cosylab

GSI Motion Control

Implementation

Revision:	1.0
Status:	Released
Repository:	/home/cvsroot/cosycvs
Project:	GSI-MotionControl
Folder:	Operations/Projects/GSI-MotionControl
Document ID:	CSL-DOC-07-34913
File:	GSI-MotionControl-implementation
Owner:	jdedic
Last modification:	July 12, 2007
Created:	July 5, 2007

Document History

Revision	Date	Changed/ reviewed	Section(s)	Modification
1.0	2007-7-12	jdedic/ gpajor	All	Created.

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 motion control system for GSI, specifically for a septum device.

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

GUI ..... Graphical User Interface

HW ..... Hardware

mEB ..... microIOC-M-Box-PMAC extension box

PMAC ..... Programmable Multi-Axis Controller

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, XC95 series CPLD:
www.xilinx.com/products/silicon_solutions/cplds/xc9500_series/xc9500/index.htm

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

[6] RS-485 module, 8-channel Analog Input Module, 16-bit differential:
www.icpdas.com/products/Remote_IO/i-7000/i-7017.htm

[7] RS-485 module, 8-channel Isolated Digital Input and 8-channel Isolated Digital Output:
www.icpdas.com/products/Remote_IO/i-7000/i-7055d.htm

[8] 104-ICOM-2S, Serial Communications Card
www.accesio.com/go.cgi?p=../104/104-icom-2s.html

[9] schematics and wiring documentation:
`mEB-wiring.pdf`
`mEB-interface-board-sch.pdf`

Table of Contents

Figures

Tables

1. SYSTEM OVERVIEW

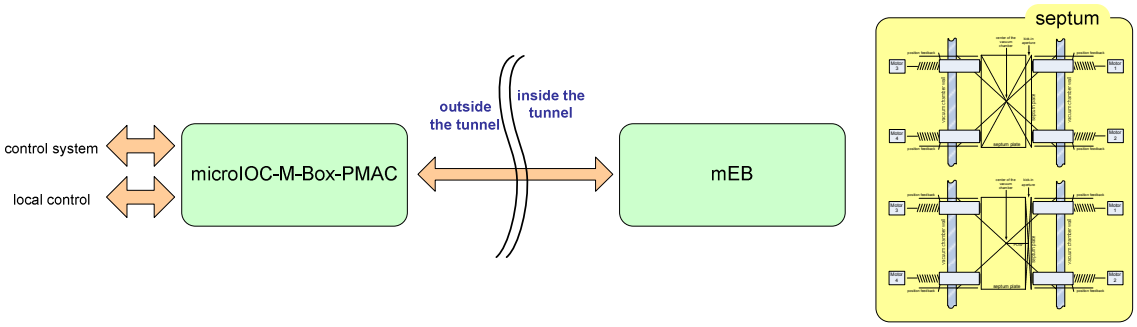

Figure 1: Septum control system description

This document describes the system for controlling a septum device. System requirements specifications are provided in [1]. Figure 1 gives a system overview of the system. Cosylab has provided two cases:

- microIOC-M-Box-PMAC (Figure 2) – microIOC-based product, offering an advanced motion control functionality and control of general-purpose signals. microIOC is very flexible SW platform, which provides control of all the modules and provides a hosting environment for control-system application (product benefits and features are described in document [2]). Motion control is implemented by a PMAC controller [3] that handles the motion control of up to eight axes. Additionally, a PMAC interface 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 Xilinx CPLD XC95216, [4]) by the means of signals' mappings that the board passes through (inverted signals, applying logical functions, etc). PMAC interface board also extends available PMAC's per-axis input/output signals to provide control of the additional functionality (such as brake control, interlocks, etc.). To provide communication with remote analog module (for position feedback) and digital input/output module an RS-485 communication card with optically isolated outputs (addressing large distances and possible noisy environments) was added.
- mEB (Figure 3) – microIOC-M-Box-PMAC extension box – provides power supply [5] for 4 stepper drives Vexta CSD 5828 N-T, RS-485 analogue inputs module [6], RS-485 digital inputs/outputs module [7] and simple mEB interface circuit (signal adaptation).

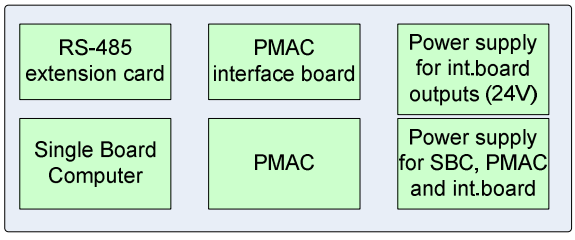

Figure 2: microIOC-M-Box-PMAC block diagram

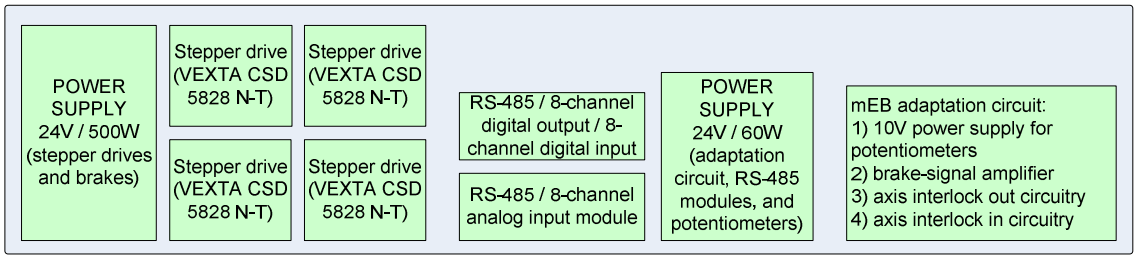

Figure 3: mEB block diagram

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

2. SIGNAL DESCRIPTIONS

2.1. MICROIOC-M-BOX-PMAC

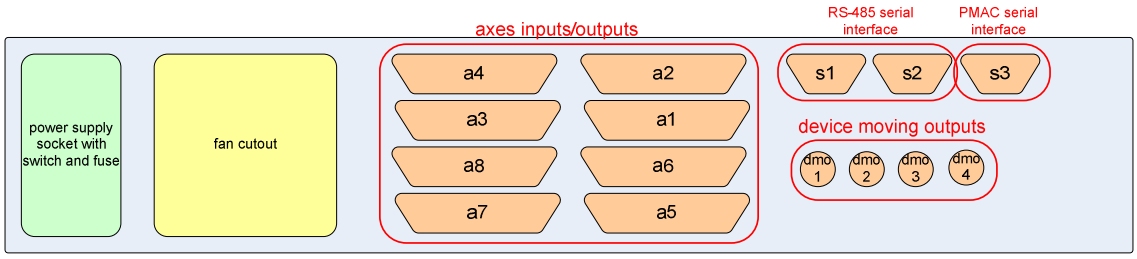

Figure 4: microIOC-M-Box-PMAC back panel

2.1.1. Axes inputs/outputs

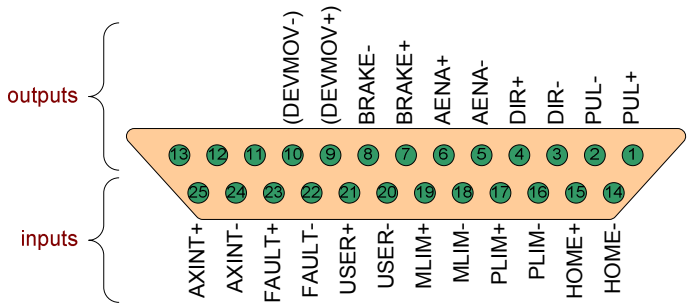

Figure 5: axis connector (DB-25F)

Description of the signals (per axis, totally 8 axes available:

- PUL+/-, DIR+/-, AENA+/-: signals to control the stepper drive. These are PMAC's signals and therefore handled by its motion control programs. Interface board applies only optical isolation and level translation. If required, PUL signal can be conditionally gated by CPLD (e.g. to provide additional safety level). Outputs of these signals are prepared for direct connection to optocoupler inputs of stepper drive.
- BRAKE+/-: this output is controlled by a CPLD and optical isolation and level translation is applied. BRAKE signal is handled by PMAC within its motion control programs and is not intended to be controlled by higher level SW (see 3.3.2 for details). This output is of type short-circuit / open-circuit and is used for mEB interface board to drive brake amplifier (output is similar to the output shown in Figure 6).
- DEVMOVE+/-: an output signal to alert other devices about active movement. This output is controlled by a CPLD and optical isolation and level translation is applied. DEVMOVE signal is controlled by means of writing to a CPLD register at address 0B: e.g.: **`writecpld 0B data`** (data can be any of 0x00-0xFF and each bit controls a DEVMOVE output of corresponding axis). When a 1 is written to a specific bit, its corresponding output is closed (short-circuited). Similarly, when a 0 is written to a specific bit, its corresponding output open

(open-circuited). This is a floating output (optoisolated) and is intended to short circuit external signals of U<24V and I<100mA (see Figure 6). In Figure 5 DEVMOVE+/- signals are shown in parenthesis as for axes 1-4 this signal is wired to device moving outputs (Figure 4, LEMO connectors dmo1-dmo4). See 2.1.2 for device-moving connector pinout.

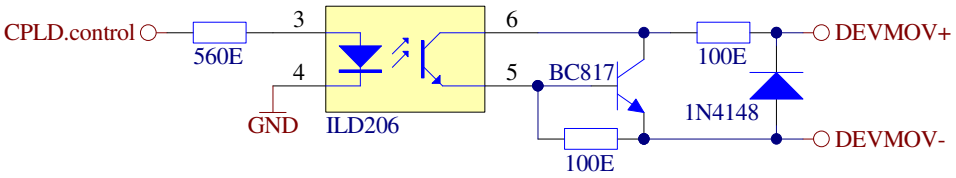

Figure 6: DEVMOV output

CPLD register @ 0x0B – read & write (1 – output is closed, 0 – output is open)							
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit1
DEVMOVE axis 8	DEVMOVE axis 7	DEVMOVE axis 6	DEVMOVE axis 5	dmo4 (LEMO)	dmo3 (LEMO)	dmo2 (LEMO)	dmo1 (LEMO)

- HOME+/-, PLIM+/-, MLIM, USER+/-: these signals are routed to PMAC (interface board applies signal adaptation and optical isolation). Purpose of these signals is to provide limit (PLIM, MLIM) and home switches (HOME) and additional general purpose input (USER). PMAC's motion control programs can provide response to them. CPLD also reads these inputs and can provide any actions regarding the states of these inputs. For a septum none of these signals are used, but nevertheless, inside mEB they are connected to potentiometer feedback connector (for future use and testing purposes; HOME, PLIM, MLIM). See 2.2.4 for connection details. To prevent ground-floating issues this inputs shout be controlled by a floating closed / open switch (e.g. mechanical switch or output of an opto-coupler). See Figure 7 for explanation.

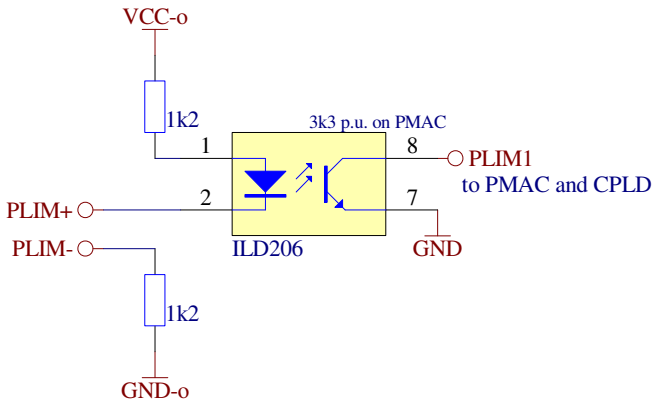

Figure 7: PLIM input (also valid for HOME, MLIM and USER inputs)

- FAULT+/-: PMAC has dedicated amplifier fault (which can be used for example to stop the motion programs), however this signal is routed to PMAC through CPLD to provide additional control over it. This way, FAULT signal can be inverted or other functions can be

applied (to stop the movements also by other means). See Figure 8 for explanation. For the septum project this input is not used.

- AXINT+/-: this input signal provides axis interlock input capability. It is controlled by applying close or open contact on its input. Because it is routed to CPLD, its polarity and/or functionality are fully programmable (for a septum project any AXINT causes amplifier fault on PMAC's axes 1-4, which prevents any movement). To prevent ground-floating issues this inputs shout be controlled by a floating closed / opened switch (taken care on the mEB interface board). See Figure 8 for explanation.

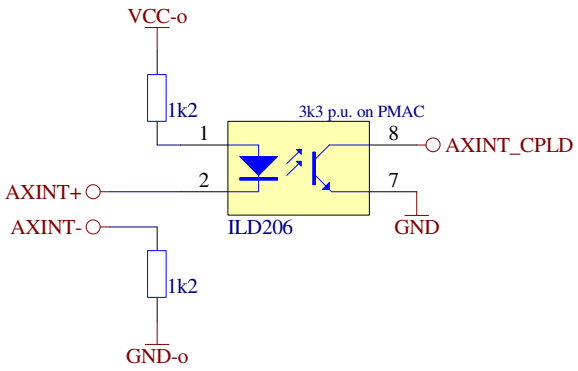

Figure 8: AXINT input (also valid for FAULT input)

2.1.2. Device moving outputs

As described above, device moving signals (DEVMOVE) for axes 1-4 are available on LEMO connectors and for axes 5-8 are available on DB-25 pins 9 and 10. For septum project only axes 1-4 are used.

Contact	Signal
1	DEVMOV+
2	DEVMOV-

Table 1: device moving output (dmo1-dmo4), LEMO, socket ERA.OS.302.CLL

2.1.3. RS-485 serial connectors

microIOC-M-Box-PMAC is equipped with a two-port RS-485 serial card [8]. The card has optically isolated outputs to enable noise-immune communication in a noisy environment. For a septum project only one serial communication channel is used, which is used for communication to both RS-485 modules inside mEB.

Contact	Signal
1	RX+,TX+
2	RX-,TX-
3-9	n.c.

Table 2: RS-485 serial connector pinout, DB9F

2.1.4. PMAC serial connector

This is RS-232 serial connector provided for a direct connection to PMAC and provides capability of advanced debugging and programming using a development SW from DeltaTau.

2.2. mEB

mEB complements a microIOC-M-Box-PMAC with a stepper drive capability and remote acquisition of feedback-potentiometer voltages. Its block diagram is shown in Figure 3 and back panel in Figure 9. For a septum device, mEB is capable of serving four axes.

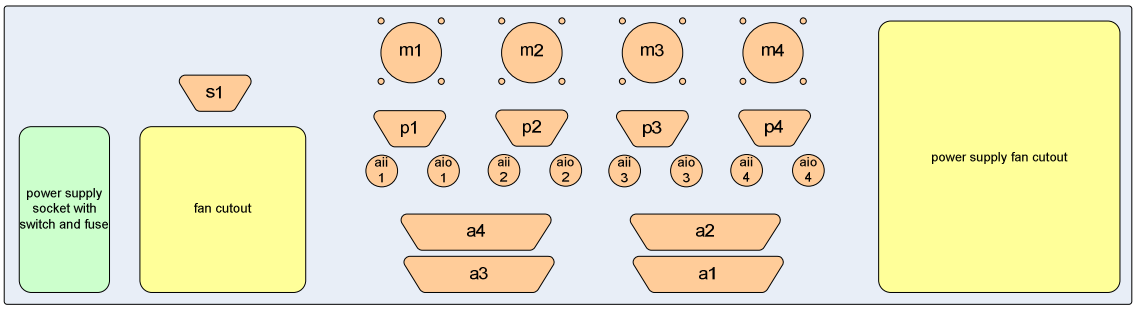

Figure 9: mEB back panel

2.2.1. Axes inputs/outputs

Axes input/output connectors (Figure 9, a1-a4) are prepared for a direct 1-to-1 connection to microIOC-M-Box-PMAC. Pinout is thus identical to the one shown in Figure 5, with the direction reversed.

2.2.2. Axis-interlock inputs

Four axis-interlock inputs (Figure 9, aii1-aii4) are connected through mEB interface circuit, which provides floating closed/open switch towards microIOC-M-Box-PMAC.

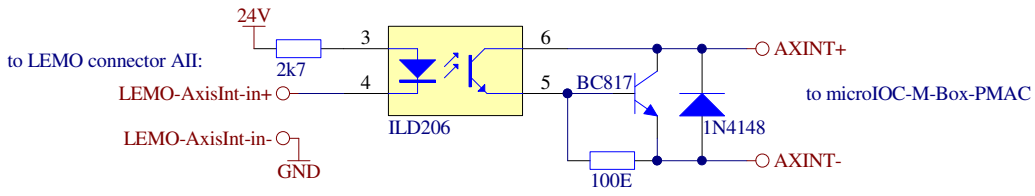

Figure 10: axis interlock input

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

Table 3: axis interlock input connector, LEMO socket, ERA.OS.302.CLL

2.2.3. Axis-interlock outputs

Four axis interlock outputs (Figure 9, aio1-aio4) are connected through mEB interface circuit and controlled by RS-485 module digital outputs. Check section 2.2.6 for RS-485 digital input/output module description and SW control.

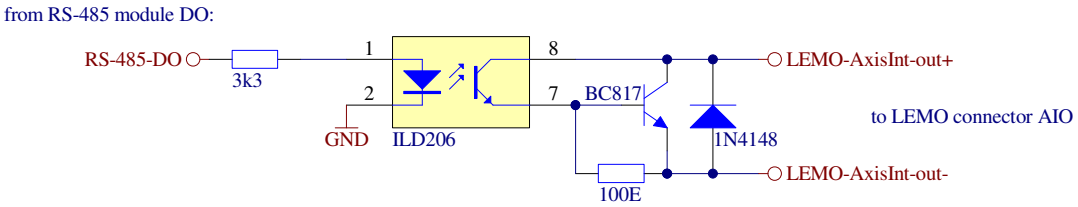

Figure 11: axis interlock output

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

Table 4: axis interlock output connector, LEMO socket, ERA.OS.302.CLL

2.2.4. Potentiometer/feedback connector

In, out and touching switch are connected to PLIM, MLIM and HOME signals on the microIOC-M-Box-PMAC. Reference voltage (10V), mEB interface circuit) is applied to potentiometer and voltage is differentially measured (RS-485:AI+/AI-) between potentiometer sliding and reference voltage GND connector.

contact	signal	connection
1	Limit Switch In	PLIM+ (microIOC-M-Box-PMAC)
2	Touching Limit Switch	HOME+ (microIOC-M-Box-PMAC)
3	GND Limit Switch	PLIM-, MLIM-, HOME-
4	Limit Switch Out	MLIM+ (microIOC-M-Box-PMAC)
5	N.C.	
6	U-Ref GND	Vref_GND (10V-), RS-485 AI-
7	Potentiometer slider	RS-485 AI+
8	U-Ref +	Vref+ (10V+)
9	N.C.	

Table 5: Potentiometer/feedback connector pinout (DB9F)

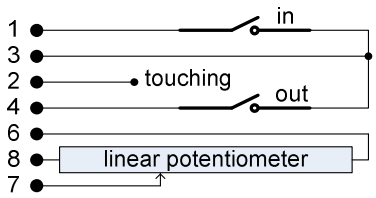

Figure 12: potentiometer and limit switch connector (DB9F)

2.2.5. Stepper motor connector

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 6: stepper motor connector (UTG01412S, SOURIAU) and connection to stepper drive

2.2.6. Serial module digital input/output signals

Serial module I-7055D [7] (built inside mEB) provides 8 digital outputs and 8 digital inputs.

4 digital outputs are used as 4 interlock output signals (mEB interface circuit applies optical isolation and level translation, type: open/closed switch), available on the back-panel of the mEB (see 2.2.3 for electrical specifications, aio1-aio4).

4 digital outputs are used for per-axis selection of half/full step select on the stepper drive. Because 24V is applied to output stage of the module, a 2k7 resistor is used in series for connection to stepper-drive optocoupler input (giving approx. 8mA current).

4 digital inputs are used for per-axis monitoring of overheat signal from stepper-drive. Stepper-drive optocoupler output is directly connected to module input (between DIx and DI.GND).

output	description	SW control
0	axis interlock out 1	1 = output closed, 0 = output open
1	axis interlock out 2	
2	axis interlock out 3	
3	axis interlock out 4	
4	half/full step select 1	1 = half step, 0 = full step
5	half/full step select 2	
6	half/full step select	
7	half/full step select	

Table 7: serial module output signals

Input	description	SW control
0	overheat stepper drive 1	1 = overheat, 0 = OK
1	overheat stepper drive 2	
2	overheat stepper drive 3	
3	overheat stepper drive 4	
4	n.c.	
5	n.c.	
6	n.c.	
7	n.c.	

Table 8: serial module input signals

For instructions how to read input signals and set output signals see section 3.2.

2.2.7. Serial module analog input signals

Serial module I-7017 [6] provides eight 16-bit differential channels. For potentiometer position read-back module is configured for input voltage range of +/-10V.

Input	description
0	voltage from potentiometer 1
1	voltage from potentiometer 2
2	voltage from potentiometer 3
3	voltage from potentiometer 4
4	reference voltage
5	not used
6	not used
7	not used

Table 9: serial module input output signals

For instructions how to read potentiometer voltages see section 3.2.

2.3. mEB WIRING SCHEME

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

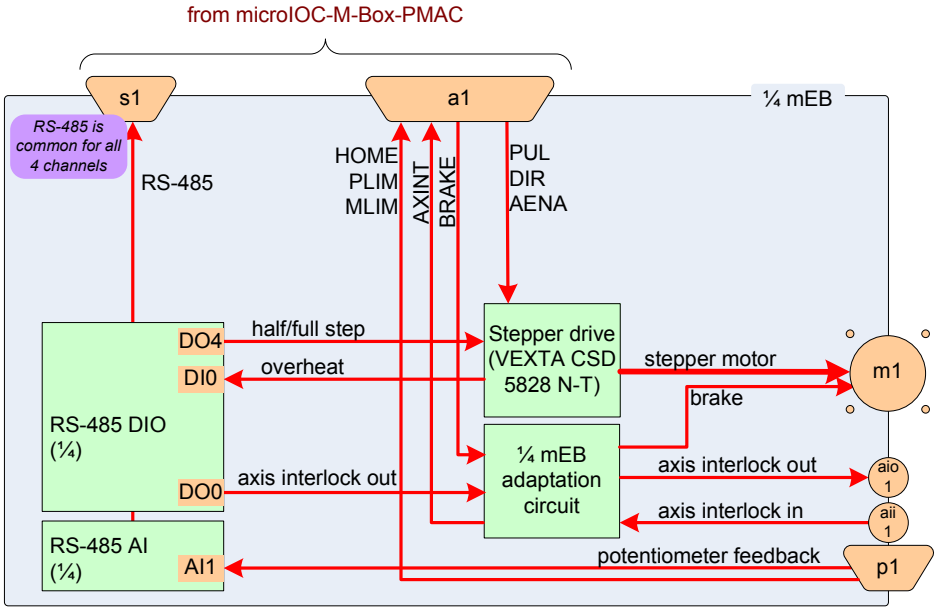

Figure 13: principle wiring scheme within mEB

3. SOFTWARE

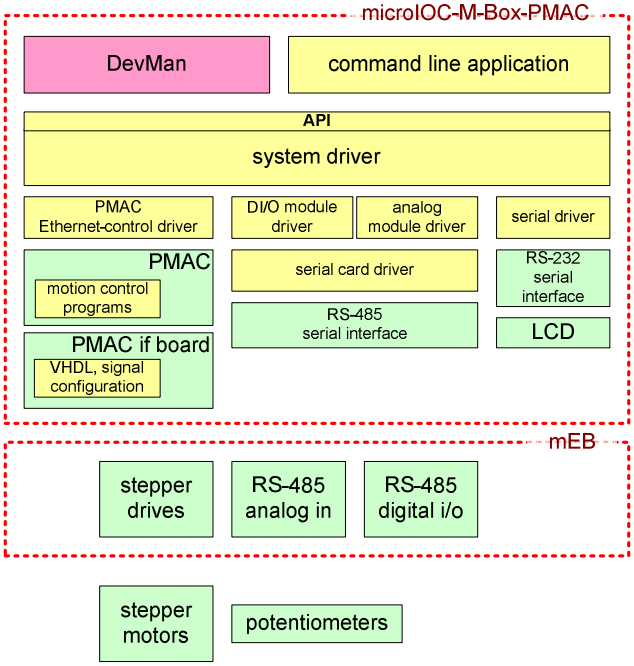

Figure 14: system overview

3.1. SYSTEM DRIVER ROLES

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 individual 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 remembered. The next parameter to be captured is the number of steps of the stepper motor for the whole allowed range of the movement. Using these parameters, the required transformation steps-to-actual-movement-in-mm can be calculated.

Furthermore, based on the agreement, commands provided by the driver allow moving motors in manner which are not necessary parallel. It is the responsibility of the calling application (e.g. device manager) to issue calls that would only result in moves, which are allowed for a given mechanics (by using the above mentioned steps-to-movement and readback-voltage-to-actual position relation).

Once the required parameters are known, the motor-pairs should be set in parallel and PMAC's motor positions should be reset. API provides the methods to move single motor to absolute position or parallel axis for relative position. The PMAC provides all the required help for doing parallel movements, but it has no way to check if the movements have really 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. To enforce the safety, we have programmed PMAC in way that

it only allows limited movements. This is a safety feature of the PMAC, which relies that the calling application will check the position of the axes (i.e. voltages) every time it will issue another move.

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), maximum allowed single-call movement, internal position counters (steps sent to stepper driver), required position and others. See next section for details what can be configured. Also there are some parts of the motion control that PMAC can handle automatically based on the data it reads. This is for example not allowing movements when axis interlock input signal is active – when this happens, the control system calls can not override this, but it can read the status and figure out why the movement is not happening.

Furthermore, the preciseness of the potentiometer-voltage read-back has to be figured out by testing. Most reliable way to make checks is to make them inside device server where there is all information available: information from the system driver plus information about calibration and accuracy of driver data.

We have discussed different possibilities, how to handle the impreciseness and calibration of the feedback potentiometers and we came to conclusion that it is the best to provide GSI low-level methods, which enable reading all the relevant raw information and handle it to suite for particular needs. However, if there will be some special functionality required later that would be better suited in lower-level SW, this could be arranged. At start it is best to allow the device manager to have all the control.

3.2. SYSTEM DRIVER, API AND CONSOLE APPLICATION

System driver provides communication path for controlling the following subunits: PMAC, serial module with analog inputs, serial module with digital inputs/outputs, and LCD.

API of the system driver provides the higher-level software all the necessary methods to control the underlying hardware. The system driver was intentionally developed as "pass-through" (to the degree possible) not to limit the possibilities of application SW. Command line application is provided that demonstrates the use of the API. Application **`gsiConsole`** is located in `/opt/gsi` folder of the provided microIOC-M-Box-PMAC. It is run by typing **`./gsiConsole`**.

command	command description	signal description
readpos <motor_number>	returns motor readback position	returns the voltage of the potentiometer from the serial AD module, check 2.2.4 & 2.2.7
readsetpos [<motor_number>]	returns motor setpoint position	position that was specified by a command to PMAC
readpmacpos [<motor_number>]	returns PMAC internal readback motor position	PMAC's internal position counters for keeping track of the subsequent moves
resetpos <pos1> <pos2> <pos3> <pos4>	resets pmac internal readback position	whenever this command is issued all axes are killed (amplifier disabled and brake enabled)
movepos <motor_number> <pos>	moves motor to new position	absolute single axis movement

move12 <steps>	moves motors 1&2 for steps	relative parallel movement of two axes (value adds or subtracts current value)
move34 <steps>	moves motors 3&4 for steps	relative parallel movement of two axes (value adds or subtracts current value)
stop	stops all motion	
killmot <motor_number>	kills motor and enables brake	when further moves are no longer expected stepper drive should be shut down and brake engaged, see 2.2.1 / BRAKE for description
kill12	kills motor 1&2 and enables brake	as above
kill34	kills motor 3&4 and enables brake	as above
setvelo [<motor_number>] <velocity>	sets velocity for one or all motors in cts/s	defines the pulse frequency given to stepper drive (in PMAC it is limited to 10.000 and even if set higher, PMAC uses 10.000p)
readvelo	reads velocity for all motors in cts/s	
setacc [<motor_number>] <velocity>	sets acceleration time for one or all motors in msec	defined as time in which motor achieves its maximum velocity
readacc	reads acceleration time for all motors in msec	
motstatus	returns motor statuses	described in API header file
pmacstatus	returns PMAC statuses	described in API header file
doistatus	returns DIO status	higher byte represents output, lower byte represents input
setdo <do values>	sets digital output	interlock output 1-4 (see 2.1.3) & half/full step select 1-4 (outputs of the serial module), see 2.2.6
readcpld <address>	reads cpld at specified address	see 3.3.3.1 & 3.4
writecpld <address> <data>	writes to a cpld at specified address	see 3.3.3.2 & 3.4
writelcd <line1> <line2>	writes to LCD	
readrefv	reads voltage reference	reference voltage applied to potentiometers (10V)
maxsteps	returns max allowed steps	value is set at construction time and can not be changed while usage
readhigh <motor_number>	returns high position limit	same as above
readlow <motor_number>	returns low position limit	same as above
exit		

Table 10: console application commands

3.2.1. Examples

setdo F1 – set axis interlock out for axes 1-4 (aio1-aio4) and select half stepping for 1st motor (see 2.2.6)

doistatus – read output settings (axis interlock out and half/full step select) of outputs and input signals (stepper overheat) (see 2.2.6)

readpos 1 – read potentiometer voltage of the 1st axis (see 2.2.7)

writelcd GSI_demo_system 192.168.0.109 – writes data to LCD

writecpld 7 0F – turn on red LEDs for axes 1-4 (see 3.4 and Table 12)

readcpld 9 – read CPLD firmware version (see 3.4 and Table 12)

readpmacpos – read PMAC's internal position counters

movepos 1 2000 – move axis 1 to absolute position 2000 counts

move12 -500 – apply relative move of 500 counts backward

3.3. PMAC SOFTWARE

PMAC is very flexible and high-performance motion control system. System runs motion control programs that are specific to 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].

3.3.1. PMAC software installation and startup

Software that is running on PMAC motor controller (Programmable Multi-Axis Controller) is generated in two files and downloaded from the IOC to PMAC only when new software is installed. First default PMAC configuration is downloaded to the PMAC (default-conf-3-3-2006.CFG file) and subsequently project specific file (gsiSeptum.pmac file). This is done from the IOC (/opt/pmac folder) with the following commands:

- Download default PMAC card configuration:

```
./sendfile 192.6.94.5 default-conf-3-3-2006.CFG
```

- Download project settings and software:

```
./sendfile 192.6.94.5 gsiSeptum.pmac
```

At the end of the installation procedure configuration and software on the PMAC is automatically saved and card is reset.

After each power-up (or PMAC reset) 'initialization' PMAC PLC program is run once to put system in the start-up state. This includes:

- Setting PMAC variables that require setting at startup
    - I7m04 - Servo IC m PWM Deadtime / PFM Pulse Width Control,
    - M32, M34 - Turbo PMAC2 General-Purpose I/O Port, Direction control

- Erasing all coordinate system definitions
- Disabling the motor drivers (killing all motors) and enable motor brakes
- Clearing all programs running flags. Program running flag is Turbo PMAC P variable which is set at the beginning of the program and is reset at the end of each program.

Therefore, at startup ALL motors get killed and brakes on ALL motors are applied.

3.3.2. PMAC motion programs execution

There are 6 motion programs stored in PMAC non-volatile memory (EEPROM). Two programs are used for simultaneous move of two motor pairs (motor #1, #2 and motor #3, #4) and 4 for individual move (one for each motor). All moves are done with use of these programs.

When move command is issued to the PMAC, motors are put in appropriate coordinate system and put in closed loop (AENA gets activated). After short pause (50 msecs) motion program releases the brake and after another short pause (50 msecs, to ensure mechanical brake is released) motor is moved to specified position.

Moving a motor to a new position sometimes demands a sequence of short moves, which means same motion program is launched several times. For this reason we do not disable the driver and enable the brake each time motion program finishes, but only when specific 'moving finished' command is sent from PMAC driver. This command sets the 'Moving finished 'Flag Variable to 0 (PMAC P variable) which is picked up by PMAC PLC program. This PLC enables the brake and kills the motor (again two 50 msecs delays are implemented to allow mechanics to come to its position before brake is applied and to make sure brake is applied before motor is killed).

PMAC program	program number	launch command	'moving finished' flag variable
motors #1 and #2 (simultaneous move)	1	&1#1->X#2->Y#1j/#2j/b1r	P45
motors #3 and #4 (simultaneous move)	2	&2#3->X#4->Y#3j/#4j/b2r	P51
move of motor #1	7	&3#1->X#1j/b7r	P57
move of motor #2	8	&4#2->X#2j/b8r	P63
move of motor #3	9	&5#3->X#3j/b9r	P69
move of motor #4	10	&6#4->X#4j/b10r	P75

Table 11: PMAC Motion Programs

3.3.3. PMAC-CPLD memory access

To enable flexible reconfiguration of the PMAC interface board, it is equipped with a CPLD IC (see 3.4). In a CPLD there are some configuration and status register 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)

3.3.3.1. Read process

To read from a given CPLD, register user must do the following:

M48=ADDRESS ;set the CPLD ADDRESS bits from which we are reading

;timer (wait for i.e. 1ms)

M58 ;read 8bit DATA that is accessible on M58 variable

3.3.3.2. Write process

To write to a given CPLD register user must do the following:

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 i.e. 1ms)

M48=$D4 ;set the ADDRESS bits to 0xD4 which signals 'write command' to CPLD logic

3.4. 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 general purpose input output pins, which can have user defined relation among them. For example, logical functions between signals or state machines can be applied.

Because additional signals (besides PMAC natively supported ones) are required, these additional signals are created using CPLD. BRAKE and DEVMOVE signals are one of them. A CPLD implement registers that control additionally required signals. The access to registers is implemented via PMAC expansion ports (see 3.3.3).

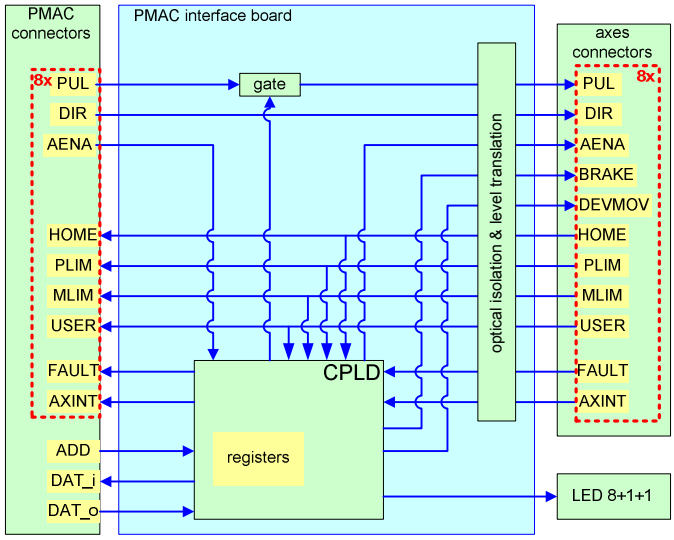

Figure 15: PMAC interface board overview

address	value	R/W	
0x01	PLIM	R	read limit switches, amplifier fault or axis interlock inputs: MSB: axis-8 → LSB: axis-1 1 = input opened, 0 = input closed
0x02	MLIM	R	
0x03	HOME	R	
0x04	USER	R	
0x05	FAULT	R	
0x06	AXINT	R	
0x07	reg1	R/W	axes red LEDs: LSB: axis-1 → MSB: axis-8 1 = LED on, 0 = LED off
0x08	reg2	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	VHDL version	R	read version of the CPLD configuration
0x0A	reg3	R/W	BRAKE output: MSB: axis-8 → LSB: axis-1 1 = brake is off, 0 = brake is on this signal should be only controlled via PMAC, see 3.3.2
0x0B	reg4	R/W	DEVMOV output : MSB: axis-8 → LSB: axis-1 1 = output is closed, 0 = output is opened

Table 12: 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`** (see 3.2)

