DIN EN ISO 9001:2000
certified

ADDI-DATA®

Technical support:
+49 7229 1847-0

Technical description

PA 302

Analog input board

Edition: 17.04 – 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.

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 Inc.

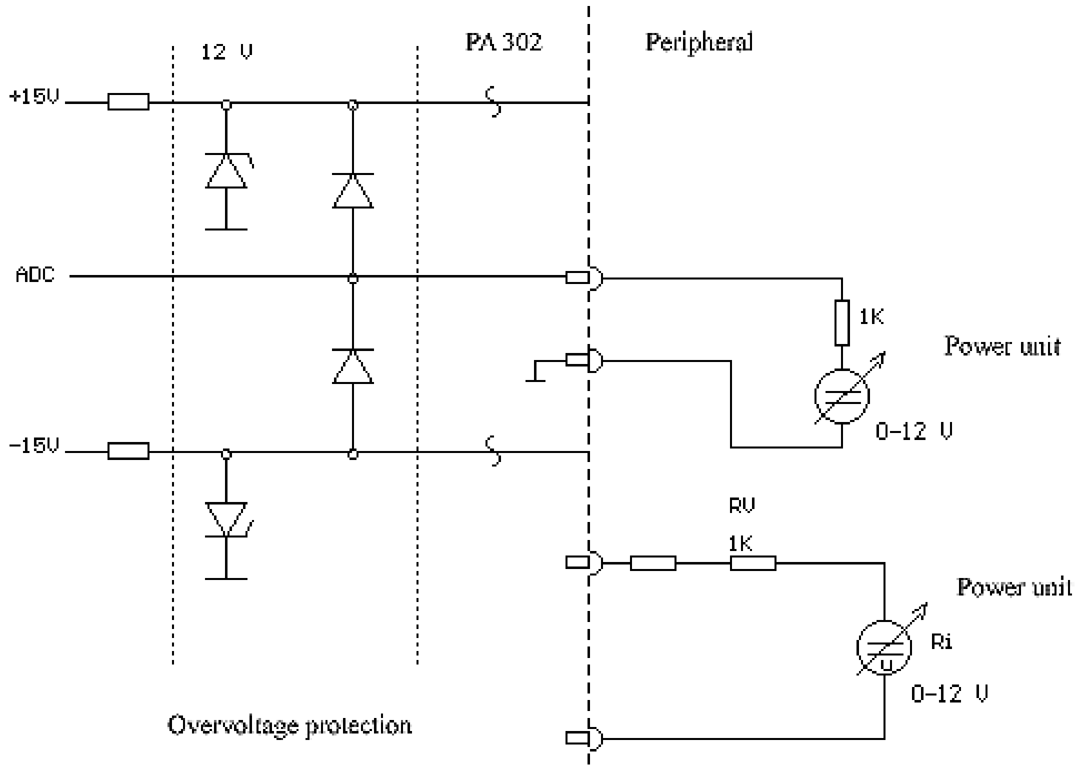
INDEX A

Figures

Tables

Caution when using a power unit!

If a power unit is connected to the analog inputs for tests, please consider the following points (see figure below):



- the inputs are provided internally with diodes protecting against over-voltage

- the resistor limiting the current at the inputs must be > = 1 kΩ.
  Series resistor (RV) or internal resistor (Ri) of the signal source (power unit).

- the voltage of the power unit must be limited to max. 12 V.

1 DEFINITION OF APPLICATION

1.1 Intended use

The **PA 302** 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 **APA 302** 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 302** 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 302** board must not be used in potentially explosive atmospheres.

The **PA 302** 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 302** 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 board.

The board has 16 (8) single-ended or 8 (4) differential input channels.
They are intended for processing analog signals.
The **PX 901** screw terminal panel allows to connect the analog signals through a shielded cable.

The use of the **PA 302** board in combination with external terminal panels is to occur in a closed switch cabinet; the installation is to be effected competently.

Check the shielding capacity of the PC and the cable prior to putting the device into operation.

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

The use of the board according to its intended purpose includes observing the advice given in this manual and 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.

Make sure that the board remains in the protective packing **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 THE BOARD

Fig. 3-1: Wrong handling

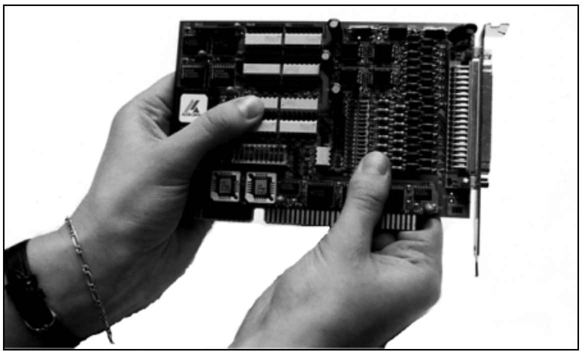
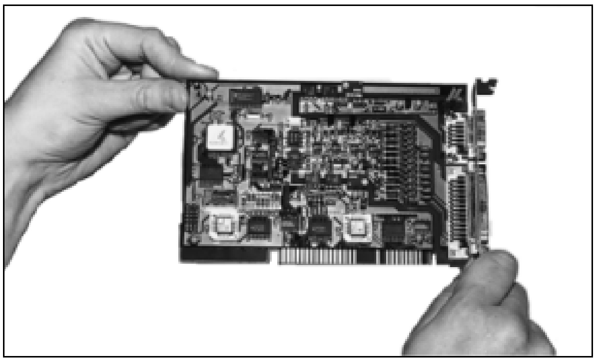


Fig. 3-2: Correct handling



4 TECHNICAL DATA

4.1 Electromagnetic compatibility (EMC)

The board **PA 302** 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.

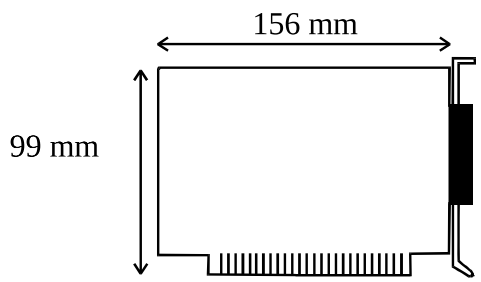
The respective EMC test report is available on request.

WARNING!
The EMC tests have been carried out in a specific appliance configuration. We guarantee these limit values **only** in this configuration[1)].

Consider the following aspects:
- your test program must be able to detect operation errors.
- your system must be set up so that you can find out what caused errors.

4.2 Physical set-up of the board

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



Weight:	150 g
Installation in:	XT / AT slot
Connection to the peripheral	through 37-pin SUB-D male connector
Standard cables:	**ST010** or **ST011**
Screw terminal panels:	**PX 901-A**, **PX 901-AG**

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

1) We transmit our appliance configuration on request.

4.3 Limit values

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
Bus speed: .... 8 MHz
Energy requirements:
Operating voltage: .... 5 V ± 5%
Current consumption: .... 600 mA typ.
Internal auxiliary voltage: .... ± 15 V

Transmission
Resolution: .... 12-bit ± 1LSB
Number of analog inputs: .... **PA302-8**: 8 single/4 diff.
PA302-16: 16 single / 8 diff
Max. throughput (single ended without INA): 125 kHz

Analog inputs
ADC input voltage ranges: .... 0 to 10 V, -5 V to +5 V, -10 V to +10 V
Max. input voltage: .... ± 20 V (S, D)
Max. input voltage for linear operation: .... ± 15 V
Multiplexer impedance, OFF: .... max 10^{11} Ω
Multiplexer impedance, ON: .... 750 Ω typ.
Total impedance: 10^{11}+ (750 x 2) + (100 x 2): approx. 10^{11}
Input leakage current: .... 0.05 nA
Output leakage currents, all channels are blocked: 0.02 nA
Output leakage current for an overvoltage of +16 V < 0.35 mA
Output leakage current for an overvoltage of -16 V < 0.65 mA

Temperature stability of ADC
System precision
Unipolar: .... ± 15 ppm/°C
Bipolar: .... ± 10 ppm/°C
Linearity drift: .... max ± 6 ppm/°C of FSR

Reference voltages
Positive output: .... 2.5 V
Positive output drift: .... ± 5 ppm/°C

Precision of ADC

0 to +10 V, ± 10 V: .... ± 0.048% of FSR
Linearity: .... ± 0.024% of FSR
Differential linearity: .... ± 0.048% of FSR
System gain error (adjustable to 0): .... ± 0.1%
System error (adjustable to 0): .... ± 0.1% von FSR

Dynamic precision of ADC

Aperture time: .... 13 ns

Delay

ADC conversion time: .... 3 µs
Basic delay (without INA): .... 3.3 µs
adjustable up to: .... max. 4.74 ms

Outputs

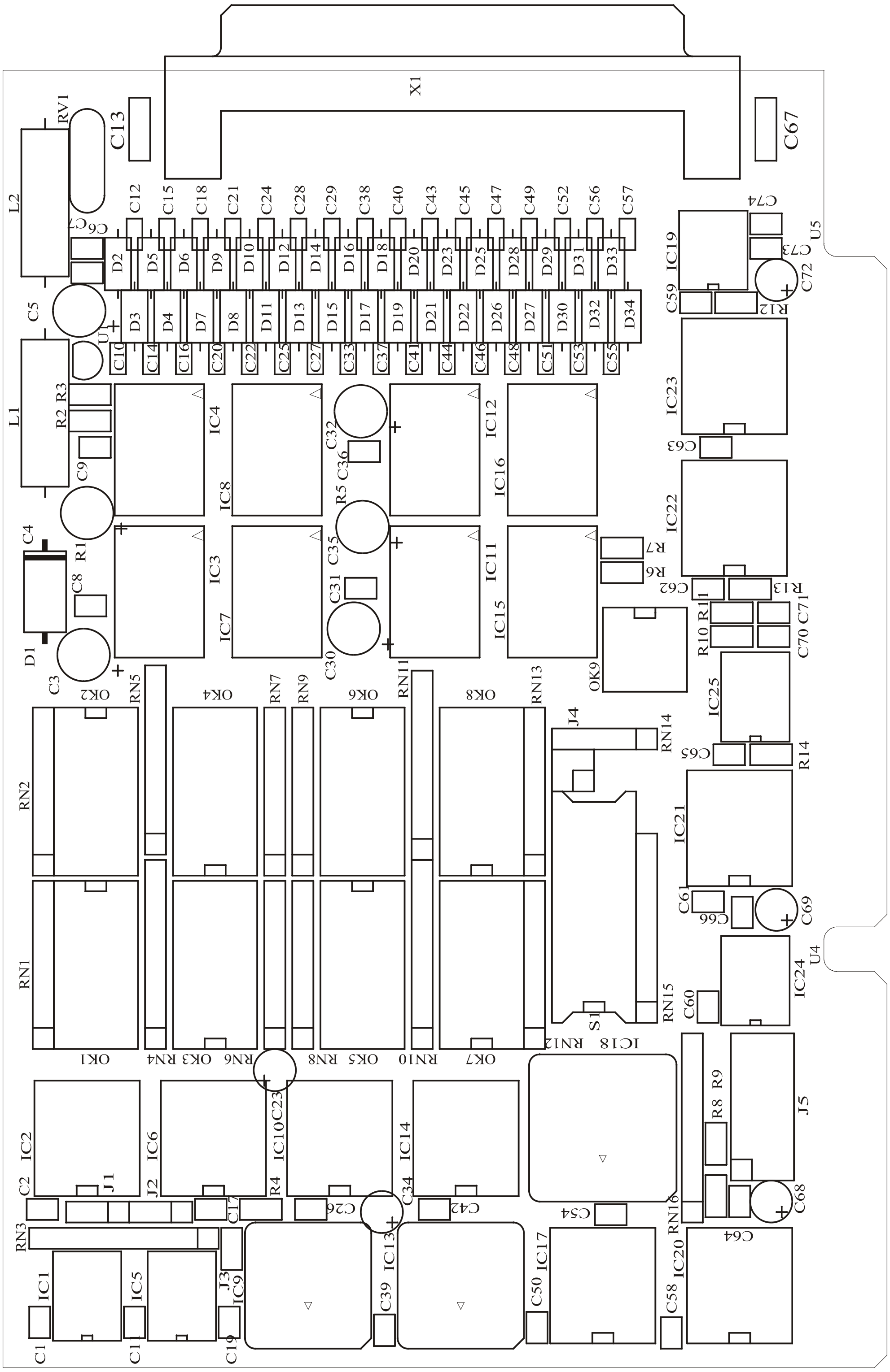
Open collector (NPN transistor): .... max 25 V/50 mA

5 SETTINGS

5.1 Settings at delivery

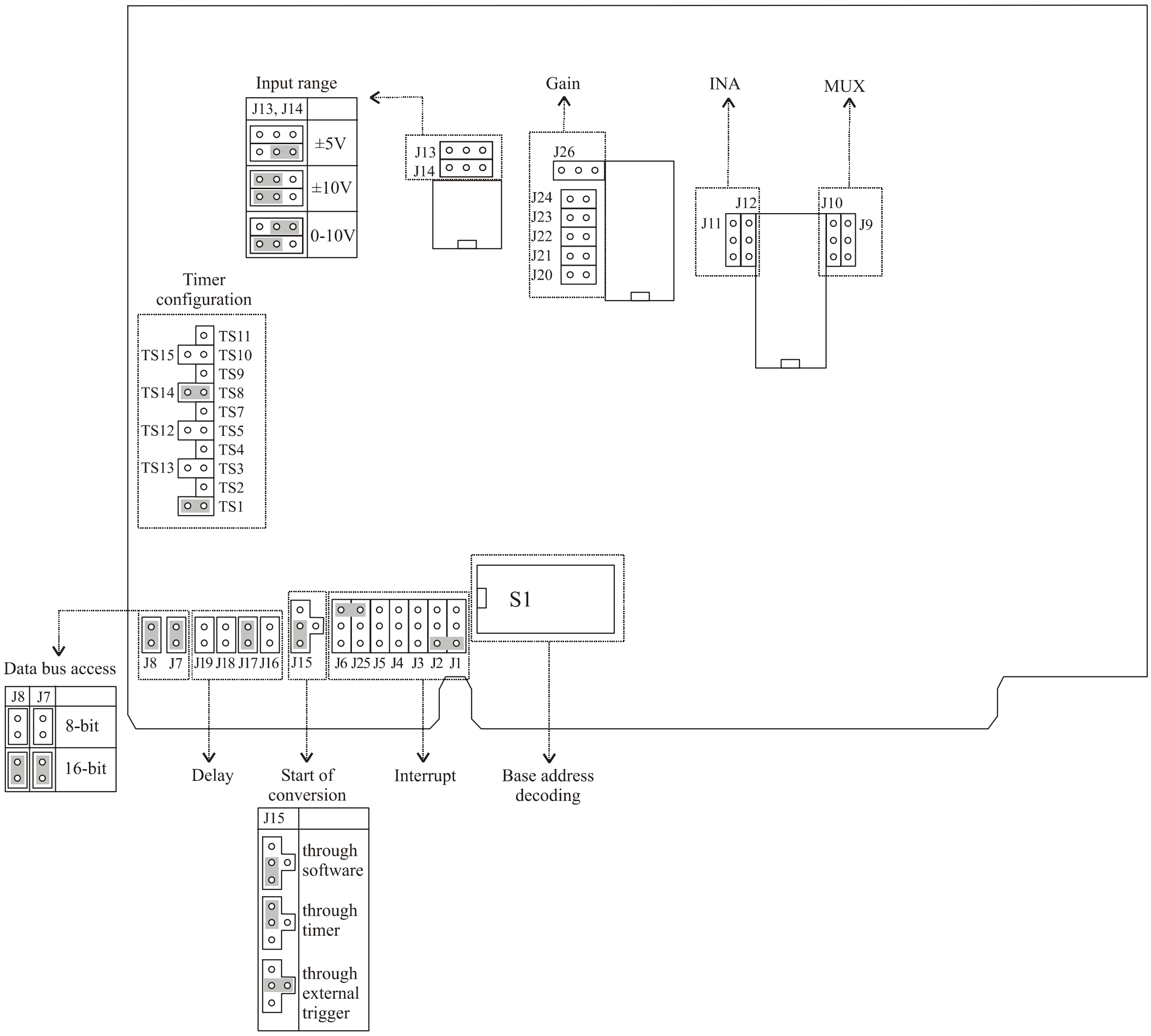
5.1.1 Component scheme

Fig. 5-1: Component scheme



5.1.2 Jumper location and settings at delivery

Fig. 5-2: Jumper location



5.1.3 Jumper settings at delivery

IMPORTANT!
The jumper settings depend on the version of the board.
The settings indicated below are common to all versions.

Data bus access	J8, J7 are set	Selection of the data bus width: 16-bit data bus access
Start of conversion		J15 Conversion started through software
Interrupt		No interrupt

5.2 I/O mapping

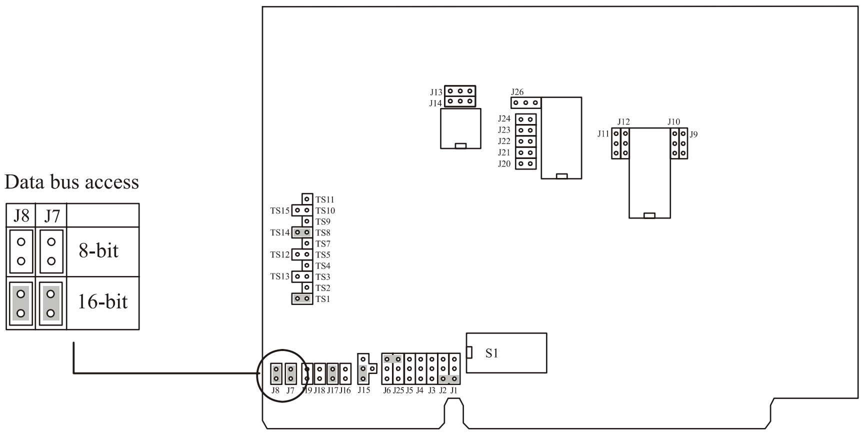
One of the outstanding characteristics of the **PA 302** is the simple use of the I/O interface. The board reacts to I/O-Read and I/O-Write commands. The decoding of the I/O address allows commanding the board within the 64KB I/O address space. The board itself occupies 8 bytes within the I/O address space.

The address is set through a block of 8 DIP switches.

For reading in analog data from the input channels, you can select either the 8-bit or the 16-bit access (only for PC/AT computers). The 16-bit access is selected when jumpers J7 and J8 are installed. The 8-bit access is selected when jumpers J7 and J8 are removed.

The 16-bit mode is selected at delivery.

Fig. 5-3: Selection of the data bus access



5.2.1 8-bit access (J7 and J8 removed)

	/IOWR								/IORD							
Address	D7...D0								D7...D0							
Base +0	X	X	X	X	M3	M2	M1	M0	B7	B6	B5	B4	B3	B2	B1	LSB
Base +1	X	X	X	X	X	X	T2	T1	EOC	UB+	UB-	X	MSB	B10	B9	B8
Base +2	Not decoded								Not decoded							
Base +3	Not decoded								Not decoded							
Base +4	Timer 0								Timer 0							
Base +5	Timer 1								Timer 1							
Base +6	Timer 2								Timer 2							
Base +7	Timer Control								Timer Status							

Base = Base address

5.2.2 16-bit access (J7 and J8 set)

	IORD															
Address	D15 ....D0															
Base+0	EOC	UB+	UB-	X	MSB	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	LSB

If the jumpers J7 and J8 are adjusted, the base address is accessed in 16-bit mode (16-bit ISA slot) and the addresses **Base +4 to +7** are accessed in 8-bit mode.

5.2.3 Selecting the analog input channels

IOWR on **Base +0** .[1]				Selected analog input			
				PA302-8		**PA302-16**.	
M3	M2	M1	M0	SE	Diff.	SE	Diff.
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	2	2	2	2
0	0	1	1	3	3	3	3
0	1	0	0	4	-	4	4
0	1	0	1	5	-	5	5
0	1	1	0	6	-	6	6
0	1	1	1	7	-	7	7
1	0	0	0	-	-	8	-
1	0	0	1	-	-	9	-
1	0	1	0	-	-	10	-
1	0	1	1	-	-	11	-
1	1	0	0	-	-	12	-
1	1	0	1	-	-	13	-
1	1	1	0	-	-	14	-
1	1	1	1	-	-	15	-

Ex.: Input 5 is selected over command
IOWR 0390H,5 (M0=1,M1=0,M2=1,M3=0)
In this example the base address is set at 0390Hex.

[1] Conversion start only when triggered by software

5.2.4 Controlling the open collector output channels

By writing Tx on **Base +1**, the open collectors are responded as follows:

T1 = "0"	Open collector 1 OFF (also after reset)
T1 = "1"	Open collector 1 ON
T2 = "0"	Open collector 2 OFF (also after reset)
T2 = "1"	Open collector 2 ON

5.2.5 Controlling the supply voltage

The internal ± 15 V supply voltage is controlled through the bits UB+ and UB-. When UB+ resp. UB- is set to "1", the supply voltage produced by the DC converter is comprised in the authorized tolerance range (12-15 V).
For analyzing conversion correctly, it is recommended to analyze also the bits UB+ and UB-.

6 INSTALLATION

IMPORTANT!
If you want to install simultaneously **several** ADDI-DATA boards, consider the following procedure.

- **Install and configure** the boards one after the other.
  You will thus avoid configuration errors.

1. Switch off the PC
2. Install the **first** board
3. Start the PC
4. Install the software (once is enough)
5. Configure the board
6. Switch off the PC
7. Install the **second** board
8. Start the PC
9. Configure the board

etc

You will find additional information to these different steps in the sections 6.1 to 6.5.

6.1 Setting the base address through DIP switches

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

Before installing the board
At delivery, the base address is set on the address 0390H.

Check, that
- the base address is free
- the address range required by the board is not already used by the PC or by boards already installed in the PC.

If the base address or the address range **are wrong**
- **Select** another base address with the block of 8 DIP switches S1**.**

Decoding the base address
The base address is decoded in steps of each time 8 I/O addresses.
It can be selected between 0 and 7FFH within the PC I/O address space.

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

Table 6-1: Decoding table

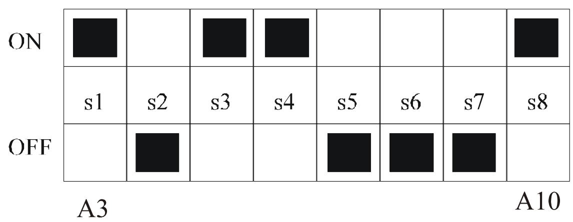
MSB … LSB

Decoded address bus	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Wished base address (hex)	0				3				9				0			
Wished base address (binary)	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0
DIP switch S1 Logic "0"= ON Logic "1" = OFF	*	*	*	*	*	s8 ON	s7 OFF	s6 OFF	s5 OFF	s4 ON	s3 ON	s2 OFF	s1 ON	X	X	X

X: decoded address range of the board
* : permanently decoded on logic "0"

Fig. 6-1: DIP switch S1

IMPORTANT!
You will find the switch **s1 on the left** of the block of DIP switches!



6.2 Inserting the board

i

IMPORTANT!
Do observe the safety instructions!

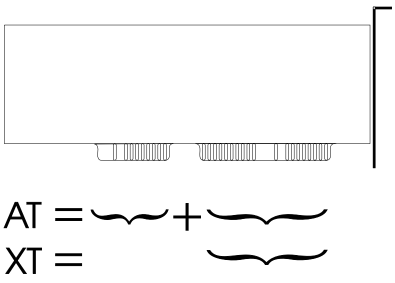
6.2.1 Opening the PC

- 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.

1. **Select a free ISA slot.**

Fig. 6-2: types of slots



The board can be inserted either in an XT or AT slot.
It can also be inserted in EISA slots.

2. **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.

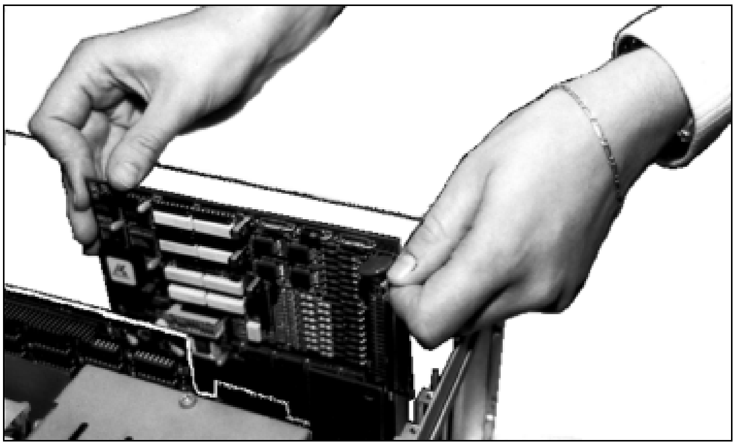
3. **Discharge yourself** from electrostatic charges.

4. **Take the board from its protective pack.**

6.2.2 Plugging the board into the slot

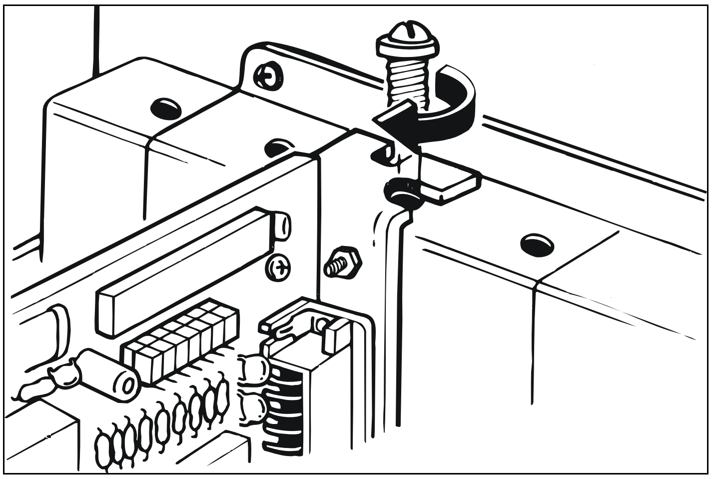
- **Discharge yourself** from electrostatic charges.
- **Insert the board** vertically into the chosen slot.

Fig. 6-4: Inserting the board



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

Fig. 6-5: Securing the board at the back cover



- **Tigthen the loose screws.**

6.2.3 Closing the PC

- Close your PC as described in the manual of the PC manufacturer.

6.3 Installing the software

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

i

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 2000/98,
- the ADDIREG registration program for Windows NT 4.0 and Windows XP/2000/98.

6.4 Board configuration with ADDIREG

The ADDIREG registration program is a 32-bit program for Windows NT 4.0/ 95. The user can register the 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.

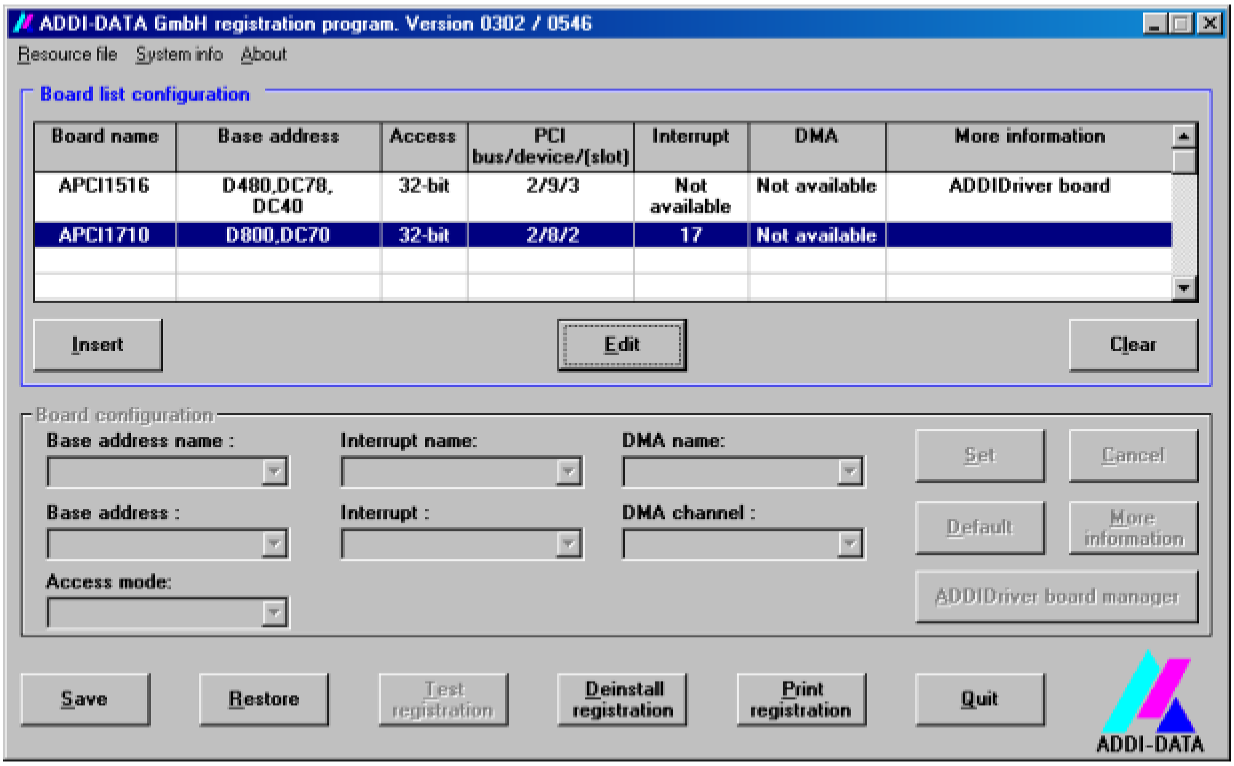
6.4.1 Program description

IMPORTANT!
Insert the ADDI-DATA boards to be registered before starting the ADDIREG program.

If the board is not inserted, the user cannot test the registration.

Once the program is called up, the following dialog box appears.

Fig. 6-6: ADDIREG registration program



The table in the middle lists the registered boards and their respective parameters.

Board name:
Names of the different registered boards (eg.: APCI-3120).
When you start the program for the first time, no board is registered in this table.

Base address:
Selected base address of the board.

i **IMPORTANT!**
The base address selected with the ADDIREG program 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.

PCI bus / slot:
Used PCI slot. If the board is no PCI board, the message "NO" is displayed.

Interrupt:
Used interrupt of the board. If the board uses no interrupt, the message "Not available" is displayed.

i **IMPORTANT!**
The interrupt selected with the ADDIREG program must correspond to the one set through DIP-switches.

ISA DMA:
Indicates the selected DMA channel or "Not available" if the board uses no DMA.

More information:
Additional information like the identifier string (eg.: PCI1500-50) or the installed COM interfaces.

Text boxes:

Under the table you will find 6 text boxes in which you can change the parameters of the board.

Base address name:
When the board operates with several base addresses (One for port 1, one for port 2, etc.) you can select which base address is to be changed.

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:
When the board must support different interrupt lines (common or single interrupts), you can select them in this box.

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

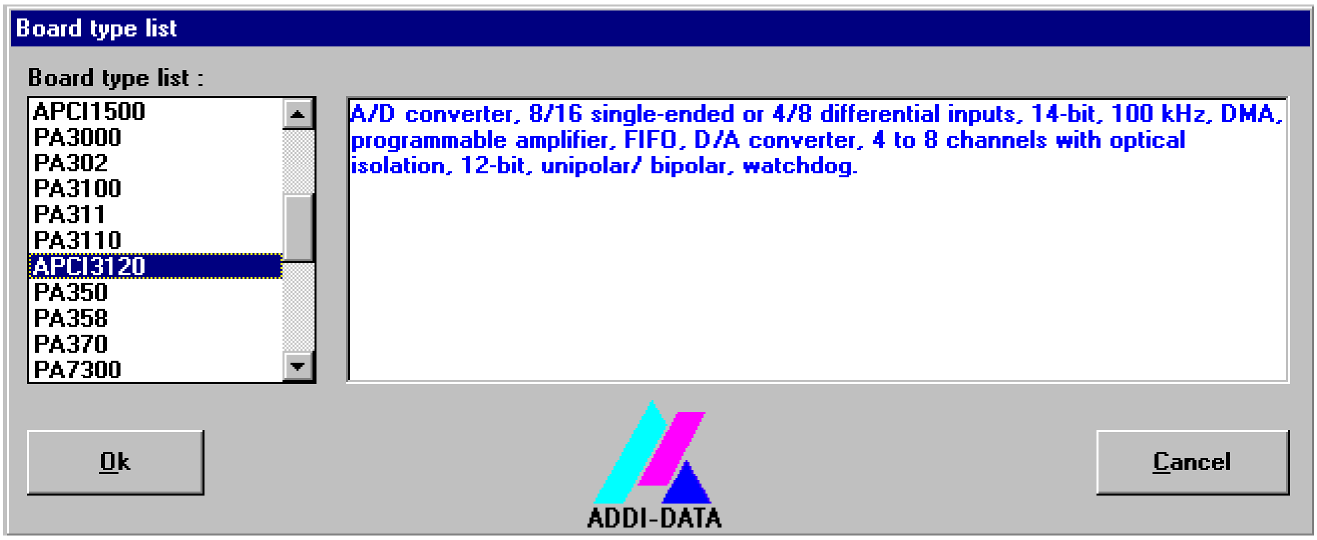
DMA name:
When the board supports 2 DMA channels, you can select which DMA channel is to be changed.

DMA channel:
Selection of the used DMA channel.

Buttons:

Edit [1]:
Selection of the highlighted board with the different parameters set in the text boxes. Click on "Edit" to activate the data or click twice on the selected board.
Insert:
When you want to insert a new board, click on "Insert". The following dialog window appears:

Fig. 6-7: Configuring 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 informations 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 parametered 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:
You can change the board specific parameters like the identfier string, the COM number, the operating mode of a communication board, etc...

---

1 "x": Keyboard shortcuts; e.g. "Alt + e" for Edit

If your board does not support these information, you cannot activate this button.

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.

6.4.2 Registering a new board

i

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. The figure 6-7 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.

6.4.3 Changing the registration of a board

i

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.

6.5 Software downloads from the Internet

You can download the latest version of the device driver for the PA302 board**.**

http://www.addi-data.com

i

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.

If you have any questions, do not hesitate to send us an e-mail:

info@addi-data.de or
hotline@addi-data.com

7 FUNCTIONS

7.1 Introduction

The **PA 302** board is a high quality analog data acquisition board.
The 12-bit accuracy of the board is guaranteed through:

- the A/D converter ADS 7800 of BURR BROWN,
- the isolation of analog and digital signals through earthing and filtering of the operating voltage.

Protection circuitry, optional input filters and current-voltage converters allow to use the **PA 302** in noisy environments.
For the acquisition of very small input signals, the **PA 302** is equipped with INA[1], which gain can be set through resistor or jumper.

The board needs no software initialisation. It is ready to operate immediately after applying the operating voltage and after power ON reset.

In addition to analog data acquisition, the board offers a timer module 82C54 and 2 open collector outputs (T1 and T2). The timer module is composed of 3 programmable 16-bit counters. When connected internally they begin A/D conversion cycles automatically; when connected externally they are linked to other applications.

The **PA 302-16** board accepts up to 16 analog signals through the 37-pin front connector. The number of possible analog input channels depends on whether the **PA 302** is operated in single ended or differential mode:

PA 302-16	Single ended:	16 analog input channels
	Differential:	8 analog input channels
PA302-8	Single ended:	8 analog input channels
	Differential:	4 analog input channels

The board is connected as follows:
- directly with cable to the analog signal transmitters
- with our standard cable **ST010** or **ST011** to the screw terminal panel **PX 901-A** or **PX901-AG**

7.2 Features

- Complete analog data acquisition board
  - 16 single-ended or 8 differential inputs (**PA 302-16**)
  - 8 single-ended or 4 differential inputs (**PA 302-8**)
  - 12-bit analog/digital conversion
  - Conversion time: 3µs / channel
  - Each channel can be selected through software
  - Gain adjustable through resistor or jumper.

[1] INA = Instrumentation amplifier

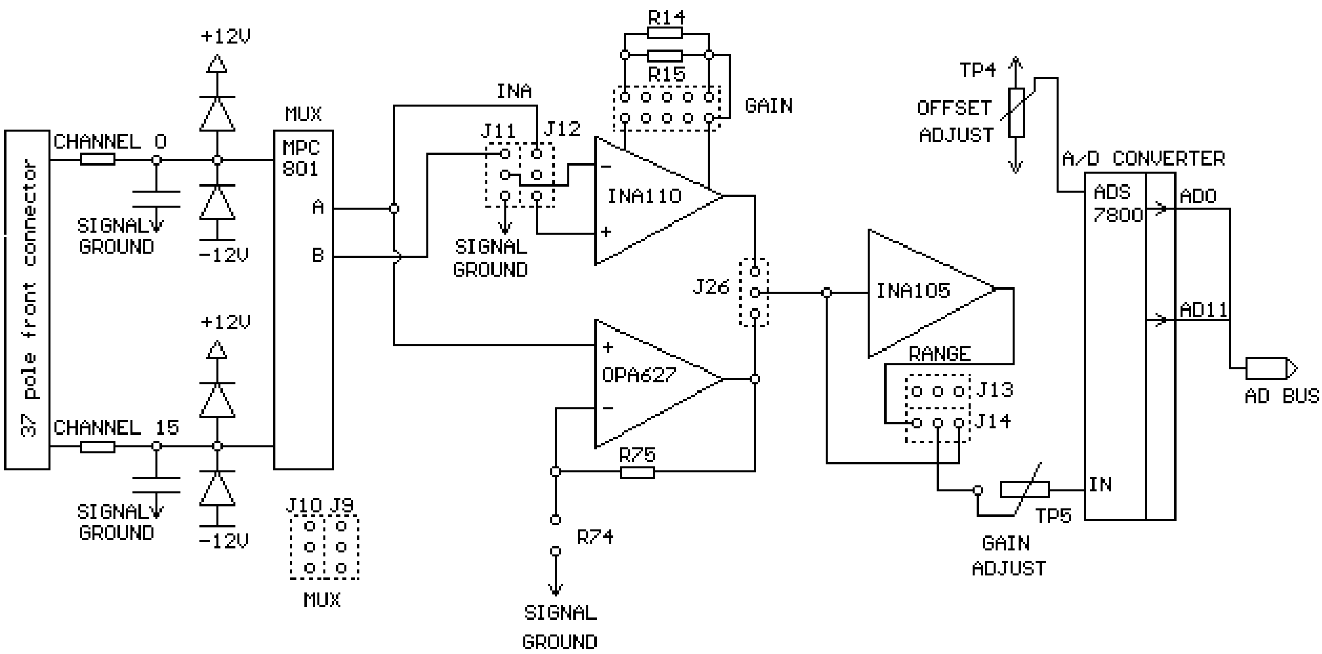
Standard gain factors selectable through jumper: 10, 100, 200, 500 and 800.
Other factors can be selected through resistor.
- Input protection circuitry (protection up to ± 20 V)
  input filter (fc = 150 kHz)
- Input ranges (0-10 V, ± 5 V, ± 10 V) selectable through jumper
- End of conversion (EOC) can be asked by software
- Possibility of interrupt at end of conversion (EOC)
- Two software-programmable open collector outputs, for example for triggering alarm transmitting devices in case limit values are exceeded or for the connection of lamps, relays, etc..
- Three 16-bit timers (82C54) freely programmable
- If the **PA 302** board is installed in an 16-bit ISA slot, you can use a data bus access of resp. 8-bit or 16-bit

Options:
SF: Filter for 16 single ended inputs
DF: Precision filter for 8 differential inputs
SC: 0-20 mA or 4-20 mA current inputs for 16 single ended inputs
DC: 0-20 mA or 4-20 mA current inputs for 8 differential inputs

Remark: with a current range of 4-20 mA the precision may be altered.

7.3 Block diagram

Fig. 7-1: Block diagram



7.4 Selection of the input range

The analog input channels can be set on three ranges

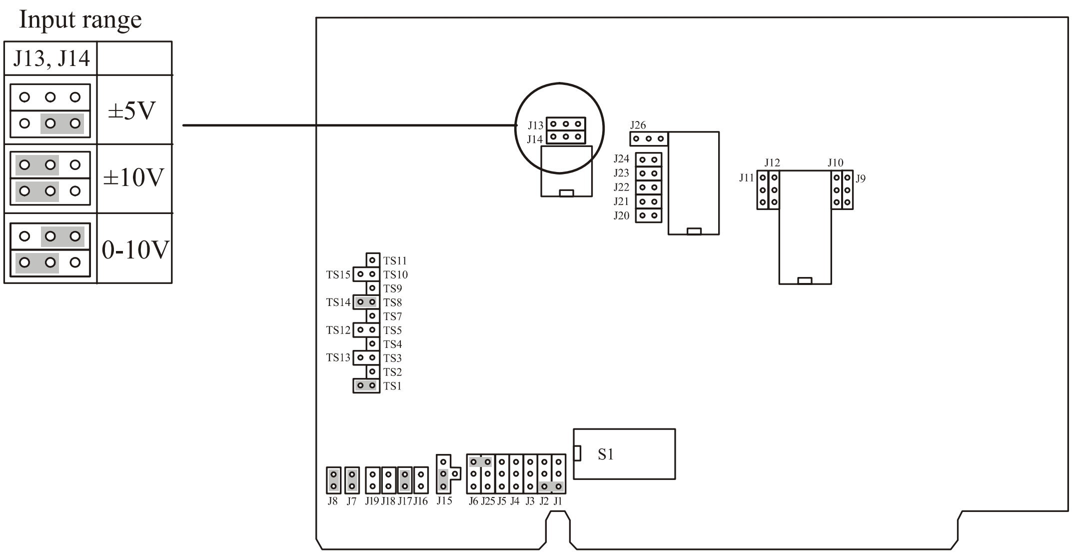
• 0 to +10 V (unipolar)
• - 5 to +5 V (bipolar)
• -10 to +10 V (bipolar)

The board supplies all the necessary precision reference voltages.

The **PA 302** is built in order to work with a supply voltage of only +5 V. The auxiliary voltages +15 V and -15 V are supplied through the DC/DC converter for processing analog signals. A precision DC/DC converter is therefore used with an integrated Pi-filter. The output voltages are additionally LC-filtered as well as the +5 V voltage supplying the analog components.

If you wish to use the internal +5 V through pin 28 of the front connector (wired to digital ground), first install on the board on position F1 a microfuse of < 500 mA. This installation is at your own risk. You will find the position of F1 in the component scheme.

Fig. 7-2: Selection of the input range



The adjustments shown above are for gain = 1.

If the board is used with the INA amplifier, the input ranges will be different depending on the selected gain factor.

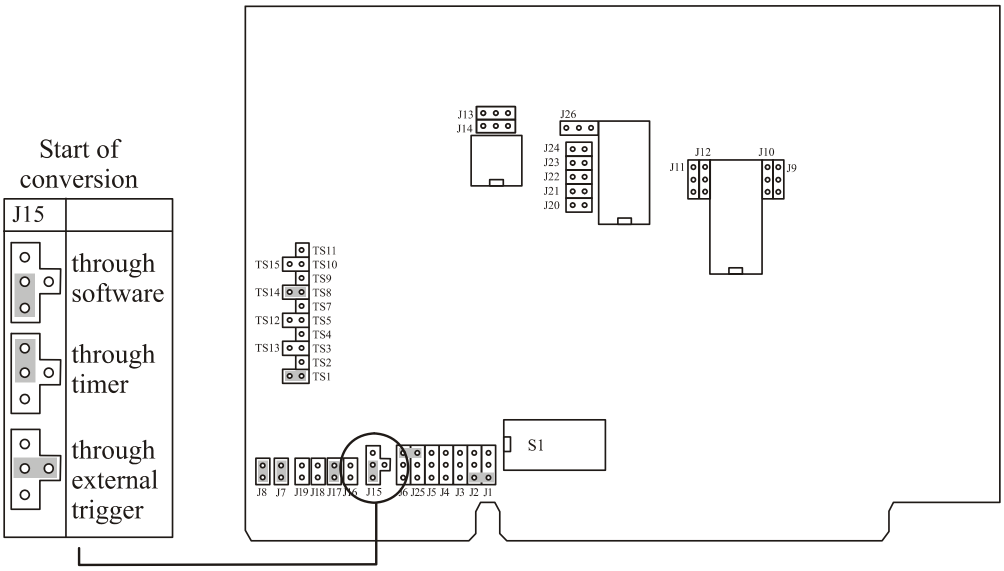
7.5 Start of conversion

7.5.1 Mode selection for starting the conversion

After the input to convert has been selected in **Base +0**,
conversions can be triggered through software, timer or external trigger.

The type of conversion start is chosen over jumper field J15.
Setting at delivery: conversion triggered through software

Fig. 7-3: Start of conversion - Selection through J15



7.5.2 Conversion triggered through software

The input channel is selected by writing data on **Base +0**.
The conversion starts.

7.5.3 Conversion triggered through timer

The timer component 82C54 can be configured to trigger the conversion.

The three timers are freely programmable and can be cabled externally or internally on the board. It is possible to generate with the timers a cyclic time interrupt, or to start a conversion over timer.

The input to be converted must be first selected in **Base +0**.

Fig. 7-4: Conversion triggered through timer - Jumper settings

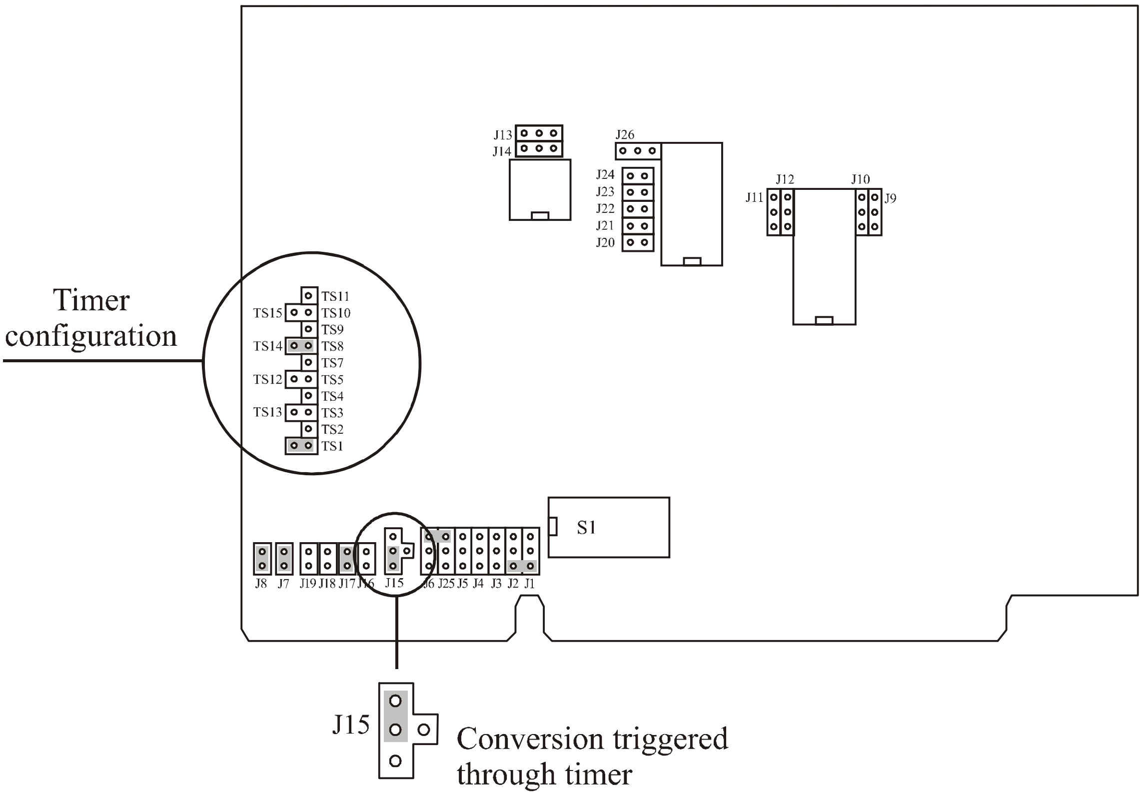
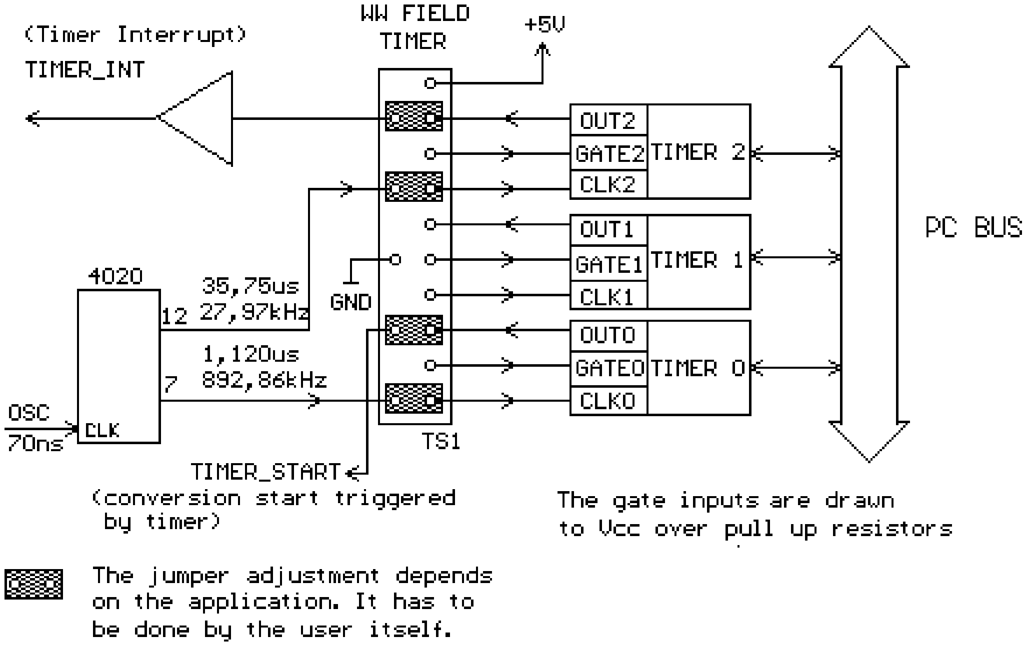


Fig. 7-5: Timer configuration - Jumper settings



Example of a conversion started through timer

```
PROGRAM PA302_Timer_start;
uses dos,crt;
CONST
(************Addresses of the different boards**********************)
     pa302         = $390;  (* Base address of PA302 *)
     IRQ_CTRL1     = $20;
  (* Base address of the MasterInterrupt Controller   *)
(******************************** Control bytes *****************)
     EOI            = $20; (* End of Interrupt for the IRQ Controller*)
     IRQ3_ENA       = $F7;                    (* Enable mask for IRQ3*)
     IRQ3_VECT_NR      = $0B;   (* Vector number associated with IRQ3*)
(****************************** Type definition ********************)
TYPE
     results = array[0..15] of word;
(******************************** Global variables *****************)
VAR
      oldvector    : pointer;              (* Old value of IRQ vector *)
      oldstatus    : byte;* Old status Of IRQ Controller Enable Mask  *)
      EOC              : boolean;
      mux,i            : byte;
      value            : results;        (* Array of converted values *)
(**************************** Function prototypes ******************)
Procedure  Disable;
BEGIN
inline ( $FA );                                               (* CLI *)
END;
(*****************************************************************)
Procedure  Enable;
BEGIN
inline ( $FB );                                                (* STI *)
END;
(*****************************************************************)
Procedure New_Irq3_Handler( flags,cs,ip,ax,bx,cx,dx,si,di,ds,es,bp:word);
interrupt;
BEGIN
disable;                                       (* Disable interrupts *)
value[ mux ]:= portw[ PA302 ] and $0FFF;  (* Read the converted value *)
If ( mux = 15 ) then  Mux := 0 (* If channel > 15 then convert channel0*)
               else  Inc (Mux);      (* Otherwise select next channel *)
port[ PA302 ]:= Mux;
 (* Change channel on the multiplexer *)
port[ irq_ctrl1 ]:= EOI;                                      (* EOI *)
enable;                                         (* Enable interrupts *)
END;
(*****************************************************************)
Procedure Install_New_Irq_3;
BEGIN
Disable;
getintvec (irq3_vect_nr , oldvector);                (* Save old IRQ vector *)
setintvec ( irq3_vect_nr , @new_irq3_handler);      (* Set the new IRQ vector *)
oldstatus := port[ irq_ctrl1 + 1 ];
port[ irq_ctrl1 + 1 ] := port[ irq_ctrl1 + 1 ] and irq3_ena;
                                                   (* Enable IRQ channel 3 *)
port[ irq_ctrl1 ] := EOI;                                        (* EOI *)
Enable;
END;
```

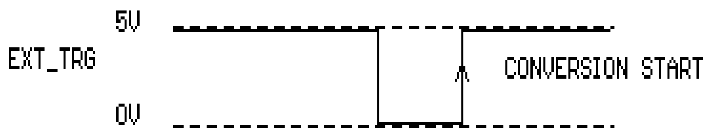
```
(**********************************************************************)
Procedure  Desinstall_Irq_3;
BEGIN
Disable;
setintvec( irq3_vect_nr , oldvector );      (* Set the old IRQ vector *)
port[ irq_ctrl1 + 1 ] := oldstatus;       (* Set the old Enable Mask *)
Enable;
END;
(**********************************************************************)
Procedure Timer_Convert_Start ( divisor : word );
CONST timer_mode = $34;                          (* TIMER 0 in mode 2 *)
VAR   lowbyte , highbyte : byte;
Begin
 lowbyte  := divisor Mod 256;
 highbyte := divisor Div 256;
 port[ pa302 + 7 ] := Timer_Mode;             (* Timer 0 control byte *)
 delay(1);
 port[ pa302 + 4 ] := Lowbyte;      (* Timer 0 initial value low byte *)
 delay(1);
 port[ pa302 + 4 ] := Highbyte;    (* Timer 0 initial value high byte *)
End;

(**********************************************************************)
(*                              MAIN                                  *)
(**********************************************************************)

BEGIN
  clrscr;
  switchboard_on;
  Install_New_Irq_3;                             (* Install the new IRQ *)
  mux := 0;                 (* Initialise to convert first channel 0 *)
  Timer_Convert_Start ( 900 );         (* Timer conversion every 1 ms *)
  repeat;
  for I := 0 to 15 do               (* Print the values on the screen *)
  begin
    gotoxy(5,10+ I ); write(' Channel [ ',I,'] = ',value [ I ] );
  end;
  delay(100);
  clrscr;
  until keypressed;

  Desinstall_Irq_3;
  switchboard_off;
END.
```

7.5.4 Conversion triggered externally

The conversion start can be triggered through an external TTL signal (pin 9 EXT_TRG). When this mode is selected, the conversion of the input selected previously is started with the leading edge of this signal.

The input to be converted must be selected previously in **Base +0**.

Fig. 7-6: Conversion triggered externally



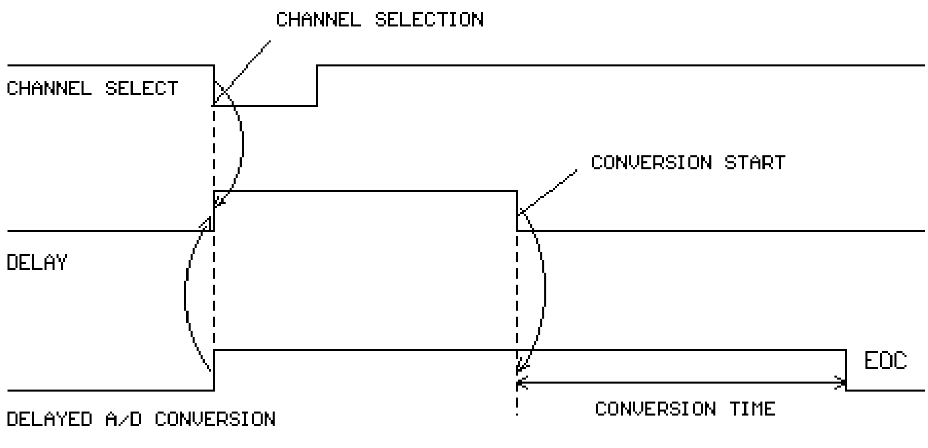
This input is protected against voltage reversal and overvoltage (± 20 V).

7.5.5 Delay

This function can only be used if the conversion is triggered through software. A delay can be defined for the A/D conversion. The delay time is started by the selection of the input address.

The delay time depends on three factors:
- gain of INA110 (if used)
- internal resistance of the source (Rq)
- limit frequency of the input filters (options SF and DF)

Fig. 7-7: Delayed A/D conversion



Influence of the internal resistance of the signal source

The delay time allows the internal capacities to recharge to the new voltage value.

The conversion time is 3 µs in the standard version.
This time relates to a signal source with an internal resistance smaller than 200 R.

With the falling edge of delay, the analog value is held in the "Sample and Hold" amplifier and conversion is begun.

In this case it is guaranteed that the multiplexer capacities are recharged quickly enough to the new voltage value. If the signal sources used are of larger internal resistance (>200R), the internal capacities will need more time for recharging to the new voltage value. The delay time must be prolonged for this reason.

The delay time is selected through the jumper field DELAY.

Fig. 7-8: Jumper field DELAY

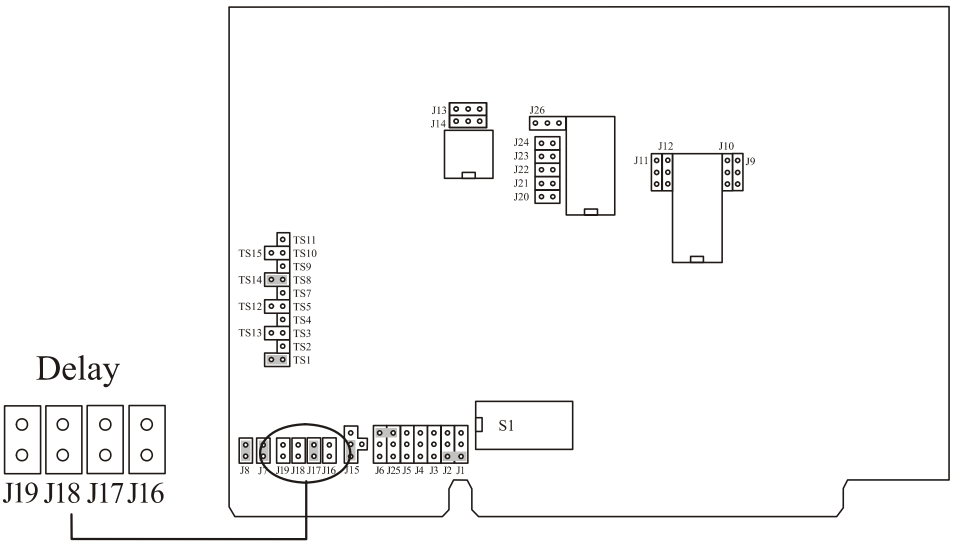


Table 7-1: Delay time selected through jumpers

J16	C = 3.3 nF	Delay approx. 3.3 µs	Rq < 200 R
J17	C = 10 nF	Delay approx. 10 µs	Rq < 1 K
J18	C = 33 nF	Delay approx. 33 µs	Rq < 10 K
J19	C = 4.7 uF	Delay approx. 4.7 ms	Rq < 1 M

Rq = internal resistance of signal source

Several jumpers can be adjusted simultaneously. In this case the times are added up.

Example

The jumpers J16 and J17 are set.
The delay time is 13.3 µs.
The conversion time is 3µs.
The total time is 16.3 µs and the throughput is 61 kHz.

Influence of the gain defined for INA110 on the delay time

When a signal is amplified by the INA, you must take into account the delay time which corresponds to the settling time (t_R) of the amplifier (see the following table).

The settling time of INA110 at 0.01% and with a voltage jump of 20 V is:
(Source: data sheet BURR BROWN)

Table 7-2: Settling time of the INA110 component

Gain = 1	t_R = 5 µs
Gain = 10	tR = 3 µs
Gain = 100	tR = 4 µs
Gain = 200	tR = 7 µs
Gain = 500	tR = 16 µs

7.6 Reading the converted data

For converting one single analog value, two resp. three program operations have to be executed, depending on the 8-bit or 16-bit bus access.
The EOC bit indicates the end of the conversion.

7.6.1 8-bit data access

1) The input number is written on **Base +0**.
   The defined delay which starts the A/D conversion is automatically initiated.
2) The End of conversion is established by analyzing the EOC bit (bit 7).
   The 4 highest bits of conversion are read on **Base +1**.
   Bit0-bit 3 corresponds to the conversion bits B8 - MSB.
3) The 8 lowest bits of conversion are read on **Base +0**. The bits LSB - B7 resp. B8-MSB of the converted analog value are available.

The 8-bit data access is selected with the jumpers J7 and J8 not being set.

7.6.2 16-bit data access

1) The input number is written on **Base +0**.
   The defined delay which starts the A/D conversion is automatically initiated.
2) The access occurs through a 16-bit read command on **Base +0**.
   The high byte is available on **Base +1** and the low byte on **Base +0**.
   D15 corresponds to EOC and D0-D11 to data bits LSB-MSB.

The 16-bit data access is selected with the jumpers J7 and J8 being set.

7.6.3 Analyzing the EOC bit

The EOC bit (end of conversion) indicates whether the conversion is completed or not.

EOC = "0" Conversion is completed
EOC = "1" Conversion is running

The EOC bit is set to "1" with a new conversion start.

7.6.4 Program example in C

```
main ()

  {

  int value,EOC bit;

  outportb (0x390,0);             /*   Conversion start input 0       */

  do

  {

  value = inport (0x390);         /*   Read value with 16-bit access  */

  EOC = value & 0x8000;           /*   Mask EOC                       */

  }

  while (EOC bit != 0);           /*   If conversion not completed    */

  value = inport (0x390) & 0x0FFF;  /*   12-bit value                 */

  printf ("value =", value);      /*   Screen printing                */
```

7.7 Operating modes

The board offers the following operating modes:
- Single ended: without INA
- Single ended: with INA
- Differential: with INA

These modes are selected through jumpers.

Fig. 7-9: Components and jumpers involved

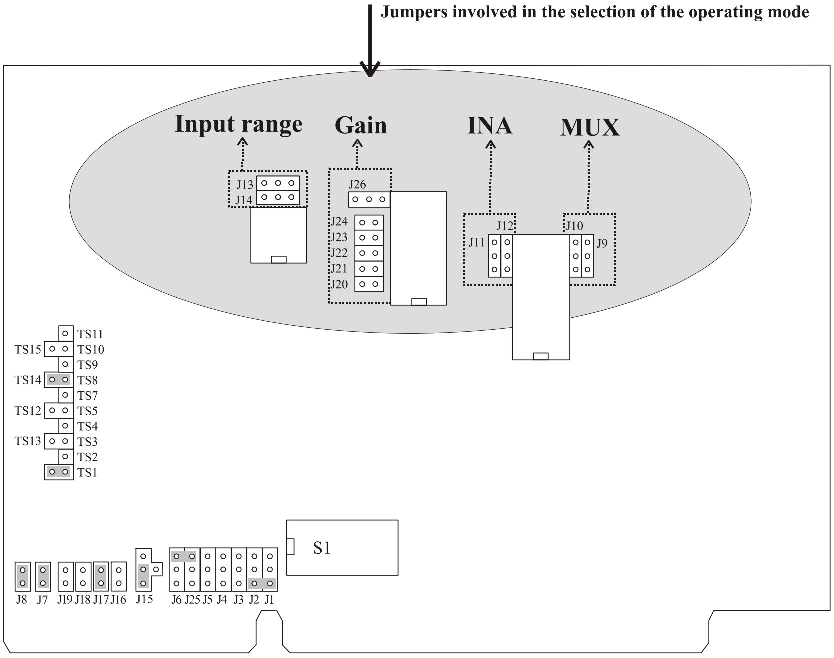
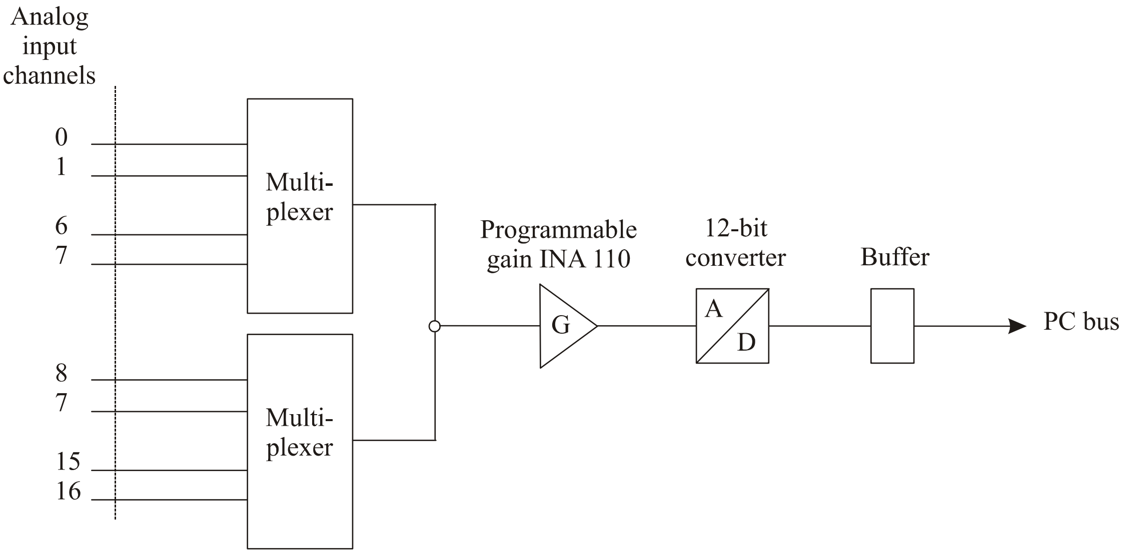
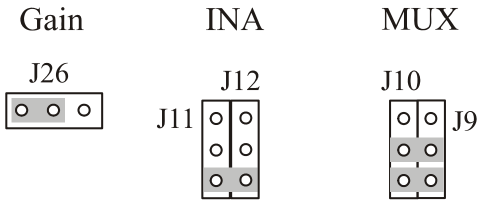


Fig. 7-10: Analog conversion in single ended MODE



7.7.1 Single ended without INA

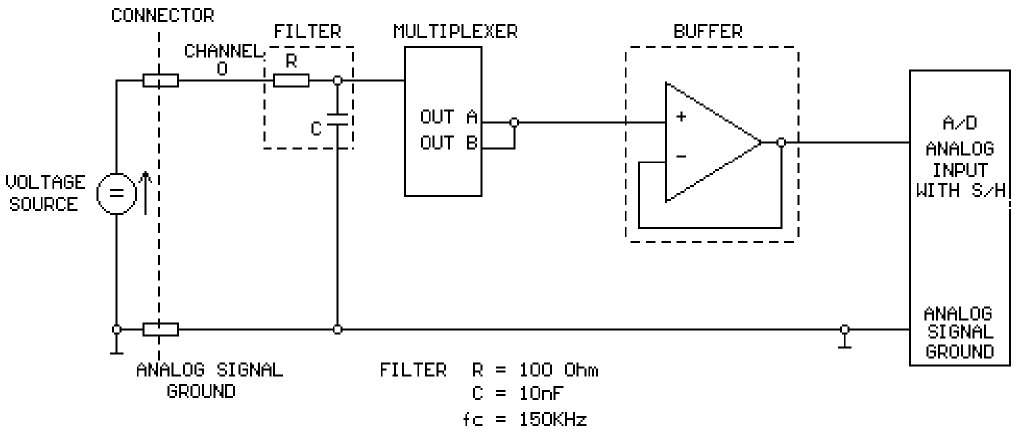
Fig. 7-11: Single ended without INA - Jumper settings



Signal and ground connection in single ended mode without INA

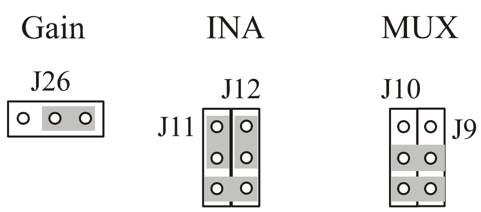
If the **PA 302** board is operated in single ended mode, the return line of all input signals is common and is connected to the analog ground of the A/D converter. The multiplexer outputs are connected with the positive input of the "Sample and Hold" amplifier (see configuration on fig.4g).
The signal ground of the A/D converter is connected with the analog ground pins of the front connector.

Fig. 7-12: Single ended mode without INA - Signal and ground connection



7.7.2 Single ended mode with INA

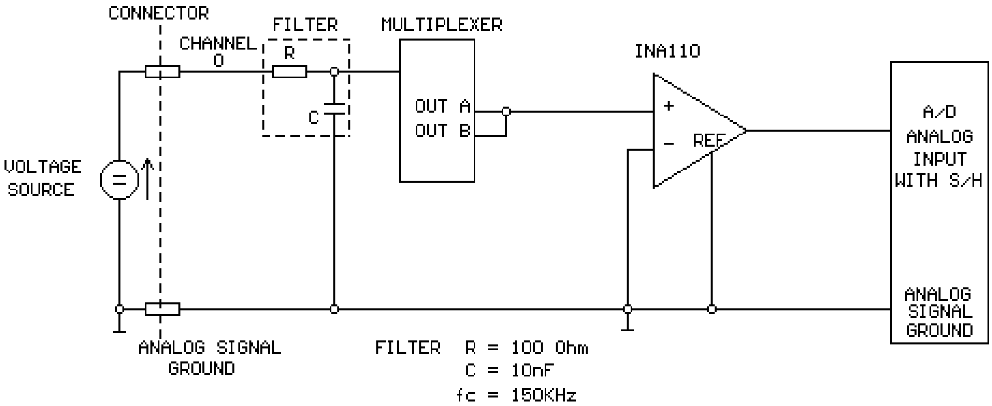
Fig. 7-13: Single ended mode with INA - Jumper settings



Signal and ground connection in single ended mode with INA

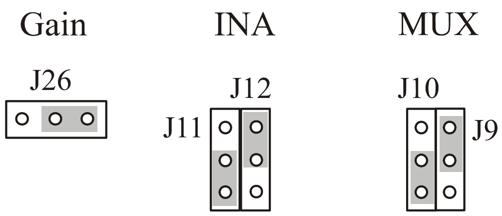
If the board is used with the adjustable amplifier (INA), the multiplexer outputs are linked with the positive input of INA. The negative input of INA is connected to the analog ground.
The output of INA is connected to the input of the "Sample and Hold" amplifier.

Fig. 7-14: Single ended mode with INA - Signal and ground connection



7.7.3 Differential mode with INA

Fig. 7-15: Differential mode with INA - Jumper settings



The selectable gain factors are described in chapter „Gain“.

Signal and ground connection in differential mode

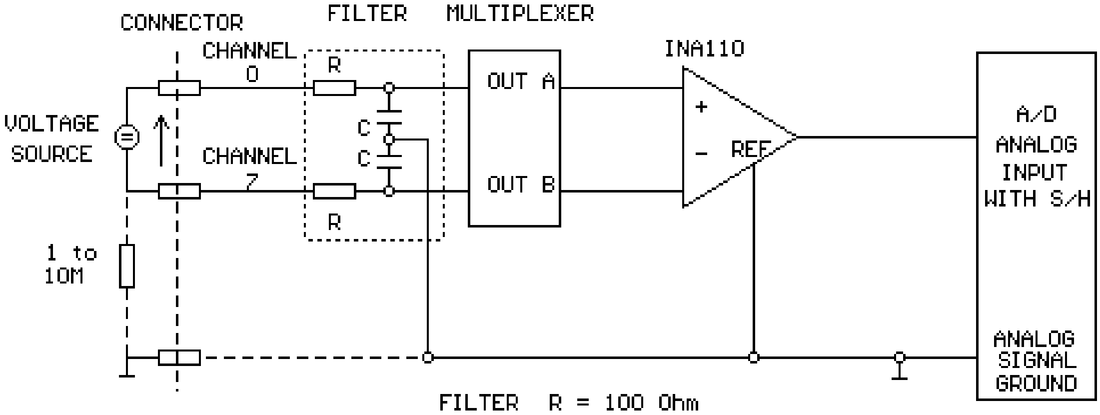
In differential mode, the board is generally operated with INA.

The output of the multiplexer - with channels 0 to 7 - is connected to the positive input of the INA. The output of the multiplexer - with channels 8 to 15 - is connected to the negative input of the INA. The output of the INA is connected to the input of the "Sample and Hold" amplifier.

The analog input signal has no ground connection in differential mode.

If the voltage is located outside the common-mode range of INA, it may occur that the differential voltage can not be constituted at the INA input (+/-15 V voltage supply). In this case a virtual ground has to be added with a 1M to 10M resistor.

Fig. 7-16:Differential mode with INA



7.8 Inputs with 4-20 mA or 0-20 mA current loop (option)

With the option C (current), the PA 302 board is equipped with current inputs.

In single ended mode, all 16 (8) channels are equipped with a current-voltage converter (option SC).
Each resistor is installed between the analog input pin of the front connector and the analog ground.

In differential mode, all 8 (4) channels are equipped with a current-voltage converter (option DC). A high-precision measuring resistor of 250R 1/4W is used for converting current into voltage.
The resistor is adjusted parallel to the input signal.

A current input signal of 0(4)-20 mA produces a voltage drop of 0 (1) to 5 V at the measuring resistor.
With current inputs should be set at the range of 0-10 V (gain = 2).

The gain is adjusted through resistors on the positions R14 and R15.You can install the option C yourself by inserting resistors of a value of 249R 0,1% RM 10 on the positions mentioned below.
In this case, the installation of the resistors occur at your own risk.
For the location of the resistors, see the component scheme.

Remark: with a current range of 4-20 mA the precision is altered.

7.8.1 Resistors for current inputs in single ended mode (option SC)

The following resistors are to be installed:
R29, R26, R35, R31, R40, R36, R45, R41, R50, R46, R55, R51, R60, R56, R65, R61

7.8.2 Resistors for current inputs in differential mode (option DC)

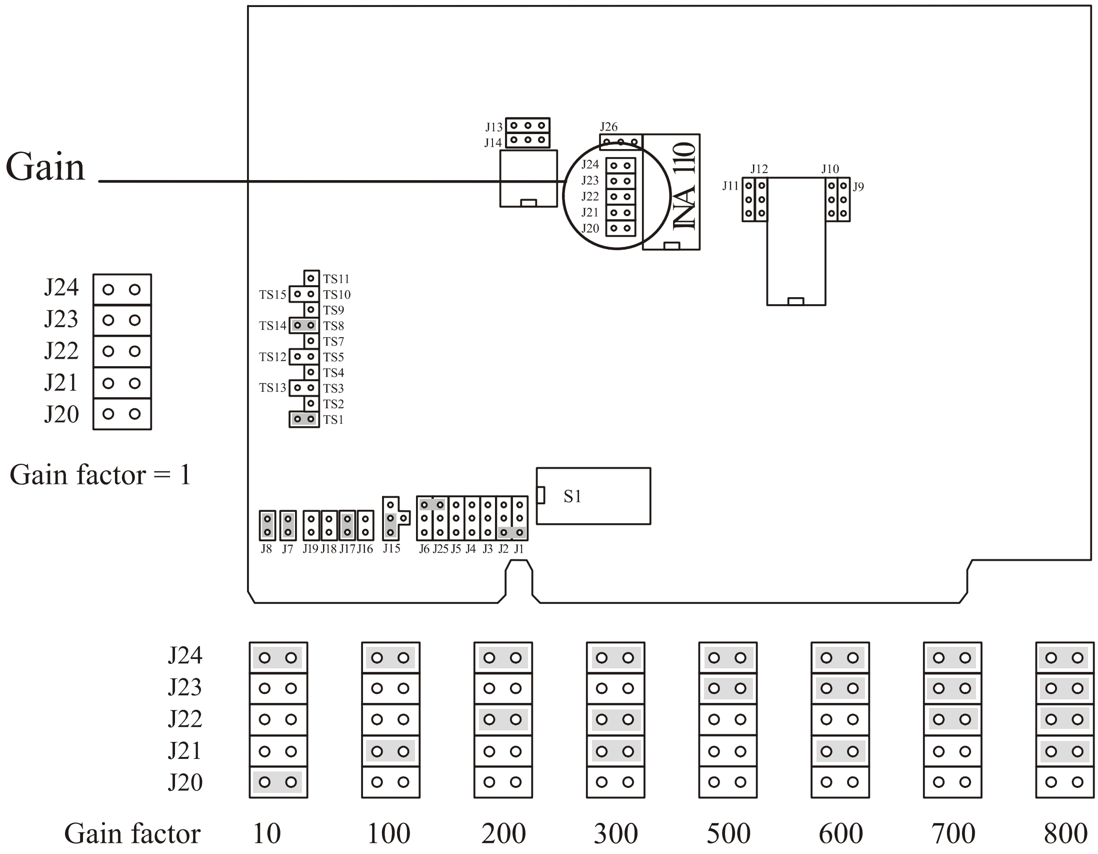
The following resistors are to be installed:
R28, R33, R38, R43, R48, R53, R58, R63

7.9 Gain

The instrumentation amplifier INA110 allows to change the gain factor.
The gain is adjusted through the jumpers J20, J21, J22, J23, J24.

Configuration at delivery : gain 1 (no jumpers set)

Fig. 7-17: Gain adjustment - Jumper setting



Intermediate values are obtained by wiring resistor Rg which is calculated as follows:

$$Rg = \frac{40k}{G - 1} - 50\ ohm$$

G = wished gain factor
Jumpers J23, J22 and J21 are adjusted.

Please use **only** metal film resistors.

You may possibly not find Rg among the standard resistors. Rg is obtained by combining two standard resistors. They are installed on positions R14 and R15 (see component scheme). In the layout, these positions are connected in parallel. Rg is calculated as follows:

$$Rg = \frac{R14 * R15}{R14 + R15}$$

When using the INA amplifier be sure that the time for triggering the start of A/D conversion is adapted to the gain (see chapter „Delay“).

Table 7-3: Input range according to the selected gain factor

A/D conversion range			
Gain	**Input 0 to 10 V**	**Input -10 V to +10 V**	**Input -5 V to +5 V**
1	0-10 V	-10 to +10 V	-5 to +5 V
10	0-1 V	-1 to +1 V	-0,5 to +0,5 V
20	0-500 mV	-500 to +500 mV	-250 to +250 mV
50	0-200 mV	-200 to +200 mV	-100 to +100 mV
100	0-100 mV	-100 to +100 mV	-50 to +50 mV
200	0-50 mV	-50 to +50 mV	-25 to +25 mV
500	0-20 mV	-20 to +20 mV	-10 to +10 mV

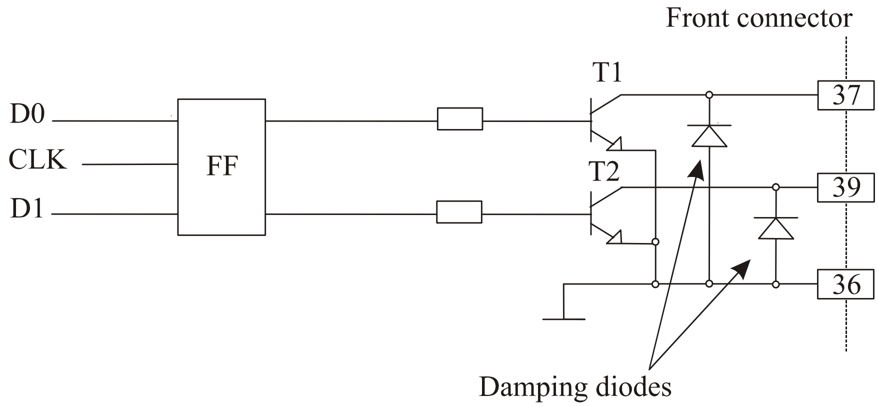
The table above is not exhaustive. You can set intermediate values by selecting the appropriate resistor value Rg. Maximum gain factor = 800.

7.10 Open Collector outputs

The board is equipped with 2 software programmable open collectors. They can be turned on or off through software.

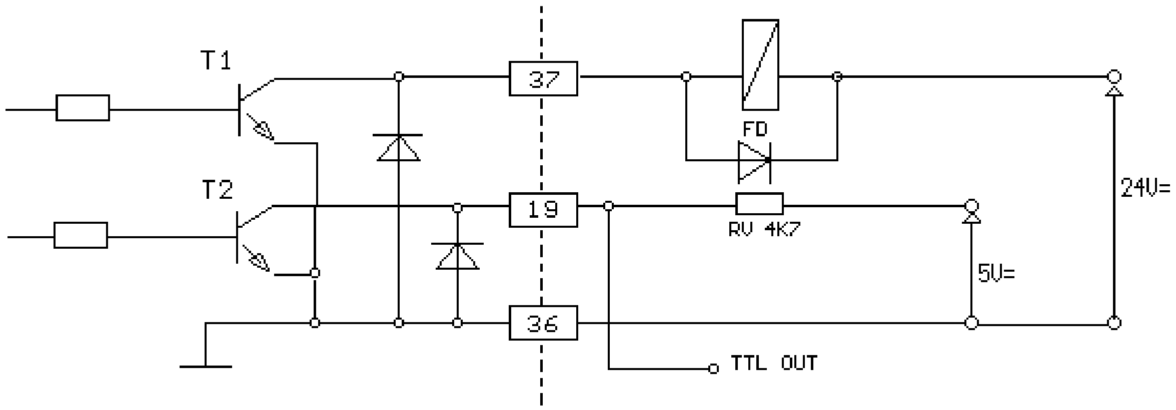
Fig. 7-18: Schematic connection of the open collector output channels

i max = 50 mA, Vcc = 25 V



Both open collectors operate independently from one another.
- Control through a flipflop (type D).
- When logic "1" is written in, they are switched ON.
  It means that the collector-emitter is connected through the digital ground.
- When logic "0" is written in the flipflop, they are switched OFF.
  T1 and T2 are responded on **Base +1**.
- The open collectors are on logic "0" after a system reset.
- They function with max. 25 V.
  The rest voltage of a turned on open collector is 0.5 V at 50 mA.
- Can be used a lamp or relay driver (T1).
  If you want to use a relay, make sure that the relay is equipped with a free-wheeling diode (FD). **It is necessary in order to avoid disturbances.**
- Can be used as a TTL compatible logic output (T2).

Fig. 7-19: Typical open collector wiring



7.10.1 TTL compatibility

The output is TTL-compatible:

- if you insert a 4K7 pull up resistor between the open collector output (pin 19 and 37 of the front connector)
- if you use a +5 V voltage supply.

Under these conditions, the outputs can control 10 standard TTL loads.

7.11 Calibration of the A/D converter

At delivery the board has already been calibrated.

If a new calibration is necessary, you have three potentiometers at your disposal.

- the reference voltage +5VREF is calibrated over TP3.
  It has already been adjusted by the manufacturer.
- zero alignment occurs over TP5
- gain is calibrated over TP4.

You will find the position of the potentiometers in the component scheme.

Before a new calibration, make sure that the board is in 8-bit data access (J7 and J8 not set).

The following program in BASIC simplifies the calibration.

Table 7-4: Calibration program in Basic

```
10    CLS:WIDTH 80                    ; SCREEN FORMAT
20    LOCATE 1,1                      ;
30    FOR A=0 TO 15                   ; NUMBER OF CHANNELS
                                      ; EX. 16 SINGLE ENDED
40    OUT &H390,A                     ; CONVERSION START
50    F=INP(&H391)                    ; READ STATUS
60    C=F                             ; TEMPORARY STORAGE OF
                                        STATUS BYTE
70    F=F AND &H80                    ; MASK EOC
80    IF F= &H80 GOTO 50              ; WHEN CONVERSION NOT
                                         COMPLETED
90    B=INP(&H390)                    ; READ LOW BYTE
100   C= (C AND &HOF)*256             ; H-BYTE, MASK, SHIFT
110   D=B+C                           ; 12-BIT VALUE
120   PRINT A,D,:PRINT HEX (D)        ;
130   NEXT A                          ; NEXT CHANNEL
140   GOTO 20                         ;
```

7.11.1 Calibration procedure

1. Select input range ± 5 V is selected
2. Set Gain = 1
3. Feed -1.22 mV in analog input 0
4. With TP5 adjust display value so that it moves between 0800H and 07FFH (zero alignment)
5. Feed +4.9963 V in analog input 0.
6. With TP4 adjust display value so that it is between 0FFEH and 0FFFH (gain adjustment)
7. Execute points 3 to 6 until no further calibration improvement is possible.
8. The calibration is completed
9. All analog inputs have to be checked with the voltage end values.

Table 7-5: Calibration voltages for zero alignment

A/D Input range	Input calibration voltage	Value
0 to +10 V	2,44 mV	0001H
-5 V to + 5 V	-4,997V	0001H
-10 V to +10 V	-9,995 V	0001H

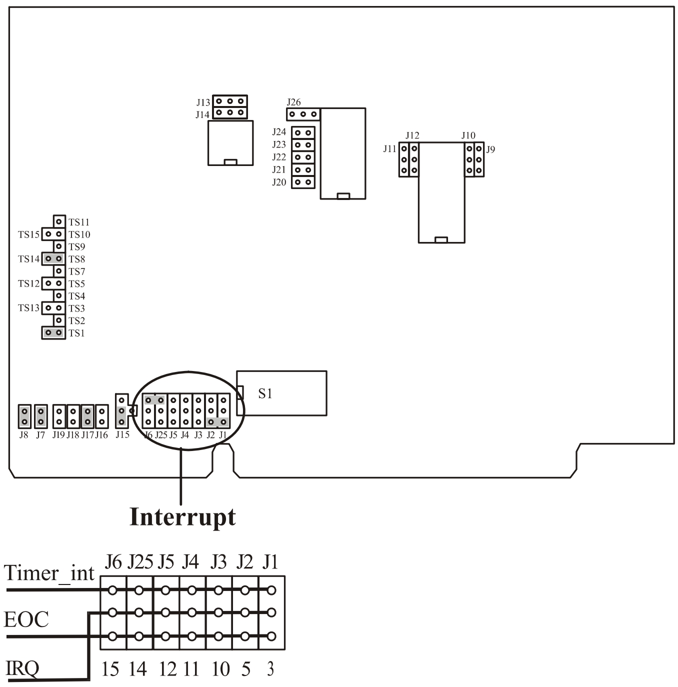
Table 7-6: Calibration voltages for gain calibration

A/D Input range	Input calibration voltage	Value
0 to +10 V	10 V	0FFFH
- 5 V to + 5 V	5 V	0FFFH
-10 V to +10 V	10 V	0FFFH

7.12 Interrupt

The board has two sources which can generate an interrupt request.
For XT: IRQ3, IRQ5
For AT: additionally IRQ10, 11, 12, 14, 15.

Fig. 7-20: Interrupt -Jumper settings



7.12.1 Interrupt at the end of conversion

An interrupt request is sent to the PC when a conversion is completed.
The interrupt is reset with a conversion start or a write command on a channel.

7.12.2 Timer interrupt

The timer can interrupt the PC cyclically with the signal TIMER_INT.
The cycle time is defined by software as described in the following example.

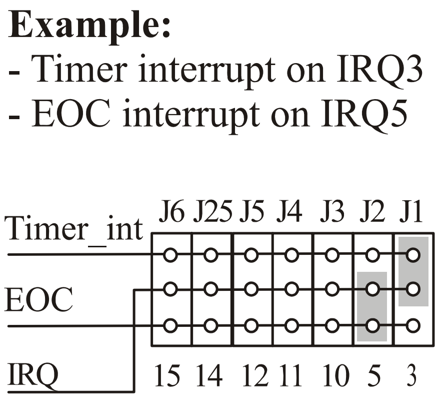
7.12.3 Example

The PC should be interrupted every 50 ms ± 1 ms.
Channel 0 is verified in the interrupt routine.
If the value YFF is exceeded, an open collector output is activated.

Procedure

1) Selection of the timer mode
   Timer 2 is used as a time generator in mode 2.
   Input frequency = 27.97 kHz.
2) Selection of the interrupt line. IRQ3 is selected through jumper J1.

Fig. 7-21: Interrupt - Example of jumper settings



3) Definition of the divider factor

The timer factor for timer 2 is calculated as follows:

$$0{,}050\ s = x * \frac{1}{27{,}97 * 10^{3}} = x * 35{,}75 * 10^{-6}\ s$$

$$x = \frac{0{,}05}{35{,}75 * 10^{-6}} = 1398\ \text{decimal} = \boxed{576\ \text{Hex}}$$

7.12.4 Interrupt routine

```
Program Pa302_Timer;
uses dos,crt;
CONST
(************************ Addresses of the different boards ******)
    pa302     = $390;                 (* Base address of PA 302 *)
    IRQ_CTRL1 = $20;(* Base address of Master Interrupt Controller*)
(******************** Control bytes ************************)
     EOI       = $20; (* End of interrupt for the IRQ Controller*)
     IRQ3_ENA = $F7;                  (* Enable mask for IRQ3 *)
     IRQ3_VECT_NR = $0B;   (* Vector number associated with IRQ3 *)
     LEVEL     = 3098;(* Maximum value before error in process  *)
(****************************** Type definition  ****************)
(******************** Global variables  ************************)
VAR
      oldvector           : pointer;
                                     (* Old value of IRQ vector *)
      oldstatus           : byte;
                         (* Old status of IRQ Controller EnaMask *)
      EOC                 : boolean;
      mux,i               : byte;
      value               : word;            (* Converted value *)
(************************ Function prototypes*****************)
Procedure  Disable;
BEGIN
inline ( $FA );                                    (*  CLI    *)
END;
(******************************************************************)
Procedure  Enable;
BEGIN
inline ( $FB );                                     (*  STI  *)
END;
(******************************************************************)
Procedure  New_Irq3_Handler(
flags,cs,ip,ax,bx,cx,dx,si,di,ds,es,bp:word);
interrupt;
BEGIN
disable;                              (* Disable interrupts *)
port[ PA302 ] := 0;                (* Start convert channel 0
*)
repeat
  value := portw[ PA302 ];            (* Read a 16-bit word *)
until (( value and $8000 ) = 0);     (* until EOC bit is low *)
if (value > level )                  (* Test if the level is *)
  then                                        (* passed over *)
  port[ PA302 + 1 ] := $01;                  (* Set output 1 *)
port[ irq_ctrl1 ] := EOI;                             (* EOI *)
enable;                                (* Enable interrupts *)
END;
(******************************************************************)
Procedure Install_New_Irq_3;
BEGIN
Disable;
getintvec (irq3_vect_nr , oldvector);     (* Save OLD IRQ vector *)
setintvec(irq3_vect_nr , @new_irq3_handler);
                                     [*Set the NEW IRQ vector *)
oldstatus := port[ irq_ctrl1 + 1 ];
port[ irq_ctrl1 + 1 ] := port[ irq_ctrl1 + 1 ] and irq3_ena;
                                     (* Enable IRQ channel 3 *)
port[ irq_ctrl1 ] := EOI;                             (* EOI *)
Enable;
END;
(******************************************************************)
Procedure  Desinstall_Irq_3;
BEGIN
Disable;
setintvec( irq3_vect_nr , oldvector ); (* Set the OLD IRQ vector *)
port[ irq_ctrl1 + 1 ] := oldstatus;   (* Set the OLD enable mask *)
Enable;
END;
```

```
(*****************************************************************)
Procedure Timer_Start ( divisor : word );
CONST timer_mode = $B4;                         (* TIMER 2 im mode 2 *)
VAR   lowbyte , highbyte : byte;
Begin
 lowbyte  := divisor Mod 256;
 highbyte := divisor Div 256;
 port[ pa302 + 7 ] := Timer_Mode;       (* TIMER 2  control byte *)
 delay(1);
 port[ pa302 + 6 ] := Lowbyte; (* TIMER 2 initial value low byte *)
 delay(1);
 port[ pa302 + 6 ] := Highbyte;(*TIMER 2 initial value high byte *)
End;
(*****************************************************************)
(*                               MAIN                            *)
(*****************************************************************)
BEGIN
  clrscr;
  value  := 0;
  Install_New_Irq_3;                          (* Install the NEW IRQ *)
  Timer_Start ( $576 );          (* Timer conversion every 50 mS *)

                                                (*  User application   *)
  Desinstall_Irq_3;
```

8 OPTIONS

8.1 Option SF, DF

With option XF two cablings with filter are possible.
The filters used are low-pass filters of 1st order -20dB/decade.

Fig. 8-1: PA 302 board SF (single-ended with input filter)

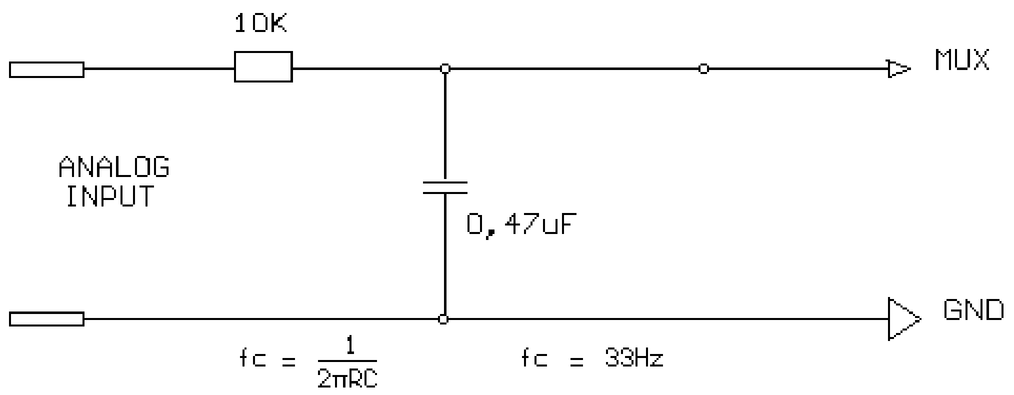
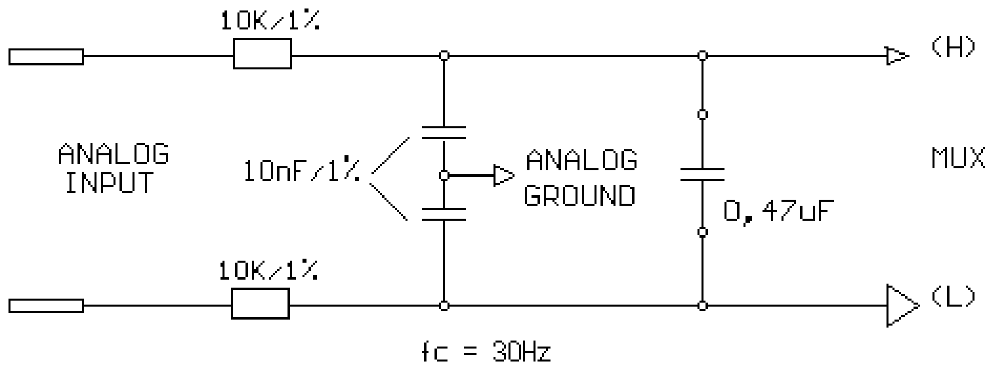


Fig. 8-2: PA 302 board DF (differential with input filter)



8.2 Option SC, DC

With option C the inputs are wired as follows :

Fig. 8-3: PA 302 board SC (single ended with current converter)

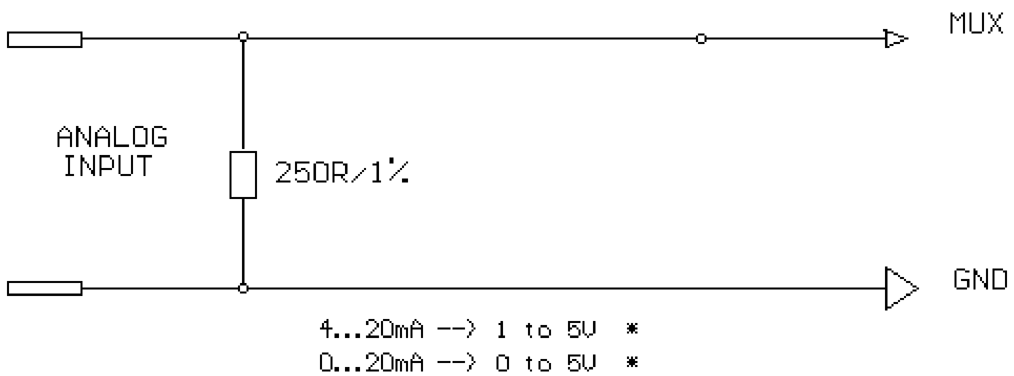
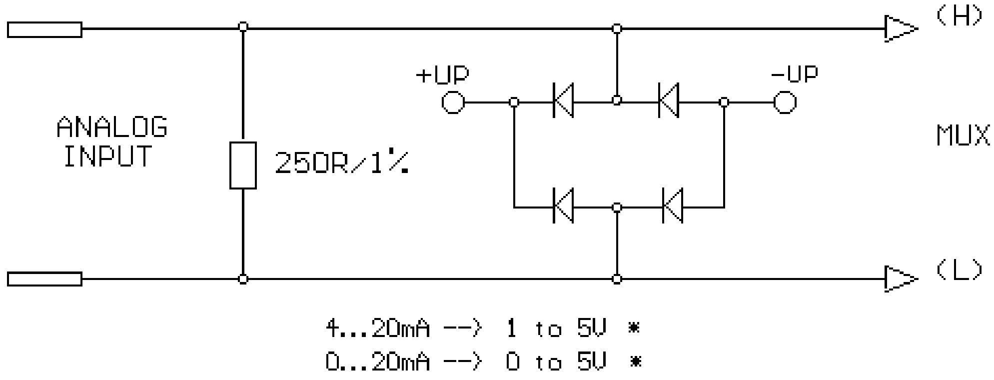


Fig. 8-4: PA 302 board DC (differential with current converter)



* With option C, the voltage range should be set at 0-10 V and the gain factor at 2.

9 CONNECTION TO THE PERIPHERAL

9.1 Connector pin assignment

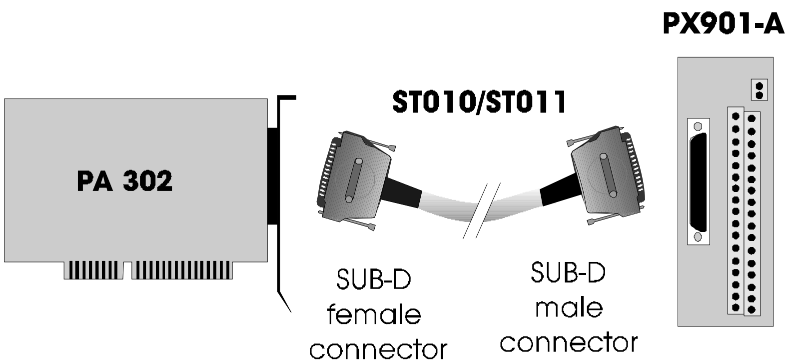
Fig. 9-1: Connector pin assignment

User designation	DIFF	SE	Pin
	Logic driver 0	Logic driver 0	19
	Analog GND	Analog GND	18
	Analog GND	Analog GND	17
	Analog GND	Analog GND	16
	Analog GND	Analog GND	15
	Analog GND	Analog GND	14
	Analog GND	Analog GND	13
	Analog GND	Analog GND	12
	Analog GND	Analog GND	11
	Analog GND	Analog GND	10
	Ext. trigger	Ext. trigger	9
	(-) An. input 4	(+) An. input 12	8
	(-) An. input 5	(+) An. input 13	7
	(-) An. input 6	(+) An. input 14	6
	(-) An. input 7	(+) An. input 15	5
	(+) An. input 7	(+) An. input 11	4
	(+) An. input 6	(+) An. input 10	3
	(+) An. input 5	(+) An. input 9	2
	(+) An. input 4	(+) An. input 8	1

Pin	SE	DIFF	User designation
37	Logic driver 1	Logic driver 1	
36	Digital GND	Digital GND	
35	Analog GND	Analog GND	
34	Analog GND	Analog GND	
33	Analog GND	Analog GND	
32	Analog GND	Analog GND	
31	Analog GND	Analog GND	
30	Analog GND	Analog GND	
29	Analog GND	Analog GND	
28	+ 5 V	+ 5 V	
27	(+) An. input 4	(-) An. input 0	
26	(+) An. input 5	(-) An. input 1	
25	(+) An. input 6	(-) An. input 2	
24	(+) An. input 7	(-) An. input 3	
23	(+) An. input 3	(+) An. input 3	
22	(+) An. input 2	(+) An. input 2	
21	(+) An. input 1	(+) An. input 1	
20	(+) An. input 0	(+) An. input 0	

9.1.2 Connection example

Fig. 9-2: Connection to the screw terminal panel PX 901-A



10 DEVICE DRIVER

IMPORTANT!
Note the following conventions in the text:

Function:	"i_PA302_SetBoardInformation"
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

Table 10-3: Define value

Define name	Decimal value	Hexadecimal value
DLL_COMPILER_C	0	0
DLL_COMPILER_VB	1	1
DLL_COMPILER_PASCAL	2	2
DLL_COMPILER_LABVIEW	3	3
DLL_COMPILER_VB_5	4	4
PA302_DISABLE	0	0
PA302_ENABLE	1	1
PA302_SYNCHRONOUS_MODE	1	1
PA302_ASYNCHRONOUS_MODE	0	0
PA302_LOW_FREQUENCY	0	0
PA302_HIGH_FREQUENCY	1	1
PA302_8BIT	8	8
PA302_16BIT	16	10

10.1 Base address and interrupt

10.1.1 Base address

IMPORTANT!
This function is only available for DOS and Windows 3.11.

1) i_PA302_SetBoardInformation (...)

Syntax:

```
<Return Value> = i_PA302_SetBoardInformation
                        (UINT     ui_Address,
                         BYTE     b_EndOfConvertInterruptNbr,
                         BYTE     b_Timer2InterruptNbr,
                         BYTE     b_AccessMode,
                         BYTE     b_AnalogInputChannelNbr,
                         PBYTE    pb_BoardHandle)
```

Parameter:

UINT	ui_Address	Base address of the **PA 302** board
BYTE	b_EndOfConvertInterruptNbr	Interrupt line of the board for the end of conversion (IRQ3, 5, 10, 11, 12, 14 or 15) At 0 no interrupt line is used.
BYTE	b_Timer2InterruptNbr	Interrupt line of the board for the third timer (IRQ3, 5, 10, 11, 12, 14 or 15) At 0 no interrupt line is used.
BYTE	b_AccessMode	**PA302** access mode PA302_8BIT: 8-bit access PA302_16BIT: 16-bit access
BYTE	b_AnalogInputChannelNbr	Number of analog inputs
PBYTE	pb_BoardHandle	Handle[1)] of the **PA 302** board for using the functions

Task:

Verifies if the **PA 302** board is present. Stores the base address and the number of analog inputs. A handle is returned to the user which allows to use the next functions. Handles allow to operate several boards.
Make sure that one of the parameter b_EndOFConvertInterruptNbr or b_Timer2InterruptNbr are not 0.

Calling convention

ANSI C :

```
int          i_ReturnValue;
unsigned char    b_BoardHandle;
i_ReturnValue = i_PA302_SetBoardInformation(0x390,
                                             0,
                                             3,
                                             PA1500_8BIT,
                                             16,
                                             &b_BoardHandle,);
```

Return value:

0: No error
-1: Number of analog inputs is wrong
-2: No handle available for the board (only 10 handles available)
-3: Error by opening the driver under Windows NT/95.
-4: Interrupt number already occupied
-5: Interrupt number not available
-6: 2 interrupts cannot be the same.

2) i_PA302_SetBoardInformationWin32 (...)

i

IMPORTANT!
This function is only available under Windows (32-bit).

Syntax:

```
<Return Value> = i_PA302_SetBoardInfomationWin32
                        (PCHAR    pc_Identidier,
                         BYTE     b_AccessMode,
                         BYTE     b_AnalogInputChannelNbr,
                         PBYTE    pb_BoardHandle)
```

Parameter:

PCHAR	pc_Identifier	Identifikations-Zeichenkette der PA302 („PA302-XX“)
BYTE	b_AccessMode PA302_8BIT: 8-bit access PA302_16BIT: 16-bit access	**PA302** access mode
BYTE	b_AnalogInputChannelNbr	Number of analog inputs
PBYTE	pb_BoardHandle	Handle[1] of the **PA 302** board for using the functions

Task:

Verifies if the **PA 302** board is present. Stores the number of analog inputs. A handle is returned to the user which allows to use the next functions. Handles allow to operate several boards.
The identifier string is as follows:
„PA302-XX“ XX corresponds to the number given to the board in ADDIREG

Calling convention:

ANSI C :

```
int           i_ReturnValue;
unsigned char    b_BoardHandle;

i_ ReturnValue = i_PA302_SetBoardInformationWin32 („PA302-00“,
                                                   PA1500_8BIT,
                                                   16,
                                                   &b_BoardHandle,);
```

Return value:

0: No error

[1] Identifikationsnummer der Karte

-1: Number of analog inputs is wrong
-2: No handle available for the board (up to 10 handles available)
-3: Error by opening the driver under Windows NT/95.
-4: Interrupt number already occupied
-5: Interrupt number not available
-6: 2 interrupts cannot be the same.

3) i_PA302_GetHardwareInformation(...)

Syntax:

```
<Return value> =      i_PA302_GetHardwareInformation
(BYTE                        b_BoardHandle,
 PUINT                       pui_BaseAddress,
 PBYTE                       pb_EOCIRQNbr,
 PBYTE                       pb_Timer2IRQNbr)
```

Parameter:

BYTE	b_BoardHandle	Handle of the **PA302** board
PUINT	pui_BaseAddress	Base address
PBYTE	pb_EOCIRQNbr	EOC interrupt number.
PBYTE	pb_Timer2IRQNbr	Interrupt number of timer2

Task:
Returns the base address and the interrupt lines.

Calling convention:
ANSI C :

```
int                 i_ReturnValue;
unsigned int        ui_BaseAddress;
unsigned char       b_Timer2IRQNbr;
unsigned char       b_EOCIRQNbr;
unsigned char       b_BoardHandle;

i_ReturnValue = i_PA302_GetHardwareInformation   (b_BoardHandle,
                        &ui_BaseAddress,
                        &b_EOCIRQNbr,
                        &b_Timer2IRQNbr);
```

Return value:
0: No error
-1: Wrong board handle parameter

10.1.2 Interrupt

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

1) i_PA302_SetBoardIntRoutineDos (..)

Syntax:
<Return value> = i_PA302_SetBoardIntRoutineDos
(BYTE b_BoardHandle,
VOID v_FunctionName (BYTEb_BoardHandle,
BYTE b_InterruptMaske,
PUINT pui_AnalogInputValue))

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302 board**
VOID	v_FunctionName	name of the user interrupt routine.

IMPORTANT!
B_EndOfConvertInterruptNbr and *b_Timer2InterruptNbr* cannot have the same value and cannot be on 0 simultaneously. (See i_PA302_SetBoardInformationXX(...)).

Task:
This function can be called up several times.

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

If you operate several **PA 302** boards which have to react to interrupts, call up the function as often as you operate **PA 302** boards. The variable *v_FunctionName* is only relevant **for the first calling**.

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

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

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.

An interrupt is enabled
- when timer2 has run down
- at the end of conversion

The interrupt management is easier by using the function i_PA302_SetBoardIntRoutine

The user interrupt routine must have the following syntax:

VOID *v_FunctionName* (BYTE *b_BoardHandle*,
BYTE *b_InterruptMask*,
PUINT *pui_AnalogInputValue*)

v_FunctionName Name of the user interrupt routine

b_BoardHandle Handle of the **PA 302** which has generated the interrupt

b_InterruptMask Mask of the events which have generated the interrupt.

pui_AnalogInputValue The values of the analog input channels are returned.
- When *b_InterruptMask* equals 1, *pui_AnalogInputValue [0]* contains the number of the last analog input and *pui_AnalogInputValue [1]* the last value of the cyclic conversion
- When *b_InterruptMask* equals 2, *pui_AnalogInputValue [0]* contains the number of the last analog input and *pui_AnalogInputValue [1]* the last value of the conversion driven by timer.

Table 10-4: Interrupt mask

Mask	Meaning
0000 000**1**	End of Conversion (EOC)
0000 00**1**0	Conversion driven by timer is completed
0000 0**1**00	Timer 2 has run down

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

Calling convention

ANSI C :

```
void    v_FunctionName     (unsigned char   b_BoardHandle,
                            unsigned char   b_InterruptMaske,
                            unsigned int *  pui_AnalogInputValue)
                           int              i_ReturnValue;
                           unsigned char    b_BoardHandle;
        i_ReturnValue = i_PA302_SetBoardIntRoutineDos(b_BoardHandle,
                                                        v_FunctionName );
```

Return value:
0: No error
-1: Handle parameter of the board is wrong
-2: All interrupt lines cannot have the value 0 or
interrupt already installed

i **IMPORTANT!**
This function is only available for Visual Basic DOS.

2) i_PA302_SetBoardIntRoutineVBDos (..)

Syntax:
<Return value> = i_PA302_SetBoardIntRoutineVBDos
(BYTEb_BoardHandle)

Parameter:
Input:

BYTE	b_BoardHandle	Handle of the PA 302 board

Output:
No output signal has occurred

Task:

This function must be called up for each **PA302** on which you want to enable an interrupt. If an interrupt occurs, a Visual basic event is generated.
Refer to the calling convention.

From the first callup of the function:
- interrupts are enabled for the selected board.

If you operate several **PA302** boards which have to react to interrupts, call up the function as often as you operate **PA302** boards.

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

Controlling the interrupt management
Please use instead the following functions

"ON UEVENT GOSUB xxxxxxxxx" of Visual Basic for DOS

and

"i_PA302_TestInterrupt"

This function tests the interrupt of the **PA302** board. It is used to obtain the values of *b_BoardHandle* , *b_InterruptMask*, *pui_AnalogInputValue*.

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_AnalogInputValue( )    As Long

IntLabel:

  i_ReturnValue = i_PA302_TestInterrupt   (i_BoardHandle, _
                                           i_InterruptMask, _
                                           l_AnalogInputValue (0) )
  .
  .
  .

  Return


ON UEVENT GOSUB IntLabel
UEVENT ON
i_ReturnValue = i_PA302_SetBoardIntRoutineVBDos    (b_BoardHandle)
```

Return value:

0: No error
-1: Handle parameter of the board is wrong
-2: All interrupt lines cannot have the value 0 or
interrupt already installed

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

3) i_PA302_SetBoardIntRoutineWin16 (..)

Syntax:

```
<Return value> = i_PA302_SetBoardIntRoutineWin16
            (BYTE     b_BoardHandle,
             VOID     v_FunctionName  (BYTE     b_BoardHandle,
                                       BYTE     b_InterruptMaske,
                                       PUINT    pui_AnalogInputValue))
```

Parameter:

- Input:

BYTE	b_BoardHandle	Handle of the **PA 302**
VOID	v_FunctionName	Name of the user interrupt routine.

IMPORTANT!
B_EndOfConvertInterruptNbr and *b_Timer2InterruptNbr* cannot have the same value and cannot be set to 0 simultaneously.
(In i_PA302_SetBoardInformationXX(...)).

- Output:
No output signal has occurred

Task:

***This function can be called up several times*.**

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

If you operate several boards **PA 302** which have to react to interrupts, call up the function as often as you operate boards **PA 302**. The variable *v_FunctionName* is only relevant **for the first calling**.

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

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

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.

An interrupt is enabled
- when timer2 has run down
- at the end of conversion

The interrupt management is easier by using the function i_PA302_SetBoardIntRoutine

The user interrupt routine must have the following syntax:

VOID *v_FunctionName* (BYTE*b_BoardHandle*,
BYTE *b_InterruptMask*,
PUINT *pui_AnalogInputValue*)

v_FunctionName Name of the user interrupt routine
b_BoardHandle Handle of the **PA 302** which has generated the interrupt
b_InterruptMask Mask of the events which have generated the interrupt.
pui_AnalogInputValue The values of the analog input channels are returned.
- When *b_InterruptMask* equals 1, *pui_AnalogInputValue [0]* contains the number of the last analog input and *pui_AnalogInputValue [1]* the last value of the cyclic conversion
- When *b_InterruptMask* equals 2, *pui_AnalogInputValue [0]* contains the number of the last analog input and *pui_AnalogInputValue [1]* the last value of the conversion driven by timer.

Table 10-3: Interrupt mask

Mask	Meaning
0000 000**1**	End of Conversion (EOC)
0000 00**1**0	Conversion driven by timer is completed
0000 0**1**00	Timer 2 has run down

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

Calling convention

ANSI C :

```
void     v_FunctionName     (unsigned char   b_BoardHandle,
                             unsigned char   b_InterruptMaske,
                             unsigned int *  pui_AnalogInputValue)

int         i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA302_SetBoardIntRoutine Win16        (b_BoardHandle,
                                                          v_FunctionName );
```

Return value:

0: No error

-1: Handle parameter of the board is wrong

-2: All interrupt lines cannot have the value 0 or interrupt already installed

IMPORTANT!
This function is only available under Windows NT and Windows 95.

4) i_PA302_SetBoardIntRoutineWin32 (..)

Syntax:

```
<Return value> = i_PA302_SetBoardIntRoutineWin32
        (BYTE      b_BoardHandle,
         BYTE      b_UserCallingMode,
         ULONG     ul_UserSharedMemorySize,
         VOID      ** ppv_UserSharedMemory,
         VOID      v_FunctionName (BYTE   b_BoardHandle,
                        BYTE     b_InterruptMask,
                        PUINT    pui_AnalogInputValue,
                        BYTE     b_UserCallingMode,
                        VOID *   pv_UserSharedMemory))
```

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of the **PA 302** board
BYTE	b_UserCallingMode	PA302_SYNCHRONOUS_MODE : The user routine is directly called by the driver interrupt routine. PA302_ASYNCHRONOUS_MODE : The user routine is called by the driver interrupt thread.
VOID	v_FunctionName	Name of the user interrupt routine
ULONG	ul_UserSharedMemorySize	Determines the size in byte of the user shared memory. Only used if you have selected PA302_SYNCHRONOUS_MODE

- Output:

VOID ** ppv_UserSharedMemory	User shared memory address Only used if you have selected PA302_SYNCHRONOUS_MODE

Task:

If you use Visual Basic 5.0 :
- only the asynchronous mode is available.

<u>Windows 32-bit information</u>

For Windows NT and Windows 95, 4 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. The 2 pointers are not configured under the same address.

This function must be called up for each **PA 302** for which an interrupt is to be enabled. It installs one 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
- user shared memory is allocated if PA302_SYNCHROUNOUS_MODE has been selected.

If you operate several **PA 302** boards which have to react to interrupts, call up the function as often as you operate **PA 302** boards. The variable *v_FunctionName* is only relevant **for the first calling**.

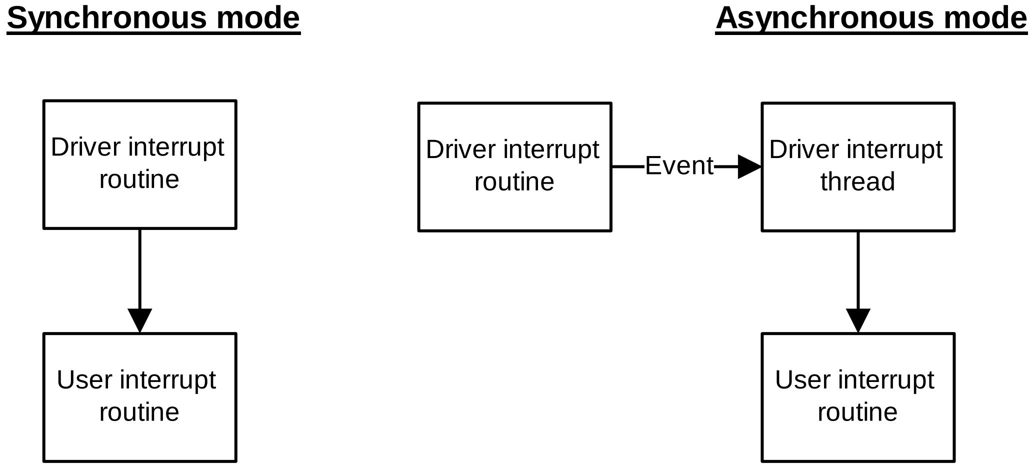
From the second call of the function (next board):
- interrupts are enabled.

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

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 have 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 the interrupt and the user interrupt routine is reduced.
RESTRICTIONS	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 still use a shared memory.
	The user routine can only call **PA 302** driver functions with the following extension "i_PA302_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 302** driver functions with the following extension: "i_PA302_XXXX"
RESTRICTIONS	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 PA302_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,
                        PUINT     b_AnalogInputValue,
                        BYTE      b_UserCallingMode,
                        VOID *    pv_UserSharedMemory)
```

v_FunctionName	Name of the user interrupt routine
b_BoardHandle	Handle of the **PA 302** which has generated the interrupt
b_InterruptMask	Mask of the events which have generated the interrupt.
pui_AnalogInputValue	The latched values of the counter are returned.
b_UserCallingMode	PA302_SYNCHRONOUS_MODE : The user routine is directly called by driver interrupt routine. PA302_ASYNCHRONOUS_MODE : The user routine is called by driver interrupt thread
pv_UserSharedMemory	Pointer of the user shared memory.

i

IMPORTANT!
If you use Visual Basic 4 the following parameters have no meaning. You must used the „i_PA302_TestInterrupt“ function.

```
BYTE       b_UserCallingMode,
ULONG      ul_UserSharedMemorySize,
VOID **    ppv_UserSharedMemory,
VOID       v_FunctionName      (BYTE b_BoardHandle,
                                BYTE        b_InterruptMask,
                                PUINT       pui_AnalogInputValue,
                                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 int    *ui_AnalogInputValue,
                               unsigned char   b_UserCallingMode,
                               void *          pv_UserSharedMemory)
          {
          str_UserStruct * ps_InterruptSharedMemory;

          ps_InterruptSharedMemory = (str_UserStruct *) pv_UserSharedMemory;
          .
          .
          }
int       i_ReturnValue;
unsigned char    b_BoardHandle;

i_ReturnValue = i_ PA302_SetBoardIntRoutineWin32
                                        (b_BoardHandle,
                                         PA302_SYNCHRONOUS_MODE,
```

sizeof (str_UserStruct),
(void **)
&ps_UserSharedMemory,
v_FunctionName);

Return value:
0: No error
-1: Wrong board handle parameter
-2: All interrupt lines cannot have the value 0 or
interrupt already installed
-3: Parameter b_UserCallingMode ist falsch.

5) i_PA302_ResetBoardIntRoutine (..)

Syntax:
<Return value> = i_PA302_ResetBoardIntRoutine (BYTE b_BoardHandle)

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board

Task:
Stops the interrupt management of PA 302 board**.**
Uninstalls the user interrupt routine if the management of interrupts of all boards **PA 302** is stopped.

Calling convention:
ANSI C :

```
int           i_ReturnValue;
unsigned char b_BoardHandle;

i_ReturnValue = i_PA302_ResetBoardIntRoutine (b_BoardHandle);
```

Return value:
0: No error
-1: Wrong board handle parameter
-2: The user interrupt routine is not installed

6) i_PA302_TestInterrupt (..)

Syntax:

<Return value> = i_PA302_TestInterrupt (PBYTE pb_BoardHandle,
PBYTE pb_InterruptMask ,
PUINT pui_AnalogInputValue)

Parameter:

PBYTE	pb_BoardHandle	Handle of the **PA 302** board which has generated the interrupt
PBYTE	pb_InterruptMask	Mask of the event(s) which has (have) generated the interrupt. Several events can occur simultaneously
PUINT	pui_AnalogInputValue	The values of the analog inputs are returned.

- If *b_InterruptMask* equals 1, *pui_AnalogInputValue [0]* contains the number of the last analog input and *pui_AnalogInputValue [1]* the last value of the cyclic conversion.
- If *b_InterruptMask* equals 2, *pui_AnalogInputValue [0]* contains the number of the last analog input and *pui_AnalogInputValue [1]* the last value of the conversion driven by timer.

Mask	Meaning
0000 0001	End of Conversion (EOC)
0000 0010	Conversion driven by timer is completed
0000 0100	Timer 2 has run down

Task:

Verifies if a **PA 302** board has generated an interrupt. If yes, the function returns the board's handle and the interrupt source.

i

IMPORTANT!
This function is only available under Visual Basic for DOS and Windows.

Calling convention:

ANSI C :

```
int            i_ReturnValue;
unsigned char  b_BoardHandle;
unsigned char  b_InterruptMask;
unsigned int   ui_AnalogInputValue [XX];

i_ReturnValue = i_PA302_TestInterrupt      (&b_BoardHandle,
                                            &b_InterruptMask,
                                            ui_AnalogInputValue);
```

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

7) i_PA302_InitCompiler (..)

Syntax:
<Return value> = i_PA302_InitCompiler (BYTE b_CompilerDefine)

Parameter:

BYTE	b_CompilerDefine	You choose the language (under Windows) in which you want 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_LABVIEW:	The user programs in Labview.
	- DLL_COMPILER_PASCAL:	The user programs in Pascal

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

i

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

Calling convention:
ANSI C :
```
int i_ReturnValue;

i_ReturnValue = i_PA302_InitCompiler (DLL_COMPILER_C);
```

Return value:
0: No error
-1: Wrong compiler parameter

8) i_PA302_CloseBoardHandle (...)

i

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

Syntax:
<Return value> = i_PA302_CloseBoardHandle (BYTE b_BoardHandle)

Parameter:
BYTE b_BoardHandle Handle of **PA 302** board

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

Calling convention:
ANSI C :

```
int           i_ReturnValue;
unsigned char b_BoardHandle;

i_ReturnValue = i_PA302_CloseBoardHandle (b_BoardHandle);
```

Return value:
0: No error
-1: Wrong board handle parameter

10.2 Direct conversion of the analog inputs

1) i_PA302_Read1AnalogInput (...)

Syntax:

```
<Return value> =  i_PA302_Read1AnalogInput
                        (BYTEb_BoardHandle,
                         BYTE       b_Channel,
                         PUINT      pui_AnalogInputValue)
```

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board
BYTE	b_Channel	Number of the analog input to be read.
PUINT	pui_AnalogInputValue (0 to 4095)	The analog value is returned

Task:

Reads the current values of the analog input *b_Channel*.

Calling convention:

ANSI C :

```
int            i_ReturnValue;
unsigned char  b_BoardHandle;
unsigned int   ui_AnalogInputValue;


i_ReturnValue = i_PA302_Read1AnalogInput
                      (b_BoardHandle,
                       1,
                       &ui_AnalogInputValue);
```

Return value:

0: No error
-1: Wrong board handle parameter
-2: The number of the analog input is wrong. See function "i_PA302_SetBoardInformation"

2) i_PA302_ReadSevAnalogInput (...)

Syntax:

```
<Return value> =  i_PA302_ReadSevAnalogInput
                         (BYTEb_BoardHandle,
                          BYTE      b_FirstChannelNbr,
                          BYTE      b_NbrOfChannel,
                          PUINT     pui_AnalogInputValueArray)
```

Parameter:

BYTE b_BoardHandle		Handle of the **PA 302** board
BYTE b_FirstChannelNbr		Number of the first analog input
BYTE b_NbrOfChannel		Number of analog inputs to be read
PUINT pui_AnalogInputValue		The table of the output values is returned

Task:

Reads the current value of several analog inputs.

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned int    ui_AnalogInputValue[4];


   i_ReturnValue = i_PA302_ReadSevAnalogInput
                          (b_BoardHandle,
                    1,
                    4,
                    ui_AnalogInputValue[0]);
```

Return value:

0: No error
-1: Wrong board handle parameter
-2: Number of the first analog input is wrong.
See function "i_PA302_SetBoardInformation"
-3: The number of analog inputs you want to read is wrong
See function "i_PA302_SetBoardInformation"

10.3 Cyclic conversion of analog inputs

1) i_PA302_InitAnalogInputAcquisition (...)

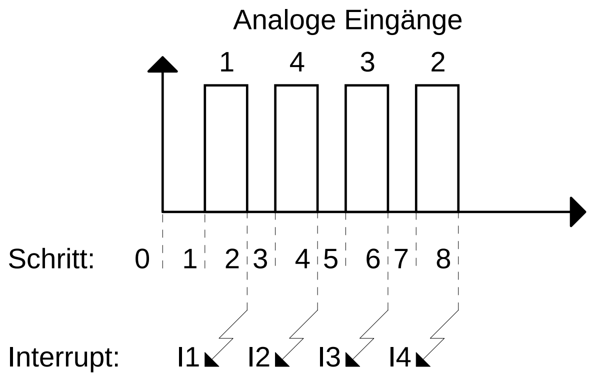
Syntax:
<Return value> = i_PA302_InitAnalogInputAcquisition
(BYTEb_BoardHandle)

Parameter:
BYTE b_BoardHandle Handle of the **PA 302** board

Task:
Allows to acquire several analog inputs successively.

An interrupt is generated at the end of conversion. In your user interrupt routine a „1“ is passed with the variable *b_InterruptMask*.
The value of the analog input is passed with the variable *pui_AnalogInputValue*.
See function "i_PA302_SetBoardIntRoutine".
You start the next conversion with the function "i_PA302_SetNextAnalogInput".

Example:



- Step 0: You call up the function "i_PA302_InitAnalogInputAcquisition (...)"
- Step 1: You start the conversion of analog input 1 with function "i_PA302_StartAnalogInputAcquisition (...)"
- Step 2: An interrupt is generated when the conversion of analog input 1 is completed (I1)
- Step 3: You start the conversion of analog input 4 with function "i_PA302_SetNextAnalogInput (...)"
- Step 4: An interrupt is generated when the conversion of analog input 4 is completed (I2)
- Step 5: You start the conversion of analog input 3 with function "i_PA302_SetNextAnalogInput (...)"
- Step 6: An interrupt is generated when the conversion of analog input 3 is completed (I3)
- Step 7: You start the conversion of analog input 2 with function "i_PA302_SetNextAnalogInput (...)"
- Step 8: An interrupt is generated when the conversion of analog input 2 is completed (I4)

You stop acquisition with function "i_PA302_StopAnalogInputAcquisition (...)"

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA302_InitAnalogInputAcquisition(b_BoardHandle);
```

Return value:
0: No error
-1: Wrong board handle parameter
-2: The user interrupt routine is not installed. See function "i_PA302_SetBoardIntRoutine"

2) i_PA302_StartAnalogInputAcquisition (...)

Syntax:

<Return value> = i_PA302_StartAnalogInputAcquisition
(BYTEb_BoardHandle,
BYTE b_Channel)

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board
BYTE	b_Channel	Number of the first analog input to be read.

Task:
Starts the cyclic conversion of several analog inputs.

Calling convention:
ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue =
i_PA302_StartAnalogInputAcquisition(b_BoardHandle,1);
```

Return value:
0: No error
-1: Wrong board handle parameter
-2: Number of the analog input is wrong. See function "i_PA302_SetBoardInformation"
-3: The cyclic conversion of analog inputs has not been initialised. Please use function "i_PA302_InitAnalogInputAcquisition"

3) i_PA302_SetNextAnalogInput (...)

Syntax:
<Return value> = i_PA302_SetNextAnalogInput
(BYTE b_BoardHandle,
BYTE b_Channel)

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board
BYTE	b_Channel	Number of the next analog input to be read.

Task:
Selects the number of the next analog input (*b_Channel)* to be acquired.

Calling convention:
ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA302_SetNextAnalogInput(b_BoardHandle,2);
```

Return value:
0: No error
-1: Wrong board handle parameter
-2: Number of the analog input is wrong. See function "i_PA302_SetBoardInformation"
-3: The cyclic conversion of analog inputs has not been started. Please use function "i_PA302_StartAnalogInputAcquisition"

4) i_PA302_StopAnalogInputAcquisition (...)

Syntax:
<Return value> = i_PA302_StopAnalogInputAcquisition
(BYTEb_BoardHandle)

Parameter:
BYTE b_BoardHandle Handle of the **PA 302** board

Task:
Stops the cyclic conversion of several analog inputs.

Calling convention:
ANSI C :

```
int            i_ReturnValue;
unsigned char  b_BoardHandle;

i_ReturnValue = i_PA302_StopAnalogInputAcquisition(b_BoardHandle)
```

Return value:
0: No error
-1: Wrong board handle parameter
-2: The cyclic conversion of analog inputs has not been started.
Please use function "i_PA302_StartAnalogInputAcquisition"

10.4 Conversion of analog inputs driven by timer

1) i_PA302_InitTimerAnalogInputAcquisition (...)

Syntax:

<Return value> = i_PA302_InitTimerAnalogInputAcquisition
(BYTEb_BoardHandle,
BYTE b_ChannelNbr,
BYTE b_TimerInputFrequency,
LONG l_AcquisitionTiming)

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board
BYTE	b_ChannelNbr	Number of analog inputs to be converted. The first input is always analog input 1, You enter the number of the last analog input with *b_ChannelNbr*
BYTE	b_TimerInputFrequency	Input frequency of timer 0: PA302_LOW_FREQUENCY: 27,901 kHz
	PA302_HIGH_FREQUENCY:	892,857 kHz
LONG	l_AcquisitionTiming	Time interval in µs between 2 conversions of successive analog inputs

Task:

Allows to acquire several analog inputs successively without starting conversion for each input per software.
An interrupt is generated at the end of conversion. In your user interrupt routine a „2“ is passed with the variable *b_InterruptMask*.
The value of the analog input is passed with the variable *pui_AnalogInputValue*. See function "i_PA302_SetBoardIntRoutineXX".
You enter the time between two conversions with the variable *l_AcquisitionTiming*.

Table 10-5: Selecting the time interval

b_TimerInputFrequency	l_AcquisitionTiming minimum	l_AcquisitionTiming maximum
PA302_LOW_FREQUENCY	3 µs	73399 µs
PA302_HIGH_FREQUENCY	72 µs	2348841 µs

Calling convention:

ANSI C :

```
int            i_ReturnValue;
unsigned char  b_BoardHandle;

i_ReturnValue = i_PA302_InitTimerAnalogInputAcquisition
                                        (b_BoardHandle,
                                         8,
                                         PA302_LOW_FREQUENCY,
                                         1000);
```

Return value:

0: No error
-1: Wrong board handle parameter
-2: The user interrupt routine is not installed. See function "i_PA302_SetBoardIntRoutineXX"
-3: Number of the analog inputs is wrong
-4: Selected input frequency is wrong
-5: Selected time interval is wrong

2) i_PA302_StartTimerAnalogInputAcquisition (...)

Syntax:

<Return value> = i_PA302_StartTimerAnalogInputAcquisition
(BYTEb_BoardHandle)

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board

Task:

Starts the conversion of successive inputs.
Starts the conversion of the first analog input.

Calling convention:

ANSI C :

```
int            i_ReturnValue;
unsigned char  b_BoardHandle;

i_ReturnValue = i_PA302_StartTimerAnalogInputAcquisition
                                        (b_BoardHandle);
```

Return value:

0: No error
-1: Wrong board handle parameter
-2: The time interval between the conversion of successive inputs has not been initialised. See function "i_PA302_InitTimerAnalogInputAcquisition"

3) i_PA302_StopTimerAnalogInputAcquisition (...)

Syntax:
<Return value> = i_PA302_StopTimerAnalogInputAcquisition
(BYTEb_BoardHandle)

Parameter:
BYTE b_BoardHandle Handle of the **PA 302** board

Task:
Stops the conversion of successive inputs.

Calling convention:
ANSI C :

```
int           i_ReturnValue;
unsigned char b_BoardHandle;

i_ReturnValue = i_PA302_StopTimerAnalogInputAcquisition
                                        (b_BoardHandle);
```

Return value:
0: No error
-1: Wrong board handle parameter
-2: The time interval between the conversion of successive inputs has not been started. See "i_PA302_StartTimerAnalogInputAcquisition"

10.5 Timer

1) i_PA302_InitTimer1 (...)

Syntax:

```
<Return value> = i_PA302_InitTimer1
                     (BYTEb_BoardHandle,
                      BYTE      b_TimerInputFrequency,
                      LONG      l_DelayValue)
```

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board
BYTE	b_TimerInputFrequency	Input frequency of the timer: PA302_LOW_FREQUENCY: 27,901 kHz PA302_HIGH_FREQUENCY: 892,857 kHz
LONG	l_DelayValue	Time interval of timer 2 in µs. See table 10a

Task:
Initialises timer 1 as an edge generator

Calling convention:
ANSI C :

```
int             i_ReturnValue;
unsigned char       b_BoardHandle;

i_ReturnValue = i_PA302_InitTimer1 (b_BoardHandle,
                                    PA302_LOW_FREQUENCY,
                                    1000);
```

Return value:
0: No error
-1: Wrong board handle parameter.
-2: Variable *b_InterruptFlag* is wrong
-3: Selected input frequency for timer 1 is wrong
-4: Selected time interval for timer 1 is wrong

2) i_PA302_StartTimer1 (...)

Syntax:
<Return value> = i_PA302_StartTimer1 (BYTE b_BoardHandle)

Parameter:
BYTE b_BoardHandle Handle of the **PA 302** board

Task:
Starts timer 1.

Calling convention:
ANSI C :

```
int           i_ReturnValue;
unsigned char b_BoardHandle;

i_ReturnValue = i_PA302_StartTimer1 (b_BoardHandle);
```

Return value:
0: No error
-1: Wrong board handle parameter.
-2: Timer 1 has not been initialised.

3) i_PA302_StopTimer1 (...)

Syntax:
<Return value> = i_PA302_StopTimer1 (BYTEb_BoardHandle)

Parameter:
BYTE b_BoardHandle Handle of the **PA 302** board

Task:
Stops timer 1.

Calling convention:
ANSI C :

```
int           i_ReturnValue;
unsigned char b_BoardHandle;

i_ReturnValue = i_PA302_StopTimer1 (b_BoardHandle);
```

Return value:
0: No error
-1: Wrong board handle parameter.
-2: Timer 1 has not been initialised.

4) i_PA302_ReadTimer1 (...)

Syntax:

```
<Return value> = i_PA302_ReadTimer1
                              (BYTEb_BoardHandle
                               PUINT       pui_ReadValue)
```

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board
PUINT	pui_ReadValue	Current timer value

Task:
Reads the current value of timer 1.

Calling convention:

ANSI C :

```
int         i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned int    ui_ReadValue;

        i_ReturnValue = i_PA302_ReadTimer1 (b_BoardHandle,
                                            &ui_ReadValue);
```

Return value:
0: No error
-1: Wrong board handle parameter.
-2: Timer 1 has not been initialised.

5) i_PA302_WriteTimer1 (...)

Syntax:

```
<Return value> = i_PA302_WriteTimer1     (BYTE      b_BoardHandle
                                          UINT      ui_WriteValue)
```

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board
UINT	ui_WriteValue	New timer value

Task:
Writes a new value in timer 1.

Calling convention:

ANSI C :

```
int         i_ReturnValue;
unsigned char   b_BoardHandle;

        i_ReturnValue = i_PA302_WriteTimer1 (b_BoardHandle,
                                             1000);
```

Return value:
0: No error
-1: Wrong board handle parameter.
-2: Timer 1 has not been initialised.

6) i_PA302_InitTimer2 (...)

Syntax:

```
<Return value> = i_PA302_InitTimer2
                    (BYTEb_BoardHandle,
                     BYTE      b_TimerInputFrequency,
                     LONG      l_DelayValue,
                     BYTE      b_InterruptFlag)
```

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board
BYTE	b_TimerInputFrequency	Input frequency of the timer: PA302_LOW_FREQUENCY: 27,901 kHz PA302_HIGH_FREQUENCY: 892,857 kHz
LONG	l_DelayValue	Time interval of timer 2 in µs. See table 10a
BYTE	b_InterruptFlag	PA302_ENABLE : An interrupt is generated after the time interval. PA302_DISABLE : No interrupt is generated after the time interval.

Task:
Initialises timer 2 as an edge generator

Calling convention:

ANSI C :

```
int          i_ReturnValue;
unsigned char    b_BoardHandle;

        i_ReturnValue = i_PA302_InitTimer2 (b_BoardHandle,
                                            PA302_LOW_FREQUENCY
                                            1000,
                                            PA302_DISABLE);
```

Return value:

0: No error
-1: Wrong board handle parameter.
-2: The user interrupt routine is not installed. See function "i_PA302_SetBoardIntRoutine"
-3: Variable *b_InterruptFlag* is wrong
-4: Selected input frequency for timer 2 is wrong
-5: Selected time interval for timer 2 is wrong

7) i_PA302_StartTimer2 (...)

Syntax:
<Return value> = i_PA302_StartTimer2 (BYTE b_BoardHandle)

Parameter:
BYTE b_BoardHandle Handle of the **PA 302** board

Task:
Starts timer 2.

Calling convention:
ANSI C :

```
int            i_ReturnValue;
unsigned char  b_BoardHandle;

               i_ReturnValue = i_PA302_StartTimer2 (b_BoardHandle);
```

Return value:
0: No error
-1: Wrong board handle parameter.
-2: Timer 2 has not been initialised.

8) i_PA302_StopTimer2 (...)

Syntax:
<Return value> = i_PA302_StopTimer2 (BYTEb_BoardHandle)

Parameter:
BYTE b_BoardHandle Handle of the **PA 302** board

Task:
Stops timer 2.

Calling convention:
ANSI C :

```
int            i_ReturnValue;
unsigned char  b_BoardHandle;

               i_ReturnValue = i_PA302_StopTimer2 (b_BoardHandle);
```

Return value:
0: No error
-1: Wrong board handle parameter.
-2: Timer 2 has not been initialised.

9) i_PA302_ReadTimer2 (...)

Syntax:
<Return value> = i_PA302_ReadTimer2
(BYTEb_BoardHandle
PUINT pui_ReadValue)

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board
PUINT	pui_ReadValue	Current timer value

Task:
Reads the current value of timer 2.

Calling convention:
ANSI C :

```
int            i_ReturnValue;
unsigned char  b_BoardHandle;
unsigned int   ui_ReadValue;

        i_ReturnValue = i_PA302_ReadTimer2 (b_BoardHandle,
                                            &ui_ReadValue);
```

Return value:
0: No error
-1: Wrong board handle parameter.
-2: Timer 2 has not been initialised.

10) i_PA302_WriteTimer2 (...)

Syntax:
<Return value> = i_PA302_WriteTimer2
(BYTEb_BoardHandle
UINT ui_WriteValue)

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board
UINT	ui_WriteValue	New timer value

Task:
Writes a new value in timer 2.

Calling convention:
ANSI C :

```
int            i_ReturnValue;
unsigned char  b_BoardHandle;

        i_ReturnValue = i_PA302_WriteTimer2 (b_BoardHandle,
                                             1000);
```

Return value:
0: No error
-1: Wrong board handle parameter.
-2: Timer 2 has not been initialised.

10.6 Digital outputs

1) i_PA302_Set1DigitalOutputOn (...)

Syntax:
<Return value> = i_PA302_Set1DigitalOutputOn
(BYTEb_BoardHandle,
BYTE b_Channel)

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board
BYTE	b_ChannelNbr (1 or 2).	Number of the digital output

Task:
Activates the output which is provided with the passed parameter *b_Channel*. Activate means setting on high.

Calling convention:
ANSI C :

```
int          i_ReturnValue;
unsigned char    b_BoardHandle;

i_ReturnValue = i_PA302_Set1DigitalOutputOn (b_BoardHandle,
                                             1);
```

Return value:
0: No error
-1: Wrong board handle parameter.
-2: Number of the digital output is wrong

2) i_PA302_Set1DigitalOutputOff (...)

Syntax:

```
<Return value> = i_PA302_Set1DigitalOutputOff
                        (BYTEb_BoardHandle,
                         BYTE       b_Channel)
```

Parameter:

BYTE	b_BoardHandle	Handle of the **PA 302** board
BYTE	b_ChannelNbr (1 or 2).	Number of the digital output

Task:

Deactivates the output which is provided with the passed parameter *b_Channel*. Deactivate means setting on low.

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA302_Set1DigitalOutputOff (b_BoardHandle,
                                              1);
```

Return value:

0: No error
-1: Wrong board handle parameter
-2: Digital output number is wrong

10.7 Functions to be used in Kernel mode

1) i_PA302_KRNL_Read1AnalogInput (...)

Syntax:

```
<Return value> = i_PA302_KRNL_Read1AnalogInput
                 (UINT    ui_Address,
                  BYTE    b_AccessMode,
                  BYTE    b_Channel,
                  PUINT   pui_AnalogInputValue)
```

Parameter:

UINT	ui_Address	Address of the PA 302 board
BYTE	b_AccessMode	access mode of the PA 302 board PA302_8BIT: 8-bit access PA302_16BIT: 16-bit access
BYTE	b_Channel	Number of the input channel to be read
PUINT	pui_AnalogInputValue :	the analog value is returned (0 to 4095)

Task:

reads the current value of the analog input *b_Channel*.

Calling convention:

ANSI C :

```
int           i_ReturnValue;
unsigned char b_BoardHandle;
unsigned int  ui_AnalogInputValue;


i_ReturnValue = i_PA302_KRNL_Read1AnalogInput
                (b_BoardHandle,
                 PA302_16BIT,
                 1,
                 &ui_AnalogInputValue);
```

Return value:

0: No error

2) i_PA302_KRNL_Set1DigitalOutputOn (...)

Syntax:

<Return value> = i_PA302_KRNL_Set1DigitalOutputOn
(UINT ui_Address,
BYTE b_Channel)

Parameters:

UINT	ui_Address	Address of the PA 302 board
BYTE	b_Channel	Number of the output channel to be set (1 or 2).

Task:

Sets the output channel which has been passed with the parameter b_Channel.
Setting an output channel means setting an output channel to „High“.

Calling convention:

ANSI C :

```
int       i_ReturnValue;

i_ReturnValue = i_PA302_KRNL_Set1DigitalOutputOn (0x390, 1);
```

Return value:

0: No error.
-2: Wrong digital output number

INDEX