



ISO 9001 certified



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



Technical description

PA 2000

Digital output board, optically isolated

Edition: 05.04 - 11/2006

Copyright

All rights reserved. This manual is intended for the manager and its personnel.
No part of this publication may be reproduced or transmitted by any means.
Offences can have penal consequences.

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 ADDISET and ADDIMON. The right for using this 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 carriers).
- 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.



Declaration of Conformity

Document-Number/Month-Year: B-25807 / 10.1995

Manufacturer/Importer: ADDI-DATA GmbH
Dieselstraße 3
D-77833 OTTERSWEIER

Type: **PA 2000**

Product description: **Board to be inserted in an ISA slot of a PC**
32 digital outputs
Timer, watchdog

The above named product complies with the following European directives:

Directive 72/23/EEC of 19 February 1973 on the harmonization of the laws of Member States relating to electrical equipment designed for use within certain voltage limits.

Directive 89/336/EEC of 3 May 1989 on the approximation of the laws of the Member States relating to electromagnetic compatibility.

The following norms have been applied:

IEC 61010-1 2002-08

IEC 61326-2 2004

2004/11/10

Date

H. J. J. J.

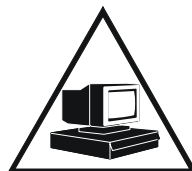
Legally valid signature of the manufacturer

WARNING

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



people may be injured



the board, PC and peripheral devices may be damaged



the environment may be polluted

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

- **Read the yellow safety leaflet carefully!**

If this leaflet is not supplied with the documentation, please contact us.

- **Observe the instructions in the manual!**

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

- **Symbols used**



WARNING!

designates a possibly dangerous situation.

If the instructions are ignored **the board, PC and/or peripheral devices may be damaged.**



IMPORTANT!

designates useful information.

- **Do you have any questions?**

Do not hesitate to contact us. Our technical support is always glad to help you.

1	INTENDED PURPOSE OF THE BOARD	1
1.1	Limits of use	1
2	USER	2
2.1	Qualification	2
2.2	Personal protection	2
3	HANDLING THE BOARD	3
4	TECHNICAL DATA	4
4.1	Electromagnetic compatibility (EMC)	4
4.2	Physical set-up of the board	4
4.3	Limit values	5
5	SETTINGS	7
5.1	Settings at delivery	7
5.1.1	Component scheme	7
5.1.2	Jumper location and settings at delivery	8
5.1.3	Jumper settings at delivery	8
	Base address	8
	Address decoding logic	8
	Data bus access	9
	Control of the output channels 31 and 32	9
	Selection of the interrupt line to the PC bus	10
5.2	Setting the base address through DIP switches	10
6	INSTALLATION	12
6.1	Inserting the board	13
6.1.1	Opening the PC	13
6.1.2	Plugging the board into the slot	14
6.1.3	Closing the PC	14
6.2	Installing the software	15
6.2.1	Software installation under MS-DOS and Windows 3.11	15
6.2.2	Software installation under Windows NT/95/98	15
6.3	Board configuration with ADDIREG	16
6.3.1	Program description	16
6.3.2	Registering a new board	19
6.3.3	Changing the registration of a board	20
6.3.4	Uninstalling the ADDIREG program	21
6.4	Technical support and software downloads	21
7	CONNECTION TO THE PERIPHERAL	22

7.1	Connector pin assignment.....	22
7.2	Connection principle	23
7.3	Connection examples.....	23
7.3.1	Connection to the external peripheral through a PX 901-D	24
7.3.2	Connection to the external peripheral through a PX 9000.....	25
8	FUNCTIONS OF THE BOARD.....	26
8.1	I/O mapping	26
	8-bit access	28
	16-bit access	30
8.2	Block diagram.....	31
8.3	Digital outputs.....	32
8.3.1	Features of the outputs	32
8.3.2	Setting the outputs 1 to 32	33
8.3.3	Special functions.....	34
	Output channels 31 and 32 as generators of square wave signals	34
	Timer	34
8.4	Interrupt.....	35
8.5	Timer.....	35
	Data.....	36
9	STANDARD SOFTWARE	37
9.1	Introduction	37
9.2	DIN 66001- Graphical symbols	39
9.3	Address.....	40
	1) i_PA2000_SetBoardAddress (...)	40
	2) i_PA2000_SetBoardIntRoutine (..).....	42
	3) i_PA2000_InitCompiler (..)	46
	4) i_PA2000_SetBoardInformation (..)	48
	5) i_PA2000_SetBoardInformationWin32 (..).....	50
	6) i_PA2000_GetHardwareInformation (...)	52
	7) i_PA2000_CloseBoardHandle	54
9.4	Interrupt.....	56
	1) i_PA2000_SetBoardIntRoutineDos (..).....	56
	2) i_PA2000_SetBoardIntRoutineVBDOs (..)	59
	3) i_PA2000_SetBoardIntRoutineWin16	61
	4) i_PA2000_SetBoardIntRoutineWin32 (..)	64
	5) i_PA2000_TestInterrupt (..).....	70
	6) i_PA2000_ResetBoardIntRoutine (..).....	72
9.5	Digital Output channels	74
	1) i_PA2000_SetOutputMemoryOn (...).....	74
	2) i_PA2000_SetOutputMemoryOff (...).....	76
	3) i_PA2000_Set1DigitalOutputOn (...).....	78
	4) i_PA2000_Set1DigitalOutputOff (...).....	80

5) i_PA2000_Set8DigitalOutputOn (...)	82
6) i_PA2000_Set8DigitalOutputOff (...)	84
7) i_PA2000_Set16DigitalOutputOn (...)	86
8) i_PA2000_Set16DigitalOutputOff (...)	88
9) i_PA2000_Set32DigitalOutputOn (...)	90
10) i_PA2000_Set32DigitalOutputOff (...)	92
9.6 Square wave generator.....	94
1) i_PA2000_InitSquareGenerator1 (...)	94
2) i_PA2000_InitSquareGenerator2 (...)	97
3) i_PA2000_StartSquareGenerator1(...)	100
4) i_PA2000_StartSquareGenerator2(...)	102
5) i_PA2000_StopSquareGenerator1 (...)	104
6) i_PA2000_StopSquareGenerator2 (...)	106
7) i_PA2000_ReadSquareGenerator1 (...)	108
8) i_PA2000_ReadSquareGenerator2 (...)	110
9.7 Timer/watchdog	112
1) i_PA2000_InitTimerWatchdog (...)	112
2) i_PA2000_StartTimerWatchdog(...)	115
3) i_PA2000_StopTimerWatchdog(...)	117
INDEX	A

Figures

Fig. 3-1: Wrong handling	3
Fig. 3-2: Correct handling.....	3
Fig. 5-1: Component scheme of the PA 2200 board.....	7
Fig. 5-2: Jumper location on the PA 2000	8
Fig. 5-3: DIP switches S1	11
Fig. 6-1: Types of slots	13
Fig. 6-3: Inserting the board.....	14
Fig. 6-4: Securing the board at the back cover.....	14
Fig. 6-5: ADDIREG registration program	16
Fig. 6-6: Configuring a new board	18
Fig. 7-1: 37-pin SUB-D male connector	22
Fig. 7-2: Connection principle	23
Fig. 7-3: Connection examples.....	23
Fig. 7-4: Connection to screw terminal panels and relay boards.....	24
Fig. 7-5: Connection to the external peripheral through a PX 901-D	24
Fig. 7-6: Connection to the external peripheral through a PX 9000.....	25
Fig. 8-1: Block diagram of the PA 2000	31
Fig. 8-2: Output circuitry	34
Fig. 8-3: Timer circuitry	36

Tables

Table 5-1: Address decoding logic	8
Table 5-2: Data bus width.....	9
Table 5-3: Control of the output channels 31 and 32	9
Table 5-4: Selection of the interrupt line.....	10
Table 5-5: Decoding table	11
Table 8-1: I/O map for 8-bit access (J3 open).....	27
Table 8-2: I/O map for 16-bit access (J3 is set).....	27
Table 8-3: Description of the address Base +0 (8-bit write access for the outputs 1-8)	28
Table 8-4: Description of the address Base + 1 (8-bit write access for the outputs 9-16)	28
Table 8-5: Description of the address Base + 2 (8-bit write access for the outputs 17-24)	28
Table 8-6: Description of the address Base + 3 (8-bit write access for the outputs 25-32)	28
Table 8-7: Description of the address Base + 8 (8-bit write access for the COMMAND_STATUS_REGISTER)	29
Table 8-8: Description of the address Base + 8 (8-bit write access for the ERROR_STATUS_REGISTER)	29
Table 8-9: Description of the address Base + 0 (16-bit write access for the outputs 1-16)	30
Table 8-10: Description of the address Base + 2 (16-bit write access for the outputs 1-16)	30
Table 9-1: Type Declaration for Dos and Windows 3.1X.....	37
Table 9-2: Type Declaration for Windows 95/98/NT	37
Table 9-3: Define value	38
Table 9-4: Interrupt mask	57
Table 9-5: Synchronous and asynchronous modes.....	66

1 INTENDED PURPOSE OF THE BOARD

The **PA 2000** board is the interface between an industrial process and a personal computer (PC).

It is to be used in a free 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  mark.

Data exchange between the PA 2000 board and the peripheral is to occur through a shielded cable. It has to be connected to the 37-pin SUB-D male connector of the PA 2000 board.

The board has 32 output channels for processing digital 24 V signals.

An external 24 V supply voltage is necessary to run the outputs.

The terminal panel **PX 901** and the relay board **PX 8500** allow connecting the 24 V supply voltage through a shielded cable.

The use of the PA 2000 board in combination with external screw terminal panels or relay boards is to occur in a closed switch cabinet; the installation is to be effected competently.

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

The connection with our standard cable ST010 complies with the specifications:

- metallized plastic hoods
- shielded cable shield folded back and firmly screwed to the connector housing.

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.

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

1.1 Limits of use

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.

Our boards are not to be used for securing emergency stop functions.

The emergency stop functions are to be secured separately.

This securing must not be influenced by the board or the PC.

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:

- 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

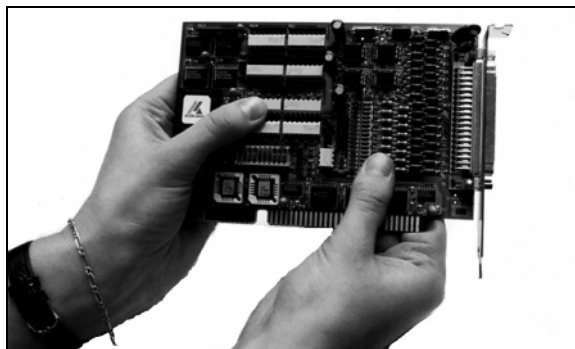
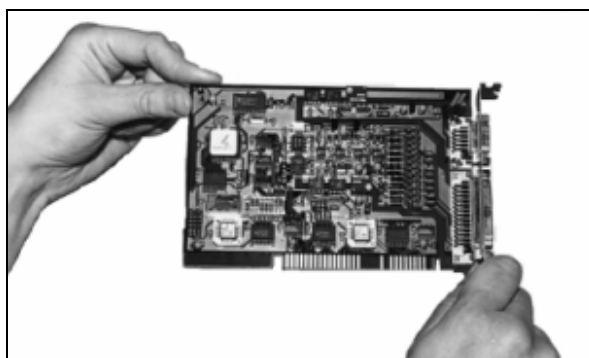


Fig. 3-2: Correct handling



4 TECHNICAL DATA

4.1 Electromagnetic compatibility (EMC)

The board has been subjected to EMC tests in an accredited laboratory in accordance with the norms EN50082-2, EN55011, EN55022

The board complies as follows with the limit values set by the norm EN50082-2:

	<u>True value</u>	<u>Set value</u>
ESD.....	4 kV	4 kV
Fields	10 V/m	10 V/m
Burst.....	4 kV	2 kV
Conducted radio interferences	10 V	10 V
50 Hz magnetic field	30 A/m	30 A/m



WARNING!

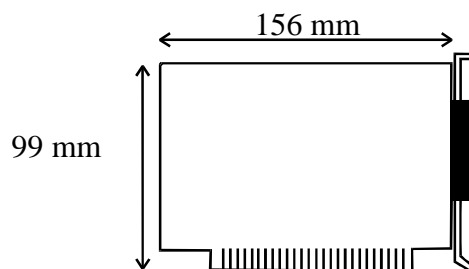
The EMC tests have been carried out in a specific appliance configuration. We guarantee these limit values **only** in this configuration¹⁾.

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:	160 g
Installation in:	XT / AT slot
Connection to the peripheral Standard cables:	through 37-pin SUB-D male connector ST010, ST011 or ST010-S for high currents
Screw terminal panels:	PX 901-D, PX 901-DG or PX 9000
Relay board:	PX 8500 cascable with cable ST8500
See Fig 7-4: Connection to screw terminal panels and relay boards	

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: 30% to 95% non condensing

Minimum PC requirements:

ISA bus interface
 Bus speed: 8 MHz

Energy requirements:

Operating voltage: 5 V \pm 5%
 Current consumption: 186 mA \pm 10 % typ.
 External operating voltage/current consumption
 (24 V industrial network - without connected peripheral)
 at 24 V 40 mA \pm 5 mA

24 V digital output channels

Output type: high side (load at ground)
 Number of output channels: 32
 Nominal voltage: 24 VDC
 Supply voltage range: 10 V to 36 VDC (min. 5 V)
 Maximum output current for the
 channels 1 to 16: 3 A typ. (protected through
 self resetting fuse)
 Maximum output current for the
 channels 17 to 32: 3 A typ. (protected through
 self resetting fuse)
 Maximum output current / channel: 500 mA
 Short-circuit current / output; shut-down at 24 V,
 $R_{load} < 0,1 R$: 1.5 A max.
 (switches off the output)
 $R_{DS\ ON}$ resistance: 0.4 R max.
 Switch ON time at 24 V, R_{load} , 350 mA: 150 μ s typ.
 Switch OFF time at 24 V, R_{load} , 350 mA: 40 μ s typ.
 Overtemperature: 170°C (switches off the
 components = 4 outputs).
 Temperature hysteresis: 20°C

Safety

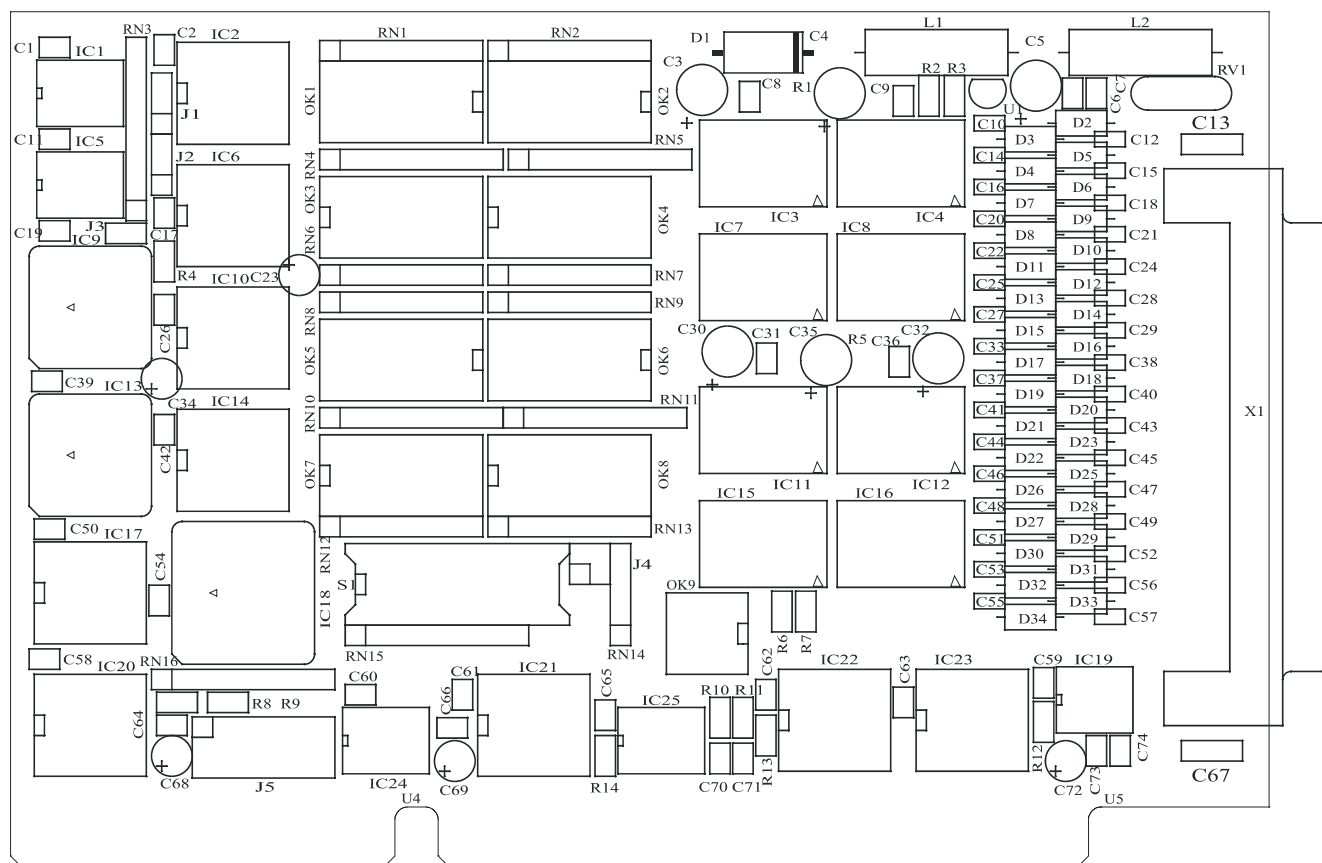
Optical isolation (DIN VDE 0411-100):	1000 V (from the PC to the external peripheral).
Shut down logic:	When the 24 V ext. voltage supply drops below 5 V, the output channels are switched off. Diagnostic through status bit or interrupt to the PC
Counters or timers:	3
Watchdog:	Timer-programmable. Resets all the outputs if no software trigger has happened. Times from 2 ms to 65 s are available.

5 SETTINGS

5.1 Settings at delivery

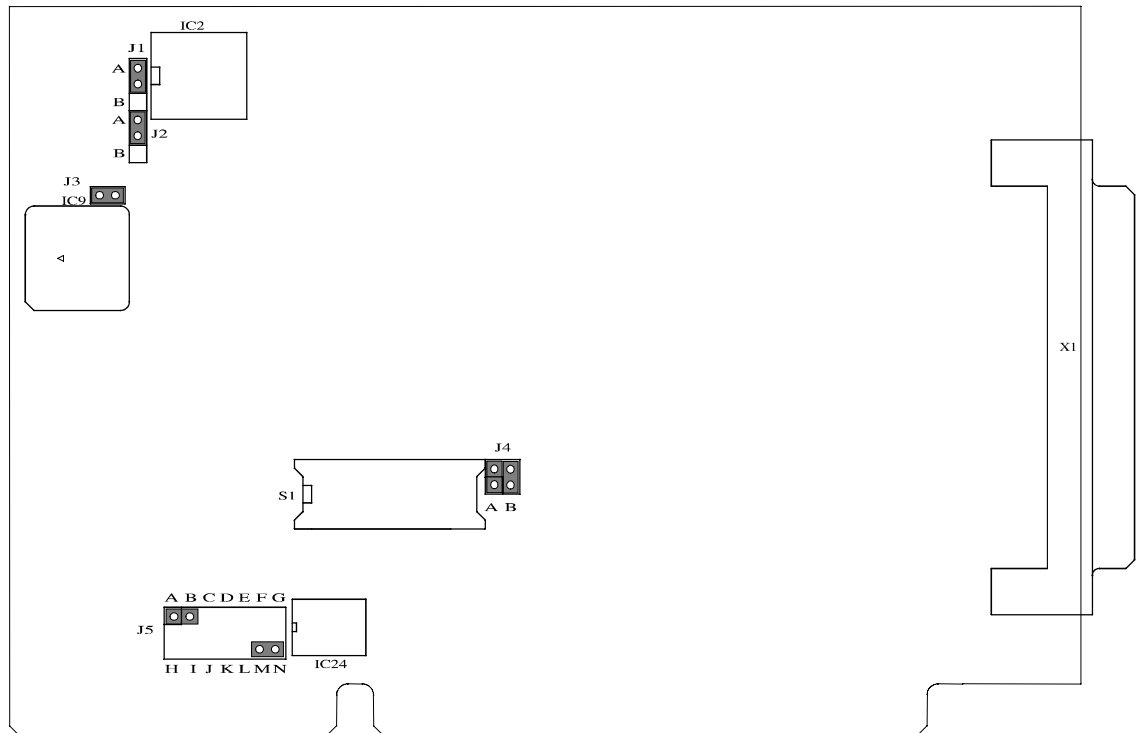
5.1.1 Component scheme

Fig. 5-1: Component scheme of the PA 2200 board



5.1.2 Jumper location and settings at delivery

Fig. 5-2: Jumper location on the PA 2000



5.1.3 Jumper settings at delivery

i

IMPORTANT!


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

Base address

S1 DIP switches set to 0390H .
See paragraph 5.2



Address decoding logic

Table 5-1: Address decoding logic

Jumper settings	Function	Settings at delivery
<p>J4</p>  <p>A B</p>	Address bits A15 and A14 are decoded to "0".	✓









Data bus access

Table 5-2: Data bus width

Jumper settings	Function	Settings at delivery
J3 	8-bit data bus access	
J3 	16-bit data bus access on the addresses Base +0, Base +2	✓

Control of the output channels 31 and 32

Table 5-3: Control of the output channels 31 and 32

Jumper settings	Function	Settings at delivery
J1 A  B 	Output 31 is controlled through I/O commands.	✓
J1 A  B 	Output 31 is controlled through Timer 0.	
J2 A  B 	Output 32 is controlled through I/O commands.	✓
J2 A  B 	Output 32 is controlled through Timer 1.	

Selection of the interrupt line to the PC bus

Table 5-4: Selection of the interrupt line

Jumper settings	Function	Settings at delivery
<div><div>J5</div><div><div><div>A</div><div>B</div><div>C</div><div>D</div><div>E</div><div>F</div><div>G</div></div><div><div><div><div></div><div></div><div></div></div><div><div></div><div></div><div></div></div><div><div></div><div></div><div></div></div></div><div><div>H</div><div>I</div><div>J</div><div>K</div><div>L</div><div>M</div><div>N</div></div><div><div>15</div><div>14</div><div>12</div><div>11</div><div>10</div><div>5</div><div>3</div></div></div><div><div>Timer</div><div>Error outputs</div></div></div></div>	No jumper set: no interrupt line is selected	✓
<div><div>J5</div><div><div><div>A</div><div>B</div><div>C</div><div>D</div><div>E</div><div>F</div><div>G</div></div><div><div><div><div></div><div></div><div></div></div><div><div></div><div></div><div></div></div><div><div></div><div></div><div></div></div></div><div><div>H</div><div>I</div><div>J</div><div>K</div><div>L</div><div>M</div><div>N</div></div><div><div>15</div><div>14</div><div>12</div><div>11</div><div>10</div><div>5</div><div>3</div></div></div><div><div>Timer</div><div>Error outputs</div></div></div></div>	<div>Example:</div> <div>J5-A: IRQ15 is set for the timer interrupt</div> <div>J5-N: IRQ3 is set for the errors on the outputs</div>	

5.2 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 to 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 10-pin block of DIP switches S1 and the jumper field J4 .

Decoding the base address

The base address is decoded in steps of each time 16 I/O addresses.

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

In table 5-5 the address 0390H is decoded. (settings at delivery).

Table 5-5: Decoding table

	MSB								LSB							
Decoded address bus	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Hex base address to be set	0				3				9				0			
Binary base address to be set	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0
DIP switches S1 Logic "0"= ON Logic "1" = OFF	–	–	s10	s9	s8	s7	s6	s5	s4	s3	s2	s1	X	X	X	X
			ON	ON	ON	ON	OFF	OFF	OFF	ON	ON	OFF				
Jumper field J4 ON = jumper set OFF = jumper open	J4-B ON	J4-A ON	–	–	–	–	–	–	–	–	–	–	X	X	X	X

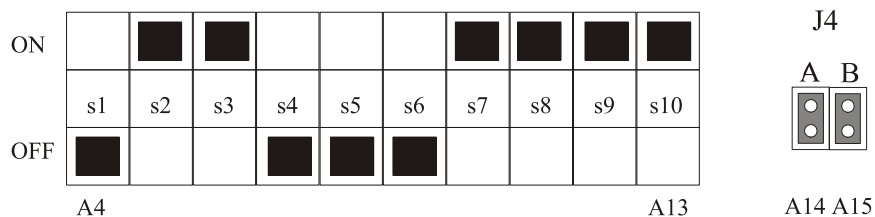
X : Decoded address range of the board (16 I/O addresses)

– : Not selectable through this component

Fig. 5-3: DIP switches S1

IMPORTANT!

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



6 INSTALLATION

i

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.

i

IMPORTANT!

You have installed already **one or more** ADDI-DATA boards in your PC, and you wish to install **an additional** board?

Proceed as if you wished to install one single board.

6.1 Inserting the board

i**IMPORTANT!**

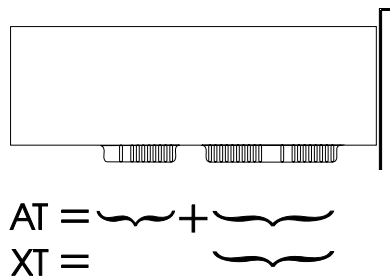
Do observe the safety instructions.

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: Types of slots



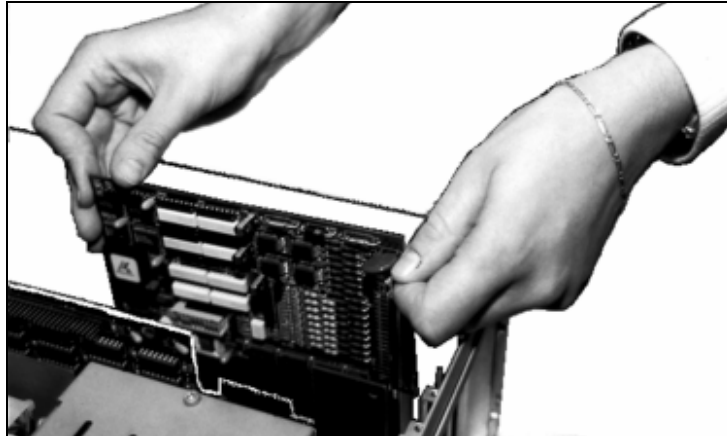
The board can be inserted either in a slot XT or AT. It can also be inserted in EISA slots with respect of certain conditions.

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.**

6.1.2 Plugging the board into the slot

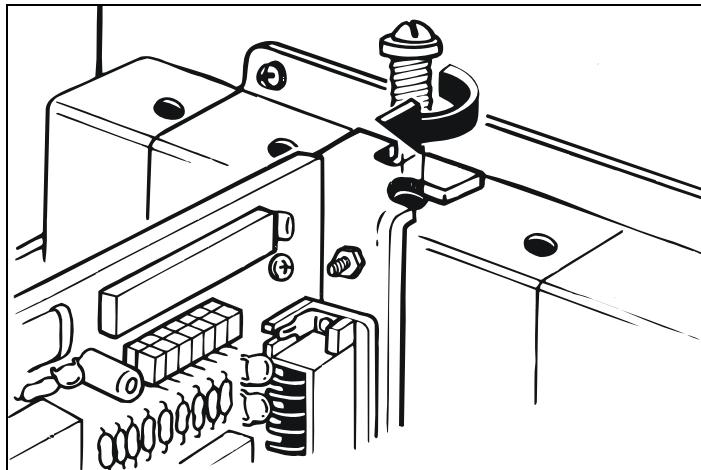
- **Discharge yourself from electrostatic charges**
- Insert the board **vertically** into the chosen slot.

Fig. 6-3: Inserting the board



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

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 PA2000\16bit on a diskette.
If several diskettes are to be used, the directory content is stored in several sub-directories (Disk1, Disk2, Disk3...).
- Insert the (first) diskette into a driver 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 PA2000\32bit\Disk1 corresponding to the board.
- 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 attentively the "Software License" and "Readme".

In "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/ 95/98. The user can registrate 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 registrated 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: ADDIREG registration program

Board name	Base address	Access	PCI bus/slot	Interrupt	ISA DMA	More information

Buttons: Insert, Edit, Clear

Board configuration:

Base address name: [dropdown] Interrupt name: [dropdown] DMA name: [dropdown] [Set] [Cancel]

Base address: [dropdown] Interrupt: [dropdown] DMA channel: [dropdown] [Default] [More information]

Access mode: [dropdown]

Buttons: Save, Restore, Test registration, Deinstall registration, Print registration, Quit

ADDI-DATA logo

Table

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

Board name:

Names of the different registered boards (e.g.: APCI-3120).

When you start the program for the first time, no board is registrated 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 (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**Edit¹:**

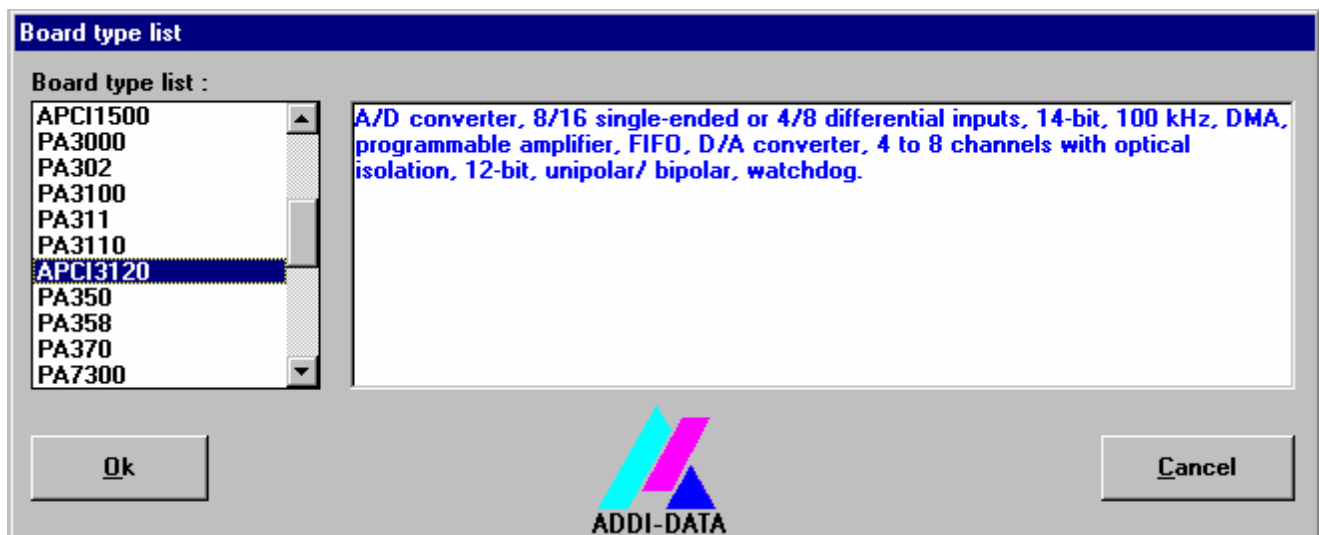
Selection of the highlighted board with the different parameters set in the text boxes.

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



All boards you can registrate 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 registrations of all board 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

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 6-6 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

i

IMPORTANT!

To change the registration of a board, you must have administrator rights. Only an administrator is allowed to registrate 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.

6.3.4 Uninstalling the ADDIREG program

The ADDI_UNINSTALL program is delivered on the CD-ROM.

- Install the ADDI_UNINSTALL program on your computer.
- Start the ADDIREG program and click on "Deinstall registration"
- Quit ADDIREG
- Start the ADDI_UNINSTALL program
- Proceed as indicated until the complete removing of ADDIREG.

You can also download the program from Internet.

6.4 Technical support and software downloads

You can download the latest version of the device driver for the **PA 2200** board from

<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 CONNECTION TO THE PERIPHERAL

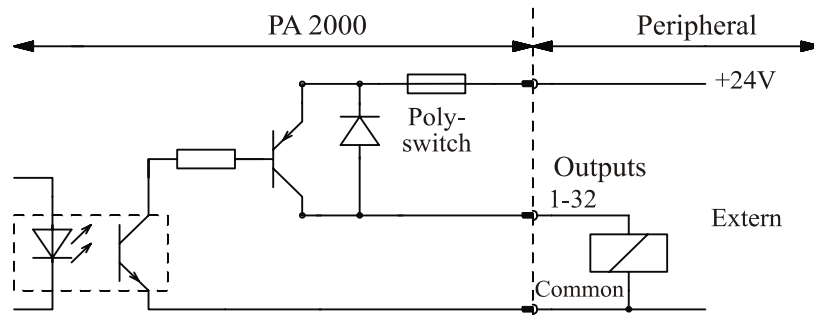
7.1 Connector pin assignment

Fig. 7-1: 37-pin SUB-D male connector

User designation					User designation
	Reserve	19	● ●	37	Dig. output 32
	Dig. output 31	18	● ●	36	Dig. output 30
	Dig. output 29	17	● ●	35	Dig. output 28
	Dig. output 27	16	● ●	34	Dig. output 26
	Dig. output 25	15	● ●	33	Dig. output 24
	Dig. output 23	14	● ●	32	Dig. output 22
	Dig. output 21	13	● ●	31	Dig. output 20
	Dig. output 19	12	● ●	30	Dig. output 18
	Dig. output 17	11	● ●	29	0 V ext.
	0 V ext.	10	● ●	28	24 V ext.
	24 V ext.	9	● ●	27	Dig. output 16
	Dig. output 15	8	● ●	26	Dig. output 14
	Dig. output 13	7	● ●	25	Dig. output 12
	Dig. output 11	6	● ●	24	Dig. output 10
	Dig. output 9	5	● ●	23	Dig. output 8
	Dig. output 7	4	● ●	22	Dig. output 6
	Dig. output 5	3	● ●	21	Dig. output 4
	Dig. output 3	2	● ●	20	Dig. output 2
	Dig. output 1	1	●		

7.2 Connection principle

Fig. 7-2: Connection principle



7.3 Connection examples

Fig. 7-3: Connection examples

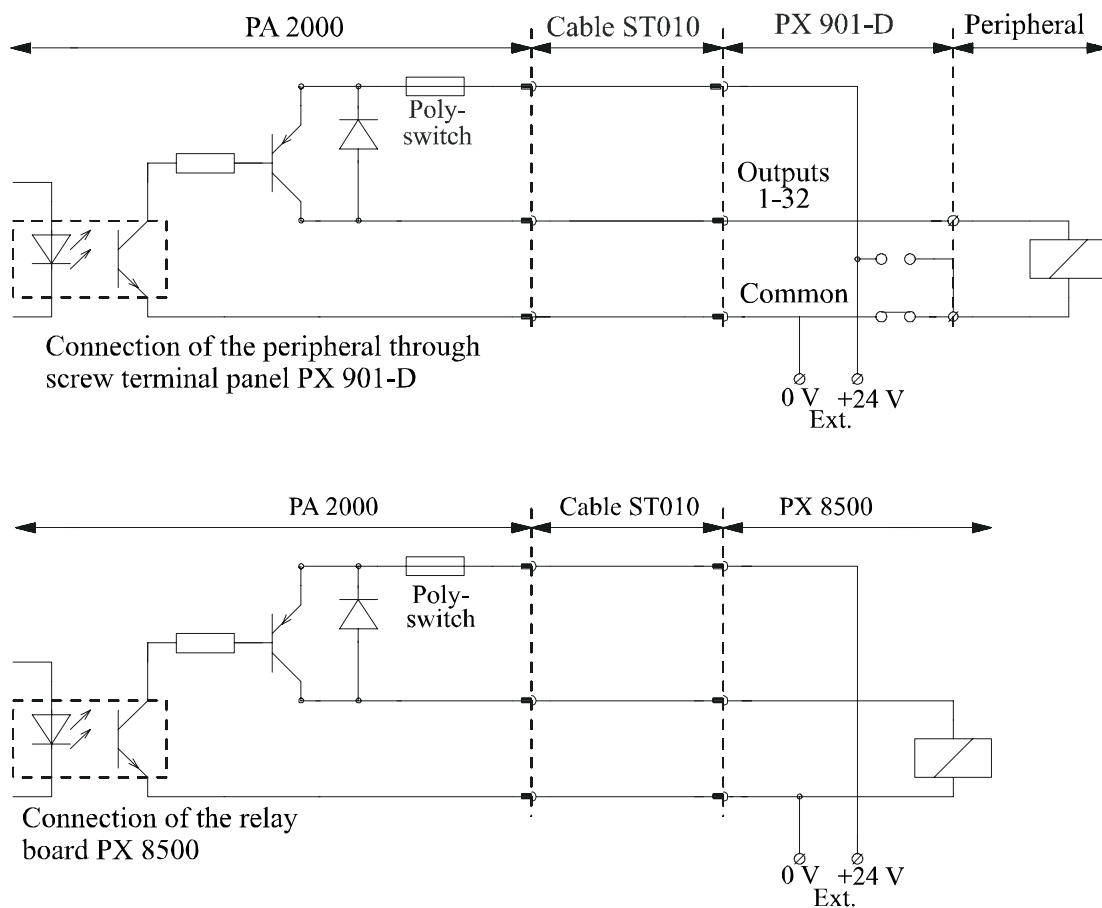
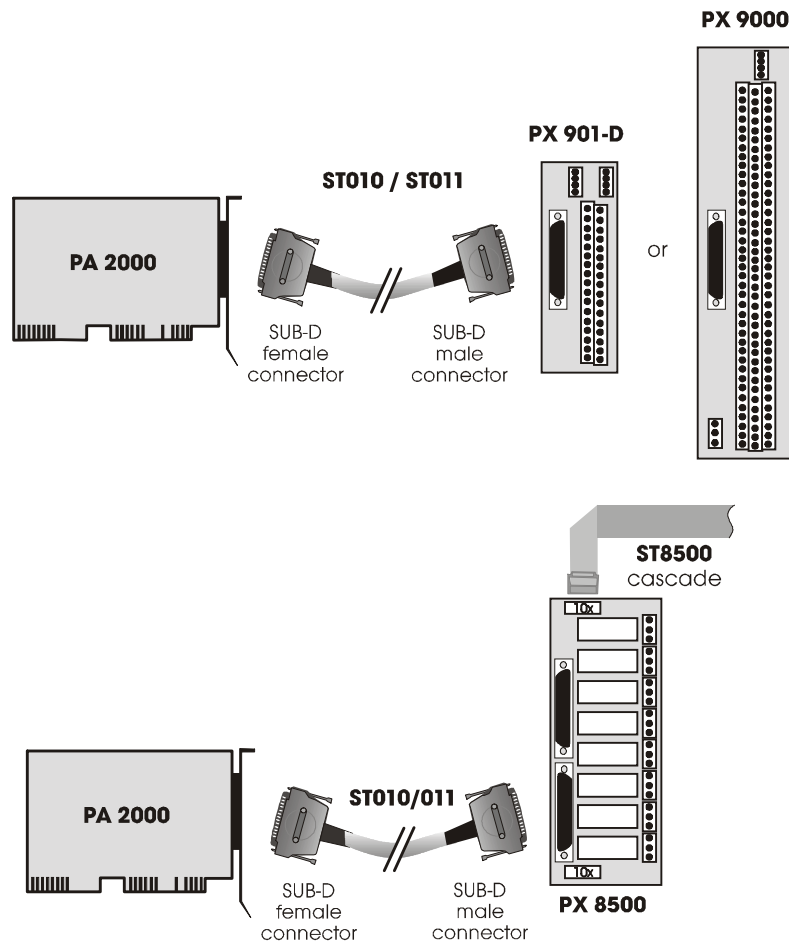


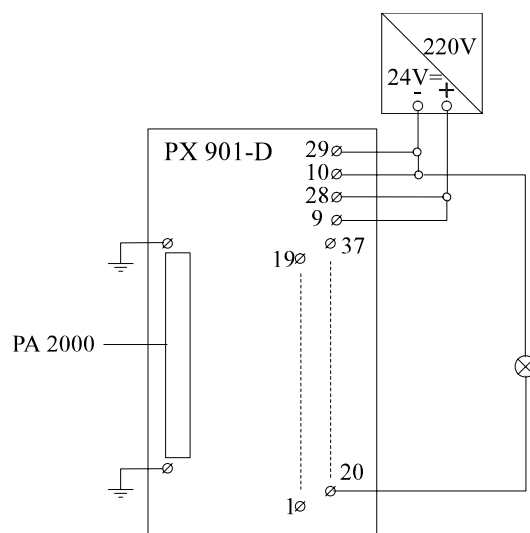
Fig. 7-4: Connection to screw terminal panels and relay boards



Our technical support is at your disposal for further information about our cables and screw terminal panels or relay boards.

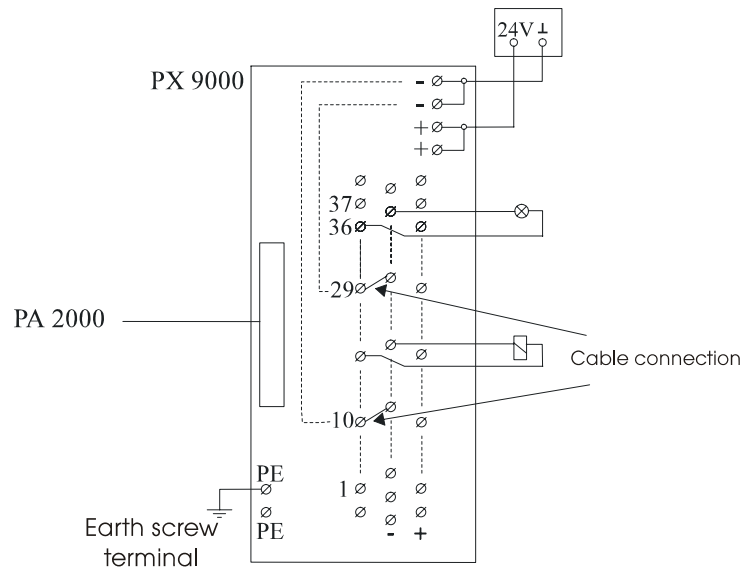
7.3.1 Connection to the external peripheral through a PX 901-D

Fig. 7-5: Connection to the external peripheral through a PX 901-D



7.3.2 Connection to the external peripheral through a PX 9000

Fig. 7-6: Connection to the external peripheral through a PX 9000



8 FUNCTIONS OF THE BOARD

The board is used in the industry for the parallel output of digital 24 V signals. The peripheral and the system are simultaneously isolated.

32 output channels are available on the board.

Technical features:

- Short-circuit proof against ground
- Protection against overtemperature
- On- resistance
- High range of supply voltage
- The outputs are switched off if the supply voltage is < 5 V.

Are also available:

- 3 x 16-bit timers (82C54), programmable through software
- timer2 can be used as a Watchdog for the outputs or as an interrupt generator for the PC

Timer0 and Timer1 can generate square-wave signals with 24 V output level. The base address is set through the 10-pin block of DIP switches and the jumper J4.

Interrupt lines available:

- IRQ3, IRQ5 for XT
- IRQ10, IRQ11, IRQ12, IRQ14, IRQ15 for AT

16-bit data bus convertible in 8-bit data bus.

8.1 I/O mapping

The PA 2000 board requires an address range of 9 I/O addresses within the I/O address range of the PC.

The functionality of the board is accessed with a write or read command on this address range.

The decoding logic is related to the whole 64 KB I/O address range of the PC.

- **Set the base address** (beginning of the address range) with the block of DIP switches S1 and the jumper field J4 in steps of 16 I/O addresses.

- **The data bus width** is determined with jumper J3.
(See jumper settings in chapter 5)

There are therefore **two** I/O maps.

Table 8-1: I/O map for 8-bit access (J3 open)

	I/O Read	I/O Write
Base +0	-	OUTPUT 1-8
Base +1	-	OUTPUT 9-16
Base +2	-	OUTPUT 17-24
Base +3	-	OUTPUT 25-32
Base +4	TIMER0	TIMER0
Base +5	TIMER1	TIMER1
Base +6	TIMER2	TIMER2
Base +7	TIMER_CONTROL_REGISTER	TIMER_CONTROL_REGISTER
Base +8	ERROR_STATUS_REGISTER	COMMAND_STATUS_REGISTER

-: not decoded

Table 8-2: I/O map for 16-bit access (J3 is set)

	I/O Read	I/O Write
Base +0	-	OUTPUT 1-16
Base +2	-	OUTPUT 17-32
Base +4	TIMER0	TIMER0
Base +5	TIMER1	TIMER1
Base +6	TIMER2	TIMER2
Base +7	TIMER_CONTROL_REGISTER	TIMER_CONTROL_REGISTER
Base +8	ERROR_STATUS_REGISTER	COMMAND_STATUS_REGISTER

- : not decoded

8-bit access

The **32 digital outputs** (24 V) are set directly with a write command.

Ex.: Output 1 corresponds to bit D0 of the address **Base +0**

Output 16 corresponds to bit D7 of the address **Base +1**

Output 17 corresponds to bit D0 of the address **Base +2**

Output 32 corresponds to bit D7 of the address **Base +3**

**Table 8-3: Description of the address Base + 0
(8-bit write access for the outputs 1-8)**

D7				D0			
Output 8	Output 7	Output 6	Output 5	Output 4	Output 3	Output 2	Output 1

**Table 8-4: Description of the address Base + 1
(8-bit write access for the outputs 9-16)**

D7				D0			
Output 16	Output 15	Output 14	Output 13	Output 12	Output 11	Output 10	Output 9

**Table 8-5: Description of the address Base + 2
(8-bit write access for the outputs 17-24)**

D7				D0			
Output 24	Output 23	Output 22	Output 21	Output 20	Output 19	Output 18	Output 17

**Table 8-6: Description of the address Base + 3
(8-bit write access for the outputs 25-32)**

D7				D0			
Output 32	Output 31	Output 30	Output 29	Output 28	Output 27	Output 26	Output 25

Addresses Base +4 to Base +7

They are intended to program the functionality's of the timer (82C54).

Ex.: timer (watchdog).

**Table 8-7: Description of the address Base + 8
(8-bit write access for the COMMAND_STATUS_REGISTER)**

D7				D0			
-	-	-	-	GATE_1	GATE_0	INTER_EN	WATCH_EN

GATE_1:	if "0"	TIMER1 disabled (state after reset)
	if "1"	TIMER1 active
GATE_0:	if "0"	TIMER0 disabled (state after reset)
	if "1"	TIMER0 active
INTER_EN:	if "0"	Interrupts to the PC bus are disabled (state after reset)
	if "1"	Interrupts to the PC are enabled
WATCH_EN:	if "0"	Watchdog not active (state after reset)
	if "1"	Watchdog active

**Table 8-8: Description of the address Base + 8
(8-bit write access for the ERROR_STATUS_REGISTER)**

D7				D0			
-	-	-	-	"0"	WATCH	ERR_2	ERR_1

D3 **firmly decoded on "0"**

WATCH:	if "0"	Watchdog has run down Only relevant if the watchdog is active
	if "1"	Watchdog has not run down Is only relevant if the watchdog is active.
ERR_2:	if "0"	External voltage supply is OK
	if "1"	External voltage supply is not OK
ERR_1:	if "0"	No short-circuit or overtemperature has happened on the outputs
	if "1"	Short-circuit or overtemperature has happened on the outputs

On these addresses the accesses occur **only in 8 bit data bus width** (independently from jumper J3).

16-bit access

The 32 digital outputs (24V) are set directly with a write command.

**Table 8-9: Description of the address Base + 0
(16-bit write access for the outputs 1-16)**

D15	D14					D1	D0
Output 16	Output 15	-----	-----	-----	-----	Output 2	Output 1

Ex.: Output 1 corresponds to bit D0 of the address **Base +0**
Output 16 corresponds to bit D15 of the address **Base +0**

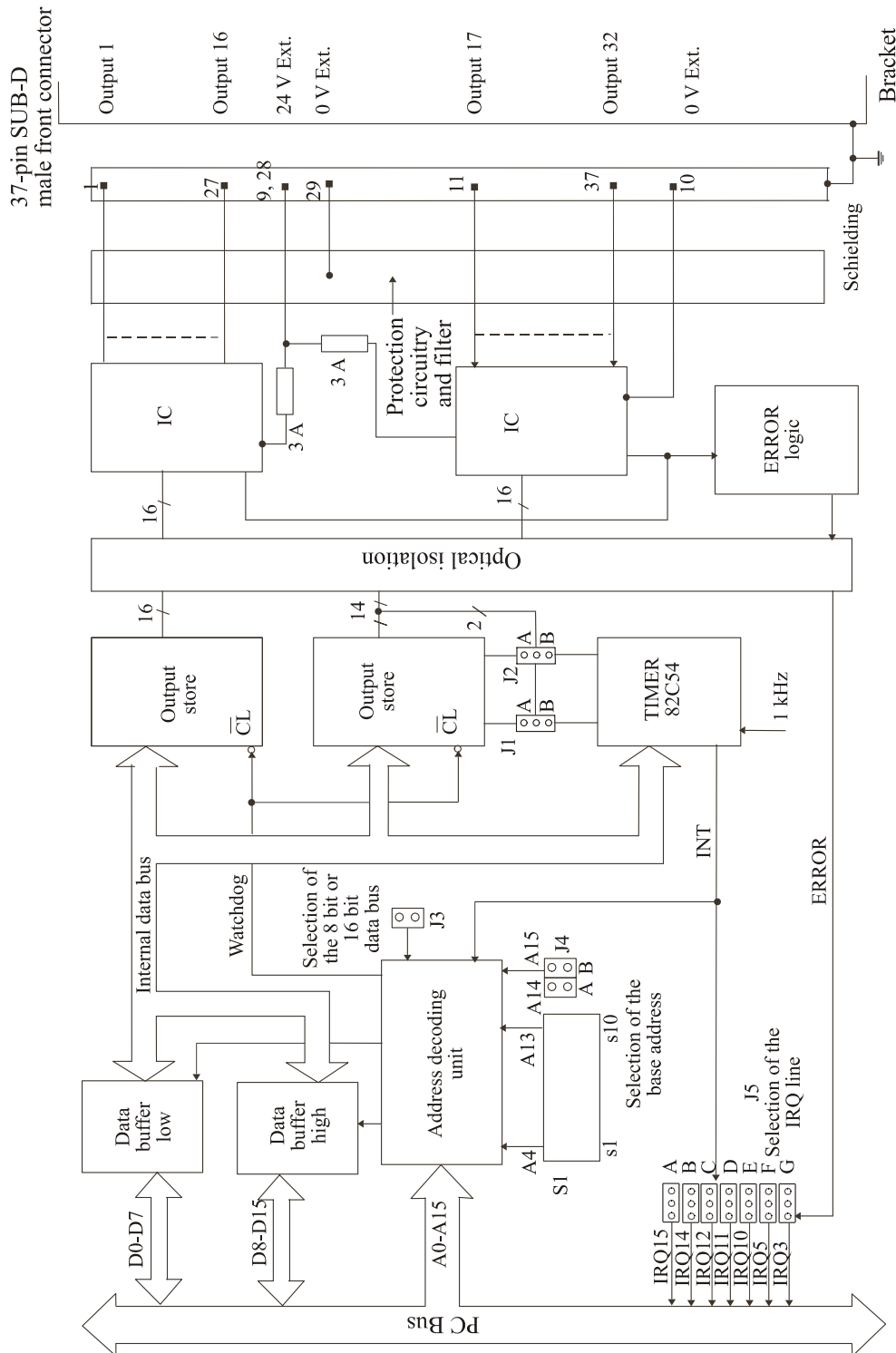
**Table 8-10: Description of the address Base + 2
(16-bit write access for the outputs 1-16)**

D15	D14					D1	D0
Output 16	Output 15	-----	-----	-----	-----	Output 2	Output 1

Ex: Output 17 corresponds to bit D0 of the address **Base +2**
Output 32 corresponds to bit D15 of the address **Base +2**

8.2 Block diagram

Fig. 8-1: Block diagram of the PA 2000



8.3 Digital outputs

The PA 2000 board is provided with 32 optoisolated outputs.
The outputs comply with the 24 V industry standard (IEC1131-2)

Positive logic is used

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

The output channels (switches) switch the **+24V Ext.** outside to the load.
One end of the load is connected with the 0V EXT ground.
The outputs have two common ground lines:
0V EXT (outputs) at the 37-pin SUB-D male connector.



WARNING!

If you operate all the outputs with the same voltage supply,
the voltage supply must deliver at least the power which is necessary for
your application.

The maximum supply voltage is 36 V.
A current of 500 mA can be switched for each output.

The current is limited for the outputs 1-16 on approx. 3 A through a self
resetting fuse.

The current is limited for the outputs 17-32 on approx. 3 A through a self
resetting fuse.

If all 32 outputs are active, only a current of approx. 200 mA/output can flow.

8.3.1 Features of the outputs

- Short-circuit proof against ground
the output is switched off.
- Protection against overtemperature: the IC is switched off
i.e. each time four outputs: 1 to 4, 5 to 8, 9 to 12, 13 to 16 ...
- The outputs are switched off if the supply voltage is < 5 V.
- Report for diagnostic (voltage drop, short-circuit, overtemperature)

Transorb diodes, LC filters and optical couplers filter the noise coming from the
peripheral side to the system bus side. The effects of inductive and capacitive
noise are thus reduced.

Possible noise emissions are also reduced by LC filters.

Except for the timer function, the board requires no initialisation to output the
24 V digital information. You can program the output channels immediately
after successful power ON reset.

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

8.3.2 Setting the outputs 1 to 32

Four addresses are available in the I/O address range (8-bit data bus access):

- **Base +0** for the output channels 1 to 8,
- **Base +1** for the output channels 9 to 16,
- **Base +2** for the output channels 17 to 24,
- **Base +3** for the output channels 25 to 32.

Example with the DEBUG Program under DOS

```
C:> DEBUG (CR)
- o 390, 01 (CR)      (* Set output 1 *)
- o 391, 80 (CR)      (* Set output 16 *)
- o 392, 01 (CR)      (* Set output 17 *)
- o 393, 80 (CR)      (* Set output 32 *)
- q                  (* Quit the DEBUG program *)
```

Example with Basic Program under DOS

```
OUT &H390, 1          (* Set output 1 *)
OUT &H391, &H80        (* Set output 16 *)
```

If you selected the 16-bit data bus access (Jumper J3 is set), **two** addresses are available in the I/O address range:

Base +0 for the output channels 1-16,

Base +2 for the output channels 17-32.

Example in Assembler

```
.....
MOV DX, 390      ; Set the base address on 0390H
MOV AX, 0FFFFH
OUT DX, AX       ; Set the outputs 1 to 16
.....
```

Example in Pascal

```
Procedure Set_Outputs( w_Value: WORD);
```

```
begin
```

```
(* set the outputs in 16-bit data bus width w_Base := $390 *)
```

```
    portw[ w_Base + 2] := w_Value;
```

```
end;
```

8.3.3 Special functions

Different diagnostic bits are set:

- if a short-circuit has happened on an output channel,
- if a component has overtemperature
- or if the external voltage supply drops under 5V

This error data is available through an interrupt routine.

See API functions: `i_PA2000_SetBoardIntRoutine`,
`i_PA2000_ResetBoardIntRoutine`.

Output channels 31 and 32 as generators of square wave signals

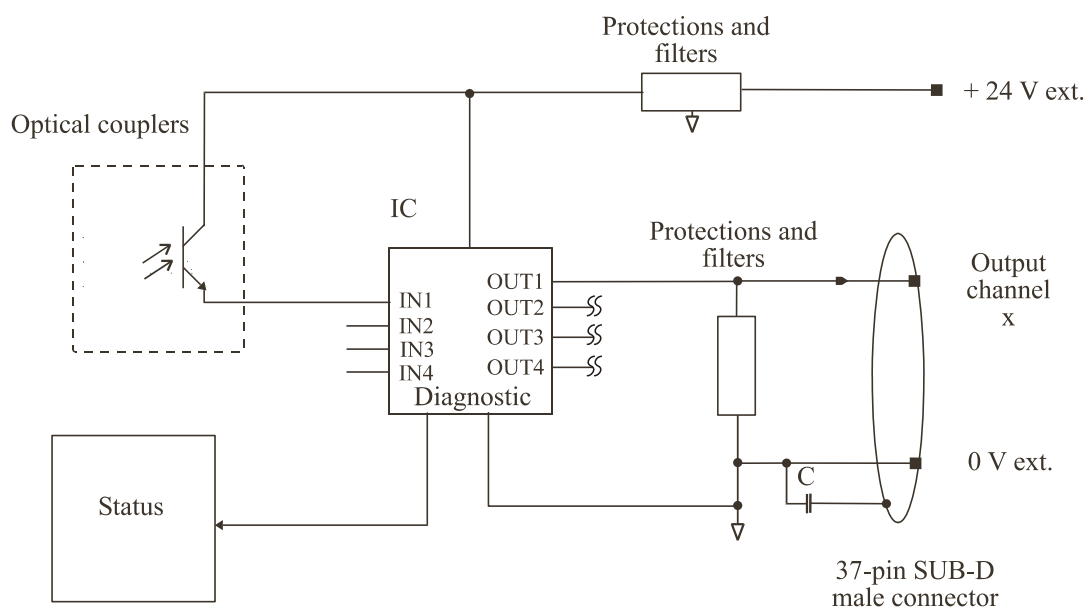
You can control these output channels as follows:

Output 31: set/reset through TIMER0 or I/O commands

Output 32: set/reset through TIMER1 or I/O commands

The selection occurs through jumper J1 and J2.

Fig. 8-2: Output circuitry



x = 1 to 32 Number of the output channel

Timer

- Maximum output frequency through 24 V outputs: 500 Hz, 24 V level
- Minimum output frequency through 24 V outputs: 0,015 Hz, 24 V level

8.4 Interrupt

The board **PA 2000** has two interrupt sources. They must be connected through the jumper field J5 to an interrupt line of the PC bus.

The following interrupt lines are available:

IRQ3, IRQ5, IRQ10, IRQ11, IRQ12, IRQ14, IRQ15.

You must enable the interrupts first

through bit INT-EN of the COMMAND_STATUS_REGISTER.

Possible interrupt sources :

Timer:

- the watchdog time has run down. The outputs are reset
- if Timer2 is used as an interrupt generator

Outputs:

- voltage error: the external voltage supply has dropped below 5V,
- short-circuit error,
- overtemperature error.

The interrupt source information are at the user program's disposal through an interrupt routine

See API functions: i_PA2000_SetBoardIntRoutine,
i_PA2000_ResetBoardIntRoutine

8.5 Timer

Three 16-bit timers (82C54) are available on the PA 2000 board.
Each timer can be programmed through software.

The input frequency of the three timers is 1 kHz (1 ms).

TIMER0 and TIMER1 are used for controlling the outputs 31 and 32.
They are readable and can thus be used for software timings as well.

Enabling TIMER0 and TIMER1

- TIMER0 through the bit **GATE_0**,
- TIMER1 through the bit **GATE_1**
of the COMMAND_STATUS_REGISTER (**Base +8**).

Special function of TIMER2: watchdog timer

The function **watchdog timer** allows to supervise the software or the PC.

Programmable watchdog time interval: 2 ms to 65,54 s

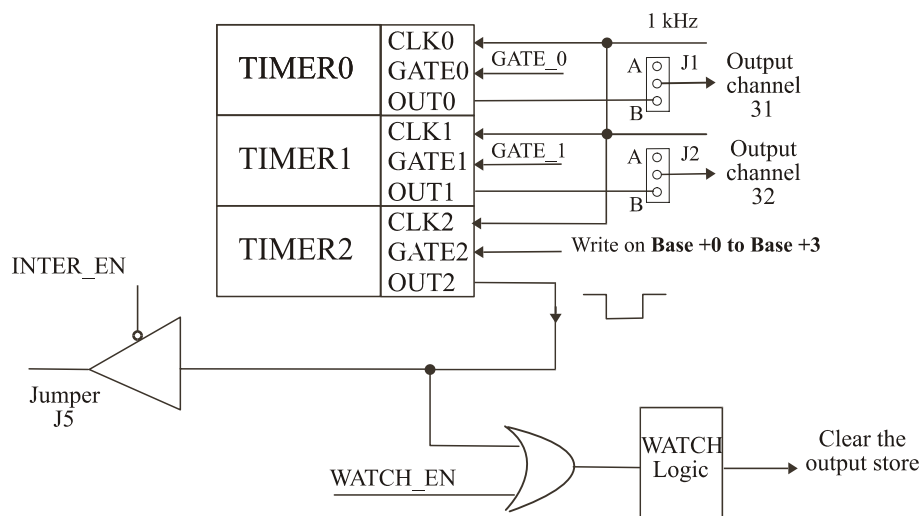
The principle is the following:

- the TIMER2 is programmed as a "hardware triggered strobe" that is mode 5 of component 82C54.
- the programmed time value determines the time interval while the outputs must be accessed again.
- the watchdog circuit is enabled by the bit **WATCH_EN** of the **COMMAND_STATUS_REGISTER (Base +8)**.
- the timer is started when it is written for the first time on the outputs.
- if it is not written within the programmed time on **Base +0 to Base +3**, the outputs are reset (switch on OFF). An interrupt can then be triggered.

The user software must be set up so that the accesses always occur on the addresses **Base +0 to Base +3** during the programmed watchdog time. You thus avoid that the watchdog time runs down.

TIMER2 can also generate periodically an interrupt request to the PC bus (as a "rate generator").

Fig. 8-3: Timer circuitry

**Data**

Programmable watchdog time interval: 2 ms up to 65,54 s.

9 STANDARD SOFTWARE

9.1 Introduction

i

IMPORTANT!

Note the following conventions in the text:

Function: "i_PA2000_SetBoardAddress"

Variable *ui_Address*

Table 9-1: 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 9-2: Type Declaration for Windows 95/98/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 9-3: Define value


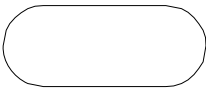
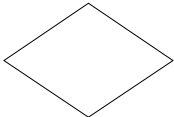

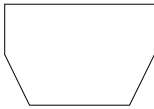
Define name	Decimal value	Hexadecimal value
DLL_COMPILER_C	0	0
DLL_COMPILER_VB	1	1
DLL_COMPILER_PASCAL	2	2
DLL_LABVIEW	3	3
DLL_COMPILER_VB_5	4	4
PA2000_DISABLE	0	0
PA2000_ENABLE	1	1
PA2000_8BIT	8	8
PA2000_16BIT	16	10
PA2000_TIMER	52	34
PA2000_WATCHDOG	58	3A
PA2000_RATE_GENERATOR	52	34
PA2000_BAUD_RATE_GENERATOR	54	36
PA2000_SELECT_TIMER_0	0	0
PA2000_SELECT_TIMER_1	64	40
PA2000_SELECT_TIMER_2	128	80
PA2000_MIN_GENERATOR_VALUE	16	10
PA2000_MAX_GENERATOR_VALUE	500000	10
PA2000_FREQUENCY_MIN	2	2
PA2000_FREQUENCY_MAX	65535	FFFF

9.2 DIN 66001- Graphical symbols

This chapter describes all software functions (API) necessary for the operation of the **PA 2000** board.

To illustrate these functions, we designed flow charts with the graphical symbols listed below.

It gives the user a quick overview of the hierarchy between the different functions.

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

9.3 Address

1) i_PA2000_SetBoardAddress (...)

Syntax:

```
<Return value> = i_PA2000_SetBoardAddress (UINT_ ui_Address,
                                             BYTE_ b_AccessMode,
                                             PBYTE_ pb_BoardHandle)
```

Parameter:

- Input:

UINT	ui_Address	Base address of the PA 2000
BYTE	b_AccessMode	Data bus width
		PA2000_8BIT: 8-bit access
		PA2000_16BIT: 16-bit access

- Output:

PBYTE	pb_BoardHandle	Handle ¹⁾ of the PA 2000 to use the functions of the board
-------	----------------	--

Task:

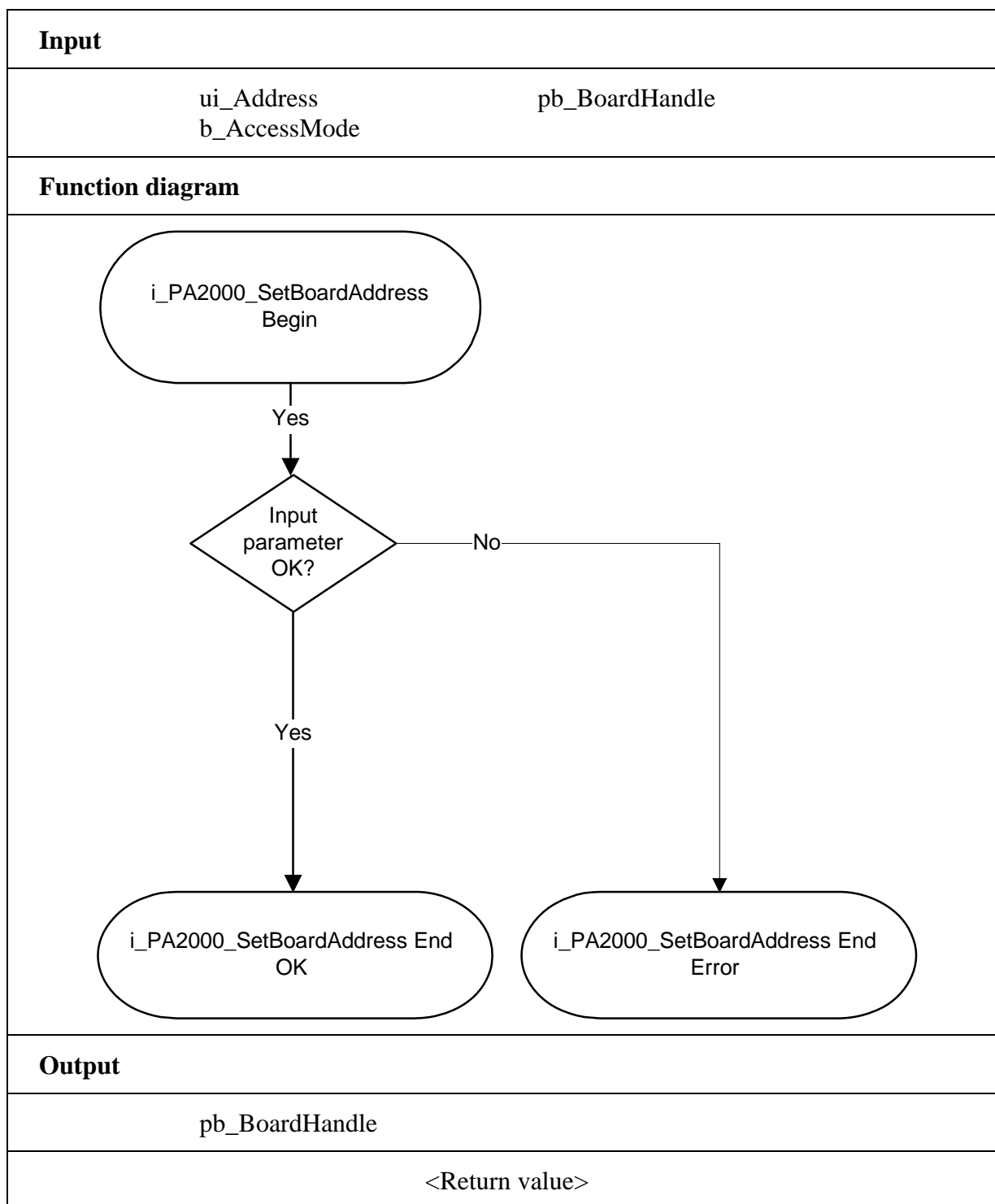
Stores the base address.

A handle is returned to the user which allows him/her to use the following functions. With handles several boards can be operated.

Return value:

- 0: No error
- 1: Base address already used
- 2: No handle is available for the board (up to 10 handles can be used)
- 3: Data bus width is wrong. Another value as PA2000_8BIT or PA2000_16BIT was passed.

¹ Identification number of the board



2) i_PA2000_SetBoardIntRoutine (..)

Syntax:

```
<Return value> = i_PA2000_SetBoardIntRoutine
                                (BYTE_      b_BoardHandle,
                                 BYTE_      b_ErrorInterruptNbr,
                                 BYTE_      b_WatchdogTimerInterruptNbr,
                                 VOID_      v_FunctionName
                                (BYTE_ b_BoardHandle,
                                 BYTE_ b_InterruptMask))
```

Parameter:

- Input:

BYTE_	b_BoardHandle	Handle of the PA 2000
BYTE_	b_ErrorInterruptNbr	Interrupt line of the board for the error management (IRQ3, 5, 10, 11, 12, 14 or 15). When 0, no interrupt line is used.
BYTE_	b_TimerWatchdogInterruptNbr	Interrupt line for the timer/watchdog (IRQ3, 5, 10, 11, 12, 14 or 15). When 0, no interrupt line is used.
VOID_	v_FunctionName	Name of the user interrupt routine.

i

IMPORTANT!

b_ErrorInterruptNbr and *b_TimerWatchdogInterruptNbr* cannot have the same value. The two functions / variables cannot be set to 0 at the same time.

Task:

The function can be called up several times.

First callup (first board):

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

If you operate several **PA 2000** boards which have to react to interrupts, call up the function as often as you operate **PA 2000** boards.

The variable *v_FunctionName* is only relevant **for the first callup**.

From the second callup 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 timer has run down
- the watchdog has run down
- an error has occurred on the outputs

The following errors are possible

- overtemperature
- short-circuit
- no voltage is available

To control easily the interrupt management, please use the function "i_PA2000_SetBoardInterruptRoutine"

The user interrupt routine must have the following syntax:

```
VOID_ v_BenutzerRoutineName (BYTE_ b_BoardHandle,
                             BYTE_ b_InterruptMask)
v_BenutzerRoutineName      Name of the user interrupt routine
b_BoardHandle              Number of the PA 2000 handle which
                             has generated the interrupt
b_InterruptMask            Mask of the events which have
                             generated the interrupt
```

Mask	Meaning
0000 0001	Voltage error
0000 0010	Short-circuit error
0000 0100	Watchdog has run down
0000 1000	Timer has run down

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

i

IMPORTANT!

If you use Visual Basic DOS, the following parameter is not available.

```
VOID_ v_FunctionName (BYTE_ b_BoardHandle,
                      BYTE_ b_InterruptMask)
```

Controlling the interrupt management

Please use the following functions

"ON UEVENT GOSUB xxxxxxxxx" of Visual Basic Dos

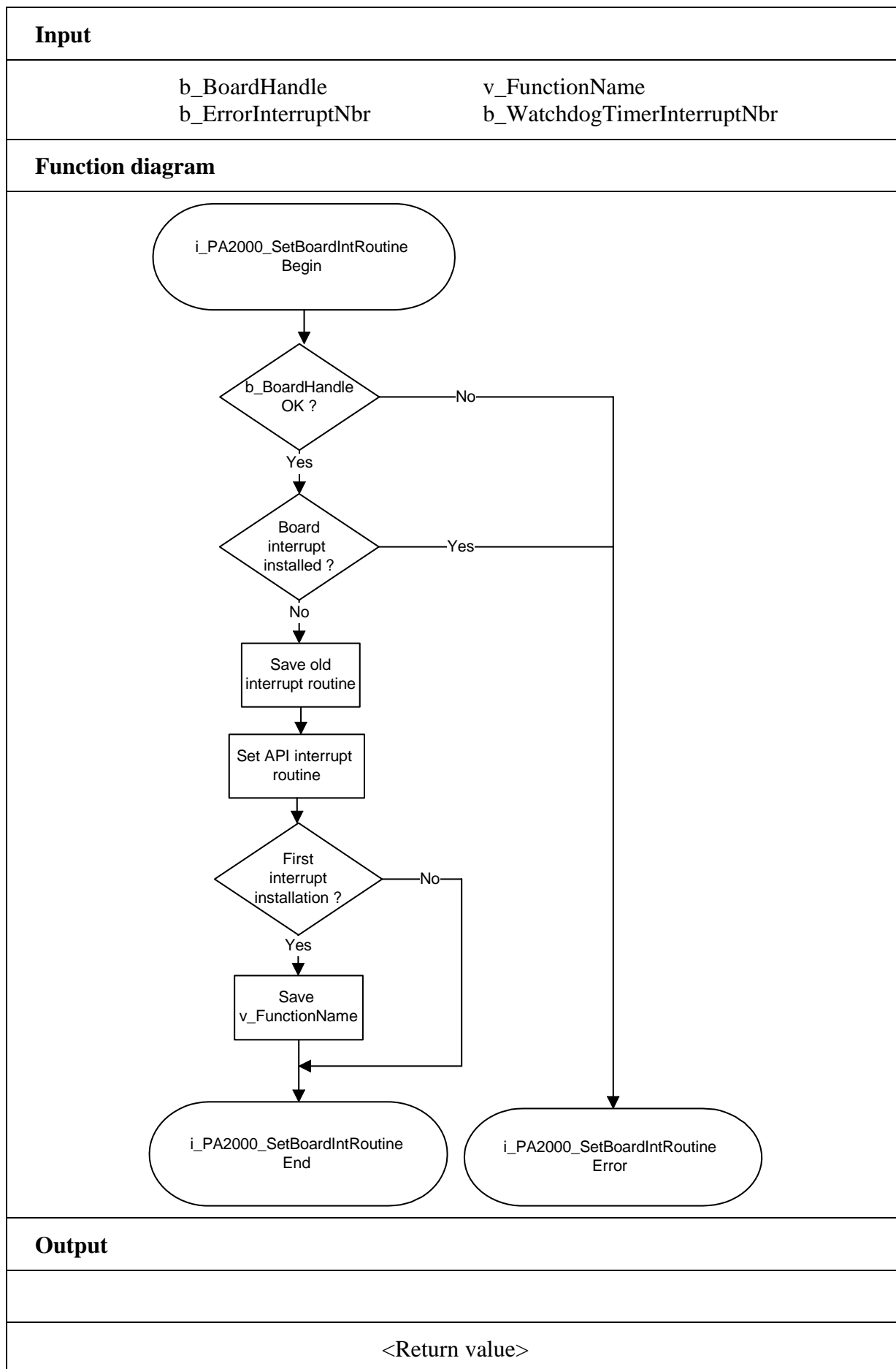
and

"i_PA2000_TestInterrupt"

This function tests the interrupt of the **PA 2000**. It is used to obtain the values of *b_BoardHandle*, *_InterruptMask*.

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong
- 2: Interrupt line already used
- 3: Interrupt line not available
- 4: Both interrupt numbers cannot be the same
- 5: Both interrupt numbers cannot be set to 0 at the same time



3) i_PA2000_InitCompiler (..)

Syntax:

<Return value> = i_PA2000_InitCompiler (BYTE b_CompilerDefine)

Parameter:**- 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 5.0.

- 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, parameter in what language you want to program. This function must be the first to be called up.

i

IMPORTANT!

This function is only available under Windows.

Calling convention:

ANSI C :

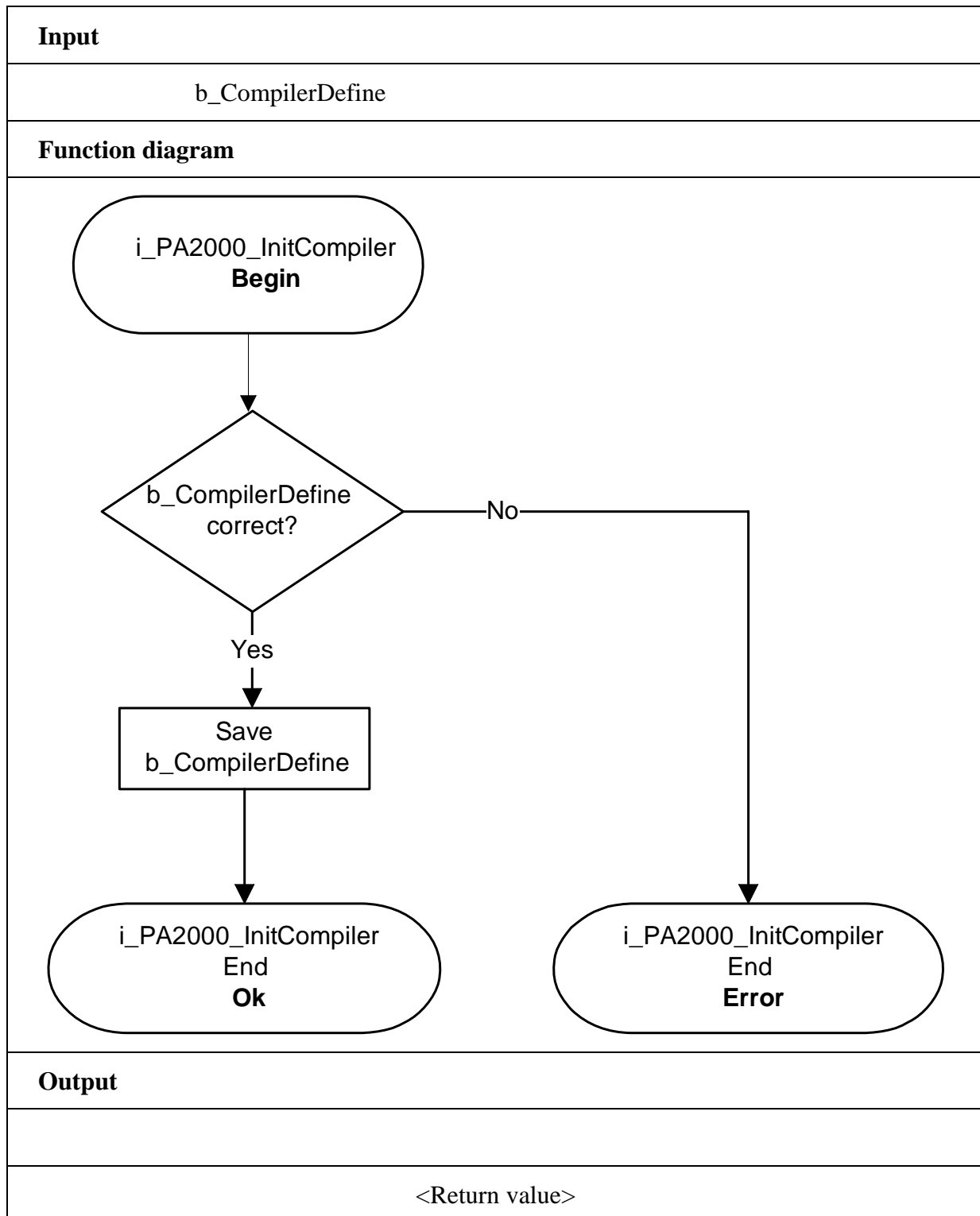
int i_ReturnValue;

i_ReturnValue = i_PA2000_InitCompiler (DLL_COMPILER_C);

Return value:

0: No error

-1: Compiler parameter is wrong



i**IMPORTANT!**

This function is only available for DOS and Windows 3.11 applications

4) i_PA2000_SetBoardInformation (..)**Syntax:**

<Return value> = i_PA2000_SetBoardInformation (UINT ui_BaseAddress,
 BYTE b_AccessMode,
 BYTE b_InterruptNbr,
 PBYTE pb_BoardHandle)

Parameter:**- Input:**

UINT	ui_BaseAddress	Base address of the PA 2000 board
BYTE	b_AccessMode	Hardware access mode PA2000_8BIT: 8-bit access mode PA2000_16BIT: 16-bit access mode
BYTE	b_InterruptNbr	Interrupt number (3, 5, 10, 11, 12, 14 or 15). If 0, no interrupt is used

- Output:

PBYTE	pb_BoardHandle	Handle ¹ of the PA 2000 board for using the functions
-------	----------------	--

Task:

Verifies if a **PA 2000** board is present. Stores the following information:

- the base address,
- the hardware access mode,
- the interrupt number.

A handle is returned which allows the user to use the next functions.

Handles allow the operation of several boards.

Calling convention:ANSI C :

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

```
i_ReturnValue = i_PA2000_SetBoardInformation (0x300,  

  PA2000_16BIT,  

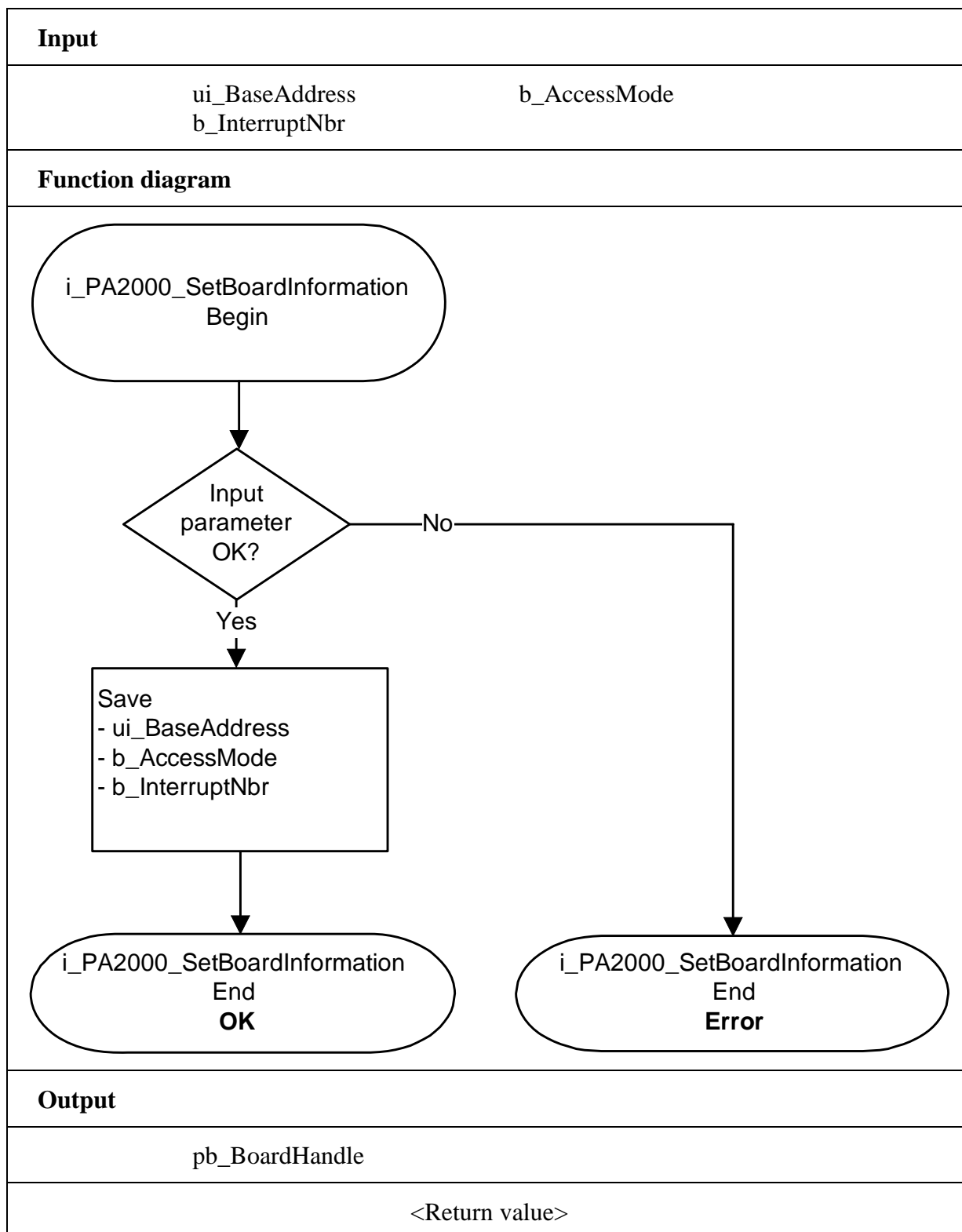
  0,  

  &b_BoardHandle);
```

¹ Identification number of the board

Return value:

- 0: No error
- 1: Board not present
- 2: Access mode parameter is wrong
- 3: Interrupt number is wrong or already used by another **PA 2000**
- 4: No handle is available for the board (up to 10 handles can be used)



i**IMPORTANT!**

This function is only available for Windows NT/95/98 applications.

5) i_PA2000_SetBoardInformationWin32 (...)**Syntax:**

```
<Return value> = i_PA2000_SetBoardInformationWin32
                                (PCHAR      pc_Identifier
                                PBYTE       pb_BoardHandle)
```

Parameter:**- Input:**

PCHAR	pc_Identifier	Identifier string for the selection of PA 2000 The identifier string is determined by the ADDIREG registration program.
-------	---------------	---

- Output:

PBYTE	pb_BoardHandle	Handle of the PA 2000 board for using the functions
-------	----------------	---

Task:

Call up all hardware information about the **PA 2000** given by the ADDIREG registration program and store the following information:

- the base address,
- the interrupt number.

Then verify if the PA 2000 board is present.

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

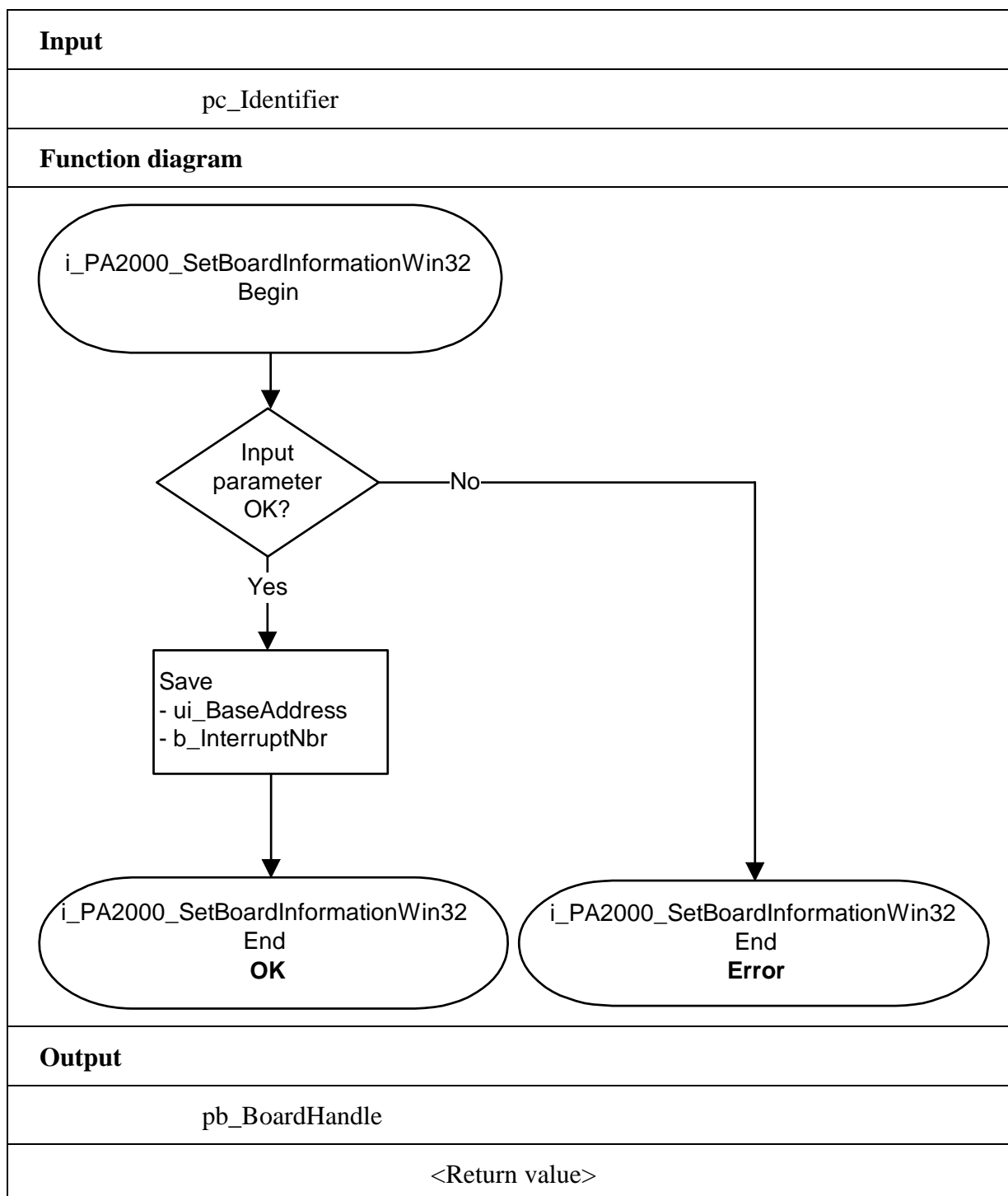
Calling convention:ANSI C :

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

```
i_ReturnValue = i_PA2000_SetBoardInformationWin32
                                ("PA2000-00", &b_BoardHandle);
```

Return value:

- 0: No error
- 1: Board not present
- 2: No handle is available for the board (up to 10 handles can be used)
- 3: Error when opening the driver under Windows NT/95/98



6) i_PA2000_GetHardwareInformation (...)**Syntax:**

```
<Return value> = i_PA2000_GetHardwareInformation
                    (BYTE   b_BoardHandle,
                     PUINT  pui_BaseAddress,
                     PBYTE  pb_InterruptNbr)
```

Parameter:**- Input:**

BYTE	b_BoardHandle	Handle of the PA 2000 board
------	---------------	-----------------------------

- Output:

PUINT	pui_BaseAddress	PA 2000 base address
PBYTE	pb_InterruptNbr	PA 2000 interrupt line.

Task:

Returns the base address, the interrupt and DMA number of the **PA 2000**.

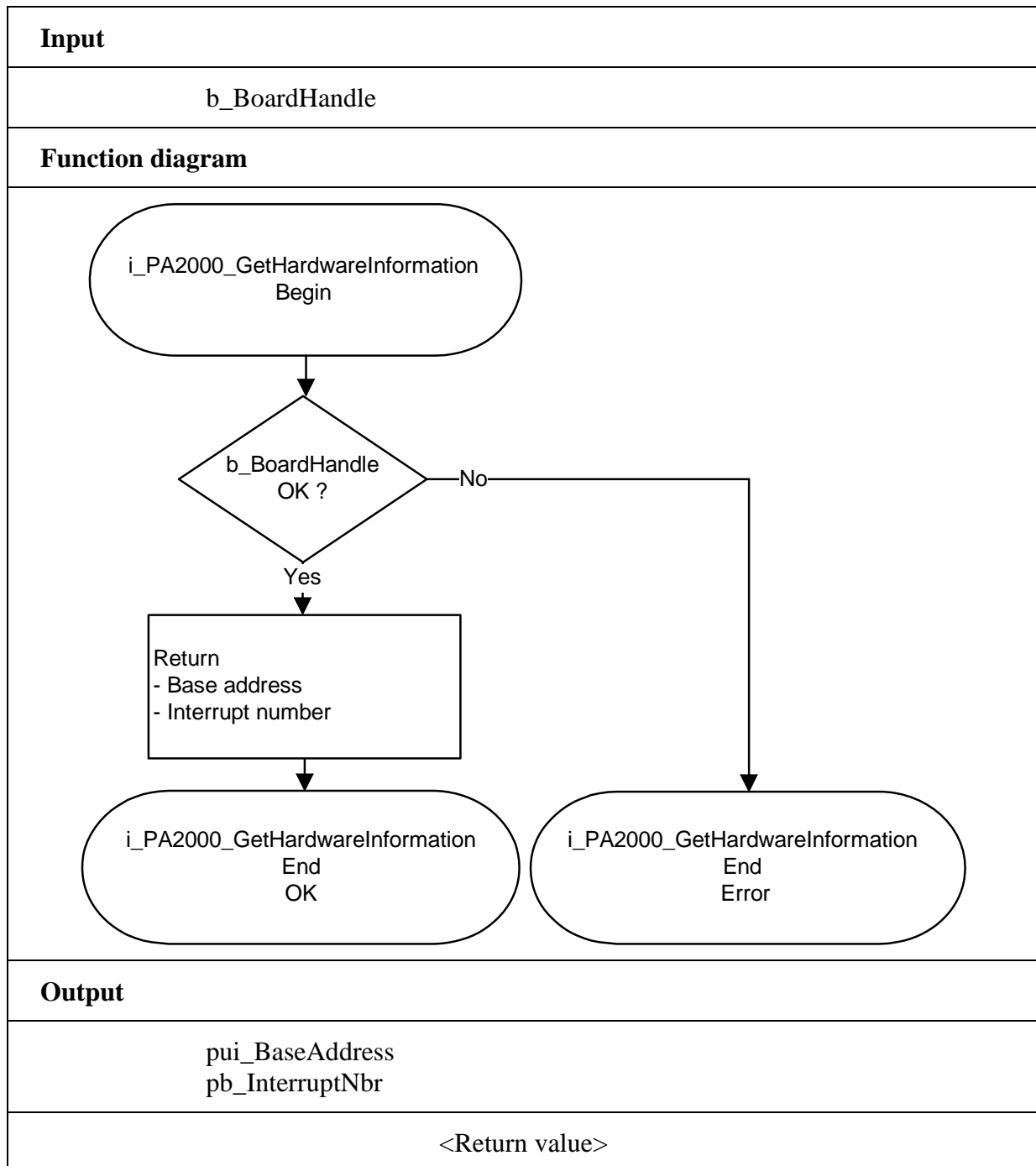
Calling convention:ANSI C:

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

```
i_ReturnValue = i_PA2000_GetHardwareInformation (b_BoardHandle,
                                                  &ui_BaseAddress,
                                                  &b_InterruptNbr);
```

Return value:

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



7) i_PA2000_CloseBoardHandle

Syntax:

<Return value> = i_PA2000_CloseBoardHandle (BYTE b_BoardHandle)

Parameter:**- Input:**

BYTE b_BoardHandle Handle of the **PA 2000** board

- Output:

No output signal has occurred.

Task:

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

Calling convention:ANSI C :

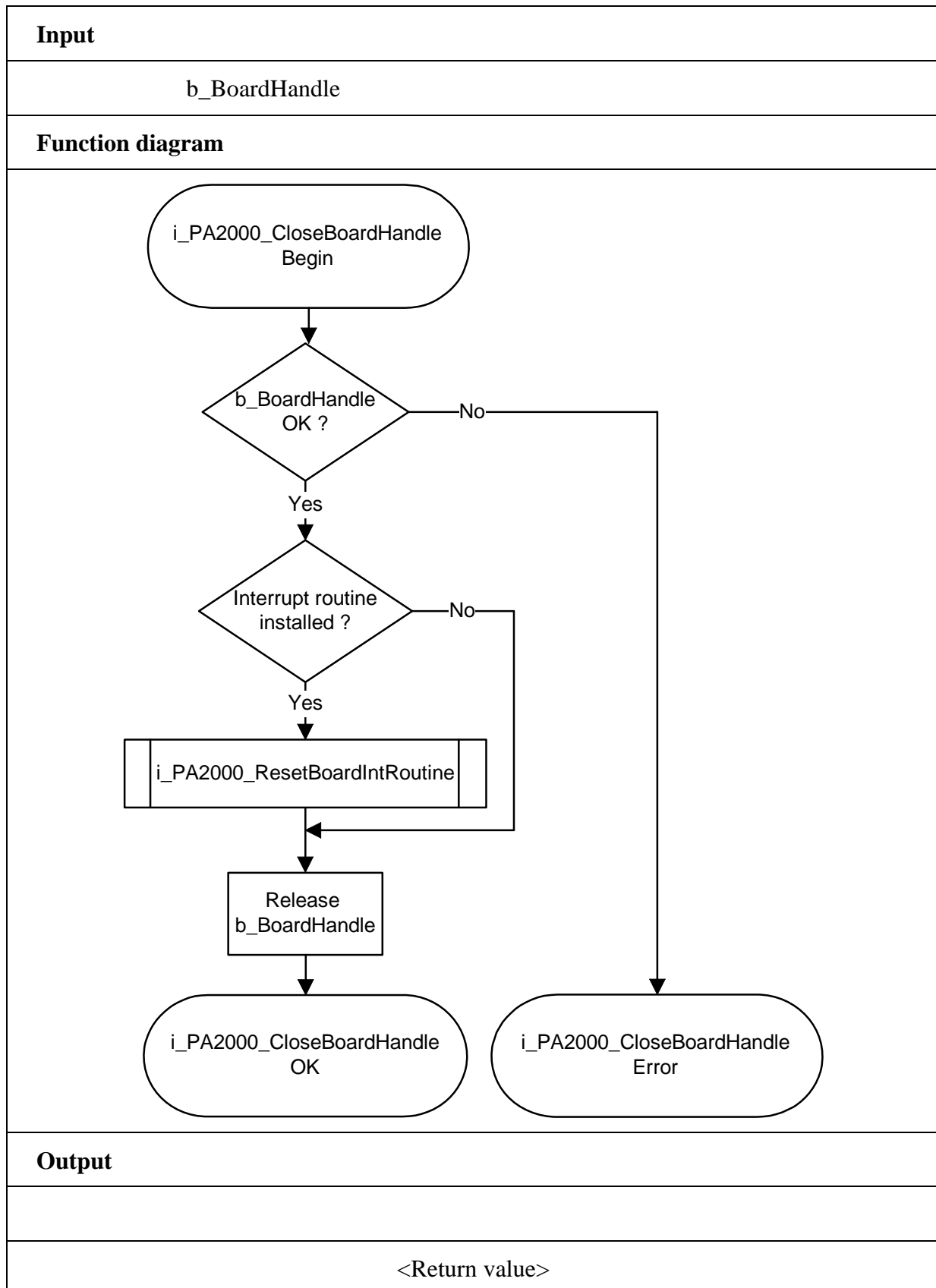
```
int          i_ReturnValue;  
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_PA2000_CloseBoardHandle (b_BoardHandle);
```

Return value:

0: No error

-1: The handle parameter of the board is wrong



9.4 Interrupt

i

IMPORTANT!

This function is only available for Windows NT/95/98 applications.

1) i_PA2000_SetBoardIntRoutineDos (..)

Syntax:

```
<Return value> = i_PA2000_SetBoardIntRoutineDos
                                (BYTE  b_BoardHandle
                                VOID   v_FunctionName
                                (BYTE  b_BoardHandle,
                                BYTE  b_InterruptMask,
                                BYTE  b_InputChannelNbr))
```

Parameter:

- Input:

BYTE	b_BoardHandle	Handle of the PA 2000 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 2000** 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 2000** boards which have to react to interrupts, call up the function as often as you operate **PA 2000** 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 counter/timer has run down
- an event is generated

You can make the interrupt management easier with the function
„i_PA2000_SetBoardIntRoutineDos“

The user interrupt routine must have the following syntax:

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

v_FunctionName Name of the user interrupt routine
b_BoardHandle Handle of the **PA 2000** which has generated the interrupt
b_InterruptMask Mask of the events which have generated the interrupt.
b_InputChannelNbr If an interrupt is generated with a Mask 0000 0001 this variable gives the number of the inputs which have generated the interrupt.

Table 9-4: Interrupt mask

Mask	Meaning
0000 0001	Event 1 has occurred
0000 0010	Event 2 has occurred
0000 0100	Counter/timer 1 has run down
0000 1000	Counter/timer 2 has run down
0001 0000	Counter/timer 3 has run down

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

Calling convention:

ANSI C :

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

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

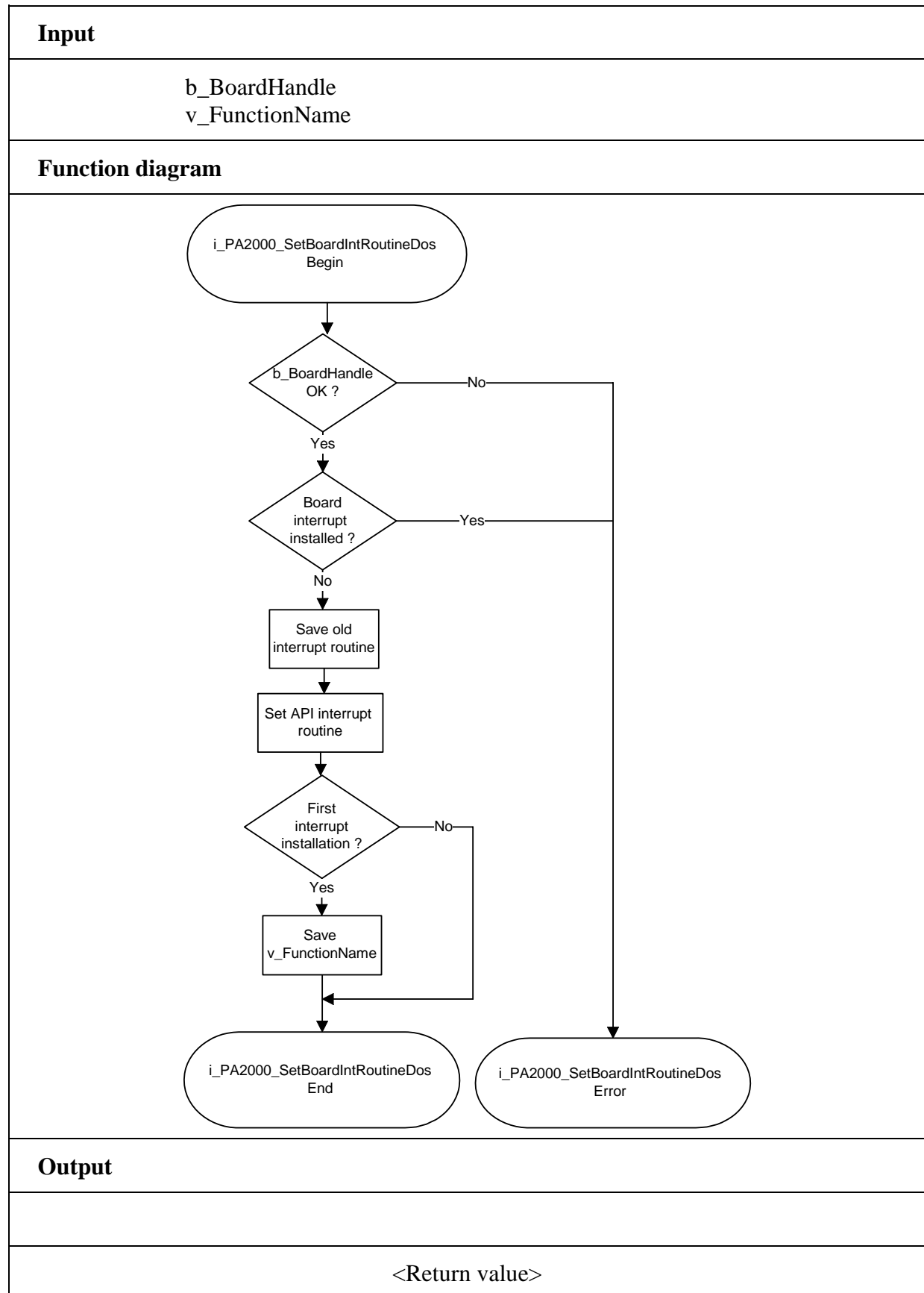
```
i_ReturnValue = i_PA2000_SetBoardIntRoutineDos (b_BoardHandle,
                                                v_FunctionName );
```

Return value:

0 : No error

-1: The handle parameter of the board is wrong

-2: Interrupt is already installed



i

IMPORTANT!

This function is only available for Visual Basic DOS.

2) i_PA2000_SetBoardIntRoutineVBDos (..)

Syntax:

<Return value> = i_PA2000_SetBoardIntRoutineVBDos
(BYTE b_BoardHandle)

Parameter:

- Input:

BYTE b_BoardHandle Handle of the **PA 2000** board

- Output:

No output signal has occurred.

Task:

This function must be called up for each **PA 2000** 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 2000** boards which have to react to interrupts, call up the function as often as you operate **PA 2000** boards.

Interrupt

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

Controlling the interrupt management

Please use the following functions instead: "ON UEVENT GOSUB xxxxxxxxx" of Visual Basic for DOS and "i_PA2000_TestInterrupt"

This function tests the interrupt of the **PA 2000**. It is used to obtain the values of *b_BoardHandle* , *b_InterruptMask*, *b_InputChannelNbr*.

Calling convention:

Visual Basic DOS:

Dim Shared i_ReturnValue	As Integer
Dim Shared i_BoardHandle	As Integer
Dim Shared i_InterruptMask	As Integer
Dim Shared i_InputChannelNbr	As Integer

IntLabel:

```
i_ReturnValue = i_PA2000_TestInterrupt (i_BoardHandle, _
                                         i_InterruptMask, _
                                         i_InputChannelNbr)
```

.
.

.

Return

ON UEVENT GOSUB IntLabel

UEVENT ON

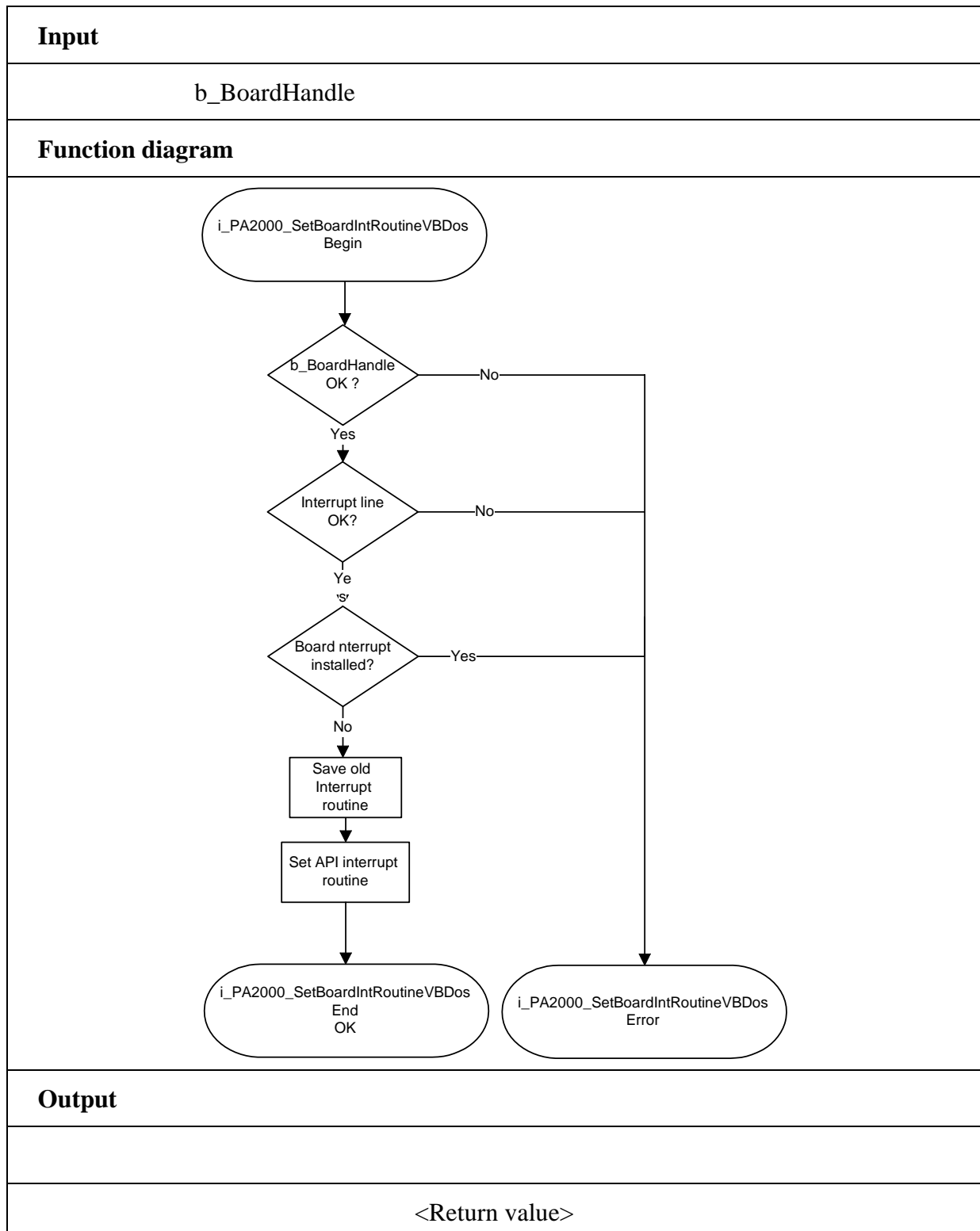
i_ReturnValue = i_PA2000_SetBoardIntRoutineVBDos (b_BoardHandle)

Return value:

0: No error

-1: The handle parameter of the board is wrong

-2: Interrupt already installed



3) i_PA2000_SetBoardIntRoutineWin16

Syntax:

```
<Return value> = i_PA2000_SetBoardIntRoutineWin16
                    (BYTE  b_BoardHandle
                     VOID   v_FunctionName
                     (BYTE  b_BoardHandle,
                      BYTE  b_InterruptMask,
                      BYTE  b_InputChannelNbr))
```

Parameter:

- Input:

BYTE	b_BoardHandle	Handle of the PA 2000 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 2000** on which an interrupt action is to be enabled.

First callup (first board):

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

If you operate several **PA 2000** boards which have to react to interrupts, call up the function as often as you operate PA 2000 boards.

The variable *v_FunctionName* is only relevant **for the first callup**.

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 counter/timer has run down
- an event is generated

The interrupt management can be simplified with the function

„i_PA2000_SetBoardIntRoutineDos“

The user interrupt routine must have the following syntax:

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

<i>v_FunctionName</i>	Name of the user interrupt routine
<i>b_BoardHandle</i>	Handle of the PA 2000 which has generated the interrupt
<i>b_InterruptMask</i>	Mask of the events which have generated the interrupt.
<i>b_InputChannelNbr</i>	If an interrupt is generated with a 0000 0001 Mask this variable gives the number of inputs which have generated the interrupt.

i

IMPORTANT!

If you use Visual Basic for Windows, the following parameter has no significance. Use the „i_PA2000_TestInterrupt“ function.

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

Calling convention:

ANSI C:

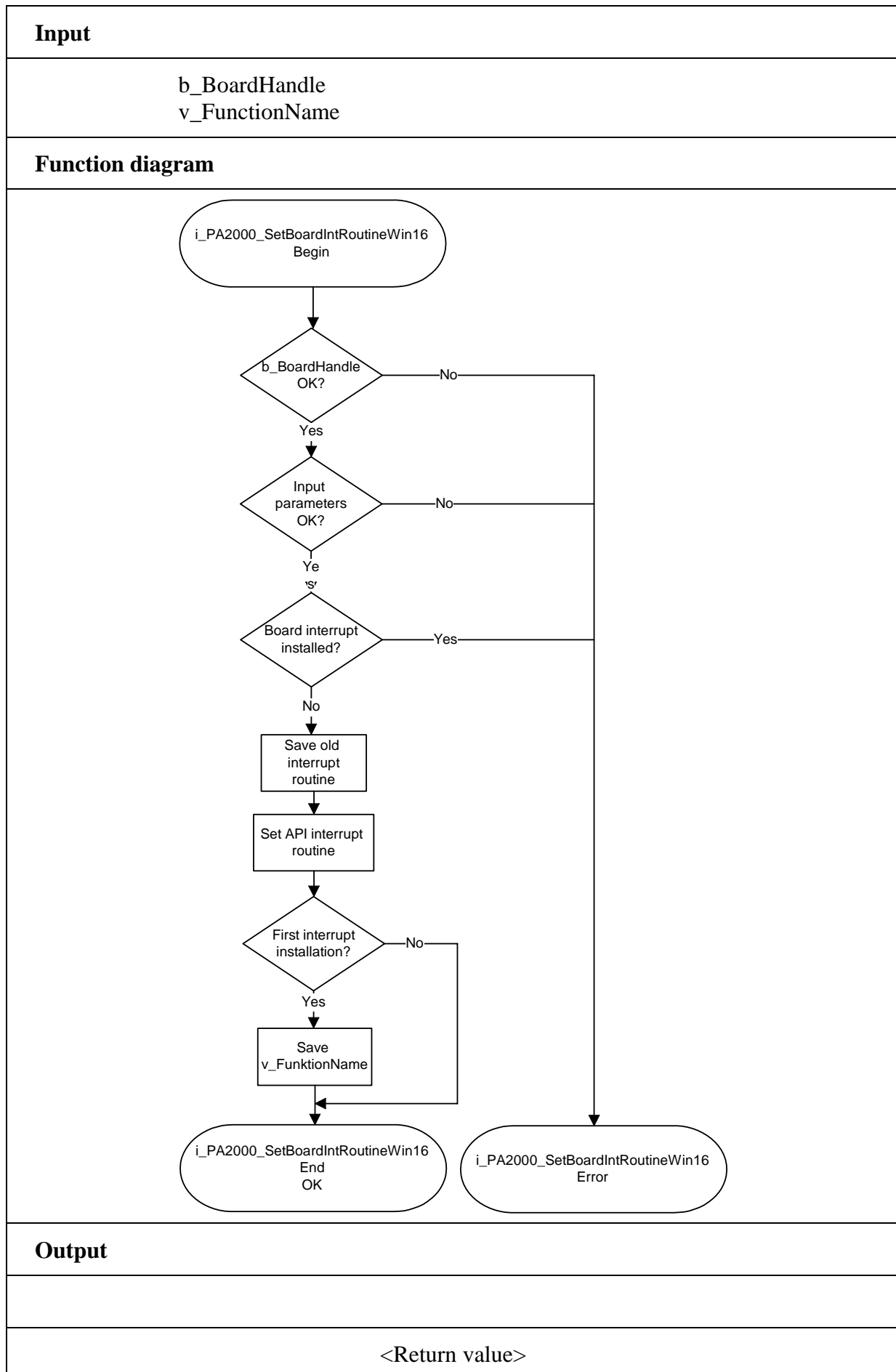
```
void v_FunctionName (unsigned char b_BoardHandle,
                    unsigned char b_InterruptMask,
                    unsigned char b_InputChannelNumber)
{
    .
    .
}
```

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

```
i_ReturnValue = i_PA2000_SetBoardIntRoutineWin16 (b_BoardHandle,
                                                  v_FunctionName);
```

Return value:

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



i**IMPORTANT!**

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

4) i_PA2000_SetBoardIntRoutineWin32 (..)

Syntax:

```
<Return value> = i_PA2000_SetBoardIntRoutineWin32
    (BYTE      b_BoardHandle,
     BYTE      b_UserCallingMode,
     ULONG     ul_UserSharedMemorySize,
     VOID **   ppv_UserSharedMemory,
     VOID      v_FunctionName
     (BYTE  b_BoardHandle,
      BYTE  b_InterruptMask,
      BYTE  b_InputChannelNumber,
      BYTE  b_UserCallingMode,
      VOID * pv_UserSharedMemory))
```

Parameter:- Input:

BYTE	b_BoardHandle	Handle of the PA 2000 board
BYTE	b_UserCallingMode	PA2000_SYNCHRONOUS_MODE: The user routine is directly called by the driver interrupt routine. PA2000_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 of the user shared memory in bytes. Only used if you have selected PA2000_SYNCHRONOUS_MODE

i**IMPORTANT!**

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

Output:

VOID **	ppv_UserSharedMemory	User shared memory address Only used if you have selected PA2000_SYNCHRONOUS_MODE
---------	----------------------	--

Task:

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

i**Windows 32-bit information**

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 drivers operate in ring 0 and give access to hardware. Ring 0 has no direct access to the global variable of ring 3. It has to use a shared memory.

- Ring 0 and ring 3 have a pointer that points to this shared memory.
The two pointers are not configured under the same address.

This function must be called up for each **PA 2000** for which an interrupt is to be enabled. It installs one user interrupt function in all the boards on which an interrupt is to be enabled.

First callup (first board):

- the user interrupt routine is installed
- interrupts are enabled
- the user shared memory is allocated, if PA2000_SYNCHRONOUS_MODE has been selected.

If you operate several PA 2000 boards which have to react to interrupts, call up the function as often as you operate PA 2000 boards. The variable *v_FunctionName* is only relevant **for the first calling**.

From the second callup 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 operates directly in ring 0.
- by the driver interrupt thread (asynchronous mode). An event is generated and the interrupt thread calls up the user interrupt routine. The code of the user interrupt routine operates in ring 3.

The driver interrupt thread has the highest priority (31) in the system.

Table 9-5: Synchronous and asynchronous modes

	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/she 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 PA2000_SYNCHRONOUS_MODE you have no access to global variables. But you have the possibility of creating 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 of the selected user type in byte. 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 (BYTEb_BoardHandle,
                     BYTE      b_InterruptMask,
                     BYTE      b_InputChannelNumber,
                     BYTE      b_UserCallingMode,
                     VOID *     pv_UserSharedMemory)
```

<i>v_FunctionName</i>	Name of the user interrupt routine
<i>b_BoardHandle</i>	Handle of the PA 2000 which has generated the interrupt
<i>b_InterruptMask</i>	Mask of the events which have generated the interrupt.
<i>b_InputChannelNbr</i>	If an interrupt is generated with a Mask 0000 0001, this variable gives the input number which has generated the interrupt.

b_UserCallingMode PA2000_SYNCHRONOUS_MODE:
The user routine is directly called by the driver interrupt routine.
PA2000_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 significance. Use the „i_PA2000_TestInterrupt“ function.

```

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

```

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

Calling convention:

ANSI C :

```

typedef struct
{
    .
    .
    .
}str_UserStruct;

str_UserStruct * ps_UserSharedMemory;

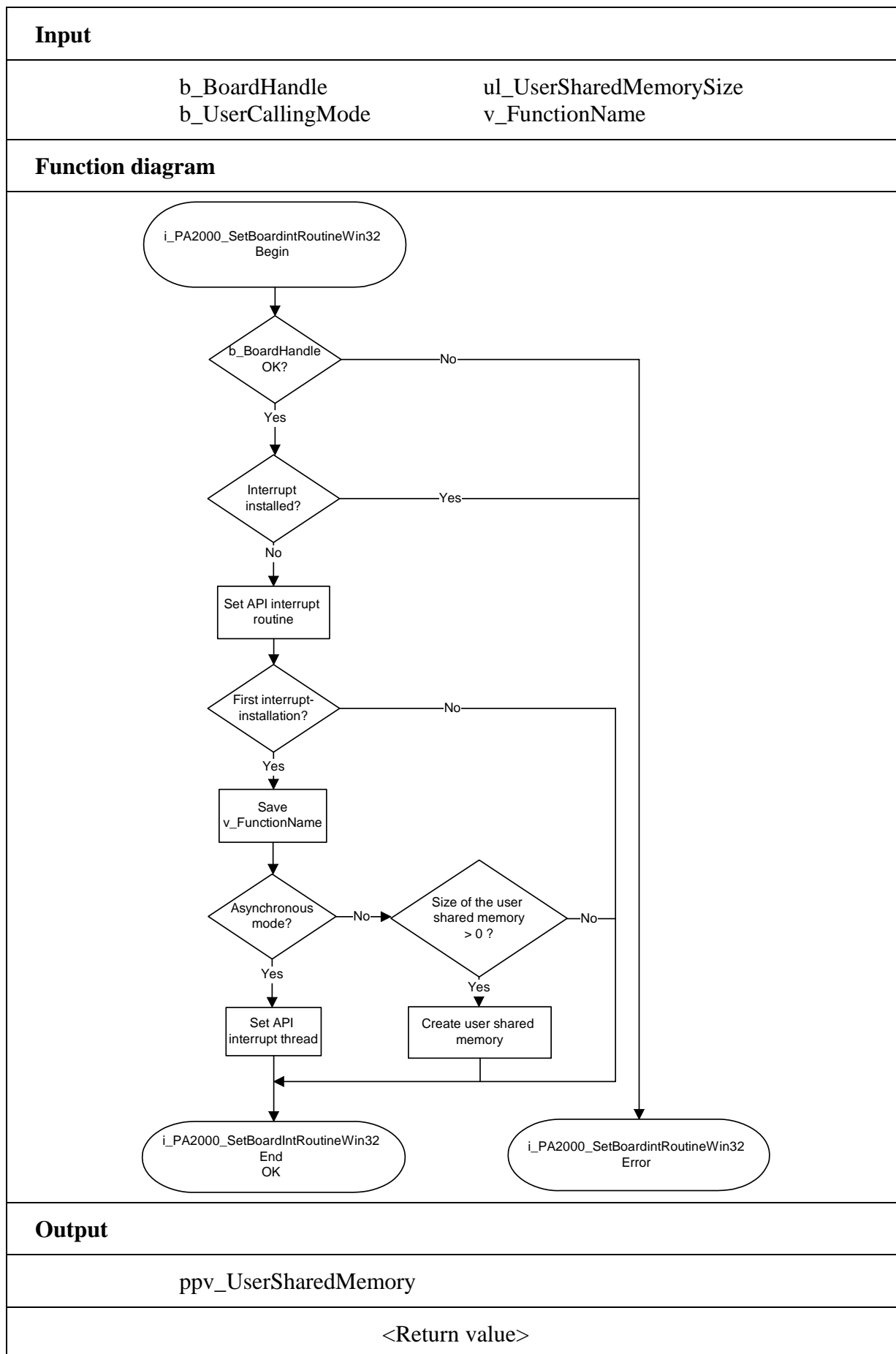
void          v_FunctionName      (unsigned char b_BoardHandle,
                                   unsigned char b_InterruptMask,
                                   unsigned char b_InputChannelNbr,
                                   unsigned char b_UserCallingMode,
                                   void *       pv_UserSharedMemory)
{
    str_UserStruct * ps_InterruptSharedMemory;
    ps_InterruptSharedMemory = (str_UserStruct *) pv_UserSharedMemory;
    .
    .
}
int          i_ReturnValue;
unsigned char b_BoardHandle;

```

```
i_ReturnValue = i_PA2000_SetBoardIntRoutineWin32  
                (b_BoardHandle,  
                 PA2000_SYNCHRONOUS_MODE,  
                 size of (str_UserStruct), (void **)  
                 &ps_UserSharedMemory, v_FunctionName);
```

Return value:

- 0: No error
- 1: The board handle parameter is wrong
- 2: Interrupt already installed
- 3: Parameter b_UserCallingMode is wrong.



5) i_PA2000_TestInterrupt (..)

Syntax:

<Return value> = i_PA2000_TestInterrupt (PBYTE pb_BoardHandle,
PBYTE pb_InterruptMask,
PBYTE pb_ChannelNbr)

Parameter:
- Input:

No input signal has occurred.

- Output:

PBYTE pb_BoardHandle

Handle of the PA 2000 board which has generated the interrupt, Error mask of the event which has generated the interrupt. Several errors can occur simultaneously.

PBYTE pb_InterruptMask

PBYTE pb_ChannelNbr

If an interrupt occurs with the 0000 0001 mask, this variable returns the number of the input which has generated the event.

Mask	Meaning
0000 0001	Event 1 has occurred
0000 0010	Event 2 has occurred
0000 0100	Counter/timer 1 has run down
0000 1000	Counter/timer 2 has run down
0001 0000	Counter/Timer3 has run down

Task:

Checks if a **PA 2000** board has generated an interrupt. If yes, the function returns the board handle and the interrupt source.

i

IMPORTANT!

This function is only available under Visual Basic Dos and Windows.

Calling convention:
ANSI C:

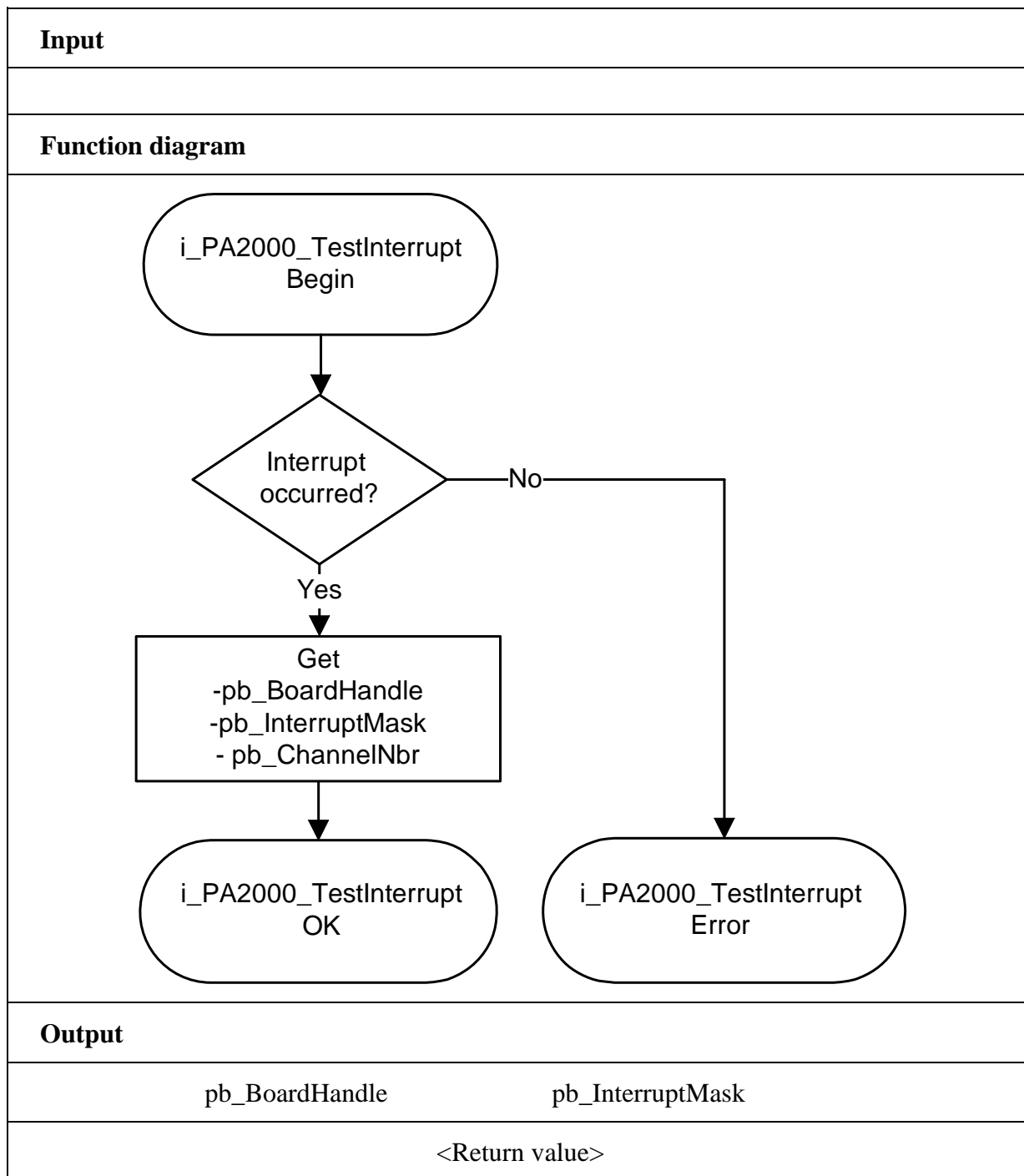
```
unsigned char b_BoardHandle;
unsigned char b_InterruptMask;
unsigned char b_ChannelNbr;
int          i_Irq;
```

```
i_Irq = i_PA2000_TestInterrupt (&b_BoardHandle,
                                &b_InterruptMask,
                                &b_ChannelNbr);
```

Return value:

-1: No interrupt

> 0: IRQ number



6) i_PA2000_ResetBoardIntRoutine (..)

Syntax:

<Return value> = i_PA2000_ResetBoardIntRoutine (BYTE b_BoardHandle)

Parameter:**- Input:**

BYTE b_BoardHandle Handle of the **PA 2000** board

- Output:

No output signal has occurred

Task:

Stops the interrupt management of the **PA 2000** board.

Deinstalls the interrupt routine if the interrupt management of all **PA 2000** boards is stopped.

Calling convention:ANSI C:

```
unsigned char      b_BoardHandle;
```

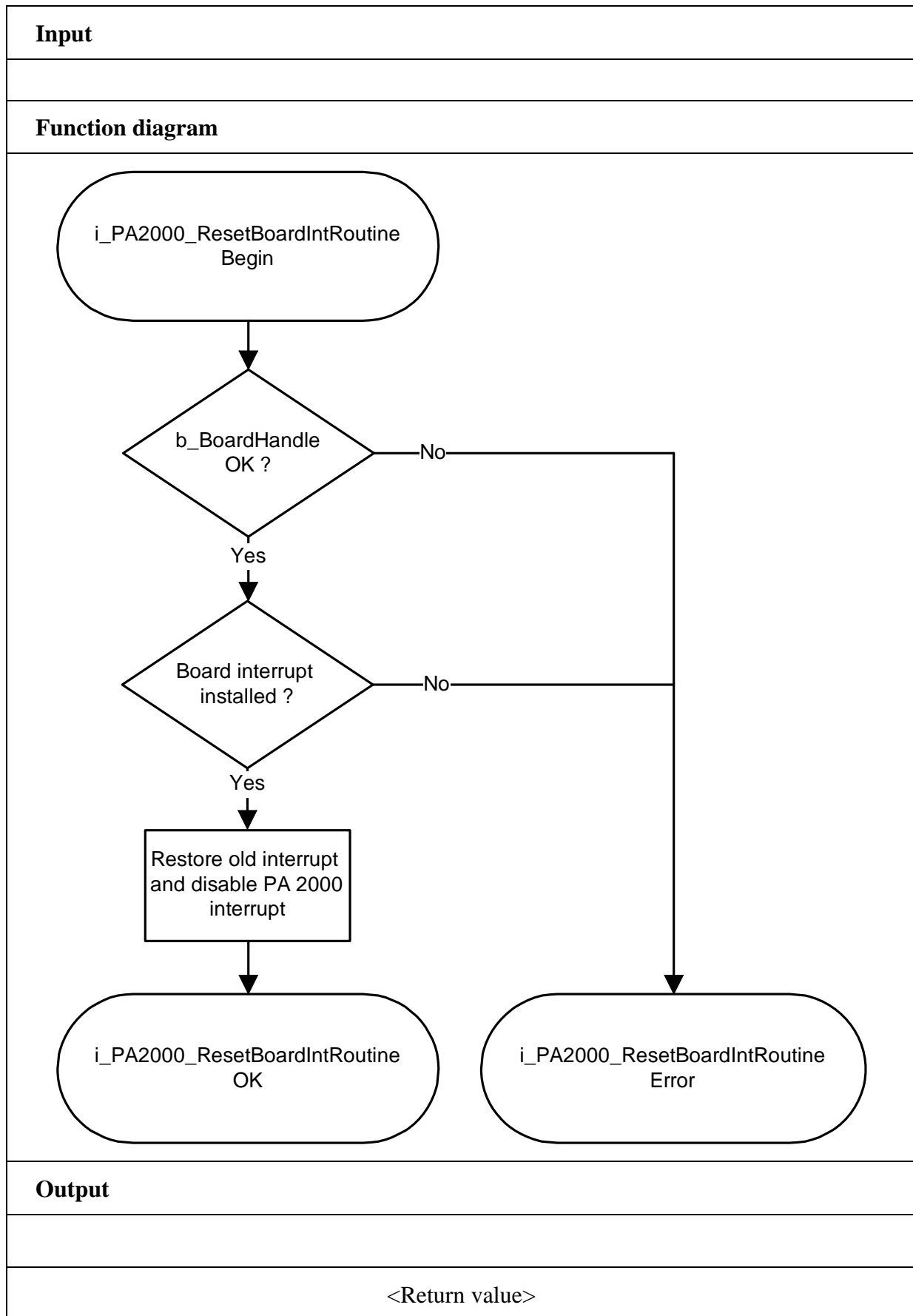
```
i_PA2000_ResetBoardIntRoutine(b_BoardHandle);
```

Return value:

0: No error

-1: Handle parameter of the board is wrong

-2: Interrupt routine is not installed



9.5 Digital Output channels

1) i_PA2000_SetOutputMemoryOn (...)

Syntax:

<Return value> = i_PA2000_SetOutputMemoryOn (BYTE b_BoardHandle)

Parameters:**- Input:**

BYTE b_BoardHandle Handle of the **PA 2000**

- Output:

No output signal has occurred.

Task:

Activates the digital output memory.

After calling this function, the outputs you have previously activated

with the functions "i_PA2000_SetXDigitalOutputOn" are not reset.

You can reset them with the function "i_PA2000_SetXDigitalOutputOff".

Calling convention:ANSI C :

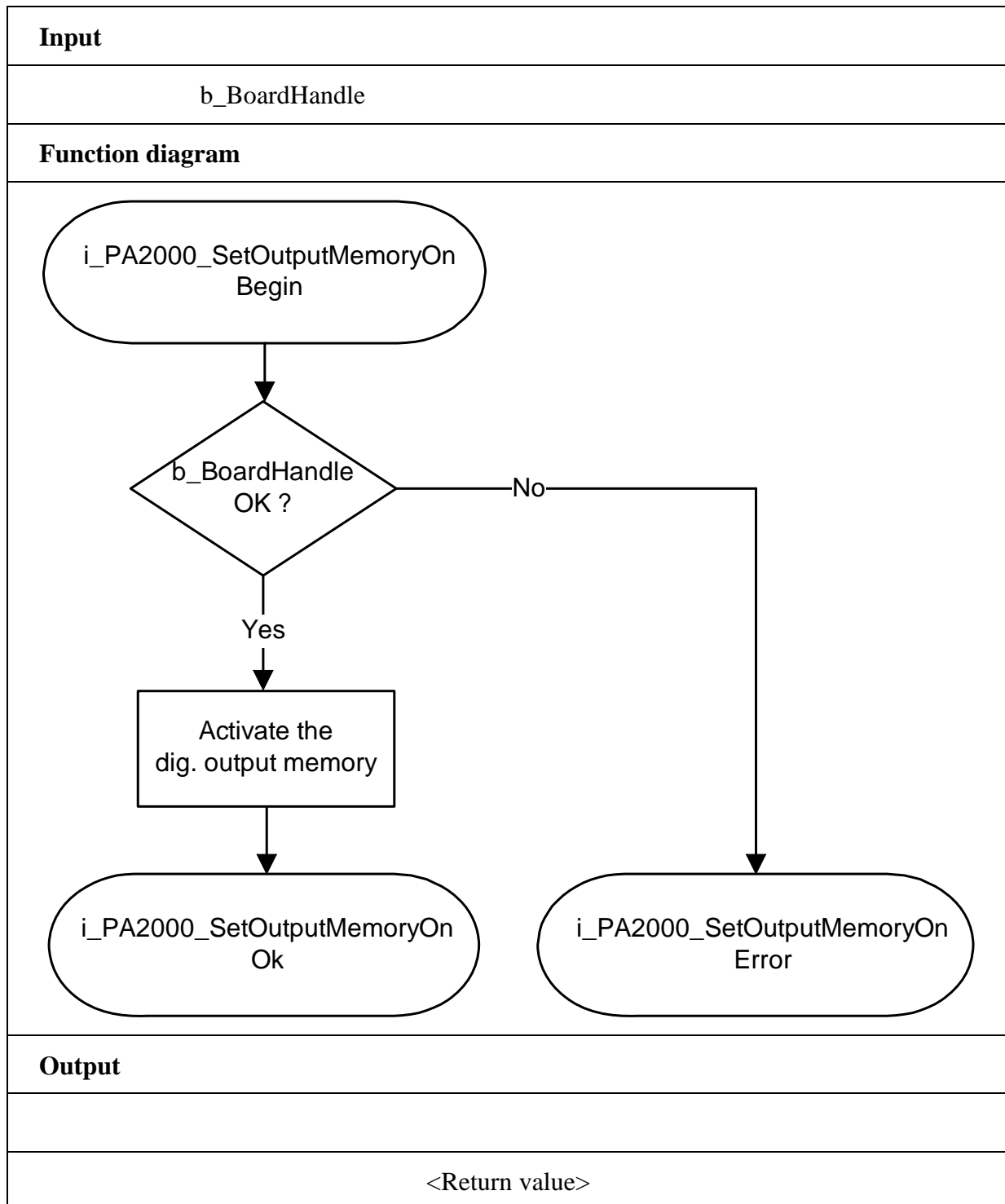
```
int          i_ReturnValue;  
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_PA2000_SetOutputMemoryOn (b_BoardHandle);
```

Return value:

0: No error

-1: Handle parameter of the board is wrong



2) i_PA2000_SetOutputMemoryOff (...)

Syntax:

<Return value> = i_PA2000_SetOutputMemoryOff (BYTE b_BoardHandle)

Parameters:**- Input:**

BYTE b_BoardHandle Handle of the **PA 2000**

- Output:

No output signal has occurred.

Task:

Deactivates the digital output memory.

Calling convention:ANSI C :

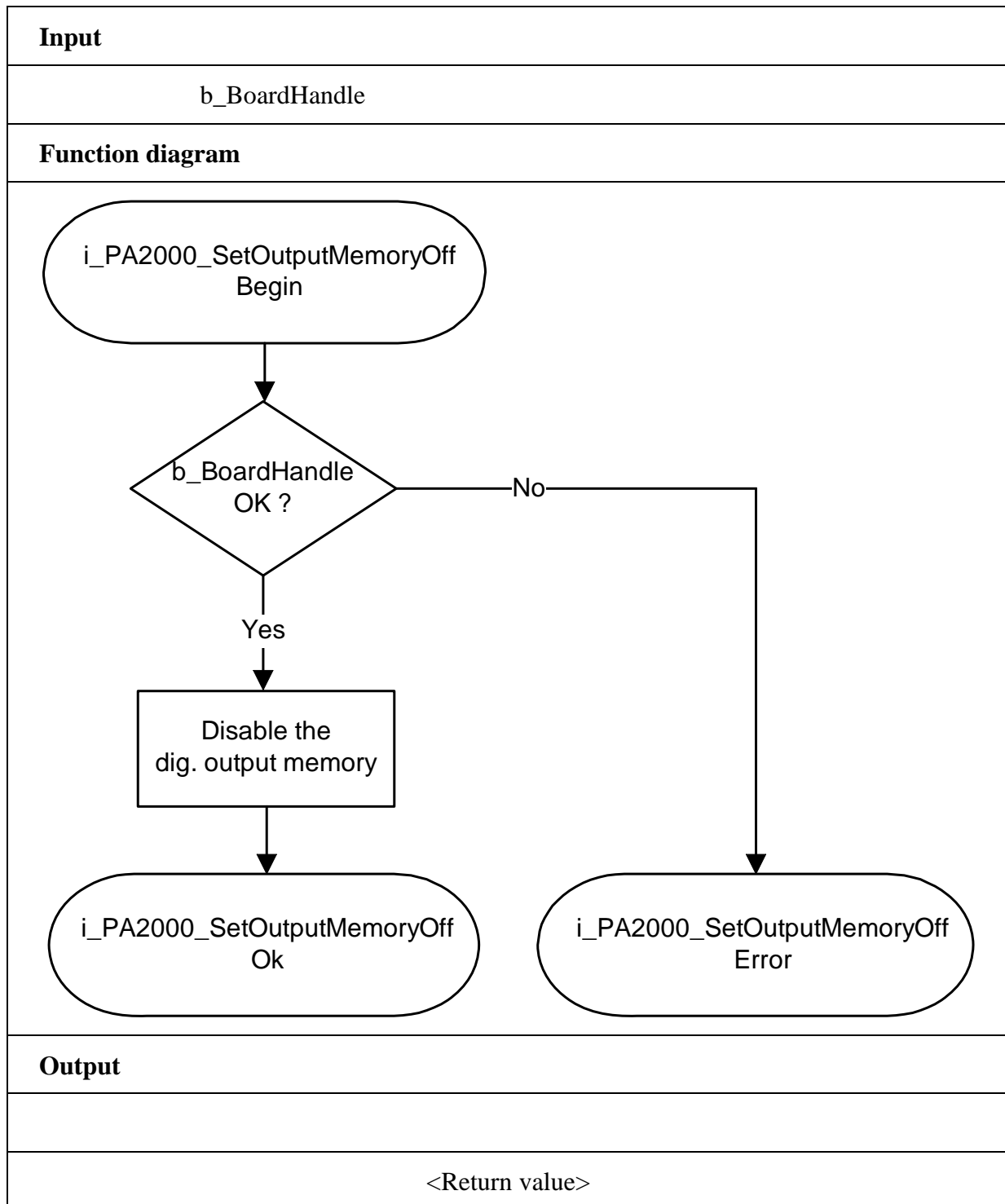
```
int                      i_ReturnValue;  
unsigned char        b_BoardHandle;
```

```
i_ReturnValue = i_PA2000_ SetOutputMemoryOff (b_BoardHandle);
```

Return value:

0: No error

-1: Handle parameter of the board is wrong



3) i_PA2000_Set1DigitalOutputOn (...)**Syntax:**

```
<Return value> = i_PA2000_Set1DigitalOutputOn
                    (BYTE b_BoardHandle,
                     BYTE b_Channel)
```

Parameters:**- Input:**

BYTE	b_BoardHandle	Handle of the PA 2000
BYTE	b_Channel	Number of the output you want to set (1 to 16)

- Output:

No output signal has occurred.

Task:

Sets the output which has been passed with *b_Channel*. Setting an output means setting it to high.

Switching on the digital output memory (ON)

see function "i_PA2000_SetOutputMemoryOn (...)
b_Channel= 1

The output 1 is set. The other outputs hold their state.

Switching off the digital output memory (OFF)

see function "i_PA2000_SetOutputMemoryOff (...)
b_Channel= 1

The output 1 is set. The other outputs are reset.

If you have switched off the digital output memory (OFF), all other inputs are set to "0".

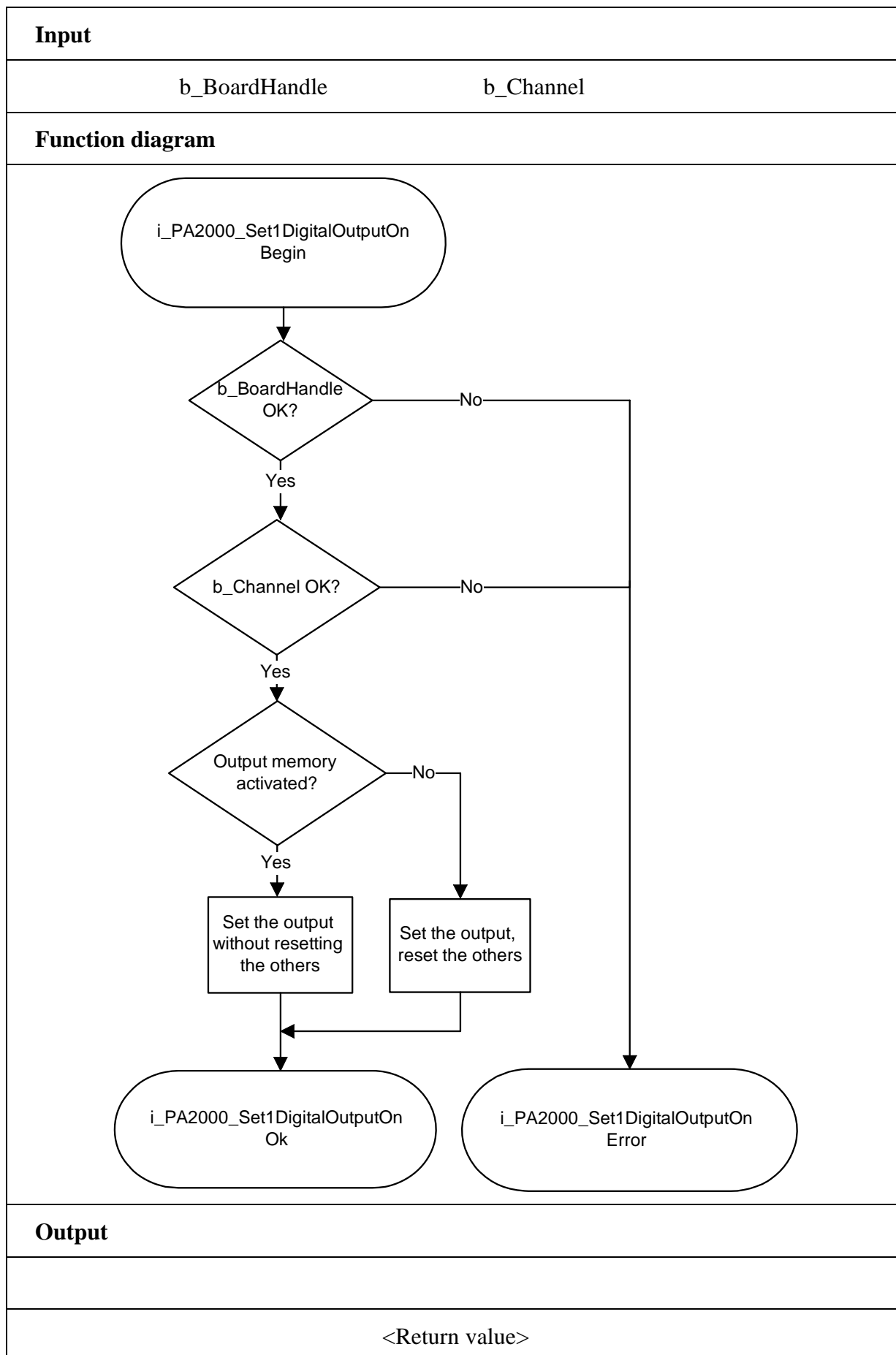
Calling convention:ANSI C:

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

```
i_ReturnValue = i_PA2000_Set1DigitalOutputOn (b_BoardHandle,
                                                0);
```

Return value:

0: No error
-1: Handle parameter of the board is wrong
-2: Input number is not between 1 and 32



4) i_PA2000_Set1DigitalOutputOff (...)**Syntax:**

```
<Return value> = i_PA2000_Set1DigitalOutputOff
                                     (BYTE b_BoardHandle,
                                      BYTE b_Channel)
```

Parameters:**- Input:**

BYTE	b_BoardHandle	Handle of the PA 2000
BYTE	b_Channel	Number of the output you want to reset (1 to 16)

- Output:

No output signal has occurred.

Task:

Resets the output you have passed with *b_Channel*. Resetting an output means setting it to "Low".

i**IMPORTANT!**

**You can use this function only if the digital output memory is ON.
See function "i_PA2000_SetOutputMemoryOn (..)".**

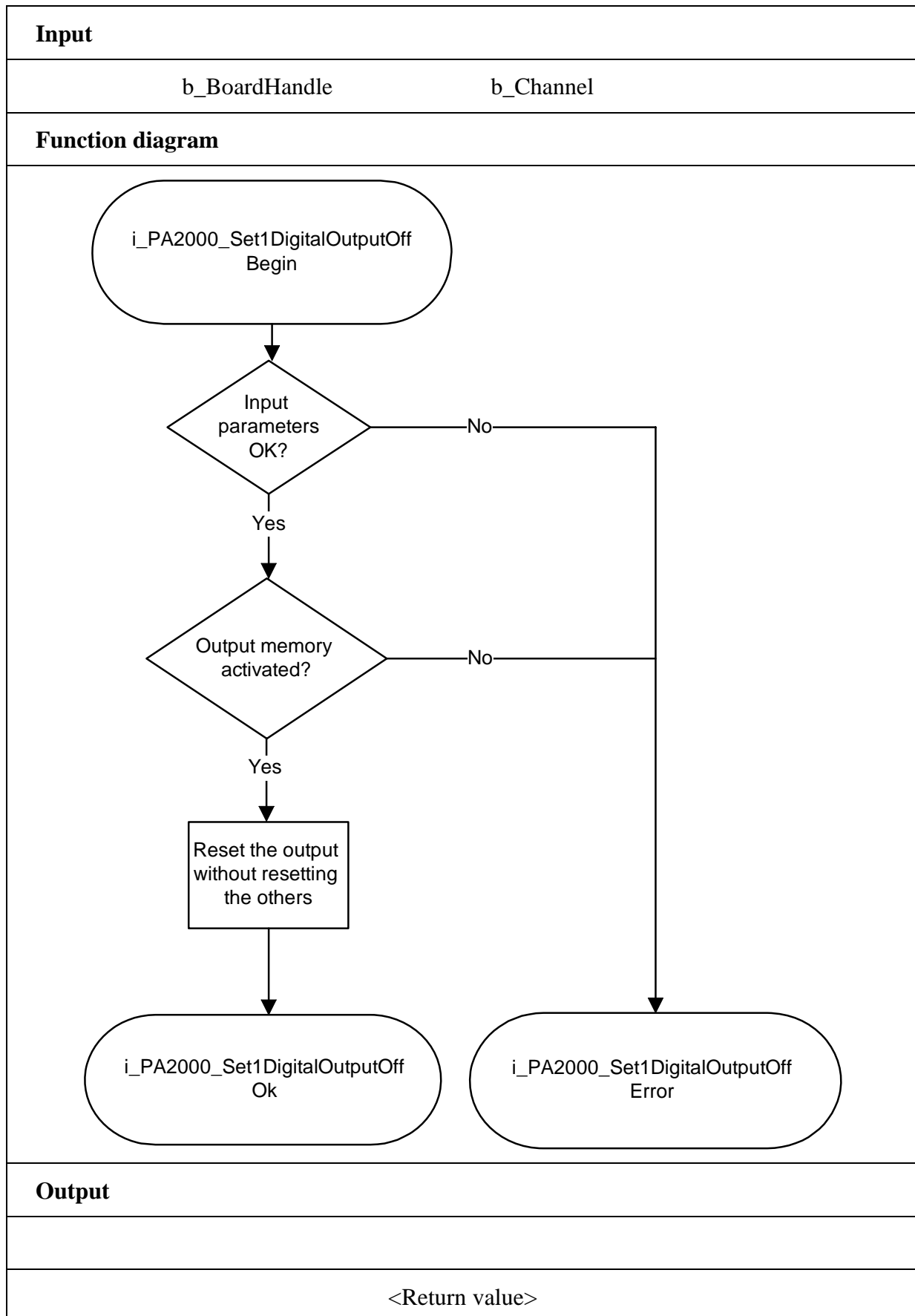
Calling convention:ANSI C:

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

```
i_ReturnValue = i_PA2000_Set1DigitalOutputOff (b_BoardHandle,
                                                0);
```

Return value:

0: No error
 -1: The handle parameter of the board is wrong
 -2: The input number is not between 1 and 16
 -3: The digital output memory is OFF. First use the function
 "i_PA2000_SetOutputMemoryOn"



5) i_PA2000_Set8DigitalOutputOn (...)**Syntax:**

<Return value> = i_PA2000_Set8DigitalOutputOn
 (BYTE b_BoardHandle,
 BYTE b_Port,
 BYTE b_Value)

Parameters:**_Input:**

BYTE	b_BoardHandle	Handle of the PA 2000 board
BYTE	b_Port	Number of the output port (1, 2, 3 or 4)
BYTE	b_Value	Output value (0 to 255)

- Output:

No output signal has occurred.

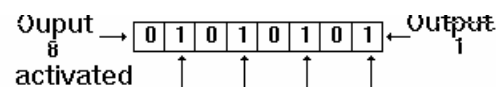
Task:

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

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

see function "i_PA2000_SetOutputMemoryOn (...)"

b_Port = 1 b_Value = 55 Hex

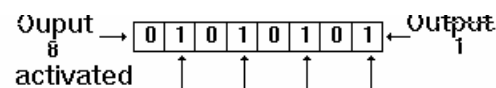


The outputs 1, 3, 5, 7 are set. The other outputs hold their state.

Switching off the digital output memory (OFF)

see function "i_PA2000_SetOutputMemoryOff (...)"

b_Port = 1
 b_Value = 55 Hex



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

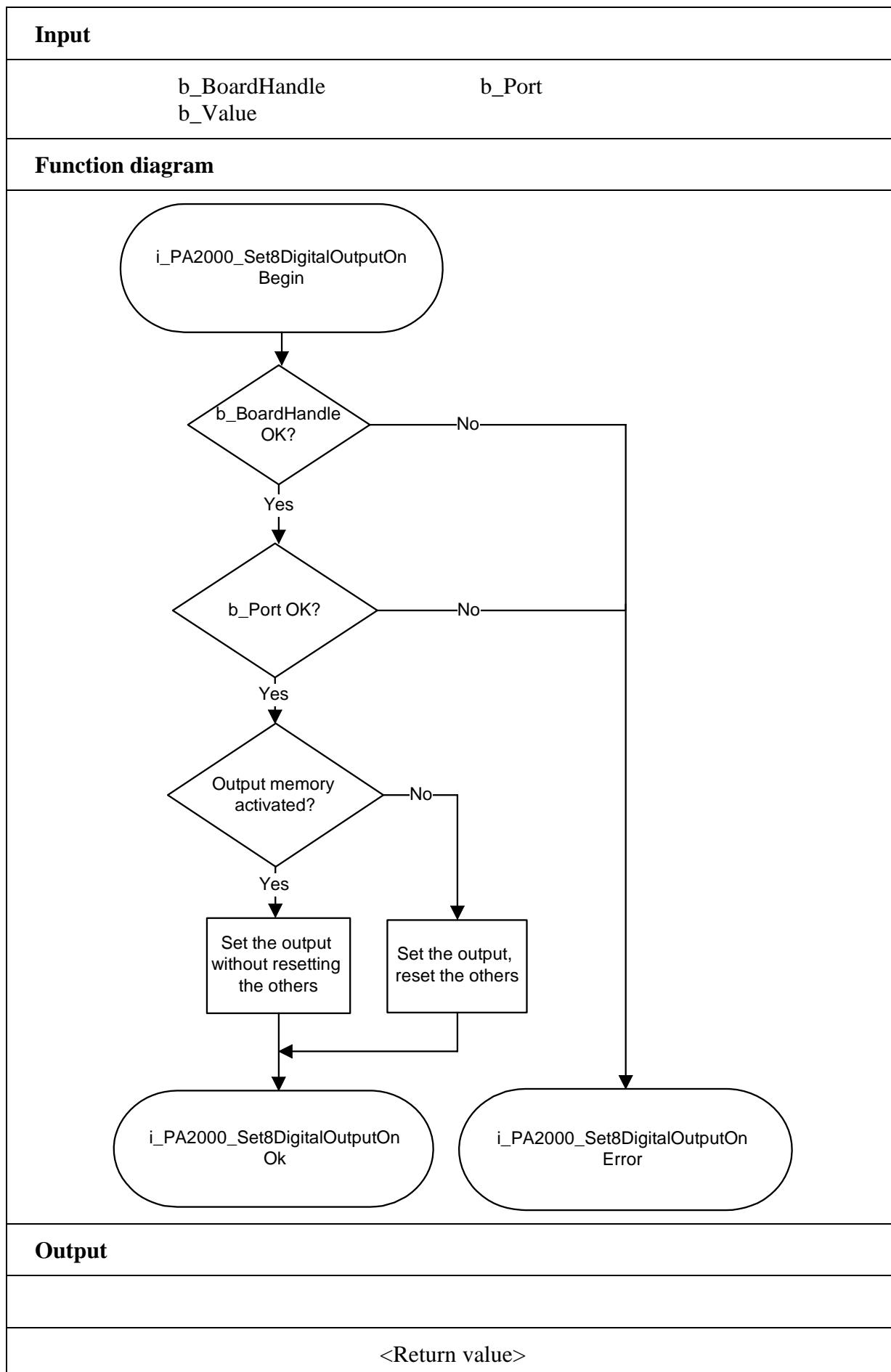
Calling convention:**ANSI C :**

```
int          i_ReturnValue;
unsigned char b_BoardHandle;

i_ReturnValue = i_PA2000_Set8DigitalOutputOn (b_BoardHandle,
                                              0,
                                              255);
```

Return value:

0: No error
 -1: The handle parameter of the board is wrong
 -2: The port number is not 1, 2, 3 or 4



6) i_PA2000_Set8DigitalOutputOff (...)

Syntax:

<Return value> = i_PA2000_Set8DigitalOutputOff (BYTE b_BoardHandle,
 BYTE b_Port,
 BYTE b_Value)

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the PA 2000
BYTE	b_Port	Number of the output port (1, 2, 3 or 4)
BYTE	b_Value	Output value (0 to 255)

- Output:

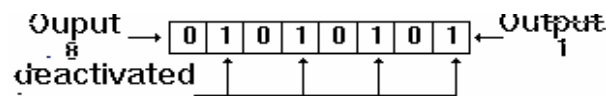
No output signal has occurred.

Task:

Resets one or several outputs of one port. Resetting an output means setting it to "High".

Example:

b_Port = 1
 b_Value = 55 Hex



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

i

IMPORTANT!

**You can use this function only if the digital output memory is ON.
 See function “i_PA2000_SetOutputMemoryOn (..)”.**

Calling convention:

ANSI C :

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

```
i_ReturnValue = i_PA2000_Set8DigitalOutputOff (b_BoardHandle,
  0,
  127);
```

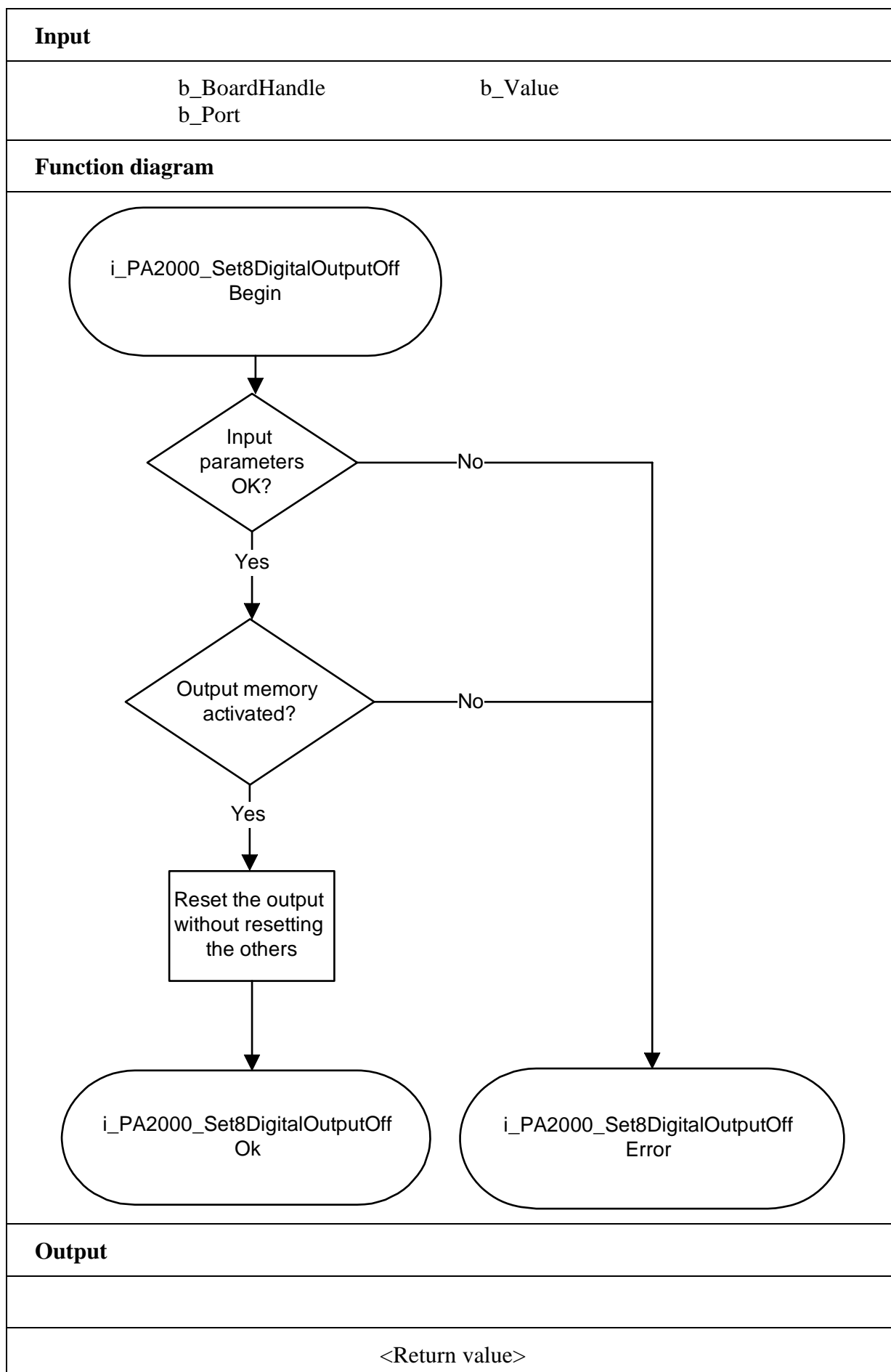
Return value:

0: No error

-1: The handle parameter of the board is wrong

-2: The port number is not 1, 2, 3 or 4

-3: The digital output memory is OFF. Please use the function
 “i_PA2000_SetOutputMemoryOn” first



7) i_PA2000_Set16DigitalOutputOn (...)**Syntax:**

```
<Return value> = v_PA2000_Set16DigitalOutputOn
                                     (BYTE b_BoardHandle,
                                     BYTE b_Port,
                                     LONG l_Value)
```

Parameters:**- Input:**

BYTE	b_BoardHandle	Handle of the PA 2000
BYTE	b_Port	Number of the output port (1 or 2)
LONG	l_Value	Output value (0 to 65535)

- Output:

No output signal has occurred.

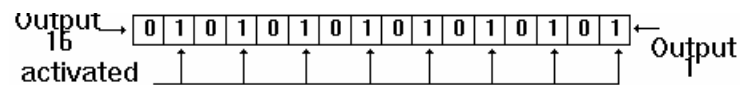
Task:

Sets one or several outputs of the PA 2000 board

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

see function "i_PA2000_SetOutputMemoryOn (...)"

l_Value = 5555 Hex

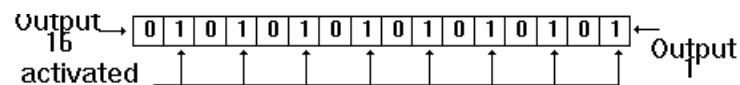


The outputs 1, 3, 5, 7, 9, 11, 13, 15 are set. The other outputs hold their state.

Switching off the digital output memory (OFF)

see function "i_PA2000_SetOutputMemoryOff (...)"

l_Value = 5555 Hex



Outputs 1, 3, 5, 7, 9, 11, 13, 15 are set.

The outputs 2, 4, 6, 8, 10, 12, 14, 16 are reset.

Calling convention:**ANSI C :**

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

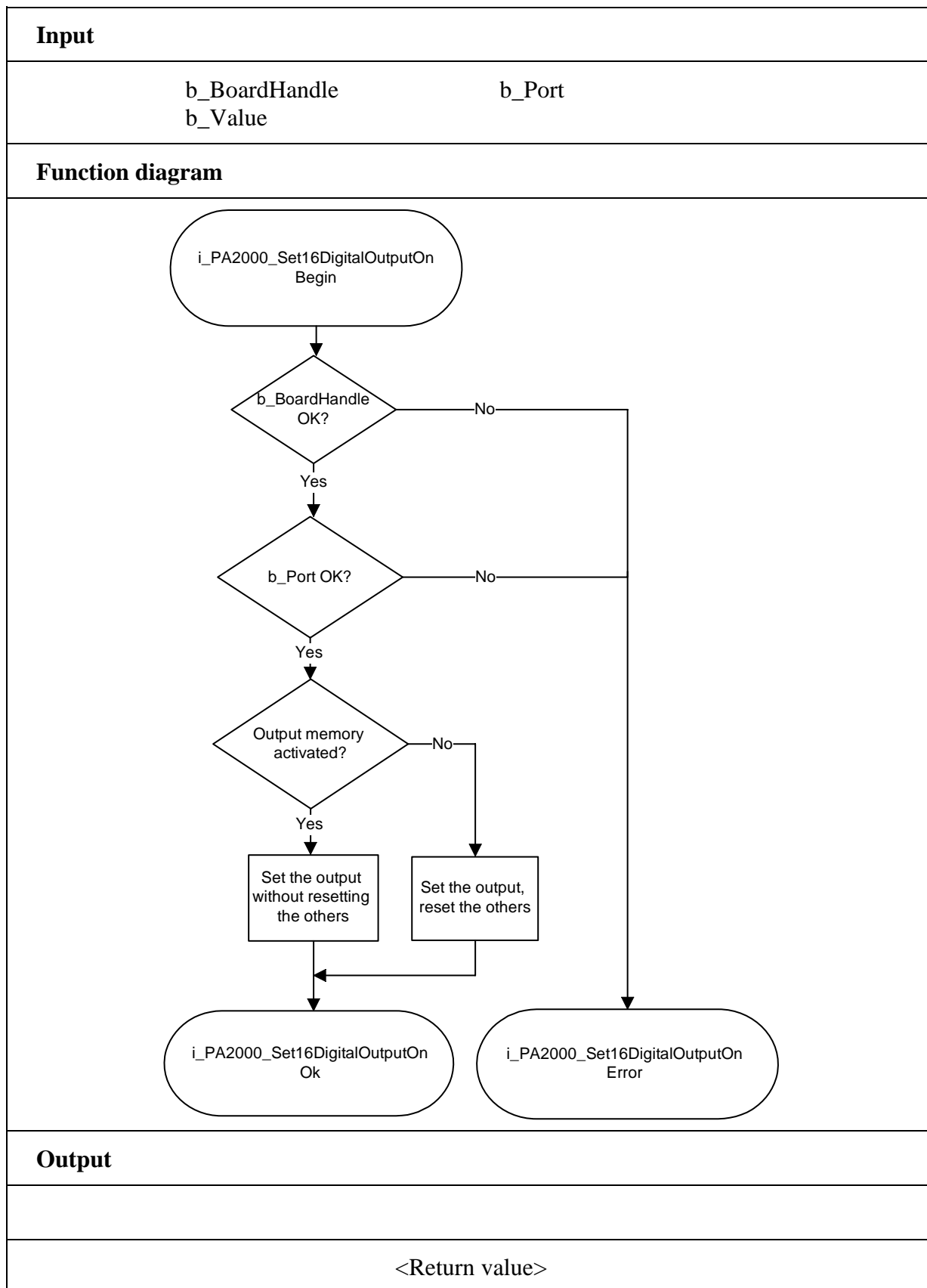
```
i_ReturnValue = i_PA2000_Set16DigitalOutputOn (b_BoardHandle,
                                                0,
                                                65535);
```

Return value:

0: No error

-1: The handle parameter of the board is wrong

-2: The port number is not 1 or 2



8) i_PA2000_Set16DigitalOutputOff (...)**Syntax:**

<Return value> = v_PA2000_Set16DigitalOutputOff
 (BYTE b_BoardHandle,
 BYTE b_Port,
 LONG l_Value)

Parameters:**- Input:**

BYTE	b_BoardHandle	Handle of the PA 2000
BYTE	b_Port	Number of the output port (1 or 2)
LONG	l_Value	Output value (0 to 65535)

- Output:

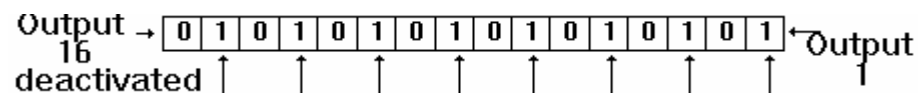
No output signal has occurred.

Task:

Resets one or several outputs of the **PA 2000** board.

Example:

l_Value = 5555 Hex



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

i

IMPORTANT!

**You can use this function only if the digital output memory is ON.
 See function “i_PA2000_SetOutputMemoryOn (..)”.**

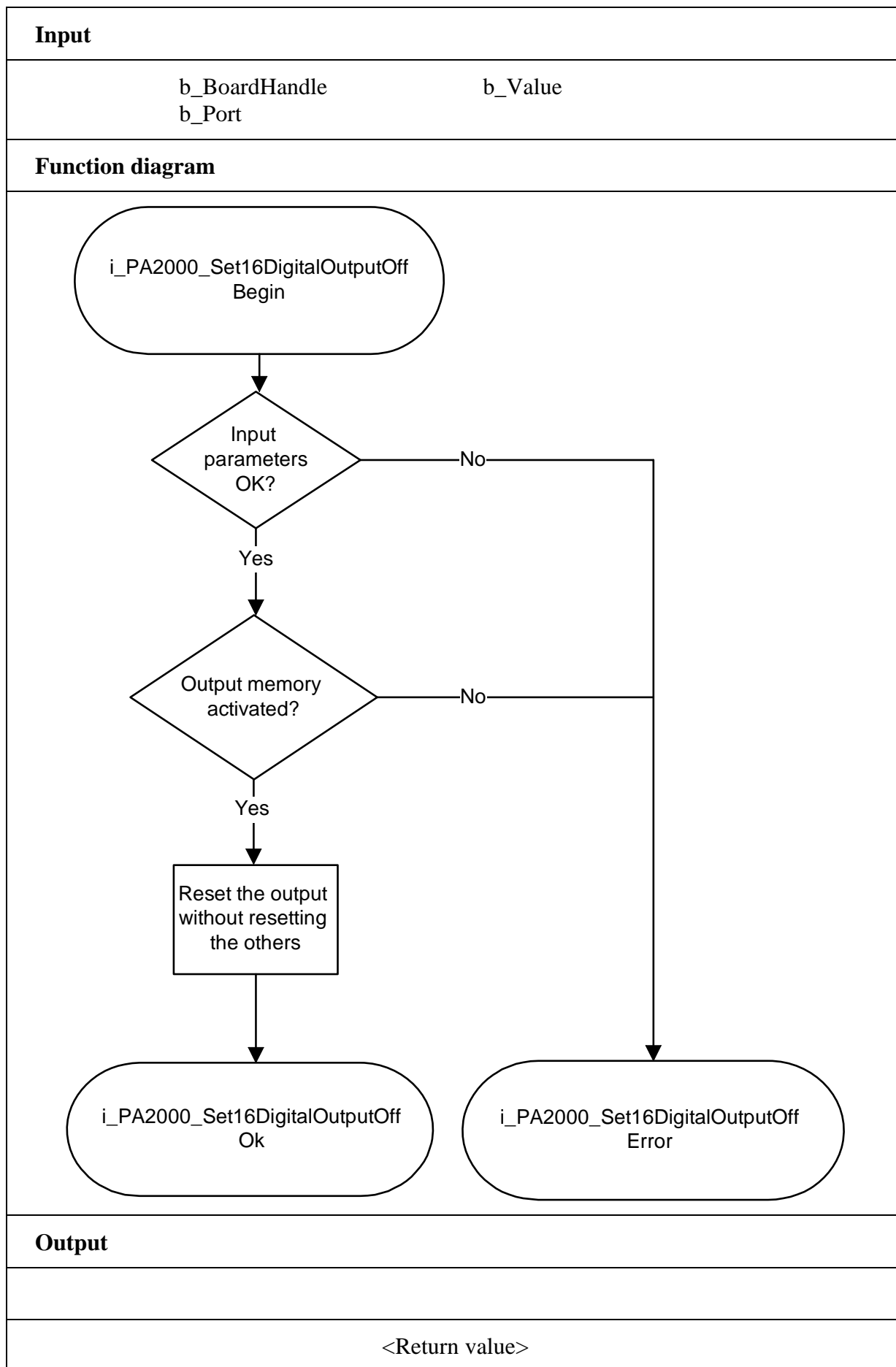
Calling convention:ANSI C:

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

```
i_ReturnValue = i_PA2000_Set16DigitalOutputOff (b_BoardHandle,
0,
32767);
```

Return value:

0: No error
 -1: The handle parameter of the board is wrong
 -2: The port number is not 1 or 2
 -3: The digital output memory is OFF. Please first use the function
 “i_PA2000_SetOutputMemoryOn”



9) i_PA2000_Set32DigitalOutputOn (...)

Syntax:

<Return value> = v_PA2000_Set32DigitalOutputOn
(BYTE b_BoardHandle,
LONG l_Value)

Parameters:

- Input:

BYTE b_BoardHandle Handle of the **PA 2000**
LONG l_Value Output value (0 to 2³²)

- Output:

No output signal has occurred.

Task:

Sets one or several outputs of the **PA 2000** board

Example:

Switching on the digital output memory (ON)

see function "i_PA2000_SetOutputMemoryOn (...)

l_Value = 55555555 Hex

Output 32 activated →

0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 ← Output 1

The outputs 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31 are set.

The other outputs hold their state.

Switching off the digital output memory (OFF)

see function "i_PA2000_SetOutputMemoryOff (...)

l_Value = 55555555 Hex

Output 32 activated →

0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 ← Output 1

The outputs 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31 are set.

The outputs 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32 are reset.

Calling convention:

ANSI C:

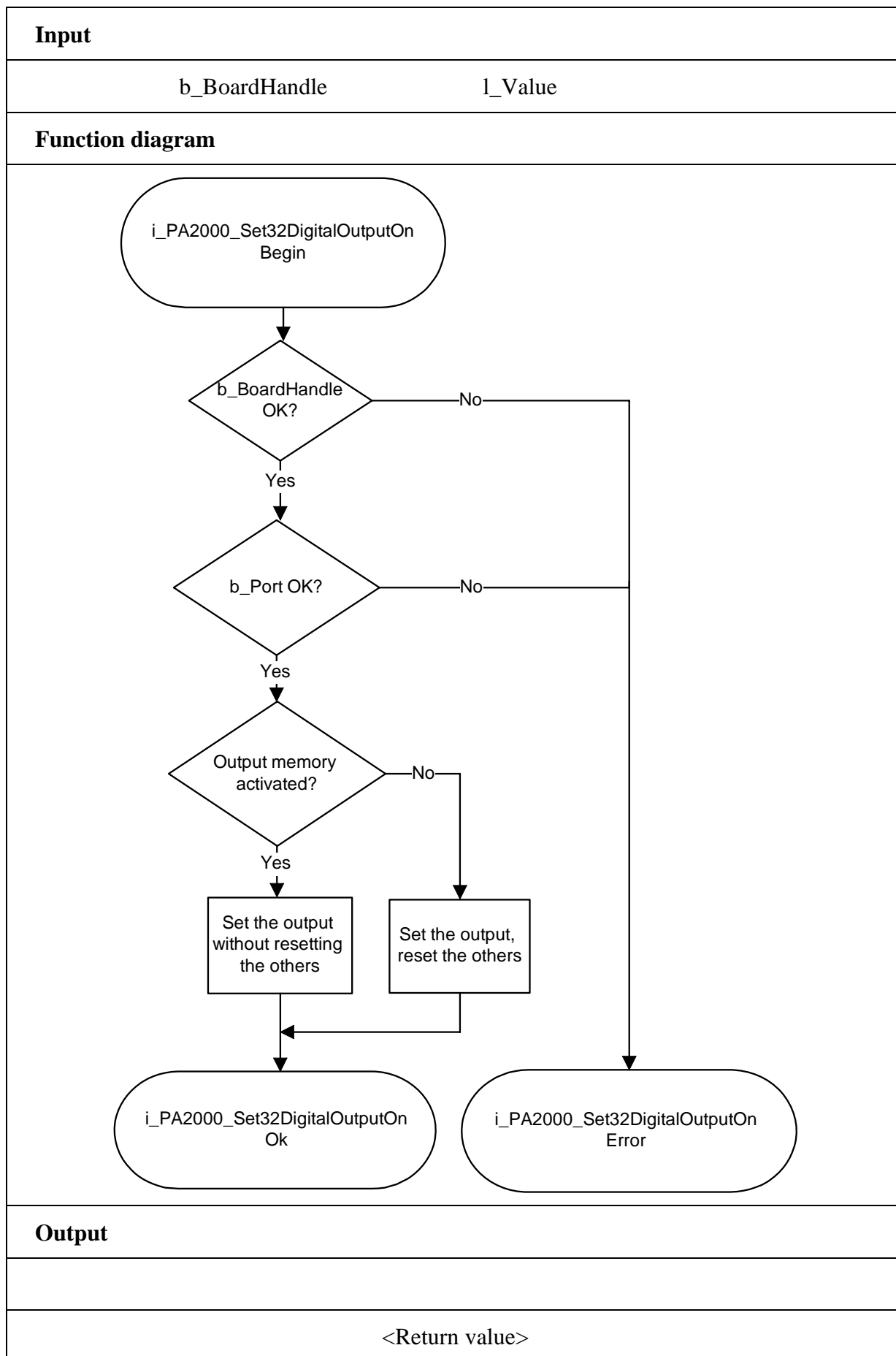
```
int          i_ReturnValue;
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_PA2000_Set32DigitalOutputOn (b_BoardHandle,
                                                2147483647);
```

Return value:

0: No error

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



10) i_PA2000_Set32DigitalOutputOff (...)**Syntax:**

<Return value> = v_PA2000_Set32DigitalOutputOff (BYTE b_BoardHandle,
LONG l_Value)

Parameters:**- Input:**

BYTE	b_BoardHandle	Handle of the PA 2000
LONG	l_Value	Output value (0 to 2 ³²)

- Output:

No output signal has occurred.

Task:

Resets one or several outputs of the **PA 2000** board.

Example:

l_Value = 55555555 Hex

Outputs 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31 are reset.

i

IMPORTANT!

**You can use this function only if the digital output memory is ON.
See function “i_PA2000_SetOutputMemoryOn (..)”.**

Calling convention:ANSI C:

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

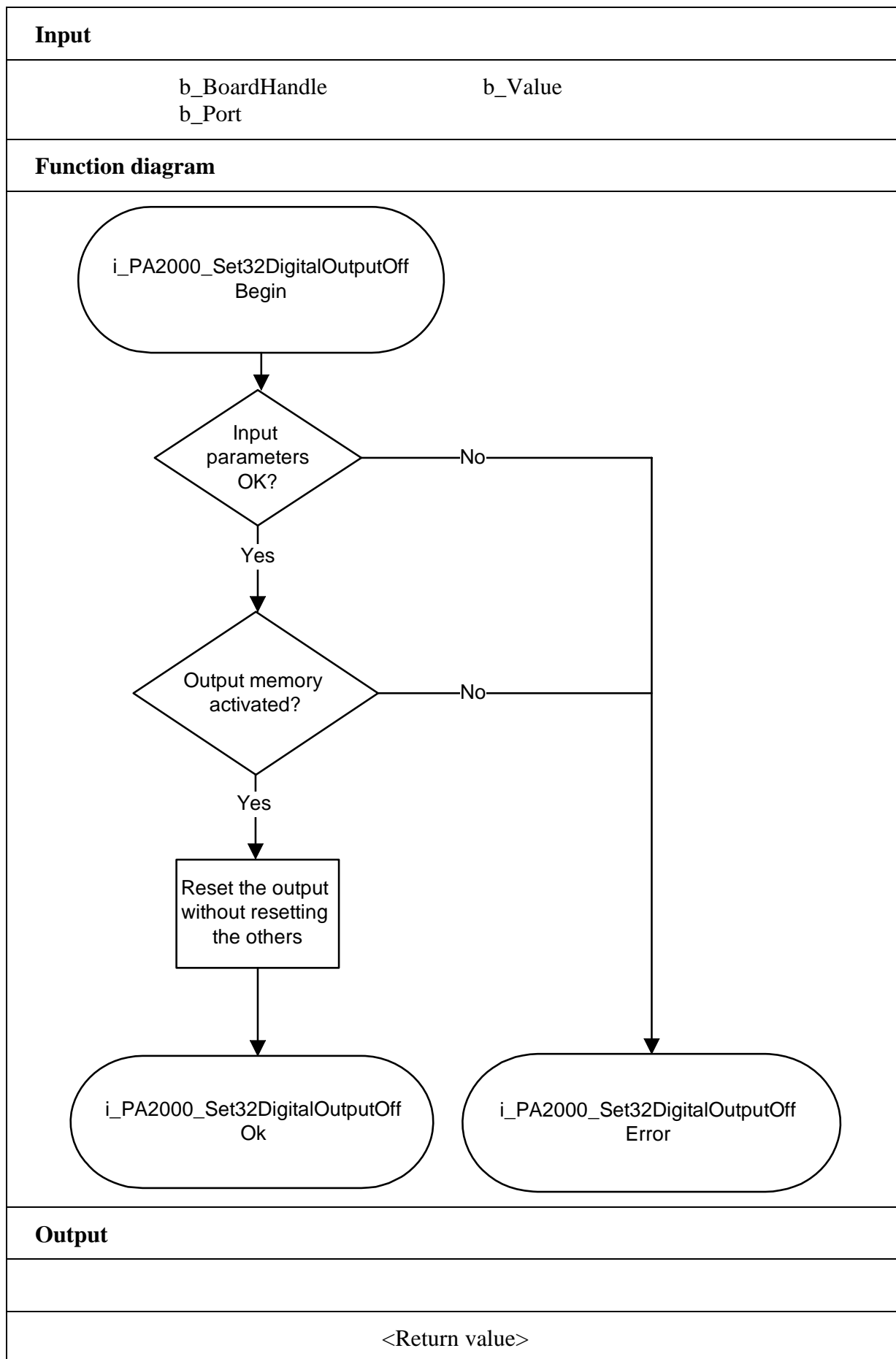
```
i_ReturnValue = i_PA2000_Set32DigitalOutputOff (b_BoardHandle,
                                                1073741823);
```

Return value:

0: No error

-1: The handle parameter of the board is wrong

-2: The digital output memory is OFF. Please first use the function
"i_PA2000_SetOutputMemoryOn"



9.6 Square wave generator

1) i_PA2000_InitSquareGenerator1 (...)

Syntax: <Return value> = i_PA2000_InitSquareGenerator1
 (BYTE_ b_BoardHandle,
 BYTE_ b_GeneratorMode,
 LONG_ l_FrequencyValue)

Parameter:

- Input:

BYTE_ b_BoardHandle
 BYTE_ b_GeneratorMode

Handle of the **PA 2000** board

Selects the operating mode of the first square wave generator

- PA2000_RATE_GENERATOR:

The first generator is used as a pulse generator (constant high state time, period and cyclic rate not constant)

- PA2000_BAUD_RATE_GENERATOR:

The first generator is used as a baud rate generator (constant cyclic rate, period and high rate time not constant).

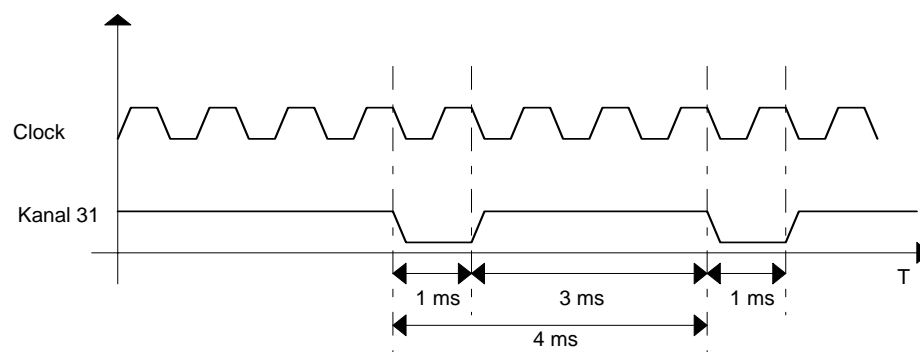
LONG_ l_FrequencyValue:

This parameter determines the output frequency of the square wave generator. The frequency basis is 1 MHz. The value must be between 16 and 500,000.

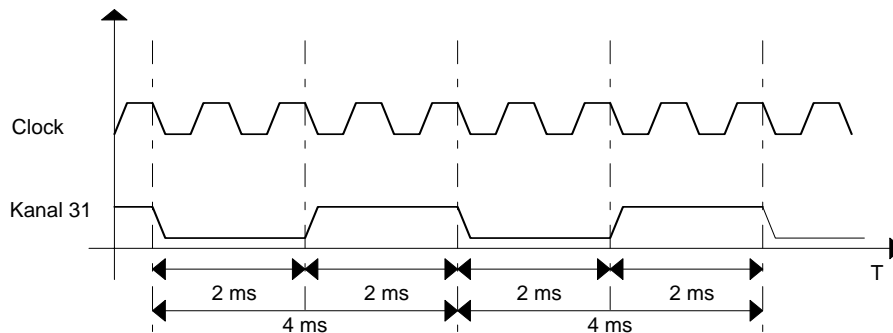
Example:

b_GeneratorMode = PA2000_RATE_GENERATOR

l_FrequencyValue = 250000 MHz



```
b_GeneratorMode = PA2000_BAUD_RATE_GENERATOR
l_FrequencyValue = 250000 MHz
```



- Output:
No output signal has occurred.

Task:

Selection of the operating mode for the first square wave generator.

You have to determine:

- whether the square wave generator is used as a pulse generator or as a baud rate generator
- the output frequency of the square wave generator.

Calling convention:

ANSI C:

```
int          i_ReturnValue;
unsigned char b_BoardHandle;

i_ReturnValue = i_PA2000_InitSquareGenerator1
                (b_BoardHandle,
                 PA2000_RATE_GENERATOR,
                 1000);
```

Return value:

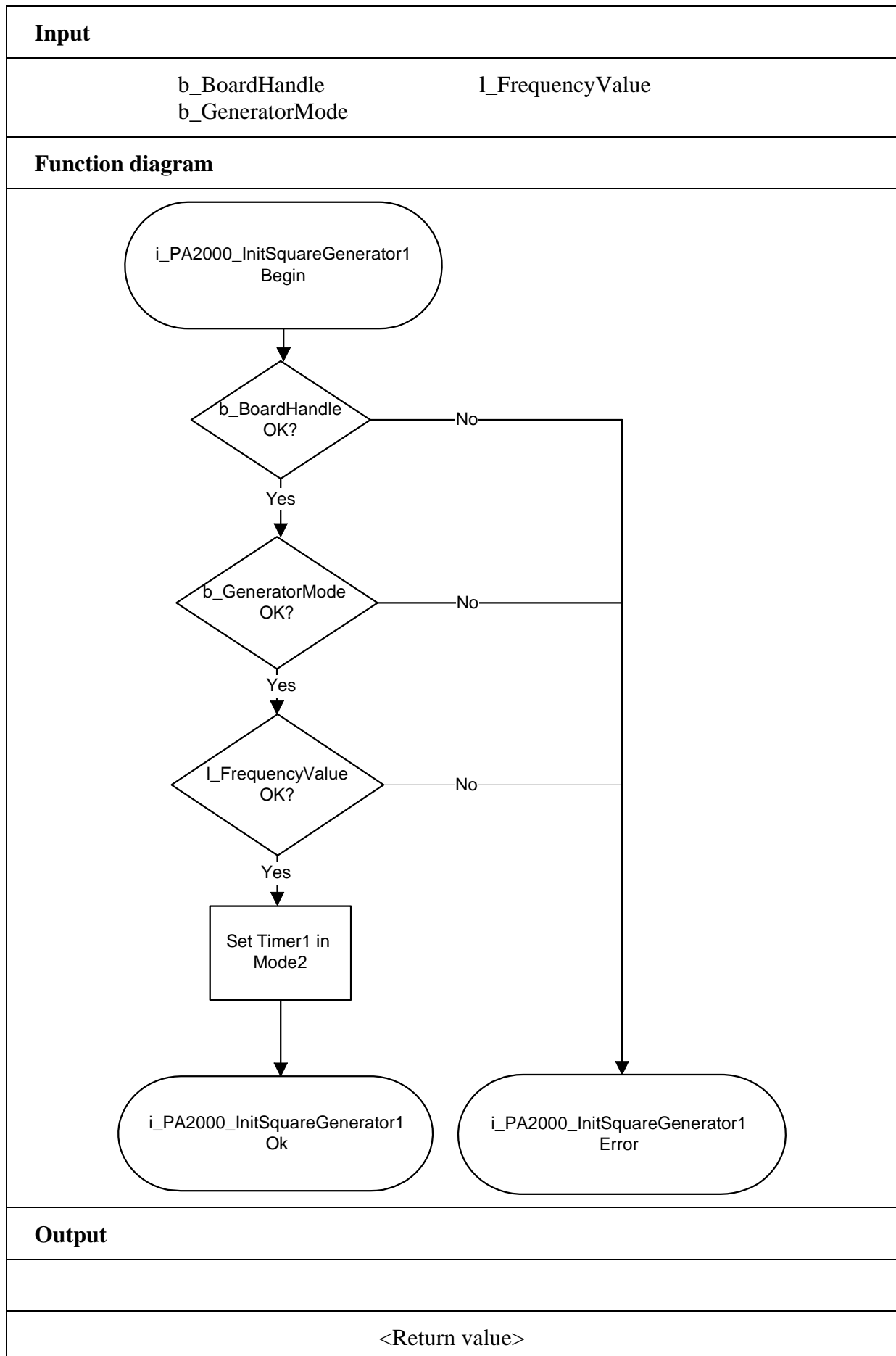
0: No error

-1: Wrong board handle parameter

-2: The value of the output frequency is not between 16 and 500,000

-3: Wrong operating mode parameter

(PA2000_RATE_GENERATOR or
PA2000_BAUD_RATE_GENERATOR)



2) i_PA2000_InitSquareGenerator2 (...)

Syntax:

Syntax: <Return value> = i_PA2000_InitSquareGenerator1
(BYTE_ b_BoardHandle,
BYTE_ b_GeneratorMode,
LONG_ l_FrequencyValue)

Parameter:

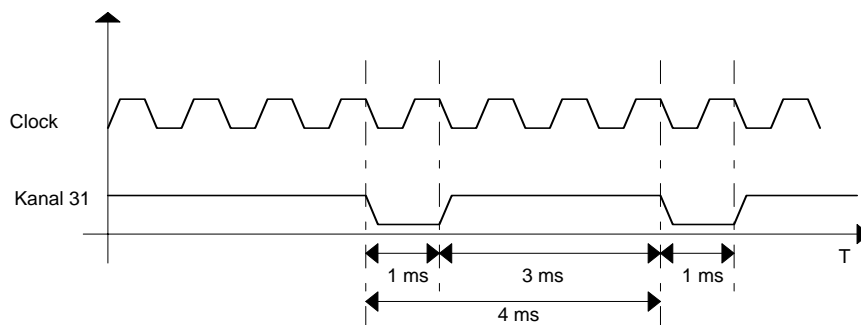
- Input:

BYTE_ b_BoardHandle
BYTE_ b_GeneratorMode

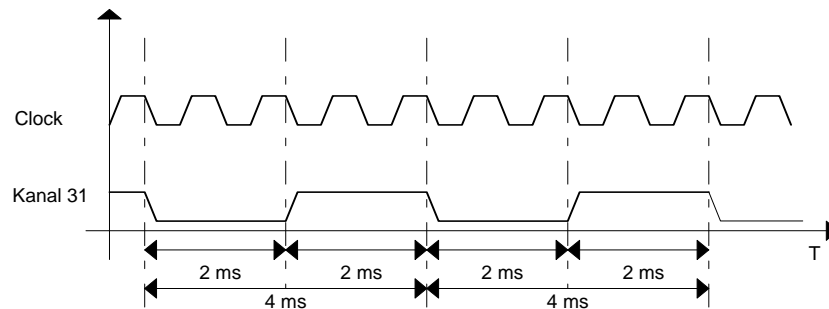
Handle of the **PA 2000** board
Selects the operating mode of the first square wave generator
- PA2000_RATE_GENERATOR:
The first generator is used as a pulse generator (constant high state time, period and cyclic rate not constant)
- PA2000_BAUD_RATE_GENERATOR:
The first generator is used as a baud rate generator (constant cyclic rate, period and high rate time not constant).
LONG_ l_FrequencyValue: This parameter determines the output frequency of the square wave generator. The frequency basis is 1 MHz. The value must be between 16 and 500,000.

Example:

b_GeneratorMode = PA2000_RATE_GENERATOR
l_FrequencyValue = 250000 MHz



```
b_GeneratorMode = PA2000_BAUD_RATE_GENERATOR
l_FrequencyValue = 250000 MHz
```



- Output:

No output signal has occurred.

Task:

Selecting the operating mode of the second square wave generator.

You have to determine:

- whether the square wave generator is used as a pulse generator or as a baud rate generator
- the output frequency of the square wave generator.

Calling convention:

ANSI C :

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

```
i_ReturnValue = i_PA2000_InitSquareGenerator2
                (b_BoardHandle,
                 PA2000_BAUD_RATE_GENERATOR,
                 1000);
```

Return value:

0: No error

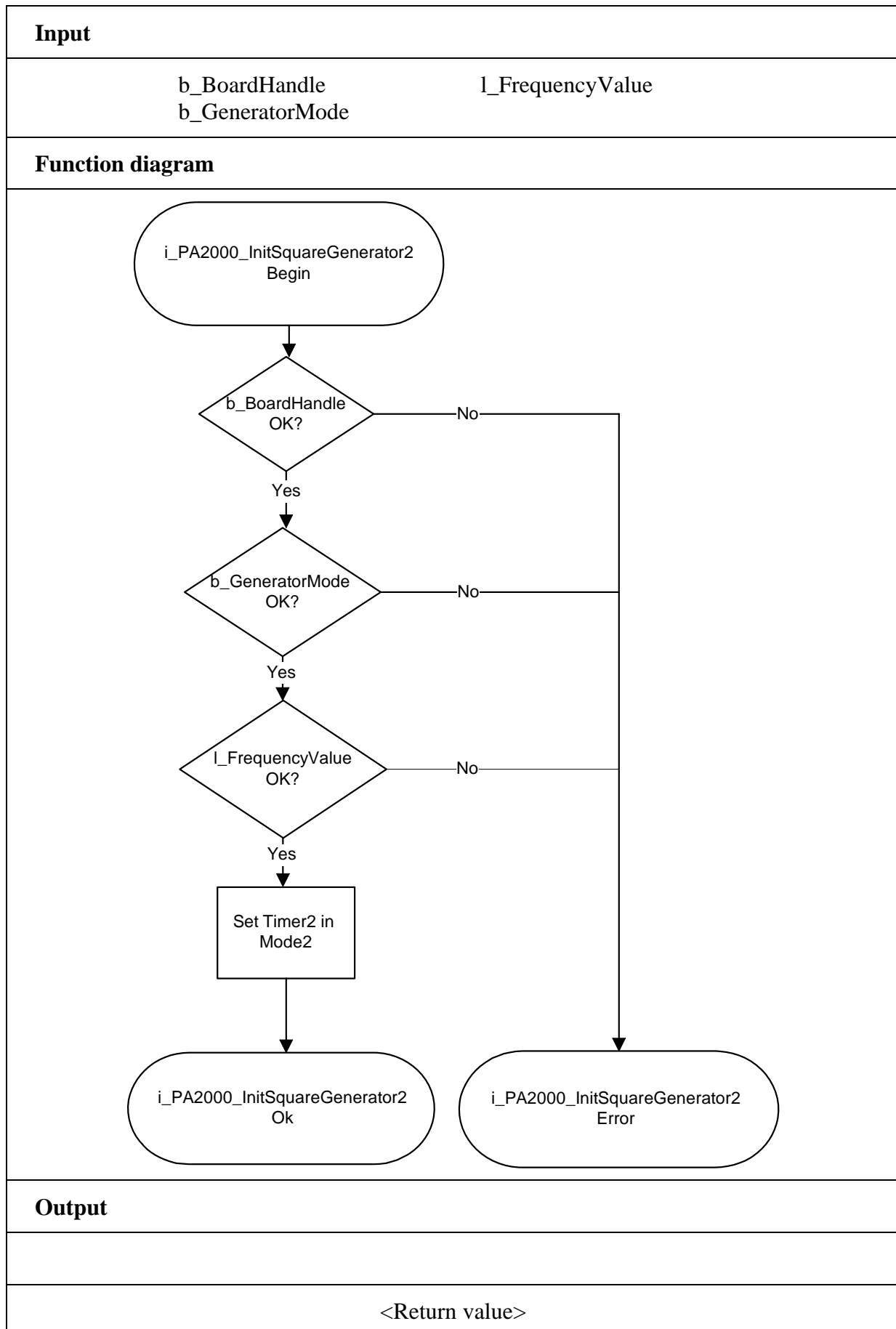
-1: Wrong board handle parameter

-2: Value of the output frequency is not between 16 and 500,000

-3: Wrong operating mode parameter

(PA2000_RATE_GENERATOR or

PA2000_BAUD_RATE_GENERATOR)



3) i_PA2000_StartSquareGenerator1(...)

Syntax:

<Return value> = i_PA2000_StartSquareGenerator1
(BYTE_ b_BoardHandle)

Parameter:**- Input:**

BYTE_ b_BoardHandle Handle of the PA 2000 board

- Output:

No output signal has occurred.

Task:

Starts the first square wave generator after it has been initialised by "i_PA2000_InitSquareGenerator1".

Calling convention:ANSI C:

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

```
i_ReturnValue = i_PA2000_StartSquareGenerator1 (b_BoardHandle);
```

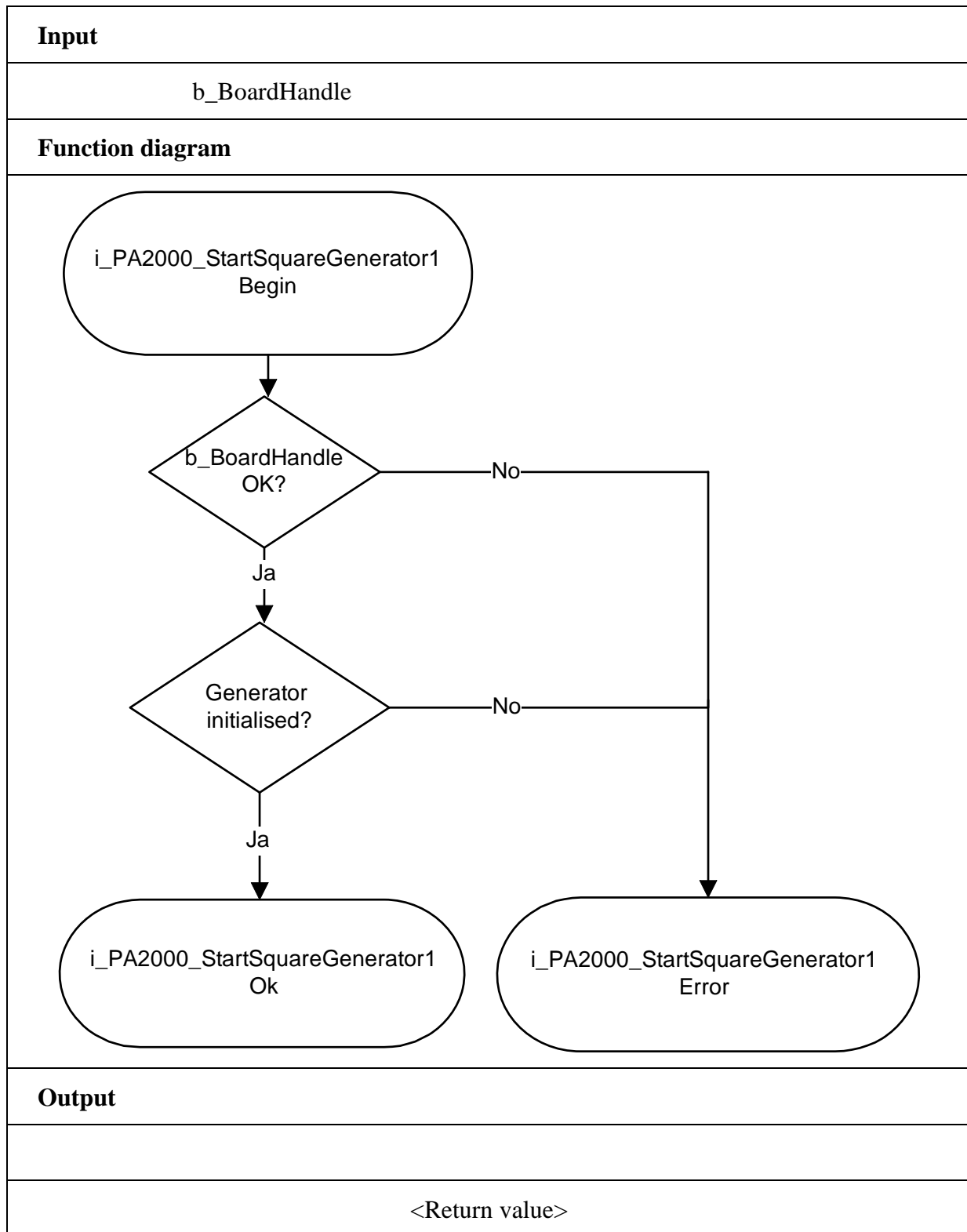
Return value:

0: No error

-1: Wrong board handle parameter

-2: The square wave generator was not initialised.

"i_PA2000_InitSquareGenerator1"



4) i_PA2000_StartSquareGenerator2(...)

Syntax:

<Return value> = i_PA2000_StartSquareGenerator2
(BYTE_ b_BoardHandle)

Parameter:**- Input:**

BYTE_ b_BoardHandle Handle of the **PA 2000** board

- Output:

No output signal has occurred.

Task:

Starts the second square wave generator after it has been initialised by
"i_PA2000_InitSquareGenerator2"

Calling convention:ANSI C:

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

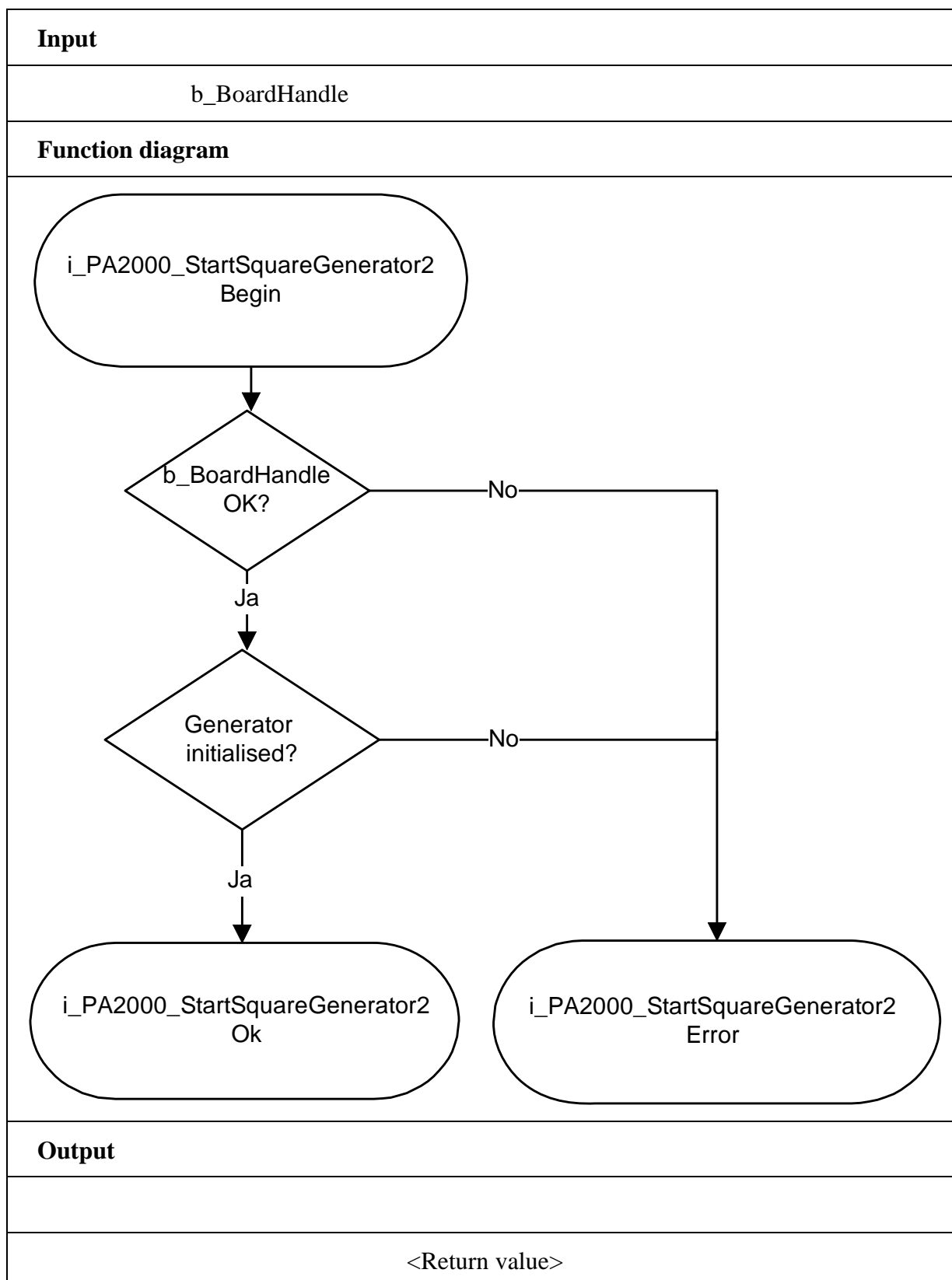
```
i_ReturnValue = i_PA2000_StartSquareGenerator2 (b_BoardHandle);
```

Return value:

0 : No error

-1: Wrong board handle parameter

-2: Square wave generator was not initialised. "i_PA2000_InitSquareGenerator2"



5) i_PA2000_StopSquareGenerator1 (...)

Syntax:

<Return value> = i_PA2000_StopSquareGenerator1
(BYTE_ b_BoardHandle)

Parameter:**- Input:**

BYTE_ b_BoardHandle Handle of the **PA 2000** board

- Output:

No output signal has occurred.

Task:

Stops the first square wave generator. Its value is kept. It has the same influence as a hardware gate.

ANSI C :

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

```
i_ReturnValue = i_PA2000_StopSquareGenerator1 (b_BoardHandle);
```

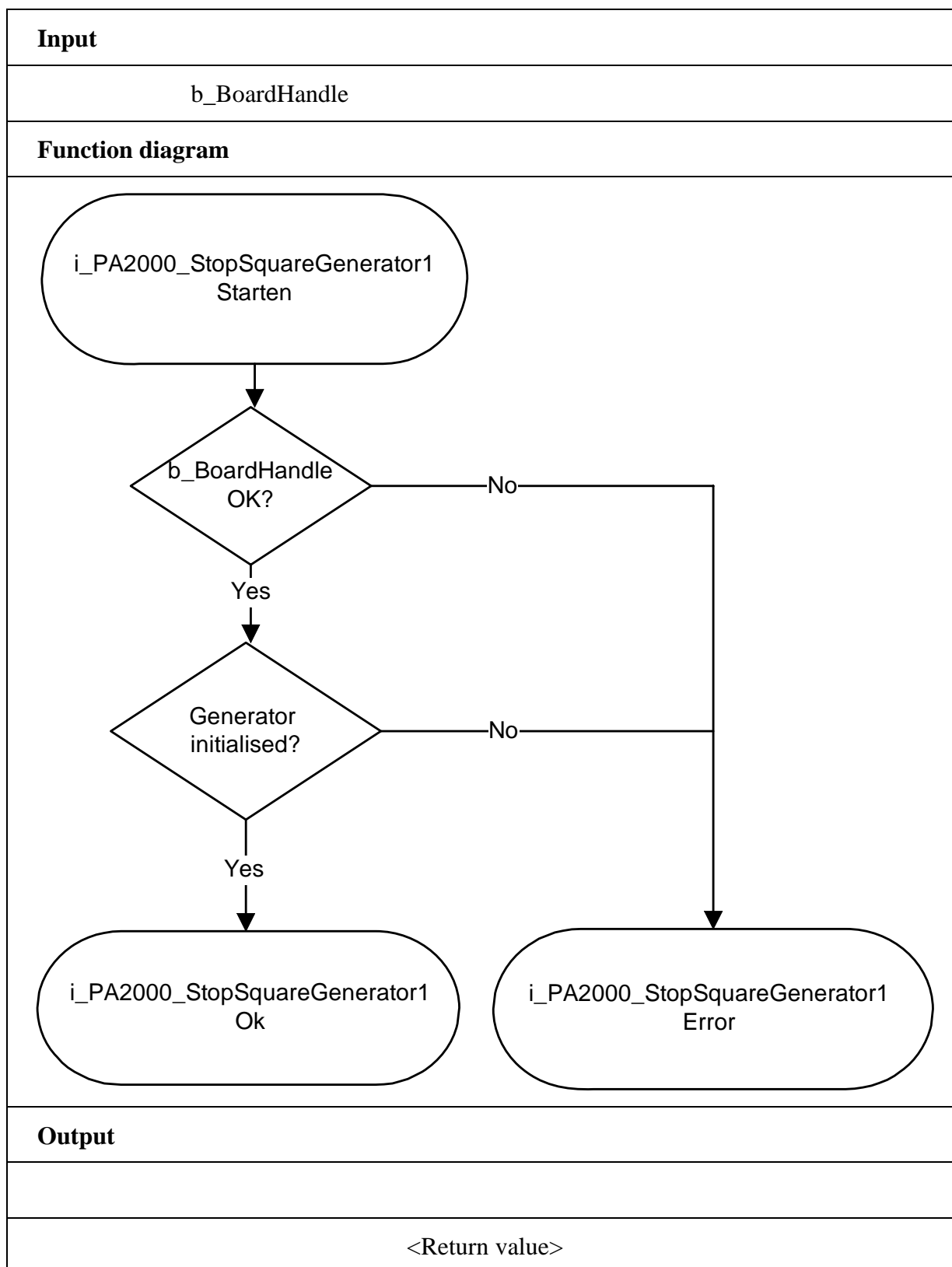
Return value:

0: No error

-1: Wrong board handle parameter

-2: Square wave generator was not initialised.

Use "i_PA2000_InitSquareGenerator1"



6) i_PA2000_StopSquareGenerator2 (...)

Syntax:

<Return value> = i_PA2000_StopSquareGenerator2
(BYTE_ b_BoardHandle)

Parameter:

- Input:

BYTE_ b_BoardHandle Handle of the **PA 2000** board

- Output:

No output signal has occurred.

Task:

Stops the second square wave generator. Its value is kept. It has the same influence as a hardware gate.

Calling convention:

ANSI C:

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

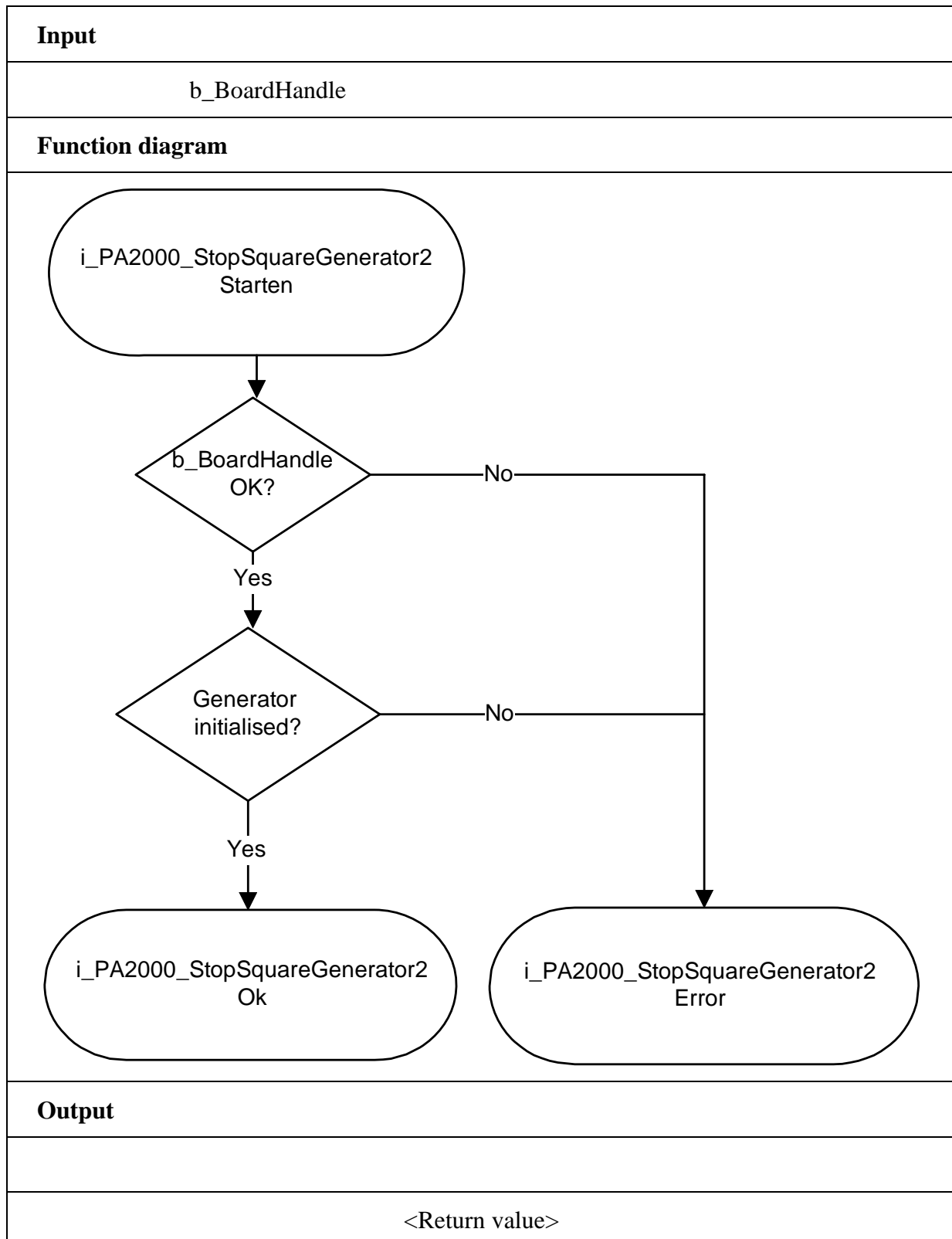
```
i_ReturnValue = i_PA2000_StopSquareGenerator2 (b_BoardHandle);
```

Return value:

0: No error

-1: Wrong board handle parameter

-2: Square wave generator was not initialised. Use
"i_PA2000_InitSquareGenerator2"



7) i_PA2000_ReadSquareGenerator1 (...)**Syntax:**

```
<Return value> = i_PA2000_ReadSquareGenerator1
                                     (BYTE_ b_BoardHandle,
                                     PLONG_ pl_ReadValue)
```

Parameter:**- Input:**

BYTE_ b_BoardHandle	Handle of the PA 2000 board
---------------------	-----------------------------

- Output:

PLONG_ pl_ReadValue	This parameter returns the current value of the square wave generator.
---------------------	--

Task:

Reads the current value of the first square wave generator

Calling convention:ANSI C :

```
int          i_ReturnValue;
unsigned char b_BoardHandle;
unsigned long l_ReadValue;
```

```
i_ReturnValue = i_PA2000_ReadSquareGenerator1 (b_BoardHandle
                                                &l_ReadValue);
```

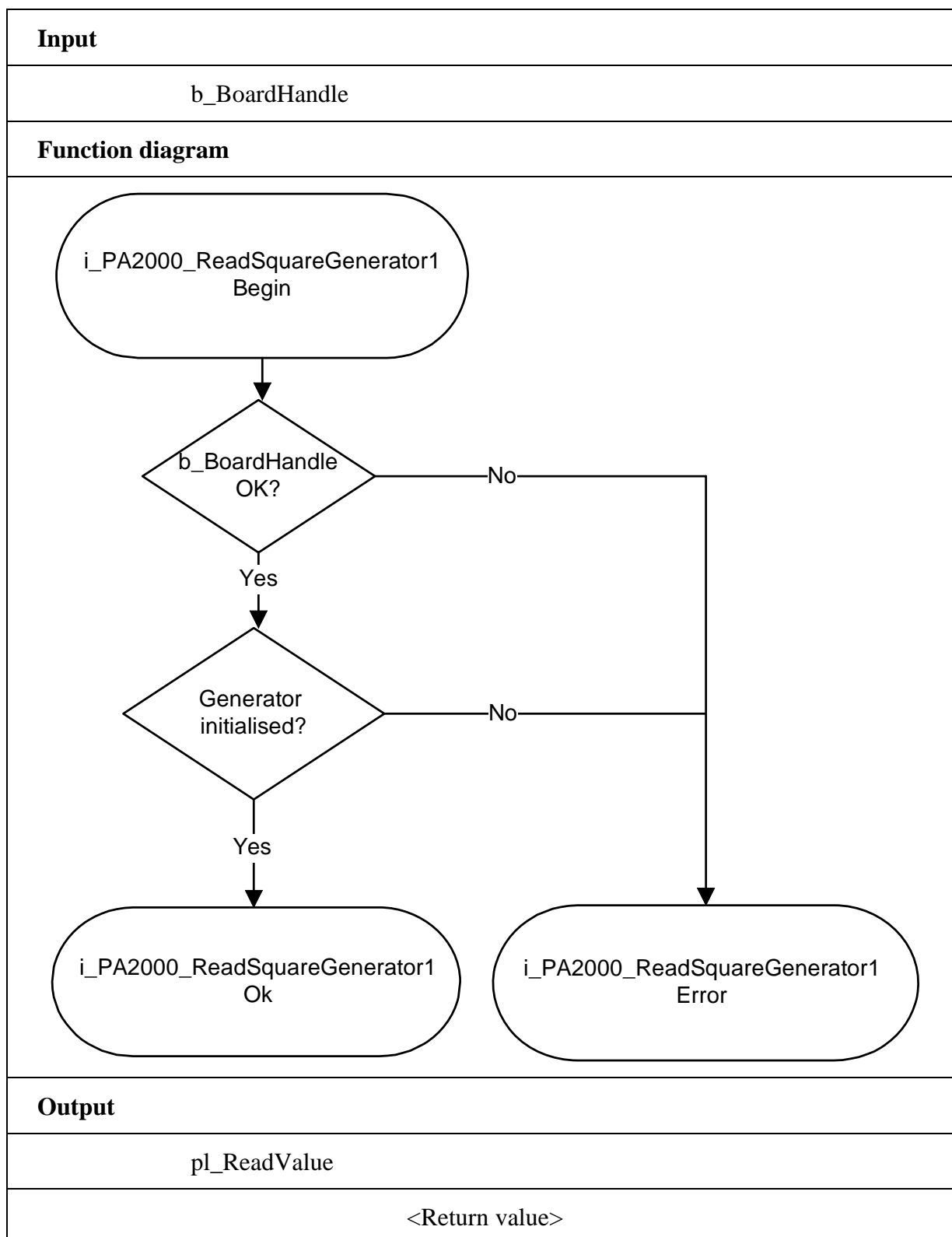
Return value:

0: No error

-1: Wrong board handle parameter

-2: Square wave generator was not initialised.

Use "i_PA2000_InitSquareGenerator1"



8) i_PA2000_ReadSquareGenerator2 (...)**Syntax:**

```
<Return value> = i_PA2000_ReadSquareGenerator2
                                (BYTE_ b_BoardHandle,
                                PLONG_ pl_ReadValue)
```

Parameter:**- Input:**

BYTE_ b_BoardHandle	Handle of the PA 2000 board
PLONG_ pl_ReadValue	This parameter returns the current value of the square wave generator

- Output:

No output signal has occurred.

Task:

Reads the current value of the second square wave generator

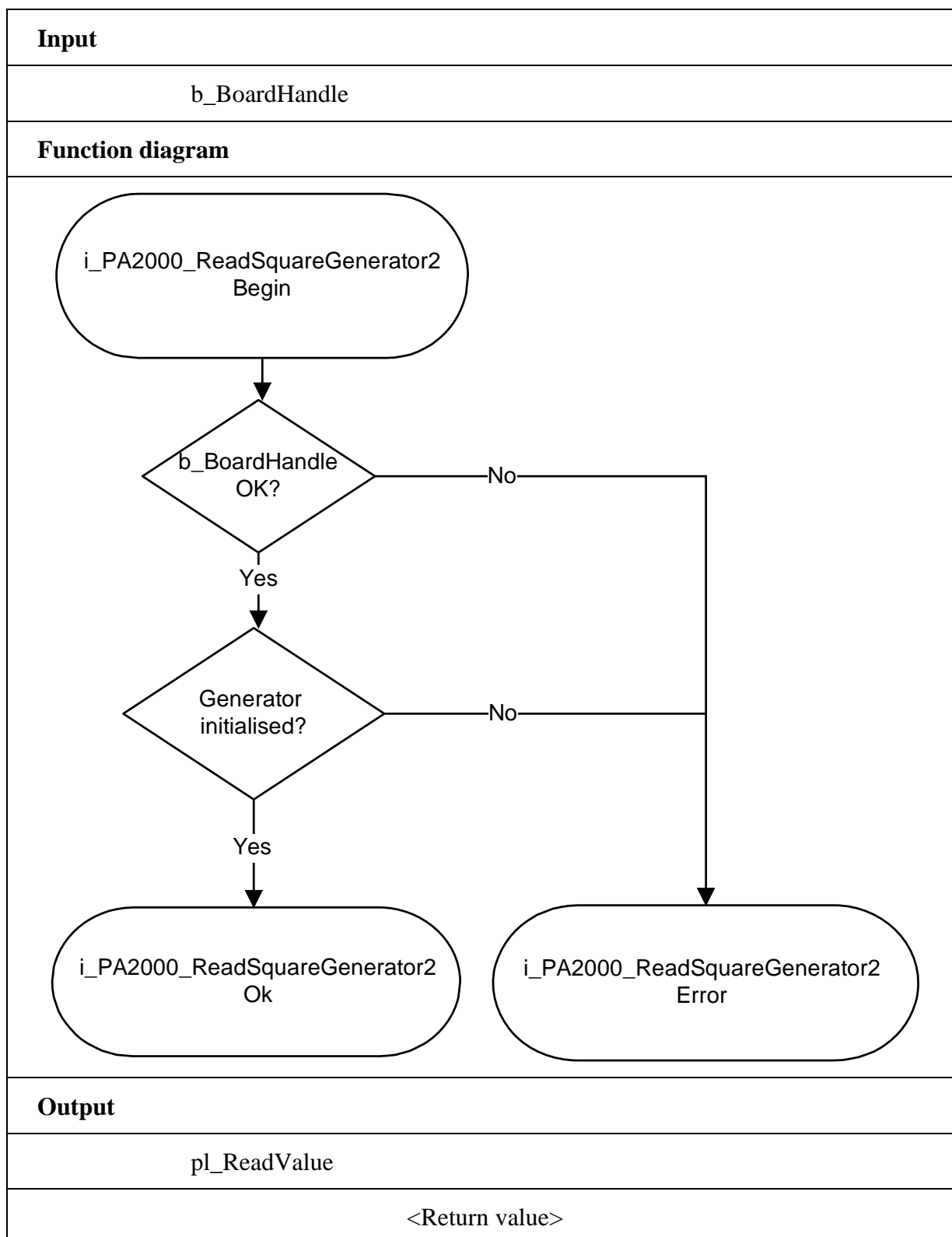
Calling convention:ANSI C :

```
int          i_ReturnValue;
unsigned char b_BoardHandle;
unsigned long l_ReadValue;
```

```
i_ReturnValue = i_PA2000_ReadSquareGenerator2 (b_BoardHandle
                                                &l_ReadValue);
```

Return value:

0: No error
 -1: Wrong board handle parameter
 -2: Square wave generator was not initialised.
 Use "i_PA2000_InitSquareGenerator2"



9.7 Timer/watchdog

1) i_PA2000_InitTimerWatchdog (...)

Syntax:

```
<Return value> = i_PA2000_InitTimerWatchdog
                    (BYTE_ b_BoardHandle,
                     BYTE_ b_TimerWatchdogMode,
                     LONG_ l_TimingValue,
                     BYTE_ b_InterruptHandling)
```

Parameter:

- Input:

BYTE_ b_BoardHandle	Handle of the PA 2000 board
BYTE_ b_TimerWatchdogMode	Selecting the operating mode of the timer/watchdog: - PA2000_TIMER: The timer/watchdog is used as a timer. - PA2000_WATCHDOG: The timer/watchdog is used as a watchdog.
LONG_ l_TimingValue:	This parameter determines the period of time of the timer/watchdog. If the timer/watchdog is used as a timer, this parameter returns the interval between two interrupts. If the timer/watchdog is used as a watchdog this parameter returns the time lapse of the watchdog. The values must be between 2ms and 65535 ms.
BYTE_ b_InterruptHandling:	An interrupt can be generated when the watchdog/timer has run down. With this parameter the user determines, whether he/she wants to use the interrupts or not. - PA200_ENABLE: Interrupts enabled. - PA200_DISABLE: Interrupts disabled.

- Output:

No output signal has occurred.

Task:

Selects the operating mode of the timer/watchdog.

If you initialise the timer/watchdog as a timer, it starts immediately after the initialisation.

You have to determine:

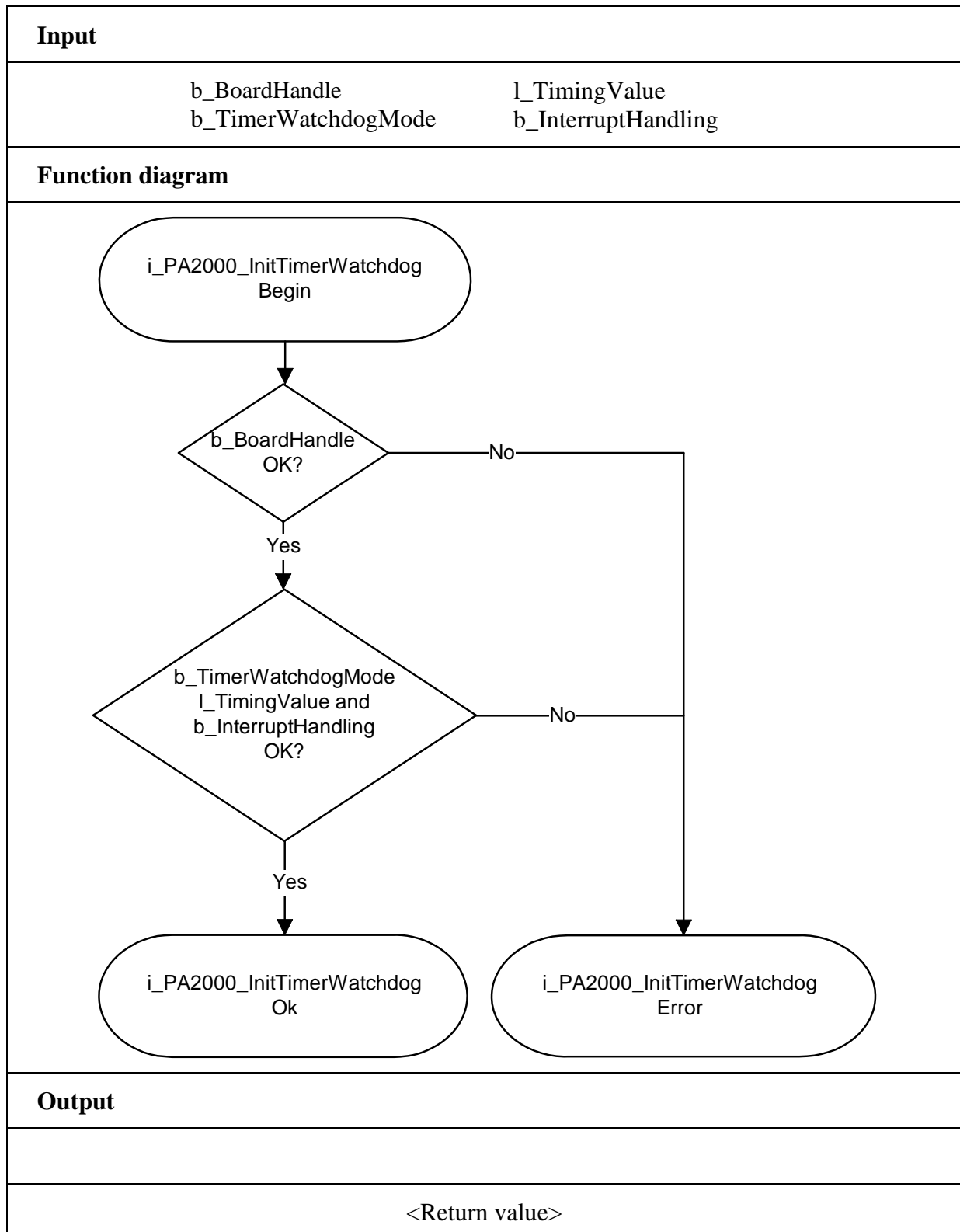
- whether the timer/watchdog is used as a timer or as a watchdog
- the of the timer/watchdog.

Calling convention:ANSI C:

```
int          i_ReturnValue;  
unsigned char b_BoardHandle;  
i_ReturnValue = i_PA2000_InitTimerWatchdog (b_BoardHandle,  
                                             PA2000_WATCHDOG,  
                                             1500,  
                                             PA2000_DISABLE);
```

Return value:

- 0: No error
- 1: Wrong board handle parameter
- 2: Value of the time interval/ time lapse is not between 2 and 65535.
- 3: Wrong operating mode parameter
(PA2000_TIMER or PA2000_WATCHDOG)
- 4: Wrong interrupt parameter
- 5: User interrupt routine is not installed



2) i_PA2000_StartTimerWatchdog(...)

Syntax:

<Return value> = i_PA2000_StartTimerWatchdog
(BYTE_ b_BoardHandle)

Parameter:**- Input:**

BYTE_ b_BoardHandle Handle of the PA 2000 board

- Output:

No output signal has occurred.

Task:

Starts the timer/watchdog after it has been initialised with
"i_PA2000_InitTimerWatchdog".

Calling convention:ANSI C:

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

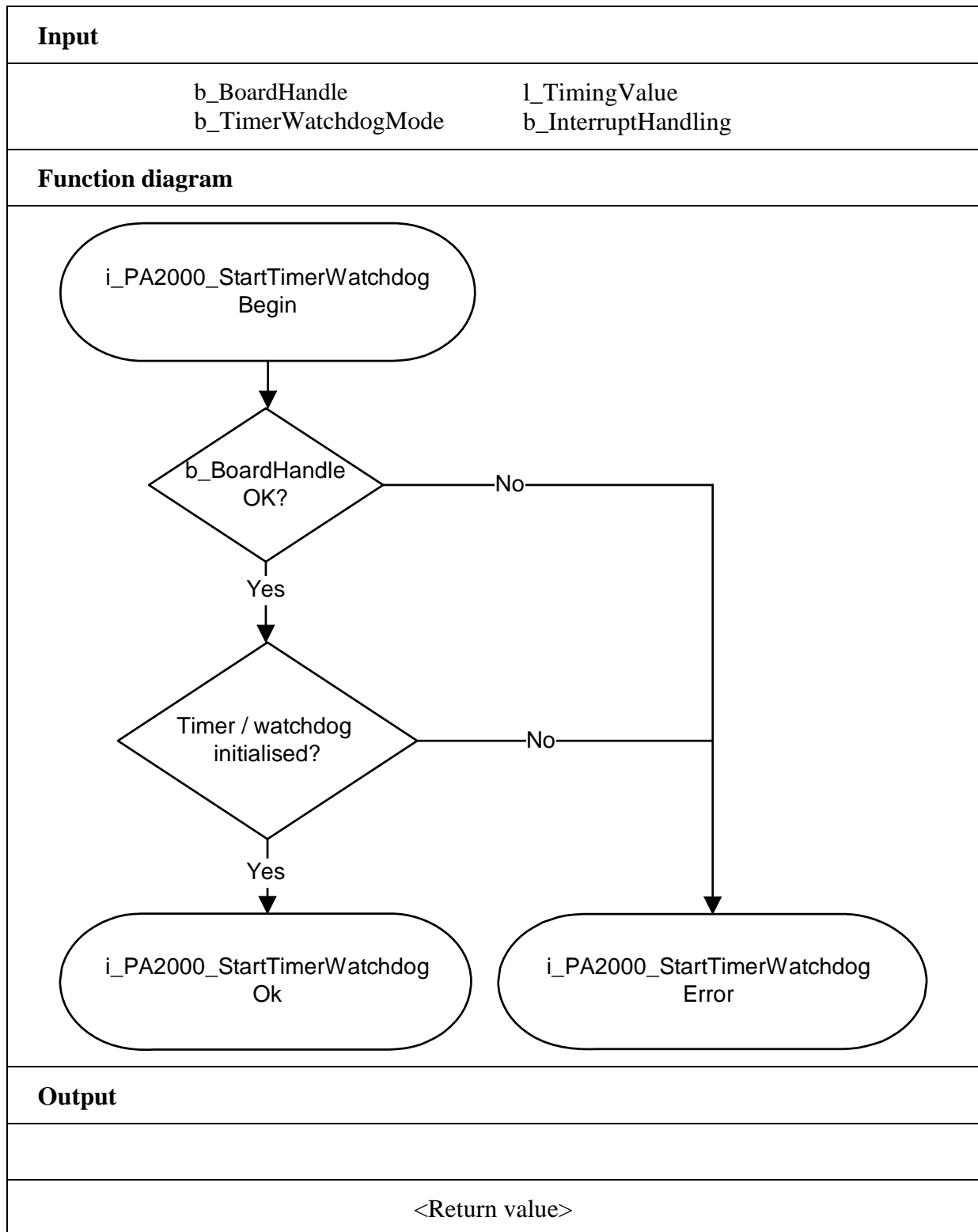
```
i_ReturnValue = i_PA2000_StartTimerWatchdog (b_BoardHandle);
```

Return value:

0: No error

-1: Wrong board handle parameter

-2: Timer watchdog was not initialised. "i_PA2000_InitTimerWatchdog"



3) i_PA2000_StopTimerWatchdog(...)

Syntax:

<Return value> = i_PA2000_StopTimerWatchdog
(BYTE_ b_BoardHandle)

Parameter:**- Input:**

BYTE_ b_BoardHandle Handle of the PA 2000 board

- Output:

No output signal has occurred.

Task:

Stops the timer/watchdog after it has been initialised by
"i_PA2000_InitTimerWatchdog".

Calling convention:ANSI C:

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

```
i_ReturnValue = i_PA2000_StopTimerWatchdog (b_BoardHandle);
```

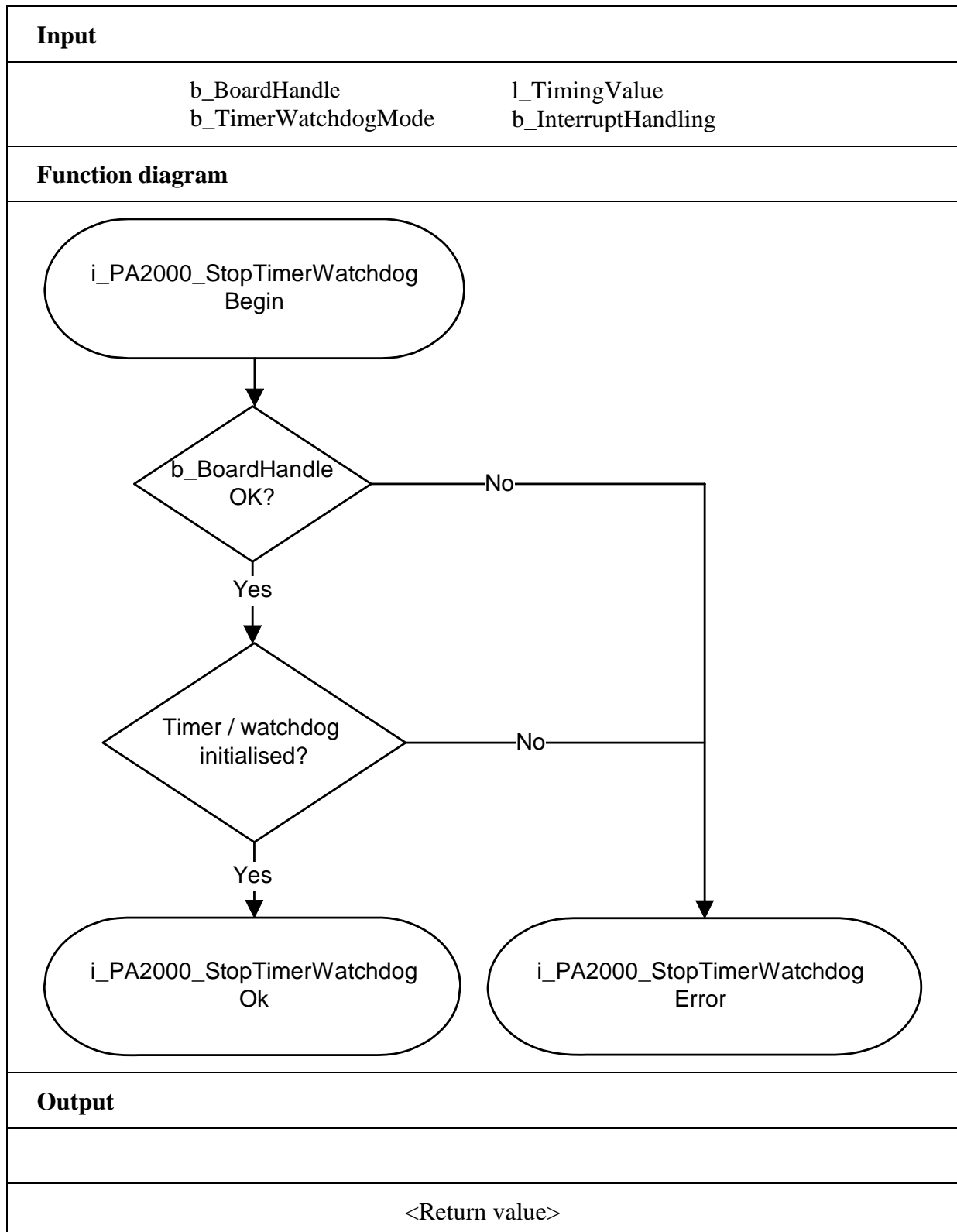
Return value:

0: No error.

-1: Wrong board handle parameter.

-2: Timer/watchdog was not initialised.

"i_PA2000_InitTimerWatchdog"



INDEX

ADDIREG

- changing the configuration 20
- removing 21

board

- inserting 13
- physical set-up 4
- plugging 14
- slot 4

connection cable 1

diagnosis

- outputs 34

DIP switches 10

EMC 4

enable

- interrupts 35
- Timer0 35
- Timer1 35

frequency, output 34

intended purpose of the board 1

Internet

- error analysis 21

interrupt 35

- enable 35

interrupt lines, possible 35

interrupt sources 35

IRQ line see interrupt line

output frequency 34

outputs

- diagnosis 34
- set 33

outputs 31 and 32

- generators of square wave signals 34

PC

- opening 13

slot

- types 13

software

- installation 15

square wave signals, generators of outputs 31 and 32 34

supervise the PC or software

- Timer2 36

timer 35–36

Timer0

- enable 35

Timer1

- enable 35

Timer2

- supervise the PC or software 36

use, limits of 1