DIN EN ISO 9001:2000
certified

ADDI-DATA GmbH
Airpark Business Center
Airport Boulevard B210
77836 Rheinmünster
Germany

Technical support:
+49 7229 1847– 0

Technical description

PA 1500

Digital I/O board, optically isolated

Edition: 08.07 - 05/2008

Product information

This manual contains the technical installation and important instructions for correct commissioning and usage, as well as production information according to the current status before printing.
The content of this manual and the technical product data may be changed without prior notice.
ADDI-DATA GmbH reserves the right to make changes to the technical data and the materials included herein.

Warranty and liability

The user is not permitted to make changes to the product beyond the intended use, or to interfere with the product in any other way.
ADDI-DATA shall not be liable for obvious printing and phrasing errors. In addition, ADDI DATA, if legally permissible, shall not be liable for personal injury or damage to materials caused by improper installation and/or commissioning of the board by the user or improper use, for example, if the board is operated despite faulty safety and protection devices, or if notes in the operating instructions regarding transport, storage, installation, commissioning, operation, thresholds, etc. are not taken into consideration. Liability is further excluded if the operator changes the board or the source code files without authorisation and/or if the operator is guilty of not monitoring the permanent operational capability of working parts and this has led to damage.

Copyright

ADDI-DATA software product licence

Please read this licence carefully before using the standard software. The customer is only granted the right to use this software if he/she agrees with the conditions of this licence.
The software must only be used to set up the ADDI-DATA boards.
Reproduction of the software is forbidden (except for back-up and for exchange of faulty data carriers). Disassembly, decompilation, decryption and reverse engineering of the software are forbidden. This licence and the software may be transferred to a third party if this party has acquired a board by purchase, has agreed to all the conditions in this licence contract and the original owner does not keep any copies of the software.

Trademarks

- ADDI-DATA is a registered trademark of ADDI-DATA GmbH.
- Turbo Pascal, Delphi, Borland C, Borland C++ are registered trademarks of Borland Insight Company.
- Microsoft C, Visual C++, Windows XP, 98, Windows 2000, Windows 95, Windows NT, EmbeddedNT and MS DOS are registered trademarks of Microsoft Corporation.
- LabVIEW, LabWindows/CVI, DasyLab, Diadem are registered trademarks of National Instruments Corp.
- CompactPCI is a registered trademark of PCI Industrial Computer Manufacturers Group.
- VxWorks is a registered trademark of Wind River Systems.
-

WARNING

The following risks result from improper implementation and from use of the board contrary to the regulations:

- **Personal injury**

- **Damage to the baord, PC and peripherals**

- **Pollution of the environment**

- **Protect yourself, the others and the environment!**

- **Read carefully the safety precautions (yellow leaflet).**

  If this leaflet is not with the documentation, please contact us and ask for it.

- **Observe the instructions of the manual.**

  Make sure that you do not forget or skip any step. We are not liable for damages resulting from a wrong use of the board.

- **Used symbols:**

IMPORTANT!
designates hints and other useful information.

WARNING!
It designates a possibly dangerous situation.
If the instructions are ignored the board, PC and/or peripheral may be destroyed.

Figures

Tables

1 DEFINITION OF APPLICATION

1.1 Intended use

The **PA 1500** board must be inserted in a PC with ISA slots which is used as electrical equipment for measurement, control and laboratory pursuant to the norm EN 61010-1 (IEC 61010-1). The used personal computer (PC) must fulfil the requirements of IEC 60950-1 or EN 60950-1 and 55022 or IEC/CISPR 22 and EN 55024 or IEC/CISPR 24.

The use of the board **PA 1500** in combination with external screw terminal panels requires correct installation according to IEC 60439-1 or EN 60439-1 (switch cabinet / switch box).

1.2 Usage restrictions

The **PA 1500** board must not to be used as safety related part (SRP).

The board must not be used for safety related functions, for example for emergency stop functions.

The **PA 1500** board must not be used in potentially explosive atmospheres.

The **PA 1500** board must not be used as electrical equipment according to the Low Voltage Directive 2006/95/EC.

1.3 General description of the board

Data exchange between the **PA 1500** board and the peripheral is to occur through a shielded cable. This cable must be connected to the 37-pin SUB-D male connector of the **PA 1500** board

The board has 16 inputs and 16 output channels for processing 24 V digital signals.

An external 24 V supply voltage is necessary to run the output channels. The screw terminal panel **PX 901** and the relay board **PX 8500** allow connecting the 24 V supply voltage through a shielded cable

The use of the board **PA 1500** in combination with external screw terminal panels or relay boards is to occur in a closed switch cabinet.

The installation is to be effected competently. **Check the shielding capacity** of the PC housing and of the cable prior to putting the device into operation.

The connection with our standard cable **ST010** complies with the following specifications:
- metallized plastic hoods
- shielded cable
- cable shield folded back and firmly screwed to the connector housing.

Please only use the board:
- in conditions providing absolute security
- in a closed housing which is adequately protected against environmental influences
- **with the accessories we recommend**

The use of the board according to its intended purpose includes observing all advises given in this manual and in the safety leaflet.

Uses beyond these specifications are not allowed. The manufacturer is not liable for any damages which would result from the non-observance of this clause.

WARNING!
The EMC tests have been carried out in a specific appliance configuration. We guarantee these limit values **only** in this configuration

Make sure that the board remains in its protective blister pack **until it is used**.

Do not remove or alter the identification numbers of the board.
If you do, the guarantee expires.

2 USER

2.1 Qualification

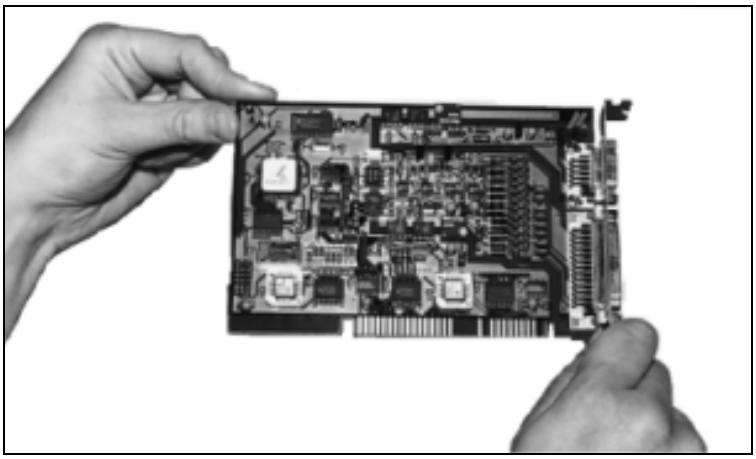
Only persons trained in electronics are entitled to perform the following works:
- installation
- use
- maintenance.

2.2 Country-specific regulations

Consider the country-specific regulations about:
- the prevention of accidents
- electrical and mechanical installations
- radio interference suppression.

3 HANDLING OF THE BOARD

Fig. 3-1: Correct handling



4 TECHNICAL DATA

4.1 Electromagnetic compatibility (EMC)

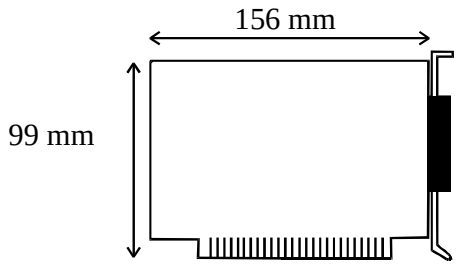
The board **PA 1500** complies with the European EMC directive. The tests were carried out by a certified EMC laboratory in accordance with the norm from the EN 61326 series (IEC 61326). The limit values as set out by the European EMC directive for an industrial environment are complied with.

The respective EMC test report is available on request.

4.2 Physical set-up of the board

The board is assembled on a 4-layer printed circuit card.

Dimensions:



Weight: ..... approx. 150 g
Installation in: ..... XT/AT slot V
Connection to the peripheral: ..... 37-pin SUB-D male connector
Accessories[1]:
Standard cable: ..... **ST010(-S)**, **ST011(-S)**
Screw terminal panel: ..... **PX 901**
Relay output board: ..... **PX 8500** cascadable with cable **ST8500**
See Fig. 8-4: Connection to screw terminal panels and relay output boards

WARNING!
The supply lines must be installed safely against mechanical loads.

1 Not included in the standard delivery.

4.3 Limit values

Max. altitude: ..... 2000 m
Operating temperature: ..... 0 to 60°C
Storage temperature: ..... -25 to 70°C
Relative humidity at indoor installation
50% at +40 °C
80% at +31 °C

Minimum PC requirements:
ISA bus interface: ..... 8 MHz
Operating system: ..... Windows 2000, XP

Energy requirements:
- Operating voltage of the PC: ..... 5 V ± 5%
- Current consumption (without load): ..... typ. see table ± 10%

	PA 1500
+ 5 V from PC	229 mA

24 V digital inputs
Input type: ..... common ground
in accordance with IEC1131-2
Number of inputs: 16
Interruptible inputs: 14
Interrupt lines: ..... IRQ 3, 5 for XT;
IRQ 10, 11, 12, 14, 15 for AT
Nominal voltage: ..... 24 VDC
Input current at nominal voltage: ..... 6 mA
Logic input levels:
U_H[1)] max.:..... 30 V, current 9 mA typ.
U_H min.: ..... 19 V, current 3.5 mA typ.
U_L[2)] max.: ..... 14 V, current 0.75 mA typ.
U_L min.:..... 0 V, current 0 mA typ.
Signal delay: ..... 70 µs (at nominal voltage)
Maximum input frequency: ..... 5 kHz (at nominal voltage)

[1] U_H: input voltage which corresponds to logic "1"

[2] U_L: input voltage which corresponds to logic "0"

24 V digital outputs

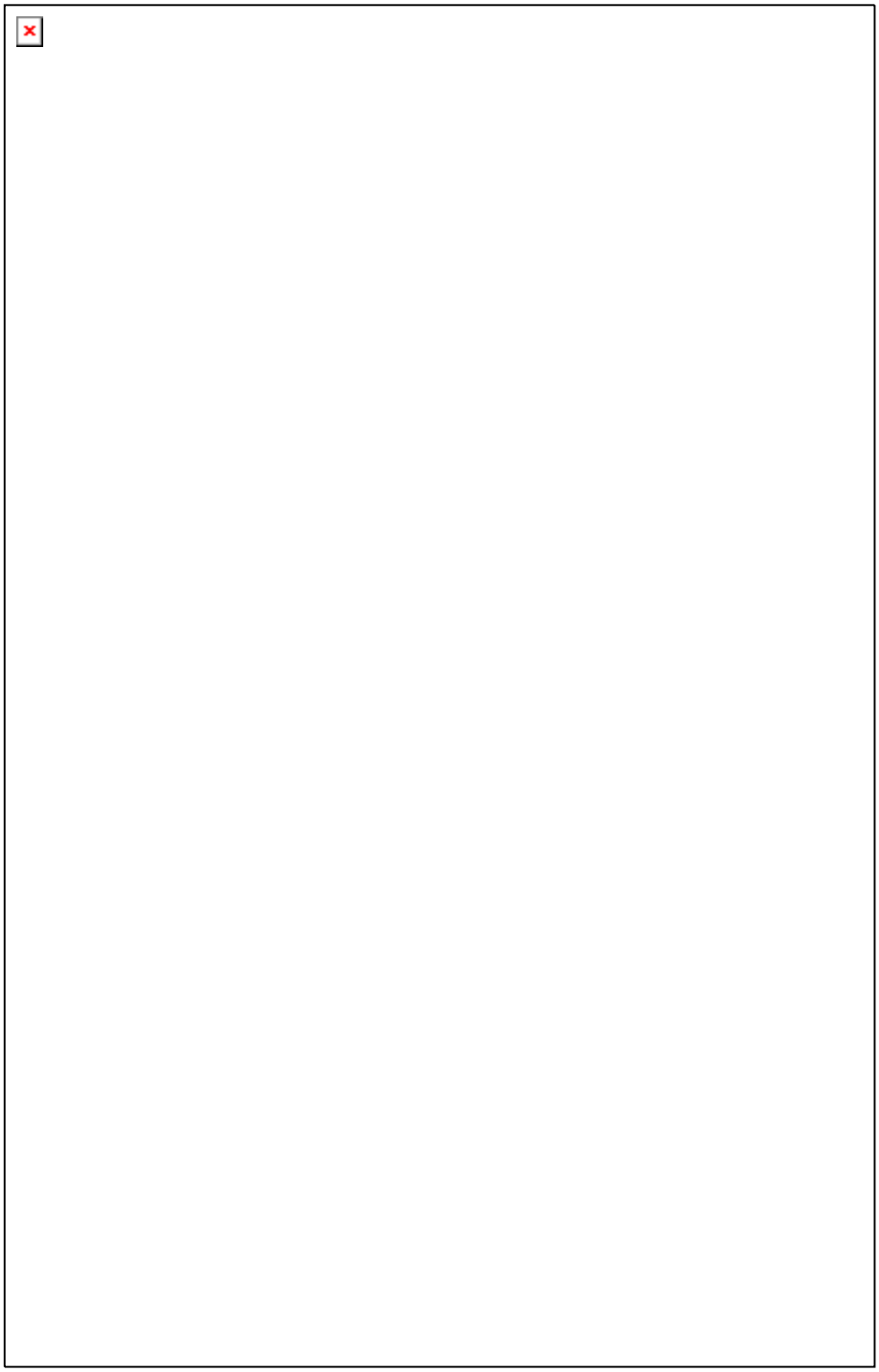
Output type:	high side (load at ground)
Number of outputs:	16
Nominal voltage:	24 VDC
Supply voltage range:	10 V to 36 VDC (min. 5 V)
Maximum output current for the 16 outputs:	3 A typ. (protected by self resetting fuse)
Maximum output current / output:	500 mA
Short-circuit current / output; shut-down at R_{load} < 0,1 R:	24 V, 1,5 A max. (switches off the output)
R_{DS} ON resistance:	0,4 R max.
Switch ON time at 24 V, Rload, 500 mA:	120 µs typ.
Switch OFF time at 24 V, Rload, 500 mA:	40 µs typ.
Overtemperature:	170°C (switches off the components= 4 outputs).
Temperature hysteresis:	20°C

Safety

Optical isolation (DIN VDE 0411-100):	1000 V (from the PC to the external peripheral).
Shut down logic:	When the 24 V ext. voltage supply drops below 5 V, the outputs are switched off. Diagnostic through status bit or interrupt to the PC
Counters or timers:	3
Watchdog:	Timer-programmable. Resets all the outputs if no software trigger has happened. Times from 9 µs to 37 s are available.

4.4 Component scheme

Fig. 4-1: Component scheme



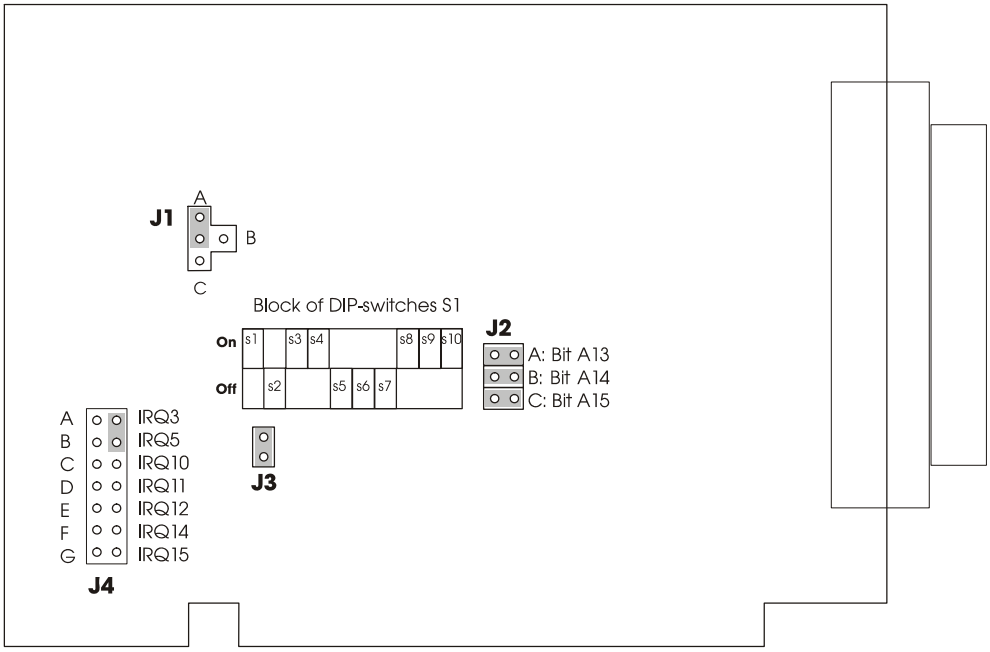
5 SETTINGS OF THE BOARD

IMPORTANT!
Do observe the safety precautions (yellow leaflet)!

5.1 Settings at delivery

5.1.1 Jumper location at delivery

Fig. 5-1: Jumper location on the board (settings at delivery)



5.1.2 Boards settings

IMPORTANT!
J1-A. It means that jumper J1 is set in position A .

J1 Selection of the time base for the timers or the watchdog.
J1-A Timer: 111.5 kHz ± 1 %
Watchdog: 8.94 µs to 586 ms
Settings at delivery
J1-B Timer: 3.45 kHz ± 1 %
Watchdog: 286.6 µs to 18.76 s
J1-C Timer: 1.75 kHz ± 1 %
Watchdog: 573.2 µs to 37.52 s

J2 Address decoding logic
J2-A Address bit A13, decoded to 0
J2-B Address bit A14, decoded to 0
J2-C Address bit A15, decoded to 0
All positions are set at delivery (J2-A to J2-C)

J3 Selection of the data bus width
J3 set 16-bit data bus access on the addresses Base +0, Base +2
J3 open 8-bit data bus access on the addresses Base +0, Base +2

J4 Selection of an interrupt line to the PC bus
No interrupt line is selected at delivery.

IMPORTANT!
IRQ 10 is almost always free on the PC.

5.1.3 Setting the base address through DIP switches

WARNING!
If the base address set is wrong, the board and/or the PC may be destroyed

At delivery the base address is set to 0390H.

- **Check if the base address is free on your PC.**
- **Check if the required address range is not already used by the PC or by another inserted board.**

Windows NT

To check it, open Start/Programs/Administrative tools (common)/Windows NT diagnostics. Click on "I/O Port".

Windows XP/2000/9x

To check it, start the device manager under Start/Settings/Control Panel/ System/Hardware. Set the view to "resources by type".

If the set base address is not displayed, it is then free. If it is occupied by another device, you have to set a new base address through the block of DIP switches of the board.

Decoding the base address

The base address is decoded in steps of each time 8 I/O addresses.

The base address can be selected between 0100H and 0FFFFH within the PC I/O address space.

In table 6-1 the address 0390H is decoded. (Settings at delivery).

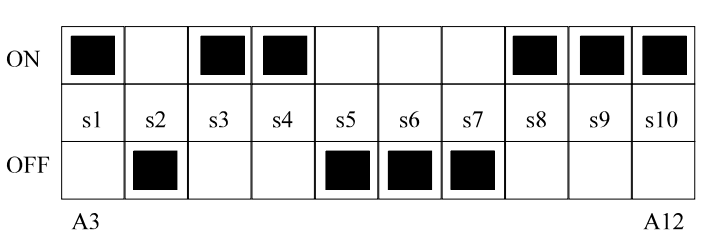
Table 5-1: Decoding table (0390H)

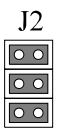
	MSB															LSB
Decoded address bus	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Hex base address to be set	0				3				9				0			
Binary base address to be set	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0
DIP switches S1 Logic "0"= ON Logic "1" = OFF	–	–	–	s10 ON	s9 ON	s8 ON	s7 OFF	s6 OFF	s5 OFF	s4 ON	s3 ON	s2 OFF	s1 ON	X	X	X
Jumper field J2 ON = jumper set OFF = jumper open	J2-C ON	J2-B ON	J2-A ON	–	–	–	–	–	–	–	–	–	–	X	X	X

X : Decoded address range of the board (8 I/O addresses)
– : Not selectable through this component

Fig. 5-2: Block of DIP switches S1

IMPORTANT!
You will find the switch **s1 on the left** of the DIP switches! See Fig. 5-1





6 INSTALLATION OF THE BOARD

IMPORTANT!
Do observe the safety precautions (yellow leaflet)!

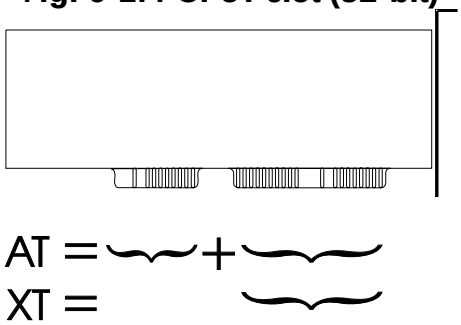
6.1 Opening the PC

- **Switch off your PC and all the units connected to the PC**
- **Pull the PC mains plug from the socket.**
- **Open your PC as described in the manual of the PC manufacturer.**

6.2 Selecting a free slot

Insert the board in a free ISA XT/AT slot.

Fig. 6-1: PCI-5V slot (32-bit)



Remove the back cover of the selected slot according to the instructions of the PC manufacturer. Keep the back cover. You will need it if you remove the board

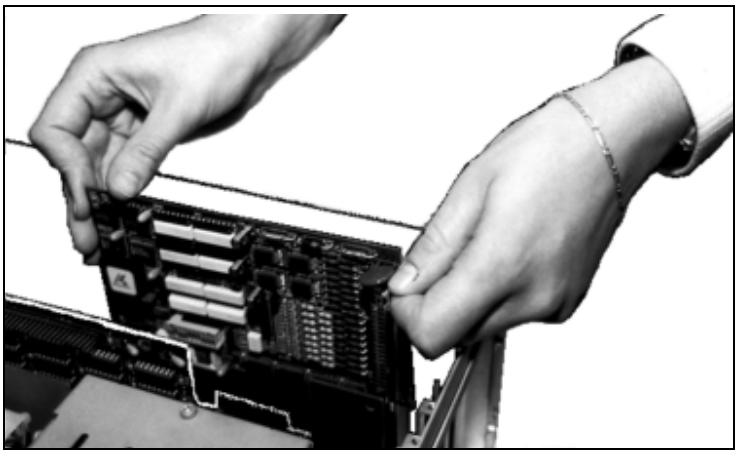
Discharge yourself from electrostatic charges.

Take the board out of its protective pack.

6.3 Plugging the board into the slot

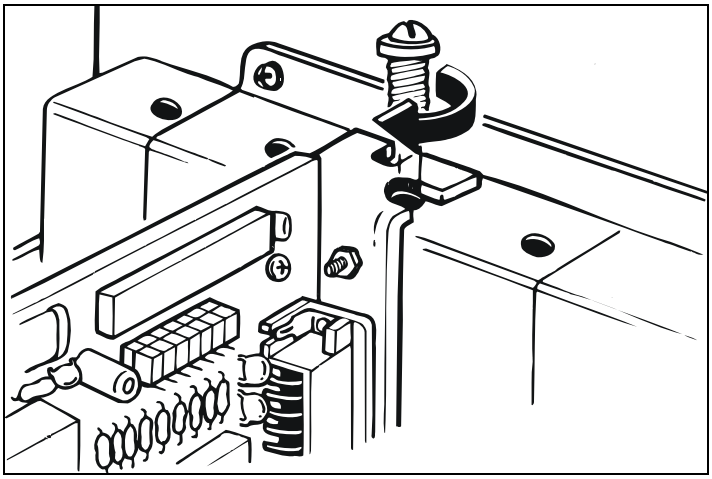
- Insert the board **vertically into the chosen slot.**

Fig. 6-2: Inserting the board



♦ Fasten the board **to the rear of the PC housing with the screw which was fixed on the back cover.**

Fig. 6-3: Fastening the board at the back cover



♦ **Tighten all the loosen screws.**

6.4 Closing the PC

♦ **Close your PC as described in the manual of the PC manufacturer.**

7 SOFTWARE

In this chapter you will find a description of the delivered software and its possible applications.

IMPORTANT!
Further information for installing and uninstalling the different drivers is to be found in the delivered description **"Installation instructions for the ISA bus".**

A link to the corresponding PDF file is available in the navigation pane (Bookmarks) of Acrobat Reader.

The board is supplied with a CD-ROM (CD1) containing
- the driver and software samples for Windows NT 4.0 and Windows XP/2000/98,
- the ADDIREG registration program for Windows NT 4.0 and Windows XP/2000/98.

7.1 Board registration with ADDIREG

The ADDIREG registration program is a 32-bit program for Windows NT XP/2000/NT 4.0/ 9x. The user can register all hardware information necessary to operate the ADDI-DATA PC boards.

IMPORTANT!
If you use one or several resources of the board, you cannot start the ADDIREG program.

7.1.1 Installing a new board

Fig. 7-1: ADDIREG registration program (example)

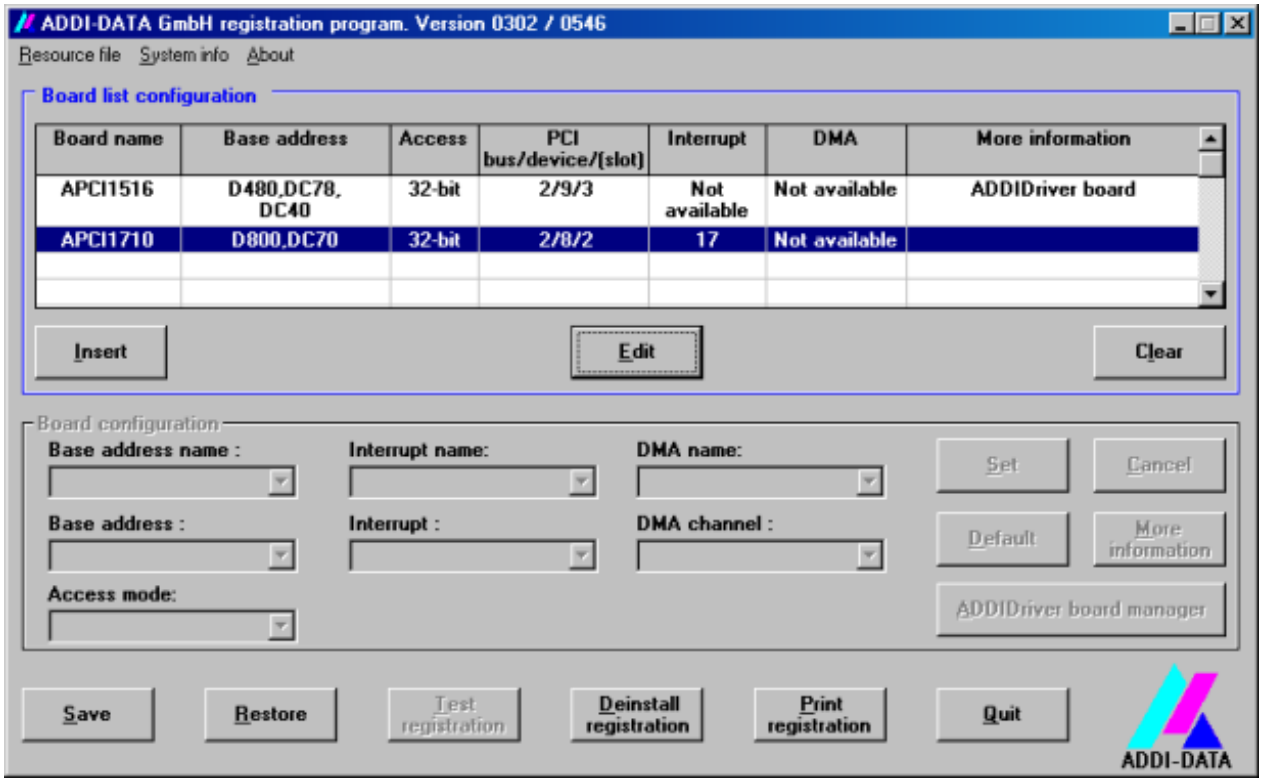


Table:

Board name:
Names of the different registered boards (e.g.: APCI-1710).

Base address:
Selected base address of the board. For PCI boards the base address is allocated through BIOS.

WICHTIG!
The base address set in ADDIREG must correspond to the one set through DIP switches.

Access:
Selection of the access mode for the ADDI-DATA digital boards.
Access in 8-bit or 16-bit or 32-bit mode.

PCI bus/device/(slot):
Number of the used PCI bus, slot, and device. If the board is no PCI board, the message "NO" is displayed.

Interrupt:
Used interrupt of the board. If the board supports no interrupt, the message "Not available" is displayed. **For PCI boards the interrupt is allocated through BIOS.**

WICHTIG!
The interrupt set in ADDIREG must correspond to the one set through jumper.

ISA DMA (ISA boards only):
Indicates the selected DMA channel or "Not available" if the board uses no DMA or if the board is no ISA board.

More information:
Additional information like the identifier string or the installed COM interfaces. It also displays whether the board is programmed with ADDIDRIVER or if a **PCI DMA** memory is allocated to the board.

Text boxes:

Base address name:
Description of the used base addresses for the board. Select a name through the pull-down menu. The corresponding address range is displayed in the field below (Base address).

Base address:
In this box you can select the base addresses of your PC board. The free base addresses are listed. The used base addresses do not appear in this box.

Interrupt name:
Description of the used IRQ lines for the board. Select a name through the pull-down menu. The corresponding interrupt line is displayed in the field below (Interrupt).

Interrupt:
Selection of the interrupt number which the board uses.

DMA name (for ISA boards only):
When the board supports 2 DMA channels, you can select which DMA channel is to be changed.

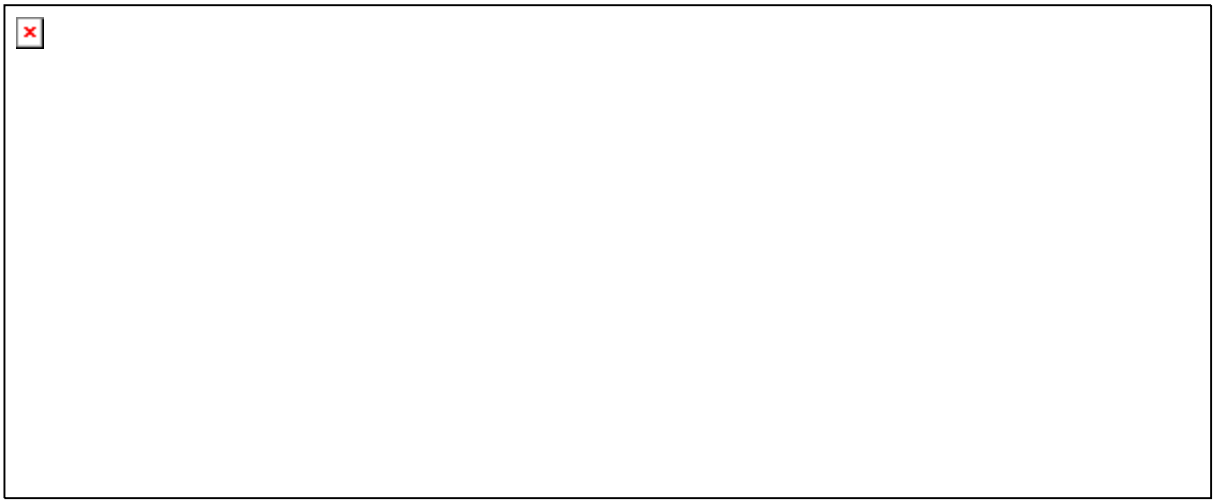
DMA channel (for ISA boards only):
Selection of the used DMA channel.

Buttons:

Edit:
Selection of the highlighted board with the different parameters set in the text boxes.

Insert:
When you want to insert a new board, click on "Insert". The following dialog window appears:

Fig. 7-2: Selecting a new board



All boards you can register are listed on the left. Select the wished board. (The corresponding line is highlighted).
On the right you can read technical information about the board(s).
Activate with "OK"; You come back to the former screen.

Clear:
You can delete the registration of a board. Select the board to be deleted and click on "Clear".

Set:
Sets the parameterized board configuration. The configuration should be set before you save it.

Cancel:
Reactivates the former parameters of the saved configuration.

Default:
Sets the standard parameters of the board.

More information (not available for the boards with ADDIPACK)
You can change the board specific parameters like the identifier string, the COM number, the operating mode of a communication board, etc...
If your board does not support these information, you cannot activate this button.

ADDIDriver Board Manager (only for the boards with ADDIPACK):
Under Edit/ADDIDriver Board Manager you can check or change the current settings of the board set through the ADDEVICE Manager.
ADDevice Manager starts and displays a list of all resources available for the virtual board.

Save:
Saves the parameters and registers the board.

Restore:
Reactivates the last saved parameters and registration.

Test registration:
Controls if there is a conflict between the board and other devices.
A message indicates the parameter which has generated the conflict. If there is no conflict, "OK" is displayed.

Deinstall registration:
Deinstalls the registrations of all board listed in the table.

Print registration:
Prints the registration parameter on your standard printer.

Quit:
Quits the ADDIREG program.

7.1.2 Registering a new board

IMPORTANT!
To register a new board, you must have administrator rights.
Only an administrator is allowed to register a new board or change a registration.

♦ **Call up the ADDIREG program.**

Fig. 7-1 is displayed on the screen.

♦ **Click on "Insert".**

♦ **Select the wished board.**

♦ **Click on "OK".**
The default address, interrupt, and the other parameters are automatically set in the lower fields. The parameters are listed in the lower fields.
If the parameters are not automatically set by the BIOS, you can change them. Click on the wished scroll function(s) and choose a new value.
Activate your selection with a click.

♦ **Once the wished configuration is set, click on "Set".**

♦ **Save the configuration with "Save".**

You can test if the registration is "OK".
This test controls if the registration is right and if the board is present.
If the test has been successfully completed you can quit the ADDIREG program.
The board is initialised with the set parameters and can now be operated.

In case the registration data is to be modified, it is necessary to boot your PC again. A message asks you to do so. When it is not necessary you can quit the ADDIREG program and directly begin with your application.

7.1.3 Changing the registration of a board

IMPORTANT!
To change the registration of a board, you must have administrator rights. Only an administrator is allowed to register a new board or change a registration.

♦ **Call up the ADDIREG program.**

♦ **Select the board to be changed.**
The board parameters (Base address, DMA channel, ..) are listed in the lower fields.

♦ **Click on the parameter(s) you want to set and open the scroll function(s).**

♦ **Select a new value.**

♦ **Activate it with a click. Repeat the operation for each parameter to be modified.**

♦ **Once the wished configuration is set, click on "Set".**

♦ **Save the configuration with "Save".**
You can test if the registration is "OK".
This test controls if the registration is right and if the board is present.
If the test has been successfully completed you can quit the ADDIREG program.
The board is initialised with the set parameters and can now be operated.

In case the registration data is to be modified, it is necessary to boot your PC again. A message asks you to do so. When it is not necessary you can quit the ADDIREG program and directly begin with your application.

7.2 Questions and software downloads on the web

Do not hesitate to e-mail us your questions.
per e-mail: info@addi-data.de or
hotline@addi-data.de

Free downloads of standard software
You can download the latest version of the software for the board **PA 1500**
http://www.addi-data.com

IMPORTANT!
Before using the board or in case of malfunction during operation, check if there is an update of the product (technical description, driver). The current version can be found on the internet or contact us directly.

8 CONNECTING THE PERIPHERAL

8.1 Connector pin assignment

Fig. 8-1: 37-pin SUB-D male connector

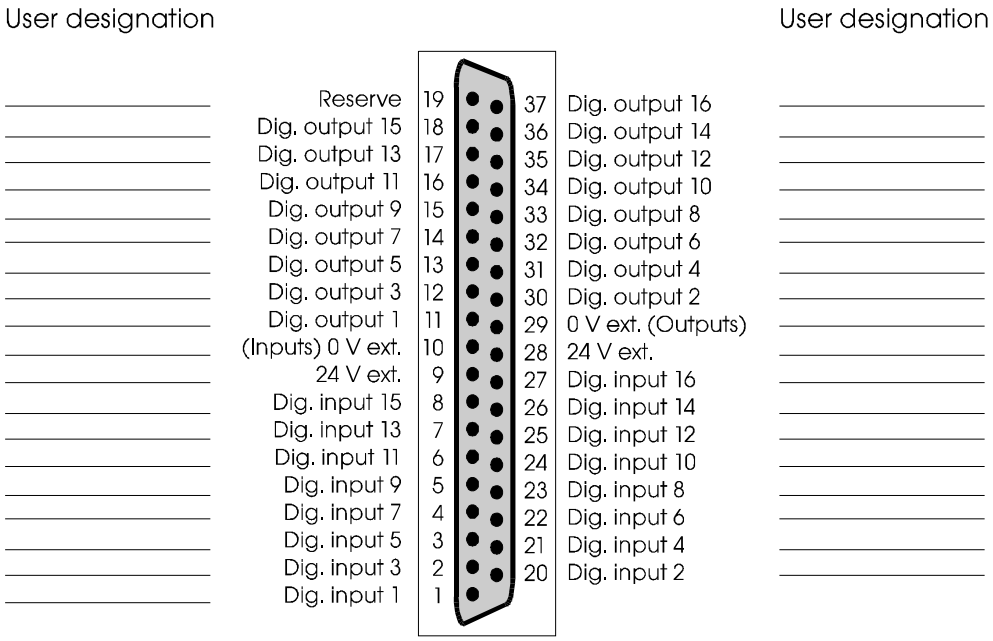
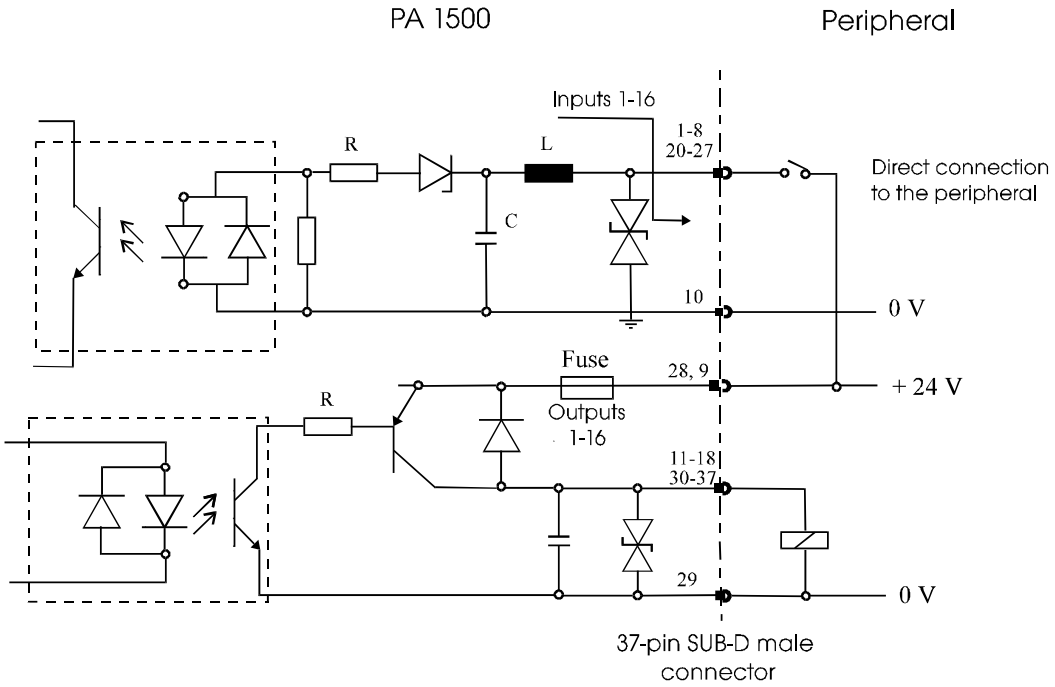


Fig. 8-2: Connection principle



8.2 Connection examples

Fig. 8-3: Connection example

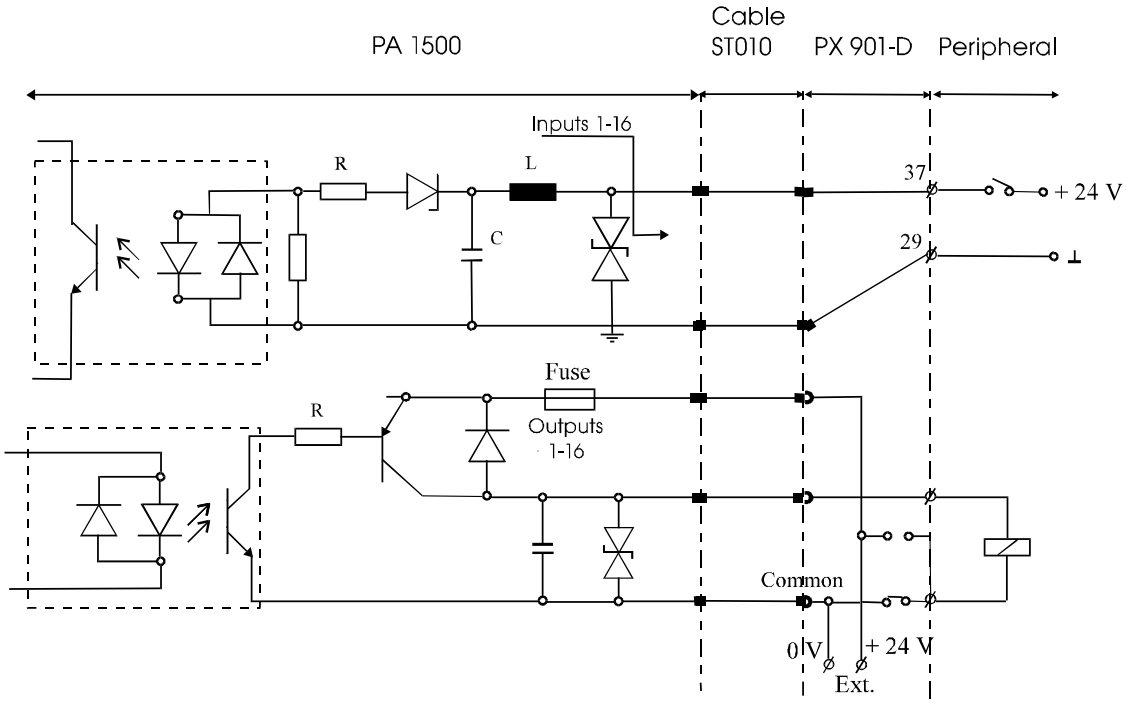
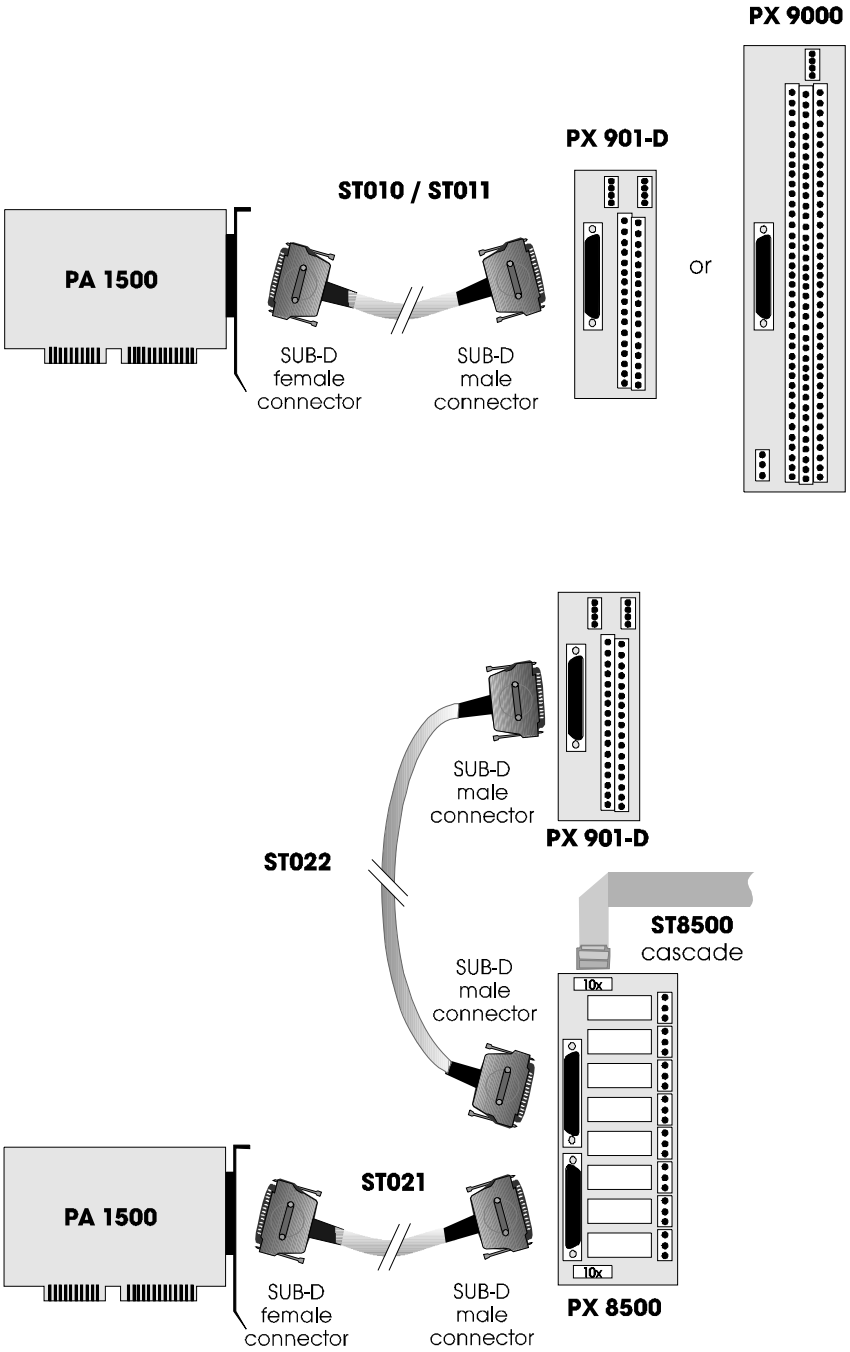


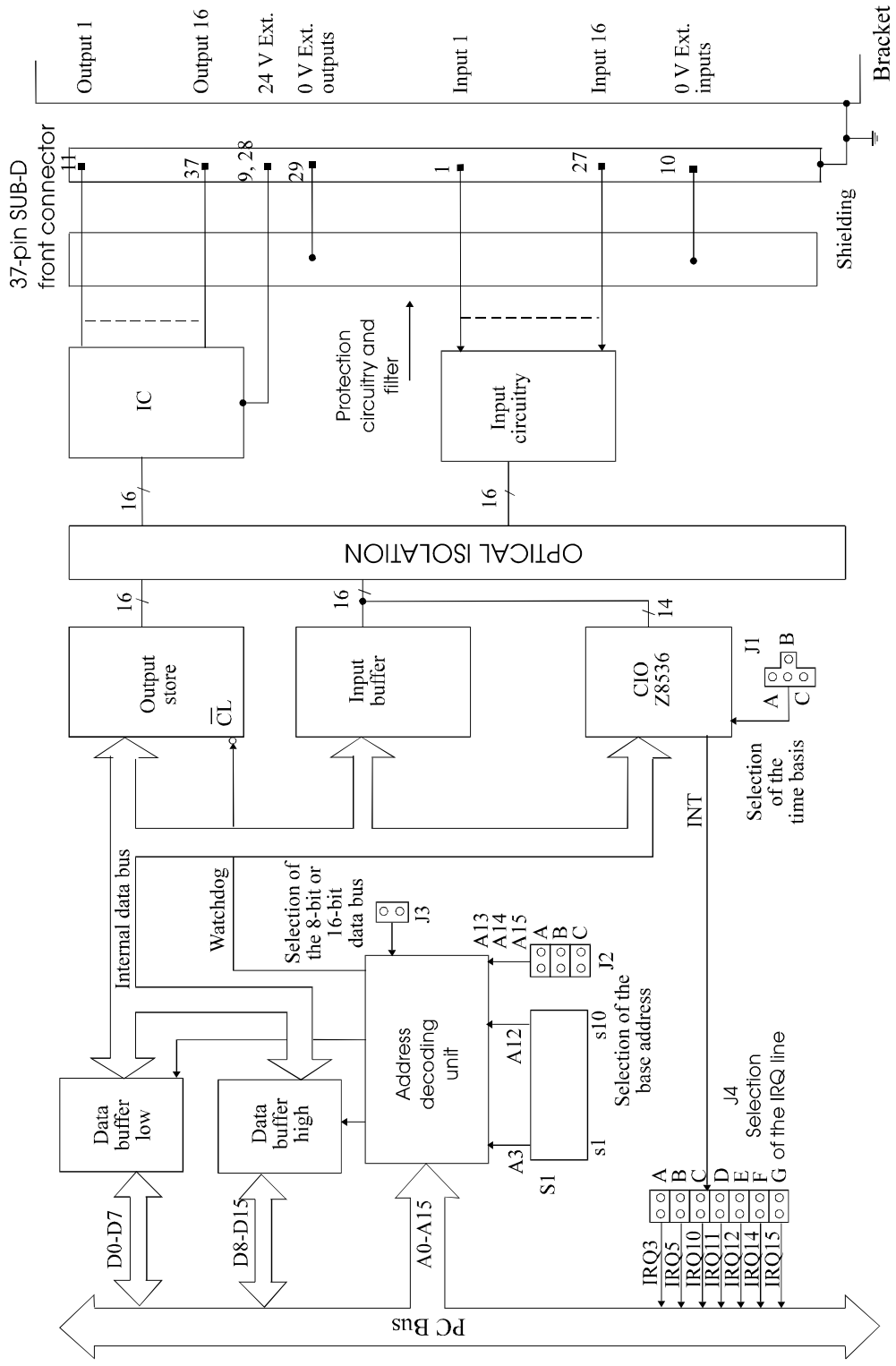
Fig. 8-4: Connection to screw terminal panels and relay output boards



9 FUNCTIONS OF THE BOARD

9.1 Block diagram

Fig. 9-1: Block diagram of the PA 1500



9.2 General description

The board **PA 1500** is intended for parallel input/output for digital signals in 24 V industrial environment.
The peripheral and the system have a simultaneous optical isolation.

The board offers:
- 16 digital inputs: 14 are interruptible.
- 3 counters (or timers): programmable by software
- 1 timer: can be used as a watchdog for the outputs.
- 16 digital outputs are available:
- short-circuit current,
- protection against overtemperature,
- small On resistor,
- wide supply voltage range,
- the outputs are switched off if the voltage drops below the limit value (5 V).

The base address is set with a 10-pin block of DIP switches and jumper J2.
Decoding occurs on the 64 KB I/O address range of the PC.
The board requires 8 I/O addresses within the I/O address range of the PC.

Available interrupt lines:
IRQ3, IRQ5 on XT
IRQ10, IRQ11, IRQ12, IRQ14, IRQ15 on AT

The 16-bit data bus can be switched over to 8-bit data bus.

EMC: design in accordance with CE regulations.

WARNING!
Do not operate the board simultaneously in several modes. Otherwise you may damage the board, PC and/or the peripheral.

Make sure to set only the jumpers required for the respective functions.

9.3 Digital inputs

The board **PA 1500** supplies 16 optically isolated inputs.
The inputs comply with the 24 V industry standard (IEC1131-2):
- logic "1" corresponds to an input voltage > 16 V
- logic "0" corresponds to an input voltage < 15 V.

All the inputs have a common current ground:
0V Ext. (inputs), pin 10 of the 37-pin SUB-D male connector.

The current input is at 6 mA with a nominal voltage of 24 V.

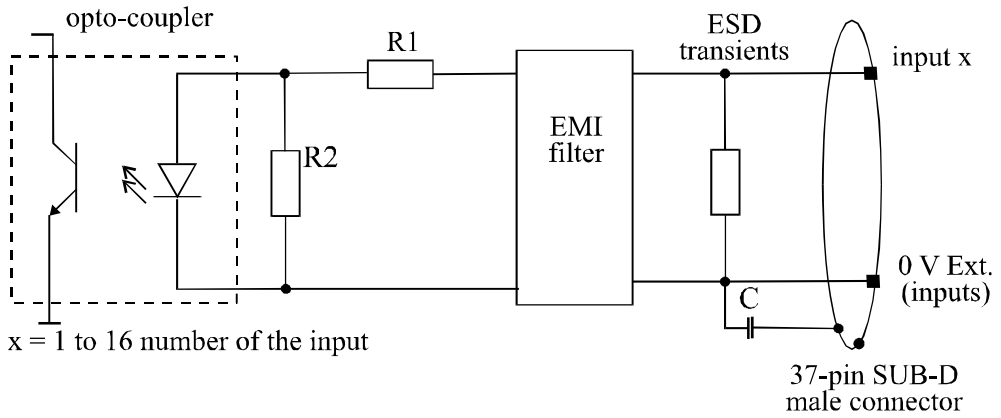
WARNING!
Do you operate all inputs with the same voltage supply? The voltage supply must deliver at least 16 x 6 = 96 mA**.**

The maximum input voltage is 30 V.

Transil diodes, Z diodes, LC filters and opto-couplers protect the system bus from noise emitted by the peripheral. The effects of inductive and capacitive noise are thus reduced.

The board requires no initialisation to read the 24 V digital information. After successful power ON reset, data is immediately available on the board.

Fig. 9-2: Protection circuitry for the inputs



9.3.1 Read the inputs 1 to 16

Two addresses are available in the I/O address range (only in 8-bit data bus access):
- **Base +0** for the inputs 1 to 8
- **Base +1** for the inputs 9 to 16.

Example with the DEBUG program under DOS:

```
c:> DEBUG (CR)
- i 390 (CR)   (* Read the inputs 1-8          *)
00             (* All the inputs are on logic "0" *)
- i 391 (CR)   (* Read the inputs 9-16         *)
00             (* All the inputs are on logic "0" *)
- q            (* Quit the DEBUG program *)
```

EXAMPLE in BASIC:

```
A = INP(&H390)  ;  (* Read the inputs 1-8*)
B = INP(&H391)  ;  (* Read the inputs 9-16*)
                   (* Test input 4 *)
IF (A and &H08) THEN PRINT" INPUT4 = 1" ELSE
PRINT" INPUT4= 0 "
```

Did you select the 16-bit data bus access? (Jumper J3 is set)
One address is available in the I/O address range:
Base +0 for the inputs 1-16

Example in ASSEMBLER

```
.....
MOV DX, 390        ; Base address on 0390H
IN    AX, DX       ; Read the inputs 1 to 16
.......
```

Example in Pascal

```
Function Read_Inputs ( w_Base : WORD) : WORD;
begin  (*  Read the inputs in the 16-bit data bus width w_Basis := $390 *)
  Read_Inputs := portw[ w_Base ];
end;
```

9.3.2 Special input functions

Interrupt

The inputs 1 to 14 can generate an interrupt.

Inputs 1 to 8: It is possible to declare an **OR** or **AND** event to generate an interrupt.

Inputs 9 to 14: It is possible to declare an **OR** event to generate an interrupt.

Inputs 15 & 16: are not interruptible and are not used in events.

See Boards settings

Counter

Counter 1:	**Input 14** signal input
Counter 2:	**Input 10** signal input
	Input 11 can be used for a „trigger“ function
	Input 12 can be used for a „gate“ function.
Counter 3	**Input 15** signal input
	Input 16 can be used as a „gate“ function.

Jumper

♦ **Select the data bus access with jumper J3 (See 5.1.2)**

J3 is set :	16-bit access
J3 is open:	8-bit access

9.3.3 Digital outputs

The board **PA 1500** supplies 16 optically isolated outputs. The outputs comply with the 24 V industry standard (IEC1131-2)

The positive logic is used

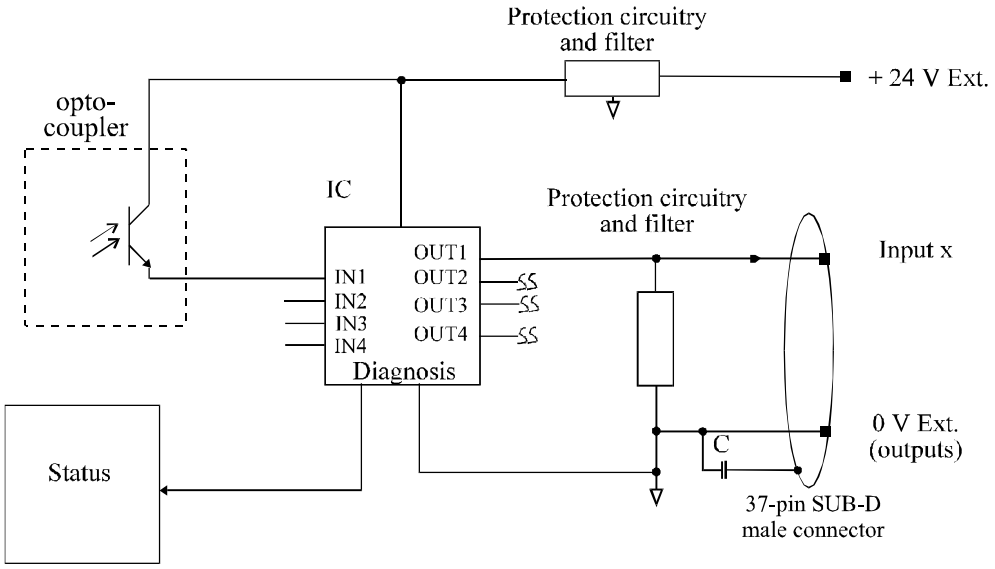
- logic "1": sets the output by software (switch on ON),
- logic "0": resets the output (switch on OFF).

The outputs switch the **+24V ext.** outside to the load. One end of the load is connected with the ground of 0V EXT (outputs).
All outputs have a common ground: 0V ext. (outputs) pin 29 of the 37-pin SUB-D male connector.

WARNING!
Do you use all outputs with the same voltage supply?
The voltage supply must deliver at least the power required for your application.

The maximum supply voltage is 36 V. Each output can switch 500 mA current. But the current is limited for all the outputs on approx. 3 A by a self-resetting fuse.

Fig. 9-3: Protection circuitry for the outputs



x = 1 to 16 Input number

Features of the outputs

- Short-circuit current for the 16 outputs
- Protection against overtemperature: shut down logic. Each group of 4 outputs is switched off: 1 to 4, 5 to 8, 9 to 12, 13 to 16.
- The outputs are switched off if the ext. supply voltage drops below 5 V.
- Diagnostic report through status register or interrupt to the PC in case of short-circuit, overtemperature, voltage drop or watchdog.
- Transorb diodes, C filters and opto-couplers filter noise from the peripheral to the system bus. Thus the effects of inductive and capacitive noise are reduced. Possible noise emissions are also reduced by C filters.

The board requires no initialisation to output the 24V digital information. You can program the outputs immediately after successful power ON reset

State after power ON reset : all the outputs are reset (switch on OFF).

Set the outputs 1 to 16

Two addresses are available in the I/O address range (only in 8-bit data bus access):
- Base +2 for the outputs 1 to 8,
- Base +3 for the outputs 9 to 16.

Example with the DEBUG program under DOS
C:> DEBUG (CR)
- o 392, 01 (CR) (* Set output 1 *)
- o 393, 80 (CR) (* Set output 16 *)
- q (* Quit the DEBUG program *)

Example in Basic
OUT &H392, 1 (* Set output 1 *)
OUT &H393, &H80 (* Set output 16 *)

Did you select the 16-bit data bus access? (Jumper J3 is set)
One address is available in the I/O address range:
Base +2 for the outputs 1-16.

Example in ASSEMBLER
MOV DX, 392 ; Base address on 0392H
MOV AX, 0FFFFH
OUT DX, AX ; Read the outputs 1-->16

Example in Pascal
Procedure Set_Outputs (w_Value : **WORD**);
begin
(* Set the outputs the 16-bit data bus width w_Base := $392 *)
portw[w_Base + 2] := w_Value;
end;

Special functions

Different diagnostic bits are set:
- if a short-circuit has happened on an output,
- if a component has overtemperature
- or if the external voltage supply drops.

These error data are available through an interrupt routine.
See API functions: i_PA1500_SetBoardIntRoutineXX,
i_Pa1500_ResetBoardIntRoutine.

Jumper

♦ **Select the data bus access with jumper J3.**
- J3 is set : 16-bit access
- J3 is open: 8-bit access

9.4 Interrupt

The board **PA 1500** has an interrupt line.
It must be connected through the jumper field J4 to an interrupt line of the PC bus.

The following lines are available:
IRQ3, IRQ5, IRQ10, IRQ11, IRQ12, IRQ14, IRQ15.

Possible interrupt sources :
- Event 1 has occurred (input 1-8),
- Event 2 has occurred (input 9-14),
- Counter/Timer 1 has run down
- Counter/Timer 2 has run down
- Counter/Timer 3 has run down
- Watchdog has run down, the outputs are reset,
- Voltage error (the external voltage supply has dropped below 5 V),
- Short-circuit error, overtemperature error.

The interrupt source information are available on the user program through an interrupt routine (See API functions: i_PA1500_SetBoardIntRoutineXX and i_PA1500_ResetBoardIntRoutine.

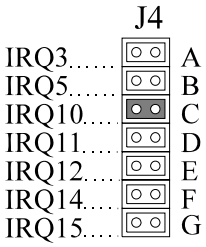
An **event** indicates a change of status (level):
- on an input (ex. "0"-->"1")
- or on several inputs if a logic has been defined between the inputs.

Running-down: if the counter changes from 1 → 0.

Jumper

♦ **Select a free interrupt line on the PC bus with jumper field J4 (See 5.1.2).**

Fig. 9-4: Selection of the interrupt line through jumper field J4



In this example Jumper J4-C is set: IRQ10 is selected

9.5 Counter/timer

On the board **PA 1500** three 16-bit counters/timers are available in the component Z8536 (downwards counting). Each counter/timer can be programmed by software.

If the component Z8536 operates as a counter, the corresponding inputs are used as follows:

9.5.1 Counter

Counter 1:	Input 14	signal input.
Counter 2:	Input 10	signal input
	Input 11	can be used as a "trigger“ function
	Input 12	can be used as a "gate" function.

Counter 3	Input 15	signal input
	Input 16	can be used as a "gate" function.

9.5.2 Timer

If the component Z8536 is used as a timer, the frequency is used as a reference. You set it with jumper J1. "Gate" and "trigger" are possible through the inputs.

Gate

The gate can be driven by software or an input can be set.
The polarity of the input can be programmed. This "gate" stops counting when it is set.

Trigger

The trigger can be driven by software or an input can be set.
The polarity of the input can be programmed. This "trigger" re-loads the counter/timer with the initial counting value.

The following functionalities are available:
- Initialising the counters/timers,
- Starting the counters/timers,
- Stopping the counters/timers,
- Reading the counting value of the counters/timers

The counter/timer 3 has a special function: **Watchdog Timer**.
The function **Watchdog Timer** allows to supervise the software or PC.

The principle is: The counter/timer 3 is programmed as a not re-loadable timer. The timer is started. The outputs are reset when the timer has run down (switch OFF).

The user software must be built in such a way that accesses always occur on the adresses **Base + 2, Base + 3.** You thus avoid that the watchdog runs down.

Jumper

♦ **Select with jumper J1 the input frequency for the timer.**
J1-A : 111.5 kHz ± 1%,
J1-B : 3.45 kHz ± 1%,
J1-C : 1.75 kHz ± 1%.

Data

Approximate watchdog times:
8.94 µs → 5862 ms **J1-A**
286.6 µs → 18.76 s **J1-B**
573.2 µs → 37.52 s **J1-C**

10 STANDARD SOFTWARE

10.1 Introduction

IMPORTANT!
Note the following conventions in the text:

Function: "i_PA1500_SetBoardAddress"
Variable *ui_Address*

Table 10-1: Type Declaration for Dos and Windows 3.1X

	Borland C	Microsoft C	Borland Pascal	Microsoft Visual Basic Dos	Microsoft Visual Basic Windows
VOID	void	void	pointer		any
BYTE	Unsigned char	unsigned char	byte	integer	integer
INT	int	int	integer	integer	integer
UINT	unsigned int	unsigned int	word	long	long
LONG	long	long	longint	long	long
PBYTE	Unsigned char *	unsigned char *	var byte	integer	integer
PINT	int *	int *	var integer	integer	integer
PUINT	Unsigned int *	unsigned int *	var word	long	long
PCHAR	char *	char *	var string	string	string

Table 10-2: Type Declaration for Windows 95/NT

	Borland C	Microsoft C	Borland Pascal	Microsoft Visual Basic Dos	Microsoft Visual Basic Windows
VOID	void	void	pointer		any
BYTE	unsigned char	unsigned char	byte	integer	integer
INT	int	int	integer	integer	integer
UINT	unsigned int	unsigned int	long	long	long
LONG	long	long	longint	long	long
PBYTE	unsigned char *	unsigned char *	var byte	integer	integer
PINT	int *	int *	var integer	integer	integer
PUINT	unsigned int *	unsigned int *	var long	long	long
PCHAR	char *	char *	var string	string	string

10.2 Software functions (API)

10.2.1 Base address

1) i_PA1500_InitCompiler (..)

Syntax:
<Return value> = i_PA1500_InitCompiler
(BYTE b_CompilerDefine)

Parameters:
- Input:

BYTE	b_CompilerDefine	The user has to choose the language under Windows in which he/she wants to program - DLL_COMPILER_C: The user programs in C. - DLL_COMPILER_VB: The user programs in Visual Basic for Windows. - DLL_COMPILER_VB_5: The user programs in Visual Basic 5 for Windows NT or Windows 95. - DLL_COMPILER_PASCAL: The user programs in Pascal or Delphi. - DLL_LABVIEW :The user programs in Labview.

- Output:
No output signal has occurred

Task:
If you want to use the DLL functions, choose the language which you want to program in. This function must be the first to be called up.

IMPORTANT!
This function is only available with a Windows environment.

Calling convention:
ANSI C :
int i_ReturnValue;
i_ReturnValue = i_PA1500_InitCompiler (DLL_COMPILER_C);

Return value:
0: No error
-1: Compiler parameter is wrong

2) i_PA1500_SetBoardAddress (...)

Syntax:

```
<Return value> =    i_PA1500_SetBoardAddress
                                (UINT       ui_BaseAddress,
                                 BYTE       b_AccessMode,
                                 PBYTE      pb_BoardHandle)
```

Parameters:

- Input:

UINT	ui_BaseAddress:	Base address of the **PA 1500** board
BYTE	b_AccessMode:	PA 1500 access mode - PA1500_8BIT: 8-bit access - PA1500_16BIT: 16-bit access

- Output:

PBYTE	pb_BoardHandle:	Handle[1] of board **PA 1500** to use the functions

Task:

Checks if the board **PA 1500** is present and stores the base address.
A handle is returned to the user which allows to use the following functions.
Handles allow to operate several boards.

Return value:

0: No error
-1: Not available base address
-2: Error in the access mode parameter
-3: Board not present
-4: No handle is available for the board (up to 10 handles can be used)
-5: Error by opening the kernel driver under Windows NT / 95

[1] Identification number of the board

3) i_PA1500_CloseBoardHandle (..)

IMPORTANT!
Call up this function each time you want to quit the user program!

Syntax:
<Return value> = i_PA1500_CloseBoardHandle
(BYTEb_BoardHandle)

Parameters:
- Input:
BYTE b_BoardHandle Handle of board **PA 1500**
- Output:
No output signal has occurred

Task:
Releases the board handle. Blocks the access to the board.

Calling convention:
ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1500_CloseBoardHandle    (b_BoardHandle);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong

10.2.2 Interrupt

IMPORTANT!
This function is only available for C/C++ and Pascal for DOS

1) i_PA1500_SetBoardIntRoutineDos (..)

Syntax:

```
<Return value> = i_PA1500_SetBoardIntRoutineDos
                                (BYTE    b_BoardHandle,
                                 BYTE    b_InterruptNbr,
                                 VOID    v_FunctionName
                                       (BYTE       b_BoardHandle,
                                        BYTE       b_InterruptMask
                                        BYTE       b_InputChannelNbr))
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1500**
BYTE	b_InterruptNbr	**PA 1500** interrupt number (3, 5, 10, 11, 12, 14 or 15)
VOID	v_FunctionName	Name of the user interrupt routine

- Output:

No output signal has occurred

Task:

This function must be called up for each **PA 1500** on which an interrupt action is to be enabled.

First calling (first board):
- the user interrupt routine is installed
- interrupts are enabled.

If you operate several boards **PA 1500** which have to react to interrupts, call up the function as often as you operate boards **PA 1500**.
The variable *v_FunctionName* is only relevant **for the first calling**.

From the second call of the function (next board):
- interrupts are enabled. The first board can receive IRQs.

Interrupt

The user interrupt routine is called up by the system when an interrupt is generated.

An interrupt is generated when:
- the counter/timer has run down
- an event is generated
- the watchdog has run down

The following errors are possible:
- overtemperature
- short-circuit
- no voltage is available

If several boards are operated and if they have to react to interrupts, the variable *b_BoardHandle* returns the identification number (handle) of the board which has generated the interrupt.

The user interrupt routine must have the following Syntax:

VOID v_FunctionName (BYTE b_BoardHandle,
BYTE b_InterruptMask,
BYTE b_InputChannelNbr)

v_FunctionName Name of the user interrupt routine

b_BoardHandle Handle of the **PA 1500** which has generated the interrupt

b_InterruptMask Mask of the events which have generated the interrupt.

*b_InputChannelNbr*If an interrupt is generated with a Mask 0000 000**1** and if you use the OR-PRIORITY logic, this variable gives the input number which have generated the interrupt.

Table 10-3: Interrupt mask

Mask	Meaning
0000 000**1**	Event 1 has occurred
0000 00**1**0	Event 2 has occurred
0000 0**1**00	Counter/timer 1 has run down
0000 **1**000	Counter/timer 2 has run down
000**1** 0000	Counter 3 has run down
00**1**0 0000	Watchdog has run down
0**1**00 0000	Voltage error
1000 0000	Short-circuit error

The user can give another name for *v_FunctionName, b_BoardHandle, b_InterruptMask, b_InputChannelNbr.*

Calling convention:
ANSI C :

```
void    v_FunctionName      (unsigned char   b_BoardHandle,
                             unsigned char   b_InterruptMask,
                             unsigned int    b_InputChannelNumber)
        {
        .
        .
        }

int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned char   b_InterruptNbr;

i_ReturnValue = i_PA1500_SetBoardIntRoutineDos
        (b_BoardHandle,
         b_InterruptNbr,
         v_FunctionName );
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: Interrupt already installed
-3: Interrupt number error

IMPORTANT!
This function is only available for Visual Basic DOS

2) i_PA1500_SetBoardIntRoutineVBDos (..)

Syntax:

```
<Return value> = i_PA1500_SetBoardIntRoutineVBDos
                                    (BYTE    b_BoardHandle,
                                     BYTE    b_InterruptNbr)
```

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1500**
BYTE	b_InterruptNbr	**PA 1500** interrupt number (3, 5, 10, 11, 12, 14 or 15)

- Output:
No output signal has occurred

Task:
This function must be called up for each **PA 1500** on which an interrupt action is to be enabled. It installs an user interrupt function in all boards on which an interrupt is to be enabled.

First calling (first board):
- the user interrupt routine is installed
- interrupts are enabled.

If you operate several boards **PA 1500** which have to react to interrupts, call up the function as often as you operate boards **PA 1500**. The variable *v_FunctionName* is only relevant **for the first calling**.

Interrupt
The user interrupt routine is called up by the system when an interrupt is generated.

The following errors are possible
- overtemperature
- short-circuit
- no voltage is available

Controlling the interrupt management
Please use the following functions
"ON UEVENT GOSUB xxxxxxxxx" of Visual Basic for DOS
and
"i_PA1500_TestInterrupt"
This function tests the interrupt of the **PA 1500**. It is used for obtaining the values of *b_BoardHandle* , *b_InterruptMask* and *b_InputChannelNbr*.

Calling convention:
Visual Basic DOS:

```
Dim Shared i_ReturnValue        As Integer
Dim Shared i_BoardHandle        As Integer
Dim Shared i_InterruptMask      As Integer
Dim Shared l_InputChannelNbr    As Integer

IntLabel:
    i_ReturnValue = i_PA1500_TestInterrupt      (i_BoardHandle, _
                                                 i_InterruptMask, _
                                                 i_InputChannelNbr)
    .
    .
    .

    Return

   ON UEVENT GOSUB IntLabel
   UEVENT ON
i_ReturnValue = i_PA1500_SetBoardIntRoutineVBDos    (i_BoardHandle,
                                                     i_InterruptNbr)
```

Return value:
 0: No error
-1: The handle parameter of the board is wrong
-2: Interrupt already installed
-3: Interrupt number error

IMPORTANT!
This function is only available for Windows 3.1 and Windows 3.11

3) i_PA1500_SetBoardIntRoutineWin16 (..)

Syntax:
<Return value> = i_PA1500_SetBoardIntRoutineWin16
(BYTE b_BoardHandle,
BYTE b_InterruptNbr,
VOID v_FunctionName
(BYTE b_BoardHandle,
BYTE b_InterruptMask,
BYTE InputChannelNbr))

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1500**
BYTE	b_InterruptNbr	**PA 1500** interrupt number (3, 5, 10, 11, 12, 14 or 15)
VOID	v_FunctionName	Name of the user interrupt routine

- Output:
No output signal has occurred

Task:
This function must be called up for each **PA 1500** on which an interrupt action is to be enabled.

First calling (first board):
- the user interrupt routine is installed
- interrupts are enabled.

If you operate several boards **PA 1500** which have to react to interrupts, call up the function as often as you operate boards **PA 1500**. The variable *v_FunctionName* is only relevant for the first calling.

From the second call of the function (next board):
- interrupts are allowed.

Interrupt
The user interrupt routine is called up by the system when an interrupt is generated.

An interrupt is generated when:
- the counter/timer has run down
- an event is generated
- the watchdog has run down

The following errors are possible
- overtemperature
- short-circuit
- no voltage is available

If several boards are operated and if they have to react to interrupts, the variable *b_BoardHandle* returns the identification number (handle) of the board which has generated the interrupt.

The user interrupt routine must have the following Syntax:

```
VOID    v_FunctionName      (BYTE  b_BoardHandle,
                             BYTE  b_InterruptMask,
                             BYTE  b_InputChannelNbr)
```

v_FunctionName	Name of the user interrupt routine
b_BoardHandle	Handle of the **PA 1500** which has generated the interrupt
b_InterruptMas	Mask of the events which have generated the interrupt.
b_InputChannelNbr	If an interrupt is generated with a Mask 0000 000**1** and if you use the OR-PRIORITY logic, this variable gives the input number which have generated the interrupt.

Mask	Meaning
0000 000**1**	Event 1 has occurred
0000 00**1**0	Event 2 has occurred
0000 0**1**00	Counter/timer 1 has run down
0000 **1**000	Counter/timer 2 has run down
000**1** 0000	Counter 3 has run down
00**1**0 0000	Watchdog has run down
0**1**00 0000	Voltage error
1000 0000	Short-circuit error

The user can give another name for *v_FunctionName, b_BoardHandle, b_InterruptMask,and b_InputChannelNbr.*

IMPORTANT!
If you use Visual Basic for Windows the following parameters have no meaning. You must use the „i_PA1500_TestInterrupt“ function.

```
VOID        v_FunctionName      (BYTE  b_BoardHandle,
                                 BYTE  b_InterruptMask,
                                 BYTE  b_InputChannelNbr)
```

Calling convention:
ANSI C :

```
void    v_FunctionName      (unsigned char   b_BoardHandle,
                             unsigned char   b_InterruptMask,
                             unsigned char   b_InputChannelNbr)
        {
        .
        .
        }

int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned char   b_InterruptNbr;

i_ReturnValue = i_PA1500_SetBoardIntRoutineWin16
                             (b_BoardHandle,
                              b_InterruptNbr,
                              v_FunctionName );
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: Interrupt already installed
-3: Interrupt number error

IMPORTANT!
This function is only available for 32-Bit Betriebsysteme.

4) i_PA1500_SetBoardIntRoutineWin32 (..)

Syntax:

```
<Return value> = i_PA1500_SetBoardIntRoutineWin32
                                (BYTE     b_BoardHandle,
                                BYTE      b_InterruptNbr,
                                BYTE      b_UserCallingMode,
                                 ULONGul_UserSharedMemorySize,
                                 VOID ** ppv_UserSharedMemory,
                                 VOID      v_FunctionName
                                           (BYTE   b_BoardHandle,
                                            BYTE   b_InterruptMask,
                                            BYTE   b_InputChannelNbr,
                                            BYTE   b_UserCallingMode
                                            VOID * pv_UserSharedMemory))
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1500**
BYTE	b_InterruptNbr	**PA 1500** interrupt number (3, 5, 10, 11, 12, 14 or 15)
BYTE	b_UserCallingMode	PA1500_SYNCHRONOUS_MODE : The user routine is directly called by driver interrupt routine. PA1500_ASYNCHRONOUS_MODE : The user routine is called by driver interrupt thread.
VOID	v_FunctionName	Name of the user interrupt routine
ULONG	ul_UserSharedMemorySize	Determines the size in bytes of the user shared memory. Only used if you have selected PA1500_SYNCHRONOUS_MODE

IMPORTANT!
The size of the User Shared Memory is limited to 63 MB. It Could cause problems if more memory is required.

- Output:

VOID **	ppv_UserSharedMemory	User shared memory address. Only used if you have selected PA1500_SYNCHRONOUS_MODE.

Task:

IMPORTANT: WINDOWS 32-BIT INFORMATION
For Windows NT and Windows 95, 4 running rings (ring 0 to ring 3) are available

The user application operates in ring 3. This ring does not give access to hardware.
- VXD and SYS driver operate in ring 0 and give access to hardware.
- Ring 0 has no direct access to global variable from ring 3. It has to use a shared memory.
- Ring 0 and ring 3 have a pointer that points on this shared memory. These 2 pointers are not configured under the same address.

This function must be called up for each **PA 1500** for which an interrupt is to be enabled.

First calling (first board):
- the user interrupt routine is installed
- interrupts are enabled
- user shared memory is allocated if PA1500_SYNCHROUNOUS_MODE has been selected.

If you operate several boards **PA 1500** which have to react to interrupts, call up the function as often as you operate boards **PA 1500**. The variable *v_FunctionName* is only relevant **for the first calling**.

From the second call of the function (next board):
- interrupts are allowed.

Interrupt
The user interrupt routine is called up by the system when an interrupt is generated.
An interrupt is generated when:
- the counter/timer has run down
- an event is generated
- the watchdog has run down
- The following errors are possible:
- overtemperature
- short-circuit
- no voltage is available

If several boards are operated and if they have to react to interrupts, the variable *b_BoardHandle* returns the identification number (handle) of the board which has generated the interrupt.

User interrupt routine can be called :
- directly by driver interrupt routine (Synchronous mode). The code of the user interrupt routine directly operates in ring 0.
- by the driver interrupt thread (Asynchronous mode). An event is generated and the interrupt thread calls up the user interrupt routine. The code of the user interrupt routine operates in ring 3.
- The driver interrupt thread has the highest priority (31) in the system.

	SYNCHRONOUS MODE
ADVANTAGE	The code of the user interrupt routine is directly called by driver interrupt routine (ring 0). The time between interrupt and the user interrupt routine is reduced.
RESTRICTION	The user cannot debug the user interrupt routine.
	The user routine cannot call Windows API functions.
	The user routine cannot call functions which give access to global variables. The user can yet use a shared memory.
	The user routine can only call **PA 1500** driver functions with the following extension “i_PA1500_KRNL_XXXX“
	This mode is not available for Visual Basic

	ASYNCHRONOUS MODE
ADVANTAGE	The user can debug the user interrupt routine provided he has not programmed in Visual Basic 5
	The user routine can call Windows API functions.
	The user routine can call functions which give access to global variables.
	The user routine can call all **PA 1500** driver functions with the following extension: “i_PA1500_XXXX“
RESTRICTION	The code of the user interrupt routine is called by driver interrupt thread routine (ring 3). The time between interrupt and the user interrupt routine is increased.

Shared memory
If you have selected the PA 1500_SYNCHRONOUS_MODE, you cannot have access to global variables. But you have the possibility to create a shared memory (*ppv_UserSharedMemory*). The user shared memory can have all predefined compiler types or user define types.

The variable ul_UserSharedMemorySize indicates the size in byte of the selected user type. A pointer of the variable *ppv_UserSharedMemory* is given to the user interrupt routine with the variable *pv_UserSharedMemory* .This is not possible for Visual Basic.

The user interrupt routine must have the following Syntax:

```
VOID    v_FunctionName      (BYTE       b_BoardHandle,
                             BYTE       b_InterruptMask,
                             BYTE       b_InputChannelNbr,
                             BYTE       b_UserCallingMode,
                             VOID *     pv_UserSharedMemory)
```

v_FunctionName	Name of the user interrupt routine
b_BoardHandle	Handle of the **PA 1500** which has generated the interrupt
b_InterruptMask	Mask of the events which have generated the interrupt.
b_InputChannelNbr	Is not used. But stays for compatibility reasons.
b_UserCallingMode	PA1500_SYNCHRONOUS_MODE: The user routine is directly called by driver interrupt routine. PA1500_ASYNCHRONOUS_MODE: The user routine is called by driver interrupt thread
pv_UserSharedMemory	Pointer of the user shared memory.

Mask	Meaning
0000 000**1**	Event 1 has occurred
0000 00**1**0	Event 2 has occurred
0000 0**1**00	Counter/timer 1 has run down
0000 **1**000	Counter/timer 2 has run down
000**1** 0000	Counter 3 has run down
00**1**0 0000	Watchdog has run down
0**1**00 0000	Voltage error
1000 0000	Short-circuit error

The user can give another name for *v_FunctionName, b_BoardHandle, b_InterruptMask, b_InputChannelNbr, b_UserCallingMode, pv_UserSharedMemory*.

IMPORTANT!

If you use Visual Basic 4 the following parameters have no meaning. You must used the „i_PA1500_TestInterrupt“ function

```
BYTE      b_UserCallingMode,
ULONG     ul_UserSharedMemorySize,
VOID **   ppv_UserSharedMemory,
VOID      v_FunctionName      (BYTE  b_BoardHandle,
                               BYTE  b_InterruptMask,
                               BYTE  b_InputChannelNbr,
                               BYTE       b_UserCallingMode,
                               VOID *     pv_UserSharedMemory)
```

Calling convention:

ANSI C :

```
   typedef struct
      {
       .
            .
            .
    }str_UserStruct;
    str_UserStruct * ps_UserSharedMemory;
void    v_FunctionName      (unsigned char   b_BoardHandle,
                             unsigned char   b_InterruptMask,
                             unsigned char   b_InputChannelNbr,
                             unsigned char   b_UserCallingMode,
                             void *          pv_UserSharedMemory)
                {
                str_UserStruct * ps_InterruptSharedMemory;
           ps_InterruptSharedMemory = (str_UserStruct *)
pv_UserSharedMemory;
                .
                .
                }
        int             i_ReturnValue;
        unsigned char b_BoardHandle;
        unsigned char b_InterruptNbr;
        i_ReturnValue = i_PA1500_SetBoardIntRoutineWin32
                        (b_BoardHandle,
                         b_InterruptNbr,
                         PA1500_SYNCHRONOUS_MODE,
                         sizeof (str_UserStruct),
                         (void **) &ps_UserSharedMemory,
                         v_FunctionName);
```

Visual Basic 5:

```
Sub     v_FunctionName      (ByVal i_BoardHandle        As Integer,
                             ByVal i _InterruptMask     As Integer,
                             ByVal i _InputChannelNbr   As Integer,
                             ByVal b_UserCallingMode    As Integer,
                             ByVal l_UserSharedMemory As Long)
        End Sub
        Dim i_ReturnValue   As Integer
        Dim i_BoardHandle As Integer
        Dim i_InterruptNbr  As Integer
        i_ReturnValue = i_PA1500_SetBoardIntRoutineWin32
                        (i_BoardHandle,
                         i_InterruptNbr,
                          PA1500_ASYNCHRONOUS_MODE,
                          0,
                          0,
                          AddressOf v_FunctionName)
```

Return value:

0: No error
-1: The handle parameter of the board is wrong
-2: Interrupt already installed
-3: Interrupt number error
-4: Calling mode selection of the user interrupt routine is wrong
-5: No memory available for the user shared memory

5) i_PA1500_TestInterrupt (..)

Syntax:

```
<Return value> =  i_PA1500_TestInterrupt  (PBYTE pb_BoardHandle,
                                           PBYTE pb_InterruptMaske,
                                           PBYTE pb_ChannelNbr)
```

Parameters:

- Input:

No input signal has occurred.

- Output:

PBYTE	pb_BoardHandle	Handle of the board **PA 1500** which has generated the interrupt,
PBYTE	pb_InterruptMaske	Error mask of the event which has generated the interrupt. Several errors can simultaneously occur.

Mask	Meaning
0000 000**1**	Event 1 has occurred
0000 00**1**0	Event 2 has occurred
0000 0**1**00	Counter/timer 1 has run down
0000 **1**000	Counter/timer 2 has run down
000**1** 0000	Counter 3 has run down
00**1**0 0000	Watchdog has run down
0**1**00 0000	Voltage error
1000 0000	Short-circuit error

PBYTE pb_ChannelNbr — Is not used. But stays for compatibility reasons.

Task:
Checks if a board **PA 1500** has generated an interrupt. If yes, the function returns the board handle and the interrupt source.

IMPORTANT!
This function is only available in Visual Basic Dos and Windows.

Calling convention:
ANSI C :

```
unsigned char b_BoardHandle;
unsigned char b_InterruptMaske;
unsigned char b_ChannelNbr;
int           i_Irq;
Irq = i_PA1500_TestInterrupt (&b_BoardHandle,
                              & b_InterruptMaske,
                              & b_ChannelNbr);
```

Return value:
-1: No interrupt
> 0: IRQ number

6) i_PA1500_ResetBoardIntRoutine (..)

Syntax:
<Return value> = i_PA1500_ResetBoardIntRoutine
(BYTE b_BoardHandle)

Parameters:
- Input:
BYTE b_BoardHandle Handle of board **PA 1500**
- Output:
No output signal has occurred

Task:
Stops the interrupt management of board **PA1500.**
Deinstalls the interrupt routine if the management of interrupts of all **PA 1500** is stopped.

Calling convention:
ANSI C :

```
unsigned char b_BoardHandle;

Irq = i_PA1500_ResetBoardIntRoutine (b_BoardHandle);
```

Return value:
0: No error
-1: Handle parameter of the board is wrong
-2: Interrupt routine is not installed

10.2.3 Kernel functions

1) i_PA1500_KRNL_Read16DigitalInput (...)

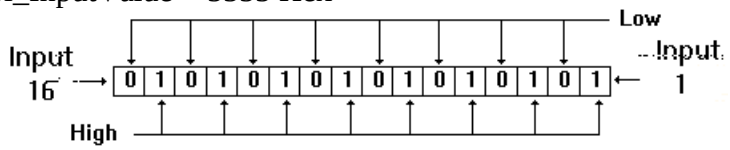
Syntax:
<Return value> = i_PA1500_KRNL_Read16DigitalInput
(UINT ui_BaseAddress,
PLONG pl_InputValue)

Parameters:
- Input:
UINT ui_BaseAddress **PA 1500** base address
- Output:
PLONG pl_InputValue State of the digital inputs of both ports (0 to 65535)

Task:
Indicates the state of both ports. Reads the 16 inputs at once.

Example:

pl_InputValue = 5555 Hex



A voltage is present on the inputs 1, 3, 5, 7, 9, 11, 13 and 15 .
A voltage is not present on the inputs 2, 4, 6, 8, 10, 12, 14 and 16.

Return value:
0: No error

2) v_PA1500_KRNL_Set16DigitalOutputOn (...)

Syntax:
<Return value> = i_PA1500_KRNL_Set16DigitalOutputOn
(UINT ui_BaseAddress,
LONG l_Value)

Parameters:
- Input:

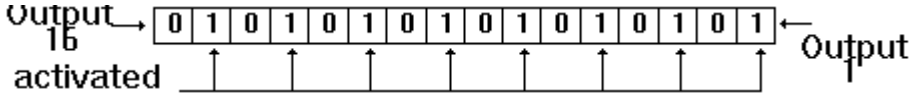
UINT	ui_BaseAddress	**PA 1500** base address
LONG	l_Value	Output value (0 to 65535)

- Output:
No output signal has occurred

Task:
Sets one or several outputs of board **PA 1500**

Example:

l_Value = 5555 Hex



The outputs 1, 3, 5, 7, 9, 11, 13, 15 are set.
The outputs 2, 4, 6, 8, 10, 12, 14, 16 are reset.

Return value:
0: No error

10.2.4 Digital inputs

1) i_PA1500_Read1DigitalInput (...)

Syntax :
<Return value> = i_PA1500_Read1DigitalInput
(BYTE b_BoardHandle,
BYTE b_Channel,
PBYTE pb_ChannelValue)

Parameters:

BYTE	b_BoardHandle	Handle of board **PA 1500**
BYTE	b_Channel	The number of the input to be read (1 to 16)
PBYTE	pb_ChannelValue	State of the digital input: 0 -> low 1 -> high

Task:
Indicates the state of an input. The variable *b_Channel* passes the input to be read (1 to 16). A value is returned with the variable *pb_ChannelValue* : 0 (low) or 1 (high).

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The input number is not between 1 and 16

2) i_PA1500_Read8DigitalInput (...)

Syntax:
<Return value> = i_PA1500_Read8DigitalInput
(BYTE b_BoardHandle,
BYTE b_Port,
PBYTE pb_PortValue)

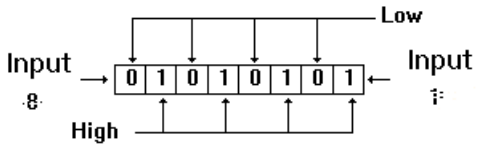
Parameters:

BYTE	b_BoardHandle	Handle of the **PA 1500**
BYTE	b_Port	Number of the input port you want to read (1 or 2)
PBYTE	pb_PortValue	State of the digital input port (0 to 255)

Task:
Indicates the state of a port. The variable b_Port passes the port to be read (1 or 2). A value is returned with the variable *pb_PortValue* .

Example:

b_Port = 1
pb_PortValue = 55 Hex



A voltage is present on the inputs 1, 3, 5, 7
A voltage is not present on the inputs 2, 4, 6, 8.

Return value:
0: No error
-1: Handle parameter of the board is wrong
-2: The parameterized port number is wrong (parameter 1 or 2)

3) i_PA1500_Read16DigitalInput (...)

Syntax:
<Return value> = i_PA1500_Read16DigitalInput
(BYTE b_BoardHandle,
PLONG_ pl_InputValue)

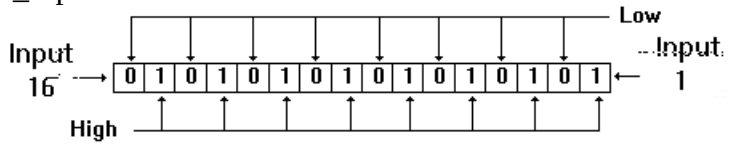
Parameters:

BYTE	b_BoardHandle	Handle of the **PA 1500**
PLONG	pl_InputValue	State of the digital inputs of both ports (0 to 65535)

Task:
Indicates the state of both ports. Reads the 16 inputs at once.

Example:

pl_InputValue = 5555 Hex



A voltage is present on the inputs 1, 3, 5, 7, 9, 11, 13, 15 .
A voltage is not present on the inputs 2, 4, 6, 8, 10, 12, 14, 16.

Return value:
0: No error
-1: The handle parameter of the board is wrong

10.2.5 Digital inputs - events

1) i_PA1500_SetInputEventMask (...)

Syntax:

<Return value> = i_PA1500_SetInputEventMask
(BYTE b_BoardHandle,
BYTE b_PortNbr,
BYTE b_Logik,
PCHAR pc_EventMask)

Parameters:

BYTE	b_BoardHandle	Handle of board **PA 1500**
BYTE	b_Port	Number of the input port to be masked (1 or 2)
BYTE	b_Logik	Event logic Three possibilities for the first port: - PA1500_AND: This logic connects the inputs with an AND logic. - PA1500_OR: This logic connects the inputs with an OR logic. PA1500_OR: This logic connects the inputs with a OR logic.
PCHAR	pc_EventMask	This 8-digit character string (port 1) and 6-digit character string (port 2) define the mask of the event. Each digit indicates the state of the input. The state is identified by one of the following characters: "X": This input is not used for event "0": The input must be on "0" "1": The input must be on "1" "2": The input reacts to a falling edge  "3": The input reacts to a rising edge  "4": The input reacts to both edges Port 1: from the left to the right, the first digit of the character string is input 8 and the last digit is input 1. Port 2: from the left to the right, the first digit of the character string is input 14 and the last digit is input 9.

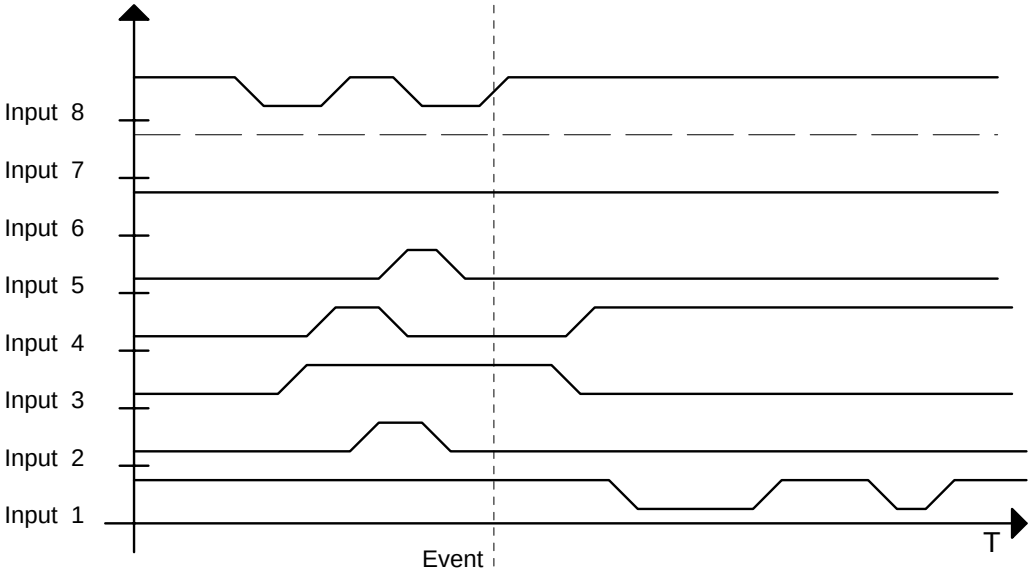
IMPORTANT!
If you use the PA1500_AND logic, you can only use one edge event.

Task:
An event can be generated for each port.
The first event is related to the first 8 inputs (port 1).
The second event is related to the next 6 inputs (port 2).
An interrupt is generated when one or both events have happened.
An event is a change of state (ex. low-> high) on one or several inputs if an link logic has been defined.

Examples:

Example 1:
b_PortNbr = 1
b_Logik = PA1500_AND
pc_EventMask = "3X100101"

An event is generated:
- when the inputs 2, 4 and 5 are on "0".
- when the inputs 1, 3 and 6 are on "1"
- and when a rising edge has been detected at input 8.



Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The parameterized port number is wrong (parameter 1 or 2)
-3: Error with the logic parameter. *b_Logik* has not the expected value
-4: Error with the mask parameter. *pc_EventMask* has not the expected value
-5: Interrupt routine not installed
-6: More than 1 edge event has been declared for an AND logic
-7: The OR PRIORITY logic does not support any edge event

2) i_PA1500_StartInputEvent (...)

Syntax :
<Return value> = i_PA1500_StartInputEvent
(BYTE b_BoardHandle,
BYTE b_PortNbr)

Parameters:

BYTE	b_BoardHandle	Handle of the **PA 1500**
BYTE	b_Port	Number of the input port (1 or 2)

Task:
As soon as the function is called up, it is possible to process an event on one port. First mask the inputs with the following function i_PA1500_SetInputEventMask .

Return value:
0: No error
-1: Handle parameter of the board is wrong
-2: The parameterized port number is wrong
-3: Event has not been initialised with the function "i_PA1500_SetInputEvent".

3) i_PA1500_StopInputEvent (...)

Syntax:
<Return value> = i_PA1500_StopInputEvent
(BYTE b_BoardHandle,
BYTE b_PortNbr)

Parameters:

BYTE	b_BoardHandle	Handle of the **PA 1500**
BYTE	b_Port	Number of the input port (1 or 2)

Task:
Once the function is called up, it is not possible to process an event on one port.

Return value:
0: No error
-1: Handle parameter of the board is wrong
-2: The parameterized port number is wrong
-3: Event has not been initialised with the function "i_PA1500_SetInputEvent"

10.2.6 Digital outputs

1) i_PA1500_SetOutputMemoryOn (...)

Syntax:
<Return value> = i_PA1500_SetOutputMemoryOn
(BYTE b_BoardHandle)

Parameters:
BYTE b_BoardHandle Handle of the **PA 1500**

Task:
Activates the digital output memory.
After calling this function, the outputs you have previously activated with the functions "i_PA1500_SetXDigitalOutputOn" are not reset.
You can reset them with the function "i_PA1500_SetXDigitalOutputOff".

Return value:
0: No error
-1: Handle parameter of the board is wrong

2) i_PA1500_SetOutputMemoryOff (...)

Syntax:
<Return value> = i_PA1500_SetOutputMemoryOff
(BYTE b_BoardHandle)

Parameters:
BYTE b_BoardHandle Handle of the **PA 1500**

Task:
Deactivates the digital output memory.

Return value:
0: No error
-1: Handle parameter of the board is wrong

3) i_PA1500_Set1DigitalOutputOn (...)

Syntax:
<Return value> = i_PA1500_Set1DigitalOutputOn
(BYTE b_BoardHandle,
BYTE b_Channel)

Parameters:
BYTE b_BoardHandle Handle of the **PA 1500**
BYTE b_Channel Number of the output you want to set (1 to 16)

Task:
Sets the output which has been passed with *b_Channel*.
Setting an output means setting an output on high.
***Switching on the digital output memory* (ON)**
see function "i_PA1500_SetOutputMemoryOn (...)
b_Channel= 1
The output 1 is set. The others outputs hold their state.
***Switching off the digital output memory* (OFF)**
see function "i_PA1500_SetOutputMemoryOff (...)
b_Channel= 1
The output 1 is set. The others outputs are reset.
If you have switched off the digital output memory (OFF), all others inputs are set to "0".

Return value:
0: No error
-1: Handle parameter of the board is wrong
-2: Input number is not between 1 and 16

4) i_PA1500_Set1DigitalOutputOff (...)

Syntax :
<Return value> = i_PA1500_Set1DigitalOutputOff
(BYTE b_BoardHandle,
BYTE b_Channel)

Parameters:

BYTE	b_BoardHandle	Handle of the **PA 1500**
BYTE	b_Channel	Number of the output you want to reset (1 to 16)

Task:
Resets the output you have passed with *b_Channel*. Resetting an output means setting on low.

IMPORTANT!
You can use this function only if the digital output memory is ON. See function i_PA1500_SetOutputMemoryOn (..).

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The input number is not between 1 and 16
-3: Digital output memory OFF.
First use the function "i_PA1500_SetDigitalOutputMemoryOn"

5) i_PA1500_Set8DigitalOutputOn (...)

Syntax:
<Return value> = i_PA1500_Set8DigitalOutputOn
(BYTE b_BoardHandle,
BYTE b_Port,
BYTE b_Value)

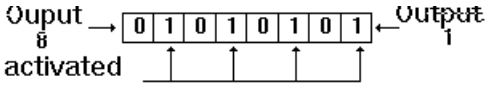
Parameters:

BYTE	b_BoardHandle	Handle of the board **PA 1500**
BYTE	b_Port	Number of the output port (1 or 2)
BYTE	b_Value	Output value (0 to 255)

Task:
Sets one or several outputs of a port. Setting an output means setting on high. If you have switched off the digital output memory (OFF), the inputs are set to "0".

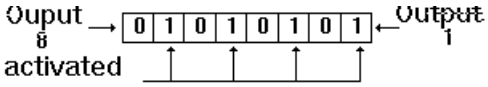
Example:

Switching on the digital output memory (ON)
see function "i_PA1500_SetOutputMemoryOn (...)
b_Port = 1
b_Value = 55 Hex



The outputs 1, 3, 5, 7 are set. The other outputs hold their state.

Switching off the digital output memory (OFF)
see function "i_PA1500_SetOutputMemoryOff (...)
b_Port = 1
b_Value = 55 Hex



The outputs 1, 3, 5, 7 are set. The other outputs are reset.

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The port number is not 1 or 2

6) i_PA1500_Set8DigitalOutputOff (...)

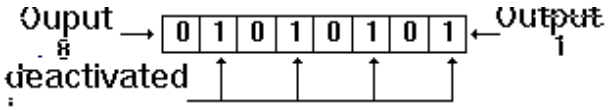
Syntax:
<Return value> = i_PA1500_Set8DigitalOutputOff
(BYTE b_BoardHandle,
BYTE b_Port,
BYTE b_Value)

Parameters:

BYTE	b_BoardHandle	Handle of the **PA 1500**
BYTE	b_Port	Number of the output port (1 or 2)
BYTE	b_Value	Output value (0 to 255)

Task:
Resets one or several outputs of one port . Resetting means setting on high.

Example:
b_Port = 1
b_Value = 55 Hex



The outputs 1, 3, 5, 7 are reset.

IMPORTANT!
You can use this function only if the digital output memory is ON. See function i_PA1500_SetOutputMemoryOn (..).

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The port number is not 1 or 2
-3: The digital output memory is OFF. Please first use the function i_PA1500_SetDigitalOutputMemoryOn

7) v_PA1500_Set16DigitalOutputOn (...)

Syntax:
<Return value> = v_PA1500_Set16DigitalOutputOn
(BYTE b_BoardHandle,
LONG l_Value)

Parameters:

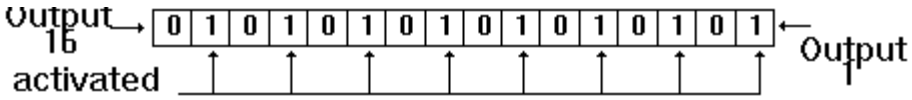
BYTE	b_BoardHandle	Handle of the **PA 1500**
LONG	l_Value	Output value (0 to 65535)

Task:
Sets one or several outputs of board **PA 1500**

Example:

Switching on the digital output memory **(ON)**

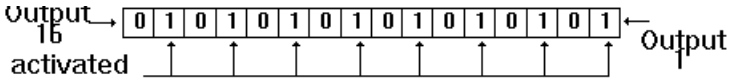
see function "i_PA1500_SetOutputMemoryOn (...)
l_Value = 5555 Hex



The outputs 1, 3, 5, 7, 9, 11, 13, 15 are set. The other outputs hold their state.

Switching off the digital output memory **(OFF)**

see function "i_PA1500_SetOutputMemoryOff (...)
l_Value = 5555 Hex



Outputs 1, 3, 5, 7, 9, 11, 13, 15 are set.Outputs 2, 4, 6, 8, 10, 12, 14, 16 are reset.

Return value:
0: No error
-1: The handle parameter of the board is wrong

8) v_PA1500_Set16DigitalOutputOff (...)

Syntax:
<Return value> = v_PA1500_Set16DigitalOutputOff
(BYTE b_BoardHandle,
LONG_ l_Value)

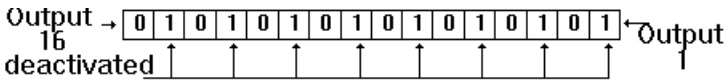
Parameters:

BYTE	b_BoardHandle	Handle of the **PA 1500**
LONG	l_Value	Output value (0 to 65535)

Task:
Resets one or several outputs of board **PA 1500**.

Example:

l_Value = 5555 Hex



The outputs 1, 3, 5, 7, 9, 11, 13, 15 are reset.

IMPORTANT!
You can use this function only if the digital output memory is ON. See function i_PA1500_SetOutputMemoryOn (..).

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The digital output memory is OFF. Please first use the function
"i_PA1500_SetDigitalOutputMemoryOn"

10.2.7 Timer/counter and watchdog

1) i_PA1500_InitTimerCounter1 (...)

Syntax:
<Return value> = i_PA1500_InitTimerCounter1
(BYTE b_BoardHandle,
BYTE b_CounterOrTimerSelect,
LONG l_ReloadValue,
BYTE b_ContinuousOrSingleCycleSelect,
BYTE b_InterruptHandling)

Parameters:

BYTE	b_BoardHandle	Handle of the **PA 1500**
BYTE	b_CounterOrTimerSelect	Select the mode of the first counter/timer - PA1500_TIMER: The first counter/timer is used as a timer - PA1500_COUNTER: The first counter/timer is used as a counter
LONG	l_ReloadValue	This parameter has 2 meanings: If the counter/timer is used as a counter, it loads the start value of the counter. If the counter/timer is used as a timer, it loads the divider factor for the output.
BYTE	b_ContinuousOrSingleCycleSelect	- PA1500_CONTINUOUS: Each time the counting value or timer value is set to "0", *l_ReloadValue* is loaded. - PA1500_SINGLE: If the counter or timer value is set to "0", the counter or timer is stopped.
BYTE	b_InterruptHandling	Interrupts can be generated, when the counter has run down, or when the timer output is on high. With this parameter the user decides if interrupts are used or not. - PA1500_ENABLE: Interrupts are enabled - PA1500_DISABLE: Interrupts are disabled

Task:
Selects the operating mode of the first counter/timer. The user enters its start value.
You have to decide:
- if the counter/timer must execute once or several times the counting operation.
- if the counter/timer is used as a counter or a timer
- and if an interrupt must be generated when the counter/timer has run down.

Return value:
0: No error
-1: The handle-parameter of the board is wrong
-2: The parameter for selecting the counter or the timer is wrong
(PA1500_COUNTER or PA1500_TIMER)
-3: Error with the interrupt selection
(PA1500_ENABLE or PA1500_DISABLE)
-4: The user interrupt routine is not installed
-5: Cycle parameter is wrong
(PA1500_CONTINUOUS or PA1500_SINGLE)

2) i_PA1500_InitTimerCounter2 (...)

Syntax:
<Return value> = i_PA1500_InitTimerCounter2
(BYTE b_BoardHandle,
BYTE b_CounterOrTimerSelect,
LONG_ l_ReloadValue,
BYTE b_ContinuousOrSingleCycleSelect,
BYTE b_HardwareOrSoftwareTriggerSelect,
BYTE b_HardwareOrSoftwareGateSelect,
BYTE b_InterruptHandling)

Parameters:

BYTE	b_BoardHandle	Handle of the **PA 1500**
BYTE	b_CounterOrTimerSelect	Selects the mode of the 2nd counter/timer - PA1500_TIMER: The 2nd counter/timer is used as a timer - PA1500_COUNTER: The 2nd counter/timer is used as a counter
LONG	l_ReloadValue	This parameter has 2 meanings. If the counter/timer is used as a counter, it loads the start value of the counter. If the counter/timer is used as a timer, it loads the divider factor for the output

BYTE	b_ContinuousOrSingleCycleSelect	- PA1500_CONTINUOUS: Each time the counting value or timer value is set to "0", *l_ReloadValue* is loaded. - PA1500_SINGLE: If the counter or timer value is set to "0", the counter or timer is stopped.
BYTE	b_HardwareOrSoftwareTriggerSelect	- PA1500_HARDWARE_TRIGGER: The input 12 is used for the trigger. If this input is on high, the start value is re-loaded. - PA1500_SOFTWARE_TRIGGER: Input 12 has no influence on the trigger
BYTE	b_HardwareOrSoftwareGateSelect	- PA1500_HARDWARE_GATE: The input 13 is used for the gate. If this input is on high, the counter/timer is started. If this input is on low, the counter/timer is stopped. - PA1500_SOFTWARE_GATE: Input 13 has no influence on the gate.
BYTE	b_InterruptHandling	Interrupts can be generated, when the counter has run down, or when the timer output is on high. With this parameter the user decides if interrupts are used or not. - PA1500_ENABLE: Interrupts are enabled. - PA1500_DISABLE: Interrupts are disabled.

Task:
Selects the operating mode of the second counter/timer. Enter its start value. You have to decide:

- if the counter/timer must execute the counting operation once or several times.
- if the counter/timer is used as a counter or a timer
- if an interrupt must be generated when the counter/timer has run down.
- if the external trigger is used and if the external gate is used.

Return value:
0: No error
-1: The handle-parameter of the board is wrong
-2: Wrong selection for counter/timer
(PA1500_COUNTER or PA1500_TIMER)
-3: Error with the interrupt selection
(PA1500_ENABLE or PA1500_DISABLE)
-4: User interrupt routine is not installed
-5: Cycle parameter is wrong
(PA1500_CONTINUOUS or PA1500_SINGLE)
-6: Wrong gate parameter
(PA1500_SOFTWARE_GATE or PA1500_HARDWARE_GATE)
-7: Wrong trigger parameter
(PA1500_SOFTWARE_TRIGGER or
PA1500_HARDWARE_TRIGGER)

3) i_PA1500_InitWatchdogCounter3 (...)

Syntax:
<Return value> = i_PA1500_InitWatchdogCounter3
(BYTE b_BoardHandle,
BYTE b_WatchdogOrCounterSelect,
LONG l_ReloadValue,
BYTE b_ContinuousOrSingleCycleSelect,
BYTE b_HardwareOrSoftwareGateSelect,
BYTE b_InterruptHandling)

Parameters:

BYTE	b_BoardHandle	Handle of the **PA 1500**
BYTE	b_WatchdogOrCounterSelect	Selects the mode of the third counter/watchdog - PA1500_WATCHDOG: The third counter/watchdog is used as a watchdog. - PA1500_COUNTER: The third counter/watchdog is used as a counter.
LONG	l_ReloadValue	This parameter has 2 meanings. If the counter/watchdog is used as a counter, it loads the limit value of the counter. If the counter/watchdog is used as a watchdog, it loads the watchdog time.
BYTE	b_ContinuousOrSingleCycleSelect	PA1500_CONTINUOUS: Each time the counting or timer value is set to "0", *l_ReloadValue* is loaded. - PA1500_SINGLE: If the counting or timer value is set to "0", the counter or timer is stopped.

BYTE	b_HardwareOrSoftwareGateSelect	- PA1500_HARDWARE_GATE: Input 16 is used for the gate. - PA1500_SOFTWARE_GATE: Input 16 has no influence on the gate.
BYTE	b_InterruptHandling	Interrupts can be generated, when the counter or watchdog has run down. With this parameter the user decides to use interrupts or not. - PA1500_ENABLE: Interrupts are enabled. - PA1500_DISABLE: Interrupts are disabled.

Task:
Selects the operating mode of the third counter/watchdog. Enter its limit. You have to decide:
- if the counter must execute once or several times the counting operation.
- if the counter/watchdog is used as a counter or a watchdog
- if an interrupt must be generated when the counter/watchdog has run down.
- and if the external gate is used (if it is used as a counter).

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The parameter for selecting the counter or the timer is wrong (PA1500_COUNTER or PA1500_WATCHDOG)
-3: Interrupt selection error (PA1500_ENABLE or PA1500_DISABLE)
-4: User interrupt routine is not installed
-5: Cycle parameter is wrong (PA1500_CONTINUOUS or PA1500_SINGLE)
-6: Gate parameter is wrong (PA1500_SOFTWARE_GATE or PA1500_HARDWARE_GATE)

4) i_PA1500_StartTimerCounter1(...)

Syntax:
<Return value> = i_PA1500_StartTimerCounter1
(BYTE b_BoardHandle)

Parameters:
BYTE b_BoardHandle Handle of the **PA 1500**

Task:
Starts the first counter/timer. Please initialise it previously with the function "i_PA1500_InitTimerCounter1".
If the counter is used, it is now ready for counting.
If the timer is used, it is now running.

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The counter or timer has not been initialised.
Please use function "i_PA1500_InitTimerCounter1"

5) i_PA1500_StartTimerCounter2 (...)

Syntax:
<Return value> = i_PA1500_StartTimerCounter2
(BYTE b_BoardHandle)

Parameters:
BYTE b_BoardHandle Handle of the **PA 1500**

Task:
Starts the second counter/timer, Please initialise it previously with the function "i_PA1500_InitTimerCounter2".
If the counter is used, it is now ready for counting.
If the timer is used, it is now running.

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The counter or timer has not been initialised.
Please use function "i_PA1500_InitTimerCounter2"

6) i_PA1500_StartCounter3 (...)

Syntax:
<Return value> = i_PA1500_StartCounter3 (BYTE b_BoardHandle)

Parameters:
BYTE b_BoardHandle Handle of the **PA 1500**

Task:
Starts the third counter. Please initialise it previously with the function "i_PA1500_InitWatchdogCounter3".

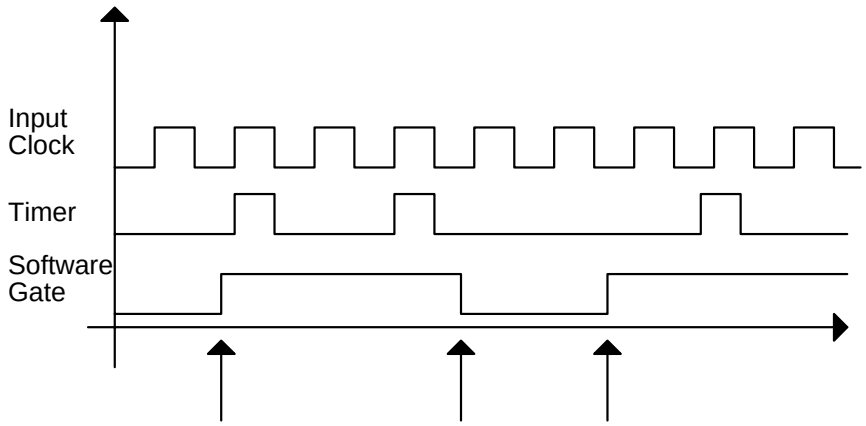
Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: Counter has not been initialised
-3: The counter/watchdog has been initialised as a watchdog.
"i_PA1500_InitWatchdogCounter3"

7) i_PA1500_StopTimerCounter1 (...)

Syntax:
<Return value> = i_PA1500_StopTimerCounter1 (BYTE b_BoardHandle)

Parameters:
BYTE b_BoardHandle Handle of the **PA 1500**

Task:
Stops the first counter/timer. The timer counting value is freezed
It has the same influence as a hardware gate.



i_PA1500_StartTimerCounter1 (..)
i_PA1500_StartTimerCounter1 (..)
i_PA1500_StopTimerCounter1 (..)

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The counter or timer has not been initialised.
Please use function "i_PA1500_InitTimerCounter1"

8) i_PA1500_StopTimerCounter2 (...)

Syntax:
<Return value> = i_PA1500_StopTimerCounter2 (BYTE b_BoardHandle)

Parameters:
BYTE b_BoardHandle Handle of the **PA 1500**

Task:
Stops the second counter/timer. The timer counting value is freezed.

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The counter or timer has not been initialised.
Please use function "i_PA1500_InitTimerCounter2"

9) i_PA1500_StopCounter3 (...)

Syntax:
<Return value> = i_PA1500_StopCounter3 (BYTE b_BoardHandle)

Parameters:
BYTE b_BoardHandle Handle of the **PA 1500**

Task:
Stops the third counter. The counting value is freezed.

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The counter has not been initialised
-3: The counter/watchdog has been initialised as a wachdog.
Please use function "i_PA1500_InitWatchdogCounter3"

10) i_PA1500_TriggerTimerCounter1 (...)

Syntax:
<Return value> = i_PA1500_TriggerTimerCounter1
(BYTE b_BoardHandle)

Parameters:
BYTE b_BoardHandle Handle of the **PA 1500**

Task:
Triggers the first counter/timer. The start value is loaded in the counter/timer.

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The counter or timer has not been initialised.
Please use function "i_PA1500_InitTimerCounter1"

11) i_PA1500_TriggerTimerCounter2 (...)

Syntax:
<Return value> = i_PA1500_TriggerTimerCounter2
(BYTE b_BoardHandle)

Parameters:
BYTE b_BoardHandle Handle of the **PA 1500**
Task:
Triggers the second counter/timer. The start value is loaded in the counter/timer.

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The counter or timer has not been initialised.
Please use function "i_PA1500_InitTimerCounter2"

12) i_PA 1500_TriggerCounter3 (...)

Syntax:
<Return value> = i_PA1500_TriggerCounter3 (BYTE b_BoardHandle)

Parameters:
BYTE b_BoardHandle Handle of the **PA 1500**

Task:
Triggers the third counter. The start value is loaded in the counter.

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The counter has not been initialised
-3: The counter/watchdog has been initialised as a watchdog.
Please use function "i_PA1500_InitWatchdogCounter3"

13) i_PA1500_ReadTimerCounter1 (...)

Syntax:
<Return value> = i_PA1500_ReadTimerCounter1
(BYTE b_BoardHandle,
PLONG_ pl_ReadValue)

Parameters:

BYTE	b_BoardHandle	Handle of the **PA 1500**
PLONG	pl_ReadValue	This parameter has 2 meanings. When the counter/timer is used as a counter, it returns the current value of the counter. When the counter/timer is used as a timer, it returns the current value of the timer.

Task:
Reads the current value of the first counter/timer if used as a counter or reads the timer status if used as a timer.
Counter: the counting value is decremented each time the input changes from low to high. This counting value can be read with this function.
Timer: the timer value is decremented each time the input clock changes from low to high. This timer value can be read with this function.

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2:The counter or timer has not been initialised.
Please use function "i_PA1500_InitTimerCounter1"

14) i_PA1500_ReadTimerCounter2 (...)

Syntax:
<Return value> = i_PA1500_ReadTimerCounter2
(BYTE b_BoardHandle,
PLONG_ pl_ReadValue)

Parameters:

BYTE	b_BoardHandle	Handle of the **PA 1500**
PLONG	pl_ReadValue	This parameter has 2 meanings. When the counter/timer is used as a counter, it returns the current value of the counter. When the counter/timer is used as a timer, it returns the current value of the timer.

Task:
Reads the current value of the second counter/timer if used as a counter or reads the timer status if used as a timer.
Counter: the counting value is decremented each time the input changes from low to high. This counting value can be read with this function.
Timer: the timer value is decremented each time the input clock changes from low to high. This timer value can be read with this function.

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The counter or timer has not been initialised.
Please use function "i_PA1500_InitTimerCounter2"

15) i_PA1500_ReadCounter3 (...)

Syntax:
<Return value> = i_PA1500_ReadCounter3
(BYTE b_BoardHandle,
PLONG pl_ReadValue)

Parameters:

BYTE	b_BoardHandle	Handle of the **PA 1500**
PLONG_	pl_ReadValue	When the counter/watchdog is used as a counter, it returns the current value of the counter

.

Task:
Reads the current value of the third counter/watchdog if used as a counter.
Counter: the counting value is decremented each time the input changes from low to high. This counting value can be read with this function.

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The counter has not been initialised.
-3: The counter/watchdog has been initialised as a watchdog.
Please use function "i_PA1500_InitWatchdogCounter3"

11 INDEX

T

U

W