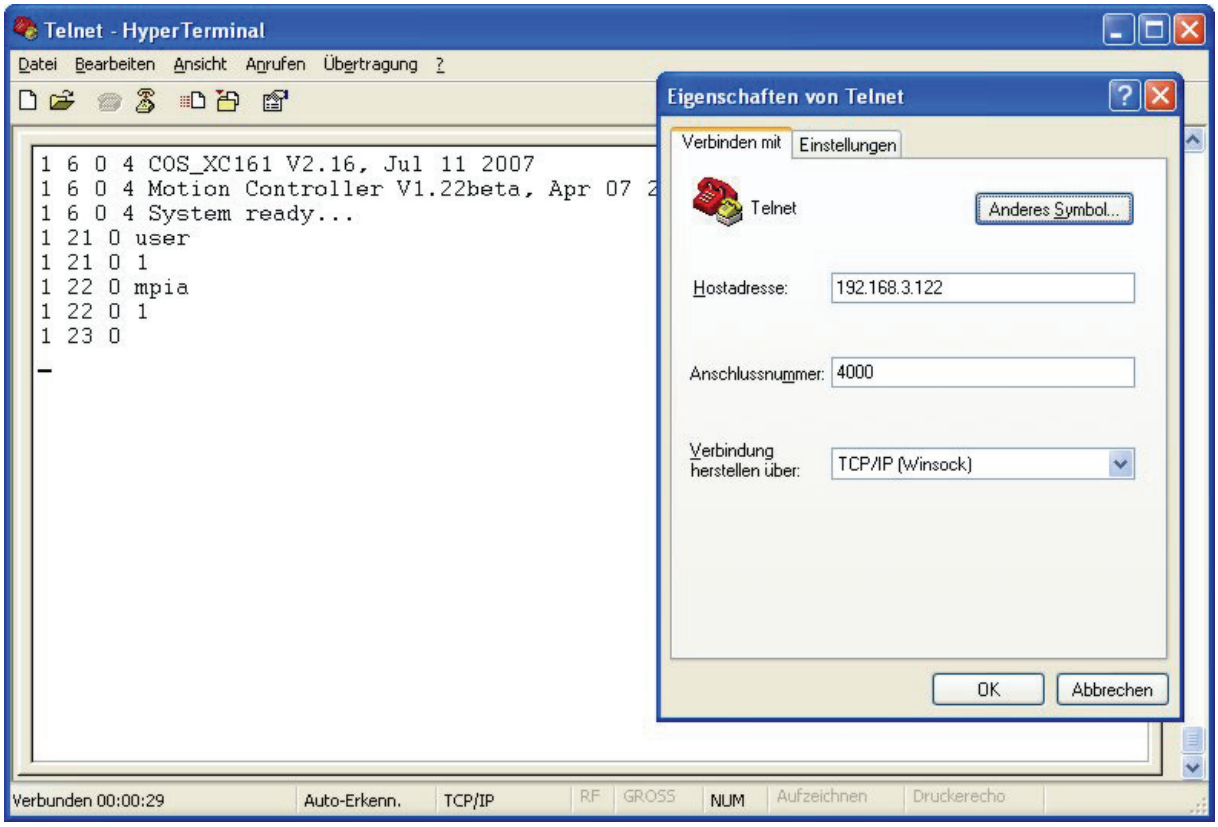


Figure 16: TCP Ethernet Connection via HyperTerminal

5.2 CAN bus interface

5.2.1 CAN bus protocol

The protocol running via the CAN bus is MPIA specified and not compatible to the CANOpen standard. A data package has a 29 bit message identifier and an 8 byte data filed. In the message identifier, the card address and command number is coded. In addition, there is the direction coding which specifies the direction of the message.

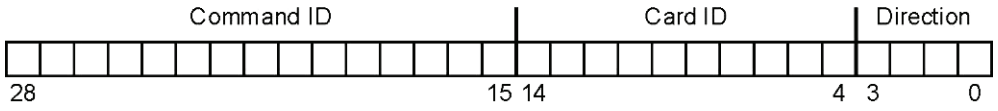


Figure 17: CAN bus message identifier

5.2.1.1 Direction (Bit 0-3)

Bit				Direction
3	2	1	0	
		0	0	Message to host
		0	1	Message to master
		1	0	Message to slave
		1	1	Message to all
0	0			Message from host
0	1			Message from master
1	0			Message from slave
1	1			not defined

Table 22: CAN bus direction bits

5.2.1.2 Card ID (Bit 4-13)

The Card Id indicates the card address. The proper range is from 1 to 1023.

5.2.1.3 Command ID (Bit 14-28)

Command Id indicates the command number for the command function. The Range is from 1 to 32768.

5.2.1.4 8-byte data field

The data field contains the module number, the message id (response message) and the parameters. The data are transmitted in the ASCII format. Is the string bigger than 8 byte, then the string is split in several packages.
The end of a command is signalled by a linefeed sign. After this sign the command goes to the command queue and will processed.

5.2.2 CAN bus settings

By means of the firmware command “SetCANRate” it is possible to set the baud rate of the CAN bus interface. The new value is not valid until a software reset. If the bit 8 of the onboard configuration switch SW1 is in OFF position, the default setting (125000 baud) is dominant.

6 Trajectory generation

6.1 Appendant Settings

Syntax <*CardNo*> ␣ 110 ␣ <*MotorNo*> ␣ <*Select*> ␣ <*Value*> ↵

Arguments	***Name***	***Description***	***Range***
	<*CardNo*>	Card number	1 ..16
	<*MotorNo*>	Motor number	
		Motor-Chip 1 installed:	1 ..4
		Motor-Chip 2 installed:	5 ..8
	<*Select*>	Input selections	1 ..7
		1⇒ Profile	0 (trapeze) 2 (s-curve)
		2⇒ Steps/Counts per revolution	1 ..8192
		3⇒ Velocity (Unit: $^{Rev}/_{min}$)	1.55 / StepsCounts ..81916250.0
		4⇒ Start velocity (Unit: $^{Rev}/_{min}$)[4)]	1.55 / StepsCounts ..81916250.0
		5⇒ Acceleration (Unit: $^{Rev}/_{min^2}$)	152600 / StepsCounts ..2147483647
		6⇒ Deceleration (Unit: $^{Rev}/_{min^2}$)	152600 / StepsCounts ..2147483647
		7⇒ Jerk (Unit: $^{Rev}/_{min^3}$)	233000 / StepsCounts ..2147483647
	<*Value*>	Parameter	see Select

[4)] This setting is only available for stepper version.

Figure 18: Basic settings

6.2 Trapezoidal point-to-point profile

For this profile, the host specifies an initial acceleration and deceleration, a velocity, and a destination position. The profile gets its name from the resulting curve (Figure 19): the axis accelerates linearly (at the programmed acceleration value) until it reaches the programmed velocity. It continues in motion at that velocity, then decelerates linearly (using the deceleration value) until it stops at the specified position.

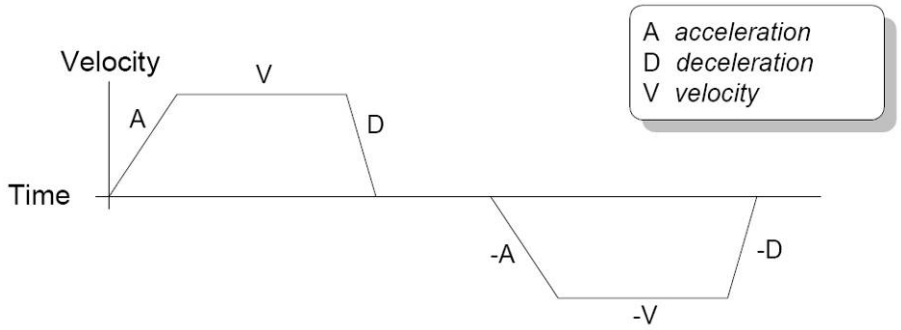


Figure 19: Simple trapezoidal point-to-point profile

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 (Figure 20).

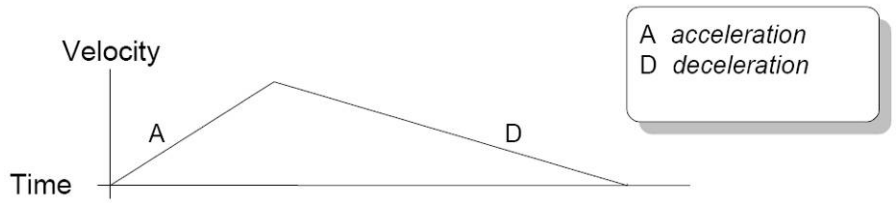


Figure 20: Simple trapezoidal point-to-point profile

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.

6.3 S-curve point-to-point profile

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

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

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

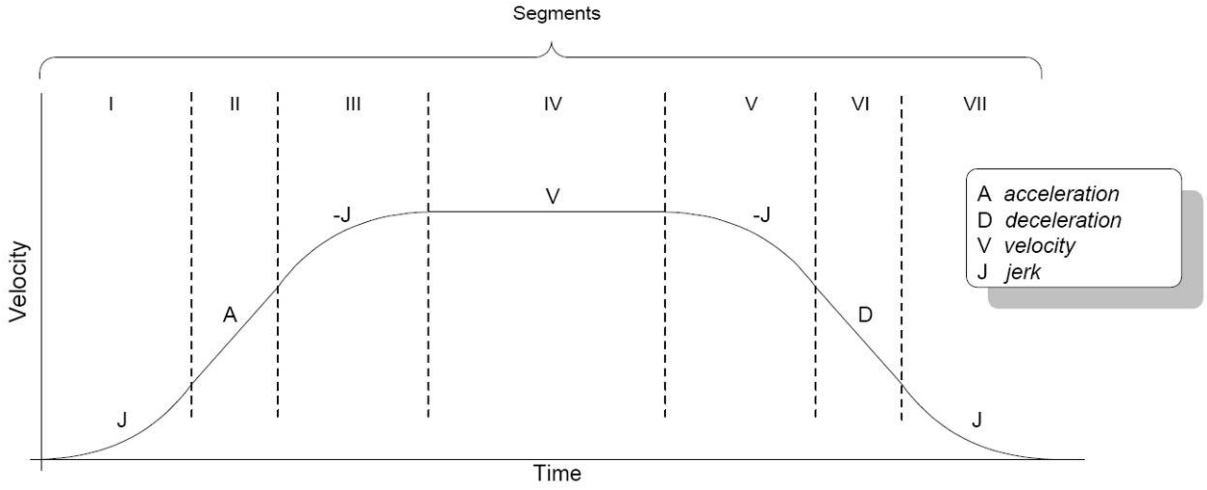


Figure 21: S-curve profile

Figure 21 shows a typical S-curve profile. In Segment I, the S-curve profile drives the axis at the specified jerk (J) until the maximum acceleration (A) is reached. The axis continues to accelerate linearly (jerk = 0) through Segment II. The profile then applies the negative value of the jerk to reduce acceleration to 0 during Segment III. The axis is now at maximum velocity (V), at which it continues through Segment IV. The profile will then decelerate in a manner similar to the acceleration stage, using the jerk value first to reach the maximum deceleration (D), and then to bring the axis to a halt at the destination.
An S-curve profile might not contain all of the segments shown in Figure 22. For example, if the maximum acceleration cannot be reached before the "halfway" point to or from the velocity, the profile would not contain a Segment II or a Segment VI. Such a profile is shown in Figure 22.

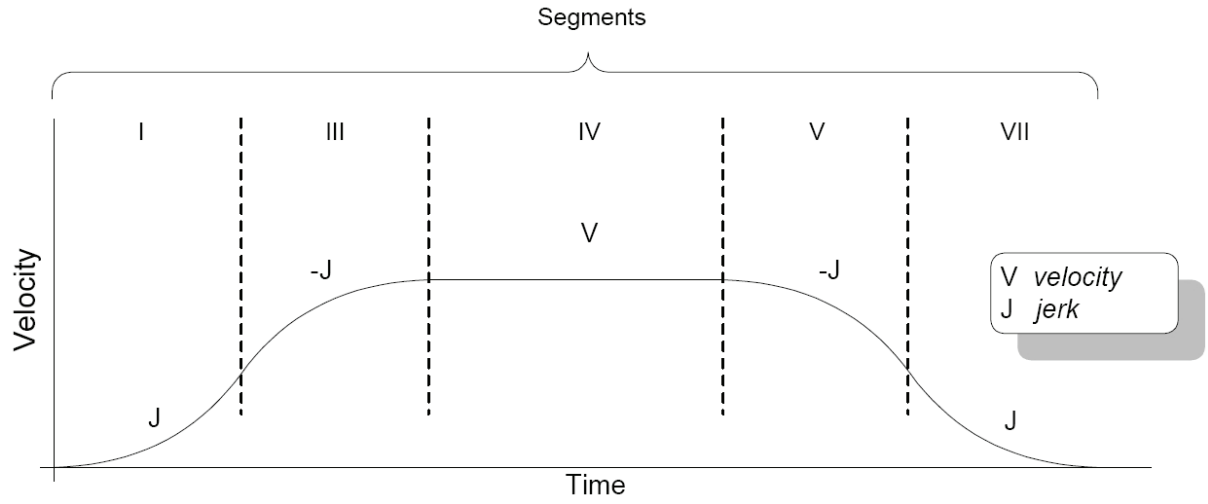


Figure 22: S-curve profile that doesn't reach maximum acceleration

Similarly, if the position is specified such that velocity is not reached, there will be no Segment IV, as shown in figure 23. There may also be no Segment II or Segment VI, depending on where the profile is "truncated".

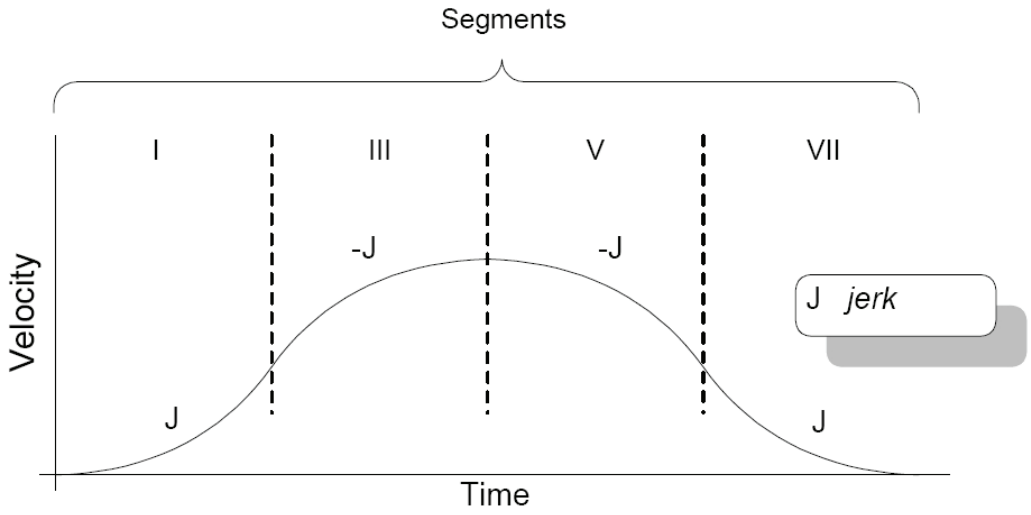


Figure 23: S-curve profile with no maximum-velocity segment

6.3 Example for Servo Settings

Figure 24 shows an example setup for a servo motor on channel 1:
Note: characters "//" and following are comment for each line and not part of the setup

```
1 110 1 1 0       // trapeze profile
1 110 1 2 2000    // counts per revolution
1 110 1 3 1000    // velocity
1 110 1 5 50000   // acceleration
1 111 1 1 4       // ref switch as a level switch (like PI-stages)
1 111 1 3 100     // velocity for home search
1 111 1 4 50      // velocity for docking to ref switch
1 111 1 5 100000  // docking distance
1 113 1 1 150     // Kp of the PID algorithm
1 113 1 2 240     // Ki of the PID algorithm
1 113 1 3 120     // Kd of the PID algorithm
1 113 1 4 0       // Kaff acceleration feed-forward
1 113 1 5 0       // Kvff velocity feed-forward
1 113 1 6 100     // Kout output scale factor
1 113 1 7 80000   // Ilim integration limit
1 113 1 8 0       // Bias dc motor offset
1 113 1 9 100     // Limit motor command limit
1 114 1 3 2000    // error limit
1 120 1           // motor initialize
```

Figure 24: Example setup for servo motor

6.4 Example for Stepper Settings

Figure 25 shows an example setup for a stepper motor on channel 5.
Note: characters "//" and following are comment for each line and not part of the setup

```
1 110 5 1 0       // trapeze profile
1 110 5 2 200     // steps per revolution
1 110 5 3 200     // velocity
1 110 5 5 5000    // acceleration
1 112 5 1 8       // microsteps
1 112 5 2 1       // microsteps adaption
1 112 5 3 1       // power off mode
1 120 5           // motor initialize
```

Figure 25: Example setup for stepper motor

7 Closed loop

7.1 Appendant settings:

Syntax	<C*ardNo*> ␣ 113 ␣ <*MotorNo*> ␣ <*Select*> ␣ <*Value*> ↲		
Arguments	***Name***	***Description***	***Range***
	<C*ardNo*>	Card number	1 ..16
	<*MotorNo*>	Motor number	
		Motor-Chip 1 installed:	1 ..4
		Motor-Chip 2 installed:	5 ..8
	<*Select*>	Input selections	1 ..9
		1⇒ K_p	0 .. 32767
		2⇒ K_i	0 .. 32767
		3⇒ K_d	0 .. 32767
		4⇒ K_{aff}	0 .. 32767
		5⇒ K_{vff}	0 .. 32767
		6⇒ K_{out} (Unit: %)	0 ..100
		7⇒ I_{lim}	0 .. 2147483647
		8⇒ Bias (Unit: %)	-100 ..100
		9⇒ Limit (Unit: %)	0 ..100
	<*Value*>	Parameter	see Select

Figure 26: Closed loop settings

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 command*, which is output as either a PWM signal to the motor amplifier, or a 16-bit input to a D/A Converter.

7.2 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:

$$\text{Output}_n = \left[K_p E_n + K_d\left(E_k - E_{(k-1)}\right) + \sum_{j=0}^{n} E_j \times Ki / 256 + K_{vff}\left(CmdVel/\ 4\right) + K_{aff}\left(CmdAccel \times 8\right) \right] \times K_{out}/65536 + \text{Bias}$$

where
- 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.

Figure 27: PID loop algorithm

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 parameter ranges, formats and interpretations are shown in the following table:

Term	Name	Representation & Range
I_{lim}	Integration Limit	unsigned 32 bits (0 to 2,147,483,647)
K_I	Integral Gain	unsigned 16 bits (0 to 32767)
K_d	Derivative Gain	unsigned 16 bits (0 to 32767)
K_P	Proportional Gain	unsigned 16 bits (0 to 32767)
K_{aff}	Acceleration feed-forward	unsigned 16 bits (0 to 32767)
K_{vff}	Velocity feed-forward	unsigned 16 bits (0 to 32767)
K_{out}	Output scale factor	unsigned 16 bits (0 to 32767)
Bias	DC motor offset	signed 16 bits (-32768 to 32767)
Limit	Motor commend limit	unsigned 16 bits (0 to 32767)

Table 23: Parameter ranges, formats and interpretations of PID loop algorithm

The structure of the digital filter is shown in Figure 28.

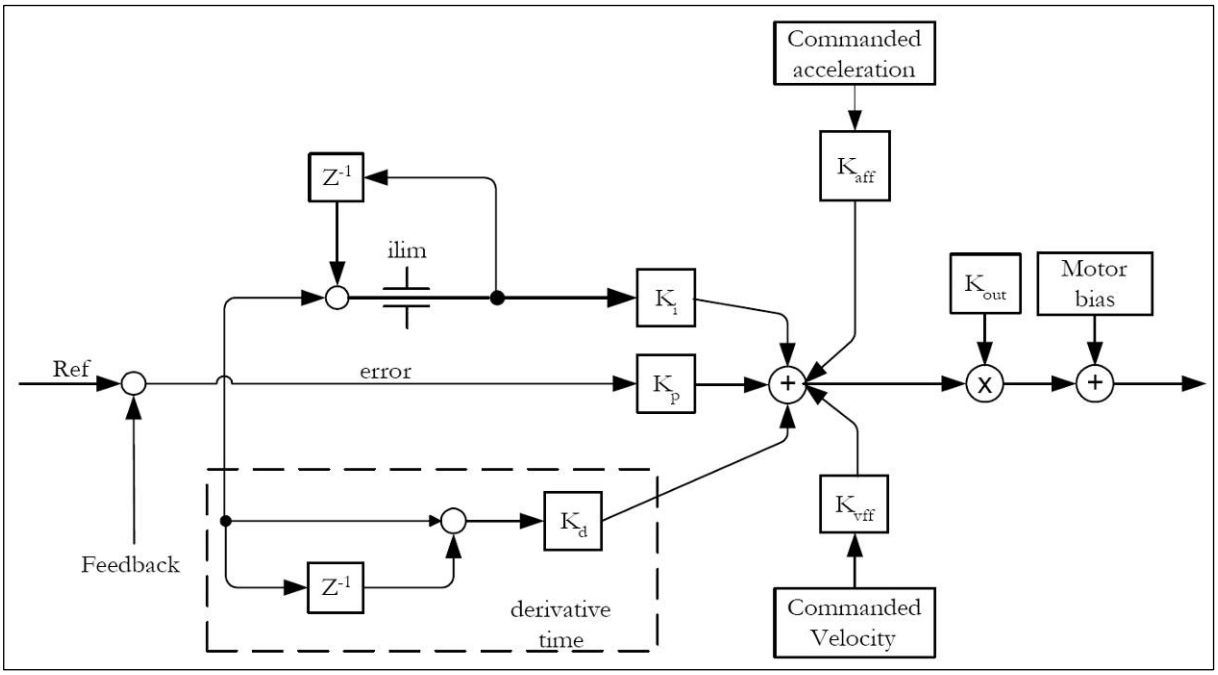


Figure 28: Digital Servo Filter

7.2.1 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.

7.2.2 Integration limit

The integration limit is used to place a boundary on the absolute value which is contributed to the PID output by the integration term. Its default value after a reset is zero, which will result in the output from the integration term of the PID filter evaluating to zero. In order to use the Ki value, the integration limit must be programmed with a value greater than zero.

7.2.3 Output scaling

The Kout parameter can be used to scale down the output of the PID filter in situations that require it. It does this by multiplying the filter result by Kout/65536. It has the effect of increasing the usable range of Kp, which is typically programmed in the 1 to 150 range when no output scaling is done.

7.2.4 Output limit

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

8 External profile

8.1 Overview

The operation of external profile mode and the setup procedure is described in this chapter. Code for the generation of profile data and the use of external profile mode is provided with C-Motion, relieving the developer from implementing any of the details and algorithms mentioned in this chapter. When an axis has its profile mode set to trapezoidal, velocity contouring or s-curve, the motion processor is executing a trajectory generation algorithm. The algorithm generates a motion profile based on a set of constraints programmed by the host. The constraints include: target position, maximum velocity, maximum acceleration, maximum deceleration and maximum jerk. Based on these constraints, the profile algorithm generates a new commanded position, velocity and acceleration once per chip cycle. These three values are then used by the servo filter or step generator to move and control the motor. This is shown in the figure below.

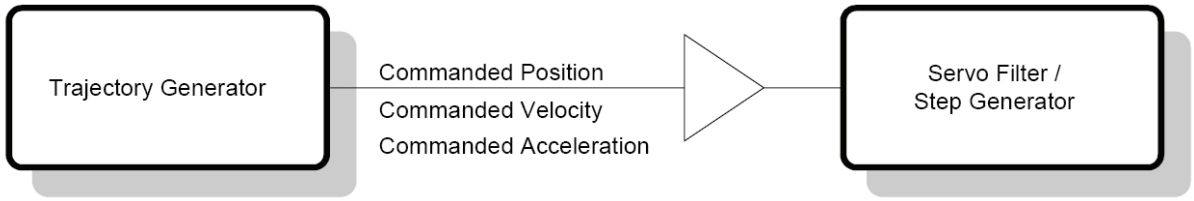


Figure 29: Internal trajectory generation

In contrast, when an axis is placed in external profile mode, the host is responsible for the calculation of the commanded position, velocity and acceleration. In effect, the host replaces the motion processor trajectory generator. It therefore becomes the responsibility of the host profile algorithm to make sure that the generated profile does not exceed any motion constraints.

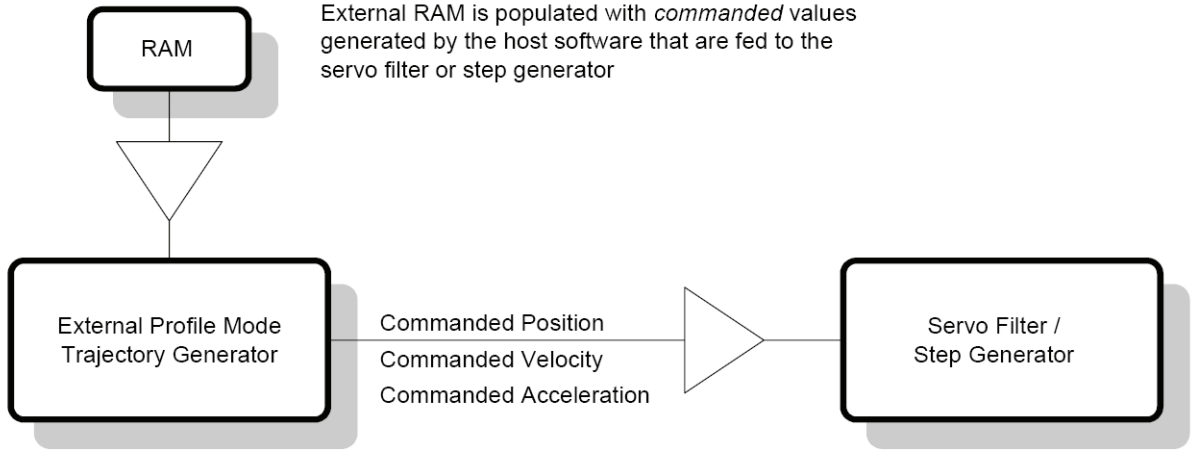


Figure 30: External trajectory generation

8.2 Commanded values

The basic operation of external profile mode is to read *commanded* values that are stored in external RAM and feed these to the servo filter or step generator. In addition, with the use of the optional time buffer (explained below) external profile mode can be used to generate intermediate *commanded* values using an iterative calculation.

A complete motion profile is made up of multiple trajectory segments, with each segment providing the parameters for a period of motion. A trajectory segment is made up of five trajectory "variables" that correspond to an internal trajectory variable. The variables and their format are shown in the table below.

Variable	Internal trajectory variable	Format	Range
Position	commanded position	32.0*	-2,147,483,648 to 2,147,483,647 counts
Velocity	commanded velocity	16.16	-32,768 to 32,767 + 65,535/65,536 counts/cycle
Acceleration	commanded acceleration	16.16	-32,768 to 32,767 + 65,535/65,536 counts/cycle^2
Jerk	commanded jerk	0.32	-2,147,483,648 *to* 2,147,483,647/4,294,967,296 counts/cycle^3
Time	segment time	32.0	0 to 2,147,483,647 cycles

Table 24: Format and range of commanded values

8.3 Buffer management

The motion processor contains instructions for setting up buffers (circular arrays) in external memory and assigning them to trajectory variables. Once setup these buffers can be used to store the trajectory data used by external profile mode. The required commands and their use are described in section 1.1. A buffer for the position trajectory variable is always required, while a buffer for the velocity, acceleration, jerk and time trajectory variables is optional. The variables that are specified and their values determine the shape of the profile that is executed by the motion processor.

An axis must have one buffer created and assigned for each trajectory variable that will be used for trajectory generation. Each variable entry within the buffer occupies 32 bits. Once instructed to do so, the motion processor gathers a set of trajectory data by reading corresponding position, velocity, acceleration, jerk and time entries from the buffers. Corresponding entries are located at the same offset within each buffer. The figure below shows a typical buffer configuration where all trajectory variables are being used. Note that each buffer must have the same length.

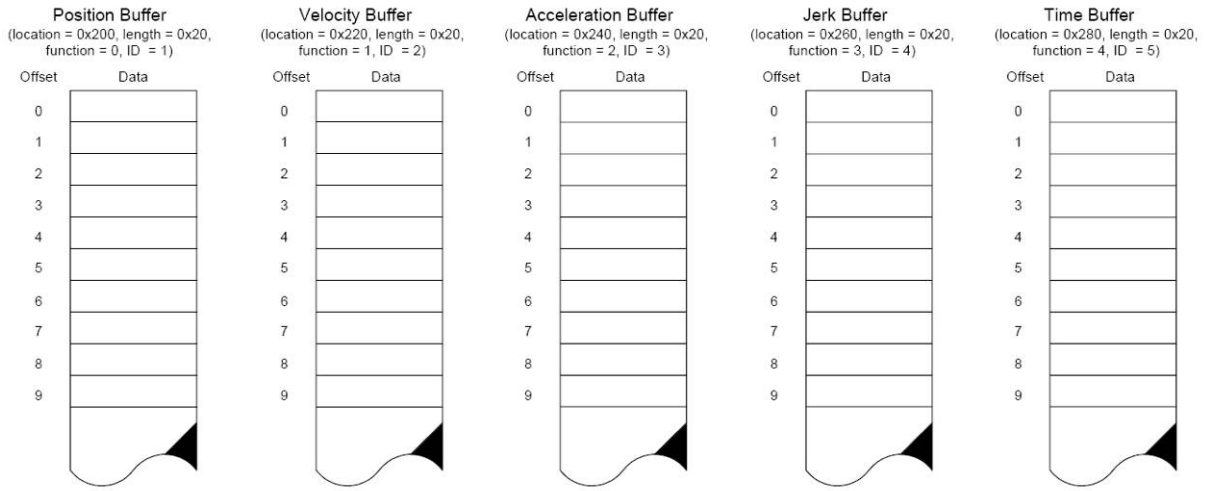


Figure 31: Example of an external memory buffer configuration

The variable buffers can be considered a table where each row corresponds to a set of values that are used for a segment of motion. When the external profile is first started, the data at offset 0 of the position, velocity, acceleration, jerk and time buffers is read. In the case where any of the velocity, acceleration or jerk buffers are not defined the corresponding internal variable is assigned a value of zero. If a time buffer is not defined, the segment time is assigned a value of one.

8.4 Pseudo code

The pseudo code below shows the high level operation of external profile mode and is included here to help explain how entries in the time buffer are utilized. This sequence is executed once every chip cycle for each axis executing an external profile, and stops when either the segment_time read from the time buffer is zero, a limit switch/motion error occurs or when a SetStopMode AbruptStop is executed.

```
if (segment_time == 0)
      // get the next set of
position,velocity,acceleration,jerk
      // and time values
      CommandedPosition = GetNextPositionBufferEntry()
      CommandedVelocity = GetNextVelocityBufferEntry()
      CommandedAcceleration =
GetNextAccelerationBufferEntry()
      CommandedJerk = GetNextJerkBufferEntry()
      segment_time = GetNextTimeBufferEntry()
      if (segment_time == 0)
            StopTrajectory()
else
      CommandedPosition += CommandedVelocity+
            CommandedAcceleration/2+CommandedJerk/6
      CommandedVelocity +=
CommandedAcceleration+CommandedJerk/2
      CommandedAcceleration += CommandedJerk

segment_time = segment_time-1
```

Figure 32: Pseudo code for high level operation of external profile mode

The sequence always starts with the motion processor reading a set of values from external memory and assigning their values to the internal trajectory variables. These values are used for one chip cycle. If the time buffer contains a value that is greater than one, at the next cycle the chip calculates a new set of trajectory parameters and decrements the segment time. The commanded position, velocity and acceleration are updated (as shown above) according to the following equations:

$$P_n = \{\ p_{n-1} + v_{n-1} + a_{n-1}/2 + j_{n-1}/6 \mid n = 2...t\ \}$$

$$V_n = \{\ v_{n-1} + a_{n-1} + j_{n-1}/2 \mid n = 2...t\ \}$$

$$A_n = \{\ a_{n-1} + j_{n-1} \mid n = 2...t\ \}$$

where:

t is the total segment time.

P, p is the commanded position

V, v is the commanded velocity

A, a is the commanded acceleration

j is the commanded jerk

Figure 33: Equation

8.5 Demonstration of buffer values

The following simple example demonstrates the result of the calculations performed when the time buffer contains a non-zero value.

	Actual Chip Time	Commanded Position	Commanded Velocity	Segment Time	Resultant Motion
Initial set of values read from external memory	11003	0	10000h	10	constant velocity
Chip performs calculations to generate intermediate values until the segment time equals 0	11004	1	10000h	9	
	11005	2	10000h	8	
	11006	3	10000h	7	
	11007	4	10000h	6	
	11008	5	10000h	5	
	11009	6	10000h	4	
	11010	7	10000h	3	
	11011	8	10000h	2	
	11012	9	10000h	1	
Next set of values read from external memory	11013	10	0	15	stationary

Table 25: Demonstration of buffer values

The value 10000h (hex) represents a value of 1.0 in the 16.16 fixed-point format. The motion processor has calculated the values shown in gray according to the equations shown above.
For the case where the time buffer contains a value of one (or if a time buffer is not created) the motion processor will only use the current set of variables for one cycle. It will not perform any calculations and will read a new set of variables during the next cycle. In this way, data generated by the host has complete control over the generated profile because it provides new commanded values for every cycle of the chip. This requires a high bandwidth for populating external memory and therefore is only recommended in designs that utilize dual port RAM.
In external profile mode all buffers are read in a circular fashion such that when the final set of data is read from the buffer (defined by the buffer length), the read index pointer wraps to offset zero.

8.6 Fixed-point encoding of trajectory parameters

The motion processors send and receive trajectory parameters using a fixed-point representation. In other words, a fixed number of bits are used to represent the integer portion of a real number, and a fixed number of bits are used to represent the fractional component of a real number. The chip uses three formats.

Format	Word size	Range	Description
32.0	32 bits	- 2,147,483,648 to +2,147,483,647	Unity scaling. This format uses an integer only representation of the number. The desired value is sent to the chip with no scaling.
16.16	32 bits	-32,768 to 32,767 + 65,535/65,536	Uses $1/2^{16}$ scaling. The chipset expects a 32 bit number which has been scaled by a factor of 65,536. For example to specify a velocity of 2.75, 2.75 is multiplied by 65,536 and the result is sent to the chip as a 32 bit integer (180,224 decimal or 2c000 hex.).
0.32	32 bits	- 2,147,483,648/4,294,967,296 to +2,147,483,647/4,294,967,296	Uses $1/2^{32}$ scaling. The chip expects a 32 bit number which has been scaled by a factor of 4,294,967,296 (2^{32}). For example to specify a value of .0075, .0075 is multiplied by 4,294,967,296 and the result is sent to the chip as a 32 bit integer (32,212,256 decimal or 1eb8520 hex).

Table 26: Trajectory parameters

8.7 Example for external profile

Figure 24 shows an example setup for an external profile generation.
Note: characters "//" and following are comment for each line and not part of the setup

```
1 226 1                          // clear buffer
1 220 1 20                       // reserve buffer
1 221 1 0 162 13288 0 1          // parameters...
1 221 1 1 162 13288 33 1
1 221 1 2 162 13288 66 1
1 221 1 3 162 13288 99 1
1 221 1 4 162 13288 132 1
1 221 1 5 162 13288 165 1
1 221 1 6 162 13288 198 1
1 221 1 7 162 13690 231 1
1 221 1 8 162 13288 265 1
1 221 1 9 162 13288 298 1
1 221 1 10 162 13288 331 1
1 221 1 11 162 13288 364 1
1 221 1 12 162 13288 397 1
1 221 1 13 162 13288 430 1
1 221 1 14 162 13288 463 1
1 221 1 15 162 13288 496 1
1 221 1 16 162 13288 529 1
1 221 1 17 162 13288 562 1
1 221 1 18 162 13416 595 1
1 221 1 19 0 0 628 1
```

Figure 34: Example setup for external profile generation

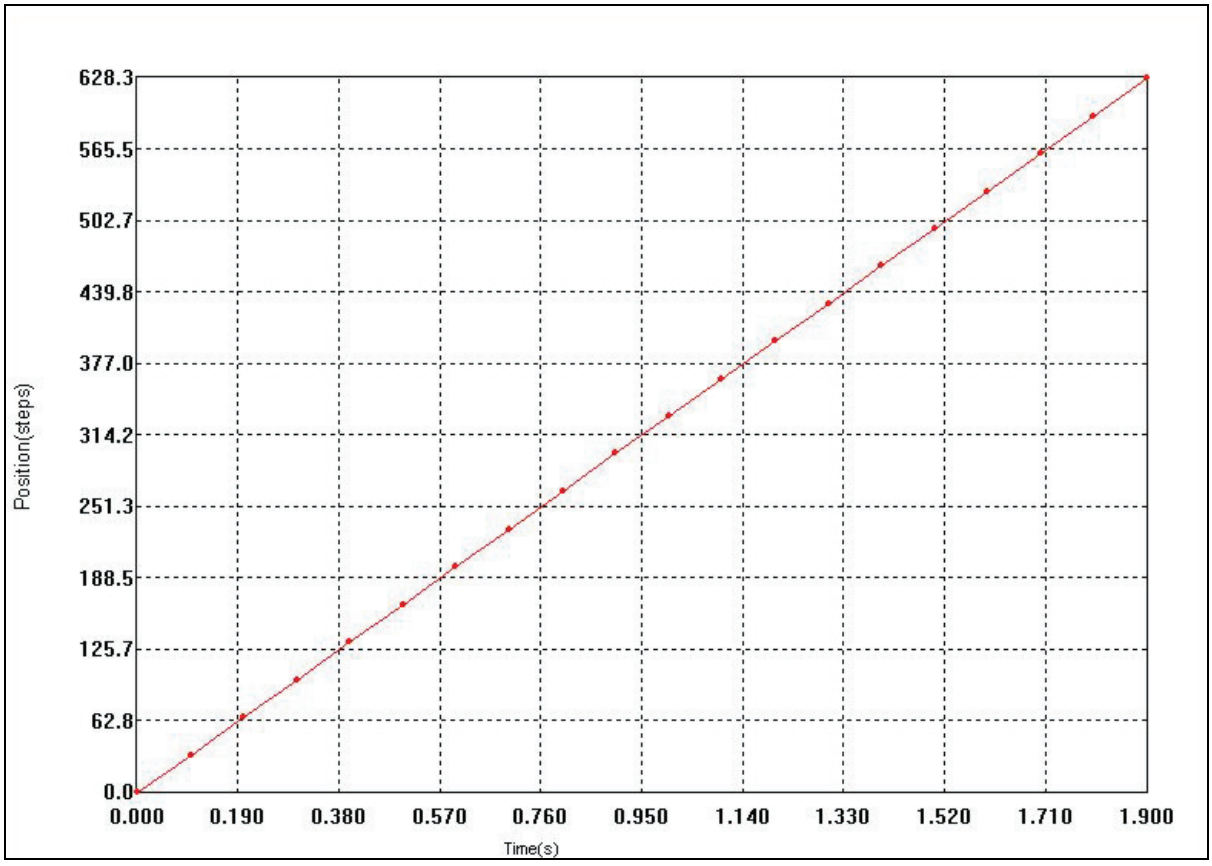


Figure 35: Example for external profile

9 Trace

9.1 Overview

Data trace is a powerful feature of the MoCon that allows various chipset parameters and registers to be continuously captured and stored to an external memory buffer. The captured data may later be downloaded by the host using standard memory buffer access commands. Data traces are useful for optimizing servo performance, verifying trajectory behavior, capturing sensor data, or to assist with any type of monitoring where a precise time-based record of the system's behavior is needed. Generally, trace data capture (by the chipset) and trace data retrieval (by the host) are executed as two separate processes. The host specifies which parameters will be captured, and how the trace will be executed. Then the chipset performs the trace, and finally the host retrieves the data after the trace is complete. It is also possible however to perform continuous data retrieval even as the chipset is continuing to collect additional trace data.
To start a trace the host must specify a number of parameters. They are listed below:

Trace variables - There are 8 separate items within the chipset that can be stored such as actual position, event status register, position error, etc. The user must select which variables, and from what axes, data will be recorded.
Trace period - The chipset can capture the value of the trace variables every single chipset cycle, every other cycle, or at any programmed frequency. The period of data collection and storage must be specified.
Trace mode - The chipset can trace in one of two modes; one-time, or rolling mode. This determines how the data is stored and whether the trace will stop automatically, or whether it must be stopped by the host.

9.2 The trace buffer

The size of the trace buffer determines the maximum number of data points that may be captured. The maximum size of the trace buffer is only limited by the amount of physical memory in the system. The MoCon memory space allows up to 2,048 megawords of RAM to be installed, all of which (with the exception of the first 1K) may be used to store trace information.

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

Syntax	<C*ardNo*> ␣ 202 ␣ 0 ␣ <*Number*> ↵		
Arguments	***Name***	***Description***	***Range***
	<C*ardNo*>	Card number	1 ..16
	<*Number*>	Number of samples	1 .. 4294967295

Figure 36: SetTraceSampleNumber command

9.3 The trace rate

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

Syntax	<*CardNo*> ␣ 201 ␣ 0 ␣ <*Cycle*> ↲		
Arguments	***Name***	***Description***	***Range***
	<*CardNo*>	Card number	1 ..16
	<*Cycle*>	Cycle time	1 ..65000 Stepper Version: 1 cycle = 614.4 µs Servo Version: 1 cycle = 614.4 µs

Figure 37: SetTraceSampleRate command

9.4 Trace variables

When traces are running, one to four chipset parameters may be stored to the trace buffer every trace period. The four *trace variable* registers are used to define which parameters are stored. Use the following commands to configure the trace variables.
The value passed and returned by the preceding two commands specifies the axis and type of data to be stored. The format of this word is as follows:

Syntax	<*CardNo*> ␣ 200 ␣ <*MotorNo*> ␣ <*Variable*> ␣ <*Type*> ↲		
Arguments	***Name***	***Description***	***Range***
	<*CardNo*>	Card number	1 ..16
	<*MotorNo*>	Motor number	
		Motor-Chip 1 installed:	1 ..4
		Motor-Chip 2 installed:	5 ..8
	<*Variable*>	Variable ID	1 ..8
	<*Type*>	Variable type	0 ..15 0 ⇒ None 1 ⇒ Commanded position 2 ⇒ Commanded velocity 3 ⇒ Commanded acceleration 4 ⇒ Actual position 5 ⇒ Actual velocity 6 ⇒ Position error 7 ⇒ PID error 8 ⇒ Event Status register 9 ⇒ Activity Status register 10 ⇒ Signal Status register 11 ⇒ Intergral 12 ⇒ Derivative 13 ⇒ Motor command 14 ⇒ Capture register 15 ⇒ Chipset time

Figure 38: SetTraceVariable command

9.5 Trace modes

As trace data is collected it is written to sequential locations in the trace buffer. When the end of the buffer is reached, the trace mechanism will behave in one of two ways depending on the mode that has been selected. If 'one-time' mode is selected then the trace mechanism will stop collecting data when the buffer is full.
If 'rolling-buffer' is selected then the trace mechanism will wrap around to the beginning of the trace buffer and continue storing data. In this mode the diagnostic trace will not end until the conditions specified in a SetTraceStop command are met.
Use the command SetTraceMode to select the trace mode. The command GetTraceMode retrieves the trace mode.

Syntax	<C*ardNo*> ␣ 203 ␣ 0 ␣ <*Mode*> ↵		
Arguments	***Name***	***Description***	***Range***
	<C*ardNo*>	Card number	1 ..16
	<*Mode*>	Sample mode	0 ..1 0 = OneTime 1 = Rolling Buffer

Figure 39: SetTraceSampleMode command

9.5 Trace Setup Example

As trace data is collected it is

```
1 200 3 1 15      // trace variable (chipset time)
1 200 3 2 4       // trace variable (actual position)
1 201 0 100       // sampling rate (100*614.4µs)
1 202 0 1000      // buffer size
1 203 0 0         // onTime
```

Figure 40: Example for trace setup

10 Connectors and terminals

10.1 Connector X1

contact no.	i shield	h	g	f	e	d	c	b	a	z shield
1		1A PWM/Pulse	1B PWM	1C PWM/ATRest	1 Sign/Dir	1 PowerOff/Brake	1 Pos Limit	1 Neg Limit	1 Ref Limit	
2		2A PWM/Pulse	2B PWM	2C PWM/ATRest	2 Sign/Dir	2 PowerOff/Brake	2 Pos Limit	2 Neg Limit	2 Ref Limit	
3		3A PWM/Pulse	3B PWM	3C PWM/ATRest	3 Sign/Dir	3 PowerOff/Brake	3 Pos Limit	3 Neg Limit	3 Ref Limit	
4		4A PWM/Pulse	4B PWM	4C PWM/ATRest	4 Sign/Dir	4 PowerOff/Brake	4 Pos Limit	4 Neg Limit	4 Ref Limit	
5		1 Axis IN	1 Axis OUT	2 Axis IN	2 Axis OUT	3 Axis IN	3 Axis OUT	4 Axis IN	4 Axis OUT	
6		MC1_D0	MC1_D1	MC1_D2	MC1_D3	MC1_D4	MC1_D5	MC1_D6	MC1_D7	
7		MC1_D8	MC1_D9	MC1_D10	MC1_D11	MC1_D12	MC1_D13	MC1_D14	MC1_D15	
8		MC1_A0	MC1_A1	MC1_DAC_CS1	MC1_DAC_CS2	MotorSel1	MotorSel2	MotorSel3	MotorSel4	
9		EPLD1_Res1	EPLD1_Res2	EPLD1_Res3	EPLD1_Res4	EPLD1_Res5	EPLD1_Res6	EPLD1_Res7	EPLD1_Res8	
10		AENC1+	AENC2+	AENC3+	AENC4+	AENC5+	AENC6+	AENC7+	AENC8+	
11		AENC1-	AENC2-	AENC3-	AENC4-	AENC5-	AENC6-	AENC7-	AENC8-	
12		M1_Enc_A+	M1_Enc_B+	M1_Enc_I+	AENCCLK	/AENCLK		/Int3	/Int4	
13		M1_Enc_A-	M1_Enc_B-	M1_Enc_I-	A15	A16	A17	A18	A19	
14		M2_Enc_A+	M2_Enc_B+	M2_Enc_I+	A10	A11	A12	A13	A14	
15		M2_Enc_A-	M2_Enc_B-	M2_Enc_I-	A5	A6	A7	A8	A9	
16		M3_Enc_A+	M3_Enc_B+	M3_Enc_I+	A0	A1	A2	A3	A4	
17		M3_Enc_A-	M3_Enc_B-	M3_Enc_I-	D11	D12	D13	D14	D15	
18		M4_Enc_A+	M4_Enc_B+	M4_Enc_I+	D6	D7	D8	D9	D10	
19		M4_Enc_A-	M4_Enc_B-	M4_Enc_I-	D1	D2	D3	D4	D5	
20		M5_Enc_A+	M5_Enc_B+	M5_Enc_I+	/CS4	/RD	/WR		D0	
21		M5_Enc_A-	M5_Enc_B-	M5_Enc_I-	CLK_OUT	CAN_H0	CAN_V+	/BOOT	I2C_SDA0	
22		M6_Enc_A+	M6_Enc_B+	M6_Enc_I+	SSC_SCLK0	CAN_L0	CAN_GND	SYNC_IN1	I2C_SCL0	
23		M6_Enc_A-	M6_Enc_B-	M6_Enc_I-	SSC_MTSR0	SSC_MRST0		SYNC_IN2	I2C_SDA2	
24		M7_Enc_A+	M7_Enc_B+	M7_Enc_I+	ETH_TXD-	ETH_TXD+	/RESET	SYNC_OUT1	I2C_SCL2	
25		M7_Enc_A-	M7_Enc_B-	M7_Enc_I-	ETH_RXD-	ETH_RXD+	/RES_IN	SYNC_OUT2		
26		+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V	
27		DGND	DGND	DGND	DGND	DGND	DGND	DGND	DGND	
28										
29										
30										
31					+2,5V					
32										
33										
34										
35										
36										
37		M8_Enc_A+	M8_Enc_B+	M8_Enc_I+	RS232_TXD0	RS232_TXD1	RS422_T0-	RS422_T0+	TTL_TXD1	
38		M8_Enc_A-	M8_Enc_B-	M8_Enc_I-	RS232_RXD0	RS232_RXD1	RS422_R0-	RS422_R0+	TTL_RXD1	
39		5A PWM/Pulse	5B PWM	5C PWM/ATRest	5 Sign/Dir	5 PowerOff/Brake	5 Pos Limit	5 Neg Limt	5 Ref Limt	
40		6A PWM/Pulse	6B PWM	6C PWM/ATRest	6 Sign/Dir	6 PowerOff/Brake	6 Pos Limit	6 Neg Limt	6 Ref Limt	
41		7A PWM/Pulse	7B PWM	7C PWM/ATRest	7 Sign/Dir	7 PowerOff/Brake	7 Pos Limit	7 Neg Limt	7 Ref Limt	
42		8A PWM/Pulse	8B PWM	8C PWM/ATRest	8 Sign/Dir	8 PowerOff/Brake	8 Pos Limit	8 Neg Limt	8 Ref Limt	
43		5 Axis IN	5 Axis OUT	6 Axis IN	6 Axis OUT	7 Axis IN	7 Axis OUT	8 Axis IN	8 Axis OUT	
44		MC2_D0	MC2_D1	MC2_D2	MC2_D3	MC2_D4	MC2_D5	MC2_D6	MC2_D7	
45		MC2_D8	MC2_D9	MC2_D10	MC2_D11	MC2_D12	MC2_D13	MC2_D14	MC2_D15	
46		MC2_A0	MC2_A1	MC2_DAC_CS1	MC2_DAC_CS2	MotorSel5	MotorSel6	MotorSel7	MotorSel8	
47		EPLD2_Res1	EPLD2_Res2	EPLD2_Res3	EPLD2_Res4	EPLD2_Res5	EPLD2_Res6	EPLD2_Res7	EPLD2_Res8	

10.2 X2 / CAN-bus

Type:	MINI-DIN 4S
contact no.	signal-name
1	CAN_V+
2	CAN_H0
3	CAN_GND
4	CAN_L0

10.3 X3 / CAN-bus

Type:	MINI-DIN 4S
contact no.	signal-name
1	CAN_V+
2	CAN_H0
3	CAN_GND
4	CAN_L0

10.4 X4 / RS232/RS422

Type:	MINI-DIN 7S
contact no.	signal-name
1	DGND
2	RS422_T0+
3	RS232_TXD0
4	RS422_T0-
5	RS232_RXD0
6	RS422_R0+
7	RS422_R0-

10.5 X5 / Ethernet

Type:	RJ45 with LEDs
contact no.	signal-name
1	ETH_TXD+
2	ETH_TXD-
3	ETH_RXD+
4	
5	
6	ETH_RXD-
7	
8	
9	+5V
10	ETH_LANLED
11	+5V
12	ETH_LINKLED
13	PE
14	PE

10.6 X6 / digital outport 1

Type:	FTSH-20P
contact no.	**signal-name**
1	OUT1_D0
2	OUT1_D1
3	OUT1_D2
4	OUT1_D3
5	OUT1_D4
6	OUT1_D5
7	OUT1_D6
8	OUT1_D7
9	DGND
10	DGND
11	OUT1_D8
12	OUT1_D9
13	OUT1_D10
14	OUT1_D11
15	OUT1_D12
16	OUT1_D13
17	OUT1_D14
18	OUT1_D15
19	DGND
20	DGND

10.7 X7 / digital outport 2

Type:	FTSH-20P
contact no.	**signal-name**
1	OUT2_D0
2	OUT2_D1
3	OUT2_D2
4	OUT2_D3
5	OUT2_D4
6	OUT2_D5
7	OUT2_D6
8	OUT2_D7
9	DGND
10	DGND
11	OUT2_D8
12	OUT2_D9
13	OUT2_D10
14	OUT2_D11
15	OUT2_D12
16	OUT2_D13
17	OUT2_D14
18	OUT2_D15
19	DGND
20	DGND

10.8 X8 / programmer port C-PLD

Type:	FTSH-10P
contact no.	signal-name
1	TCK
2	DGND
3	TDO
4	+5V
5	TMS
6	
7	
8	
9	TDI
10	DGND

10.9 X9 / digital inport 1

Type:	FTSH-20P
contact no.	signal-name
1	IN1_D0
2	IN1_D1
3	IN1_D2
4	IN1_D3
5	IN1_D4
6	IN1_D5
7	IN1_D6
8	IN1_D7
9	DGND
10	DGND
11	IN1_D8
12	IN1_D9
13	IN1_D10
14	IN1_D11
15	IN1_D12
16	IN1_D13
17	IN1_D14
18	IN1_D15
19	DGND
20	DGND

10.10 X10 / digital inport 2

Type:	FTSH-20P
contact no.	signal-name
1	IN2_D0
2	IN2_D1
3	IN2_D2
4	IN2_D3
5	IN2_D4
6	IN2_D5
7	IN2_D6
8	IN2_D7
9	DGND
10	DGND
11	IN2_D8
12	IN2_D9
13	IN2_D10
14	IN2_D11
15	IN2_D12
16	IN2_D13
17	IN2_D14
18	IN2_D15
19	DGND
20	DGND

10.11 X11 / programmer port M-PLD1

Type:	FTSH-10P
contact no.	signal-name
1	TCK
2	DGND
3	TDO
4	+5V
5	TMS
6	
7	
8	
9	TDI
10	DGND

10.12 X12 / hall sensors 1-4

Type:	FTSH-26P
contact no.	signal-name
1	MC1_Hall1A
2	MC1_Hall1B
3	MC1_Hall1C
4	
5	+5V
6	DGND
7	MC1_Hall2A
8	MC1_Hall2B
9	MC1_Hall2C
10	
11	+5V
12	DGND
13	MC1_Hall3A
14	MC1_Hall3B
15	MC1_Hall3C
16	
17	+5V
18	DGND
19	MC1_Hall4A
20	MC1_Hall4B
21	MC1_Hall4C
22	
23	+5V
24	DGND
25	
26	

10.13 X13 / programmer port M-PLD2

type:	FTSH-10P
contact no.	signal-name
1	TCK
2	DGND
3	TDO
4	+5V
5	TMS
6	
7	
8	
9	TDI
10	DGND

10.14 X14 / hall sensors 5-8

Type:	FTSH-26P
contact no.	signal-name
1	MC2_Hall1A
2	MC2_Hall1B
3	MC2_Hall1C
4	
5	+5V
6	DGND
7	MC2_Hall2A
8	MC2_Hall2B
9	MC2_Hall2C
10	
11	+5V
12	DGND
13	MC2_Hall3A
14	MC2_Hall3B
15	MC2_Hall3C
16	
17	+5V
18	DGND
19	MC2_Hall4A
20	MC2_Hall4B
21	MC2_Hall4C
22	
23	+5V
24	DGND
25	
26	

11 Troubleshooting

11.1 Quick Troubleshooting Guide

SYMTOM	CHECKS	SOLUTION
no function of MoCon	a) check level of power supply b) check MoCon fuse	a) ensure that the +5V voltage is within the specifications b) replace fuse when tripped
unintentional reset of MoCon	a) check level of +5V power supply	a) ensure that the +5V voltage is within the specifications Voltages < 4,65V will trigger a reset (refer to chapter 3.16)
no communication to MoCon via serial interface	a) check configuration of serial interface b) check communication with default values c) check software configuration of baud rate	a) refer to chapter 4.2 b) refer to chapter 11.2 c) refer to chapter 5.1.2
no communication to MoCon via Ethernet interface	a) check hardware setup of Ethernet interface b) check software setup for Ethernet interface c) check communication with default values	a) refer to chapter 4.3 b) refer to chapter 5.1.3 c) refer to chapter 11.2
no communication to slave card	a) check SW1 setting of the slave card b) check the communication chain c) check CAN bus cabling	a) refer to chapter 4.1 b) refer to chapter 11.3 c) examine for hardware failure
no communication to amplifier board	a) check amplifier power supply and fuse b) check address jumper settings on amplifier c) check amplifier communication	a) ensure that the power supply voltages are within the specifications b) ensure correct address settings c) refer to chapter 11.4

Table 27: Quick Troubleshooting Guide

11.2 MoCon test commands

For a quick communication check, use a direct serial electrical interface to get access to the motion controller. Ensure the correct settings of the communication mode and card address as described in chapter 4.1. Configure the interface setting to default values. The settings for the serial interface are: 9600 Baud, 8 data bits, 1 stop bit and no parity. The following table shows a summary of test command which may be useful for testing and debugging the system:

Test Command	Expected response	Description
1 1 0↵	1 1 0 4 COS_XC161 V2.12, Feb 21 2005 1 1 0 4 Motion Controller V1.16beta, Dec 12 2006 1 1 0 1	Version of Base - Program Version of MoCon - Firmware Command acknowledge
1 8 0↵	1 8 0 1 1 6 0 4 COS_XC161 V2.12, Feb 21 2005 1 6 0 4 Motion Controller V1.16beta, Dec 12 2006 1 6 0 4 System ready...	Command acknowledge Software reset
1 12 0↵	1 12 0 2 0 1 12 0 1	XOnXOff protocol disabled Command acknowledge
1 14 0↵	1 14 0 2 9600 1 14 0 1	Serial rate = 9600 baud Command acknowledge
1 16 0↵	1 16 0 2 125000 1 16 0 1	CAN bus rate = 125000 baud Command acknowledge
1 155 0↵	1 155 0 2 1 0 1 155 0 2 2 1 1 155 0 2 3 0 1 155 0 1	Info modus disabled Event message enabled Auto loader disabled Command acknowledge

Table 28: MoCon-1 test commands

11.3 Checking the MoCon communication chain

The CAN communication chain from the master controller MoCon towards a slave controller could be checked using the command “GetRegistration”. Table 29 shows the test command syntax and the expected response for a system with three motion controllers in a CAN bus chain.

test command	Expected respone	Description
1 7 0↵	1 7 0 2 2 1 7 0 2 3 1 7 0 1	MoCon (address 2) registration OK MoCon (address 3) registration OK Command acknowledge
1 1 0↵	1 1 0 4 COS_XC161 V2.12, Feb 21 2005 1 1 0 4 Motion Controller V1.16beta, Dec 12 2006 1 1 0 1	Version of Base - Program MoCon 1 Version of MoCon - Firmware Command acknowledge
2 1 0↵	2 1 0 4 COS_XC161 V2.12, Feb 21 2005 2 1 0 4 Motion Controller V1.16beta, Dec 12 2006 2 1 0 1	Version of Base - Program MoCon 2 Version of MoCon - Firmware Command acknowledge
3 1 0↵	3 1 0 4 COS_XC161 V2.12, Feb 21 2005 3 1 0 4 Motion Controller V1.16beta, Dec 12 2006 3 1 0 1	Version of Base - Program MoCon 3 Version of MoCon - Firmware Command acknowledge

Table 29: communication chain test commands

11.4 Checking the MoCon to amplifier communication

In combination with MPIA amplifier boards, the MoCon provides the option to read the amplifier card information. This is done by means of the command “GetAmpCardInfo”. Figure 41 shows the syntax of the command; figure 42 shows an example for MoCon1 and amplifier Board SMD8-1. The test command “1 160 1” was sent via a serial interface connection.

Syntax <*CardNo*> ␣ 160 ␣ <*MotorNo*> ↵

Arguments	***Name***	***Description***	***Range***
	<*CardNo*>	Card number	1 ..16
	<*MotorNo*>	Motor number	
		Motor-Chip 1 installed:	1 ..4
		Motor-Chip 2 installed:	5 ..8

Figure 41: Amplifier card reading command

```
1 160 1 4 --- Amplifier Card 1 Info's --
1 160 1 4 Classification:          Stepper
1 160 1 4 Identification:          0x1
1 160 1 4 Card Name:               SMD8
1 160 1 4 Version:                 2.0
1 160 1 4 Manufacturer date:       16.11.2005
1 160 1 4 Serial No.:              011
1 160 1 4 Amplifier offset:        0
1 160 1 1
```

Figure 42: Example for amplifier card info reading

12 Index

Notes