M-Box with PDC Motion Control

Implementation

Revision:	4.3
Status:	Draft
Repository:	/svn/acc/microIOC/microIOC-M-Box/
Project:	GSI-MotionControl
Folder:	Operations/Projects/GSI-MotionControl
Document ID:	CSL-DOC-11-65507
File:	DOC-microIOC-MBOX-PDC-TechnicalDocument.doc
Owner:	mlevicnik
Last modification:	July 18, 2013
Created:	February 25, 2013

Document History

Revision	Date	Changed/ reviewed	Section(s)	Modification
1.0	2007-7-12	jdedic/ gpajor	All	Created.
1.1	2007-7-16	jdedic	2.1.1 various 4	PLIM/MLIM and AXINT usage. modified Table 2, 5, 7, 12, and Figure 12 created/added
2.0	2007-10-25	jdedic	All	See below text for explanation
2.0.1	2007-11-02	jdedic	2.2.2 3.1	inserted minor changes
2.1	2007-11-08	jdedic	2.1.2 Table 16 all	modified **DevMov** register (0x0b) not used anymore inserted renamed: mEB → PDC
3.0	2011-01-27	mlevicnik	All	modified
3.1	2011-01-28	gjansa	All 3	Review Updated
3.2	2012-06-15	mlevicnik	2.1.2	**DevMov** description added
4.0	2013-03-15	mlevicnik	All	updated
4.1	2013-04-03	mlevicnik	All	Updated/changed information of PDC analog input module
4.2	2013-07-16	gjansa	3	Updated
4.3	2013-07-18	gjansa	3	Updated figure

Confidentiality

This document is classified as a **public document**. As such, it or parts thereof are openly accessible to anyone listed in the Audience section, either in electronic or in any other form.

Scope

This document covers the system specifications, requirements and implementation of a motion control system for GSI.

Audience

All Cosylab and GSI employees.

Typography

This document uses the following styles:

A box like this contains important information.

Warning!
A box like this provides information, which should not be disregarded!

Glossary of Terms

API	Application Program Interface
CPLD	Complex Programmable Logic Device
GUI	Graphical User Interface
HW	Hardware
NTC	Negative Temperature Coefficient resistor
PDC	Power Drive Case
PMAC	Programmable Multi-Axis Controller
SSI	Synchronous Serial Interface
SW	Software

References

[1] GSI Motion Control - System specifications, revision 0.7

[2] www.microioc.com/download/microIOC-PD-baseline.pdf

[3] DeltaTau PMAC2A PC/104: www.deltatau.com/Common/products/pc104.asp?node=id110&connectionstr=release

[4] Xilinx, CoolRunner XPLA3 series CPLD: www.xilinx.com/products/silicon_solutions/cplds/coolrunner_series/coolrunner_xpla3_cplds/index.htm

[5] Mean Well, SP-500-24 (24V, 20A, 480W): www.meanwell.com/search/SP-500/default.htm

[6] 8-channel Analog Input Module to RS-485, 16-bit differential: http://www.advantech.com/products/ADAM-4117/mod_9161079D-563C-428C-8DF4-AA29FFFA995D.aspx

[7] XOG008-01, Xtreme/104-Plus Opto Serial Communications Card
http://www.connecttech.com/sub/Products/PC104PlusProducts_Xtreme104PlusOpto.asp?l1=pc104plus&l2=xt104plusopto

[8] Lantronix Device Installer:

http://www.lantronix.com/device-networking/utilities-tools/device-installer.html

[9] Lantronix XPORT Pro:
http://www.lantronix.com/support/downloads/?p=XPORTPRO

[10] PDC schematics and wiring documentation

[11] G. Jansa, Stepper Motor Control – System Design, 1.0

[12] G. Gaspersic et al., Stepper Motor Control – Local control, 1.0

[13] V. Juvan et al., Stepper Motor Control – LCD, 1.0

[14] G. Jansa, Stepper Motor Control – Installation and Configuration, 1.0

Table of Contents

Figures

Tables

1. SYSTEM OVERVIEW

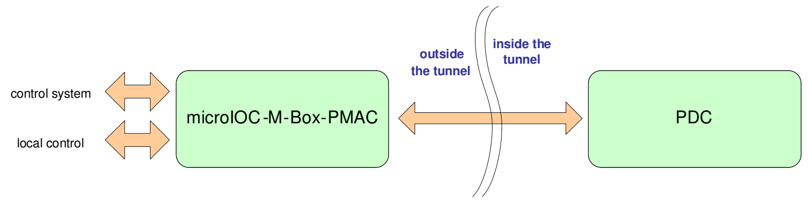


Figure 1: Control system description

This document describes the system for controlling up to 8 stepper motors. System requirements specifications are provided in [1]. Figure 1 gives a system overview of the system. The main components of the system are the microIOC M-Box-PMAC and the PDC (power drive case).

1.1. MICROIOC M-BOX-PMAC

The microIOC M-Box-PMAC shown in Figure 2 is a microIOC-based product, offering an advanced motion control functionality and control of general-purpose signals. microIOC is a very flexible software (SW) platform, which provides control of all modules and provides a hosting environment for the control-system application (product benefits and features are described in [2]). Motion control is implemented by a PMAC (programmable multi-axis controller) [3] that handles the motion control of up to eight axes. Additionally, a PMAC interface board was developed that provides voltage translation and optical isolation of signals (addressing large distances and possible noisy environments). This interface board is highly programmable (using a Xilinx CPLD [4]) by the means of mapping signals that pass through the board (signals can be inverted, logically combined, etc). The PMAC interface board also extends the available PMAC's per-axis input/output signals to provide control of additional functionality (such as brake control, interlocks, etc.). To provide communication with remote analog module (for position feedback) an RS-485 communication card with optically isolated outputs (addressing large distances and possible noisy environments) was added.

1.2. PDC

The PDC (Figure 3) contains the power supply [5] for 4 stepper drives (Vexta CSD 5828 N-T), an RS-485 analogue input module [6], and a simple PDC interface circuit (signal adaptation).

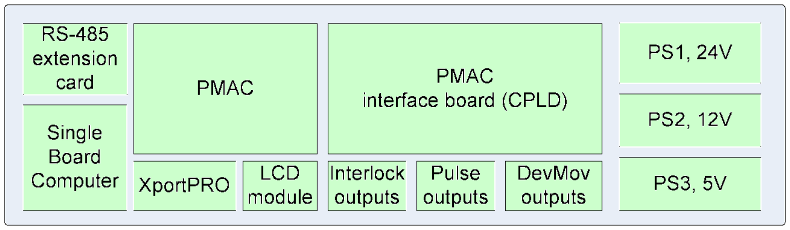


Figure 2: microIOC-M-Box-PMAC block diagram

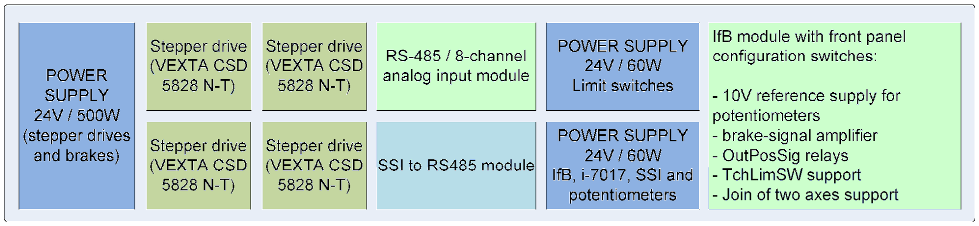

POWER SUPPLY 24V / 500W (stepper drives and brakes)
Stepper drive (VEXTA CSD 5828 N-T)
Stepper drive (VEXTA CSD 5828 N-T)
Stepper drive (VEXTA CSD 5828 N-T)
Stepper drive (VEXTA CSD 5828 N-T)
RS-485 / 8-channel analog input module
SSI to RS485 module
POWER SUPPLY 24V / 60W Limit switches
POWER SUPPLY 24V / 60W IfB, i-7017, SSI and potentiometers
IfB module with front panel configuration switches:
- 10V reference supply for potentiometers
- brake-signal amplifier
- OutPosSig relays
- TchLimSW support
- Join of two axes support

Figure 3: PDC block diagram

According to the specification requirements, the system has to be fully functional even with distances of 250 m between microIOC-M-Box-PMAC and PDC (see Figure 1). To achieve this, all the analog signals (i.e. potentiometer position feedback) and power signals (to stepper motors) are handled inside the PDC, closer to the controlled motors. Two types of communication exists between the two boxes; RS-485 (optically isolated at the microIOC side) and communication with the PMAC interface board (optically isolated, 24V signals, 10-20mA). This scheme provides a stable and noise-immune communication over the 250 m distance.

2. Signal descriptions

2.1. microIOC-M-Box-PMAC

The back panel connections for the microIOC-M-Box-PMAC are shown in Figure 4 and Figure 5.

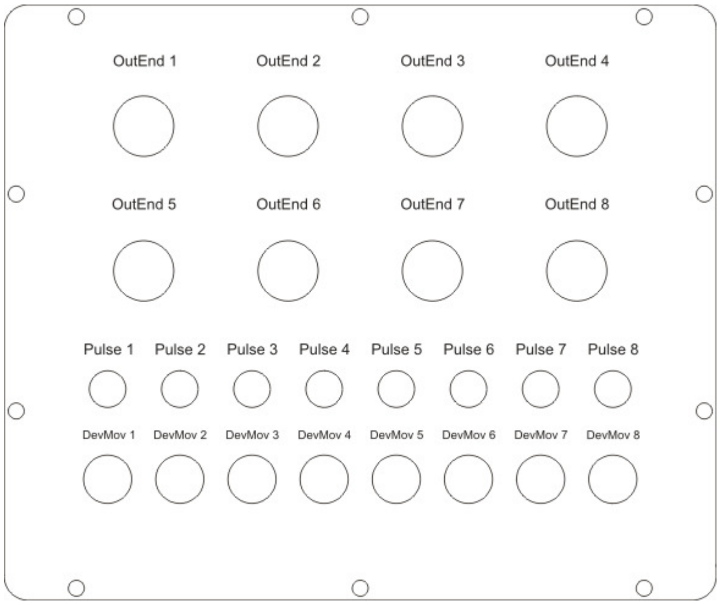


Figure 4: Back panel (right) connections of microIOC M-Box-PMAC

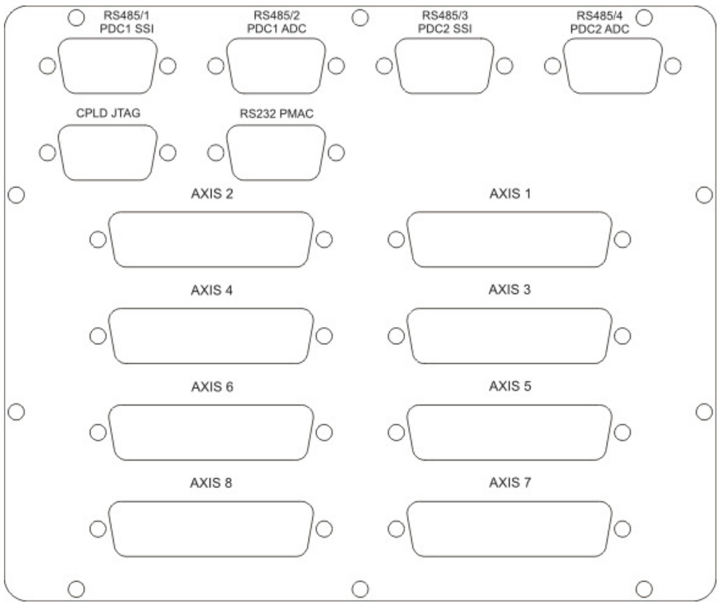


Figure 5: Back panel (left) connections of microIOC M-Box-PMAC

2.1.1. Axes inputs/outputs

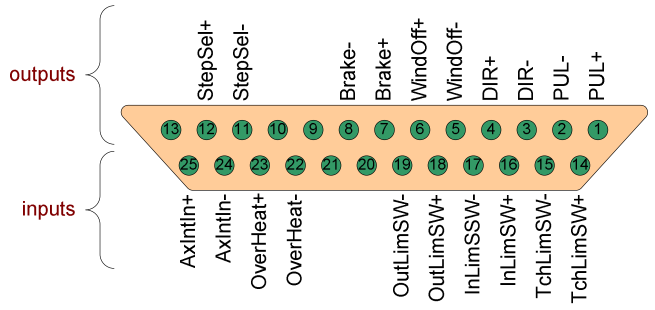


Figure 6: axis connector (DB-25F)

Each controlled axis has a dedicated DB-25 connector, so there are 8 connectors in total. Only low-power (24V, ~15mA) signals are transmitted to the PDC. The following signals are available per connector:

- **`PUL`**, **`DIR`**: signals to control the stepper drive. These are PMAC's signals and therefore handled by its motion control programs. If required, **`PUL`** signal can be conditionally gated by CPLD (e.g. to provide an additional safety level). Outputs of these signals are prepared for direct connection to optocoupler inputs of the stepper drive.
- **`WindOff`**, **`StepSel`**: signals to control the stepper drive. **`WindOff`** is handled automatically by PMAC's motion control programs. **`StepSel`** is controlled through a user accessible register; see Table 16 for a description of the CPLD registers. Outputs of these signals are prepared for direct connection to optocoupler inputs of the stepper drive.
- **`Brake`**: signal to control the brake amplifier circuitry within the PDC interface board. The signal is handled automatically by PMAC's motion control programs and is not intended to be controlled by higher level SW.
- **`TchLimSW`**, **`InLimSW`**, **`OutLimSW`**: input signals for detecting limit positions. PMAC's motion control programs provide responses to them. The CPLD also reads these inputs and can provide any actions regarding the states of these inputs. Signal **`InLimSW`** is used to detect the movement in the direction of the beam. Signal **`OutLimSW`** is used to detect the movement out of the beam. **`InLimSW`** and **`OutLimSW`** switches must be of the type that is normally-closed and floating. See Section 2.2.5 for connection details.
- **`OverHeat`**: input signal from stepper drive for detecting overheat condition. This signal is accessible through CPLD register, Table 16: CPLD registers' descriptions.
- **`AxIntIn`**: input signal from PDC (LEMO connector). Signal is routed to PMAC and if any of the input signals is not short circuited, PMAC prohibits any movement.

2.1.2. Device moving outputs

microIOC M-Box provides eight device moving output signals – one per axis. Two pin LEMO connectors are used (type EPL.0S.302.HLN). **`DevMov`** outputs are implemented by relays. These signals can be used, for example, to alert other devices about movement in progress and are directly controlled with the **WindOff** signal for the stepper driver. Implementation is fail-safe:

- axis is moving or device has no power – relay contacts are open (as if the cable is disconnected)
- axis is stopped – relay contacts are closed

State of relay output can be changed with jumpers. NO or NC contacts can be selected. Default settings of device moving outputs are NO, as seen on Figure 7.

Contact	Signal
1	**DevMov** (relay contact 1)
2	**DevMov** (relay contact 2)

Table 1: Device moving outputs pinout (applies to DevMov 1 – DevMov 8)

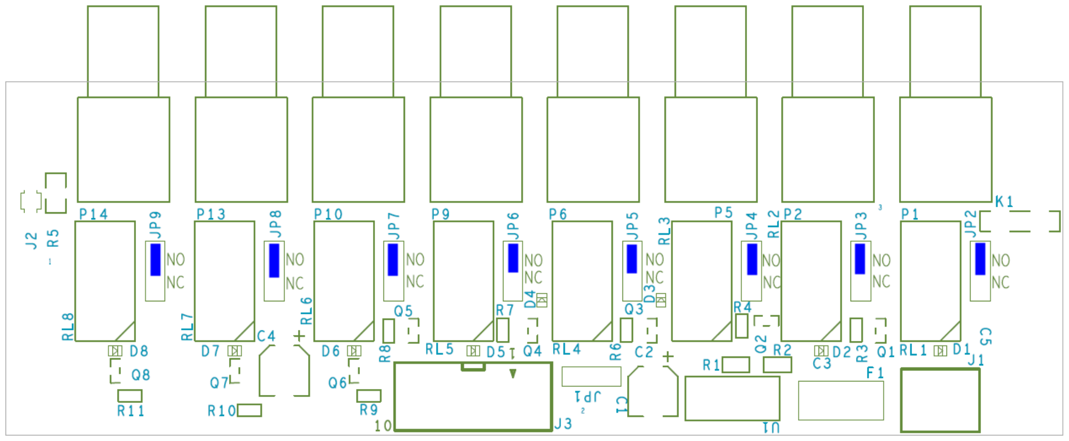


Figure 7: Device Moving output board

2.1.3. Pulse outputs

microIOC M-Box provides eight TTL pulse output signals – one per axis. One pin LEMO connectors are used (type EPL.00.250.NTN). The TTL pulse output signal is the same signal as used for driving stepper motor.

Contact	Signal
1	**Pulse**
shield	**GND**

Table 2: TTL pulse outputs pinout (applies to Pulse 1 – Pulse 8)

2.1.4. Interlock outputs

microIOC M-Box provides eight interlock output signals – one per axis. Five pole female M12 connectors are used (type CONEC M12 43-01199, A coding). Interlock outputs are implemented using relays.

- axis interlock or device has no power – relay contacts are open (as if the cable is disconnected)
- device is powered on, ready and in operation – relay contacts are closed

Type of outputs (NO or NC relay contacts) can be selected by using jumpers. Default settings can be seen on Figure 8. If inner limit is not needed to be present on output connector, in limit jumpers should be removed.

Contact	Signal
1	`Common (In and Out limit)`
2	`n.c.`
3	`Dry contact – Outer limit`
4	`n.c.`
5	`Dry contact – Inner limit`

Table 3: Interlock outputs pinout (applies to OutEnd 1 – OutEnd 8)

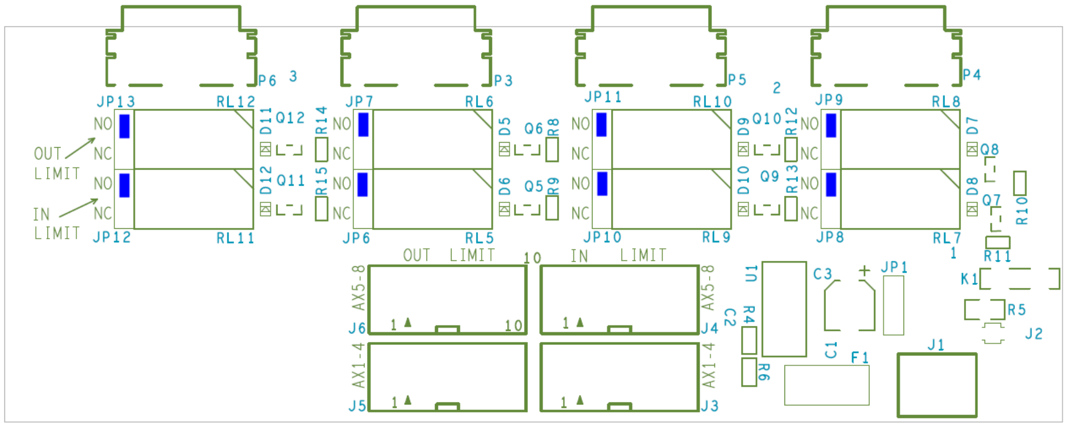
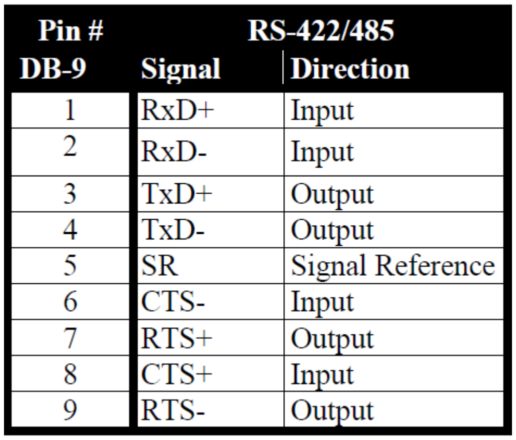


Figure 8: Interlock output board

2.1.5. RS-485 serial connectors

microIOC M-Box-PMAC is equipped with a four port optically isolated RS-485 serial card [7]. Four serial ports are used for communication with RS-485 analogue modules and SSI modules inside PDC units.



Pin #	RS-422/485	
DB-9	Signal	Direction
1	RxD+	Input
2	RxD-	Input
3	TxD+	Output
4	TxD-	Output
5	SR	Signal Reference
6	CTS-	Input
7	RTS+	Output
8	CTS+	Input
9	RTS-	Output

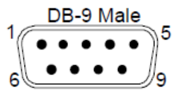


Table 4: RS-485 serial connector pinout, DB9M, (applies to RS485/1 to RS485/4)

2.1.6. PMAC serial connector

This is a standard RS-232 serial connector (DB9F) provided for a direct connection to PMAC. It is used for low-level debugging and programming of PMAC. Special development SW from DeltaTau is required.

Contact	Signal
1	PHASE – optional, insert E8 jumper on PMAC CPU board
2	TXD
3	RXD
4	DSR – internally connected to DTR
5	GND
6	DTR – internally connected to DSR
7	CTS
8	RTS
9	SERVO – optional, insert E9 jumper on PMAC CPU board

Table 5: RS232 PMAC connector pinout, DB9F

2.1.7. JTAG connector

This connector is used for updating firmware on CPLD device. The pinout is shown in Table 6.

Contact	Signal
1	VCC
2	GND
3	TCK
4	TDO
5	TDI
6	TMS

Table 6: CPLD JTAG connector pinout, DB9F

A special cable adapter is needed to connect to the Xilinx programming tool which uses a 14-pin IDC connector with 2 mm raster.

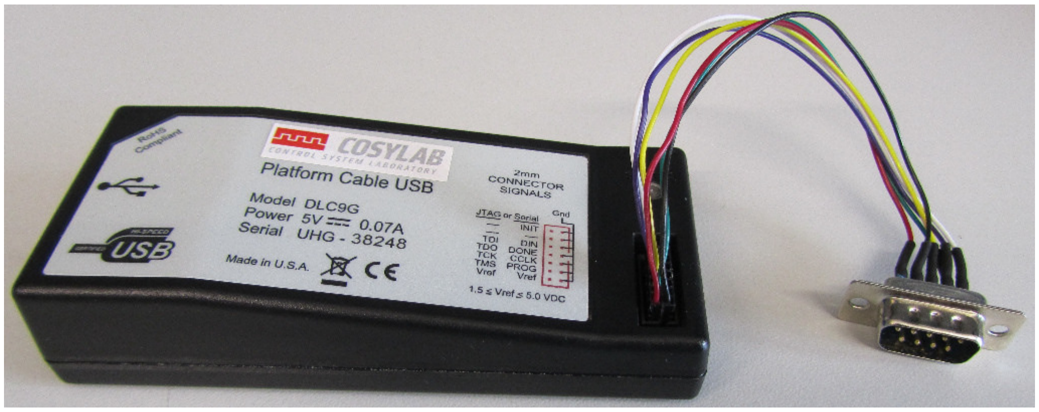


Figure 9: Xilinx programming tool connected to cable adapter

2.1.8. XPORT Pro connection

XPORT Pro connects the microIOC serial port to Ethernet. It allows the user to control microIOC remotely including access to BIOS, Grub and Linux terminal. Another feature presents the option of resetting the microIOC (hard reset) by setting RESET switch to 1 for more than 3 seconds.

To configure or use XPORT, the user can choose to use a web interface or connect to the device via a terminal (telnet, ssh,…). All that is needed is its IP address which can be obtained from your network administrator or by using a program named DeviceInstaller [8]. This will show all XPORTs on the local network. They can be differentiated by their MAC address.

More information on the device and how to use it can be found on Lantronix XPORT Pro web site [9].

2.1.8.1. Configuration of XPORT Pro and microIOC

The most practical method is to use a web interface as it is more user friendly but there is also an option to configure XPORT using a telnet connection.

The fastest way is to use an XML import option with cosylabXPORT.xml file. This configuration has a tunnel (telnet) for device terminal redirection on RS232 line and a reset switch. It uses DHCP for network configuration. Other settings like passwords and SSH keys/users need to be done manually.

If the XML import option is not used, then 4 things need to be configured: network, line and tunnel settings, CPM switch for reset button. Network configuration depends on architecture. DHCP is the most practical but it is advisable that a network administrator configures the router so it always gives the same IP to XPORTs MAC address.

Configuration from cosylabXPORT.xml file will be presented as an example.

Line

Interface: RS-232

State: Enabled

Protocol: Tunnel

Baud rate: 115200

Parity: None

Data bits: 8

Stop bits: 1

Flow control: None

Tunnel (Accept mode is used)

Mode: Always

Local port: 10001

Protocol: Telnet

Everything else is set to Disabled/None.

In order to use SSH instead of telnet additional SSH configuration needs to be done (users + keys).

CPM

Create group RESET

Add CP2 (pin 7) as OUTPUT

Check that the group is enabled. For other configuration options and explanations refer to XPORT Pro user guide [9].

microIOC configuration

In order for everything to work few adjustments need to be made in BIOS and linux system files. In BIOS → Advanced → Serial Port Console Redirection → Enable the correct COM port (in our case COM4) and check if all settings are correct.

Name	Value
Terminal Type	As desired
Bits per second	115200
Data Bits	8
Parity	None
Stop Bits	1

Table 7: COM4 port BIOS settings

The following settings require root privileges to set-up. First add permission for the serial port to login as root. To do this edit the file: "`/etc/securetty`". Add the desired ttyS* to the list (* is port number, in our case, add ttyS3).

The second file to edit is "`/etc/inittab`". At the end of the file add the following line: `T0:23:respawn:/sbin/getty -L ttyS* 115200 vt100` (* needs to be changed to desired port)

To see Grub and its boot messages edit the third file "`/boot/grub/menu.lst`". Add the following commands to the end of kernel line:

`console=ttyS*,115200n8 console=tty0` (again change * to desired port)

The line should look something like this:

```
kernel /vmlinuz root=dev/hdb1 ide-core.nodma=0.1 ro vga=794
8250.nr_uarts=5 console=ttyS3,115200n8 console=tty0
```

Also comment out/remove the `splashimage` command as it cannot be redirected to serial.

Usage

There are two options for connecting to the microIOC terminal. One requires you to know the tunnel port:

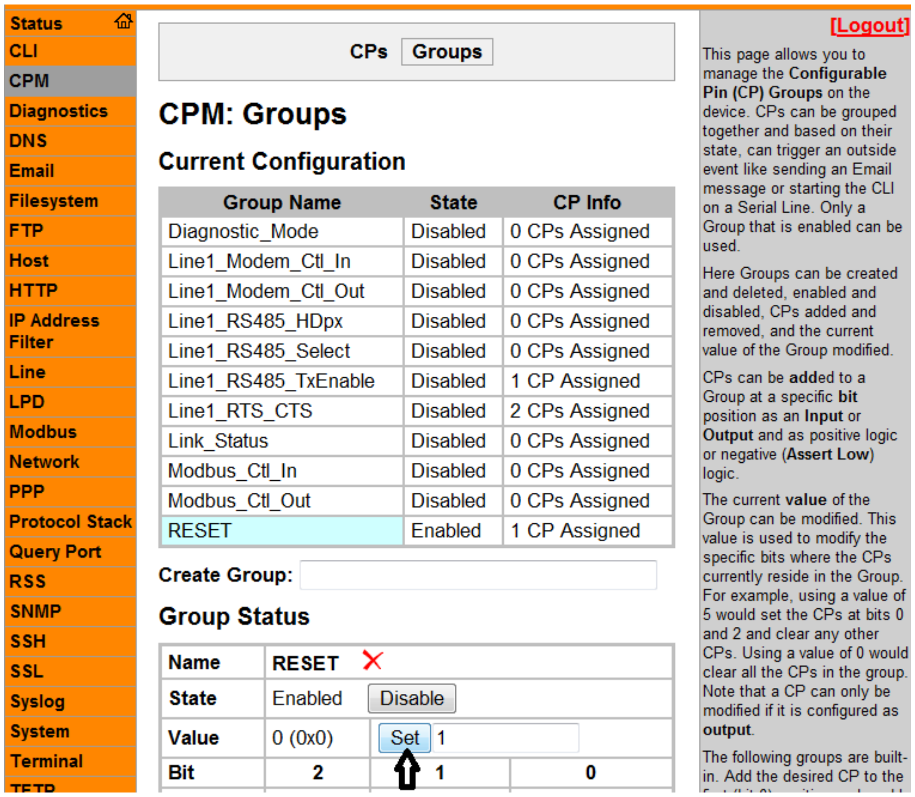
1. Connecting to XPORT with telnet on the default port will display a menu (if it was enabled) and presents choices to go to microIOC terminal (1) or to connect to XPORT's command line interface (2) (CLI). Select 1 to connect to microIOC.

```
[tango@tangobox Desktop]$ telnet 10.5.2.189
Trying 10.5.2.189…
Connected to 10.5.2.189.
Escape character is '^]'.
Connection menu: (select by number)
1) microIOC                    2) Exit to CLI
3) Log out
Selection = _
```

2. If the port for the tunnel is known, we can also directly connect to the microIOC by specifying the port in the telnet command. In our case, the command for direct connection would look like this: `telnet 10.5.2.189 10001`

To reset the device we also have two options:

1. Web interface. By clicking on CPM → Groups → RESET → Put 1 to value field → Set → Wait at least 3 seconds(can use tunnel to see if machine restarted) → Set it back to 0



CPM: Groups

Current Configuration

Group Name	State	CP Info
Diagnostic_Mode	Disabled	0 CPs Assigned
Line1_Modem_Ctl_In	Disabled	0 CPs Assigned
Line1_Modem_Ctl_Out	Disabled	0 CPs Assigned
Line1_RS485_HDpx	Disabled	0 CPs Assigned
Line1_RS485_Select	Disabled	0 CPs Assigned
Line1_RS485_TxEnable	Disabled	1 CP Assigned
Line1_RTS_CTS	Disabled	2 CPs Assigned
Link_Status	Disabled	0 CPs Assigned
Modbus_Ctl_In	Disabled	0 CPs Assigned
Modbus_Ctl_Out	Disabled	0 CPs Assigned
RESET	Enabled	1 CP Assigned

Create Group:

Group Status

Name	RESET		
State	Enabled		
Value	0 (0x0)	1	
Bit	2	1	0

This page allows you to manage the **Configurable Pin (CP) Groups** on the device. CPs can be grouped together and based on their state, can trigger an outside event like sending an Email message or starting the CLI on a Serial Line. Only a Group that is enabled can be used.

Here Groups can be created and deleted, enabled and disabled, CPs added and removed, and the current value of the Group modified.

CPs can be **added** to a Group at a specific **bit** position as an **Input** or **Output** and as positive logic or negative (**Assert Low**) logic.

The current **value** of the Group can be modified. This value is used to modify the specific bits where the CPs currently reside in the Group. For example, using a value of 5 would set the CPs at bits 0 and 2 and clear any other CPs. Using a value of 0 would clear all the CPs in the group. Note that a CP can only be modified if it is configured as **output**.

The following groups are built-in. Add the desired CP to the

Figure 10: Remote reset of microIOC using web interface

2. Telnet to CLI. Connect to XPORT and choose 2 to go to the CLI. Type the following commands:

```
 enable
cpm
set RESET 1 (wait few seconds)
set RESET 0
```

2.2. PDC

PDC extends microIOC M-Box-PMAC with power driving capability for direct connection of stepper motors and position switches. It also remotely handles voltage acquisition from feedback-potentiometers and provides it via serial RS-485 communication. The PDC block diagram is shown in Figure 3 and back panel in Figure 12. PDC is capable of serving four axes.

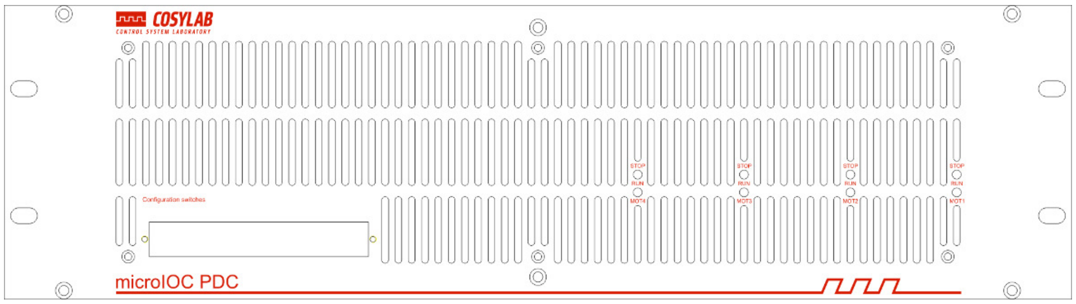


Figure 11: PDC front panel

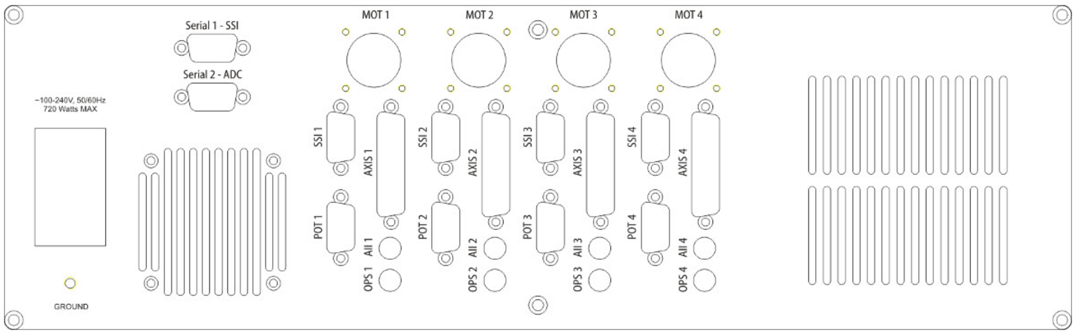


Figure 12: PDC back panel

2.2.1. Axes inputs/outputs

Axes input/output connectors (Figure 12, AXIS 1 – AXIS 4) are prepared for a direct 1-to-1 connection to microIOC M-Box-PMAC. Pinout is identical to the one shown in Figure 6. Connectors AXIS 1 – AXIS 4 are of type DB-25 female.

2.2.2. RS-485 serial connectors

Serial connector Serial 1-SSI is used for communication to the SSI module located inside PDC. Serial connector Serial 2-ADC is used for communication to the RS-485 analogue module inside PDC. Connectors are of type DB-9 female and pinouts are given in Table 8.

Contact	Signal
1, 3	RxD+, TxD+
2, 4	RxD-, TxD-
other	n.c.

Table 8: serial connections on PDC unit

2.2.3. Axis-interlock inputs

Four axis-interlock inputs (Figure 12, AII 1 – AII 4) are wired to microIOC-M-Box-PMAC.

Contact	Signal
1	Axis interlock in +
2	Axis interlock in -

Table 9: axis interlock input connector, LEMO socket type EPL.0S.302.HLN

To allow any movement, the axis interlock inputs must be short circuited.
A floating short circuit must be applied.

2.2.4. Out-position signal outputs

PDC provides four `OutPosSig` (out-position signal, Figure 12, ops1-ops4) outputs that are controlled directly by `OutLimSW` inputs. `OutPosSig` outputs are implemented by relays. Output of the relay is closed only when PDC is connected to a power supply and the out limit switch is hit (i.e. out limit switch is opened).

Contact	Signal
1	out-position signal (relay contact 1)
2	out-position signal (relay contact 2)

Table 10: out-position signal output connector, LEMO socket, EPL.0S.302.HLN

2.2.5. Potentiometer and limit switches connector

Connector wiring is shown in Table 11. PDC interface circuit provides a 10V reference voltage that is applied to potentiometer end contacts. The potentiometer middle tap voltage is differentially measured using RS-485 analogue acquisition module. **Maximum output current on pin 8 – Potentiometer reference voltage, should not exceed 30mA.**

contact	signal	connection
1	**`OutLimSW`** (out limit switch)	PDC interface bord ⇨ microIOC-M-Box-PMAC
2	**`TchLim`** (touching limit switch)	microIOC-M-Box-PMAC
3	common contact for limit switches	24V power supply
4	**`InLimSW`** (in limit switch)	PDC interface bord ⇨ microIOC-M-Box-PMAC
5	N.C.	
6	Potentiometer reference voltage GND	Reference voltage GND RS-485 analogue_in_-
7	Potentiometer middle tap	RS-485 analogue_in_+
8	Potentiometer reference voltage +10V	Reference voltage (+10V) RS-485 analogue_in_+
9	N.C.	

Table 11: Potentiometer/feedback connector pinout (DB9F)

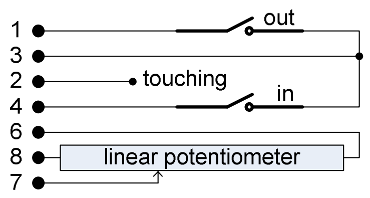


Figure 13: potentiometer and limit switch connector (DB9F)

Output signal **`OutPos`** (out-position signal, Section 2.2.4) is controlled by **`OutLimSW`** (out limit switch). **`OutLimSW`** connection position is thus important.

2.2.6. Stepper motor connector

Connector wiring is shown in Table 12.

Contact	Signal	connection	
A	Phase W1	1	Vexta-stepper drive
B	Phase W2	4	
C	Phase W3	2	
D	Phase W4	5	
E	Phase W5	3	
F	Phase W1/	2	
G	Phase W2/	5	
H	Phase W3/	3	
J	Phase W4/	1	
K	Phase W5/	4	
L	Brake +	Brake + 24 V	Brake circuitry
M	Brake GND	GND	

Table 12: stepper motor connector (UTG01412S, SOURIAU) and connection to stepper driver

2.2.7. SSI module and encoder connectors

The SSI module reads data from encoders. Position of the encoders is readable through two wire RS-485 communication. Communication speed is set to 19200 baud.

Short summary of the implemented commands:

g[N]\n	Get position of N-th sensor, where N can be in range from 0-3
ga\n	Get positions of all sensors in one command call
gt\n	Get temperature reading as voltage from NTC

In case of an invalid command, “E” is returned signaling that the input from user could not be interpreted. Connector wiring is shown in Table 13. The SSI module also provides 24V power supply for SSI encoders.

Pin on DB9F	Signal name
1	SSI_CLK +
2	SSI_CLK -
3	SSI_DATA +
4	SSI_DATA -
5	Enable
6	+24V
7	+24V
8	GND

9	GND

Table 13: SSI encoder connector pinout (DB9F)

2.2.8. Serial module and analog input signals

Serial module ADAM-4117 [6] provides eight 16-bit analogue differential channels. Module is configured for input voltage range of 0-10V to allow potentiometer position read-back. Communication speed is set to 115200 baud. Potentiometer and reference voltages can be read using commands:

#02\n	Get value of all eight channels
#02[N]\n	Get value of N-th channel, where N can be in range from 0-7

Input channel	description
0	voltage from potentiometer 1
1	reference voltage +10V
2	voltage from potentiometer 2
3	reference voltage +10V
4	voltage from potentiometer 3
5	reference voltage +10V
6	voltage from potentiometer 4
7	reference voltage +10V

Table 14: serial module input output signals

2.2.9. Middle switch configuration

Configurations switches and jumpers for middle switch are located on the PDC front panel under the cover.

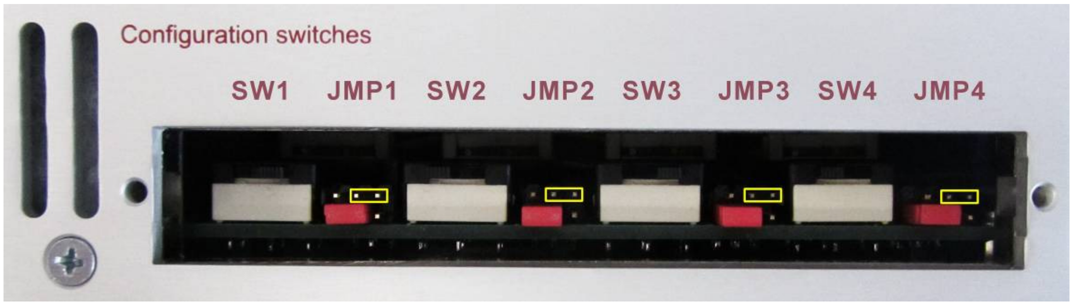


Figure 14: PDC middle switch configuration switches

Double row jumpers are used; the bottom row is suffixed with “a” and the top row is suffixed with“b”. The top row **jumpers (JMP1b to JMP4b), is used to enable/disable middle switch**

functionality. To disable middle switch functionality insert jumper between pins 2-3 (right position, Figure 14 – yellow markings).

Settings for middle switch support are detailed in Table 15.

Item	Setting for N.O. middle switch	Setting for N.C. middle switch
SW1	Right	Left
SW2	Left	Right
SW3	Right	Left
SW4	Left	Right
JMP1a	1-2	2-3
JMP2a	1-2	1-2
JMP3a	1-2	2-3
JMP4a	1-2	1-2
JMP1b to JMP4b	Not inserted	Not inserted

Table 15: PDC middle switch jumper settings

2.3. PDC WIRING SCHEME

Detailed wiring scheme and schematics of the PDC adaptation circuit is given in [9].

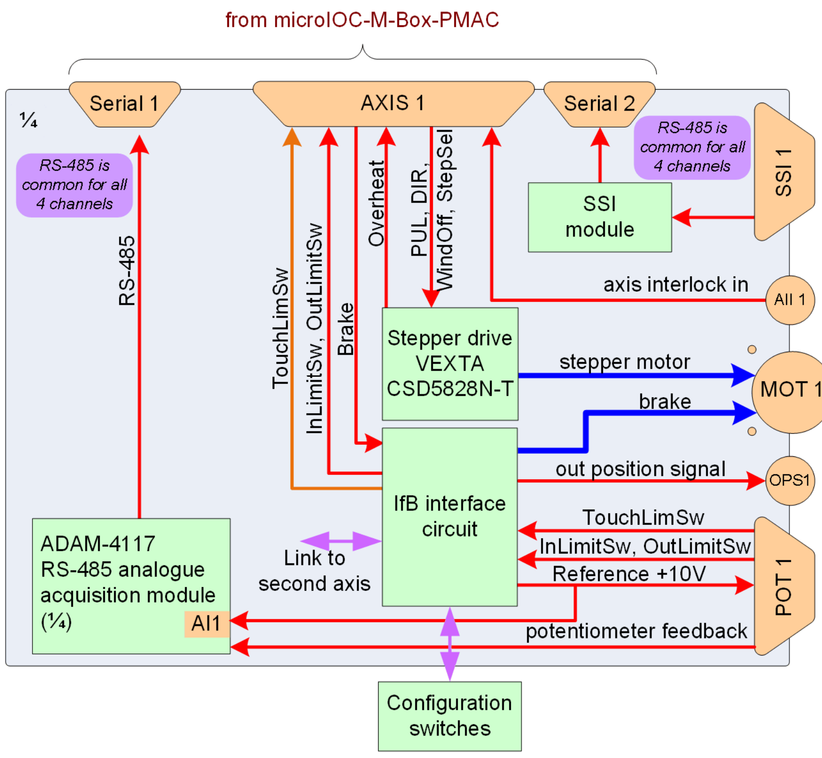


Figure 15: Principle wiring scheme within PDC

2.4. Cabling between microIOC M-Box-PMAC and PDC

2.4.1. Axis signal connection cable

Each axis requires one signal connection cable. Both sides should have identical DB-25 male connectors with one-to-one connection. This connection transmits only low-power signals: 24V / 10~15mA. Twisted-pair wiring should be used on the following pins:

- 1-2, 3-4, 5-6, 7-8, 11-12, 14-15, 16-17, 18-19, 22-23, 24-25

Pins 9, 10, 13, 20, 21 are not used. The extension cable must be shielded and shield connected to both connectors' casings.

2.4.2. PDC communication cable

PDC requires two communication cables. Straight 1:1 DB-9 serial cable with male connector on one and female connector on the other side should be used. This cable transmits only low-power RS-485 signals. Twisted-pair wiring should be used for the pins 1, 3 and 2, 4. The other pins are not used. Shielding should be connected to the connectors’ casings.

3. Software

As show on figure below main components of the software are:

- FESA device classes
  - FESA classes which model motor and pair of motors (slit)
- Local control server
  - Local control server used for motor configuration
- Local control GUI
  - Local control GUI for motor configuration
- System driver
  - C++ system driver that communicates with PMAC via TCP/IP
  - Communicates with any other additional device (e.g. encoders) via serial connection
  - Transforms requests from upper level software to PMAC commands
  - Handles system mode (local control/remote/LCD control)
- Shared memory monitor
  - Used to implement access control fall-back and shared memory cleanup in case any of the application crashes unexpectedly.
- LCD control
  - Used for controlling motors locally from the LCD on the front panel of M-Box.
- PMAC software
  - Low level software used to control single or pair of motors which resides on PMAC controller inside M-Box.
- VHDL code
  - Used for signal configuration

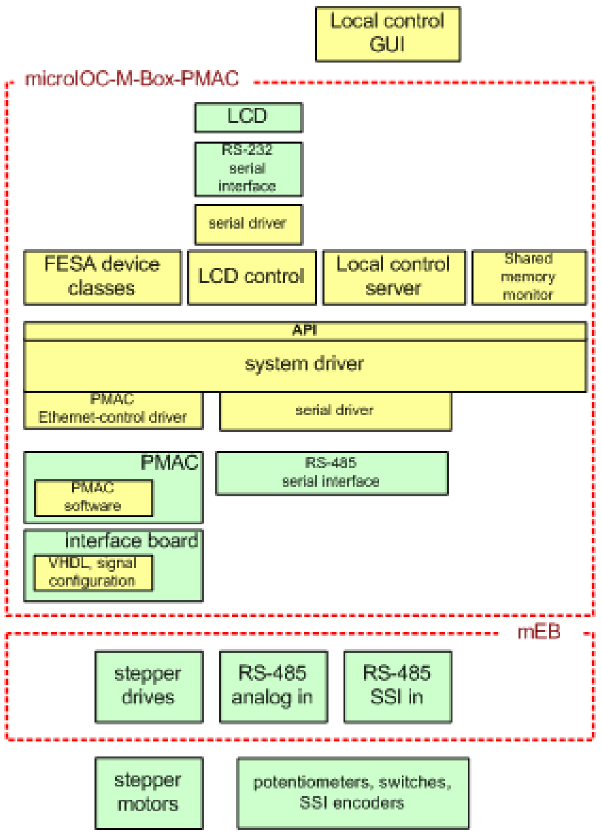


Figure 16: system overview

Details of the software can be found in [11], [12], [13] and [14].

3.1. Roles of software components

Current arrangement of the position-feedback system used in GSI (reading voltage of the potentiometer middle tap, moving in parallel with axis of the motor) does not provide the relation, for which unique calculation could be defined for different axes. In fact, each axis has to be visually (or by means of end switches) inspected for both end positions, where voltages from position potentiometers should be captured and stored. The next parameter to be captured is the number of steps of the stepper motor for the entire allowed range of movement. Using these parameters, the required transformation steps-to-actual-movement-in-mm can be calculated. To simplify this procedure local control GUI and server were developed which provides functionality for commissioning of the motors without the need for FESA device class software. Local control is explained in detail in [12].

Once the required parameters are known, the PMAC's motor positions should be reset. The system driver API provides the means to move a single motor or a pair of motors to an absolute position. The PMAC provides all the required assistance for moving a pair of motors, but it has no way to check if the movements have actually taken place. The PMAC can only issue a sequence of steps, for which it expects the motors will move accordingly. It is the role of the application to request a move (from a PMAC) and check if the axes have moved accordingly. This can be done by reading the feedback potentiometer voltages and using the above mentioned relation for calculation.

The features that are in the domain of the PMAC can be easily set using an API: maximum motor velocity, acceleration (deceleration) time (PMAC handles acceleration and deceleration automatically using these parameters), internal position counters (steps sent to stepper driver), required position and others. See [12] and [14] for details on what can be configured. There are also some parts of the motion control that PMAC can handle automatically based on the data it reads, for example, not allowing movement when the axis interlock input signal is active – when this happens, the control system calls cannot override this, but it can read the status and figure out why the movement is not happening.

3.2. PMAC SOFTWARE

PMAC is a very flexible and high-performance motion control system. The system runs motion control programs that are specific to the application being controlled. If a modified motion control scheme is required, a PMAC can be easily reconfigured and/or its API extended. For details see [3] and [12].

3.2.1. PMAC-CPLD memory access

To enable flexible reconfiguration of the PMAC interface board, it is equipped with a CPLD IC (Section 3.3). In a CPLD there are configuration and status registers that are memory mapped to PMAC memory space via JOPTO and JTHW ports. This enables 8-bit memory accesses to registers inside CPLD. PMAC software can read (or write) from (or to) CPLD registers with use of M-variables which are pointing to JOPTO and JTHW ports. Variable mapping is as follows:

```
M48->Y:$078402,8,8,U ; SEL0-7 lines treated as a byte (ADDRESS)
M58->Y:$078402,0,8,U ; DAT0-7 lines treated as a byte (DATA_IN)
M64->Y:$078400,0,8,U ; I/O00-7 lines treated as a byte (DATA_OUT)
```

3.2.1.1. Read process

PMAC can read from a given CPLD register by issuing the following procedure:

```
M48=ADDRESS  ;set the CPLD ADDRESS bits from which we are reading
;timer (wait for e.g. 1ms)
M58 ;read 8bit DATA that is accessible on M58 variable
```

3.2.1.2. Write process

PMAC can write to a given CPLD register by issuing the following procedure:

```
M48=ADRESS ;set the CPLD ADDRESS bits which user wants to write to
M64 = DATA ; set DATA (JOPTO bits) which user wants to write
;timer (wait for e.g. 1ms)
M48=$D4 ;set the ADDRESS bits to 0xD4 which signals 'write command'
```

```
to CPLD logic
```

3.3. PMAC Interface Board and CPLD Configuration

Outputs of the PMAC are not directly routed to axes connectors. To provide noise immune communication, all the signals are optically isolated and raised to 24V voltage level. For flexibility and future adaptations almost all signals are either monitored by a CPLD or the signals are passed through it. CPLD is a programmable integrated circuit with multiple general-purpose inputs/outputs, which can have user defined relation among them. For example, logical functions between signals or state machine processing can be applied.

Given that additional signals—besides PMAC natively supported signals—are required, these signals are implemented using CPLD; i.e. **`Brake`**, **`StepSel`**, **`OverHeat`**, etc. A CPLD implements PMAC memory mapped registers that control these additional signals. The access to registers is implemented via PMAC expansion ports (see 3.2.1).

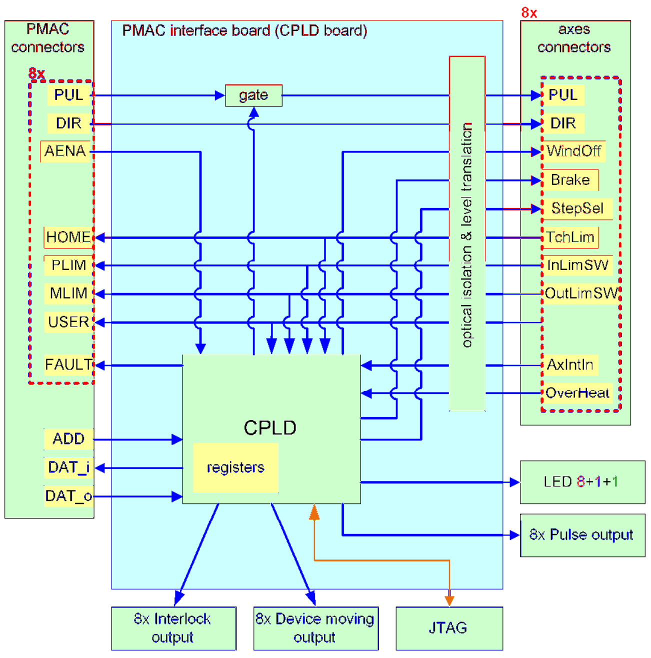


Figure 17: PMAC interface board overview

address	value	R/W	
0x01	**`InLimSW`** bit3: axis-4 → bit0: axis-1	R	In limit switch, 1 = opened, 0 = closed
0x02	**`OutLimSW`** bit3: axis-4 → bit0: axis-1	R	Out limit switch, 1 = opened, 0 = closed
0x03	**`TchLimSW`** bit3: axis-4 → bit0: axis-1	R	touch limit switch
0x04	**`USER`** bit3: axis-4 → bit0: axis-1	R	not used input on DB25(20,21) connector, will read 1
0x05	**`OverHeat`** bit3: axis-4 → bit0: axis-1	R	overheat signal from stepper drive (1 when ok, 0 when overheat)
0x06	**`AxIntIn`** Bit7: axis-8 → bit0: axis-1	R	axis interlock inputs (for movement to be allowed, inputs must be 0)
0x07	**`AxesLEDs`** bit7: axis-8 → bit0: axis-1	R/W	axes red LEDs: 1 = LED on, 0 = LED off
0x08	**`MasterLEDs`**	R/W	set green-OK-LED and red-general-fault-LED bit1: green-OK-LED, bit0: red-general-fault-LED 1 = LED on, 0 = LED off
0x09	**`VHDLversion`**	R	read version of the CPLD configuration
0x0A	**`Brake`** bit3: axis-4 → bit0: axis-1	R/W	BRAKE output: 1 = brake is off, 0 = brake is on this signal should only be controlled by PMAC
0x0B	**`DeviceMove`**	R/W	Device moving outputs
0x0C	**`AxesLedBlink`** bit7: axis-8 → bit0: axis-1	R/W	axes red LEDs blinking: 1 = LED blinks, 0 = LED do not blink
0x0D	**`StepSel`** bit3: axis-4 → bit0: axis-1	R/W	step select signal for stepper drive 1 = half step, 0 = full step

Table 16: CPLD registers' descriptions

By using the provided command line application (and API), reading and writing to CPLD registers can be done by issuing commands **`writecpld`** and **`readcpld`** (Section **Error! Reference source not found.**). All registers are 8-bit in size.