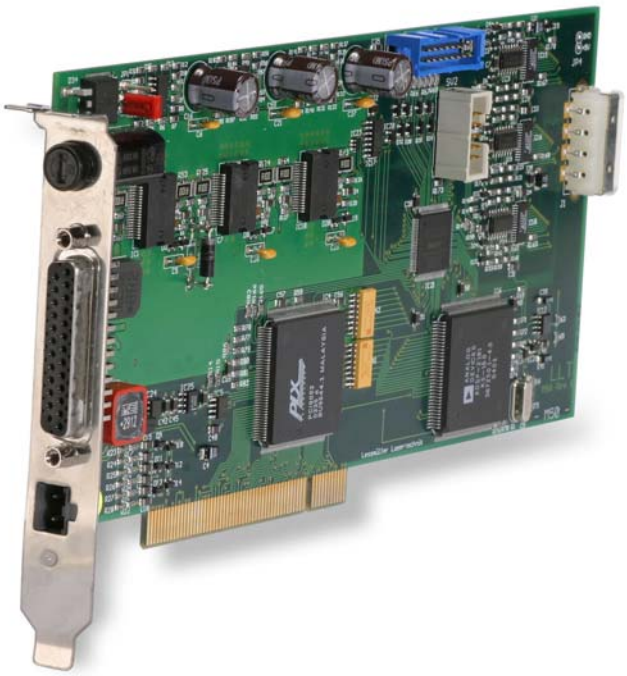
LINOS

M50.PCI Stepping Motor Card

40 2310 000



Instruction Manual

Add-in PC Card for Controlling
3 Stepping Motors in Microstep Operation

1st Issue, Juni 2005
40 2310 701a

LINOS Photonics GmbH &Co.KG
Königsallee 23
D-37081 Göttingen, Germany
Phone +49 (0)5 51 69 35 0
Fax +49 (0)5 51 69 35 166
E-mail: sales@LINOS-Photonics.DE
http://www.linos.de

1 About This Technical Literature

A number of symbols are used in this technical hardware and software description to give you quick guidance and to draw your attention to the essentials.

IMPORTANT NOTE
This symbol indicates important or additional information about the *M50.PCI stepping motor card* or literature.

CAUTION
Indicates a hazard that can damage the *M50.PCI stepping motor card* or the computer in which it is installed. Any resulting hazards for people must be avoided.

DANGER
Risk of electrocution by high electrical voltage if touched.

The information in this document is subject to change without prior notice!

2 Important Information

2.1 Information on the Use of this PC Card

This card has been designed only for controlling 2-phase stepping motors and may not be modified in any way. If any malfunction occurs, please be sure to contact our Service Department (for a list of addresses, please refer to Chapter 14 on Service). LINOS shall not be liable for any direct, indirect or consequential damage, particularly if such damage is caused by negligence, and no matter whether such damage arises in connection with the warranty, a contract, an offence or a further legal theory.

The electrical equipment described (apparatuses, systems, installation and networks) is only intended for use in industrial or scientific areas. Compliance with the legal requirements and directives concerned is essential.

If used incorrectly, this electrical equipment may generate hazardous voltage or may have “live” parts during operation. Removing the required covers or inadequate maintenance may severely endanger or impair the health of individuals or cause material damage. Therefore, the person responsible for the safety of the equipment must ensure that

- only qualified personnel are entrusted with working on or operating the equipment and machinery;
- these persons always have the appropriate instruction manual and any additional product literature as necessary for all operating steps and servicing and repair work, and that such persons are obligated to observe these instructions and information at all times;
- work on this equipment and machinery or in their vicinity is prohibited for non-qualified personnel.

Qualified personnel are persons who by virtue of their education, experience and training as well as their knowledge of the pertinent standards, requirements, occupational and work safety regulations and operating conditions have been authorized by the person responsible for the safety of the installation or system to perform the work as required and can recognized and prevent any potential hazards (see definitions for skilled labor according to VDE 105 or ICE 364).

This information is not exhaustive. If you have any questions or problems, please contact LINOS Photonics GmbH & Co. KG.

The information on processes explained in this instruction manual and circuit details apply analogously and must be reviewed before actually applying this information to your specific application.

LINOS Photonics GmbH & Co. KG shall not assume any guarantee for the suitability of the procedures and suggested circuits for specific applications.

The information in this instruction manual describes the features of the product without expressly assuring such characteristics.

We have carefully tested the equipment hardware and software as well as the product literature. However, we do not make any warranty that they are free of errors.

Our products continuously undergo further development. We reserve the right to make changes without prior notice. At the Internet address http://www.linos.com you will find information and, where available, update possibilities.

2.2 Safety Instructions

- Please follow all safety instructions including the directions for the connected equipment.
- Read this instruction manual carefully. It will enable you to best use the *M50.PCI stepping motor card* and prevent problems and damage.
- Always keep this instruction manual handy.
- When installing the *M50.PCI stepping motor card* in a PC, be sure to observe the rules and regulations in force.
- Make sure that if you connect an external power supply, the maximum voltage of 48 volts DC cannot be exceeded.
- Never expose the *M50.PCI stepping motor card* to direct sunlight, high humidity, dirt or extreme temperatures.
- The *M50.PCI stepping motor card* may only be used in dry rooms that do not have any risk posed by an explosive atmosphere.
- It is prohibited to use the *M50.PCI stepping motor card* outdoors.
- Ensure that the *M50.PCI stepping motor card* is sufficiently ventilated at all times.
- Check that the cable has the correct current-carrying capacity and that the appropriate connectors are correctly wired and attached; to avoid wire breakage or a cable break, install all cables so that they cannot cause any accidents.
- Unplug or plug in the cables only after the PC has been shut down and the power turned off.
- The *M50.PCI stepping motor card* may only be operated in a range of 0-40°C and up to 80% relative humidity.
- Do not store the *M50.PCI stepping motor card* below 0°C (32°F) or above 70°C (158°F) and only up to 80% relative humidity.
- Using the card in equipment on people, e.g. controlling a surgical robot, is strictly forbidden.
- The *M50.PCI stepping motor card* is not a toy and may not be used as such.

These instructions are a component of the *M50.PCI stepping motor card and must be kept handy. If this card is given or sold to other persons, these instructions as a component of the equipment must also be given to* these persons.

3 Contents

4 Part I: Overview

4.1 Standard Equipment Supplied

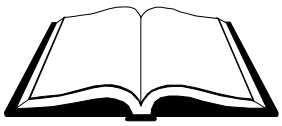
The following standard equipment is supplied when you order an *M50.PCI stepping motor card*:

M50.PCI stepping motor card

Connecting cable for external power supply

8-bit I/O extension on a slot plate

CD-ROM with software and instruction manual



This instruction manual is on a CD-ROM

4.2 Brief Technical Overview

The following points will provide you with an overview of the functions and features of the *M50.PCI stepping motor card*:

- Add-in PC card for controlling three stepping motors in microstep operation without any additional hardware
- High dynamics and speed combined with high accuracy thanks to the most advanced monolithic limit switch levels and sophisticated control software
- Future-proof and convenient with PCI bus
- 12V power supply by PC power supply unit or externally up to 48V/DC
- For bipolar 2-phase stepping motors, 2-12V in 4/5/6/8 wire technology
- Up to 2.5A phase current
- Connecting power 20W per axis
- Digital signal processor for generating acceleration ramps and pathway curves (interpolation)
- Each axis is individually programmable: motor parameters, resolution, system connection, etc.
- 6 inputs for limit switches, etc., user-configurable
- An additional 8-bit extension can be selected as input and output
- Several *M50.PCI stepping motor cards* can be operated simultaneously
- Extensive function library for 32-bit DLL
- Import files and examples in C^{++} , Delphi, LabView

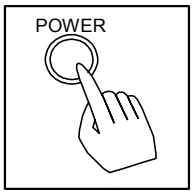
5 Part II: Installation of the *M50.PCI*

5.1 Hardware Installation: Installation in a PC

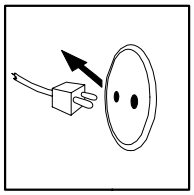
CAUTION!

DANGER!

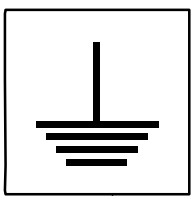
Before opening the housing, perform the following steps!



1. Turn off the computer power.

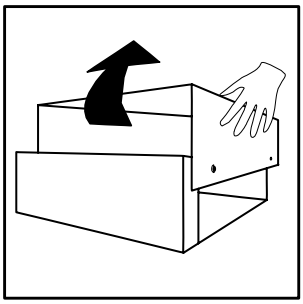


2. Disconnect the power cable from the computer.

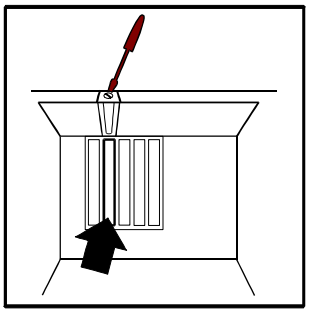


3. Make sure that you are grounded while performing all steps.

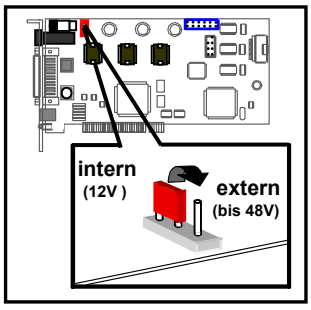
5.1.1 Installation Steps:



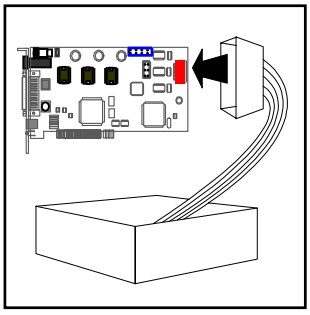
Open the housing of your computer.



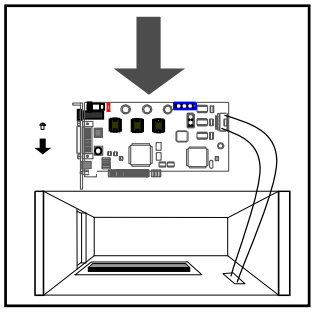
Select an available PCI port and remove the cover to expose the socket.



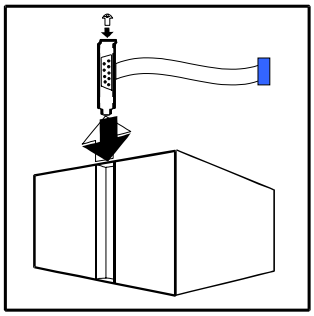
Position the red jumper (JP1) of M50.PCI according to the type of power supply you wish to use. 'Intern' stands for powering the card by your PC power supply unit and 'extern' means that you will be using an external power source.



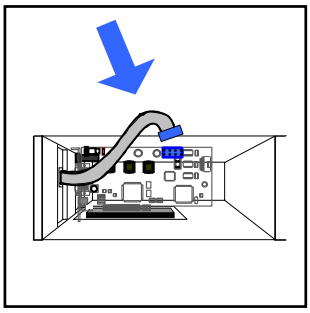
Plug an available hard disk connector into the socket on the rear side of the *M50.PCI stepping motor card*!
(Only if you are using internal power supply!)



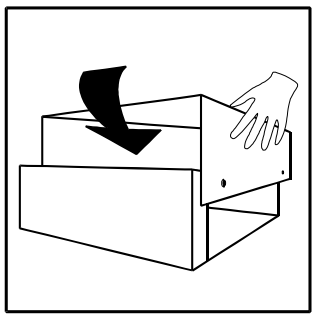
Carefully plug the *M50.PCI stepping motor card* into the socket and fasten it in place by using the screw taken from the cover plate.



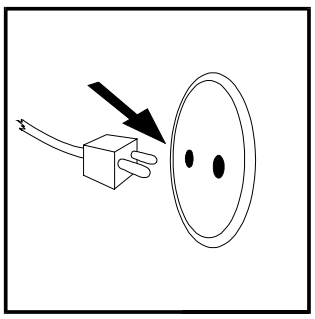
If you would like to use the additional 8 inputs/outputs, fasten the I/O slotted plate supplied in an available position on the rear panel of your PC.



Then connect the flat cable by plugging the blue connector into the *M50.PCI stepping motor card*.



Close the housing of your computer and plug the power cord into an electrical outlet.



5.2 Software Installation

The CD-ROM supplied contains the following directories:

\M50-Install	Contains the drivers required
\FirstTest	Contains a simple demo program
\M50-Delphi	Contains sources and examples for writing your own Delphi programs
\M50-VB	Contains sources and examples for writing your own VB programs
\M50-VisualC	Contains sources and examples for writing your own C++ programs
\M50-Labview	Contains sources and examples for writing your own LabView programs
\M50-Manual	Contains this instruction manual

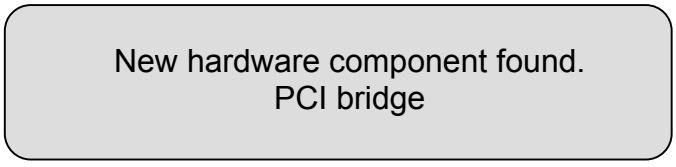
The *M50.PCI stepping motor card* is supplied with driver software for Windows® 98, Windows® 2000 und Windows® XP. The drivers and instructions for installing the *M50.PCI stepping motor card* are in the *\Install* directory on the CD-ROM supplied with this card.

5.2.1 Windows® 98

The following files are required for installation:

- *LLT-Motor.inf* Control instructions for installation
- *PCI9050.sys* Lowest operating system level for hardware
- *PLXApi.dll* Function library for PCI bridge

The first time Windows® is booted after you have installed the *M50.PCI stepping motor card* in your PC, the following hardware assistant message will appear:



Install the drivers supplied using **`LLT_Motor.inf`** on the CD-ROM by following the prompts given by the hardware assistant.

Once installation has been successfully completed, you will find the entry "LLT-STEPPER MOTOR CARD" in <*Device Manager*> of Windows® <*System Control*> under <*Other Devices Detected*>.

If you have installed several *M50.PCI stepping motor cards, you will find this entry several times as well*.

5.2.2 Windows® 2000/ Windows® XP

The following files are required for installation:

•*LLT-Motor.inf*	Control instructions for installation
•*PCI9050.sys*	Lowest operating system level for hardware
•*PLXApi.dll*	Function library for PCI bridge

The first time you boot the system after you have installed the *M50.PCI stepping motor card* in your PC, the operating system will recognize M50-PCI as new hardware and will show the following message:

> Other PCI bridge device found:
> PCI bridge

Now follow the instructions given by the installation assistant. When the assistant prompts you to install a new driver, enter the directory pathname under which the **`LT_Motor.inf`** file can be found.

Directory:\...\Install\LLT_Motor.inf

3. If you enter the correct directory pathname, a message will appear to inform you that an appropriate driver was found for the "LLT-STEPPER MOTOR CARD" device. Click on <Finish> to complete installation.

Once installation has been successfully completed, you will find the entry "LLT-STEPPER MOTOR CARD" in <*Device Manager*> of Windows® <*System Control*> under <*Other Devices Detected*>.

If you have installed several *M50.PCI stepping motor cards, you will find this entry several times as well*.

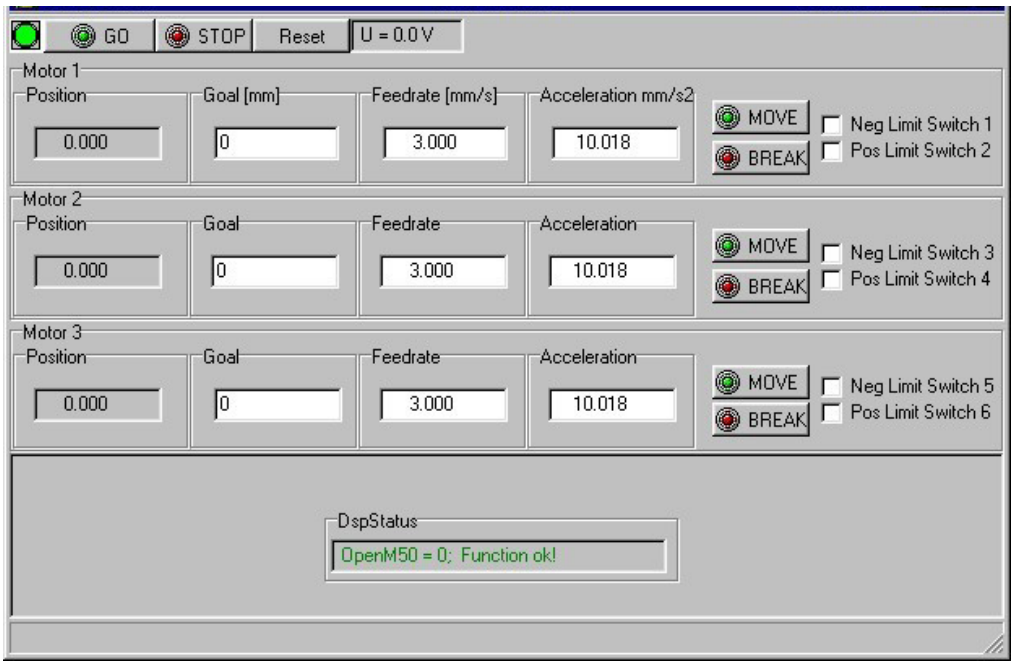
If you are running Windows® XP, a message informing you that the driver failed the "XP-Logo-Test" will appear. This message can be ignored as the test failure will not affect the reliable operation of the *M50.PCI stepping motor card* or that of the other programs.

5.2.3 Function Test after Installation

1. Create the directory ...*M50*.

2. Copy the directory *FirstTest*\ on your CD-ROM to the newly created directory.

3. You can now run the **`M50_Demo1.exe`** program in the *FirstTest* directory to test the function of the *M50.PCI stepping motor card*.
The DspStatus must accept the value by showing <*function ok*> after the DSP program has been downloaded (takes approx. 5 sec.).



The function libraries **`MX_Motor.dll`**, **`CC3260MT.dll`** and **`STLPMT45.dll`** as well as the processor program **`M50DSP.IDM`** are required every time an application program is started. Therefore, it is recommended that you copy them to the directory from which you start the application program.

5.3 Power Supply

To start up your equipment, you will need to select the power supply for the motors. The *M50.PCI stepping motor card* can be operated on 12 V from your PC power supply by plugging in the hard disk connector. For motors with a nominal voltage rating higher than 8-10V, be sure to select external power supply. This also applies to positioning systems that are to be operated at a higher velocity.

To power the card by an external power supply, plug in the two-pole connecting cable supplied with the card. A power supply source with up to 48V DC maximum can be connected. The higher the power supply, the higher the attainable velocity and acceleration. The power supply source should generate an amperage of 1.5-3.0A. We recommend using the external power supply 40 3222 000 to operate the *M50.PCI stepping motor card* with LINOS x.act positioners.

You will need to change the position of the high-current jumper on the *M50.PCI stepping motor card* depending on the type of power supply you are using (see Chapter 5.1.1, Settings: Internal Supply).
Caution: The external power supply may not exceed 48V/DC!

5.4 Electrical Current for the Motors

The electrical current for the motors is set using the software installed (see **Set Maximum Current / Get Maximum Current**). The maximum current you can set for the motors is 2.5 A per phase.

To ensure accurate microstep division, the phase current selected may not be higher than the nominal current of the motor. By contrast, a current setting that is too low will not affect microstep division.

When the stepping motors are in an idle state, a much lower torque is generally needed than when they are driving the positioner. To prevent the motors from losing power or heating up unnecessarily, the software program lets you define the idle current separately for each axis (see also **Set_Power_Down**).

The following example shows how the electrical motor power and power loss are calculated:

Coil resistance	R_{Phase} = 1.6 ohms and
Nominal current	I_{Phase} = 1.5A
Nominal voltage (DC)	V_{Phase} = R_{Ph} x I_{Ph} = 2.4V

The power loss in the switching controllers of the *M50.PCI stepping motor card* plays a subordinate role when the motors are in the idle state.

5.5 Operating State LED

The rear panel of the *M50.PCI stepping motor card* has an LED to show the current operating state. The table below lists the particular states that the LED indicates:

LED	***STATUS***
Steady yellow glow	DSP program has not yet been loaded
Flashes yellow	Program loaded, card active, motor power supply too low
Flashes green	Program loaded, card active, motor power supply ok (>11V)
Flashes red	Program loaded, card active, motor power supply too high (>50V)
Steady red glow	Excess temperature at an axis end position; motors stopped
Off	Card de-activated

5.6 Limit Switches

Six inputs are available on the *M50.PCI stepping motor card*. Each of these can be used to connect a limit switch, a reference limit switch or general inputs. Reference limit switches and inputs are not independently evaluated by the axis control program of the *M50.PCI stepping motor card*.

If an input is assigned as a positive limit switch to an axis, this axis will remain stopped in the positive motion mode immediately after this input has been activated. The axis can only move in the negative direction until the input is de-activated (the position and sentence buffer of this axis remain unchanged and can be reset, if necessary, by the application program).

Variables are available in the driver software to enable the particular operating state of the limit switches to be defined as make contacts or break contacts as well as low active and high active (see Chapter 8.3 Switch Configuration).

The CMOS level applies; in other words, input voltages <2.2 volts are evaluated as a logical '0'; voltages greater than +3 volts, as a logical '1'. The inputs when inactive are at +10V via internal pullup resistors. In their activated state, a current of 2.5mA maximum flows. The inputs are short circuit proof and protected from overvoltage and accidental interchanging of poles up to 12V.

To supply the electronic limit switches, an auxiliary supply of +10V, 100mA, is available at the connector (suggested wiring: see Appendix C).

5.7 I/O Extension

The *M50.PCI stepping motor card* has an I/O interface that provides 8 ports, which can be used independently of one another as inputs or outputs. Configuration of each pair of these ports as inputs or outputs is done by software (see Chapter 8.4).

The I/O interface also has a 5Vcc output (pin1), which can handle approx. 80-100mA and has a self-resetting electronic fuse.

If the ports are defined as **outputs**, 3.5V/20-30mA will be available via 33ohms relative to the Gnd (pin9) when these ports are in their active state. Thus, LEDs/optocouplers or small relays can be directly controlled.

If the ports are defined as **inputs**, Vcc (Pin1) can be applied to the inputs via a resistance of 200-1,000 ohms in order to activate these inputs. The contacts are designed so that they cannot be damaged even if an external load resistor is not used and they are directly connected to Vcc.

5.8 Accuracy and Resolution

The absolute resolution that can be obtained with the *M50.PCI stepping motor card* depends on the stepping motor used. A mechanical error of the motor and the stages also influence this resolution.

For a motor with a 1,500mA/phase current and 200 full steps per revolution, the absolute position error is typically < ±3µm related to the transfer of motion to a linear movement by a 2mm spindle.

The theoretical resolution of the *M50.PCI stepping motor card* is approx. 1/256 of a full step. The electromagnetic resolution attainable under real conditions depends on diverse factors; these include the ratio of the nominal torque to the torque under load, the motor power supply, and the ratio of the nominal current of the motor to the maximum current of the *M50.PCI stepping motor card*.

Under favorable conditions, a resolution of 1/100 of a full step can be achieved. In the example above, this would correspond to a resolution of 0.5-1 µm.

6 Part III: M50 Software

6.1 Overview

The*Install* directory on the CD-ROM supplied contains all the setup information for installing the drivers for Win98, Windows 2000 and Windows XP.

You will find sample programs for Delphi5.0 and higher in the *\M50-Delphi* directory. DEMO1 is a simple example written to introduce you to programming the *M50.PCI stepping motor card*. DEMOTwoCards shows the programming for two *M50.PCI stepping motor cards* using a total of 6 motors to be operated by these cards.

The function libraries `MX_Motor.dll`, `CC3260MT.dll` and `STLPMT45.dll` as well as the M50.pas import file of the M50 functions belong to each program. The `M50dsp.idm` execution program for the M50 processor is loaded and started each time the program is initialized.

The import file and C++ examples for user programming in VisualC++ are in the *\M50-VisualC* subdirectory.

The import file and examples for user programming in LabView are in the *\M-50-LabViewv* subdirectory.

The import file and examples for user programming in VB are in the *\M50-VB* subdirectory.

6.2 About the 32-bit Software

6.2.1 Testing and Initial Operation of the *M50.PCI Stepping Motor Card*

The *\M50-Delphi* directory contains the compiled programs for immediate operation of the *M50.PCI stepping motor card* on 32-bit operating systems (Win98/Win2000/ff):

`M50_Demo1.exe`	Simple sample program for controlling three motors
`MX_Motor.dll`	Dynamically linkable driver library with all card functions
`CC3260MT.dll`	Dynamically integrated supplementary library
`STLPMT45.dll`	Dynamically integrated supplementary library
`M50dsp.idm`	Program for the microprocessor on the card

6.3 Programming under Delphi

The *\M50-Delphi\M50-Demo1* subdirectory comprises all units and drivers that you need for writing your own Delphi program or for extending the sample program. This program has been written using Borland Delphi 5.0.

You can copy the files of this subdirectory as well as `MX_Motor.dll`, `M50dsp.idm` in the *..\Delphi5\Projects\M50* directory to your hard disk and access these files for your own programming.

These files are as follows:

M50.pas	Import file for dll functions
M50_Demo1.dpr	Project file for opening the M50_Demo1 project
M50_Demo1.dsk	Saves your desktop for the M50_Demo1 project
Main.dfm	Delphi Form Module for the sample program
Main.pas	Main for the sample program
M50_Demo1.pas	Source code for the sample program
MX_Motor.dll	Driver library with all card functions
M50dsp.idm	Program for the microprocessor on the card

6.3.1 Programming Using VisualC

The *\M50-VisualC* subdirectory combines all units and driversram. This program was written using C++ Vers 6.0. that you need for writing your own C++ programs or for extending the sample prog

6.3.2 Programming Using LabView

The *\M50-LabView* subdirectory combines all units and drivers that you need for writing your own LabView programs or for extending the sample program. This program was written using LabView 6.0.

7 Part IV: M50 Function Library

7.1 Brief Overview

```
---------- Card initialization ----------------------------------------
function   Open_M50 (FilePath: TFilePath);
function   Close_M50;

---------- Motor and axis settings --------------------------
function   Set_Maximum_Current (Motor, Current: INTEGER);
function   Get_Maximum_Current (Motor: INTEGER; VAR Current: INTEGER);
function   Enable_Motor  (Motor: INTEGER);
function   Disable_Motor (Motor: INTEGER);
function   Set_Steps_per_Rev (Motor, Steps_per_Rev: INTEGER);
function   Get_Steps_per_Rev (Motor: INTEGER; VAR Steps_per_Rev: INTEGER);
function   Set_Rotary (Motor, Rotary: INTEGER);
function   Get_Rotary (Motor: INTEGER; VAR Rotary: INTEGER);
function   Set Gear (Motor: INTEGER; Gear: SINGLE);
function   Get_Gear (Motor: INTEGER; VAR Gear: SINGLE);
function   Set_Pitch (Motor: INTEGER; Pitch: SINGLE);
function   Get_Pitch (Motor: INTEGER; VAR Pitch: SINGLE);
function   Set_Feedrate (Motor: INTEGER; Feedrate: SINGLE);
function   Get_Feedrate (Motor: INTEGER; VAR Feedrate: SINGLE);
function   Set_Acceleration (Motor: INTEGER; Acceleration: SINGLE);
function   Get_Acceleration (Motor: INTEGER; VAR Acceleration: SINGLE);
function   Set_Power_Down (Motor, Power_Down: INTEGER);
function   Get_Power_Down (Motor: INTEGER, VAR Power_Down: INTEGER);

---------- Switch configuration ----------------------------------
function   Set_Pos_Limit_Switch (Motor, Limit_Switch: INTEGER);
function   Get_Pos_Limit_Switch (Motor: INTEGER; VAR Limit_Switch: INTEGER);
function   Set_Neg_Limit_Switch (Motor, Limit_Switch: INTEGER);
function   Get_Neg_Limit_Switch (Motor: INTEGER; VAR Limit_Switch: INTEGER);
function   Set_Pos_Limit_Polarity (Motor, Limit_Polarity: INTEGER);
function   Get_Pos_Limit_Polarity (Motor: INTEGER; VAR Limit_Polarity: INTEGER);
function   Set_Neg_Limit_Polarity (Motor, Neg_Limit_Polarity: INTEGER);
function   Get_Neg_Limit_Polarity (Motor: INTEGER; VAR Limit_Polarity: INTEGER);
function   Set_Reference_Switch (Motor, Reference_Switch: INTEGER);
function   Get_Reference_Switch (Motor: INTEGER; VAR Reference_Switch: INTEGER);
function   Set_Reference_Polarity (Motor, Reference_Polarity: INTEGER);
function   Get_Reference_Polarity (Motor: INTEGER;  VAR Refernce_Polarity: INTEGER);
```

```
---------- I/O byte ----------------------------------------------
function  Set_ Byte_Direction (Byte_No: Integer; Direction: Word);
function  Set_Byte_Output (Byte_No: Integer; Output: Word);
function  Set_Reset_Byte_Output (Byte_No: Integer; Output: Word);
function  Byte_Input (Byte_No: Integer; Var Input: Word);

---------- Stepping motor operation ------------------------
function  Set_Goal (Motor: INTEGER; Position: SINGLE);
function  Get_Goal (Motor: INTEGER; VAR Position: SINGLE);
function  Set_Position (Motor: INTEGER, VAR Position: SINGLE);
function  Get_Position (Motor: INTEGER; VAR Position: SINGLE);
function  Go;
function  Stop;
function  Move (Motor: INTEGER);
function  Brake (Motor: INTEGER);
function  Moving (Motor: INTEGER);
function  All_Arrived;
function  End_Line;
function  Positive_Limit (Motor: INTEGER);
function  Negative_Limit (Motor: INTEGER);
function  Reference (Motor: INTEGER);
function  Input (Input_No: INTEGER);

---------- Motor power supply in V (M50 only) -----
function  Get_Motor_Voltage (Motor: INTEGER; VAR Voltage: SINGLE);

---------- Heat sink temperature in °C (M40 only) ----------
function  Get_Heatsink_Temperature (Motor: INTEGER; VAR Temperature:
                                    INTEGER);

----------System settings ----------------------------
function  Save_Configuration (FilePath: TFilePath);
function  Read_Configuration (FilePath: TFilePath);
*function  Enable_Interpolation;
*function  Disable_Interpolation;
*function  Set_Interpolation_Feedrate (IntpolFeedrate: SINGLE);
function  Get_Interpolation_Feedrate (VAR IntpolFeedrate: SINGLE);
*function  Set_Interpolation_Acceleration (IntpolAcceleration: SINGLE);
function  Get_Interpolation_ Acceleration (VAR IntpolAcceleration: SINGLE);
function  Number_of_Cards (Motor: Integer, Var Slot_Number: Integer );
```

The routines identified by an * can be called only if all axes are stopped.

- Up to three stepping motors can be controlled by the *M50.PCI stepping motor card*. A maximum of five stepping motor cards can be simultaneously operated in one PC; i.e., up to 15 motors can be controlled by PC. The valid range of values of the variable *motor* is: $1 \leq motor \leq 15$. Motor numbers 1 to 3 are assigned to the *M50.PCI stepping motor card* with the lowest PCI address; motor numbers 4 to 6 to the *M50.PCI stepping motor card* with the next higher PCI address, etc.

- All functions are of the INTEGER type. If a function is properly carried out when requested, the function value is = 0. If a function cannot be performed, it take on a negative value as an *error_code*.

The following function values occur as *error_codes:*

```
function_ok                    =  0;
error_already_open             = -1;
error_not_open                 = -2;
error_no_card                  = -3;
error_invalid_motor_no         = -4;
error_opening_config_file      = -5;
error_dsp_program_not_found    = -6;
error_mx_dll_not_found         = -7;
error_no_config_file           = -8;
error_goal_not_accepted        = -9;
error_invalid_input_np         = -10;
error_no_driver                = -11;
error_driver_function_not_found = -12;
error_mx_function_not_found    = -13;
error_mx_dll_not_loaded        = -14;
error_open_card                = -15;
error_send_dsp_program         = -16;
error_function_not_supported   = -17;
error_invalid_byte_no          = -18;
error_undervoltage             = -19;
error_overvoltage              = -20;
error_bridge_error             = -21;
error_incompatible_config_file = -22;
error_no_valid_config_file     = -23;
```

When the *M50.PCI stepping motor card is initialized,* <*OPEN_M50*>, axes one to three are set to the default values:

Maximum_Motor_Current[i]	= 1400;[mA]	
Enable_Motor[i]	= true;	
Steps_per_Rev[i]	= 200;	
Rotary[i]	= 0;	
Gear[i]	= 1.0;	
Pitch[i]	= 2.0;	[mm/Rev]
Feedrate[i]	= 3.0;	[mm/s]
Acceleration[i]	= 10.0;	[mm/s^2]
Power_Down[i]	= 50;	[%]
Pos_Limit_Switch[i]	= 2*i;	
Neg_Limit_Switch[i]	= 2*i-1;	
Reference_Switch[i]	= $ffff;	
Pos_Limit_Polarity[i]	= 0;	
Neg_Limit_Polarity[i]	= 0;	
Reference_Polarity[i]	= 0;	
Interpolation Mode[1]	= 0;	
Interpolation_Feedrate	= 3.0;	[mm/s]
Interpolation_Acceleration	= 10.0;	[mm/s^2]

These defaults can be overwritten by your own user-definable motor and axis settings or can be accepted from a data file using the <*Read_Configuration*> command.

Each time you change the geometry data of an axis (*Steps_per_Rev, Pitch, Rotary* and *Gear*), you will also need to redefine the velocity and acceleration of this axis using <*Set_Feedrate*> and <*Set_Acceleration*>. In the interpolation mode, the corresponding commands are <*Set_Interpolation_Feedrate*> and <*Set_Interpolation_Acceleration*>.

Any changes to the default and configuration settings will not become effective until you write the command <*Go*> or <*MOVE*>.

8 Description of the Functions

8.1 Card Initialization

Open_M50(FilePath: TfilePath): Integer;

When initializing an application program, this function is used to read the `M50dsp.idm` file from the *FilePath* directory and to load it into the microprocessor on the *M50.PCI stepping motor card*. At the same time, the memory area needed in the PC is dynamically allocated.
If several *M50.PCI stepping motor cards* are installed in a PC, these are automatically initialized by this function request command.

This function m u s t be requested upon each initialization.

```
Example:
procedure TMainForm.FormShow(transmitter: TObject);
begin
 (* Search in current directory: *)
  If Open_M50('M50dsp.idm')=0 Then Begin
    comment:= 'DSP-program successfully loaded'
    Exit;
  end;
 (* Search by entering entire pathname: *)
 If Open_M50('...\Borland\Delphi5\Projects\M50_Demo1\M50dsp.idm')=0
Then  begin
   comment:= 'DSP-program successfully loaded'
   exit;
 end;
 comment:= 'No M50-card present or <M50dsp.idm> not found;
end;
```

Function values:

- function_OK = 0
- error_allready_open = -1
- error_not_open = -2
- error_no_card = -3
- error_dsp_program_not_found = -6
- error_no_driver = -11
- error_driver_function_not_found = -12
- error_open_card = -15
- error_send_dsp_program = -16

Close_M50: Integer;

Close_M50 switches off all motors, closes the DSP program on the *M50.PCI stepping motor card* and releases the memory area occupied in the PC when the function <*Open_M50*> was requested. This function should be requested each time an M50 user program is closed!

```
Example:
 procedure TMainForm.FormClose(Sender:TObject; var Action: TCloseAction);
  begin
    Save_configuration('Demo1.cfg');
    Close_M40;
  end;
```

Function value: function_OK = 0 • error_not_open = -2

8.2 Motor and Axis Settings

Each time the geometry data of an axis is changed (*Steps_per_Rev, Pitch, Rotary* und *Gear*), the speed and acceleration of this axis also have to be redefined using <*Set_Feedrate*> and <*Set_Acceleration*>; in the interpolation mode, the corresponding commands are <*Set_Interpolation_Feedrate*> and <*Set_Interpolation_Acceleration*>. Changes to these settings will become effective only after they are terminated in the program by <*Go*> or <*MOVE*>.

Set_Maximum_Current(Motor, Current: INTEGER): INTEGER;

This function determines the maximum phase current and thus the torque of the motor. Normally, we select the value of the motor used for a particular axis. The phase current is divided between the two coils of a phase depending on the nominal position to be reached. The range of values is 0 - 2500mA.

Setting: *current*= 1400 [mA/phase].

```
Example:
  Set_Maximum_Current(1,2500); //set maximum current of motor1 to
                               //2500mA/phase
  Set_Maximum_Current(2, 850); //set maximum current of motor2 to
                               //850mA/phase
```

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

Get_Maximum_Current(Motor: INTEGER; VAR Current: INTEGER): INTEGER;

This function is used to read out the maximum phase current set using the variable called -*current*-.

```
Example:
procedure TMainForm.CurrentEdit(Sender: TObject);
var
  Current  : Integer;
begin
  Get_Maximum_Current(1,Current);
  CurrentEdit.Text:= 'Current of Motor1 =' + IntToStr(Current) + 'mA';
end;
```

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

Enable_Motor(Motor: INTEGER): INTEGER;

This routine activates axes that have been previously de-activated by the <*Disable_Motor*>.

Setting: Enabled

```
Enable_Motor( 1 );      (* axis1 is prepared to execute     *)
                        (* following moving commands        *)
```

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Disable_Motor(Motor: INTEGER): INTEGER;

The axis with the identifier known as *Motor* is excluded from all of the following motion commands. The operating voltge remains switched on without any changes.

Setting: Enabled

```
Disable_Motor( 1 );     (* axis1 is excluded from all *)
                        (* following moving commands  *)
```

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Set_Steps_per_Rev(Motor, Steps_per_Rev: INTEGER): INTEGER;

This function defines the number of full steps (VS) per revolution of the type of motor used. Setting the optimal microstep width is internally controlled (see Chapter 5.8).

Setting: *steps_per_rev*=200 [Full Steps]

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Get_Steps_per_Rev(Motor: INTEGER; VAR steps_per_rev: INTEGER): INTEGER;

Reads out the *steps_per_rev* variable as the number of full steps per revolution of the motor axis.

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Set_Rotary(Motor, Rotary: INTEGER): INTEGER;

This command defines the "*motor"* system axis as a rotational axis (*Rotary* = 1) or as a linear axis (*Rotary* = 0). (See also information on page 21.)

Setting: *Rotary* = 0

```
Set_Rotary ( 1 , 1 );   (* The system axis of motor 1 is *)
                        (* defined as a rotational axis    *)
```

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Get_Rotary(Motor: INTEGER; Var Rotary: INTEGER): INTEGER;

The return value indicates whether the *motor* system axis is defined as a rotational axis (*Rotary* = 1) or as a linear axis (*Rotary* = 0).

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Set_Gear(Motor: INTEGER; Gear: SINGLE): INTEGER;

This command defines an increase or a reduction ratio of the *motor* system axis. (see information on page 21).

Setting: *Gear* = 1.0 (no increase/reduction ratio).

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Get_Gear(Motor: INTEGER; VAR Gear: SINGLE): INTEGER;

Returns *Gear* as an increase or a reduction ratio of the *motor* system axis.

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Set_Pitch(Motor: INTEGER; Pitch: SINGLE): INTEGER;

Set_Pitch defines the linear movement of the *motor* system axis per axis revolution. Select the pitch according to the spindle pitch. If *motor* is not a linear axis, but has been defined as a rotary axis, (see Set_Rotary), the routine is not evaluated (see also information on page 21).

Setting: *Pitch* = 2.0 [mm/revolution].

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_moto r= -4

Get_Pitch(Motor: INTEGER; VAR Pitch: SINGLE): INTEGER ;

This returns *pitch* as the measure of the linear advance per motor axis revolution (=spindle pitch) in mm/revolution.

```
Example:
procedure TMainForm.PitchEdit(Sender: TObject);
var pitch: Integer;
begin
  Set_Pitch(5,2.25);    (* Axis5 is a linear axis with a     *)
                        (* spindle-pitch of 2.25mm/rev       *)
  Get_Pitch(5,pitch);
  PitchEdit.Text:='Pitch of Axis5= '+IntToStr(pitch)+'mm/rev';
end;
```

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Set_Feedrate(Motor: INTEGER; Feedrate: SINGLE): INTEGER;

In the point-to-point mode, this defines the speed of the *Motor* system axis in mm/s (linear axis) or π/2s (rotational axis). The range of values for the feed rate is 0.001mm/s-1000mm/s (0.1 full step/s – 100,000 full steps/s.)

(Point-to-point mode: The motors are activated independently of one another instead of their speed being related to one another's speed as in the interpolation mode; see also Chpt. 2.5, <*Enable_Interpolation*>).

Setting: *Feedrate* = 3.0 [mm/s]
(corresponds to 300[full steps/s] for a 2mm pitch and 200 full steps/360º)

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Get_Feedrate(Motor: INTEGER; VAR Feedrate: SINGLE): INTEGER;

With the *feedrate* variable, this command reads out the selected point-to-point speed of the *motor* system axis in mm/s (linear axis) or in π/2s (rotational axis).

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Set_Acceleration(Motor: INTEGER; Acceleration: SINGLE): INTEGER;

In the point-to-point mode, this command defines the acceleration of each motor axis in mm/s^2 (linear axis) or in $\pi/2s^2$ (rotational axis). To ensure the computational accuracy, acceleration is internally limited to a maximum of 800,000 full $steps/s^2$. This yields a maximum acceleration of $8000mm/s^2$ for 200 full steps/revolution and a 2 mm pitch.

Setting: Acceleration = 10 [mm/s^2]

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Get_Acceleration(Motor: INTEGER; Var Acceleration: SINGLE): INTEGER;

With the *Acceleration* variable, this command reads out the axis acceleration set in the point-to-point mode. If the motor axes are defined as a linear axes, this is read out in mm/s^2 or, for rotational axes, in $\pi/2\ s^2$.

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Set_Power_Down(Motor, Power_Down: INTEGER): INTEGER;

This function defines the percentage by which the power of the motor axis is to be reduced while the motor is stopped. *Power_Down* designates the reduction in percent based on a maximum phase current, where the current between the coils is then reduced proportionally depending on the particular position an axis has reached.

When stopped, the stepping motors need only the amount of power that enables a sufficient stop torque to be generated in order to maintain the axes' position reached. In all cases in which the motor does not receive any external torque, for eaxmple, when gears or spindles are used, the stop torque can be much lower than the travel torque. To avoid unnecessary heat loss, *Power_Down* should be set to the lowest value possible.

Setting: *Power_Down* = 50[%]

```
Set_Power_Down(1,40);   //The power of motor 1 while stopped is reduced
                        //to 40% of the Maximum_Current.
Set_Power_Down(5,25);   //The power of motor 5 while stopped is reduced
                        //to 25% of Maximum_Current
```

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Important note! Be sure that there is sufficient stop torque for axes that have external torque and force.

Get_Power_Down(Motor: INTEGER; VAR Power_Down): INTEGER;

With the *Power_Down* variable, this command returns the reduced motor power as a percentage of the maximum phase current while motion is stopped.

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

8.3 Switch Configuration

Every *M50.PCI stepping motor card* has 6 inputs. These are addressed by the integer values 1 to 6. Each input of an *M50.PCI stepping motor card* can be assigned to any of the axes of this card.

For example, *switch1* and *switch2* can be assigned as limit switches to axis 1 ("*motor1"*), and *switch3* as a reference switch for this axis. *Switch4* to *switch6* are then available for assignment to the remaining axes of this *M50.PCI stepping motor card* or can be used as inputs for any other additional functions.

M50Card PinNo	M50-DLL Value:Integer	M50.PAS Const	Setting
Pin14	1	'*switch1*'	Neg Limit Switch motor1
Pin15	2	'*switch2*'	Pos Limit Switch motor1
Pin18	3	'*switch3*'	Neg Limit Switch motor2
Pin19	4	'*switch4*'	Pos Limit Switch motor2
Pin22	5	'*switch5*'	Neg Limit Switch motor3
Pin23	6	'*switch6*'	Pos Limit Switch motor3

Tab.1: M50 pin assignment (D-Sub, 25-pin) and DLL value

If several *M50.PCI stepping motor cards* are operated together, only the inputs of the first card, *switch1* to *switch6*, can be assigned to the axes of the first card *motor1*, *motor2* and *motor3*.

The inputs of the second card (*switch1* to *switch6*) are assigned to the axes of the second card defined by the identifiers *motor4*, *motor5* and *motor6*, etc.

When not assigned, the inputs are set to "high" via pull-up resistors.

If passive switches are used that are connected to GND, define "low_active" as the input polarity.

If active switches are used, select the input polarity that corresponds to the particular circuit utilized (see Set **Polarity** and Appendix C: Pin Assignment).

Set_Pos_Limit_Switch(Motor, Limit_Switch: INTEGER): INTEGER;

This command assigns the *motor* system axis to the *Limit_Switch* input as a positive limit switch. If the input is assigned as a positive limit switch to an axis, this axis, when performing positive movement, will immediately stop as soon as the assigned input is activated. Until the input is de-activated, the axis can only be moved in the negative direction of travel. The *position* of this axis and the target buffer remain unchanged. The sentence buffer can be deleted, if necessary, using the <Brake> command in order to continue the positioning movement.

Range of values for *Limit_Switch*:[1, .. ,6] (cf. Tab 1)

Settings: *Limit_Switch* = *switch2* = 2 for *motor1*
Limit_Switch = *switch4* = 4 for *motor2*
Limit_Switch = *switch6* = 6 for *motor3*

```
Set_Pos_Limit_Switch( 1 , 1 )     (* Motor1 with positive limit switch 1 *)
Set_Pos_Limit_Switch( 2 , 3 )     (* Motor2 with positive limit switch 3 *)
Set_Pos_Limit_Switch( 3 , 5 )     (* Motor3 with positive limit switch 5 *)
```

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

Get_Pos_Limit_Switch(Motor: INTEGER; VAR Limit_Switch): INTEGER;

With the *Limit_Switch* variable, this command specifies the values of the input that is assigned as a positive limit switch to the *motor* system axis. If no positive limit switch is defined, *limit_switch* receives the value=0.

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_n o= -4

Set_Neg_Limit_Switch(Motor, Limit_Switch: INTEGER): INTEGER;

This command assigns the *Limit_Switch* as a negative limit switch to the *motor* system axis (see also FUNCTION Set_Pos_Limit_Switch).

Range of values for *Limit_Switch*:[1, .. ,6].

Settings: *Limit_Switch* = *switch1* = 1 for *motor1*
Limit_Switch = *switch3* = 3 for *motor2*
Limit_Switch = *switch5* = 5 for *motor3*

```
Set_Neg_Limit_Switch( 1 , 2 )  (*Motor 1 with negative limit switch 2*)
Set_Neg_Limit_Switch( 2 , 4 )  (*Motor 2 with negative limit switch 4*)
Set_Neg_Limit_Switch( 3 , 6 )  (*Motor 3 with negative limit switch 6*)
```

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

Get_Neg_Limit_Switch(Motor: INTEGER, VAR Limit_Switch): INTEGER;

With the variable for the *Limit_Switch,* this command specifies the values of the input that is assigned as a negative limit switch to the *motor* system axis. If no negative limit switch is defined, *limit_switch* receives the value=0.

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

Set_Pos_Limit_Polarity(Motor, Limit_Polarity: INTEGER): INTEGER;

This command defines the polarity of the positive limit switch of the *motor* system axis. When not active, the inputs are set to “high” via pull-up resistors. If passive switches connected to GND are used, the polarity has to be defined as “low_active” for this reason. If active switches are used, be sure to select the input polarity corresponding to the type of circuit employed.

Limit_Polarity = *low_active* = 0 (setting)
Limit_Polarity = *high_active* = 1

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

Get_Pos_Limit_Polarity(Motor: INTEGER; VAR Limit_Polarity: INTEGER): INTEGER;

This command indicates the polarity of the positive limit switch for the *motor* system axis.

Low_Active limit switch : *Limit_Polarity* = 0
High_Active limit switch: *Limit_Polarity* = 1

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

Set_Neg_Limit_Polarity(Motor, Limit_Polarity: INTEGER): INTEGER;

This command defines the polarity of the negative limit switch for the *motor* system axis (see also FUNCTION *Set_Pos_Limit_Polarity*).

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

Get_Neg_Limit_Polarity(Motor: INTEGER; VAR Limit_Polarity: INTEGER): INTEGER;

This command indicates the polarity of the negative limit switch for the *motor* system axis (see also FUNCTION *Get_Pos_Limit_Polarity*)

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

Set_Reference_Switch(Motor, Reference_Switch: INTEGER): INTEGER;

This command assigns the reference switch variable *Refence_Switch* to the *motor* system axis. As a result, movement is not stopped when a positioning axis reaches the reference switch end point.

Range of values for *Reference_Switch*: [1, .. ,6] (cf. Tab.2)
Setting: *Reference_Switch*= 0 (no reference switch assigned)

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

Get_Reference_Switch(Motor: INTEGER; VAR Reference_Switch: INTEGER): INTEGER;

The command returns the *Reference_Switch* variable that is assigned to the *motor* system axis (see also FUNCTION *Get_Pos_Limit_Polarity*).

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

Set_Reference_Polarity(Motor, Reference_Polarity: INTEGER): INTEGER;

This command defines the polarity of the reference switch for the *motor* system axis (see also FUNCTION *Set_Pos_Limit_Polarity*).

Setting: Reference*_Polarity* = 0 (Low_Active)

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

Get_Reference_Polarity(Motor: INTEGER; VAR Limit_Polarity: INTEGER): INTEGER;

With the *Reference_Polarity* variable, this command returns the polarity of the reference switch assigned to the *motor* system axis (see also FUNCTION *Get_Pos_Limit_Polarity*).

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

8.4 I/O Byte

Set_Byte_Direction(Byte_No: INTEGER; Direction: Word): INTEGER;

In addition to 6 inputs for the limit switches (on the motor connector), the *M50.PCI stepping motor card* has an I/O extension with 8 bits, which can be connected using the additional extension cable and slot plate supplied along with the card (see Installation). If several *M50.PCI stepping motor cards* are installed in one PC, each I/O byte is selected by *Byte_No* according to the card number.

Each of the 8 bits of the I/O extension can be defined as an **input** (default setting) or an **output**. <*Set_Byte_Direction*> along with *Direction* defines which of the 8 bits is to be evaluated as an output. These have binary codes. Both the inputs and outputs are de-activated by "low".

```
Bit1   Value=1;
Bit2   Value=2;
Bit3   Value=4;
Bit4   Value=8;
Bit5   Value=16;
Bit6   Value=32;
Bit7   Value=64;
Bit8   Value=128;

Set_Byte_Direction( 1, 1+2+4+8+16+32+64+128 ); //All 8 bits are
                                               //output
```

Function value: function_OK = 0 • error_not_open = -2 • error_function_not_supported= -17
error_invalid_byte_no= -18

Set_Byte_Output(Byte_No: INTEGER; Output: Word): INTEGER;

Use <*Set_Byte_Output*> to set a single or several bits for an I/O extension. The *Byte_No* command specifies the byte number, i.e., the card number of this extension.

```
Set_Byte_Output(1, 1);          //Sets bit1 on the card to 1 "high"
Sleep(100);
Set_Byte_Output(1, 2+128);      //Additionally sets bit2 and bit8 to "high"
Sleep(100);
```

Function value: function_OK = 0 • error_not_open = -2 • error_function_not_supported= -17
error_invalid_byte_no= -18

Set_Reset_Byte_Output(Byte_No: INTEGER; Output: Word): INTEGER;

This command is used to reset the previously activated outputs. *Output* has a binary code (see also <set_byte_direction>). Use *Byte_No* to select the I/O extension byte, which is, at the same time, equal to the number of the *M50.PCI stepping motor cards*.

```
Var output  : Word;

Set_Byte_Direction( 1, 1+2+4+8+16+32+64+128);  //Turns all 8 bits into
                                               //outputs
Set_Byte_Output(1, 1);       //Sets bit1 on the card to 1 = "high"
Sleep(100);
Set_Byte_Output(1, 2);         //Sets bit2 on the card to 1 = "high"
Sleep(100);
Set_Byte_Output(1, 4);         //Sets bit3 on the card to 1 = "high"
Sleep(100);
Set_Byte_Output(1, 8);         //Sets bit4 on the card to 1 = "high"
Sleep(100);
Set_Byte_Output(1, 16);        //Sets bit5 on the card to 1 = "high"
Sleep(100);
Set_Byte_Output(1, 32);        //Sets bit6 on the card to 1 = "high"
Sleep(100);
Set_Byte_Output(1, 64);        //Sets bit7 on the card to 1 = "high"
Sleep(100);
Set_Byte_Output(1, 128);       //Sets bit8 on the card to 1 = "high"
Sleep(400);
output:= 1 + 2 + 4 +8 +16 +32 +64 + 128
Reset_Byte_Output(1,output);   //Reverses all 8 bits back to "low"
```

Function value: function_OK = 0 • error_not_open = -2 • error_function_not_supported= -17
error_invalid_byte_no= -18

Byte_Input(Byte_No: INTEGER; Var Input: Word): INTEGER;

The <Byte_Input> command is used to request the state of the inputs of the I/O extension. Only the bits defined as inputs are addressed; the ones defined as outputs are ignored. This function returns the state of the inputs using the *Input* variable. The bits have binary codes and are evaluated as active when the bits are set to high.

```
procedure TMainForm.ByteInButtonClick(Sender: TObject);
Var BitValue : Word;
Begin
  ByteInButton.Down:= true;
  Set_Byte_Direction( 1, 0+0+0+0+0+0+0+0);
  Repeat
    Byte_Input(1, BitValue);
  until BitValue < 0;}
  ByteInButton.Down:= false;
End;
```

Function value: function_OK = 0 • error_not_open = -2 • error_function_not_supported= -17
error_invalid_byte_no= -18

8.5 Operating the Stepping Motors

Set_Goal(Motor: INTEGER; Position: SINGLE): INTEGER;

Use <*Set_Goal*> to input the absolute target positions of the axis = *motor* in mm (linear axis) or in the radian measure “rad” (rotational axis) in the sentence buffer of this axis. The commands in the sentence buffer are carried out once the axis is started by <*Move*> or once the entire system is started by <*Go*>. In the process, the values stored in the sentence buffer are selected in sequence until the sentence buffer is empty.

One or several new targets (“goals”) can also be input during positioner movement. These targets are selected one after the other as soon as the previous target position has been reached, without having to input the <*Go*> or <*Move*> command again.

Each time a new target position (goal) is entered, the previously defined values for velocity and acceleration are also saved along with this new position in the sentence buffer. The movement of each axis can be completely controlled by the command sequence <*Set_Feedrate*>, <*SetAcceleration*> and <*Set_Goal*>.

Setting: *Position* = 0 [mm]

```
Example
procedure TMainForm.Timer1Timer(Sender: TObject);
var p : Single;
begin
 Set_Goal(1, 10.00);     // axis1 shall move to position 10.00mm
 Set_Goal(2, 55.00);     // axis2 shall move to position 55.00mm
 Go;                     // axis1 and axis2 start to move
 Get_Position(1,p);
 If p > 9.00 Then begin  // If axis1 reaches the position 9.00
 Set_Point_Feedrate(1,0.8);
  Set_Goal(1, 8.35);     // 8.35 is set as new goal for axis1
  Set_Goal(2, 8.35);     // 8.35 is set as new goal for axis2
  Go;                    // with 'Go' the new goals are valid, axis1
end;                     // starts moving when position=10.00 is
                         // reached, new speed=0.8 to position 8.35
                         // while axis1 moves onwards to position 55
                         // and then back to position 8.35
```

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4
error_goal_not_accepted = -9

Get_Goal(Motor: INTEGER; VAR Goal: SINGLE): INTEGER;

The *Goal* command returns the current absolute target position (goal) for the motor system axis in mm (linear axis) or in the radian measure (rotational axis). Once the target position (goal) has been reached, it will remain stored until a new target position is set.

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor= -4

Set_Position(Motor: INTEGER, VAR Position: SINGLE): INTEGER;

This command sets the current position of the motor system axis to the *position* value. This function can also be performed while the axis is still moving.

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Get_Position(Motor: INTEGER; VAR Position: SINGLE): INTEGER;

This command returns the *position* variable along with the current position of the *motor* system axis.

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor = -4

Go: INTEGER;

Use the <*Go*> command to simultaneously start all axes. This is especially important in the interpolation mode after you have entered the targets (goals) and movement parameters for several axes. The axes will remain in motion until their particular instructions in the sentence buffer have been completely performed.

You can also set <*Go*> while a motor is running, for example, in order to have the position moved to a new target position immediately after the previously set target has been reached, or to start the axes one after another (see also <*Move*> and <*Set_Goal*>).

Function value: function_OK = 0 • error_not_open = -2 • error_undervoltage = -19
error_overvoltage = -20 • error_bridge_error = -21

Stop: INTEGER;

<*Stop*> causes all axes to stop immediately at any time it is set. There is no brake ramp so steps may be lost if the axis selected happens to be above the start/stop frequency.

The values of the position counter remain saved; the sentence buffer for all axes is cleared. As a result, this function is suitable for resetting all sentence buffers of the entire system when the axes are stopped.

While the axes remain stopped, the power in the motors is reduced according to the preset power-down parameter (see <*Set_Power_Down*>).

Function value: function_OK = 0 • error_not_open = -2

Steps may be lost when an axis is above the start/stop frequency; this may result in loss of control of the motors. Stop clears the sentence buffers for all axes.

Move(Motor: INTEGER): INTEGER;

<*Move*> starts the **individual** -*Motor*- axis. The position targets defined for this axis are selected. Once started, an axis remains in motion until the instructions in its sentence buffer have been completely executed. The remaining axes remain unaffected by the <*Move*> command.

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4
error_undervoltage = -19 • error_overvoltage = -20 • error_bridge_error = -21

Brake(Motor: INTEGER): INTEGER;

You can use the <*Brake*> command to stop the -*motor*- axis at any time or at any position. Braking is carried out with a ramp that is calculated from the currently valid value of acceleration.

Function value: function_OK = 0 • error_not_open = -2 • error_invalid_motor_no = -4

All target positions (goals) in the sentence buffer for this axis are deleted.

Moving(Motor: INTEGER): INTEGER;

When an axis is stopped (idle state), the function value is '0'. As long as the 'Motor' axis is moving, the function value remains set to '1'. If an axis moves incorrectly, the *Moving* variable will take on the negative values as error codes.

Function value: false = 0 • true = 1 • error_not_open = -2 • error_invalid_motor_no = -4

All_Arrived: INTEGER;

This command checks whether **all** system axes are stopped. The function value remains '0' as long as any of the axes is still moving or if not all the instructions in the sentence buffers have been performed. *All_arrived* takes on the function value '1' if all axes are stopped and all sentence buffers are empty. The function does not evaluate whether or which of the axes have reached their target positions (goals).

Function value: false = 0 • true = 1 • error_not_open = -2

End_Line: INTEGER;

The interpolation mode also uses a sentence buffer. To enable the program of the M50 processor to differentiate among the individual interpolation sets, <End_Line> is used. This function defines the end of an interpolated movement.

```
Example:
Set_Goal(1,10);
Set_Goal(2,15);
Set_Goal(3, 1);
End_Line;
{Go}
Set_Goal(1,-10);
End_Line;
Go;
```

Function value: function_OK = 0 • error_not_open = -2

The axes move in an interpolated manner to 10/15/1 in the x/y/z coordinate system and, right afterwards, to -10/15/1). As soon as the first End_Line command has been carried out, the axes can be started by the <*Go*> command.

Positive_Limit(Motor: INTEGER): INTEGER;

Positive_Limit allows the state of the positive limit switches to be determined. Once a limit switch has reached its defined end-point position, its function value changes from 0 to 1. The range of values for MOTOR is 1 to 3 for the *M50.PCI stepping motor card*, and 1 to 6 for two *M50.PCI stepping motor cards*, etc.

```
Example:
Procedure Home;
Begin
{+++++ Configuration: +++++}
  ....
  ....
  Set_Point_Feedrate(1,2.5);      // fast speed
  Set_Pos_Limit_Switch(1,1);      // axis1 switch1 for positive limit
  Set_Neg_Limit_Switch(1,2);      // axis1 switch2 for negative limit
  Set_Reference_Switch(1,3);      // axis1 switch3 for reference

  {+++++ Move to negative limit switch: +++++}
  Set_Goal(1,-500);               // axis1 goal in negative direction
  Go;                             // Start moving

  {+++++ Fault Condition: +++++}
  If (Negative_Limit(1)=1)and(Reference_Limit(1)=1) Then Begin
  message:='FAULT: NEGATIVE LIMIT AND REFERENCE_LIMIT NOT SEPERATED';
   Stop;
   EXIT;
  End;

  {+++++ Move to Reference Switch: +++++}
  If (Negative_Limit(1)=1)and(Reference_Limit(1)=0)Then Begin
    message:='NEGATIVE LIMIT REACHED, NOW SLOWLY TO REFEENCE...';
    Set_Point_Feedrate(1,0.25);  // slow speed
    Set_Goal(1,+5000);           // goal axis1 in positive direction
    Go;                          // Start moving
  End;

  {+++++ Finish: +++++}
  If Reference_Limit(1)=1 Then Begin
    Stop;
    Reset_Position(1);
    message:=' REACHED REFERENCE = AXIS1-HOMEPOSITION';
  End;
End;
```

Function value: true=1 • false = 0 • error_not_open = -2 • error_invalid_motor = -4

Negative_Limit(Motor: INTEGER): INTEGER;

This command checks whether the end-point position of the negative limit switch has been reached. The return value = 1 if the end-point position has been reached; otherwise, it is = 0.

The range of values is 1 to 3 for one card, and 1 to 6 for two cards, etc.

Function value: true = 1 • false = 0 • error_not_open = -2 • error_invalid_motor = -4

Reference(Motor: INTEGER) : INTEGER;

This command checks whether the end-point position of the reference switch has been reached. The return value = 1 if the end-point position has been reached; otherwise, it is = 0.

The range of values is 1 to 3 for one card, and 1 to 6 for two cards, etc.

Function value: true = 1 • false = 0 • error_not_open = -2 • error_invalid_motor = -4

Input(Input_No: INTEGER): INTEGER;

Available inputs of the M50-PCI, which are not assigned to limit or reference switches, can be used for any other purposes. The <*Input*> function evaluates all inputs. The return value is = 1 if the input was activated; otherwise, it is = 0.

The range of values for -*Input_No*- is 1 to 6 for one card, and 1 to 12 for two cards, etc.

Function value: true = 1 • false = 0 • error_not_open = -2 • error_invalid_input_no = -10

Get_Motor_Voltage (Motor: INTEGER; VAR Voltage: SINGLE): INTEGER;

The Get_Motor_Voltage function measures the voltage of the motor power supply and saves it in volts in the “Voltage” variable.

*Motor=1 (*to *motor=3)* returns the motor power supply in volts for the first *M50.PCI stepping motor card* (*card1), and motor=4 (*to *motor=6)* the power supply in volts for the second *M50.PCI stepping motor card* (*card2)*, etc.

Function value: function_OK= 0 • error_not_open= -2 • error_invalid_motor-no= -4
error_function_not_supported= -17

8.6 System Settings

Save_Configuration(FilePath : TFilePath): INTEGER;

The following settings of the initialization routines are saved for each of the 15 axes possible in the *FilePath* file and can be loaded again, if necessary:

- Setting of the phase current in mA (Set_Maximum_Current)
- Axis activated/de-activated (Enable_Motor)
- Full steps per revolution of the motor axis (Set_Steps_per_Rev)
- Linear axis / Rotational axis (Set_Rotary)
- Gear transmission ratio (Set_Gear)
- Spindle pitch in mm per revolution (Set_Pitch)
- Speed in point-to-point mode (Set_Point_Feedrate)
- Acceleration (Set_Acceleration)
- Assignment of a positive limit switch (Set_Pos_Limit_Switch)
- Assignment of a negative limit switch (Set_Neg_Limit_Switch)
- Polarity of the positive limit switch (Set_Pos_Limit_Polarity)
- Polarity of the negative limit switch (Set_Neg_Limit_Polarity)
- Assignment of a reference switch (Set_Reference_Switch)
- Polarity of the reference switch (Set_Reference_Polarity)
- Power reduction during pauses in movement (Set_Power_Down)

The following system settings are saved once for each *M50.PCI stepping motor card*:

- Interpolation activated/de-activated Interpolation_enable
- Interpolation pathway speed Interpolation_Feedrate
- Interpolation pathway acceleration Interpolation_Acceleration

```
Example:
procedure TMainForm.FormClose(Sender: TObject; var
                              Action:TCloseAction);
begin
 If Save_configuration('Demo1.cfg')=0 Then
  comment:=('Configuration saved')
 else comment:= ('Configuration not saved')
end;
```

Function value: function_OK = 0 • error_not_open = -2 • error_opening_config_file = -5

Read_Configuration(FilePath: TFilePath): INTEGER;

This command requests the settings saved by <*Save_Configuration*> from the *FilePath* file.

If this routine or the various initialization routines are not requested, the (0) settings are valid; i.e., all axes are completely switched off.

Function values: function_OK = 0 • error_not_open = -2 • error_no_config_file = -8
error_incompatible_config_file = -22 • error_no_valid_config_file = -23

Enable_Interpolation: INTEGER;

This function selects the interpolation mode.

To control several axes of a system, you can choose “point-to-point mode” or the “interpolation mode.” In the point-to-point mode, the system axes are controlled independently of one another, and each axis moves at its own rate of acceleration and speed (*Acceleration* and *Feedrate*).

In the interpolation mode, the speeds of the axes depend on one another. In this mode, the axes of each particular *M50.PCI stepping motor card* are combined in a Cartesian coordinate system; motors 1/2/3 correspond to the X/Y/Z axes.
The *SET_Goal* function specifies the X/Y/Z coordinates of a point to be reached. The movement of each axis is controlled so that all axes simultaneously reach a target position, where the speeds are calculated proportionally to the distance to be covered.

Setting: Disabled

Example: see next page

The routine cannot be carried out if a motor is still running when this command is called.

Function value: function_OK = 0 • error_not_open = -2

The maximum interpolation acceleration you select for a distance to be covered should not be higher than that which can be reliably attained by the axis with the lowest dynamics. Under certain circumstances, you may need to determine this value by the trial and error method.

```
Example:

Procedure TMainForm.FormCreate(Sender: TObject);
Var    feedrate     : INTEGER;
       acceleration : INTEGER;
begin
 If All_Arrived()=0 Then
 If Enable_Interpolation=0 Then begin
  Set_Interpolation_Feedrate(2.0);      //Travel speed in mm/s
  Set_Interpolation_Acceleration(10.0); //Travel acceleration in mm/s²
  comment:=('Interpolation Modus activated');
 end;
 Set_Goal(1,15);
 Set_Goal(2,15);
 Set_Goal(3,15);
 Go;         //The axes move in a diagonal from the center point
             //to a corner point of a square measuring 30 mm
end;
```

Disable_Interpolation: INTEGER**;**

This command changes from the interpolation mode back to the point-to-point mode.

The routine cannot be carried out if a motor is still running when the command is called.

Function value: function_OK = 0 • error_not_open = -2

Set_Interpolation_Feedrate(IntpolFeedrate: SINGLE): INTEGER;

In the interpolation mode, this command defines the travel speed in mm/s (linear axis) or in π/2s (rotational axis). The speeds of all system axes are internally controlled so that all axes reach their respective target positions simultaneously (see also Enable_Interpolation).

Setting: *IntpolFeedrate* = 3.0 (mm/s or π/2s)

The routine cannot be carried out if a motor is still running when this command is called.

Function value: function_OK = 0 • error_not_open = -2

Get_Interpolation_Feedrate(VAR IntpolFeedrate: SINGLE): INTEGER;

In the interpolation mode, this command returns *IntpolFeedrate* as the travel speed in mm/s (linear axis) or in π/2s (rotational axis).

Function value: function_OK = 0 • error_not_open = -2

Set_Interpolation_Acceleration(IntpolAcceleration: SINGLE): INTEGER;

This command defines the travel acceleration in mm/s^2 (linear axis) or in π/2s^2 (rotational axis) in the interpolation mode. The acceleration rates of all system axes are internally controlled so that all axes reach their respective target positions simultaneously (see also Enable_Interpolation).

Setting: *IntpolAcceleration* = 10.0 (mm/s^2 bzw. π/2s^2)

The routine cannot be carried out if a motor is still running when this command is called.

Function value: function_OK = 0 • error_not_open = -2

Get_Interpolation_Acceleration(VAR IntpolAcceleration: SINGLE): INTEGER;

This command returns *IntpolAcceleration* as the travel acceleration in mm/ s^2 (linear axis) or in π/2s^2 (rotational axis) in the interpolation mode.

Function value: function_OK = 0 • error_not_open = -2

Number_of_Cards (Motor: INTEGER, Var Slot_Number: INTEGER): INTEGER;

This command returns *Slot_number,* the slot number of the *M50.PCI stepping motor card* that is assigned to the motor searched for.

If at least one card in the system is found, the function value < *Number_of_Cards* > shows the number of the cards found; otherwise, an error code will be displayed.

```
Example
Var M50Cards: Integer;
.
.
M50Cards:= Number_of_Cards(1, Slot_Number); //Get slot number of motor1
if M50Cards > 0 then
 begin
  SlotCard1Edit.Text := IntToStr(Slot_Number);
 end;

M50Cards:= Number_of_Cards(4, Slot_Number); //Get slot number of
                                            //motor2 and return no.of
                                            //cards or error code
```

Function value: cards_found = < 0 ≤ 5 • error_not_open = -2 • error_invalid_motor_no = -4

8.7 Demo Program

The CD-ROM supplied contains the application program "**M50_Demo1.exe**". It is a simple programming example for controlling a *M50.PCI stepping motor card* with three motors.

User's manual for **M50_Demo1.exe**

Once the program is initialized, a user interface appears on the screen. Using this screen, you can initialize and control the *M50.PCI stepping motor card* by mouse click or keyboard input.

Position :	Shows the actual position in mm (system axis defined as a linear axis) or in radian measure (system axis defined as a rotational axis).
Goal :	Here, a target position is input in mm (system axis defined as a linear axis) or in radian measure (system axis defined as a rotational axis).
Feedrate :	Here, the system speed can be input in mm/s (linear axis) or π/2s (rotational axis).
Neg Limit Switch :	A negative limit switch can be assigned to each motor. Movement in the negative direction is interrupted once the end point of the limit switch is reached.
Pos Limit Switch :	A positive limit switch can be assigned to each motor. Movement in the positive direction is interrupted once the end point of the limit switch is reached.
"Go" Button:	Starts all motors simultaneously.
"Stop" Button:	Stops all motors.

9 Appendix A: Technical Specifications

9.1 System Data

- PC plug-in card with integrated, pulse-controlled terminating stages for three axes in microsteps
- Position controlling via signal processor
- All axes can be operated simultaneously or individually, either in the interpolation or the point-to-point mode
- Two limit switches /reference switches per axis can be configured by software as normally closed/normally open contacts
- Automatic power reduction of 0 to 100% when axes stop can be set by software
- The power of the motors can be switched off by software
- Power supply 12V generated by the PC power supply unit can be selected by the hard disk connector or externally up to 48V by jumper
- Windows® plug and play capability
- The system can be extended up to five *M50.PCI stepping motor cards*, each controlling three motors
- Demo program; source code in Delphi 5.0
- 32-bit driver for WINDOWS® 95/98/2000/XP

9.2 Performance Data

- For 3 bipolar stepping motors with 4-, 5-, 6- or 8-wire technology
- Motor voltage from 2V to 12V
- 2.5A phase current maximum; can be selected independently by software for each axis
- Microstep resolution 10-bit or 1.5 mA (corresp. to1/1000-microstep for 1.5-A motors)

Value ranges	Max.	Resolution	Typical	Unit
Position	84.5Mill	0.02	20000	full step
Speed (feed rate)	100000	0.001	2000	full steps/s
Acceleration	8000	0.1	10000	full steps/s^2

- Step frequency:
  The step frequency is controlled and monitored by a signal processor and can therefore considerably exceed the start-stop frequency of a motor. For this reason, the acceleration rates have to be carefully selected in order to prevent premature blockage of the motor.
- Operating efficiency:
  Up to 20-W power loss per motor in continuous operation; 3 motors, each with 1.6ΩV/2.5A power monitoring, and a temperature sensor monitors the operating state in each high-level stage

If several *M50.PCI stepping motor cards* are operated off one PC power supply, be sure that the total of all phase current ratings does not exceed 4.5A.

- 2 limit switch inputs per axis; can be selected as positive or negative by software
- 10V/100mA power supply for active limit switches
- Short-circuit or reverse battery protection on a short-term basis; overload protection by temperature monitoring
- Power supply:
  Internal 4-pin hard disk connector (such as Bürklin 71F972)
  External 2-pin power connector, type: WECO 120A111/02 (such as Bürklin 47F1300)
- Motor connection:
  25-contact D-Sub female connector

10 Appendix B: Structure of the M50.PCI

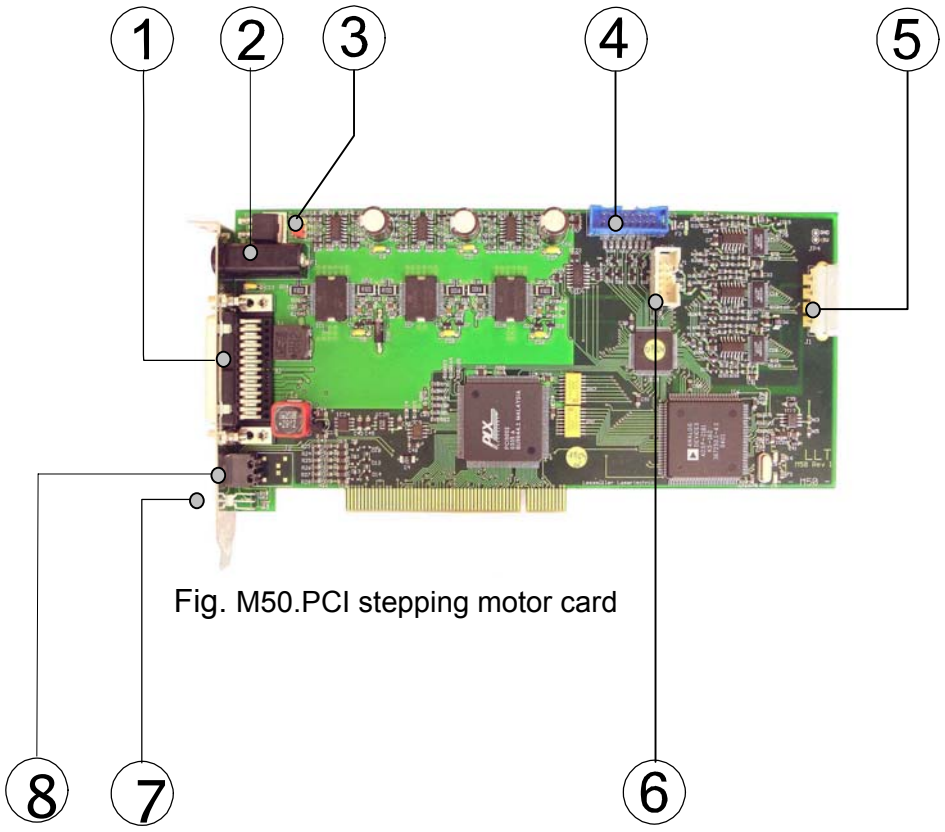


Fig. M50.PCI stepping motor card

1. Connector for motor and limit switches
2. Fuse holder, 3.15A quick-acting (fast-blow fuse)
3. Jumper for selecting the motor power supply, either internal (12-V hard disk) or external up to 48V
4. I/O extension
5. Hard disk connector for internal motor power supply
6. Programming interface for PLD (can only be configured by the manufacturer)
7. LED display for DSP
8. Connector for external power supply

11 Appendix C: Pin Assignment

11.1 Motor Connector

(25-contact D-Sub female connector)

D-Sub, 25	Signal name
1	Mot1B
2	Mot1A
3	Mot1B‘
4	Mot1A‘
5	Mot2B
6	Mot2A
7	Mot2B‘
8	Mot2A‘
9	Mot3B1
10	Mot3A1
11	Mot3B‘
12	Mot3A‘
13	+10V/100mA
14	END1+
15	END1-
16	GND
17	NC
18	END2+
19	END2-
20	GND
21	NC
22	END3+
23	END3-
24	GND
25	NC

Pin assignment of the motor connector

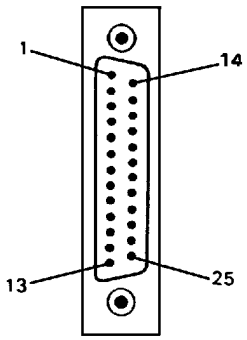


Fig.: 25-contact D-Sub motor connector showing pin assignment

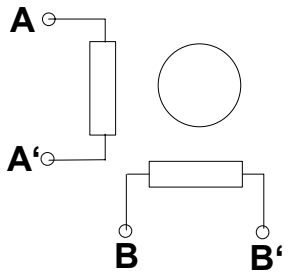


Fig. Example: Motor connection

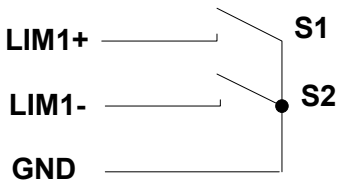


Fig. Example: Limit switch conections

11.2 I/O Extension:

(15-contact D-Sub female connector)

D-Sub, 15	Signal name
1	Vcc +5V
2	IO-0
3	IO-2
4	IO-4
5	IO-6
6	NC
7	NC
8	NC
9	GND
10	IO-1
11	IO-3
12	IO-5
13	IO-7
14	NC
15	NC

Table: I/O extension

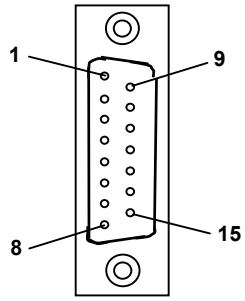


Fig.: Pin assignment for 15-contact D-Sub I/O extension

11.3 Limit Switch Wiring

Wiring of passive and active limit switches

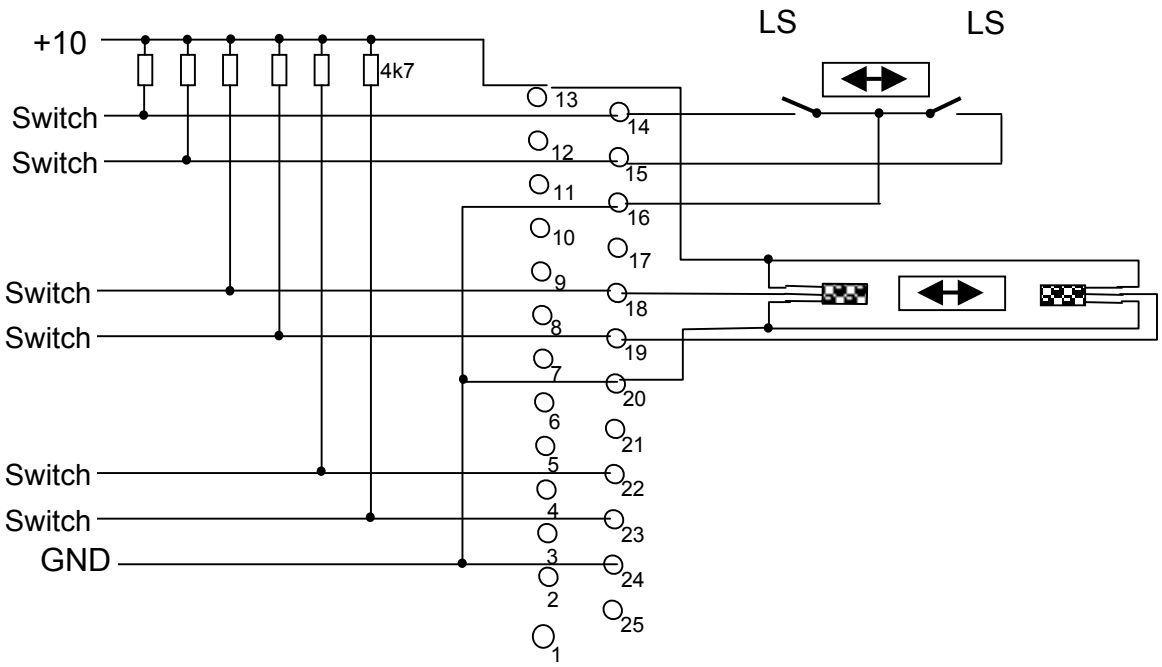


Fig. Example: wiring diagram of passive and active limit switches

12 Appendix E: Software Default Settings

M50-ConfigurationFile

Number of Cards : 1
Interpolation Mode, card 1 :0
Interpolation Feedrate, card 1 :3
Interpolation Acceleration, card 1 :10
Maximum Motor Current, motor 1 :1400
Maximum Motor Current, motor 2 :1400
Maximum Motor Current, motor 3 :1400
Enable Motor, motor 1 :1
Enable Motor, motor 2 :1
Enable Motor, motor 3 :1
Steps per Rev., motor 1 :200
Steps per Rev., motor 2 :200
Steps per Rev., motor 3 :200
Rotary axis, motor 1 :0
Rotary axis, motor 2 :0
Rotary axis, motor 3 :0
Gear, motor 1 :1
Gear, motor 2 :1
Gear, motor 3 :1
Pitch, motor 1 :2
Pitch, motor 2 :2
Pitch, motor 3 :2
Feedrate, motor 1 :3
Feedrate, motor 2 :3
Feedrate, motor 3 :3
Acceleration, motor 1 :10.0
Acceleration, motor 2 :10.0
Acceleration, motor 3 :10.0
Pos. Limit Switch, motor 1 :2
Pos. Limit Switch, motor 2 :4
Pos. Limit Switch, motor 3 :6
Neg. Limit Switch, motor 1 :1
Neg. Limit Switch, motor 2 :3
Neg. Limit Switch, motor 3 :5
Pos. Limit Polarity, motor 1 :0
Pos. Limit Polarity, motor 2 :0
Pos. Limit Polarity, motor 3 :0
Neg. Limit Polarity, motor 1 :0
Neg. Limit Polarity, motor 2 :0
Neg. Limit Polarity, motor 3 :0
Reference Switch, motor 1 :0
Reference Switch, motor 2 :0
Reference Switch, motor 3 :0
Reference Polarity, motor 1 :0
Reference Polarity, motor 2 :0
Reference Polarity, motor 3 :0
Power Down, motor 1 :50
Power Down, motor 2 :50
Power Down, motor 3 :50

13 Disposal

Warning! This LINOS product should NOT be thrown into ordinary waste disposal bins. If this LINOS product is not required any longer and you want to dispose of it, then please send it to the specified address given below for professional disposal. Thank you very much!

14 Service

For service or repair work, please contact:

LINOS-Photonics GmbH & Co. KG
Service Department
Königsallee 23
37081 Göttingen, Germany

Photonics LINOS
GmbH & Co. KG
Königsallee 23
D-37081 Goettingen
Germany

Phone++49 (0) 551 6935 0
Fax ++49 (0) 551 6935 166

LINOS Photonics Inc.
459 Fortune Boulevard
Milford MA 0 17 57 – 17 45

USA

Phone ++1 508 478 6200
Fax ++ 1 508 478 5980

LINOS-Photonics Ltd.
2 Drakes Mews
Crownhill, Milton Keynes
Buckinghamshire MK8 OER
Great Britain

Phone ++44 908 262525
Fax ++44 908 262526