ADDI-DATA®

☎ Technical support:
+49 (0)7223 / 9493-0

Attention!
Product discontinuation
due to EC RoHS directive
More info: www.addi-data.com

CE

Technical description

ADDIALOG PA 370

Board for inductive displacement transducers

6th edition 07/2001

Copyright

Guarantee and responsibility

Basically are effective our "general terms of delivery and payment". The manager receives them at the latest with the invoice. Claims for guarantee and responsibility in case of injuries and material damages are excluded, if they are due to one or some of the following causes:

- if the board has not been used for the intended purpose
- improper installation, operation and maintenance of the board
- if the board has been operated with defective safety devices or with not appropriate or non-functioning safety equipment
- nonobservance of the instructions concerning: transport, storage, inserting the board, use, limit values, maintenance, device drivers
- altering the board at the user's own initiative
- altering the source files at the user's own initiative
- not checking properly the parts which are subject to wear
- disasters caused by the intrusion of foreign bodies and by influence beyond the user's control.

Licence for ADDI-DATA software products

Read carefully this licence before using the software ADDIREG and ADDIMON. The right for using the software is given to the customer, if he/she agrees to the conditions of this licence.

- this software can only be used for configuring ADDI-DATA boards.
- copying the software is forbidden (except for archiving/ saving data and for replacing defective data media)
- deassembling, decompiling, decoding and reverse engineering of the software are forbidden.
- this licence and the software can be transferred to a third party, so far as this party has purchased a board, declares to agree to all the clauses of this licence contract and the preceding owner has not kept copies of the software.

Trademarks

Borland C and Turbo Pascal are registered trademarks of Borland International, INC.
Burr-Brown is a registered trademark of Burr-Brown Corporation
Intel is a registered trademark of Intel Corporation
AT, IBM, ISA and XT are registered trademarks of International Business Machines Corporation
Microsoft, MS-DOS, Visual Basic and Windows are registered trademarks of Microsoft Corporation
The original version of this manual is in German. You can obtain it on request.

WARNING

In case of improper handling and if the board is not used for the purpose it is intended for:

people may be injured

the board, PC and peripheral may be damaged

the environment may be polluted

★★★ Protect yourself, other people and the environment★★★

- **Do read the safety leaflet!**
  If this leaflet is not with the manual, please contact us.

- **Observe the instructions in the manual!**
  Make sure that you have not forgotten any step. We are not liable for damage resulting from a wrong use of the board.

- **Symbols used**

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

i **IMPORTANT!**
designates hints and other useful information.

- **Do you have any question?**
  Our technical support is always glad to help you.

Declaration of Conformity

This declaration is valid for the following product:

ADDIALOG PA 370
Board for inductive displacement transducers

It is made by

ADDI-DATA GmbH
Meß- und Steuerungstechnik
Dieselstraße 3
D-77833 Ottersweier

in sole responsibility and is valid on the understanding that the product is competently installed, used and maintained, according to the respective security regulations as well as to the manufacturer's instructions regarding its intended use.

This declaration states that the product complies with following EC Directives:

- **EWGRL 336/89 of 3.05.1989**
- **EWGRL 31/92 of 28.04.1992**
- **EWGRL 68/93 of 22.07.1993**

This declaration is valid for all units manufactured according to the manufacturing references listed in the form TD370.020.

Following norms have been applied to test the product regarding electromagnetic compatibility:

- **EN55011/03.91**
- **EN55022/08.94**
- **EN50082-2/03.95**

We point out that

- the conformity and herewith the permission of use expire if the user alters the product without consulting with the manufacturer.
- non-skilled users are to have the operational area of the product and the requirements resulting from it checked prior to putting into operation.
- by using this product in appliances coming under the EC EMC Directive, the user is to make sure they are conform to its regulations prior to putting into operation.
- by using this product in machines / installations coming under the EU Machine Directive, the user is to make sure they are conform to its regulations prior to putting into operation.

A copy of the EMC tests is at your disposal on request.

15 October 1995

Antonio Agnetti
Legally valid signature of the manufacturer

Figures

Tables

1 INTENDED PURPOSE OF THE BOARD

The **PA 370** board is an interface between an industrial process and a personal computer (PC). It is to be used in a free PC ISA slot. The PC is to comply with the EU directive 89/336/EEC and the specifications for EMC protection.

Products complying with these specifications bear the CE mark.

Analog signals are exchanged with the periphery through the 50-pin SUB-D male connector of the **PA 370** board. The board has an oscillator circuit followed by an amplifier and 16 analog inputs which are only intended for connecting at the most 16 inductive displacement transducers (type GT21 of Tesa) .

The use of the board in a PC could change the PC features regarding noise emission and immunity. Increased noise emission or decreased noise immunity could result in the system not being conform anymore. Check the PC and cable shielding capacity prior to putting the device into operation.

The connection with our standard cable ST370 complies with the specifications:
- metallized 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.

WARNING!
Before measuring, let the board run hot during 15 minutes, so that the components can reach the operating temperature. Temperature drift errors are then reduced.

2 USER

2.1 Qualification

Only persons trained in electronics are entitled to perform the following works:

- installation,
- use,
- maintenance.

2.2 Personal protection

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

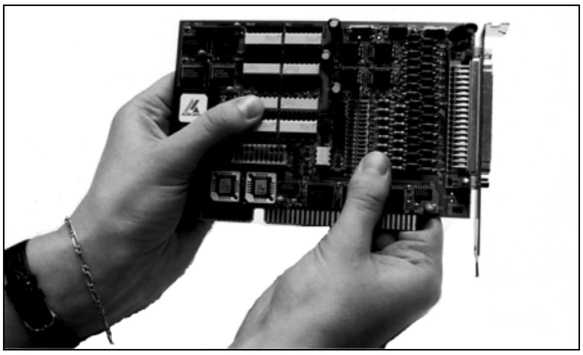
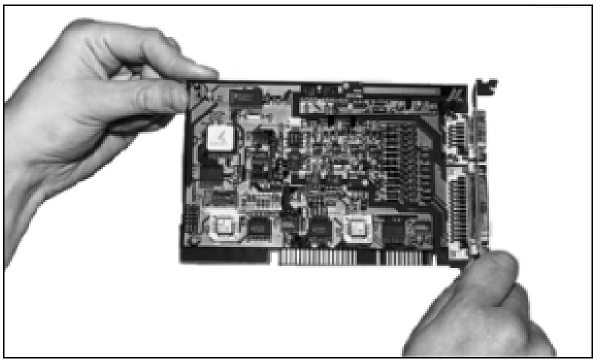


Fig. 3-2: Correct handling



4 TECHNICAL DATA

4.1 Electromagnetic compatibility (EMC)

The board has been subjected in an accepted laboratory to the EMC tests.
The board complies as follows with the limit values set by the norms EN50082-2, EN55011, EN55022:

	True value	Set value
ESD	4 kV	4 kV
Fields	10 V/m	10 V/m
Burst	2 kV	2 kV
Conducted radio interference	10 V	10 V

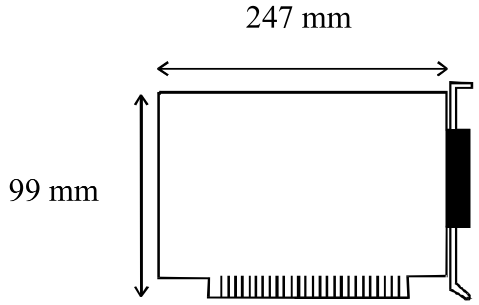
⚠ **WARNING!**
The EMC tests have been carried out in a specific appliance configuration.
We guarantee these limit values **only** in this configuration. [1]

Consider the following aspects:
- your test program must be able to recognise errors of operation.
- **Set your system up**, so that you can know what caused the errors.

4.2 Physical set-up of the board

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

Approximate board dimensions



Weight:	230 g
Installation in:	AT slot
Connection to the peripheral	50-pin SUB-D male connector

[1] We transmit our appliance configuration on request.

4.3 Versions

The **PA 370** board is available in the following versions:

PA 370-8	8 transducer inputs
PA 370-16	16 transducer inputs

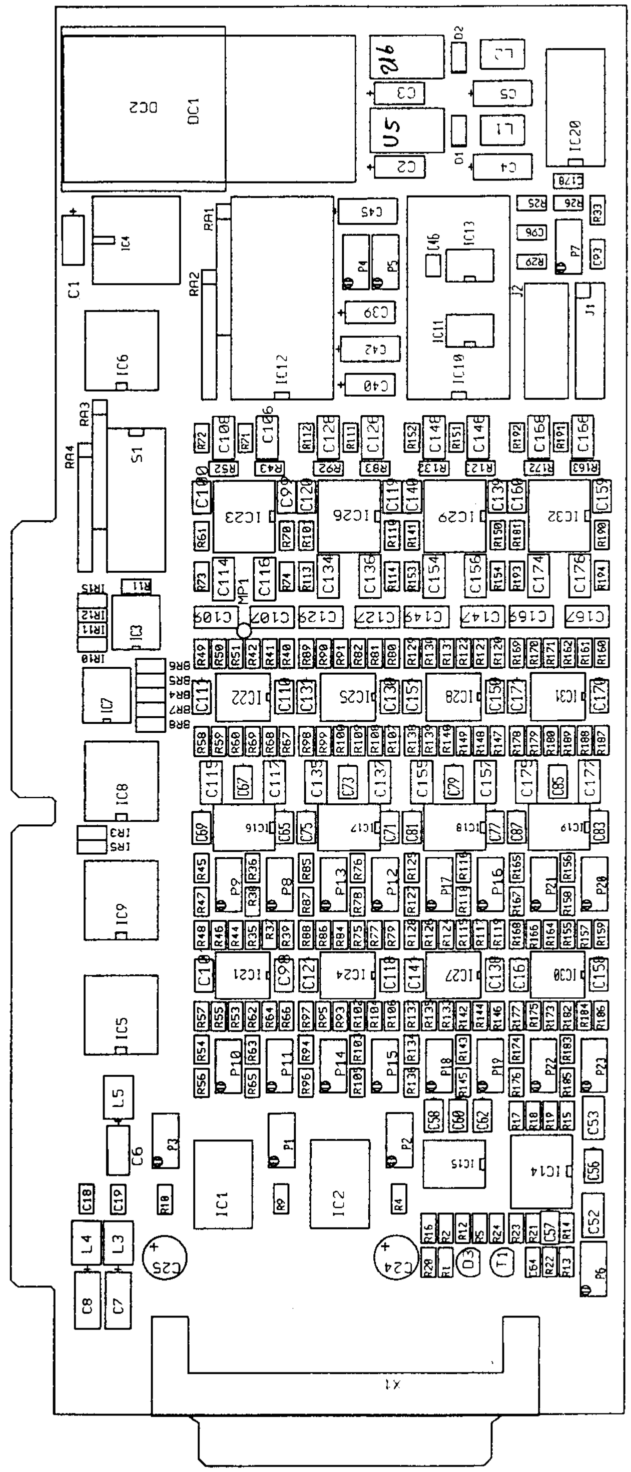
4.4 Limit values

Operating temperature:	0 to 60°C	
Storage temperature:	-25 to 70°C	
Relative humidity:	30% to 95% non condensing	
ISA-bus interface		
Bus speed:	8 MHz	
Supply signal for the transducers (up to 16):	2 x 1.5 VAC 180° phase shifted	
Amplifier sensitivity for the max. measuring range GT21:	73.75 mV/V/mm	
Number of transducer inputs:	16 (8)	
Number of voltage inputs:	8	
Max. input voltage:	± 10 V	
Energy supply		
Current consumption:	5 V ± 5%:	1.6 A
	- 12 V ± 5%:	150 mA
	+ 12 V ± 5%:	150 mA

5 SETTINGS

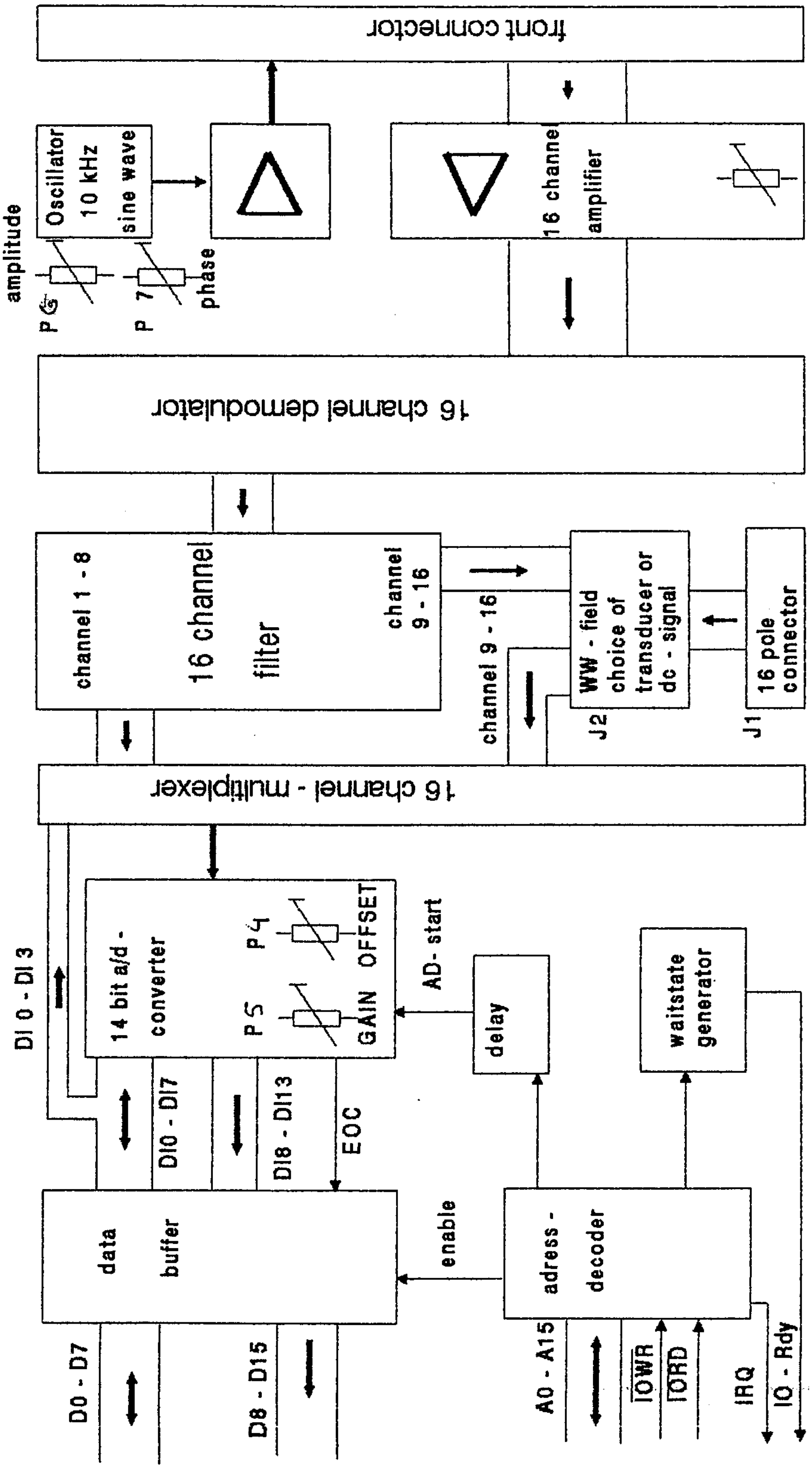
5.1 Component scheme

Fig. 5-1: Component scheme of the PA 370



5.2 Block diagram

Fig. 5-2: Block diagram of the PA 370 board



5.3 Setting the base address

The interface to the PC bus consists of two I/O addresses within the 64 KB I/O address space. The **PA 370** board uses the standard address, data, and control signals of the PC/AT bus. The board is so designed that it can fit into a free slot of the PC/AT.

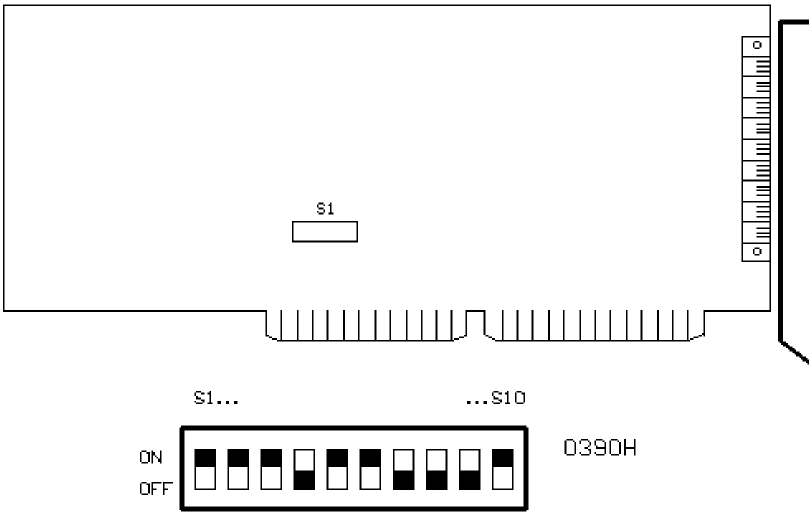
The board is delivered with the base address set to 0390H.

If this base address is already used by another board or by the computer, you have to select another base address. If the base address suits, the board is immediately operational.

The board occupies two I/O addresses within the 64 KB I/O address space. The base address can be adjusted within the 64 KB I/O address space in steps of two bytes with 10-pin block of DIP-switches.

To the switches S1-S10 correspond the address bits A1-A10.

Fig. 5-3: Dip switches



0 = logic '0' = switch in "ON" position
1 = logic '1' = switch in 'OFF' position

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 ADDIREG (once is enough)
5. Configure the board
6. Install the driver and the samples if necessary
7. Switch off the PC
8. Install the **second** board
9. Start the PC
10. Configure the board
11. Install the driver and the samples if necessary.

etc.

IMPORTANT!
To install the new version of ADDIREG, **please uninstall first the current version from your PC** with the ADDI_UNINSTALL program.

Proceed as if you wished to install one single board.

6.1 Inserting the board

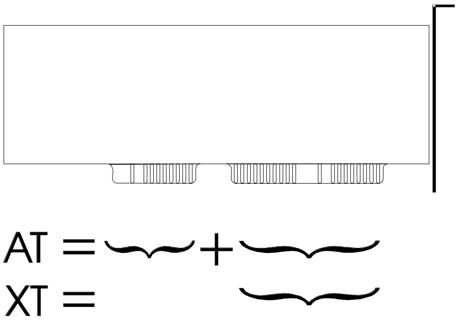
IMPORTANT!
Do observe the *safety instructions.*

6.1.1 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**

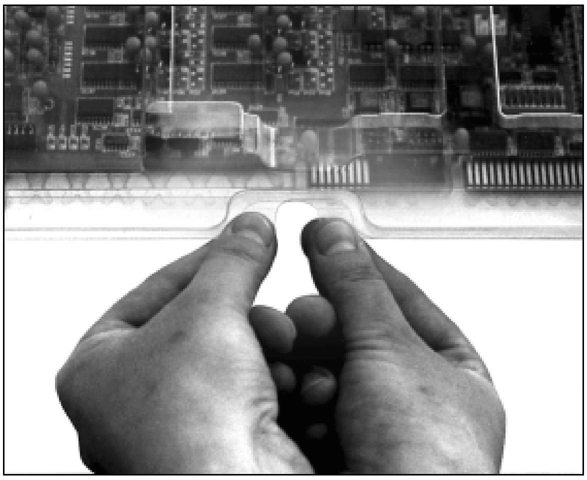
Fig. 6-1: Slot types



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

2. Remove the metal bracket of the selected slot **according to the instructions of the PC manufacturer.** Keep the bracket. You will need it if you remove the board.
3. **Discharge yourself from electrostatic charges**
4. **Take the board from its protective packing.**

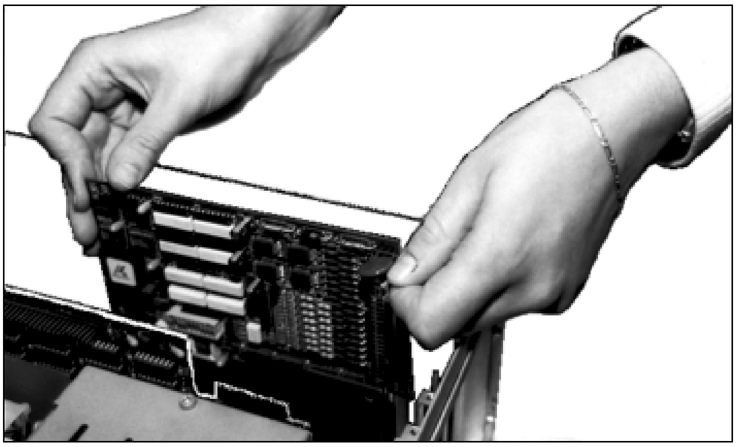
Fig. 6-2: Opening the protective blister pack



6.1.2 Plugging the board into the slot

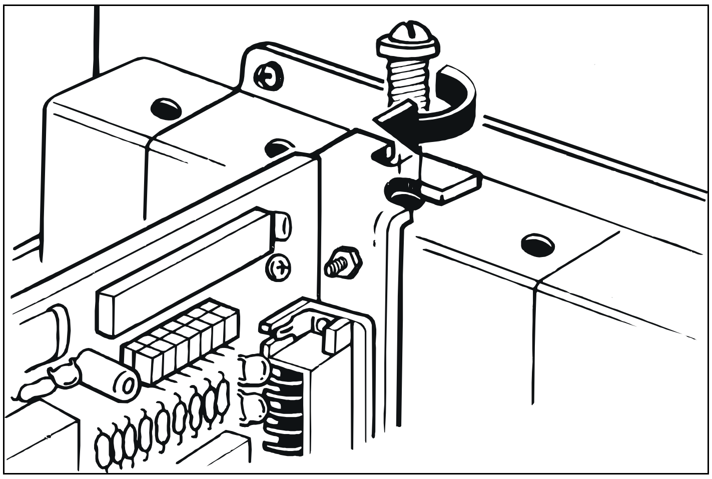
- **Discharge yourself from electrostatic charges**
- Insert the board **vertically into the chosen slot.**

Fig. 6-3: Inserting the board



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

Fig. 6-4: Securing the board at the back cover



- **Tighten all loosen screws.**

6.1.3 Closing the PC

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

6.2 Installing the software

The board is delivered with a CD-ROM containing ADDIREG for Windows NT 4.0 and Windows 95/98.

You can download the latest version of the ADDIREG program from the Internet:
http://www.addi-data.de
http://www.addi-data.com

The CD also contains standard software for the ADDI-DATA boards:

- 16-bit for MS-DOS and Windows 3.11
- 32-bit for Windows NT/95/98.

6.2.1 Software installation under MS-DOS and Windows 3.11

- Copy the contents of PA370\Dos or PA370\Win311 on a disk.
  If several disks are to be used, the directory contents is stored in several sub-directories (Disk1, Disk2, Disk3...).
- Insert the (first) disk into a drive and change to this drive.
- Enter <INSTALL>.

The installation program gives you further instructions.

6.2.2 Software installation under Windows NT / 95 / 98

- Select the directory PA370\WinNT-9x\Disk1.
- Start the set-up program "setup.exe" (double click)
- Select one of the 3 parameters
  1- typical
  2- compact
  3- custom

Proceed as indicated on the screen and read the "Software License" and "Readme". Under "custom", you can select your operating system.

The installation program gives you further instructions.

6.3 Board configuration with ADDIREG

The ADDIREG registration program is a 32-bit program for Windows NT and Windows 95/98.
The user can register all hardware information necessary to operate the ADDI-DATA PC boards.

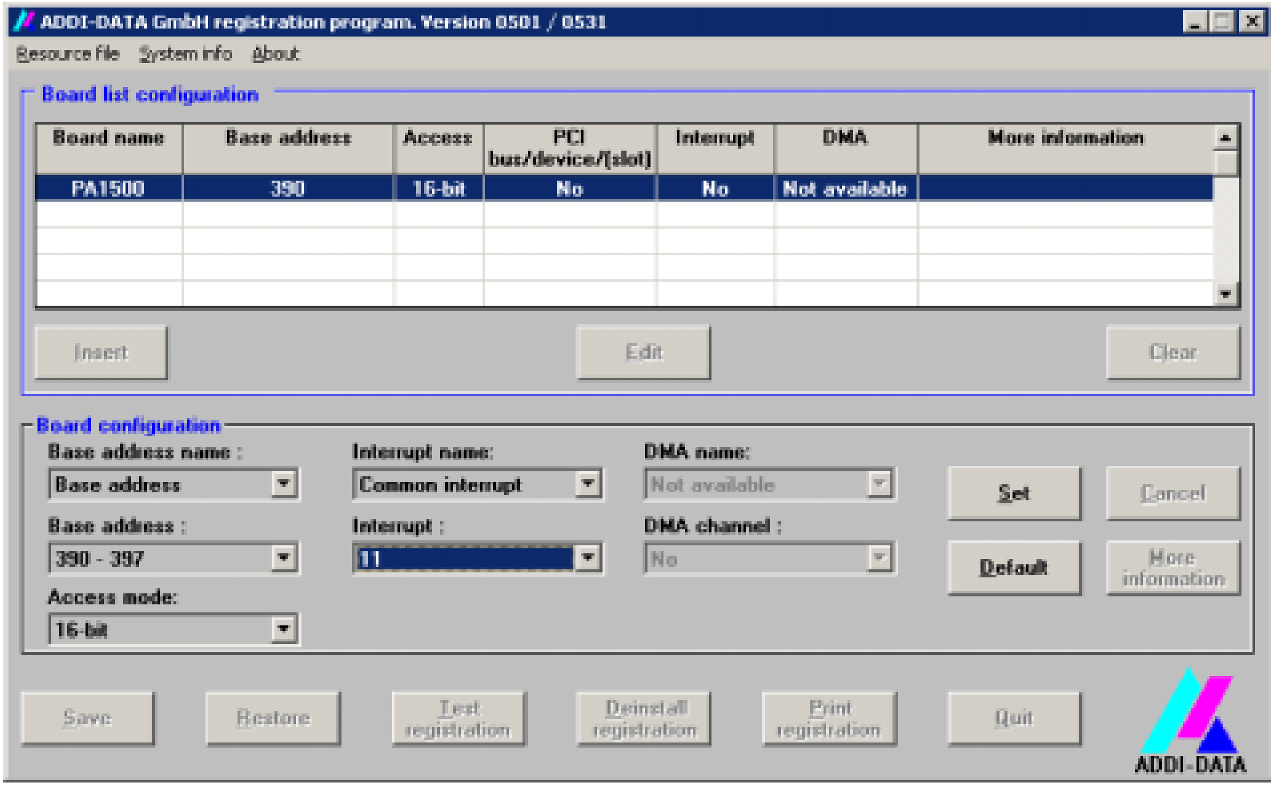
i **IMPORTANT!**
If you use one or several resources of the board, you cannot start the ADDIREG program.

6.3.1 Program description

i **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. 6-5: The ADDIREG registration program



Board name	Base address	Access	PCI bus/device/[slot]	Interrupt	DMA	More information
PA1500	390	16-bit	No	No	Not available	

Base address name : Base address
Base address : 390 - 397
Access mode: 16-bit
Interrupt name: Common interrupt
Interrupt : 11
DMA name: Not available
DMA channel : No

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

Board name:
Names of the different registered boards
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 jumpers.

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

More information:
Additional information like the identifier string (e.g.: 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.

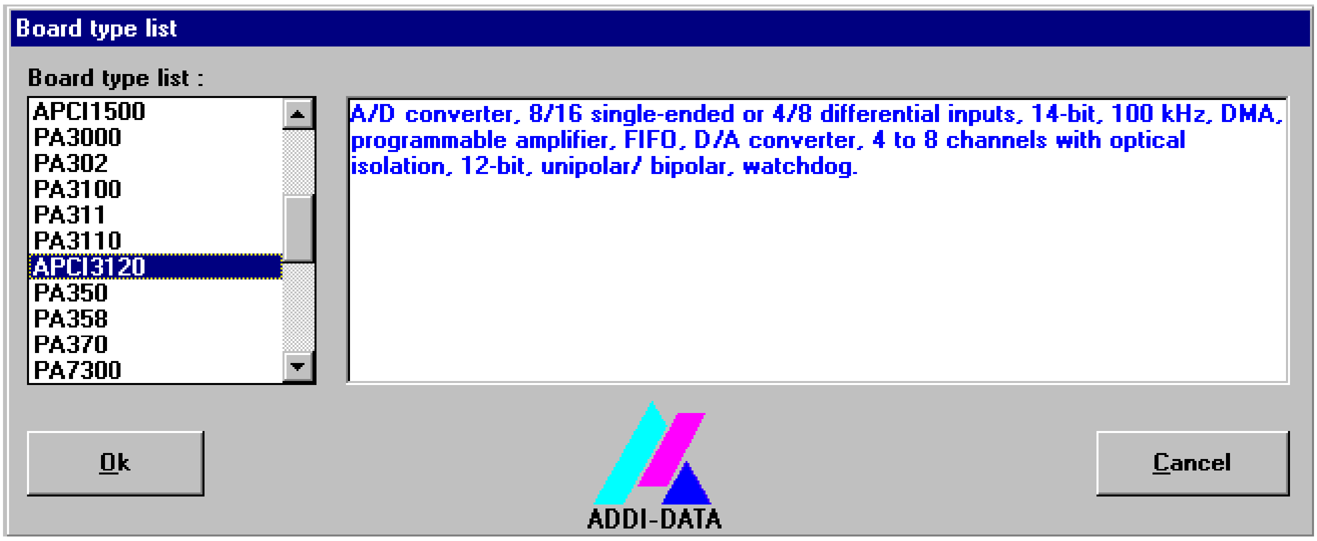
DMA channel:
Selection of the used DMA channel.

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. 6-6: Configuring a new board



Board type list

Board type list :

APCI1500
PA3000
PA302
PA3100
PA311
PA3110
APCI3120
PA350
PA358
PA370
PA7300

A/D converter, 8/16 single-ended or 4/8 differential inputs, 14-bit, 100 kHz, DMA, programmable amplifier, FIFO, D/A converter, 4 to 8 channels with optical isolation, 12-bit, unipolar/ bipolar, watchdog.

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

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

Set:
Sets the parameterised 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.

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

More information:
You can change the board specific parameters like the identifier string, the COM number, the operating mode of a communication board, etc...
If your board does not support this 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 registration of all boards listed in the table.

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

Quit:
Quits the ADDIREG program.

6.3.2 Registering a new board

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

- Call up the ADDIREG program. Figure 6-5 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.

6.3.3 Changing the registration of a board

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

- Call up the ADDIREG program. Select the board to be changed.
  The board parameters (Base address, DMA channel, ..) are listed in the lower fields.
- Click on the parameter(s) you want to set and open the scroll function(s).
- Select a new value. Activate it with a click.
  Repeat the operation for each parameter to be modified.
- Once the wished configuration is set, click on "Set".
- Save the configuration with "Save".
- You can test if the registration is "OK".
  This test controls if the registration is right and if the board is present.
  If the test has been successfully completed you can quit the ADDIREG program.
  The board is initialised with the set parameters and can now be operated.

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

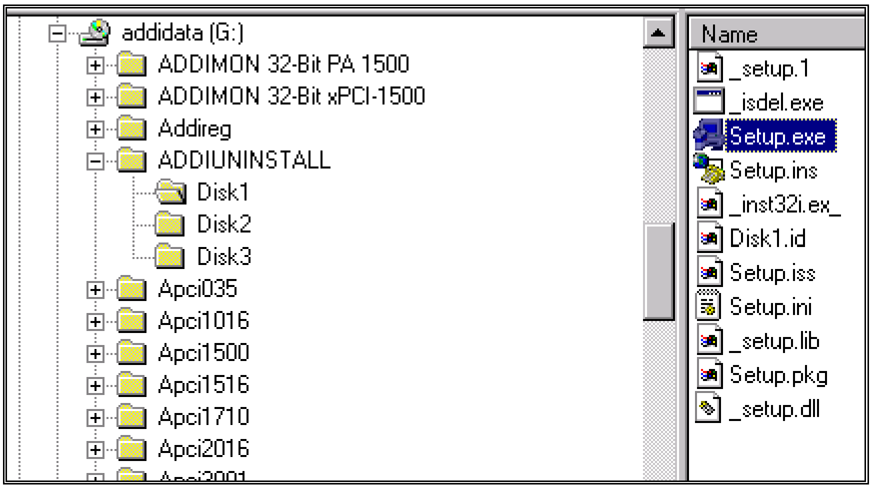
6.4 The ADDI-UNINSTALL program

6.4.1 Installation of ADDI-UNINSTALL

The ADDI_UNINSTALL program is delivered on the CD-ROM.

- Change to the CD drive and start the set-up file (double click).

Fig. 6-7: Installation of the ADDI-UNINSTALL program



addidata (G:)
ADDIMON 32-Bit PA 1500
ADDIMON 32-Bit xPCI-1500
Addireg
ADDIUNINSTALL
Disk1
Disk2
Disk3
Apci035
Apci1016
Apci1500
Apci1516
Apci1710
Apci2016
Apci3001

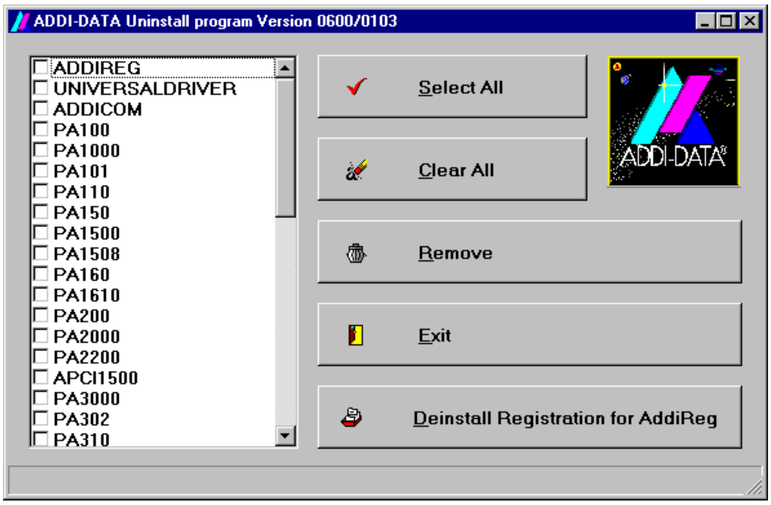
Name
_setup.1
_isdel.exe
Setup.exe
Setup.ins
inst32i.ex
Disk1.id
Setup.iss
Setup.ini
_setup.lib
Setup.pkg
_setup.dll

- Proceed as indicated on the screen.

6.4.2 Software uninstalling with ADDI-UNINSTALL

- Start the ADDI_UNINSTALL program.

Fig. 6-8: The ADDI_UNINSTALL program



ADDI-DATA Uninstall program Version 0600/0103

☐ ADDIREG
☐ UNIVERSALDRIVER
☐ ADDICOM
☐ PA100
☐ PA1000
☐ PA101
☐ PA110
☐ PA150
☐ PA1500
☐ PA1508
☐ PA160
☐ PA1610
☐ PA200
☐ PA2000
☐ PA2200
☐ APCI1500
☐ PA3000
☐ PA302
☐ PA310

- Select the software or the driver to be deinstalled. Enter it in the corresponding check box.

- Click on "Remove". Proceed as indicated until the complete removal of the program.

Uninstall ADDIREG

- Click on "Deinstall registration for AddiReg".
- Proceed as indicated until the complete removal of ADDIREG.

You can also download the ADDI-UNINSTALL program from the Internet.

6.5 Questions and software downloads from the Internet

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

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

You can download the latest version of the device driver for the **PA 370** board**.**

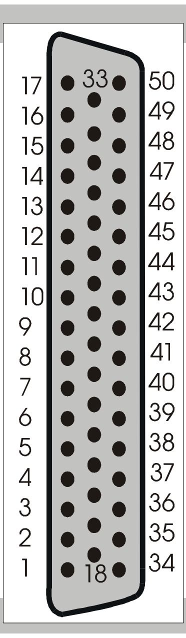
http://www.addi-data.de. or
http://www.addi-data.com

7 CONNECTION TO THE PERIPHERAL

7.1 Connector pin assignment

Fig. 7-1: 50-pin SUB-D male connector

Pin		Pin			Pin
17	Ground	33	Channel 15	Channel 16	50
16	Channel 14	32	Ground	Ground	49
15	Ground	31	Channel 12	Channel 13	48
14	Channel 11	30	Ground	Ground	47
13	Ground	29	Channel 9	Channel 10	46
12	Channel 8	28	Ground	Ground	45
11	Ground	27	Channel 6	Channel 7	44
10	Channel 5	26	Ground	Ground	43
9	Ground	25	Channel 3	Channel 4	42
8	Channel 2	24	Ground	Ground	41
7	Ground	23	Ground	Channel 1	40
6	Tminus	22	Tminus	Ground	39
5	Tminus	21	Tminus	Tminus	38
4	Tplus_r	20	Tplus	Tminus	37
3	Tplus	19	Tplus	Tplus	36
2	Tplus	18	OSZ_ back	Tplus	35
1	OSZ_ back			OSZ_ back	34



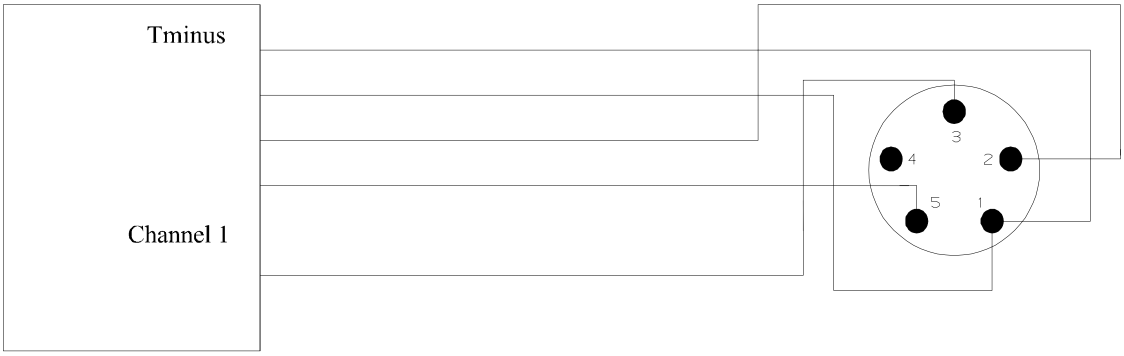
Signal settings	Meaning
OSZ_back	reference ground of the signal for LVDT-transducers
Tplus and Tminus	constitute the phase-shifted supply signal of the inductive transducer
Tminus	negative supply voltage
Tplus	positive supply voltage
Tplus_r	feedback of the supply voltage to regulate the amplitude. It serves as the true value signal of the oscillator for the supply voltage and must be connected directly to the connector of the transducer cable
Ground	shielding ground
Channel 1-16	connection of the measuring signals of the inductive transducers

7.2 Connection of the inductive displacement transducers

For the connection of the inductive displacement transducers to the board are available:
two connection boxes with 8 (PX371-8) or 16 (PX371-16) transducer female connectors and the belonging connecting cable. The connection with the board is made via a 50-pin SUB-D male connector (see pin assignment in chapter 7).

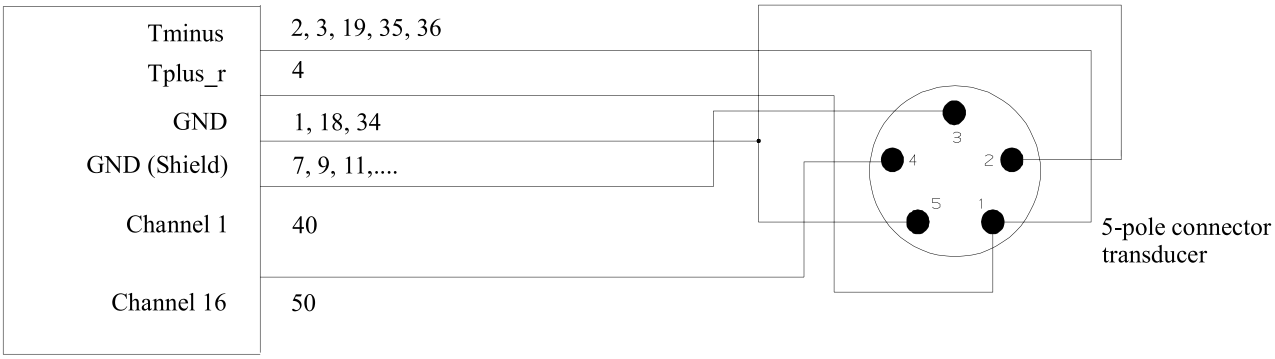
7.2.1 Connection of the half bridge transducers (Tesa)

Fig. 7-2: Connection of the half bridge transducers



7.2.2 Connection of the LVDT transducers (Tesa)

Fig. 7-3: Connection of the LVDT transducers



IMPORTANT!
If you use transducers of other manufacturers, please observe the pin assignments!

8 FUNCTIONS OF THE BOARD

8.1 Data acquisition

The **PA 370** board is basically designed for inductive displacement transducers of the type GT21 supplied by TESA or type HW 901 supplied by HOMMELWERKE. The technical data of the manufacturer TESA are used as reference data.

The supply signal of the transducer has been set to a sinusoidal frequency of 10 kHz and 3V of effective voltage. This supply signal as well as the 16 measuring signal lines of the transducers are distributed into groups of 8 on the 50 -pin SUB-D male connector. The AC measuring signal is decoded on the board into a DC voltage signal of +/-10V. This corresponds to a travel of +/-2mm for the GT21 displacement transducer.

8.1.1 Calibrating the A/D converter

The A/D converter is operated in the bipolar mode. Conversion of the analog values into digital code is carried out according to the two complement. This means that the MSB is set to logic "1" for a negative voltage. For voltages equal or superior to 0V the MSB is set to logic "0".

Table 8-1: Voltage values and their corresponding digital format

Voltage	Binary code					Hex code
	MSB				LSB	
-10.0000 V	1	0000	0000	0000	0	2000H
-0.00122 V	1	1111	1111	1111	1	3FFFH
0 V	0	0000	0000	0000	0	0000H
5.0000 V	0	1000	0000	0000	0	1000H
10.0000 V	0	1111	1111	1111	1	1FFFH

8.1.2 Adjustment of the supply signal for inductive displacement transducers

The settings have already been made at delivery and **cannot be modified afterwards**.

8.1.3 Synchronisation of oscillator and demodulator

A phase-shift circuit is used to calibrate the phase displacement between the supply signal and the measuring signals of the inductive displacement transducers.

These settings have already been made at delivery and **cannot be modified afterwards**.

8.1.4 Adjustment of the maximum voltage value of the demodulated measuring signal

The amplifier must be calibrated for setting the maximum modulation range to ±10V for each measuring channel. This adjustment has been effected for the transducer type GT21.

8.1.5 Measuring DC voltage signals

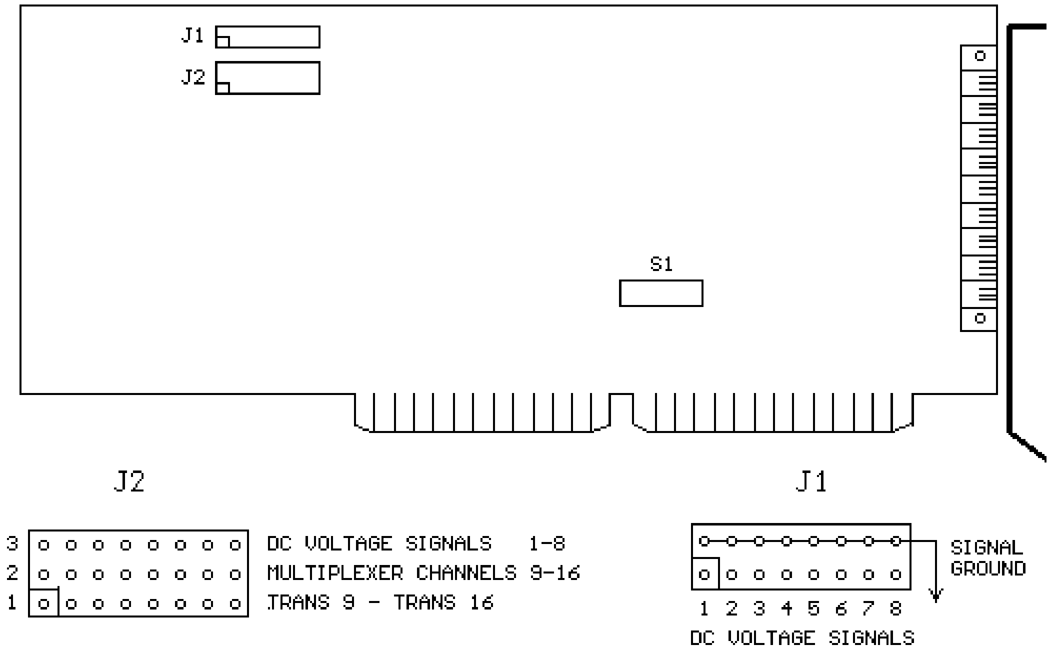
The board can acquire DC voltage signals (+/-10V) instead of transducer signals. The 16-channel multiplexer is therefore divided into two groups of channels:

- to channels 1-8 are connected the transducer signals Trans 1 to Trans 8;
- to channels 9-16 can be connected, individually for each channel, DC voltage or transducer signals (Trans 9 to Trans 16).

DC voltage respectively transducer signals (Trans 9 to Trans 16) are selected over the jumper field J2 (see fig. 8-1).

DC voltage signals (+/-10V) are led to the board via the 16 pole pin plug J1.

Fig. 8-1: Selecting DC voltage or transducer signals via J2 and position of connector J1



Jumper set between 1 and 2: Transducer signal is led to the multiplexer.
Jumper set between 2 and 3: Direct voltage signal connected to J1 is led to the multiplexer.

8.2 Interrupt

In addition to testing the end of conversion by software, the user also has a hardware interrupt line at his disposal. This interrupt line can be connected to one of the 6 interrupt request lines of the PC-AT I/O bus over jumper.

The following interrupt bus lines are available:

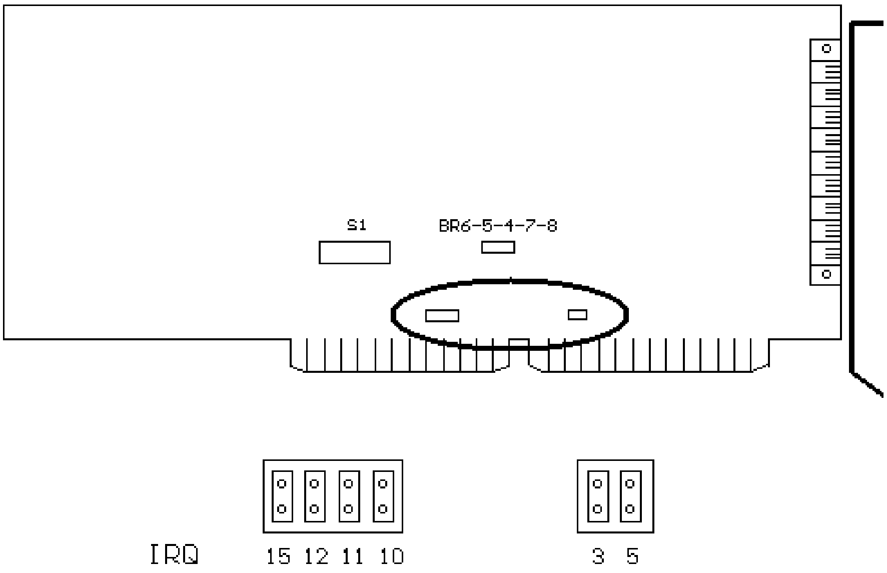
IRQ 3	XT-AT
IRQ 5	XT-AT
IRQ 10	AT
IRQ 11	AT
IRQ 12	AT
IRQ 15	AT

The **PA 370** board provides the end of conversion as a source for interrupt.

The interrupt request flip-flop is reset automatically each time when conversion is started.

The position of the jumpers is shown in the component scheme.
They are designated by BR.

Fig. 8-2: Selection of the interrupt request line



The interrupt request line is selected by **one single** jumper.

WARNING!
Prior to selecting the interrupt bus line, make sure that this line is not being used for any other components in the computer. Multiple usage is not allowed!

8.3 Measuring principle and operation

8.3.1 Introduction

The main task of the **PA 370** board is the evaluation of measuring signals of inductive displacement transducers. The board provides all the necessary signals for supplying the inductive displacement transducers.

8.3.2 The measuring principle for inductive transducers

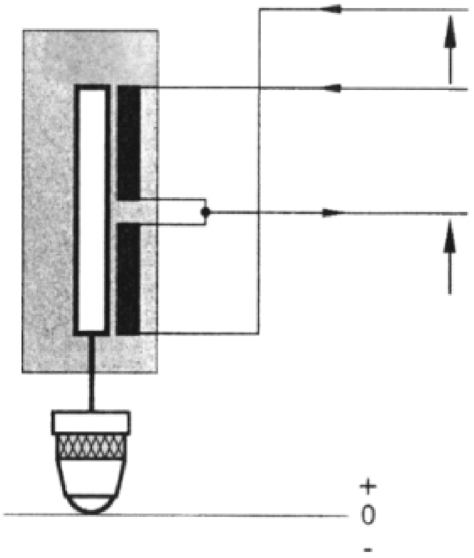
Basically, an inductive displacement transducer is used for evaluating travels. The non-electric unit "travel" is converted into an electric voltage.

This electric voltage may be generated by two different measuring principles, and therefore different evaluation electronics are required.

Half bridge transducer

The structure of this transducer consists of two inductance coils (windings). These coils are fed directly by means of a sinusoidal voltage of 10 kHz. The measuring bolt moves along the coils with its ferromagnetic core that changes the voltages in the two coils depending on its position. The measuring bolt functions like a variable voltage distributor, and the change in voltage at the coils results in the sinusoidal measuring signal to be evaluated.

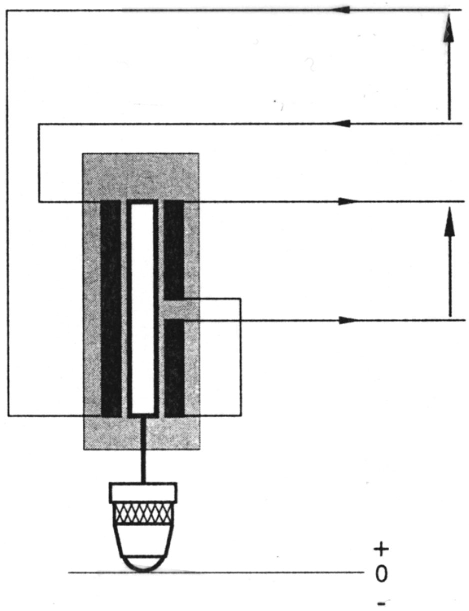
Fig. 8-3: Half bridge transducer



LVDT- transducer (Linear Variable Differential Transformer)

The LVDT transducers have three coils: one primary coil and two secondary coils. These coils are positioned concentrically around the mobile core and form two symmetrical transformers with respect to the electrical zero point of the transducer. The primary coil is fed by a sinusoidal voltage of 5 kHz whereas both secondary coils (switched in phase opposition) produce an electrical signal proportional to the measured displacement.

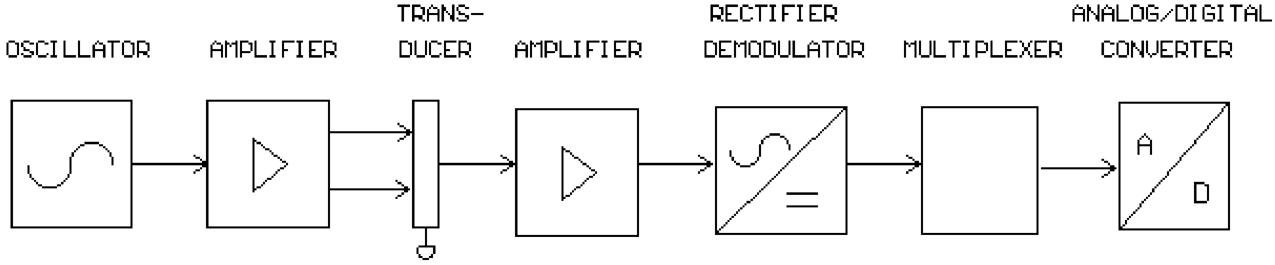
Fig. 8-4: LVDT transducer



8.4 Board operation

The **PA 370** board generates the sinusoidal supply voltage and evaluates the measuring signal. It is suitable for the two types of transducers. The electronics basically comprise six functional components.

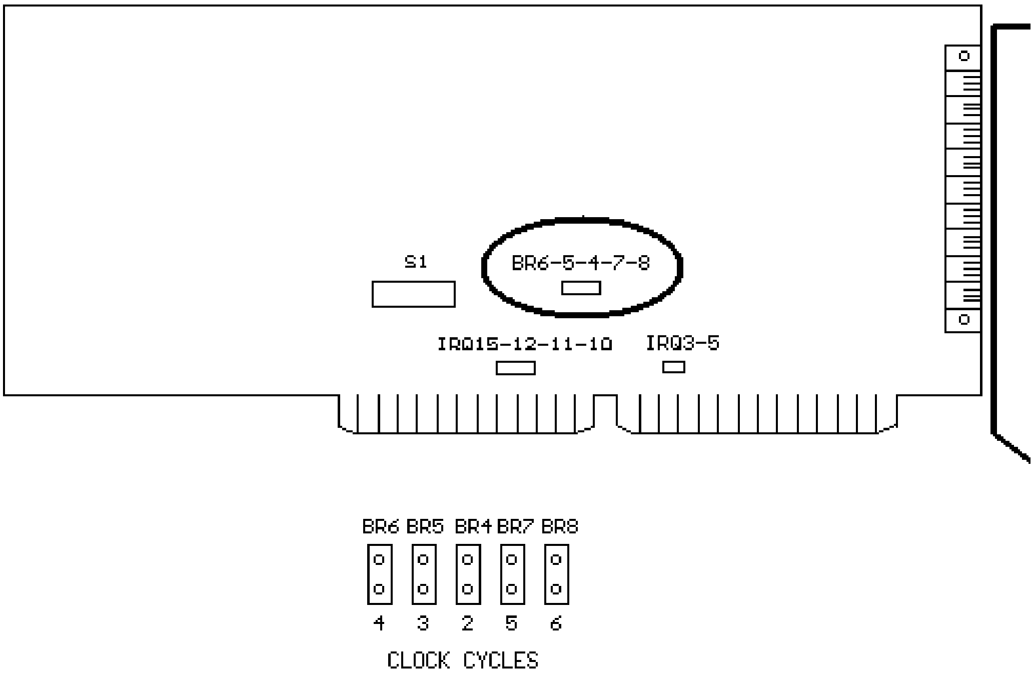
Fig. 8-5: Board operation



8.5 Waitstate generator

A Waitstate generator is used to operate PA 370 board in computers that have a high-speed I/O bus. As a result, the writing and reading cycles can be extended by 2 to 6 clock cycles. The jumpers BR4-BR8 are intended for adjusting the wait state generator. Their positions are shown in the component scheme in chapter 5.

Fig. 8-6: Waitstate generator



9 PROGRAMMING

9.1 I/O map of the board

Table 9-1: I/O word

Address	I/O function	D15	D14	D13	D12	D11	D10	D9	D8
Yyyy+00H	IOWR	X	X	X	X	X	X	X	X
Yyyy+00H	IORD	EOC	X	MSB	B12	B11	B10	B9	B8

D7	D6	D5	D4	D3	D2	D1	D0	Designation
X	X	X	X	M3	M2	M1	M0	AD-START
B7	B6	B5	B4	B3	B2	B1	LSB	AD-DATA

AD-START:
Only the bits D0-D3 are relevant. The channel to be acquired is selected on bits D0-D3. Conversion is started, after the channel has been selected by writing on M0-M3.

Table 9-2: Binary output of the channel address

Data bits					
D15-D4	**M3**	**M2**	**M1**	**M0**	**Channel**
X	0	0	0	0	1
X	0	0	0	1	2
X	0	0	1	0	3
X	0	0	1	1	4
X	0	1	0	0	5
X	0	1	0	1	6
X	0	1	1	0	7
X	0	1	1	1	8
X	1	0	0	0	9
X	1	0	0	1	10
X	1	0	1	0	11
X	1	0	1	1	12
X	1	1	0	0	13
X	1	1	0	1	14
X	1	1	1	0	15
X	1	1	1	1	16

AD-DATA:

The converted A/D value is retained in the data bits D0-D13 of the AD-DATA word. In data bit D15 is the end of conversion bit EOC.

LSB -> MSB (B0-B13): digitalized analog value

EOC bit: Indicates the end of the conversion
EOC bit = "0" conversion has stopped
EOC bit = "1" conversion being carried out

9.2 Programming

1. Select the desired channel number in the AD-START word.
2. Read the AD-DATA word and evaluate the EOC bit.
3. If EOC = 0 read the AD-DATA word and mask the 14-bit value

Programming example in C

```
main ()

  {

   int EOC, value;
    outport(0x390,0);                    /* Output channel number + Start */
                                         /* conversion */
    do
      {
      EOC = inport (0x390) & 0x8000;     /*  Read the AD-DATA word */
      }
   while (EOC ! = 0);                    /* Evaluate the EOC bit EOC */
   Wert = inport (0x390) & 0x3FFF;       /* Read value */
   printf ("value = %d", value);         /* Display the14-bit value */
  }
```

10 STANDARD SOFTWARE

10.1 Introduction

IMPORTANT!
Note the following conventions in the text:

Function:	"i_PA370_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

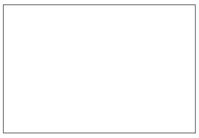
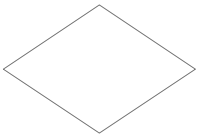
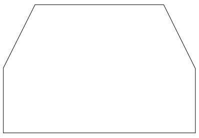
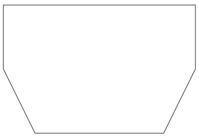
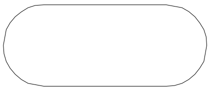
Table 10-3: Define value

Define name	Decimal value	Hexadecimal value
DLL_COMPILER_C	0	0
DLL_COMPILER_PASCAL	1	1
DLL_COMPILER_VB	2	2
DLL_LABVIEW	3	3
DLL_COMPILER_VB5	4	4
PA370_DISABLE	0	0
PA370_ENABLE	1	1
PA370_CHANNEL_1	1	1
PA370_CHANNEL_2	2	2
PA370_CHANNEL_3	3	3
PA370_CHANNEL_4	4	4
PA370_CHANNEL_5	5	5
PA370_CHANNEL_6	6	6
PA370_CHANNEL_7	7	7
PA370_CHANNEL_8	8	8
PA370_CHANNEL_9	9	9
PA370_CHANNEL_10	10	A
PA370_CHANNEL_11	11	B
PA370_CHANNEL_12	12	C
PA370_CHANNEL_13	13	D
PA370_CHANNEL_14	14	E
PA370_CHANNEL_15	15	F
PA370_CHANNEL_16	16	10
PA370_ASYNCHRONOUS_MODE	0	0
PA370_SYNCHRONOUS_MODE	1	1

10.2 Norm DIN 66001 for program operation

All the API software functions necessary to the operation of the board **PA 370** are listed in the following chapter.

Functions diagrams have been designed with the following symbols. The user is hence able to follow the different steps of the software functions.

	**Process,** general (including inputs and outputs)
	**Decision** **Selection unit**
	**Loop limit** Beginning
	**Loop limit** End
	**Terminator** (e.g. Beginning or end of a sequence, origin or place of data)

10.3 Initialisation

1) i_PA370_InitCompiler (...)

Syntax:
<Return value> =i_PA370_InitCompiler
(BYTE b_CompilerDefine)

Parameters:
- Input:
BYTE b_CompilerDefine

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

- Output:
No output signal has occurred.

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

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

Calling convention:

ANSI C :

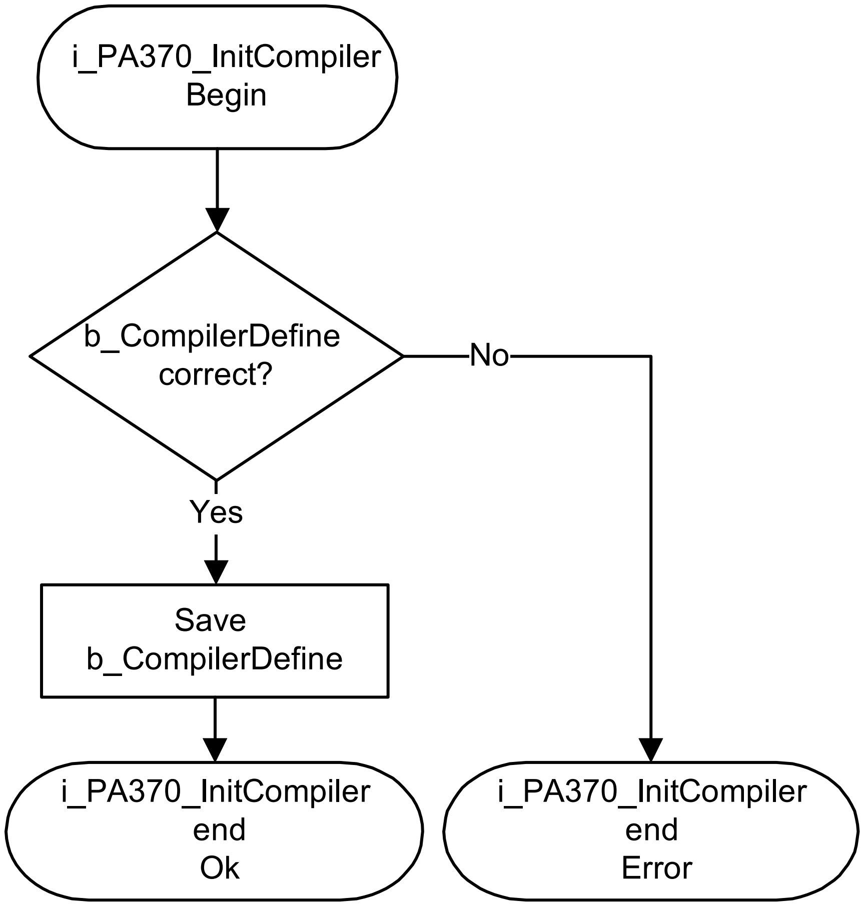
```
int             i_ReturnValue;

i_ReturnValue = i_PA370_InitCompiler  (DLL_COMPILER_C);
```

Return value:
0: No error
-1: The parameter b_CompilerDefine is wrong

Input

b_CompilerDefine

Function diagram


i_PA370_InitCompiler Begin
b_CompilerDefine correct?
No
Yes
Save b_CompilerDefine
i_PA370_InitCompiler end Ok
i_PA370_InitCompiler end Error

Output

<Return Value>

i

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

2) i_PA370_SetBoardInformation16Bit (..)

Syntax:

```
<Return Wert> = i_PA370_SetBoardInformation16BIT
                                          (UINT    ui_BaseAddress,
                                           BYTE     b_NbrOfInput,
                                           BYTE     b_InterruptNbr,
                                           PBYTE   pb_BoardHandle)
```

Parameters:

-Input:

UINT	ui_BaseAddress	Base address of the **PA370** board
BYTE	b_NbrOfInput	Number of analog inputs
BYTE	b_InterruptNbr	Interrupt number of the **PA 370** (3, 5, 10, 11, 12 or 15). If 0, no interrupt is used

- Output:

PBYTE	pb_BoardHandle	Handle[1] of the **PA 370** board to use the functions

Task:

Verifies if the board **PA370** is present. Stores the following information:
- the base address,
- the number of analog inputs,
- the interrupt number.

A handle is returned to the user which allows to use the next functions.
Handles allow to operate several boards.

Calling convention:

ANSI C :

```
int               i_ReturnValue;
unsigned char b_BoardHandle;

i_ReturnValue = i_PA370_SetBoardInformation16BIT
                                                (0x390,
                                                 16,
                                                 0,
                                                 &b_BoardHandle);
```

Return value:

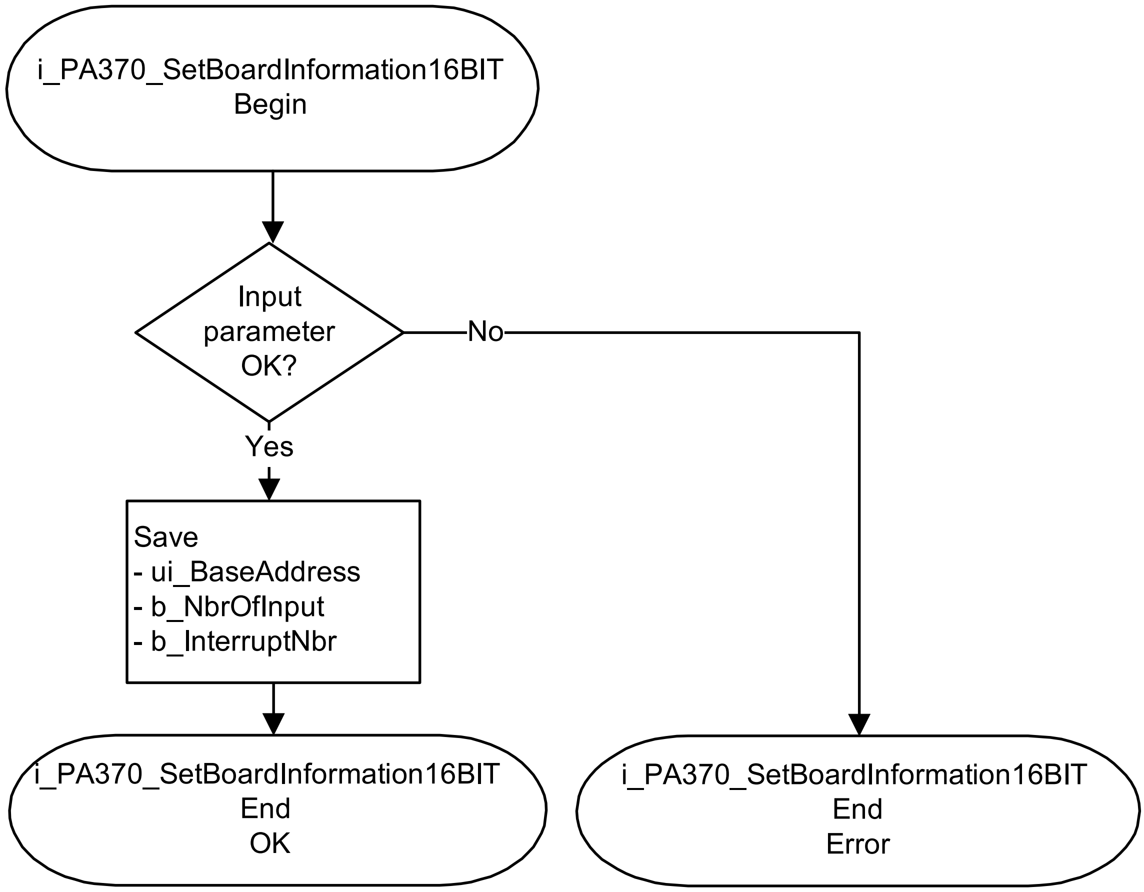
0: No error
-1: Board not present
-2: The number of analog inputs is wrong
-3: Interrupt number is wrong or already used by another **PA370**
-4: No handle is available for the board (up to 10 handles can be used)

[1] Identification number of the board

Input

ui_BaseAddress b_NbrOfInput
b_InterruptNbr

Function diagram


i_PA370_SetBoardInformation16BIT
Begin
Input
parameter
OK?
No
Yes
Save
- ui_BaseAddress
- b_NbrOfInput
- b_InterruptNbr
i_PA370_SetBoardInformation16BIT
End
OK
i_PA370_SetBoardInformation16BIT
End
Error

Output

pb_BoardHandle

<Return Value>

i **IMPORTANT!**
This function is only available for Windows NT/95 applications.

3) i_PA370_SetBoardInformationWin32 (...)

Syntax:

```
<Return value> = i_PA370_SetBoardInformationWin32
                                        (PCHAR   pc_IdentifierString,
                                         BYTE     b_NumberOfInput,
                                         PBYTE    pb_BoardHandle)
```

Parameters:

- Input:

PCHAR	pc_IdentifierString	Identifier string to be used for the board
BYTE	b_NumberOfInput	Number of analog inputs (8 or 16).

- Output:

PBYTE	pb_BoardHandle	Handle of the **PA 370** board to use the functions

Task:

Stores the following information:
- Base address
- Number of analog inputs
- Interrupt number

A handle is returned to the user which allows to use the next functions. Handles allow to operate several boards.

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA370_SetBoardInformationWin32
                                        ("PA370-00”,8,&b_BoardHandle);
```

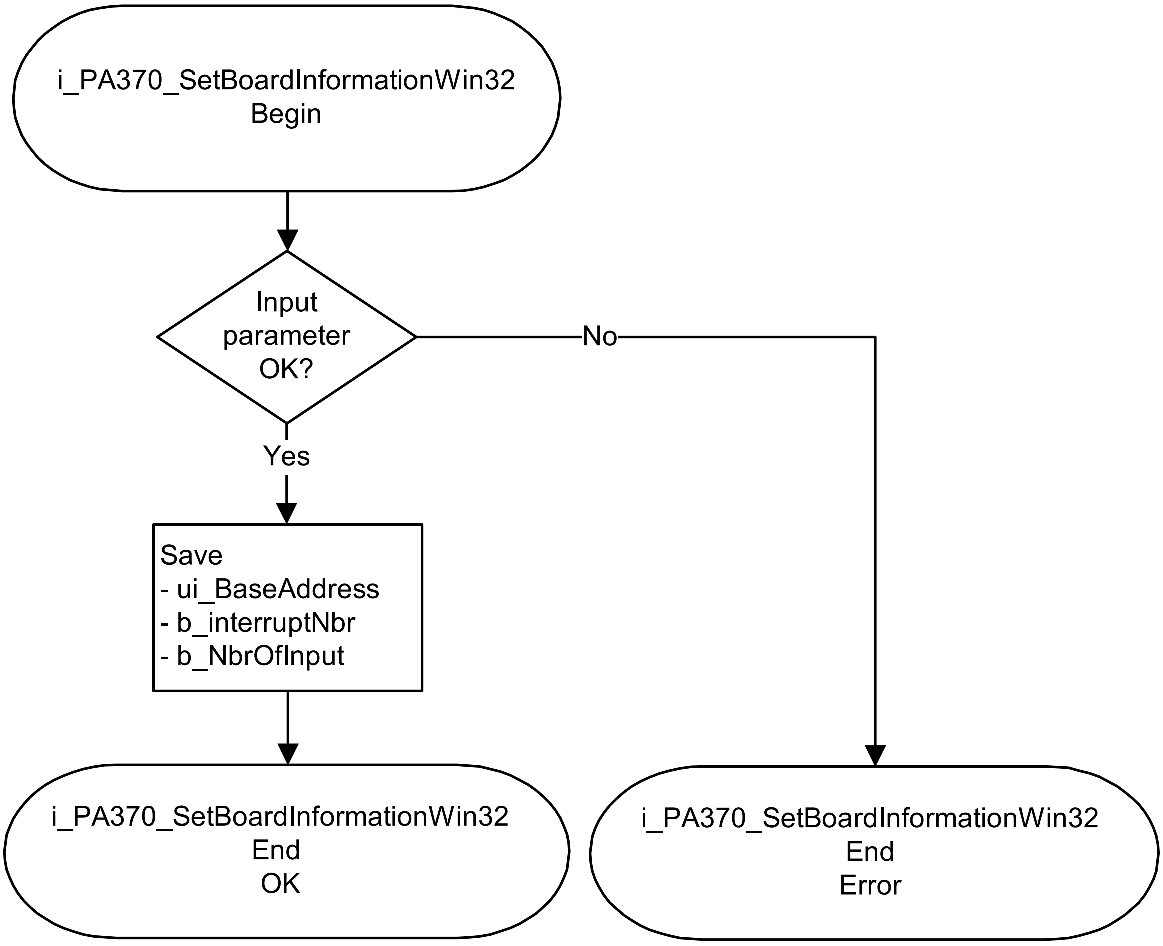
Return value:

0: No error
-1: Board not present
-2: The number of analog inputs is wrong.
-3: No handle is available for the board (up to 10 handles can be used)
-4: Error by opening the Windows 95/NT driver.

Input

pc_IdentifierString b_NumberOfInput

Function diagram


i_PA370_SetBoardInformationWin32
Begin
Input parameter OK?
No
Yes
Save
- ui_BaseAddress
- b_interruptNbr
- b_NbrOfInput
i_PA370_SetBoardInformationWin32
End
OK
i_PA370_SetBoardInformationWin32
End
Error

Output

pb_BoardHandle

<Return Value>

4) i_PA370_GetHardwareInformation (...)

Syntax:

```
<Return value> = i_PA370_GetHardwareInformation
                                          (BYTE     b_BoardHandle,
                                           PUINT    pui_BaseAddress,
                                           PBYTE    pb_InterruptNbr)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the **PA 370** board

- Output:

PUINT	pui_BaseAddress	Base address of the board.
PBYTE	pb_InterruptNbr	Interrupt number of the board.

Task:

Returns the base address and the interrupt number of the **PA 370.**

Calling convention:

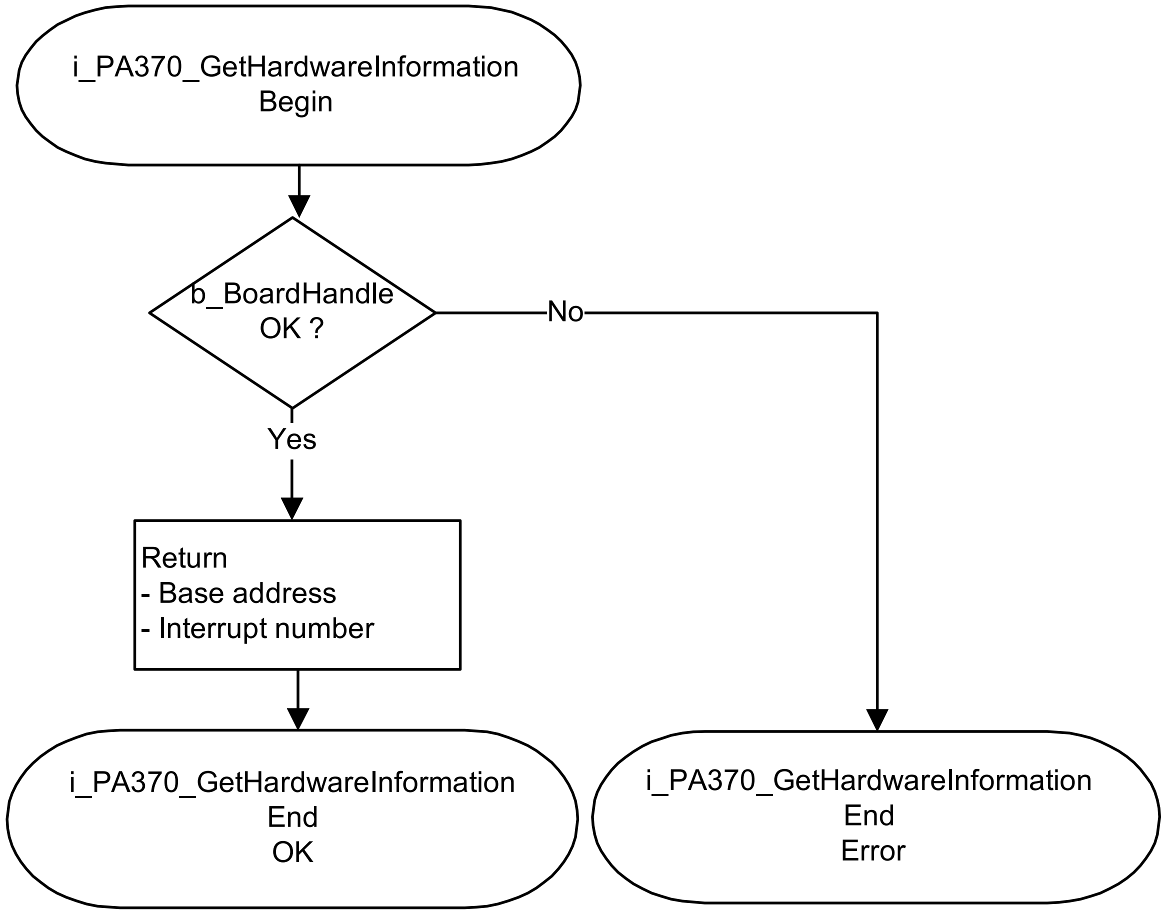
ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned int    ui_BaseAddress;
unsigned char   b_InterruptNbr;

i_ReturnValue = i_PA370_GetHardwareInformation
                        (b_BoardHandle,&ui_BaseAddress,&b_InterruptNbr);
```

Return value:

0: No error

-1: The handle parameter of the board is wrong.

Input
b_BoardHandle
Function diagram
 i_PA370_GetHardwareInformation Begin b_BoardHandle OK ? No Yes Return - Base address - Interrupt number i_PA370_GetHardwareInformation End OK i_PA370_GetHardwareInformation End Error
Output
pui_BaseAddress pb_InterruptNbr
<Return Value>

10.4 Interrupt

i

IMPORTANT!
This function is only available for Windows DOS applications.

1) i_PA370_SetBoardIntRoutineDos (..)

Syntax:

```
<Return value> = i_PA370_SetBoardIntRoutineDos
                              (BYTE    b_BoardHandle
                               VOID    v_FunctionName
                                            (BYTE    b_BoardHandle,
                                             BYTE    b_ChannelNbr,
                                             UINT    ui_ReadValue))
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the **PA 370** board
VOID	v_FunctionName	Name of the user interrupt routine

- Output:

No output signal has occurred.

Task:

This function must be called up for each **PA 370** on which an interrupt action is to be enabled.
First calling (first board):
- the user interrupt routine is installed,
- interrupts are enabled.

If you operate several **PA 370** boards which have to react to interrupts, call up the function as often as you operate **PA 370** boards.

The variable *v_FunctionName* is only relevant **for the first calling**.
From the second calling 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 generated when:
- the EOC has occurred

The user interrupt routine must have the following syntax:

```
VOID    v_FunctionName (BYTE          b_BoardHandle,
                        BYTE          b_ChannelNbr,
                        UINT          ui_ReadValue)
```

v_FunctionName Name of the user interrupt routine

b_BoardHandle Handle of the **PA 370** which has generated the interrupt

b_ChannelNbr Mask of the last analog channel which have generated the EOC interrupt

ui_ReadValue Last channel value.

The user can give another name for v_FunctionName, b_BoardHandle, b_ChannelNbr, ui_ReadValue.

Calling convention:

ANSI C :

```
void        v_FunctionName        (unsigned char   b_BoardHandle,
                                   unsigned char   b_ChannelNbr,
                                   unsigned int    ui_ReadValue)
            {
            .
            .
            }

            int              i_ReturnValue;
            unsigned char    b_BoardHandle;

i_ReturnValue = i_PA370_SetBoardIntRoutineDos        (b_BoardHandle,
                                                      v_FunctionName );
```

Return value:

0: No error

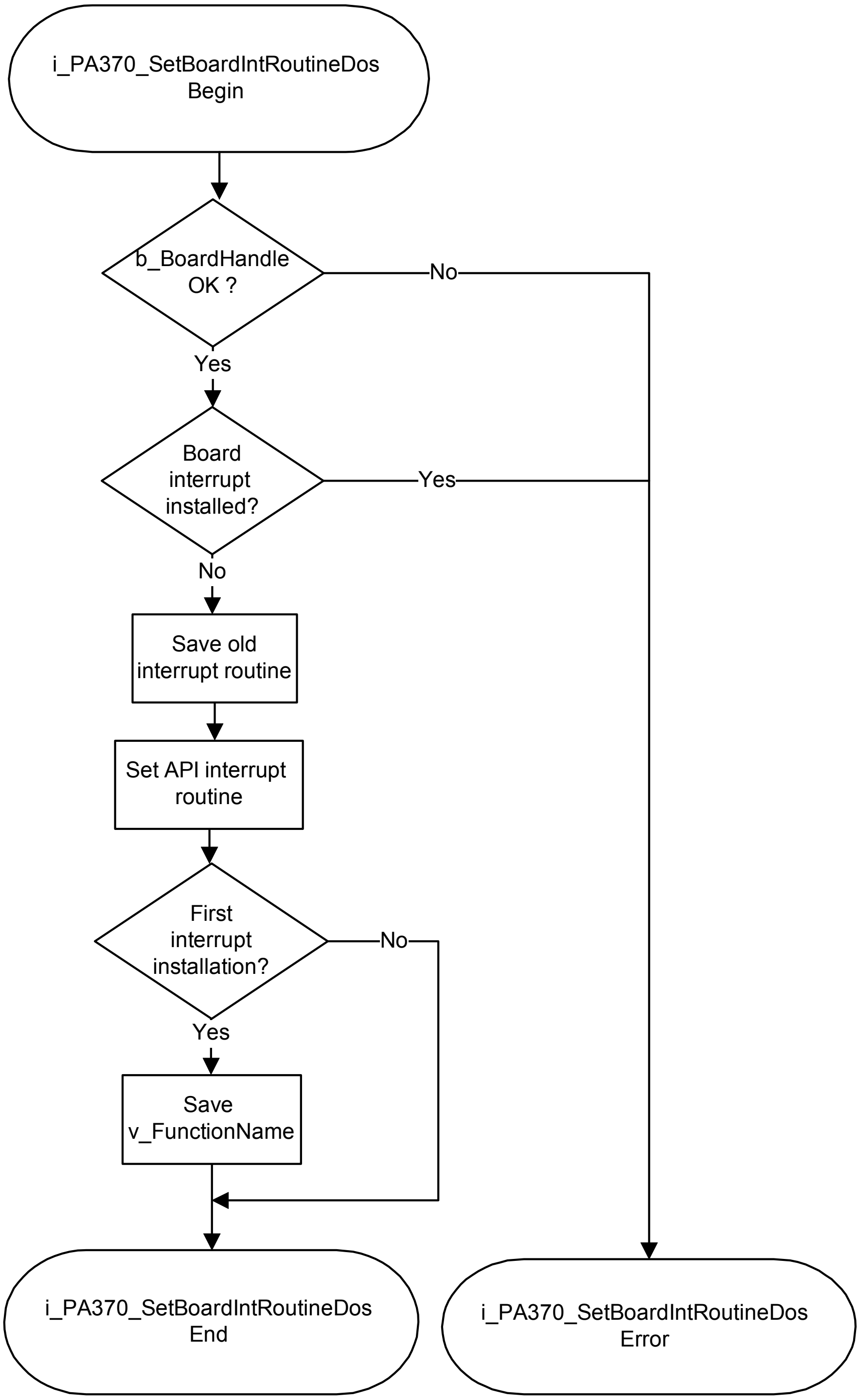
-1: The handle parameter of the board is wrong

-2: Interrupt already installed

Input

b_BoardHandle v_FunctionName

Function diagram



Output

<Return Value>

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

2) i_PA370_SetBoardIntRoutineVBDos (..)

Syntax:
<Return value> = i_PA370_SetBoardIntRoutineVBDos
(BYTE b_BoardHandle)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of **PA 370** board

- Output:
No output signal has occurred.

Task:
This function must be called up for each **PA 370** on which an interrupt is to be enabled. If an interrupt occurs, a Visual basic event is generated.
See calling convention.
When the function is called up for the first time (first board):
- interrupts are allowed for the selected board.
If you operate several **PA 370** boards which have to react to interrupts, call up the function as often as you operate **PA 370** 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_PA370_TestInterrupt"

This function tests the interrupt of the **PA 370**. It is used to obtain the values of *b_BoardHandle* , *b_ChannelNbr, ui_ReadValue*.

Calling convention:

Visual Basic DOS:

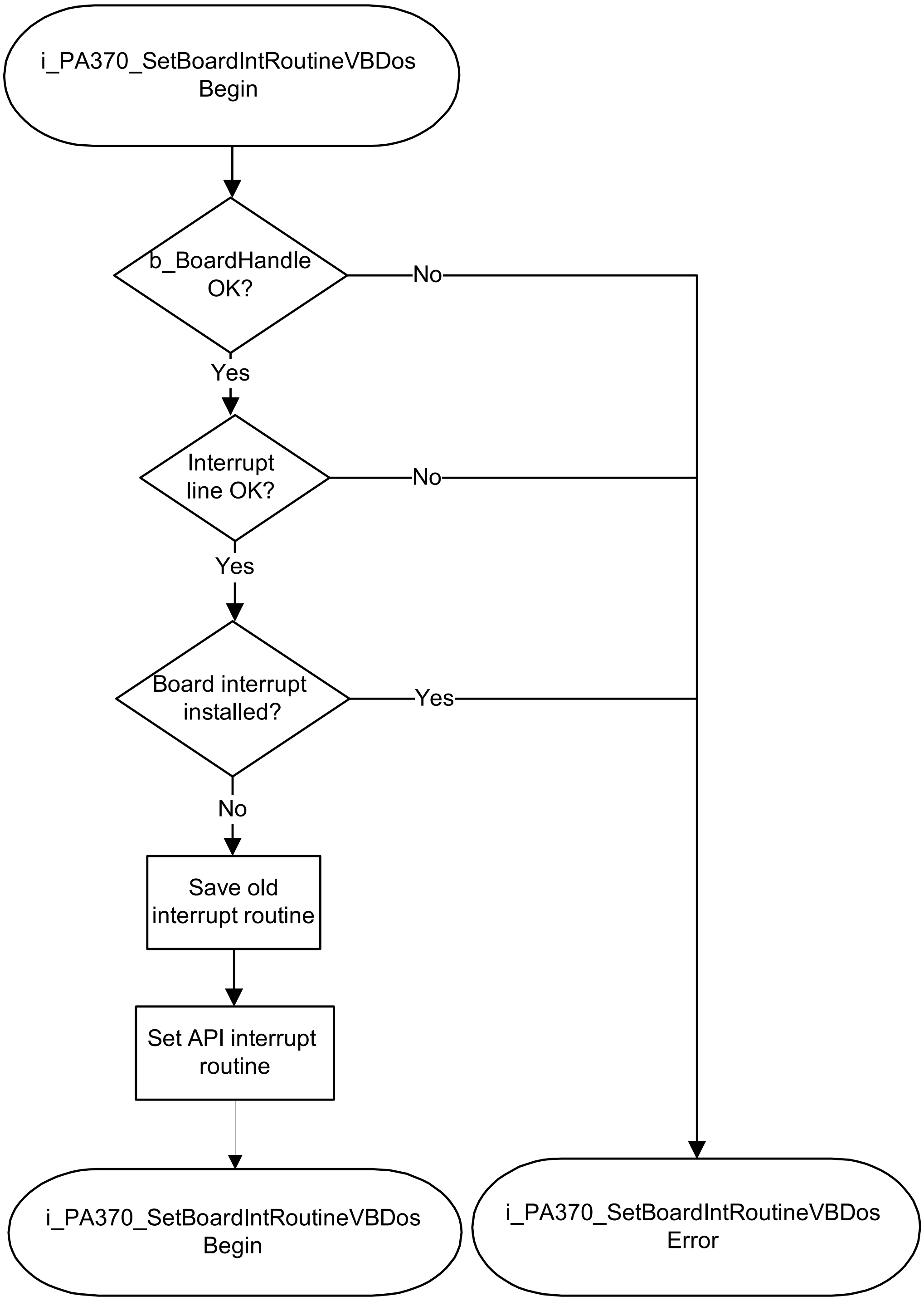
```
Dim Shared i_ReturnValue           As Integer
Dim Shared i_BoardHandle           As Integer
Dim Shared i_ChannelNbr            As Integer
Dim Shared l_ReadValue             As Long

    IntLabel:

    i_ReturnValue = i_PA370_TestInterrupt (i_BoardHandle, _
                                              i_ChannelNbr,  _
                                              l_ReadValue)
    .
    .
    .

    Return


   ON UEVENT GOSUB IntLabel
   UEVENT ON
i_ReturnValue = i_PA370_SetBoardIntRoutineVBDos    (b_BoardHandle)
```

Return value:

0: No error

-1: The handle parameter of the board is wrong

-2: Interrupt already installed

Input
b_BoardHandle
Function diagram
 i_PA370_SetBoardIntRoutineVBDos Begin b_BoardHandle OK? No Yes Interrupt line OK? No Yes Board interrupt installed? Yes No Save old interrupt routine Set API interrupt routine i_PA370_SetBoardIntRoutineVBDos Begin i_PA370_SetBoardIntRoutineVBDos Error
Output
<Return Value>

3) i_PA370_SetBoardIntRoutineWin16

Syntax:

```
<Return value> = i_PA370_SetBoardIntRoutineWin16
                              (BYTE  b_BoardHandle
                               VOID  v_FunctionName
                                          (BYTE  b_BoardHandle,
                                           BYTE  b_ChannelNbr,
                                           UINT  ui_ReadValue))
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the **PA 370** board
VOID	v_FunctionName	Name of the user interrupt routine

- Output:

No output signal has occurred.

Task:

This function must be called up for each **PA 370** on which an interrupt action is to be enabled.

First calling (first board):

- the user interrupt routine is installed
- interrupts are enabled.

If you operate several **PA 370** boards which have to react to interrupts, call up the function as often as you operate **PA 370** 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 generated when:

- the EOC has occurred.

You can make the interrupt management easier with the function
"i_PA370_SetBoardIntRoutineDos"

The user interrupt routine must have the following syntax:

```
VOID  v_FunctionName (BYTE     b_BoardHandle,
                      BYTE     b_ChannelNbr,
                      UINT     ui_ReadValue)
```

v_FunctionName	Name of the user interrupt routine
b_BoardHandle	Handle of the **PA 370** which has generated the interrupt
b_ChannelNbr	Mask of the last analog channel which has generated the EOC interrupt
ui_ReadValue	Last channel value.

i

IMPORTANT!
If you use Visual Basic (up to version 4.0) for Windows the parameter v_FunctionName has not signification. You have to use the "i_PA370_TestInterrupt" function.

Calling convention:
ANSI C :

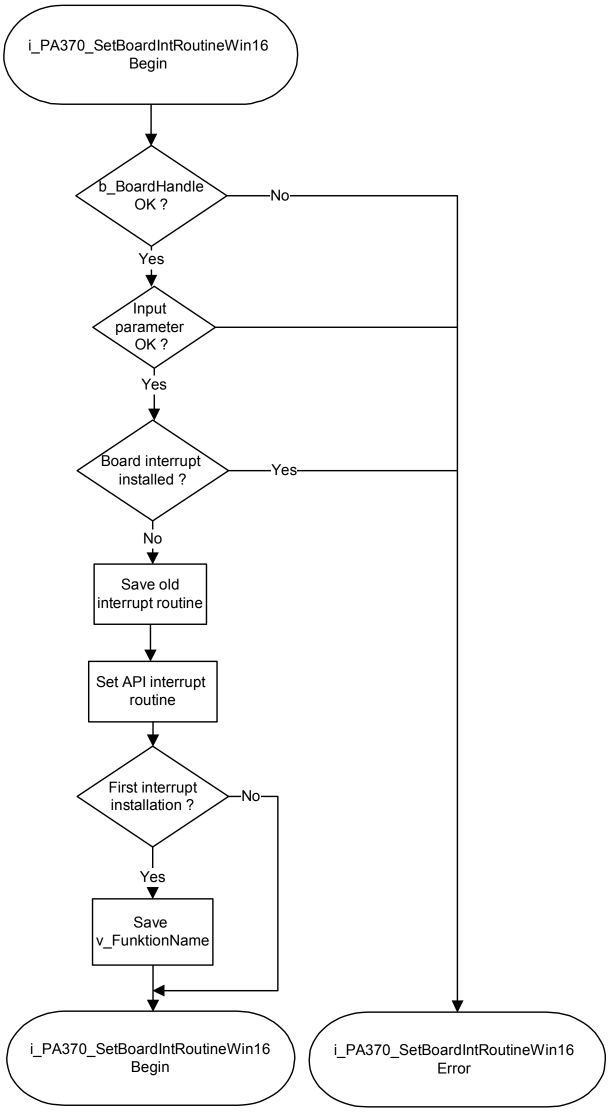
```
void        v_FunctionName        (unsigned char   b_BoardHandle,
                                   unsigned char   b_ChannelNbr,
                                   unsigned int       ui_ReadValue)
                  {
                  .
                  .
                  }

int               i_ReturnValue;
unsigned char     b_BoardHandle;

i_ReturnValue = i_PA370_SetBoardIntRoutineWin16     (b_BoardHandle,
                                                      v_FunctionName );
```

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

Input

b_BoardHandle
v_FunctionName

Function diagram


i_PA370_SetBoardIntRoutineWin16
Begin
b_BoardHandle OK ?
No
Yes
Input parameter OK ?
Yes
Board interrupt installed ?
Yes
No
Save old interrupt routine
Set API interrupt routine
First interrupt installation ?
No
Yes
Save v_FunktionName
i_PA370_SetBoardIntRoutineWin16
Begin
i_PA370_SetBoardIntRoutineWin16
Error

Output

<Return Value>

IMPORTANT!
This function is only available for Windows NT and Windows 95/98.

4) i_PA370_SetBoardIntRoutineWin32 (..)

Syntax:

```
<Return value> = i_PA370_SetBoardIntRoutineWin32
               (BYTE       b_BoardHandle,
                BYTE       b_UserCallingMode,
                ULONG      ul_UserSharedMemorySize,
                VOID ** ppv_UserSharedMemory,
                VOID       v_FunctionName    (BYTE     b_BoardHandle,
                                              BYTE     b_ChannelNbr,
                                              UINT     ui_ReadValue,
                                              BYTE     b_UserCallingMode,
                                              VOID *pv_UserSharedMemory))
```

Parameters:

-Input:

BYTE	b_BoardHandle	Handle of the **PA 370** board
BYTE	b_UserCallingMode	PA370_SYNCHRONOUS_MODE : The user routine is directly called by the driver interrupt routine. PA370_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 bytes of the user shared memory. Only used if you have selected PA370_SYNCHRONOUS_MODE

- Output:

VOID ** ppv_UserSharedMemory	User shared memory address Only used if you have selected PA370_SYNCHRONOUS_MODE

Task:

If you use Visual Basic 5.0 or 6.0:
- only the asynchronous mode is available.

<u>Windows 32-bit information:</u>
For Windows NT and Windows 95/98, 4 rings (ring 0 to ring 3) are available.

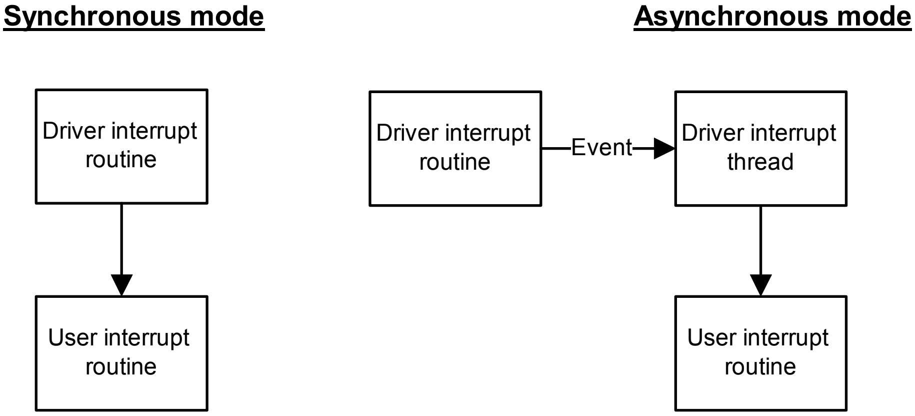
- The user application operates in ring 3. This ring does not give access to hardware.
- VXD and SYS driver operate in ring 0 and give access to hardware.
- Ring 0 has no direct access to global variable from ring 3. It has to use a shared memory.
- Ring 0 and ring 3 have a pointer that points on this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

This function must be called up for each **PA 370** 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 PA370_SYNCHROUNOUS_MODE has been selected.

If you operate several **PA 370** boards which have to react to interrupts, call up the function as often as you operate **PA 370** boards. The variable *v_FunctionName* is only relevant **for the first calling**.
From the second calling 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.

The user interrupt routine can be called:
- directly by the 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 the 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 have access to global variables. The user can still use a shared memory.
	This mode is not available for Visual Basic.

	ASYNCHRONOUS MODE
ADVANTAGES	The user can debug the user interrupt routine provided he did not program in Visual Basic 5.
	The user routine can call Windows API functions.
	The user routine can call functions which give access to global variables.
RESTRICTION	The code of the user interrupt routine is called by the driver interrupt thread routine (ring 3). The time between the interrupt and the user interrupt routine is increased.

Shared memory
If you have selected the PA370_SYNCHRONOUS_MODE you cannot have access to the Windows API 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 bytes 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_ChannelNbr,
                       UINT          ui_ReadValue,
                       BYTE          b_UserCallingMode,
                       VOID *        pv_UserSharedMemory)
```

v_FunctionName Name of the user interrupt routine
b_BoardHandle Handle of the **PA 370** which has generated the interrupt
b_ChannelNbr Mask of the last analog channel which has generated the EOC interrupt
ui_ReadValue Last channel value.

b_UserCallingMode PA370_SYNCHRONOUS_MODE: The user routine is directly called by the driver interrupt routine.
PA370_ASYNCHRONOUS_MODE: The user routine is called by the driver interrupt thread

pv_UserSharedMemory Pointer of the user shared memory.

IMPORTANT!
If you use Visual Basic 4 the following parameters have no meaning. You have to use the "i_PA370_TestInterrupt" function.

```
BYTE      b_UserCallingMode,
ULONG     ul_UserSharedMemorySize,
VOID **   ppv_UserSharedMemory,
VOID      v_FunctionName      (BYTE         b_BoardHandle,
                               BYTE         b_ChannelNbr,
                               UINT         ui_ReadValue,
                               BYTE         b_UserCallingMode,
                               VOID *       pv_UserSharedMemory)
```

The user can give another name for *v_FunctionName, b_BoardHandle, b_ChannelNbr, ui_ReadValue, b_UserCallingMode, pv_UserSharedMemory*.

Calling convention:

ANSI C :

```
 typedef struct
      {
   .
   .
   .
   }str_UserStruct;

str_UserStruct * ps_UserSharedMemory;

void      v_FunctionName      (unsigned char   b_BoardHandle,
                               unsigned char   b_ChannelNbr,
                               unsigned int    ui_ReadValue,
                               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_ PA370_SetBoardIntRoutineWin32
                                    (b_BoardHandle, PA370_SYNCHRONOUS_MODE,
                                     sizeof (str_UserStruct),
                                     (void **) &ps_UserSharedMemory,
                                     v_FunctionName);
```

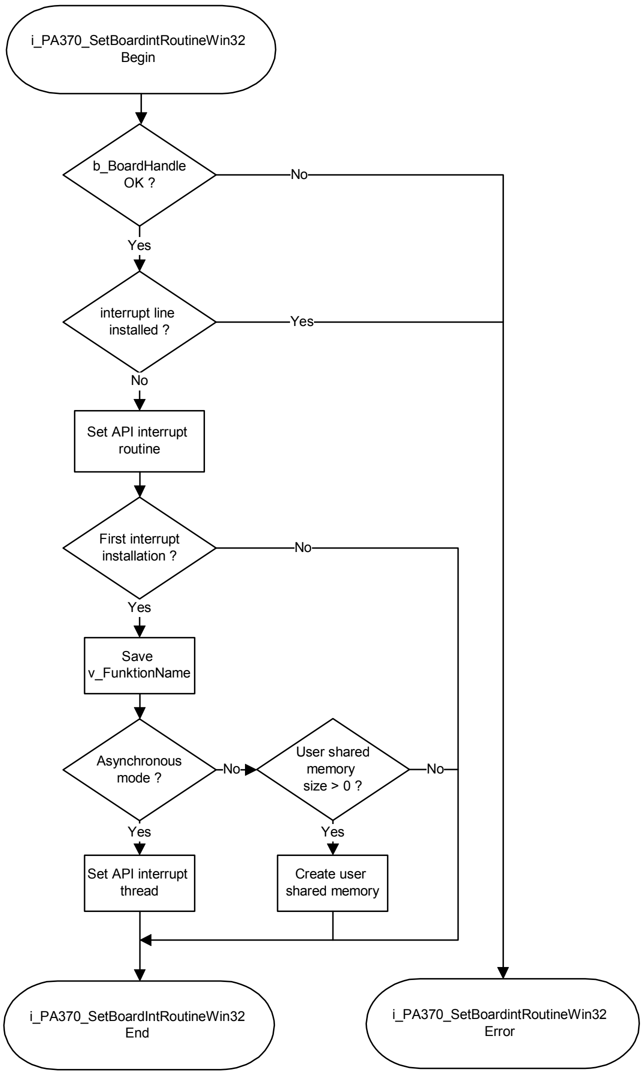
Return value:

 0: No error
-1: The handle parameter of the board is wrong
-2: Interrupt already installed
-3: Parameter b_UserCallingMode is wrong.
-4: No memory available for the user shared memory

Input

b_BoardHandle ul_UserSharedMemorySize
b_UserCallingMode v_FunctionName

Function diagram



Output

ppv_UserSharedMemory

<Return Value>

5) i_PA370_TestInterrupt (..)

Syntax:

```
<Return value> =  i_PA370_TestInterrupt  (PBYTE        pb_BoardHandle,
                                          PBYTE        pb_ChannelNbr,
                                          PINT         pi_ReadValue)
```

Parameters:

-Input:

No input signal has occurred.

- Output:

PBYTE pb_BoardHandle — Handle of the PA 370 board which has generated the interrupt,

PBYTE pb_ChannelNbr — Mask of the last analog input channel which has generated the EOC interrupt.

PINT pi_ReadValue — Last channel value.

Task:

Checks if a **PA 370** board has generated an interrupt. If yes, the function returns the board handle and the interrupt source.

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

Calling convention:

ANSI C :

```
unsigned char    b_BoardHandle;
unsigned char    b_ChannelNbr;
int              i_ReadValue;
int              i_Irq;

i_Irq = i_PA370_TestInterrupt      (&b_BoardHandle,
                                     & b_ChannelNbr,
                                     &i_ReadValue);
```

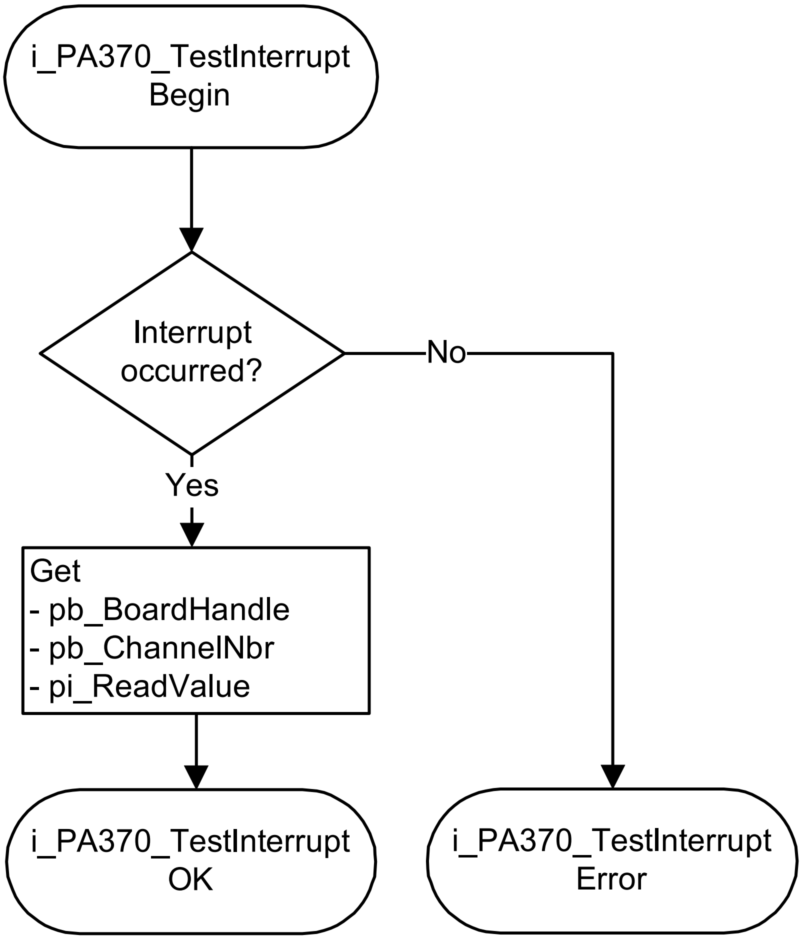
Return value:

-1: No interrupt

> 0: IRQ number

Input

Function diagram



Output

pb_BoardHandle pb_ChannelNbr
pi_ReadValue

<Return Value>

6) i_PA370_ResetBoardIntRoutine (..)

Syntax:

```
<Return value> =  i_PA370_ResetBoardIntRoutine
                                        (BYTE b_BoardHandle)
```

Parameters:

-Input:

BYTE b_BoardHandle	Handle of the **PA 370** board

- Output:

No output signal has occurred.

Task:

Stops the interrupt management of the **PA 370** board**.**
Deinstalls the interrupt routine if the interrupt management of all **PA 370** boards is stopped.

Calling convention:

ANSI C :

```
unsigned char      b_BoardHandle;

i_PA370_ResetBoardIntRoutine     (b_BoardHandle);
```

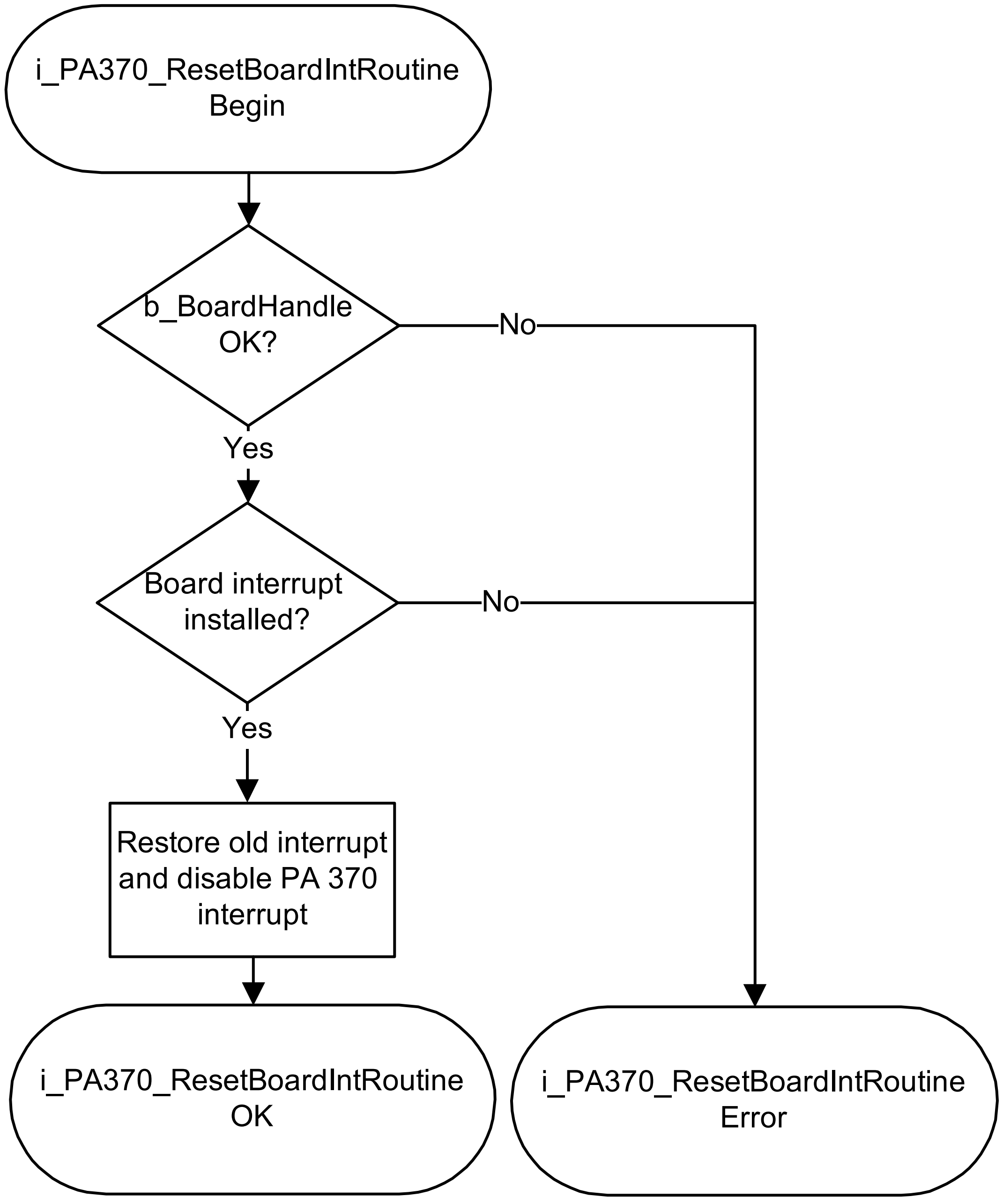
Return value:

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

Input

b_BoardHandle

Function diagram



Output

<Return Value>

7) i_PA370_CloseBoardHandle (...)

Syntax:

```
<Return value> = i_PA370_CloseBoardHandle
                                    (BYTE     b_BoardHandle)
```

Parameters:

-Input:

BYTE	b_BoardHandle	Handle of the PA 370 board

- Output:

No output signal has occurred.

Task:

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

Calling convention:

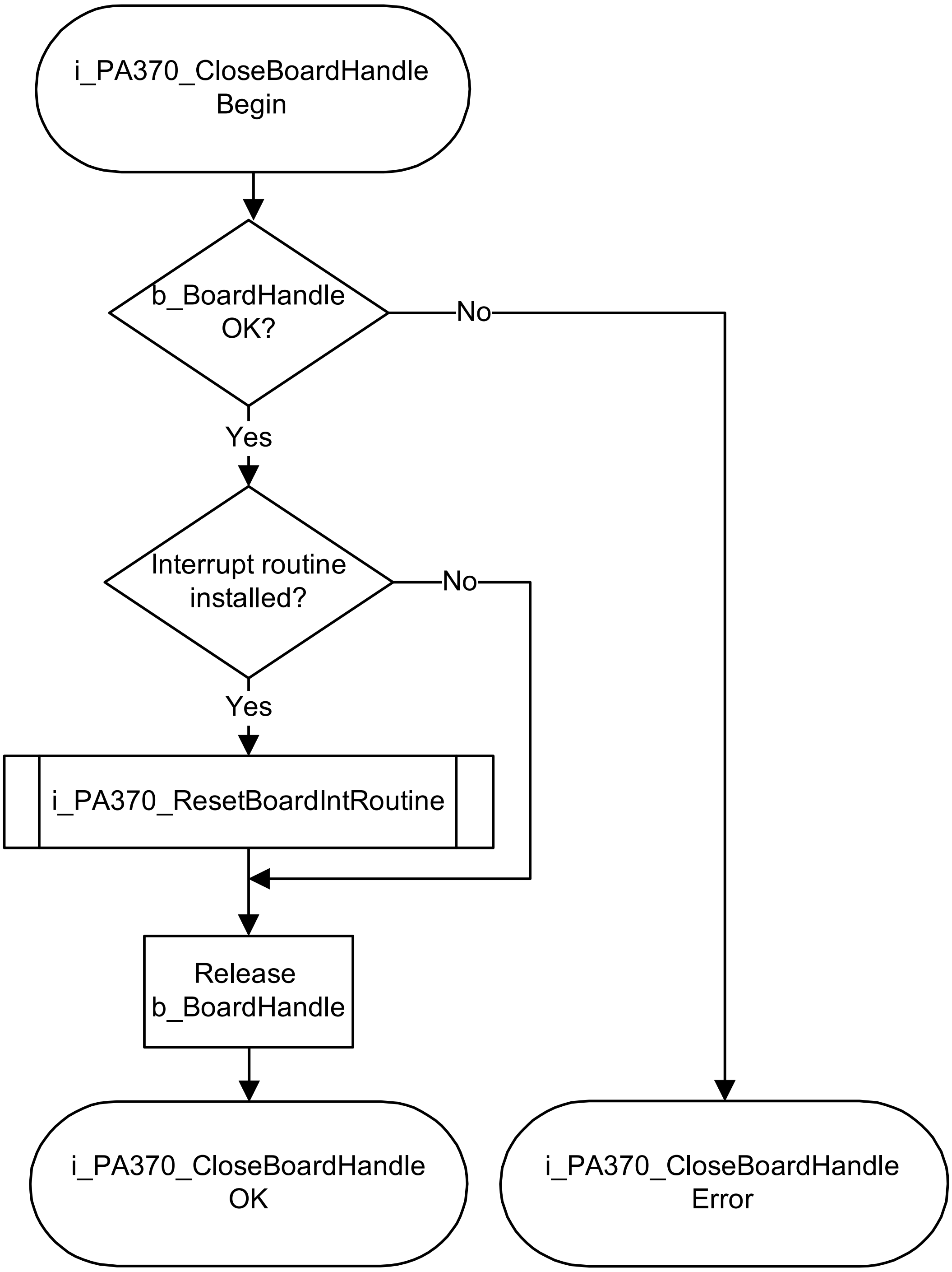
ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA370_CloseBoardHandle (b_BoardHandle);
```

Return value:

0: No error

-1: The handle parameter of the board is wrong

Input
b_BoardHandle
Function diagram
 i_PA370_CloseBoardHandle Begin b_BoardHandle OK? No Yes Interrupt routine installed? No Yes i_PA370_ResetBoardIntRoutine Release b_BoardHandle i_PA370_CloseBoardHandle OK i_PA370_CloseBoardHandle Error
Output
<Return Value>

10.5 Conversion of analog input channels

1) i_PA370_Read1AnalogInput (...)

Syntax:

```
<Return value> =  i_PA370_Read1AnalogInput
                                        (BYTE          b_BoardHandle,
                                         BYTE          b_Channel,
                                         BYTE          b_InterruptFlag,
                                         PINT          pi_AnalogInputValue)
```

Parameter:

- Input:

BYTE	b_BoardHandle	Handle of the **PA 370** board
BYTE	b_Channel	Number of the input to be read (1 to 16)
BYTE	b_InterruptFlag	PA370_ENABLE: An interrupt is generated at the end of the conversion. PA370_DISABLE: No interrupt is generated at the end of the conversion. The analog value is in the parameter pi_AnalogInputValue.

- Output:

PINT	pi_AnalogInputValue	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;
int              i_AnalogInputValue;

i_ReturnValue = i_PA370_Read1AnalogInput
                                             (b_BoardHandle,
                                              1,
                                              PA370_DISABLE,
                                              &i_AnalogInputValue);
```

Return value:

0: No error

-1: The handle parameter of the board is wrong

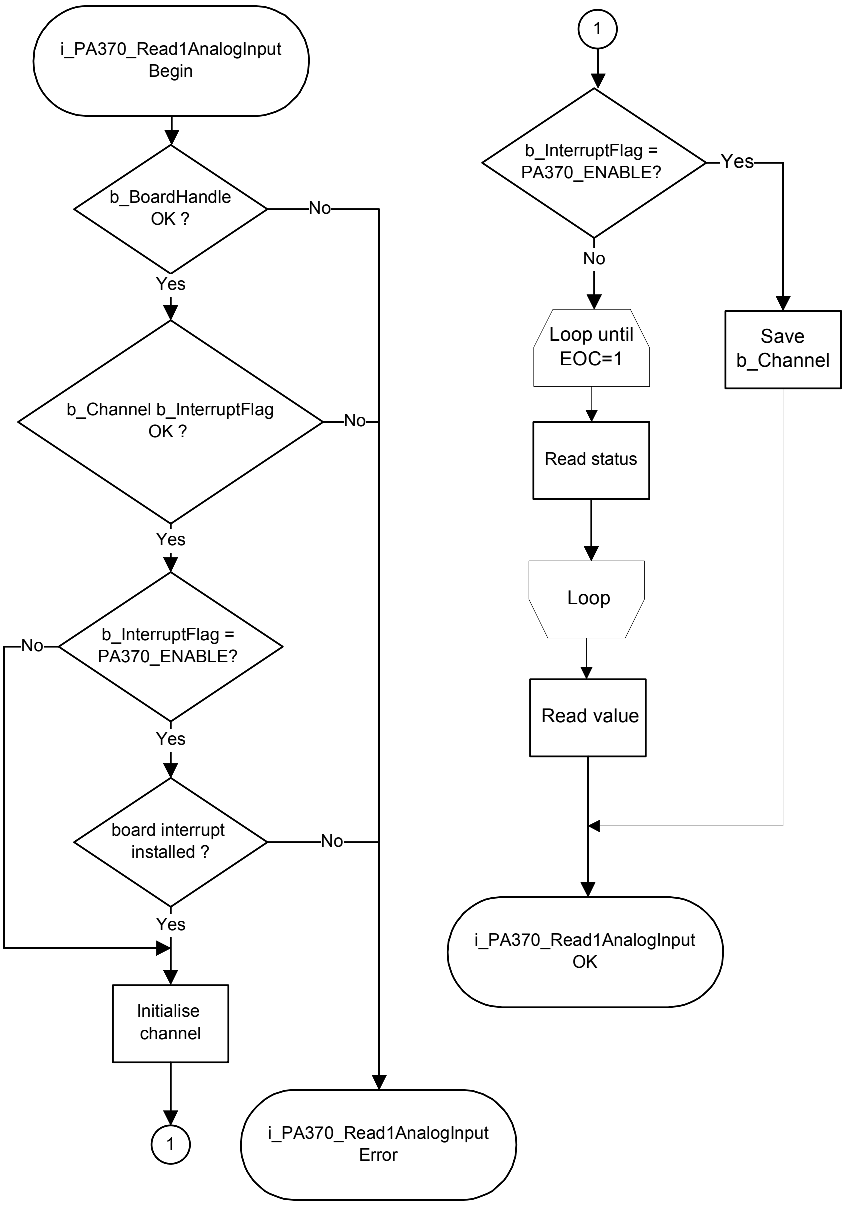
-2: The number of the analog input is wrong. See function "i_PA370_SetBoardInformationXX"

-3: A wrong parameter has been passed for b_InterruptFlag or the user interrupt routine has not been installed.

Input

b_BoardHandle b_InterruptFlag
b_Channel

Function diagram



i_PA370_Read1AnalogInput Begin
b_BoardHandle OK ?
No
Yes
b_Channel b_InterruptFlag OK ?
No
Yes
b_InterruptFlag = PA370_ENABLE?
No
Yes
board interrupt installed ?
No
Yes
Initialise channel
1
i_PA370_Read1AnalogInput Error
1
b_InterruptFlag = PA370_ENABLE?
Yes
No
Loop until EOC=1
Read status
Loop
Read value
Save b_Channel
i_PA370_Read1AnalogInput OK

Output

pi_AnalogInputValue

<Return Value>

2) i_PA370_ReadAllAnalogInput (...)

Syntax:

```
<Return value> =  i_PA370_ReadAllAnalogInput
                                          (BYTE          b_BoardHandle,
                                           PINT          pui_AnalogInputValueArray)
```

Parameter:

- Input:

BYTE	b_BoardHandle	Handle of the PA 370 board

- Output:

PINT	pui_AnalogInputValue	The analog values are returned.

Task:

Reads the current value of all analog inputs.

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
int             i_AnalogInputValue[16];


i_ReturnValue = i_PA370_ReadAllAnalogInput
                                        (b_BoardHandle,
                                         i_AnalogInputValue[0]);
```

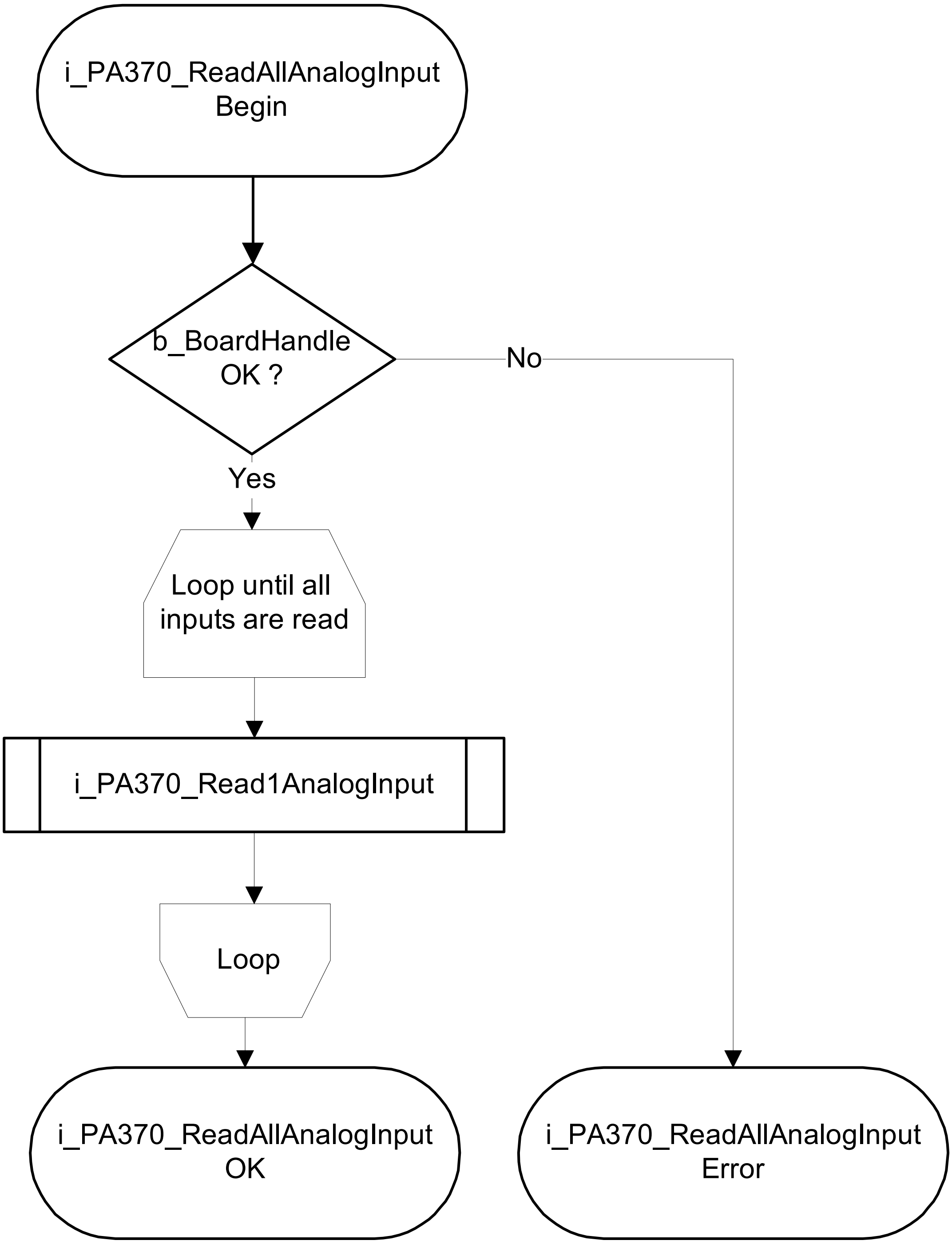
Return value:

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

Input

b_BoardHandle

Function diagram



Output

pi_AnalogInputValue

<Return Value>

10.6 Functions compliant with the former drivers (Windows NT/95)

i

IMPORTANT!
The new PA 370 driver is based on a new driver technology.

With this driver, the interrupt and I/O management is faster.
It is therefore recommended to use the functions:
- "i_PA370_SetBoardInformationWin32"
- "i_PA370_SetBoardIntRoutineWin32"

The functions "i_PA370_SetBoardAddress" and i_PA370_SetBoardIntRoutine" are only implemented in this driver to be used in your old application(s).

1) i_PA370_SetBoardInformation (...)

Syntax:

```
<Return value> = i_PA370_SetBoardInformation
                                          (UINT    ui_BaseAddress,
                                           PBYTE   pb_BoardHandle)
```

Parameters:
- Input:

UINT	ui_BaseAddress	Base address of the **PA 370** board

- Output:

PBYTE	pb_BoardHandle	Handle[1] of the **PA 370** to use the functions

Task:
Stores the following information:
- Base address

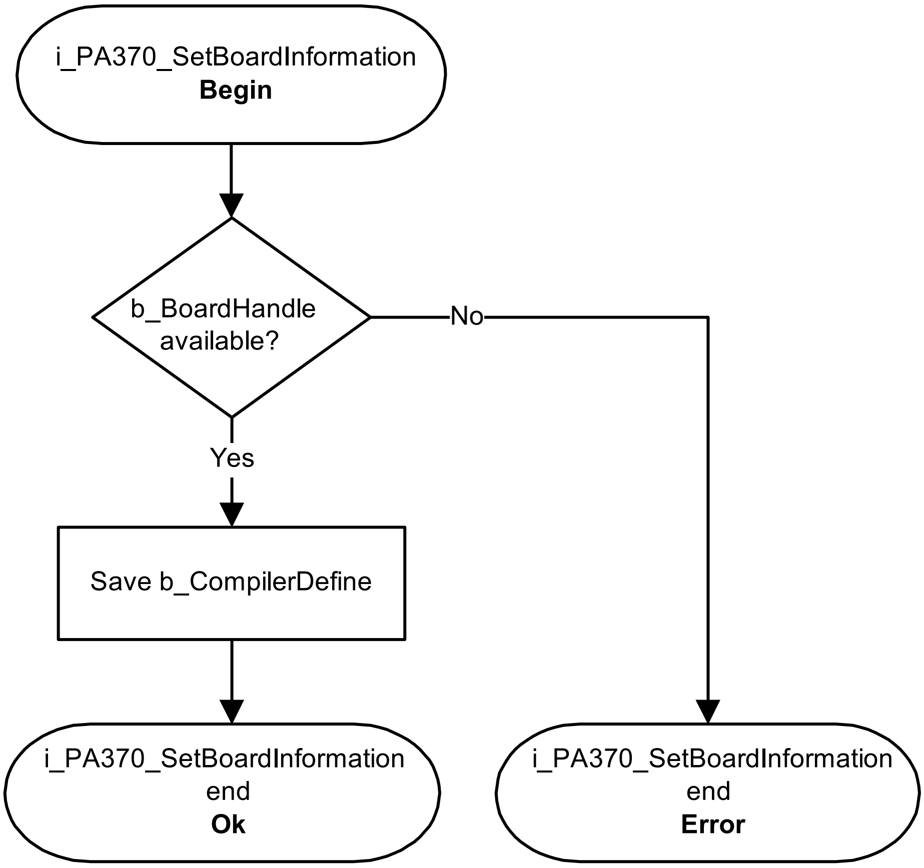
A handle is returned to the user which allows to use the next functions.
Handles allow to operate several boards.

Calling convention:
ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA370_SetBoardInformation(0x390,&b_BoardHandle);
```

Return value:
0: No error
-1: Base address already used
-2: No handle available for this board.

[1] Identification number of the board

Input
ui_BaseAddress
Function diagram
 i_PA370_SetBoardInformation **Begin** b_BoardHandle available? No Yes Save b_CompilerDefine i_PA370_SetBoardInformation end **Ok** i_PA370_SetBoardInformation end **Error**
Output
pb_BoardHandle
<Return Value>

2) i_PA370_SetBoardIntRoutine (...)

Syntax:

```
<Return value> = i_PA370_SetBoardIntRoutine
                                        (BYTE     b_BoardHandle,
                                         BYTE     b_InterruptNbr,
                                         VOID     v_FunctionName
                                                  (BYTE    b_BoardHandle,
                                                   BYTE    b_ChannelNbr,
                                                   UINT    ui_ReadValue))
```

Parameters:

- Input:

PCHAR	pc_IdentifierString	Identifier string to be used for the board
BYTE	b_InterruptNbr	Interrupt line.(3,5,10,11,12,15)
VOID	v_FunctionName	Name of the user interrupt routine

- Output:

No output signal has occurred.

Task:

This function must be called up for each **PA 370** on which you want to enable an interrupt action. It installs one user interrupt function for all boards on which you have enabled the interrupt.

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

If you operate several boards **PA 370** which have to react to interrupts, call up the function as often as you operate boards **PA 370**. The variable *v_FunctionName* is only relevant **for the first calling**.

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

An interrupt is generated when:
- an EOC has occurred

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.

The user interrupt routine must have the following syntax:

```
VOID    v_FunctionName (BYTE        b_BoardHandle,
                        BYTE        b_ChannelNbr,
                        UINT        ui_ReadValue)
```

v_FunctionName Name of the user interrupt routine

b_BoardHandle Handle of the **PA 370** which has generated the interrupt

b_ChannelNbr Mask of the last analog input channel which has generated the interrupt.

ui_ReadValue Last channel value.

The user can give another name for *v_FunctionName*, *b_BoardHandle*, *b_ChannelNbr, ui_ReadValue.*

Calling convention:

ANSI C :

```
void        v_FunctionName          (unsigned char   b_BoardHandle,
                                     unsigned char   b_ChannelNbr,
                                     unsigned int    ui_ReadValue)
            {
            .
            .
            }

int                 i_ReturnValue;
unsigned char       b_BoardHandle;

i_ReturnValue = i_PA370_SetBoardIntRoutine
                                        (b_BoardHandle,3,v_FunctionName);
```

Return value:

0: No error

-1: The handle of the board is wrong

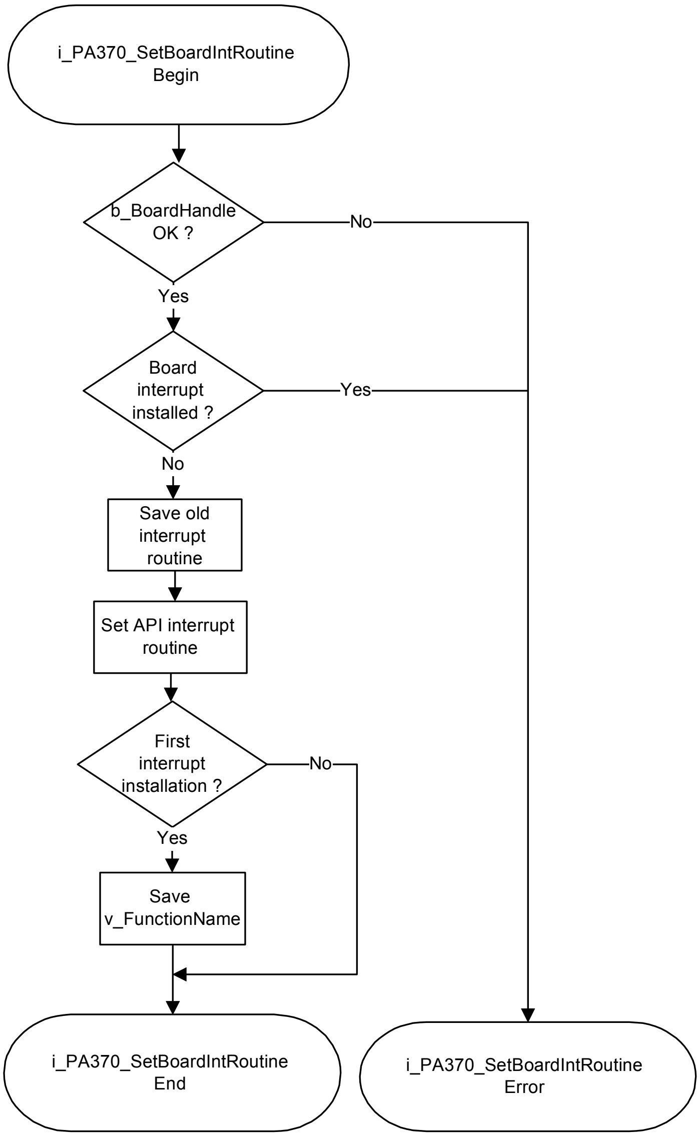
-2: Interrupt line already used.

-3: Interrupt line not available.

Input

b_BoardHandle b_InterruptNumber
v_FunctionName

Function diagram


i_PA370_SetBoardIntRoutine Begin
b_BoardHandle OK ?
No
Yes
Board interrupt installed ?
Yes
No
Save old interrupt routine
Set API interrupt routine
First interrupt installation ?
No
Yes
Save v_FunctionName
i_PA370_SetBoardIntRoutine End
i_PA370_SetBoardIntRoutine Error

Output

pb_BoardHandle

<Return Value>

INDEX