Stepper Motor Controllers
8SMCC PCI1 and 8SMCC PCI3

Index

1 General Information

1.1 Applications

The 8SMCC PCI1/PCI3 is a single/triple axis 2-phase bipolar stepper motor motion control module. It is designed to work on PCI bus of computer.
STANDA manufactures lots of motorized devices such as translation stages, rotation stages, attenuators and other equipment. All stepper motors used in STANDA's motorized devices can be controlled by 8SMCC PCI1/PCI1/PCI3 controller.
Controller can be used to drive motorizes positioners of other manufacturers if stepper motor parameters mach specifications for 8SMCC PCI1/PCI1/PCI3.

1.2 Features

1. Module controls all parameters of motion: velocity, acceleration, deceleration, amount of steps (or micro steps) and direction.
2. Controller supports three synchronization modes:
   - ❖ Starts motion after external sync pulse is received,
   - ❖ Generates sync pulse when motion is finished,
   - ❖ Periodically generates sync pulses starting from defined step with defined period.
3. Controller supports step division up to 1/8 steps (1, 1/2, 1/4, 1/8 step).
4. Two end switches and one rotation switch can be connected to the controller.
   Rotational switch operates in two modes:
   - ❖ Rotational switch is ignored
   - ❖ Controller counts amount of steps (micro steps) until switch will be pressed the first time and between all other presses and sends this information to computer.

   End switches are independent and each can operate in three modes:
   - ❖ End switch is ignored,
   - ❖ Emergency stop, when end switch is pressed,
   - ❖ “Check position” mode (computer receives information that switch was pressed and motion continues).
5. Controller is capable to stop motor with deceleration in motion mode (“soft stop”).
6. Information about status of end switches can be read from controller.
7. Motor current can be switch off or on to disconnect motor safely.
8. Controller turns of power when signal from one of two emergency switches is received.

1.3 Compatibility

1.3.1 Connectivity

Controller is designed to work with IBM AT compatible computer systems (with 80486, Pentium or better processors). The only requirement is presence of PCI slots on motherboard.

1.3.2 Version

Version of current controller card is 1.0. Version of firmware is 1.0.

1.3.3 Stepper motors

Controller can operate with stepper motors accordingly technical specification and wiring requirements.

1.4 Technical specifications

Microcontroller:	PIC16F874-C20
Command protocol:	version 1.0
Maximum amount of steps:	65535
Limit switches:	2
Emergency switches:	2
Rotational switch:	1
Logic powering:	5V DC
Powering of stepper motors from internal PC power supply:	5V DC or 12 DC
Powering of stepper motors from external power supply:	5- 40V DC, up to 5A
Step division:	1; 1/2; 1/4; 1/8 of step
Operating temperature:	Accordingly requirements to PC
Output connector:	DB-15 (8SMCC PCI1) DB-37 (8SMCC PCI3)
Power dissipation:	4W for one axis

1.5 Precautions

Reasons that might cause the 8SMCC PCI1/PCI3 card to malfunction

- The controller card was replaced in the computer while the computer was switched on.
- Any motorized device was connected to, or disconnected from the controller card, while the controller card keeps currents in the motor windings. Please avoid reconnecting the motor while a current is turned on.
- While reinstalling the card, you touched the contacts of PCI connector, or of some elements on the card.
- The microprocessor chip was taken out from its slot on the card and then reinserted in reverse (accidentally rotated 180 degrees).
- It would be better, if the controller doesn't share power-input cable with any other internal device of a computer.

1.6 Number of controller cards in one computer

Number of controller cards in one computer is limited only by amount of PCI slots on the motherboard.

1.7 Power supply

Logic is always powered from 5V on PCI bus. Powering for stepper motors can be realized in two ways:

- From internal power supply of PC (Standard configuration provides 12V powering for stepper motors, but system can be adopted to provide 5V powering accordingly customer needs). Every card is provided with a power extension lead and a group of cards can be connected to the one lead of internal power supply.
- 5-40V powering can be received from external power supply. External power supply connects directly to the connector located on the front panel of PCI card. Additional power supply can be used

if stepper motor needs a greater power than PC power supply can supply or operating voltage is not 5 and 12V.

1.8 Operation

1.8.1 Reset

Special reset button is placed on the front panel of controller card. Reset operation is performed for all three controllers together. After reset button is pressed controller stops motion and sets values of all parameters to default. Reset operation can be used when software fails to communicate with controller card and controller stays in undefined communicate stage.

1.8.2 Independent operation

The card doesn't use any resources of logic of your computer. All other programs of your multitasking environment will proceed unhindered. The card will continue operating the attached device, even after your computer reboots. The card stops operating, when computer is turned off.

1.9 Waranty

Standa warrants the controller card 8SMCC1 PCI1/PCI3 for the period of 1 year from the date of sale.

1.10 Wiring

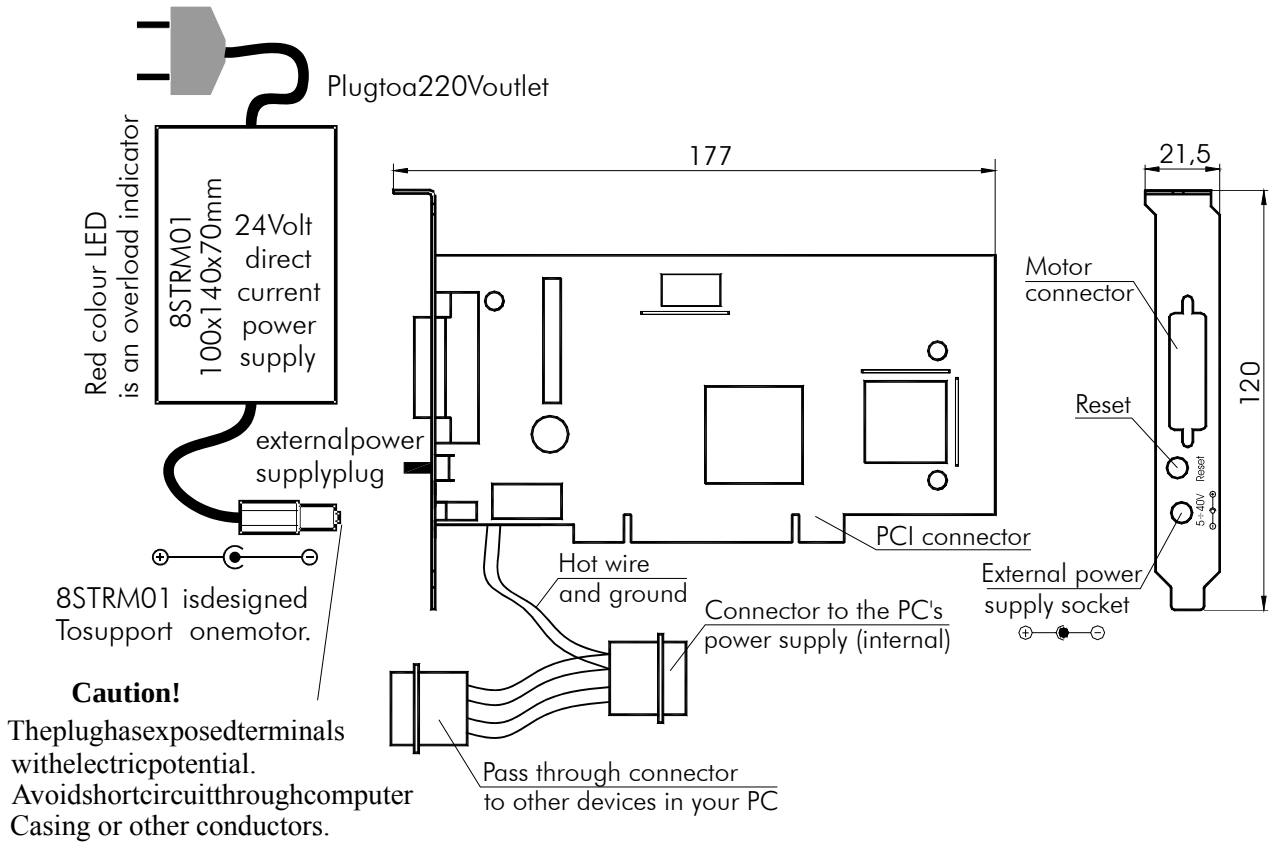


Figure 1. Wiring for the controller card 8SMCC PCI1 with 24Vdc optional external power supply

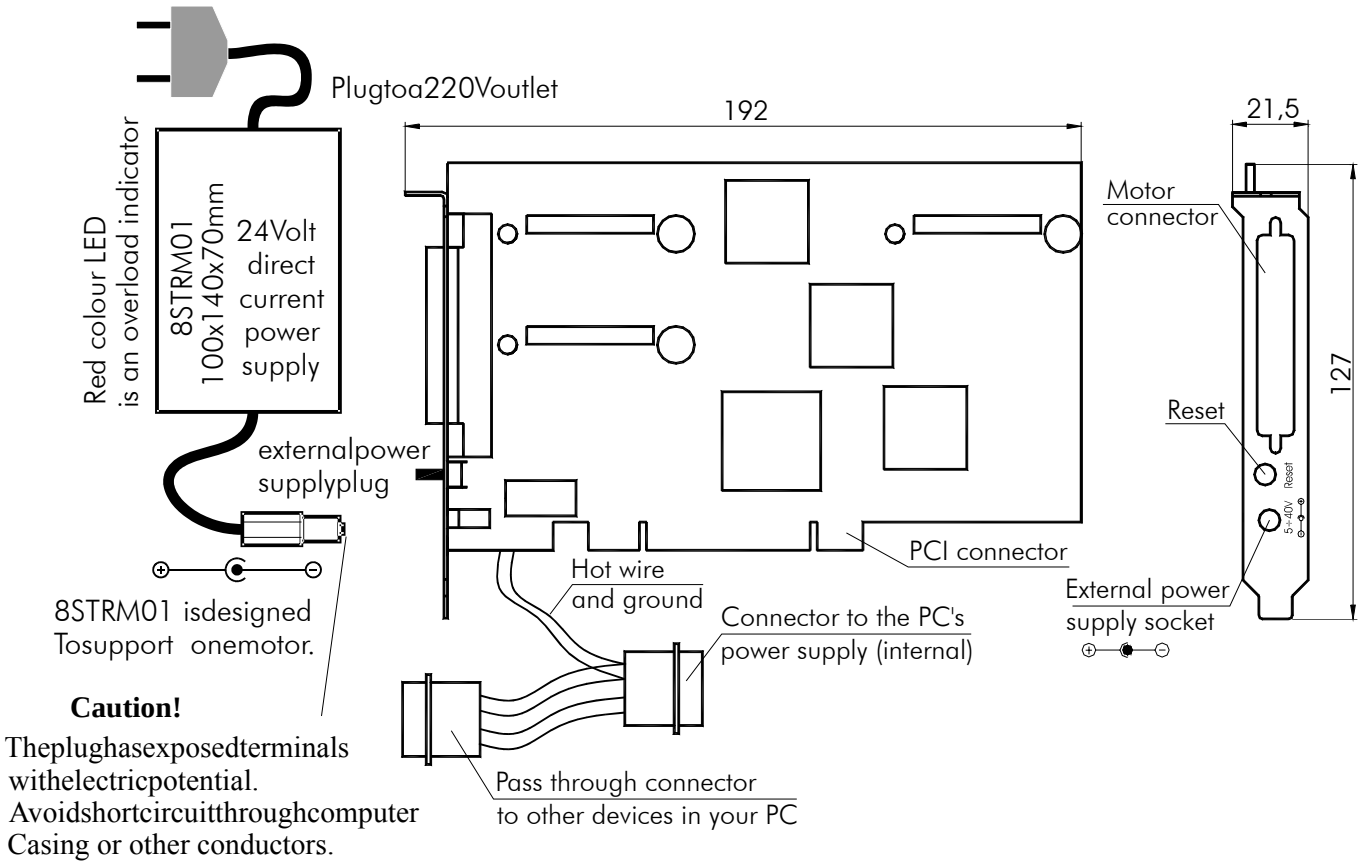


Figure 2. Wiring for the controller card 8SMCC PCI3 with 24Vdc optional external power supply

Figures 1 and 2 present wiring for 8SMCC PCI1 and 8SMCC PCI3 cards respectively.
Wiring diagram for cards connectors DB15 (for 8SMCC PCI1) and DB37 (for 8SMCC PCI3) are presented in figures 5 and 6.
The card can have up-to 40 Volts (direct current) power supply either on internal or external connector. When you plug in the external power supply, it automatically disconnects the internal hot wire from the PC power supply. Please take care to switch on the power supply or computer only after the motor and power cables are connected.
Our latest cards have optical isolation on each limit switch/synch wire. It is disables by default. Contact us for details on how to enable it.
Note. Synchronization (sync) wires are not provided by default. They should be soldered on by the user at one's convenience. Usually they are soldered onto cable.
All STANDA motorized translation stages are provided with DB9 connector to connect stepper motor and end switches. Wiring diagram is presented in figure 3.

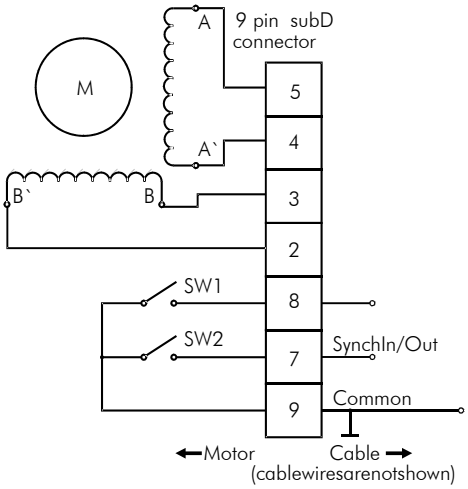


Figure 3. Motor DB9 connector wiring diagram.

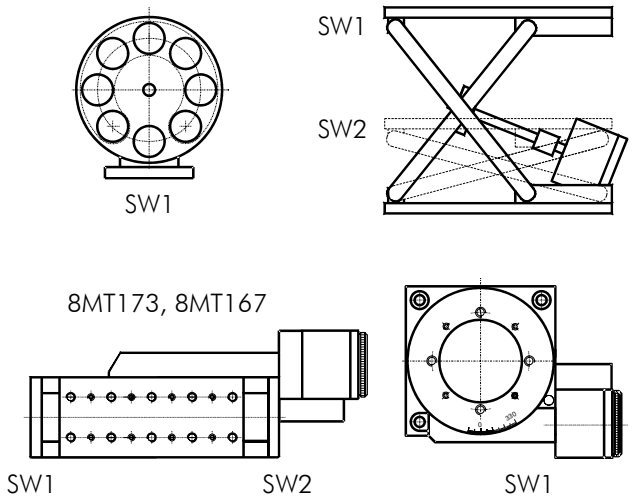


Figure 4. Some of the devices that may be driven by the card 8SMCC PCI1/PCI3.

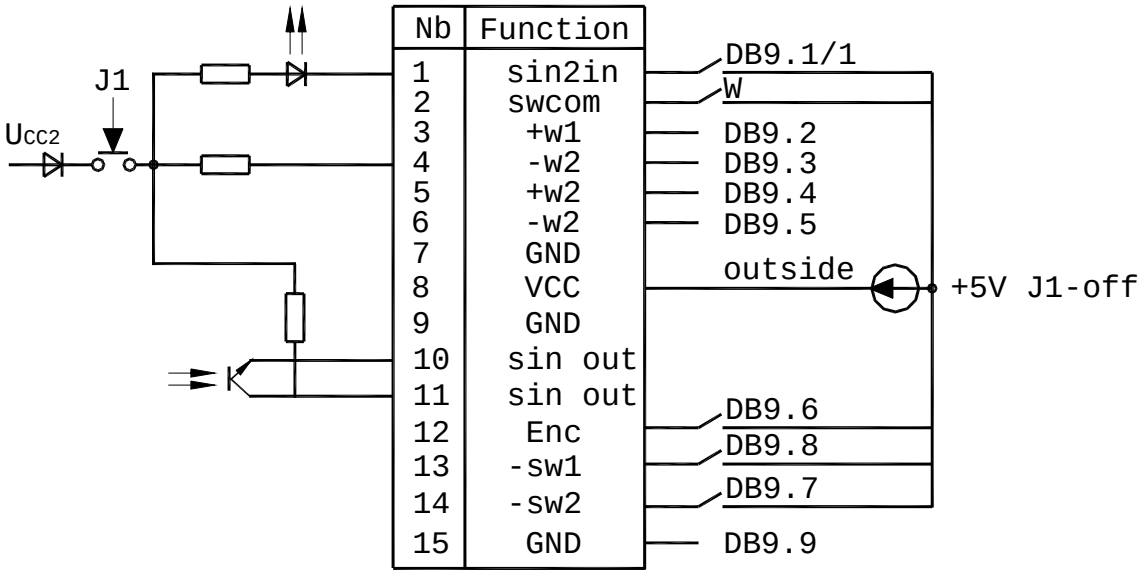


Figure 5. 8SMCC PCI1 connector DB-15 wiring diagram.

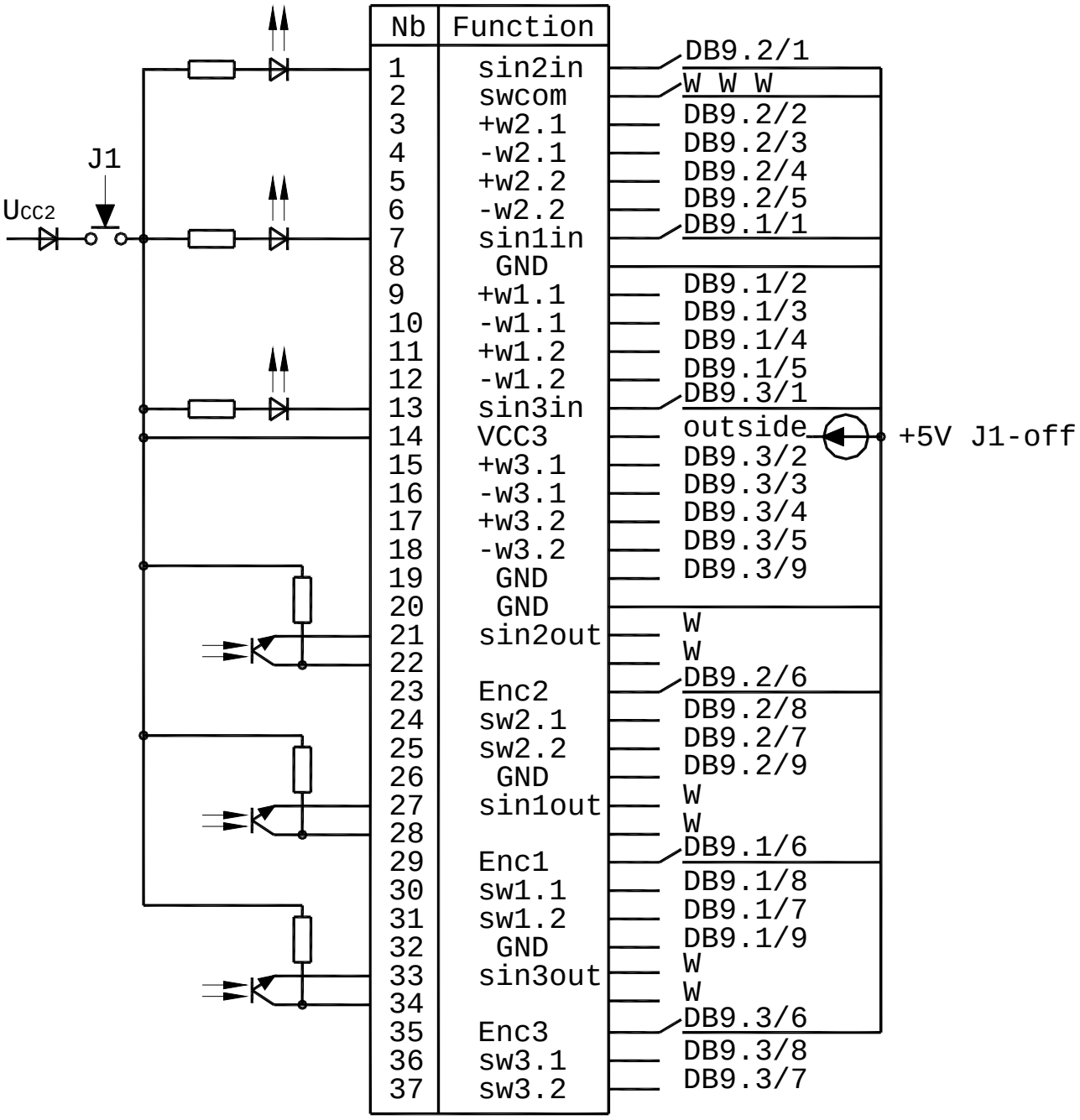


Figure 6. 8SMCC PCI3 connector DB-37 wiring diagram.

2 Installation

Installation procedure consists of several steps:

1. Installation of PCI card on computer.
2. Installation of QLIB driver and library (QLIB stands for QUANCOM driver library).
3. Installation of 8SMCC PCI1/PCI3 library and additional software.

2.1 Installation of PCI card on computer

- Turn of a computer. Disconnect all power leads and other cables.
- Remove cover to access top side of a motherboard. Computer motherboards and components contain very delicate integrated circuit (IC) chips. To protect them against damage from static electricity, you must follow some precautions whenever you work on your computer. Use a grounded wrist strap before handling computer components. If you don't have one, touch both of your hands to a safely grounded object or to a metal object, such as the power supply case
- Locate a free PCI slot. The slots are positioned at the backside of your computer. The back wall of unused slots is covered by small metal plates. Search for a free slot, detach its holding screw and remove the small metal plate belonging to it.
- Gently push 8SMCC PCI1/PCI3 PCI card to a free PCI slot. Hold components by the edges and try not to touch the integrated circuit chips, leads or circuitry.
- Use screw to fix a PCI card to the case of a computer.
- Locate free power lead from a computer's internal power supply and connect it to an extension power lead attached to a PCI card. If there is not a free power lead from internal power supply, disconnect for example power lead from a hard disc, connect it to an extension power lead attached to a PCI card and connect another connector of an extension lead back to a hard disc.
- Put cover back.
- Connect all power leads and other cables, including DB37 cable to the 8SMCC PCI1/PCI3 controller card.
- Turn on a computer.

2.2 Installation of QLIB driver and library

2.2.1 Installing the QLIB and the drivers under Windows XP / 2000

After you inserted the 8SMCC PCI1/PCI3 board in your system it will be recognized automatically by Windows during system restart.

2.2.1.1 Driver installation under Windows XP / 2000

The system detects the new hardware device and opens a dialog box „Found New Hardware Wizard".

- Insert the CD-ROM labeled "8SMCC PCI1/PCI3" in your CD-ROM drive.
- Press the "Next" button to continue the installation.
- Select the option "Search for a suitable driver for my equipment (recommended)". Press the "Next" button to continue the installation.
- The next dialog is for the selection of the driver files location. Select "Specify a location" and press the "Next" button to continue the installation.

- Windows opens a dialog to search for the manufacturer's installation disk. Click on button "Browse" to open the directory where the drivers can be found. To do this, choose to the CD-ROM drive and change to the directory QLIB\Win2000 or QLIB\WinXP, depending on the operating system you are using. Select the file QLIBXDRV.INF and click on "Open" to continue the installation. Click again on "OK" to continue the installation.
- In the next window click "OK" to continue the installation of the driver.
- In the next dialog Windows tells you that it has found a driver for this device. Press the "Next" button to continue the installation.
- Press the "Finish" button to end the installation of the drivers.

2.2.1.2 QLIB installation under Windows XP / 2000

After the driver installation you have to run the QLIB Software Setup.

- Run the quancom.exe program on the CD-ROM drive. Click on Start | Run. Type drive:\QLIB\quancom.exe and click on "OK". (Replace the string drive with the letter for you CD-ROM drive, i.e. d:\QLIB\quancom.exe).
- Depending on the system configuration and the software version of your installer the setup may asks you to reboot the system. Click "Yes" if the install asks you to reboot the system (after the system has rebooted the installer will continue the interrupted installation).
- In the next step click "OK" to continue the installation of the QLIB.
- Accept the license agreement with the "I accept the license agreement" option button and press the "Next" button to continue the installation.
- Enter your data (Name; Organization) and choose whether you want to install the software solely for the current user or for all users of the system (you need administrative rights for this option). Press the "Next" button to continue the installation.
- To change the installation directory click on the "Browse" button or press the "Next" button to continue the installation in the default directory.
- Select the installation type and press the "Next" button to continue the installation. There are three types of installation available:
  - Typical (This option installs all drivers and the API but no samples and help files),
  - Complete (This option installs all the drivers, the API, all samples and the help files),
  - Custom (Customize your installation with this option and select which parts to install).

  Only drivers and API are necessary for 8SMCC PCI1/PCI3 library. Samples and help files are optional.
- After the installer has copied all files to the destination directory press the "Finish" button to end the installation.
- The installer asks whether you want to restart the PC. Select "YES" to restart the PC and "NO" to restart the PC later.

2.2.2 Installing the QLIB and the drivers under Windows ME / 98 / 95

After you inserted the 8SMCC PCI1/PCI3 board in your system it will be recognized automatically by Windows during system restart.

2.2.2.1 Driver installation under Windows ME / 98 / 95

Windows should automatically detect your hardware and display one or more New Hardware Found dialog boxes.

- Insert the CD-ROM labeled "8SMCC PCI1/PCI3" in your CD-ROM drive.

- Press button “Next” to continue the installation.
- Select the option “Search for the best driver for your device (recommended)” and press the “Next” button to continue the installation.
- Select the option “Search location” and press “Next”.
- In the new dialog box press the button “Browse”. Change the drive and directory to the CD-ROM drive and the directory to WinME, Win98 or Win95, depending on the operating system you are using. Select the file “QUANCOM.INF” and then press the “OK” button.
- In the next window press button “Next” to continue with the installation.
- Press the “Finish” button to end the installation.

2.2.2.2 QLIB installation under Windows ME / 98 / 95

After installing the drivers you have to run the QLIB Software Setup.

- Run the quancom.exe program on the CD-ROM drive. Click on Start | Run. Type drive:\QLIB\quancom.exe and click on “OK”. (Replace the string drive with the letter for you CD-Rom drive, i.e. d:\QLIB\quancom.exe).
- Depending on the system configuration and the software version of your installer the setup may asks you to reboot the system. Click “Yes” if the install asks you to reboot the system (after the system has rebooted the installer will continue the interrupted installation).
- In the next step click “OK” to continue the installation of the QLIB.
- Accept the license agreement with the “I accept the license agreement” option button and press the “Next” button to continue the installation.
- Enter your data (Name; Organization) and choose whether you want to install the software solely for the current user or for all users of the system (you need administrative rights for this option). Press the “Next” button to continue the installation.
- To change the installation directory click on the “Browse” button or press the “Next” button to continue the installation in the default directory.
- Select the installation type and press the “Next” button to continue the installation. There are three types of installation available:
  - Typical (This option installs all drivers and the API but no samples and help files),
  - Complete (This option installs all the drivers, the API, all samples and the help files),
  - Custom (Customize your installation with this option and select which parts to install).

  Only drivers and API are necessary for 8SMCC PCI1/PCI3 library. Samples and help files are optional.
- After the installer has copied all files to the destination directory press the “Finish” button to end the installation.
- The installer asks you whether you want to restart the PC. Select “YES” to restart the PC and “NO” to restart the PC later.

2.2.3 Installing the QLIB and the drivers under Windows NT

Under Windows NT 4 the drivers for a PCI card are installed with the QLIB Software Setup.

- Insert the CD-ROM labeled “8SMCC PCI1/PCI3” in your CD-ROM drive.
- Run the quancom.exe program on the CD-ROM drive. Click on Start | Run. Type drive:\QLIB\quancom.exe and click on “OK”. (Replace the string drive with the letter for you CD-ROM drive, i.e. d:\QLIB\quancom.exe).
- Depending on the system configuration and the software version of your installer the setup may asks you to reboot the system. Click “Yes” if the install asks you to reboot the system (after the system has rebooted the installer will continue the interrupted installation).
- In the next step click “OK” to continue the installation of the QLIB.

- Accept the license agreement with the "I accept the license agreement" option button and press the "Next" button to continue the installation.
- Enter your data (Name; Organization) and choose whether you want to install the software solely for the current user or for all users of the system (you need administrative rights for this option). Press the "Next" button to continue the installation.
- To change the installation directory click on the "Browse" button or press the "Next" button to continue the installation in the default directory.
- Select the installation type and press the "Next" button to continue the installation. There are three types of installation available:
  - Typical (This option installs all drivers and the API but no samples and help files),
  - Complete (This option installs all the drivers, the API, all samples and the help files),
  - Custom (Customize your installation with this option and select which parts to install).

  Only drivers and API are necessary for 8SMCC PCI1/PCI3 library. Samples and help files are optional.
- After the installer has copied all files to the destination directory press the "Finish" button to end the installation.
- The installer asks whether you want to restart the PC. Select "YES" to restart the PC and "NO" to restart the PC later.

2.3 Installation of 8SMCC PCI1/PCI3 library

After installing QLIB library you have to run the 8SMCC PCI1/PCI3 Library Setup.

- Insert the CD-ROM labeled "8SMCC PCI1/PCI3" in your CD-ROM drive.
- Run the Setup.exe program on the CD-ROM drive. Click on Start | Run. Type drive:\8SMCC\Setup.exe and click on "OK". (Replace the string drive with the letter for you CD-ROM drive, i.e. d:\ 8SMCC\Setup.exe).
- In the next step click "Next" to continue the installation of the 8SMCC PCI1/PCI3.
- Select modules to install and press the "Next" button to continue installation. There are three optional modules available:
  - Translator application (Graphical user interface to control three motorized translators with embedded TCP/IP server/client for network communications).
  - LabVIEW library (Wrapper around 8SMCC2.dll for LabVIEW).
  - MatLAB library (Wrapper around 8SMCC2.dll for MatLAB).
- Select install option "Everyone" or "Just me". To change the installation directory click on the "Browse" button or press the "Next" button to continue the installation in the default directory.
- Click "Next" to continue the installation of the 8SMCC PCI1/PCI3.
- After the installer has copied all files to the destination directory press the "Close" button to end the installation.

3 Software layers

8SMCC PCI1/PCI3 controller card is designed on PCI bus controller produced by QUANCOM Informationssysteme GmbH. Therefore the low level communication is based on qlib32.dll library and qlibxdrv.sys driver. These two modules are installed as QLIB library before all other 8SMCC PCI1/PCI3 software. 8SMCC PCI1/PCI3 card has very simple programming interface and users can write their own programs (on any version of C, C++ compilers, Visual Basic, Delphi, etc) directly using functions from qlib32.dll.
However more recommendable way is to use task-orientated functions from 8SMCC PCI1/PCI3 library (8SMCC2.dll). This library was designed specially for 8SMCC PCI1/PCI3 controller card and it wraps all time-critical operations and card-specific communication protocols.
Library can be very easily integrated to any C, C++, and Pascal, Visual Basic or Delphi project.
To make this library even more versatile two additional supporting modules were included. LabVIEW virtual instruments library 8SMCC2.llb builds wrappers around every function from 8smcc2.dll (using "Call function" instrument). This allows building simple control applications in a few minutes with several building elements. Similarly simple wrappers were built to meet MatLAB interface. These two addition modules are installed optionally if user wants to control 8SMCC PCI1/PCI3 card from LabVIEW or MatLAB programming environments.

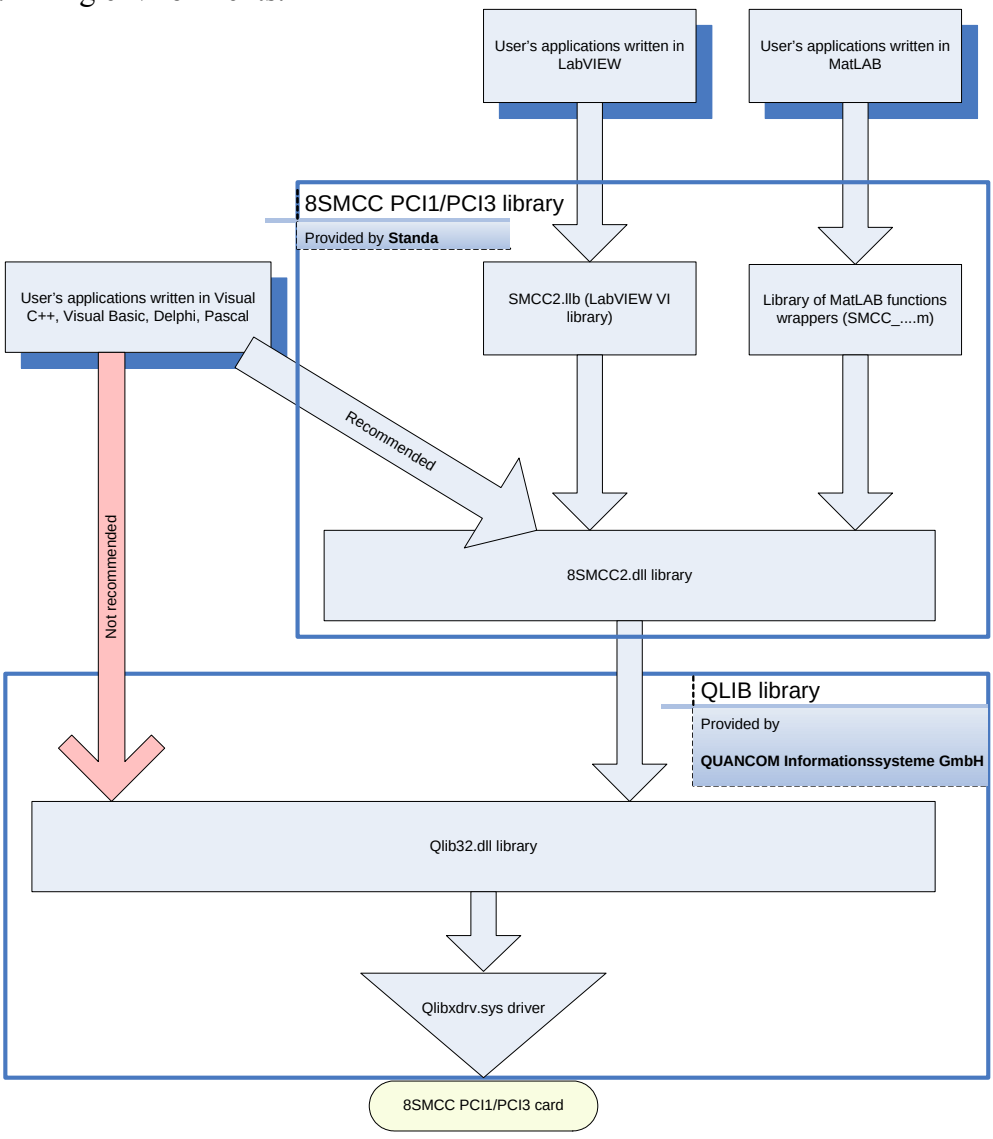


Figure 7. 8SMCC PCI1/PCI3 controller card software layers.

4 Command protocol

Controller card is controlled by mean of sending commands. Command protocol is the way of exchanging information between user's software and 8SMCC PCI1/PCI3. Every command is composed from two parts: command code and optional data segment. Command code is always 1 byte length. Data segment depending on command can be from 0 to 6 bytes length. There are only 11 different commands.

4.1 Command "Set acceleration" (code 01)

Sets acceleration parameter. Total length: 2 bytes. This command is accepted only when motor doesn't move.
Command format: 0x01, acceleration data (1 byte).
Acceleration data value can be from 1 to 119. (Default value set by controller on reset is 1).

4.2 Command "Set speed" (code 02)

Sets speed parameter. Total length: 2 bytes. This command is accepted only when motor doesn't move.
Command format: 0x02, speed data (1 byte).
Speed data value can be from 1 to 220. (Default value set by controller on reset is 220).

4.3 Command "Set steps" (code 03)

Sets number of steps to move. Total length: 4 bytes. This command is accepted only when motor doesn't move.
Command format: 0x03, steps (youngest byte), steps (higher byte), steps (highest byte).
Steps value can be from 1 to 16581375. (Default value set by controller on reset is 440).

4.4 Command "Set division factor" (code 04)

Sets step division factor. Total length: 2 bytes. This command is accepted only when motor doesn't move.
Command format: 0x04, division (1 byte).
Division value can be from 0 to 3. (Default value set by controller on reset is 0).

Value	Division factor
0	No division
1	Step divided by 2
2	Step divided by 4
3	Step divided by 8

4.5 Command "Set synchronization mode" (code 05)

Sets synchronization mode. Total length: 2 or 7 bytes. This command is accepted only when motor doesn't move.
Command format: 0x05, synchronization mode (1 byte), optional periodic sync pulse parameters (5 bytes). (Default value set by controller on reset is 0).
Structure of byte of synchronization mode:

Bit	Description

0	If 0, movement starts immediately after command "Go" is received. If 1, movement (after command "Go" is received) starts
1	If 0, no sync pulse is generated at the end of movement. If 1, sync pulse is generated at the end of movement.
2	If 0, no periodic sync pulses are generated. If 1, periodic sync pulses are generated with defined interval from defined step. If 2nd bit is set to 1, controller waits for 5 additional bytes describing parameters of periodic sync pulse.
3-7	Not used

Structure of periodic sync pulse parameters:

Byte	Description
0,1,2	Number of steps when the first sync pulse must be generated. Data format: youngest byte, higher byte, highest byte. Number of steps value can be from 1 to 16581375. (Default value set by controller on reset is 100).
3,4	Period of sync pulse generation in steps. Data format: younger byte, higher byte. Period value can be from 1 to 65535. (Default value set by controller on reset is 10).

4.6 Command "Set direction" (code 06)

Sets direction of movement. Total length: 2 bytes. This command is accepted only when motor doesn't move.
Command format: 0x06, direction (1 byte).
Steps value can be from 0 to 1. (Default value set by controller on reset is 0).

Value	Direction
0	Forward
1	Backward

4.7 Command "Set switch mode" (code 07)

Sets mode for end and rotational switches. Total length: 2 bytes. This command is accepted only when motor doesn't move.
Command format: 0x07, switch mode (1 byte).
Switch mode value can be from 0 to 31. (Default value set by controller on reset is 5).
Structure of switch mode byte:

Bit	Description
1,0	00- Left end-switch is ignored. 01- "Emergency stop" mode for left end-switch. If end-switch is pressed while motor moves, movement immediately stops. 10- "Check position" mode for left end-switch. If end-switch is pressed while motor moves, movement immediately stops, controller sends two bytes to computer:

	0x02, switch info byte (where 0 bit is set to 1 if left end-switch is pressed and 1st bit is set if right end-switch is pressed), when these two bytes are read, movement continues.
3,2	00- Right end-switch is ignored. 01- "Emergency stop" mode for right end-switch. If end-switch is pressed while motor moves, movement immediately stops. 10- "Check position" mode for right end-switch. If end-switch is pressed while motor moves, movement immediately stops, controller sends two bytes to computer: 0x02, switch info byte (where 0 bit is set to 1 if left end-switch is pressed and 1st bit is set if right end-switch is pressed), when these two bytes are read, movement continues.
3	0- Rotational switch is ignored. 1- Every time when rotational switch is pressed, controller sends 3 bytes to computer: 0x03, steps (younger byte), steps (higher byte), Where steps is amount of steps from the last push of rotational switch and one before last push of rotational switch or start of movement.
7,6,5,4	Not used

4.8 Command "Go" (code 08)

Starts movement of motor accordingly all defined parameters (acceleration, speed, number of steps, step division factor, synchronization mode, direction, and switch mode). Total length: 1 byte. This command is accepted only when motor doesn't move.
Command format: 0x08.
By default controller card turns off current of all three motors after reset. Command "Set motor status" (see chapter 4.11) must be used to turn current on before command "Go" is issued.

4.9 Command "Soft stop" (code 09)

Stops movement of motor accordingly acceleration/deceleration and speed parameters. After that controller sends 4 bytes to computer: 0x04, steps (youngest byte), steps (higher byte), steps (highest byte).
Where steps is amount of steps made from the start of movement. This parameter can be used to calculate actual position of motor after "Soft stop" command.
Total length: 1 byte. This command is accepted only when motor moves!
Command format: 0x09.

4.10 Command "Get switch info" (code 10)

Gets information from controller about state of end-switches. Total length: 1 byte. This command is accepted only when motor doesn't move.
Command format: 0x0a.
After this command is received controller sends 2 bytes to computer:
0x01, switch info byte (where 0 bit is set to 1 if left end-switch is pressed and 1st bit is set to 1, if right end-switch is pressed).

4.11 Command “Set motor status” (code 11)

Turns on or off motor current. Total length: 2 bytes. This command is accepted only when motor doesn’t move.
Command format: 0x0b, status (1 byte).
If status is 0, current is turned off and if status is 1, current is turned on. Usually this command must be issued before command “Go” after controller card reset or powering on.

5 QLIB library

5.1 General information

The QLIB, which stands for QUANCOM driver library, was developed by QUANCOM Informationssysteme GmbH with the target to allow the simple programming of all data acquisition products, produced by this company, under various operating systems. 8SMCC PCI1/PCI3 controller card uses PCI bus controller developed by QUANCOM, therefore QLIB library is used as the main agent between user's applications and hardware.

5.2 QLIB functions

There are only three functions used to communicate with 8SMMC2 PCI card. Descriptions of all other functions can be found in QLIB 32-Bit API Help. Two versions of this help (English and German) can be found on Start | Programs | QLIB 32-Bit | Help. All functions are declared in qlib.h file located by default in …\Program Files\QUANCOM\Qlib32\Inc.

5.2.1 Function QAPIExtOpenCard

```
unsigned long QAPIExtOpenCard (unsigned long cardid, unsigned long devnum)
```

With the function QAPIExtOpenCard a card is opened. This function must be called before any function of the group QAPIExt... is used. It returns the handle which is passed as first parameter to these functions. A card opened with this function must be closed with QAPIExtCloseCard. The `devnum` parameter differentiates multiple cards of the same type. Up to eight cards of same type are supported by the QLIB.
Input parameters:

`cardid`	The ID of the card which should be opened. For 8SMCC PCI1/PCI3 card `cardid` always must be equal to PCIPROTO.
`devnum`	The device number of the card which should be opened.

Output value: If the function succeeds, the return value is a handle for the card. If the function fails, the return value is NULL.

5.2.2 Function QAPIExtCloseCard

```
void QAPIExtCloseCard(unsigned long cardhandle)
```

With the function QAPIExtCloseCard a card is closed. This function must always be executed before leaving an application. Otherwise, the card resources are not freed, The handle you need as parameter is the handle you get from QAPIExtOpenCard.
Input parameters:

`cardhandle`	The handle of the opened card.

Output value: None.

5.2.3 Function QAPIExtSpecial

```
unsigned long QAPIExtSpecial (unsigned long cardhandle, unsigned long jobcode, unsigned long para1, unsigned long para2)
```

With the function QAPIExtSpecial it is possible to execute card-specific functions.
Input parameters:

`cardhandle`	Handle of an opened card.
`jobcode`	Action which the card should perform. In 8SMCC PCI1/PCI3 card case only two job codes are used: JOB_IOREAD_BYTE (Read a byte from a card) and JOB_IOWRITE_BYTE (Write a 8 bit value to a proto card).
`para1`	Offset of base address.
`para2`	Not used, when `jobcode`=JOB_IOREAD_BYTE and 8 bit value that has to be written to the port, when `jobcode`=JOB_IOWRITE_BYTE.

Output value: Read 8 bit value, when `jobcode`=JOB_IOREAD_BYTE, and not defined value, when `jobcode`=JOB_IOWRITE_BYTE.

5.3 Working with 8SMCC PCI1/PCI3 card over QLIB functions

8SMCC PCI1/PCI3 controller card has three identical and independent stepper motor drives. Four addresses are used to communicate with all three controllers:

Offset of base address	Description
0	Data write/read address for motor 1 controller.
1	Data write/read address for motor 2 controller.
2	Data write/read address for motor 3 controller.
3	Write/Read signals for motors 1, 2 and 3. Active state for all signals is low. 0 bit- WRITE signal for motor 1, 1st bit- READ signal for motor 1, 2nd bit- WRITE signal for motor 2, 3rd bit- READ signal for motor 2, 4th bit- WRITE signal for motor 3, 5th bit- READ signal for motor 3.

5.3.1 Opening communication channel with 8SMCC PCI1/PCI3

`QAPIExtOpenCard` function should be used at the beginning of communication. The first parameter `cardid` for 8SMCC PCI1/PCI3 card always must be equal to `PCIPROTO` and the second parameter `devnum` should be 0 for the first card, 1- for the second, etc.

5.3.2 Sending command to 8SMCC PCI1/PCI3

Data is written using `QAPIExtSpecial(card_handle,JOB_IOWRITE_BYTE,address,data)` function. Where `address` has value 0 for motor 1, 1- for motor 2 and 2 for motor 3. WRITE signal must be checked before writing data. If controller is ready to receive command or data, WRITE signal is set to 0. If WRITE=1, reading of last byte is still pending or motor moves. The only exclusion is command "Soft stop" (code 09). This command can be sent even if WRITE=1. Short example in C shows how to send command "Go" for motor 1:

```
unsigned long handle;
handle=QAPIExtOpenCard(PCIPROTO,0);
```

```
// wait while WRITE for 1-st motor is 1
while ((QAPIExtSpecial(handle,JOB_IOREAD_BYTE,3)&0x01)!=0);
// send command “Go”
QAPIExtSpecial(handle,JOB_IOWRITE_BYTE,0,8 /*8 is ”Go” command code*/);
//wait while motor moves
while ((QAPIExtSpecial(handle,JOB_IOREAD_BYTE,3)&0x01)!=0);
QAPIExtCloseCard(handle);
```

5.3.3 Reading data from 8SMCC PCI1/PCI3

Data is read using `QAPIExtSpecial(card_handle,JOB_IOREAD_BYTE,address)` function. Where `address` has value 0 for motor 1, 1- for motor 2 and 2 for motor 3. READ signal must be checked before reading data. If controller is ready to transmit data, READ signal is set to 0. If READ=1, no data is available for reading. Short example in C shows how to read state of end-switches:

```
unsigned long handle;
unsigned char byte1, status;
handle=QAPIExtOpenCard(PCIPROTO,0);
// wait while WRITE for 1-st motor is 1
while ((QAPIExtSpecial(handle,JOB_IOREAD_BYTE,3)&0x01)!=0);
// send command “Get switch info”
QAPIExtSpecial(handle,JOB_IOWRITE_BYTE,0,10 /*10 is ”Get switch info” command code*/);
//wait while READ for 1-st motor changes to 0
while ((QAPIExtSpecial(handle,JOB_IOREAD_BYTE,3)&0x02)!=0);
//read first byte (always should be 1)
byte1=(unsigned char) QAPIExtSpecial(handle,JOB_IOREAD_BYTE,0);
//wait for a while controller to proceed data
Sleep(1);
//wait while READ for 1-st motor changes to 0
while ((QAPIExtSpecial(handle,JOB_IOREAD_BYTE,3)&0x02)!=0);
//read switch status
status=(unsigned char) QAPIExtSpecial(handle,JOB_IOREAD_BYTE,0);
QAPIExtCloseCard(handle);
```

5.3.4 Closing communication channel with 8SMCC PCI1/PCI3

`QAPIExtCloseCard` function should be used at the end of communication.

6 8SMCC PCI1/PCI3 library

6.1 General information

This library provides task- orientated functions specially designed to control 8SMCC PCI1/PCI3. Library is delivered as set of five files located in C:\Program Files\8SMMC\Library folder by default:

8SMCC2.dll	the library itself,
8SMCC2.h	C include file with definitions of functions,
8SMCC2.lib	library file (Microsoft library format) used to link user's application to 8SMCC2.dll,
Debug\8SMCC2d.dll	debug version of library with some debug output,
Debug\8SMCC2d.lib	debug version of library file (Microsoft library format).

Note: Library doesn't not support multithreading by design. If multiple threads are going to use 8SMCC2.dll library, additional synchronization objects must be used!

6.2 8SMCC PCI1/PCI3 functions

16 functions exported from library can be divided into three groups.
Initialization functions: `SMCC_InitPCILibrary()`, `SMCC_OpenDevice`, `SMCC_CloseDevice`.
Functions that can be called only when motor is still: `SMCC_SetAcceleration`, `SMCC_SetSpeed`, `SMCC_SetDivision`, `SMCC_SetDirection`, `SMCC_SetSwitchConfig`, `SMCC_SetSynchronization`, `SMCC_Move`, `SMCC_GetSwitchStatus`, `SMCC_SetMotorStatus`. These functions fail if motor moves.
Functions that can be called only when motor is moving: `SMCC_Stop`, `SMCC_GetPositionCheckStatus`, `SMCC_GetRotationCheckStatus`.
Descriptions of all functions are presented in this chapter.

6.2.1 Function SMCC_InitPCILibrary

```
int SMCC_InitPCILibrary()
```

It initializes library and data structures. This function must be called before any other function from this library.
Input parameters: None.
Output value: 0- if failed, 1- if succeeded.

6.2.2 Function SMCC_OpenDevice

```
unsigned long SMCC_OpenDevice(int card_type, int device_no, int motor_no)
```

Returns handle to specified motor on specified controller card. This function must be called before any function of the groups SMCC_Set… and SMCC_Get… is used. A motor opened with this function must be closed with SMCC_CloseDevice.
Input parameters:

`card_type`	Type of device (SMCC_DEVICE_PCI or SMCC_DEVICE_COM). Only SMCC_DEVICE_PCI is supported in this version of library.
`device_no`	Controller card number (0- for the first card, 1- for the second etc.).
`motor_no`	Motor number on selected controller card (value from 1 to 3).

Output value: 0 if device can't be opened or devices handle (value from 1 to 16).

6.2.3 Function SMCC_CloseDevice

```
void SMCC_CloseDevice(unsigned long handle)
```

Closes handle to motor opened by SMCC_OpenDevice.
Input parameters:

`handle`	Handle to motor returned by `SMCC_OpenDevice`.

Output value: None.

6.2.4 Function SMCC_SetAcceleration

```
int SMCC_SetAcceleration(unsigned long handle, unsigned char acc)
```

Sets acceleration parameter for specified motor.
Input parameters:

`handle`	Handle to motor returned by `SMCC_OpenDevice`.
`acc`	Acceleration value (from 1 to 119).

Output value: 0 if operation failed, 1- if succeeded.

6.2.5 Function SMCC_SetSpeed

```
int SMCC_SetSpeed(unsigned long handle, unsigned char speed)
```

Sets speed parameter for specified motor.
Input parameters:

`handle`	Handle to motor returned by `SMCC_OpenDevice`.
`speed`	Speed value (from 1 to 220).

Output value: 0 if operation failed, 1- if succeeded.

6.2.6 Function SMCC_SetSteps

```
int SMCC_SetSteps(unsigned long handle, unsigned long steps)
```

Sets amount of steps to move.
Input parameters:

`handle`	Handle to motor returned by `SMCC_OpenDevice`.
`steps`	Amount of steps to move (from 1 to 16581375).

Output value: 0 if operation failed, 1- if succeeded.

6.2.7 Function SMCC_SetDivision

```
int SMCC_SetDivision( unsigned long handle, unsigned char division)
```

Sets step division factor parameter for specified motor.
Input parameters:

`handle`	Handle to motor returned by `SMCC_OpenDevice`.

`division`	Division factor: 0- step is not divided, 1- step divided by 2, 2- step divided by 4, 3- step divided by 8.

Output value: 0 if operation failed, 1- if succeeded.

6.2.8 Function SMCC_SetDirection

```
int SMCC_SetDirection(unsigned long handle, unsigned char direction)
```

Sets direction of specified motor's movement.
Input parameters:

`handle`	Handle to motor returned by `SMCC_OpenDevice`.
`direction`	0- direction forward, 1- direction backward.

Output value: 0 if operation failed, 1- if succeeded.

6.2.9 Function SMCC_SetSwitchConfig

```
int SMCC_SetSwitchConfig(unsigned long handle, unsigned char config)
```

Sets mode of end switches.
Input parameters:

`handle`	Handle to motor returned by `SMCC_OpenDevice`.
`config`	Configuration bits for determining reaction to end switches (from 0 to 31). Binary format: 000zyyxx. xx= 00- ignore end switch 1, xx= 01- "emergency stop" mode for switch 1, xx= 11- "check position" mode for switch 1, yy= 00- ignore end switch 2, yy= 01- "emergency stop" mode for switch 2, yy= 11- "check position" mode for switch 2. z=0- rotational switch is ignored, z=1- controller counts steps between press of rotational switch and transfer this information to computer. See chapter 4.7 for details about switch modes.

Output value: 0 if operation failed, 1- if succeeded.

6.2.10 Function SMCC_SetSynchronization

```
int SMCC_SetSynchronization(unsigned long handle, unsigned char sync, unsigned long start=0, unsigned long period=0)
```

Sets mode of specified motor movement synchronization and mode of generation of sync pulses.
Input parameters:

`handle`	Handle to motor returned by `SMCC_OpenDevice`.
`sync`	Synchronization mode (from 0 to 7). Binary format 0000zyx. If bit x is set to 1, motor waits for external sync pulse before starting of movement.

If bit y is set to 1, motor generates sync pulse when movement is finished.
If bit 3 is set, motor generates sync pulses every `period` steps starting from `start`.

`start` — Amount of steps to wait before generation of the first sync pulse when periodic generation of sync pulses is turned on. Ignored otherwise.

`period` — Period of steps to generate sync pulses when periodic generation of sync pulses is turned on. Ignored otherwise.

Output value: 0 if operation failed, 1- if succeeded.

6.2.11 Function SMCC_Move

```
int SMCC_Move(unsigned long handle, int wait)
```

Starts moving motor accordingly earlier set parameters: amount of steps, direction, acceleration, speed, step division, synchronization mode and end-switch mode.
Input parameters s:

`handle` — Handle to motor returned by `SMCC_OpenDevice`.

`wait` — Wait flag. If `wait`=0, function returns immediately, otherwise function returns when motor stops.

Output value: 0 if operation failed, 1- if succeeded.
Typically function `SMCC_SetMotorStatus` must be executed before the first command `SMCC_Move`, because controller card by default is in current- off mode.

6.2.12 Function SMCC_Stop

```
int SMCC_Stop(unsigned long handle, unsigned long* steps)
```

Starts to slow down motor accordingly speed and acceleration parameters while it stops. If motor doesn't move function fails.
Input parameters:

`handle` — Handle to motor returned by `SMCC_OpenDevice`.

`steps` — Amount of steps made from start of movement. This value can be used to calculate current position of the motor.

Output value: 0 if operation failed, 1- if succeeded.

6.2.13 Function SMCC_GetSwitchStatus

```
int SMCC_GetSwitchStatus(unsigned long handle, unsigned char* status)
```

Gets status of end switches.
Input parameters:

`handle` — Handle to motor returned by `SMCC_OpenDevice`.

`status` — Status of switches. 0 bit is set if left switch is pressed and 1-st bit is set if right switch is pressed.

Output value: 0 if operation failed, 1- if succeeded.

6.2.14 Function SMCC_GetPositionCheckStatus

```
int SMCC_GetPositionCheckStatus(unsigned long handle, unsigned char* status, unsigned long timeout=0)
```

Gets status of end switches in “Check position” mode. This function can be called only if “check position” mode is turned on (see `SMCC_SetSwitchConfig`) and motor is moving (see `SMCC_Move`).
Input parameters:

`handle`	Handle to motor returned by `SMCC_OpenDevice`.
`status`	Information about switch pressed in “check position” mode. 0 bit is set to 1 if left switch was pressed and 1-st bit is set if right switch was pressed.
`timeout`	Timeout value in milliseconds. Function fails if response from controller is not received after `timeout` milliseconds.

Output value: 0 if operation failed or was timeout, 1- if succeeded.

6.2.15 Function SMCC_GetRotationCheckStatus

```
int SMCC_GetRotationCheckStatus(unsigned long handle, unsigned long* steps, unsigned long timeout=0)
```

Gets amount of steps between two rotational switch presses in “rotational switch” mode. This function can be called only if “rotational switch” mode is turned on (see `SMCC_SetSwitchConfig`) and motor is moving (see `SMCC_Move`).
Input parameters:

`handle`	Handle to motor returned by `SMCC_OpenDevice`.
`steps`	Amount of steps counted between two last presses of rotational switch.
`timeout`	Timeout value in milliseconds. Function fails if response from controller is not received after `timeout` milliseconds.

Output value: 0 if operation failed or was timeout, 1- if succeeded.

6.2.16 Function SMCC_SetMotorStatus

```
int SMCC_SetMotorStatus(unsigned long handle, unsigned char status)
```

Turns current on and off for specified motor.
Input parameters:

`handle`	Handle to motor returned by `SMCC_OpenDevice`.
`speed`	Motor status: 0- turn current off, 1- turn current on.

Output value: 0 if operation failed, 1- if succeeded.
Typically this function must be executed before issuing the first `SMCC_Move` command, because controller card by default is in current- off mode.

6.3 Working with 8SMCC PCI1/PCI3 card over 8SMCC2 functions

There is simple C++ program test.cpp provided together with library to demonstrate how two use 8SMCC2.dll functions. `SMCC_InitPCILibrary` must be called before any other functions from 8SMCC2.dll. The second step is to get handle for communication with function `SMCC_OpenDevice`. After that all other functions can be called. `SMCC_CloseDevice` must be used at the end of program. A shorter version of test.cpp is listed here to demonstrate how to move motor 1 forward by 100 steps:

```
#include "stdio.h"
```

```
#include "8smcc2.h"

int main(int argc, char* argv[])
{
    if (SMCC_InitPCILibrary()==1)
    {
            printf("Library inicialization: Ok\n");
            // Get handle for motor 1 on device 0
            unsigned long handle=SMCC_OpenDevice(SMCC_DEVICE_PCI, 0, 1);
            if (handle!=0)
            {
                    SMCC_SetAcceleration(handle, 20);
                    SMCC_SetSpeed(handle, 150);
                    SMCC_SetDivision(handle, 0);
                    SMCC_SetSynchronization(handle, 0);
                    SMCC_SetDirection(handle, 0);
                    SMCC_SetSwitchConfig(handle, 5);
                    SMCC_SetSteps(handle,100);
                    SMCC_SetMotorStatus(handle,1);
                    SMCC_Move(handle, 1);
                    SMCC_SetMotorStatus(handle,0);
                    SMCC_CloseDevice(handle);
            }
            else printf("OpenDevice: Failed\n");
    }
    else printf("Library inicialization: Failed\n");

    printf("Press Enter to quit\n");
    getchar();

    return 0;
}
```

6.4 Translator.exe application

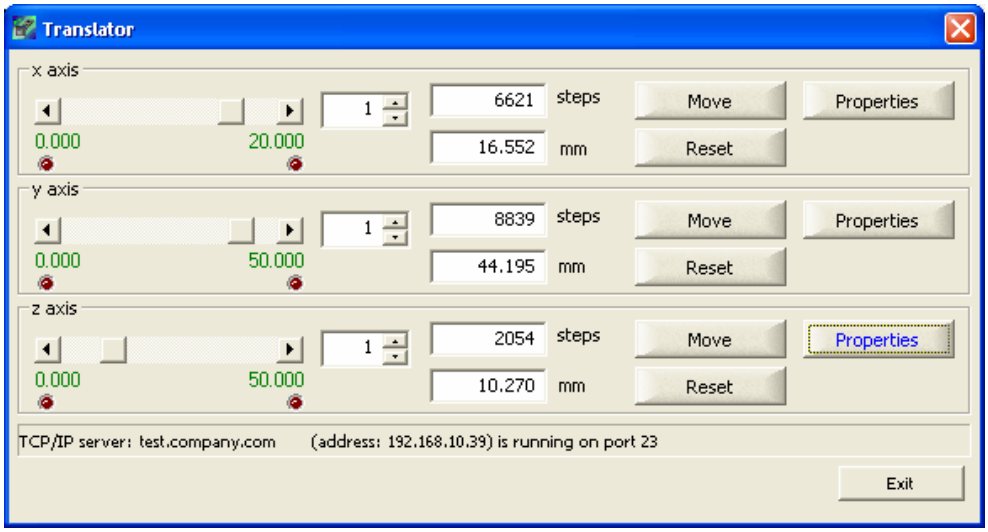
Another demonstrational application provided together with 8SMCC PCI1/PCI3 library is Translator.exe. This program can be used as simple application to control up to three translator stages. It provides simple and intuitive graphical user interface. Additionally simple TCP/IP network server is integrated to Translator. It allows controlling translator stages over TCP/IP local or worldwide network using such simple application as telnet.exe.

6.4.1 Using Translator.exe

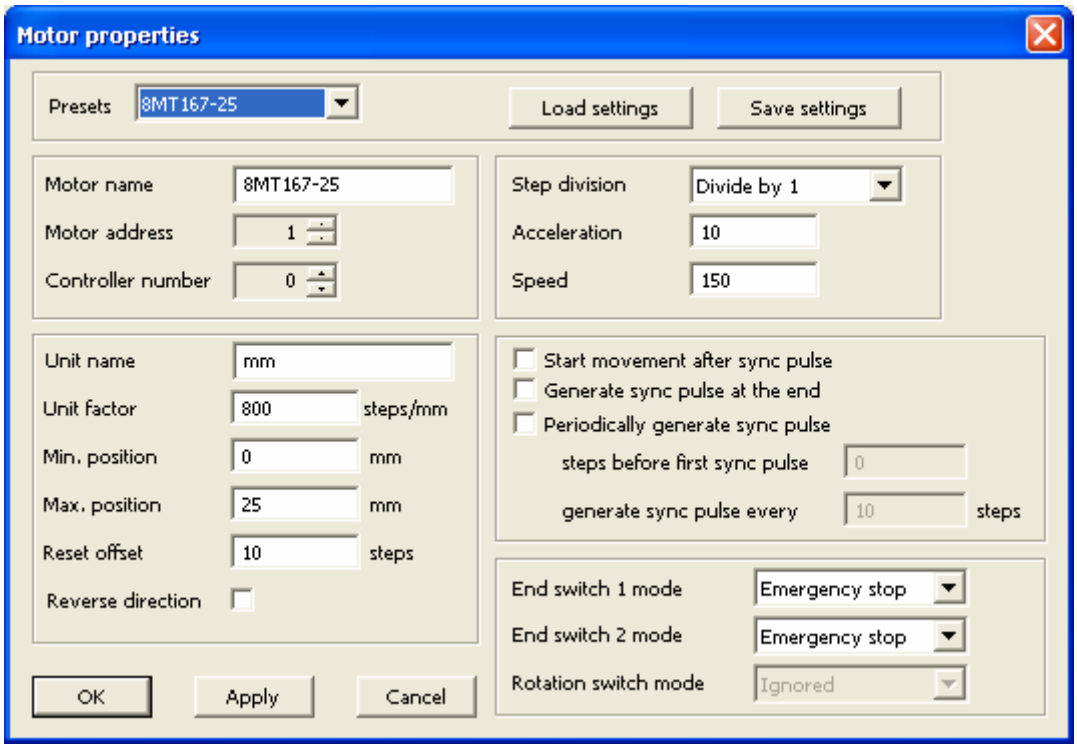
Main window of program contains three identical independent sections for every translator stage. Position of motor is presented in steps and user-defined units. There are several ways to move motor.

- Motor can be moved by mean of using scrollbar. Slider of control bar can be dragged to required position and released there. Motor will move to pointing position. Two edit boxes displaying current position of motor will show new position while dragging slider. Pressing on the arrow buttons on the left or right side of scrollbar will move motor backward or forward by amount of steps displayed in edit box on the right side of scrollbar. Pressing on the scrollbar on the left or right from slider will move motor backward or forward by amount of steps displayed in edit box on the right side of scrollbar multiplied by 10.
- Another way to move motor is directly to enter a new position of motor in steps or user-defined units to corresponding edit box and press “Move” button. Exclamation mark appears when new position is entered. It informs that position in edit boxes doesn’t correspond to current position of motor. Exclamation mark disappears when motor reaches a new position. Button “Move” is changed to “Stop”

while motor moves. Motor can be stopped at any moment by mean of pressing this button. Current position where a motor stops is displayed in position edit boxes.



Green numbers bellow scrollbar show range of translator stage in user-defined units. Red led on the left and right sides bellow scrollbar indicates state of end-switches. If led is turned on- end-switch is pressed. Information about end-switches is renewed every half of second.
Pressing of "Reset" button moves stage backward while the end-switch is pressed and after that sets position of stage to "Min position" by mean of moving forward by "Reset offset" steps.



Properties of motor can be changed by pressing button "Properties" and editing parameters in a properties window. All properties of motor are listed in table below.

Parameter	Description
Presets	Predefined parameters presets for STANDA products
Load settings	Loads settings for motor from file.
Save settings	Saves settings of motor to file.

Controller number	Number of 8SMCC PCI1/PCI3 card (0 for the first card, 1- for the second, etc.).
Motor address	Number of motor (1,2 or 3).
Motor name	Translator stage name displayed in the top left corner of stage panel.
Unit name	Name of user-defined units (for example: mm) displayed near the position in user-defined units edit box.
Unit factor	Proportional factor showing how many steps (divided or not) correspond to one user-defined unit (for example Factor=200 steps/mm).
Min. position	Minimal position of translator stage in user-defined units.
Max. position	Maximal position of translator stage in user-defined units.
Reset offset	Amount of steps moved forward from the end-switch to position corresponding to Min position in user-defined units
Reverse direction	Flag used to swap forward and backward direction of motor movement.
Acceleration	Acceleration parameter of motor.
Speed	Speed parameter of motor.
End switch 1 mode	Left end-switch mode (“Ignored” or “Emergency stop”).
End switch 2 mode	Right end-switch mode (“Ignored” or “Emergency stop”).
Start movement after sync pulse	Flag indicates if the sync pulse must be received before starting moving.
Generate sync pulse at the end	Flag indicates if the sync pulse must be generated at the end of movement.
Periodically generate sync pulse	Flag indicates that sync pulse must be generated every *t* steps starting from *b*.
Steps before first sync pulse	Parameter *b*, indicates that sync pulses must be generated starting from *b* steps.
Generate sync pulse every… steps	Parameter *t*, indicates period in steps how often sync pulses must be generated.
Step division	Motor’s step division parameter (“No division”, “Divide by 2”, “Divide by 4”, “Divide by 8”).

6.4.2 Using Translator.exe as network server

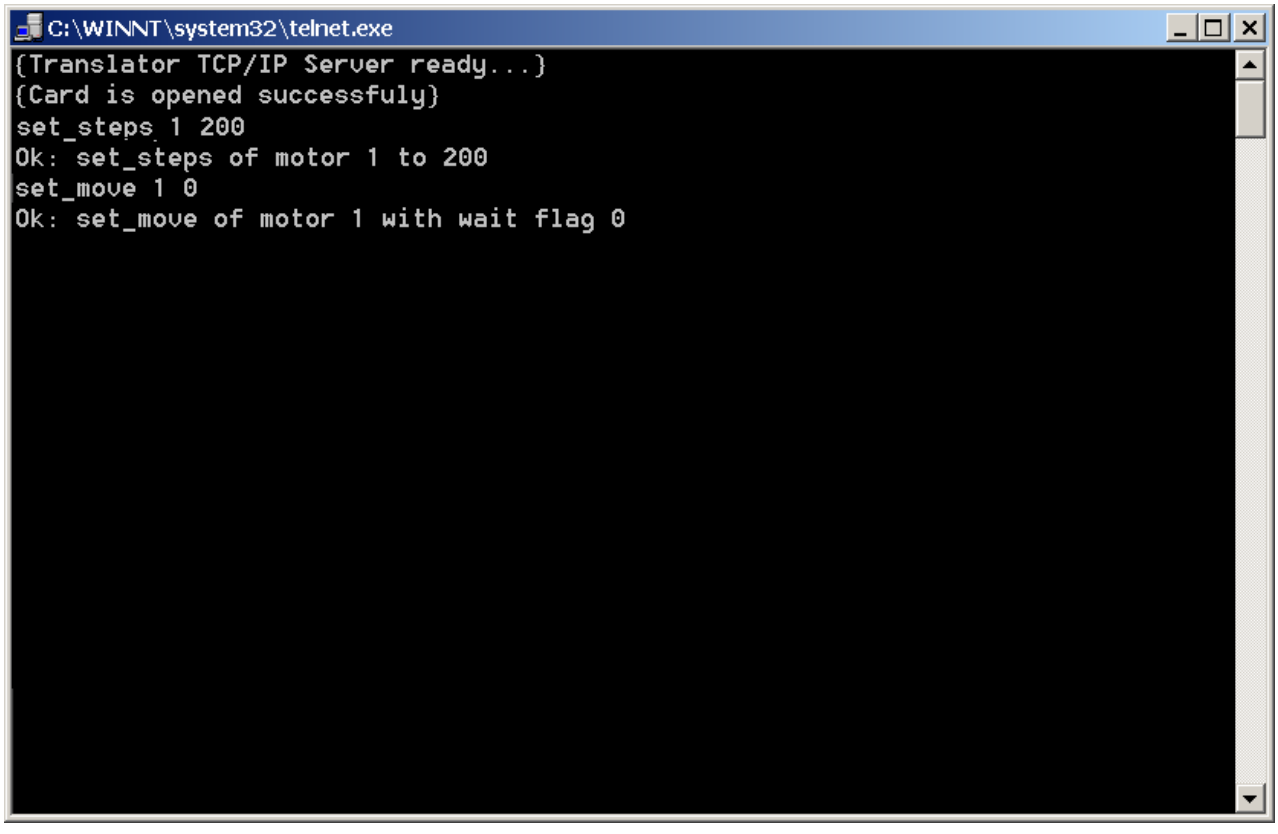
Translator.exe has integrated network server and is capable to accept commands sent by network. TCP port 23 is used for communication, those simple terminal application as telnet can be used to control Translator.exe remotely. When Translator.exe starts it displays information about server status: “TCP/IP server*: computer_name* (address *computer_IP_address*) is running on port 23”. When client connects to server, Translator changes window name from “Translator” to “Translator- (client connected)” and sends to client string “{Translator TCP/IP Server ready...}”. Status of server application is transferred as string “{Card is opened successfully}” or “{Card is not found}”. All commands are case insensitive. Commands accepted by network server are listed in table bellow.

Command	Possible answers by server	Description
quit	{Bye}	Quits communication session.
reset *no*	Ok: reset of motor *no* Failed: reset of motor *no* (incorrect motor number)	Performs reset of motor *no*.

set_acceleration *no acc*	Ok: set_acceleration of motor *no* to *acc* Failed: set_acceleration of motor *no* to *acc* Failed: set_acceleration of motor *no* to *acc* (incorrect motor number)	Sets a new acceleration value *acc* for motor *no*.
set_speed *no sp*	Ok: set_speed of motor *no* to *sp* Failed: set_speed of motor *no* to *sp* Failed: set_speed of motor *no* to *sp* (incorrect motor number)	Sets a new speed value *sp* for motor *no*.
set_steps *no steps*	Ok: set_steps of motor *no* to *steps* Failed: set_steps of motor *no* to *steps* Failed: set_steps of motor *no* to *steps* (incorrect motor number)	Sets a new steps value *steps* for motor *no*.
set_division *no div*	Ok: set_division of motor *no* to *div* Failed: set_division of motor *no* to *div* Failed: set_division of motor *no* to *div* (incorrect motor number)	Sets a new division value *div* for motor *no*.
set_synchronization *no sync start period*	Ok: set_synchronization of motor *no* to *sync start period* Failed: set_synchronization of motor *no* to *syn start period* Failed: set_synchronization of motor *no* to *sync start period* (incorrect motor number)	Sets a new set synchronization parameters *sync start period* (synchronization, steps to start, period in steps) for motor *no*.
set_direction *no dir*	Ok: set_direction of motor *no* to *dir* Failed: set_direction of motor *no* to *dir* Failed: set_direction of motor *no* to *dir* (incorrect motor number)	Sets a new direction value *dir* for motor *no*.
set_switchconfig *no sw*	Ok: set_switchconfig of motor *no* to *sw* Failed: set_switchconfig of motor *no* to *sw* Failed: set_switchconfig of motor *no* to *sw* (incorrect motor number)	Sets a new switch configuration value *sw* for motor *no*.
set_move *no wait*	Ok: set_move of motor *no* with wait flag *wait* Failed: set_move of motor *no* with wait flag *wait* Failed: set_move of motor *no* with wait flag *wait* (incorrect motor number)	Starts movement of motor *no* with flag *wait*.
set_stop *no*	Ok: set_stop of motor *no*. *steps* steps made Failed: set_stop of motor *no* Failed: set_stop of motor *no* (incorrect motor number)	Stops motor *no* and returns amount of steps made.
Set_motorstatus *no stat*	Ok: set_motorstatus of motor *no* to *stat* Failed: set_motorstatus of motor *no* to *stat* Failed: set_motorstatus of motor *no* to *stat* (incorrect motor number)	Sets motor current on (*stat*=1) or off (*stat*=0).
set_motorname *no name*	Ok: set_motorname of motor *no* to *name* Failed: set_motorname of motor *no* to *name* Failed: set_motorname of motor *no* to *name* (incorrect motor number)	Changes name of motor *no* to *name*. (*name*- string of characters).
set_unitname *no units*	Ok: set_unitname of motor *no* to *units* Failed: set_unitname of motor *no* to *units* Failed: set_unitname of motor *no* to *units* (incorrect motor number)	Changes name of user-defined units of motor *no* to *units*. (*units* - string of characters).

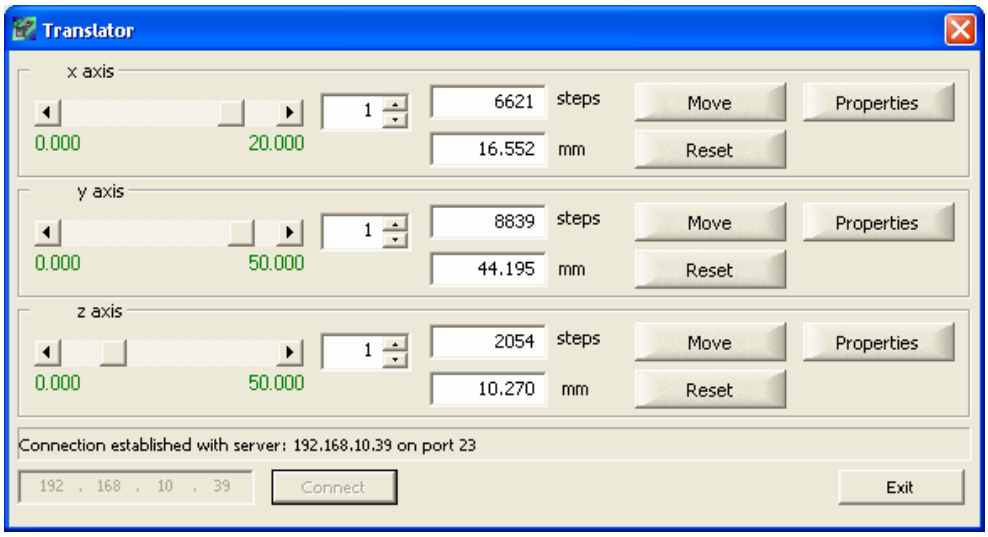
set_factor *no factor*	Ok: set_factor of motor *no* to *factor* Failed: set_factor of motor *no* to *factor* Failed: set_factor of motor *no* to *factor* (incorrect motor number)	Changes factor of user-defined units of motor *no* to *factor*. (*factor*- floating number).
set_minpos *no pos*	Ok: set_minpos of motor *no* to *pos* Failed: set_minpos of motor *no* to *pos* Failed: set_minpos of motor *no* to *pos* (incorrect motor number)	Changes min position of motor *no* to *pos*. (*pos*- floating number).
set_maxpos *no pos*	Ok: set_maxpos of motor *no* to *pos* Failed: set_maxpos of motor *no* to *pos* Failed: set_maxpos of motor *no* to *pos* (incorrect motor number)	Changes max position of motor *no* to *pos*. (*pos*- floating number).
set_resetoffset *no offset*	Ok: set_resetoffset of motor *no* to *offset* Failed: set_resetoffset of motor *no* to *offset* Failed: set_resetoffset of motor *no* to *offset* (incorrect motor number)	Changes reset offset value of motor *no* to *offset*. (*offset*- integer number).
set_reversedirection *no flag*	Ok: set_reversedirection of motor *no* to *flag* Failed: set_reversedirection of motor *no* to *flag* Failed: set_reversedirection of motor *no* to *flag* (incorrect motor number)	Changes value of reverse direction parameter of motor *no* to *flag*. (*flag*- 0 or 1).
set_newposition *no pos*	Ok: set_newposition of motor *no* to *pos* Failed: set_newposition of motor *no* to *pos* Failed: set_newposition of motor *no* to *pos* (incorrect motor number)	Moves motor *no* to a new position *pos*. (*pos*- position in steps).
update_panels	-	Updates all three panels on server window

A simple telnet session is displayed as example in picture below.



6.4.3 Using Translator.exe as network client

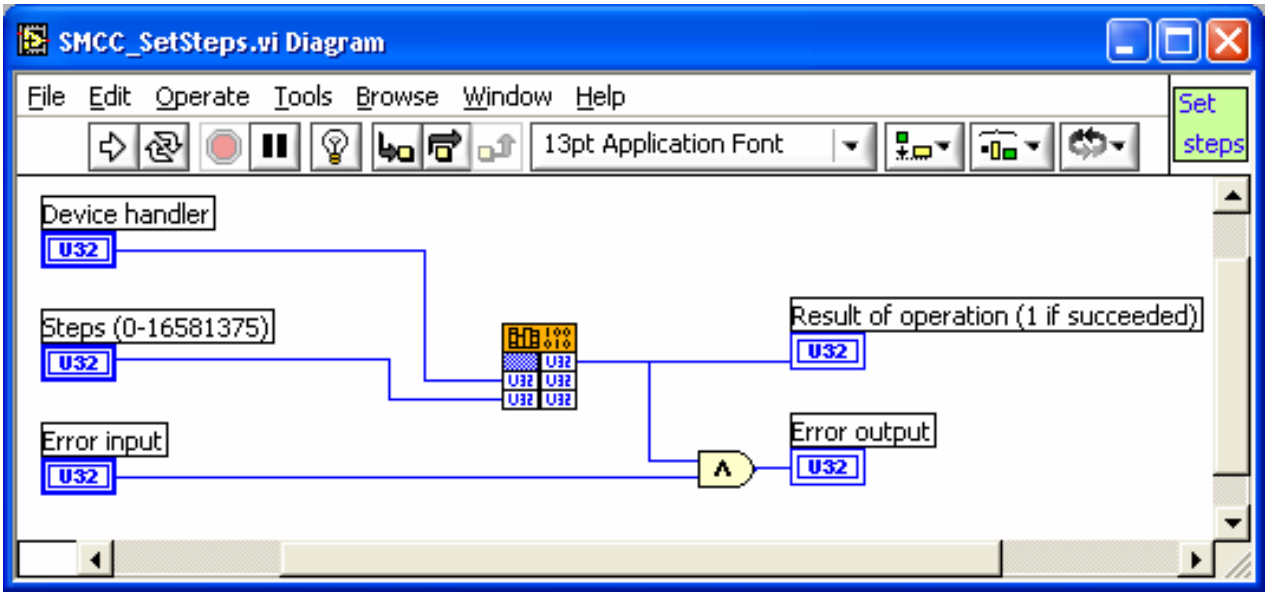
Translator.exe can be used as simple network client as well. To use Translator as network client run program with command option "/c" (or use shortcut "Start|Programs|8SMCC PCI1/PCI3|Translator network client" created by library installation). Additional IP address box and button "Connect" are displayed when Translator starts in client mode (see picture bellow). One more difference is that there are no more red led indicating state of end-switches.



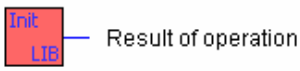
First operation to perform when Translator is started as client is to connect to server. Enter IP address of computer with 8SMCC PCI1/PCI3 card installed and Translator started in server mode (without command switch /c). If connection is established client program will display information "Connection established with server *server_IP_address* on port23". From this moment move, reset and change of properties operations can be executed in a manner similar to the controlling translators from the local computer. Note that Translator network client is not full client application and is intended only to demonstrate capabilities of controlling Translator server over the network.

7 Using 8SMCC2.dll over LabVIEW

Additional LabVIEW virtual instrument library 8SMCC2.llb is located in drive:\Program Files\8SMCC\LabVIEW by default. It contains simple wrappers for all functions exported from 8SMCC2.dll. Functions inside 8SMCC2.llb have the same names as functions in dynamic link library. Structure of every function can be viewed and modified if necessary. Every virtual instrument gets input parameters and passes them to the function in dynamic link library using "Call function" instrument implemented in LabVIEW. Return value of this function is returned as result of operation. Additionally error input and error output are added to meet LabVIEW programming paradigm. All input parameters of functions are the same as described in chapter 6.2. Result of operation is 1 if function succeeds and 0 if function fails. As example, structure of SMCC_SetSteps virtual instrument is presented here.

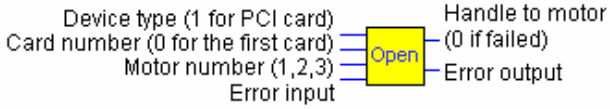


7.1 Virtual instrument "SMCC_InitPCILibrary"



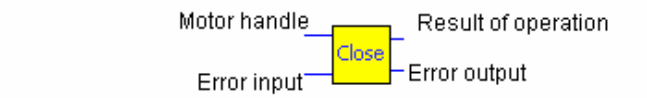
Please refer to chapter 6.2.1 for description of this function.

7.2 Virtual instrument "SMCC_OpenDevice"



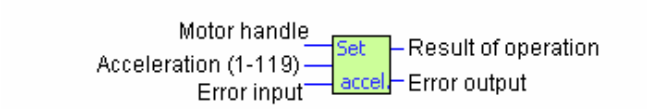
Please refer to chapter 6.2.2 for description of this function.

7.3 Virtual instrument "SMCC_CloseDevice"



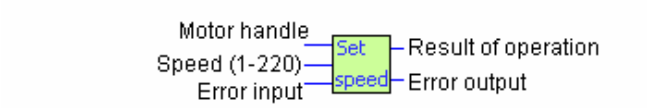
Please refer to chapter 6.2.3 for description of this function.

7.4 Virtual instrument “SMCC_SetAcceleration”



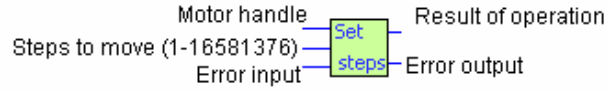
Please refer to chapter 6.2.4 for description of this function.

7.5 Virtual instrument “SMCC_SetSpeed”



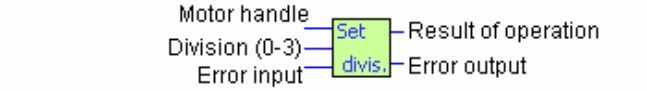
Please refer to chapter 6.2.5 for description of this function.

7.6 Virtual instrument “SMCC_SetSteps”



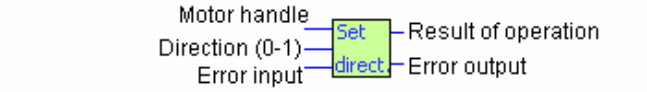
Please refer to chapter 6.2.6 for description of this function.

7.7 Virtual instrument “SMCC_SetDivision”



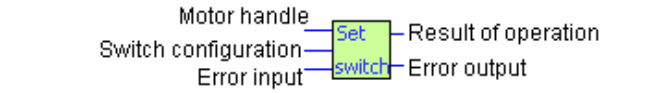
Please refer to chapter 6.2.7 for description of this function.

7.8 Virtual instrument “SMCC_SetDirection”



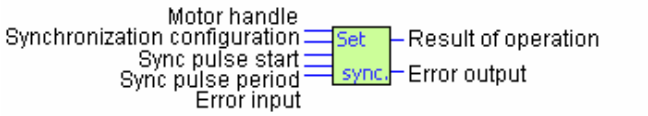
Please refer to chapter 6.2.8 for description of this function.

7.9 Virtual instrument “SMCC_SetSwitchConfig”



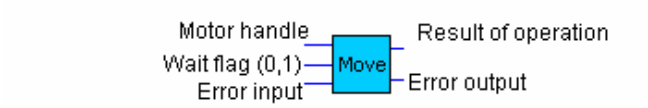
Please refer to chapter 6.2.9 for description of this function.

7.10 Virtual instrument “SMCC_SetSynchronization”



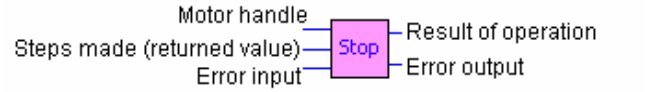
Please refer to chapter 6.2.10 for description of this function.

7.11 Virtual instrument “SMCC_Move”



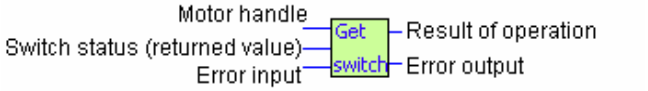
Please refer to chapter 6.2.11 for description of this function.

7.12 Virtual instrument “SMCC_Stop”



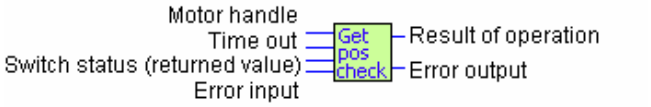
Please refer to chapter 6.2.12 for description of this function.

7.13 Virtual instrument “SMCC_GetSwitchStatus”



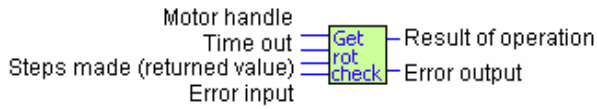
Please refer to chapter 6.2.13 for description of this function.

7.14 Virtual instrument “SMCC_GetPositionCheckStatus”



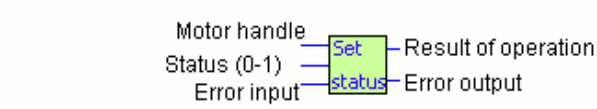
Please refer to chapter 6.2.14 for description of this function.

7.15 Virtual instrument “SMCC_GetRotationCheckStatus”



Please refer to chapter 6.2.15 for description of this function.

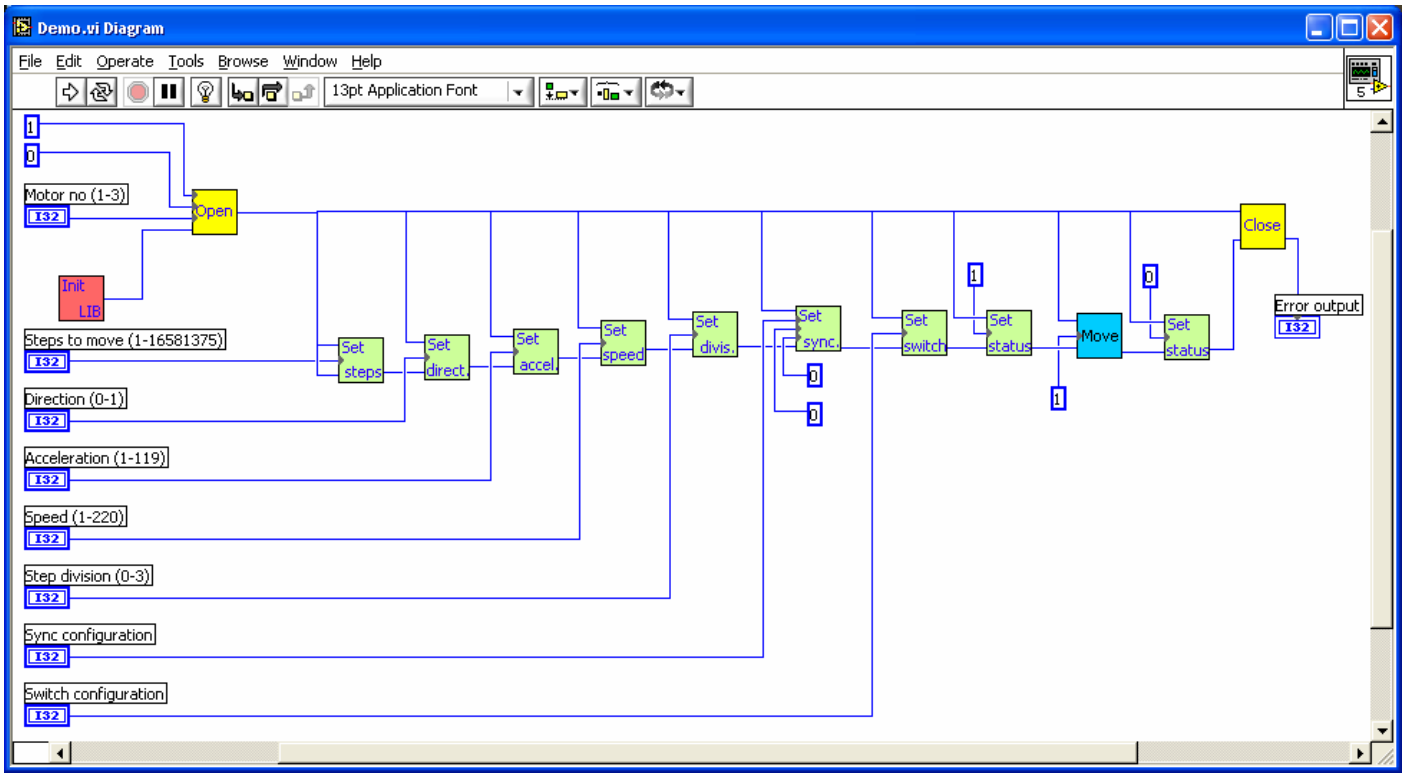
7.16 Virtual instrument “SMCC_SetMotorStatus”



Please refer to chapter 6.2.16 for description of this function.

7.17 Using LabVIEW virtual instrument library 8SMCC2.llb

There is simple example program demo.vi located in the same directory as 8SMCC2.llb. It demonstrates how to use virtual instruments to control 8SMCC PCI1/PCI3 card. Dynamic link library must be initialized at the beginning with function SMCC_InitPCILibrary. After that handle to motor must be obtained with function SMCC_OpenDevice. When handle is received all motor parameters can be transferred to controller card. The order of transfer of parameters is not important. At the end command SMCC_Move must be issued to move motor and handle must be closed with command SMCC_CloseDevice. Structure of demo.vi program is presented here.



8 Using 8SMCC2.dll over MatLAB

MatLAB can not use functions from standard dynamic link libraries directly those simple wrapper libraries for every function exported from 8SMCC2.dll are provided in folder drive:\Program Files\8SMCC\MatLAB by default. Add this path to MatLAB path list or copy these functions to MatLAB\work directory to use them. Use of functions is almost identical as described in chapter 6.2. All functions are described in table.

Function	Description
SMCC_InitPCILibrary	Initializes library and data structures. This function must be called before any other function. Returns 1 on success or 0 on failure.
SMCC_OpenDevice(*card_no*, *motor_no*)	Returns handle to specified motor *motor_no* on controller card *card_no*. This function must be called before any function of the groups SMCC_Set… and SMCC_Get… is used. A motor opened with this function must be closed with SMCC_CloseDevice. Returns 0 on failure.
SMCC_CloseDevice(*handle*)	Closes *handle* to motor opened by SMCC_OpenDevice. Always returns 1.
SMCC_SetAcceleration(*handle*, *acc*)	Sets acceleration parameter *acc* for motor specified by *handle*. Returns 1 on success or 0 on failure.
SMCC_SetSpeed(*handle*, *speed*)	Sets *speed* parameter for motor specified by *handle*. Returns 1 on success or 0 on failure.
SMCC_SetSteps(*handle*, *steps*)	Sets amount of *steps* to move for motor specified by *handle*. Returns 1 on success or 0 on failure.
SMCC_SetDivision(*handle*, *div*)	Sets step division factor *div* parameter for motor specified by *handle*. Returns 1 on success or 0 on failure.
SMCC_SetDirection(*handle*, *dir*)	Sets direction *dir* for movement of motor specified by *handle*. Returns 1 on success or 0 on failure.
SMCC_SetSwitchMode(*handle*, *mode*)	Sets *mode* of end switches for motor specified by *handle*. See chapter 6.2.9 for description of switch modes. Returns 1 on success or 0 on failure.
SMCC_SetSynchronization(*handle*, *sync*, *start*, *period*)	Sets mode of movement synchronization *sync* and mode of generation of sync pulses *start* and *period* for motor specified by *handle*. See chapter 6.2.10 for description of synchronization modes and parameters. Returns 1 on success or 0 on failure.
SMCC_Move(*handle*, *wait*)	Starts moving motor specified by *handle* accordingly earlier set parameters: amount of steps, direction, acceleration, speed, step division, synchronization mode and end-switch mode. If flag *wait* is set to 0, function returns immediately else function returns when movement is finished. Returns 1 on success or 0 on failure.
SMCC_Stop(*handle*)	Starts to slow down motor specified by *handle* accordingly speed and acceleration parameters while it stops. If motor doesn't move function fails. Function returns row of two values. First value is 1 on success and 0 on failure. The second value is amount of steps made after start of movement.
SMCC_GetSwitchStatus(*handle*)	Gets status of end switches for motor specified by *handle*. Function returns row of two values. First value is 1 on success and 0 on failure. The second value is status of end switches. See chapter 6.2.13 for description of status of end-switches.
SMCC_GetPositionCheckStatus(*handle*,	Gets status of end switches in "Check position" mode. This

timeout)	function can be called only if "check position" mode is turned on (see `SMCC_SetSwitchConfig`) and motor is moving. Function returns row of two values. First value is 1 on success and 0 on failure. The second value is status of end switches. See chapter 6.2.13 for description of status of end-switches and chapter 6.2.10 for description of synchronization modes.
SMCC_GetRotationCheckStatus(*handle*, *timeout*)	Gets amount of steps between two rotational switch presses in "rotational switch" mode. This function can be called only if "rotational switch" mode is turned on (see `SMCC_SetSwitchConfig`) and motor is moving. Function returns row of two values. First value is 1 on success and 0 on failure. The second value is amount of steps counted. See chapter 6.2.10 for description of synchronization modes.
SMCC_SetMotorStatus(*handle*, *status*)	Sets current on (status=1) or off (status=0) for motor specified by *handle*. Returns 1 on success or 0 on failure.

A simple example of MatLAB program is listed bellow.

```
SMCC_InitPCILibrary
handle=SMCC_OpenDevice(0,1)
SMCC_SetSteps(handle,200)
SMCC_SetMotorStatus(handle,1)
SMCC_Move(handle)
SMCC_SetMotorStatas(handle,0)
SMCC_CloseDevice(handle)
```