



Technical description

ADDIALOG PA 3000

Standard software

1	INTRODUCTION.....	1
2	DIN 66001- GRAPHICAL SYMBOLS.....	5
3	SOFTWARE FUNCTIONS (API).....	6
3.1	Initialisation.....	6
	1) i_PA3000_InitCompiler (..)	6
	2) i_PA3000_SetBoardInformation (...).....	8
	3) i_PA3000_SetBoardInformationWin32 (...).....	10
	4) i_PA3000_GetHardwareInformation (...).....	12
	5) i_PA3000_CloseBoardHandle	14
3.2	Interrupt.....	16
	1) i_PA3000_SetBoardIntRoutineDos (..)	16
	2) i_PA3000_SetBoardIntRoutineVBdos (..).....	20
	3) i_PA3000_SetBoardIntRoutineWin16.....	23
	4) i_PA3000_SetBoardIntRoutineWin32 (..).....	26
	5) i_PA3000_TestInterrupt (..).....	32
	6) i_PA3000_ResetBoardIntRoutine (..).....	35
3.3	Direct conversion of the analog input channels	37
	1) i_PA3000_Read1AnalogInput (...)	37
	2) i_PA3000_ReadAllAnalogInput (...)	41
3.4	Cyclic conversion of analog input channels.....	43
	1) i_PA3000_InitAnalogInputAcquisition (...)	43
	2) i_PA3000_StartAnalogInputAcquisition(...)	54
	3) i_PA3000_StopAnalogInputAcquisition(...).....	57
	4) i_PA3000_ClearAnalogInputAcquisition(...)	60
3.5	Timer 2.....	62
	1) i_PA3000_InitTimer2 (...).....	62
	2) i_PA3000_StartTimer2 (...).....	64
	3) i_PA3000_StopTimer2 (...).....	66
	4) i_PA3000_ReadTimer2 (...).....	68
	5) i_PA3000_WriteTimer2 (...).....	70
3.6	Digital input channels.....	72
	1) i_PA3000_Read1DigitalInput (...)	72
	2) i_PA3000_Read4DigitalInput (...)	74
3.7	Digital output channels.....	76
	1) i_PA3000_SetOutputMemoryOn (...)	76
	2) i_PA3000_SetOutputMemoryOff (...)	78
	3) i_PA3000_Set1DigitalOutputOn (...).....	80
	4) i_PA3000_Set1DigitalOutputOff (...).....	82
	5) i_PA3000_Set4DigitalOutputOn (...).....	84
	6) i_PA3000_Set4DigitalOutputOff (...).....	86
3.8	Functions to be used in Kernel mode	88
	1) i_PA3000_KRNL_Read1DigitalInput (...)	88
	2) i_PA3000_KRNL_Read4DigitalInput (...)	90
	3) i_PA3000_KRNL_Set1DigitalOutputOn (...).....	92
	4) i_PA3000_KRNL_Set4DigitalOutputOn (...).....	94

Tables

Table 1-1: Type Declaration for Dos and Windows 3.1X	1
Table 1-2: Type Declaration for Windows 95/NT	2
Table 1-3: Define value	3
Table 3-1: Values returned for the analog inputs.....	17
Table 3-2: Interrupt mask	17
Table 3-3: Selection of the analog input channels.....	46
Table 3-4: Gain selection	46
Table 3-5 Selection of the input voltage range	47

1 INTRODUCTION



IMPORTANT!

Note the following conventions in the text:

Function: "i_PA3000_SetBoardInformation"
 Variable: *ui_Address*

Table 1-1: 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 1-2: Type Declaration for Windows 95/NT

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

Table 1-3: 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
PA3000_DISABLE	0	0
PA3000_ENABLE	1	1
PA3000_CHANNEL0	0	0
PA3000_CHANNEL1	1	1
PA3000_CHANNEL2	2	2
PA3000_CHANNEL3	3	3
PA3000_CHANNEL4	4	4
PA3000_CHANNEL5	5	5
PA3000_CHANNEL6	6	6
PA3000_CHANNEL7	7	7
PA3000_CHANNEL8	8	8
PA3000_CHANNEL9	9	9
PA3000_CHANNEL10	10	A
PA3000_CHANNEL11	11	B
PA3000_CHANNEL12	12	C


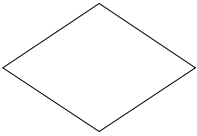
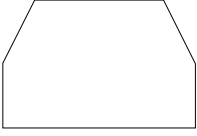

Define name	Decimal value	Hexadecimal value
PA3000_CHANNEL13	13	D
PA3000_CHANNEL14	14	E
PA3000_CHANNEL15	15	F
PA3000_1_GAIN	0	0
PA3000_2_GAIN	16	10
PA3000_5_GAIN	32	20
PA3000_10_GAIN	48	30
PA3000_20_GAIN	80	50
PA3000_50_GAIN	96	60
PA3000_100_GAIN	0	0
PA3000_200_GAIN	16	10
PA3000_500_GAIN	32	20
PA3000_1000_GAIN	48	30
PA3000_UNIPOLAR	128	80
PA3000_BIPOLAR	0	0
PA3000_SIMPLE_MODUS	0	0
PA3000_DELAY_MODUS	1	1
PA3000_DMA_NOT_USED	0	0
PA3000_DMA_USED	1	1
PA3000_SINGLE	0	0
PA3000_CONTINUOUS	1	1

2 DIN 66001- GRAPHICAL SYMBOLS

This chapter describes all software functions (API) necessary for the operation of the PA 3000 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.

	<p>Process, general (including inputs and outputs)</p>
	<p>Decision Selection unit (eg.: switch)</p>
	<p>Loop limit Beginning</p>
	<p>Loop limit End</p>

3 SOFTWARE FUNCTIONS (API)

3.1 Initialisation

1) i_PA3000_InitCompiler (..)

Syntax:

<Return value> = i_PA3000_InitCompiler (BYTE b_CompilerDefine)

Parameters:

- Input:

BYTE b_CompilerDefine	The user has to choose the language (under Windows) in which he/she wants to program
	- DLL_COMPILER_C: The user programs in C
	- DLL_COMPILER_VB: The user programs in Visual Basic for Windows
	- DLL_COMPILER_VB5: The user programs in Visual Basic 5 for Windows
	- DLL_COMPILER_PASCAL: The user programs in Pascal
	- DLL_LABVIEW: The user programs in Labview

-Output:

No output signal has occurred

Task:

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



WICHTIG!

This function is only available with a Windows environment.

Calling convention:

ANSI C:

```
int i_ReturnValue;
```

```
i_ReturnValue = i_PA3000_InitCompiler (DLL_COMPILER_C);
```

Return value:

0: No error

-1: Compiler parameter is wrong

Input
b_CompilerDefine
Function diagram
<pre> graph TD Start([i_PA3000_InitCompiler Begin]) --> Decision{b_CompilerDefine correct?} Decision -- Yes --> Process[Save b_CompilerDefine] Decision -- No --> EndError([i_PA3000_InitCompiler end Error]) Process --> EndOk([i_PA3000_InitCompiler end Ok]) </pre>
Output
<Return Value>

i**IMPORTANT!**

This function is only available for DOS and Windows 3.11 applications

2) i_PA3000_SetBoardInformation (...)**Syntax:**

```
<Return value> = i_PA3000_SetBoardInformation
                    (UINT   ui_Address,
                     BYTE   b_InterruptNbr,
                     BYTE   b_DMACHannelNbr1,
                     BYTE   b_DMACHannelNbr2,
                     BYTE   b_AnalogInputChannelNbr,
                     PBYTE  pb_BoardHandle)
```

Parameters:**- Input:**

UINT	ui_Address	Base address of board PA 3000
BYTE	b_InterruptNbr	Interrupt line of the board (IRQ3, 5, 9, 10, 11, 12, 14, 15) If 0, no interrupt line is used
BYTE	b_DMACHannelNbr1	Number of the first DMA channel (DMA5, 6 or 7). If 0, DMA is not used.
BYTE	b_DMACHannelNbr2	Number of the second DMA channel (DMA5, 6 or 7). If 0, DMA is not used.
BYTE	b_AnalogInputChannelNbr	Number of the analog input channels (1 to 16)

- Output:

PBYTE	pb_BoardHandle	Handle ¹ of board PA 3000 to use the functions
-------	----------------	--

Task:

Verifies if board **PA 3000** is present. Stores the following information:

- base address,
- the interrupt number
- the number of the first DMA channel
- the number of the second DMA channel
- and the number of analog input channels.

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

Handles allow to operate several boards.

Calling convention:ANSI C:

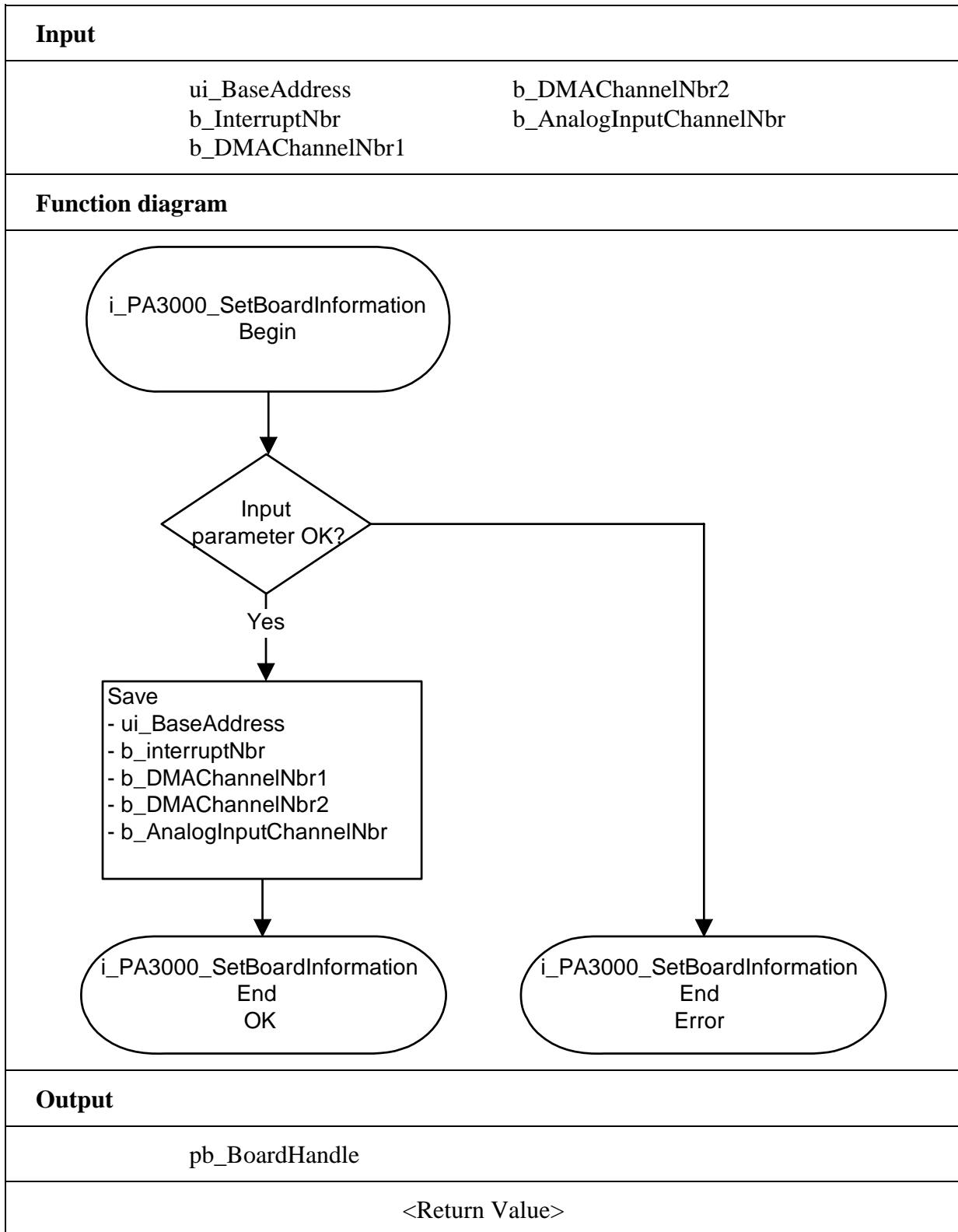
```
int          i_ReturnValue;
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_PA3000_SetBoardInformation (0x390, 0,0,0 16,
                                              &b_BoardHandle);
```

¹ Identification number of the board

Return value:

- 0: No error
- 1: Board not present
- 2: Number of the DMA channel is wrong
- 3: IRQ number is wrong
- 4: Number of analog input channels is wrong
- 5: 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 applications

3) i_PA3000_SetBoardInformationWin32 (...)**Syntax:**

```
<Return value> = i_PA3000_SetBoardInformationWin32
                                (PCHAR      pc_Identifier
                                BYTE        b_AnalogInputChannelNbr
                                PBYTE      pb_BoardHandle)
```

Parameters:**- Input:**

PCHAR pc_Identifier Identifier string for the selection of **PA 3000**
 The identifier string is determined by the
 ADDIREG registration program.

BYTE b_AnalogInputChannelNbr
 Number of analog inputs (0 to 16)

- Output:

PBYTE pb_BoardHandle Handle of the board PA 3000 to use the functions

Task:

Call up all hardware information about the **PA3000** given by the ADDIREG registration program and stores the following information:

- the base address,
- the interrupt number,
- the first DMA channel number,
- the second DMA channel number,
- the number of analog inputs

Then verifies if board **PA3000** is present.

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

Handles allow to operate several boards.

Calling convention:

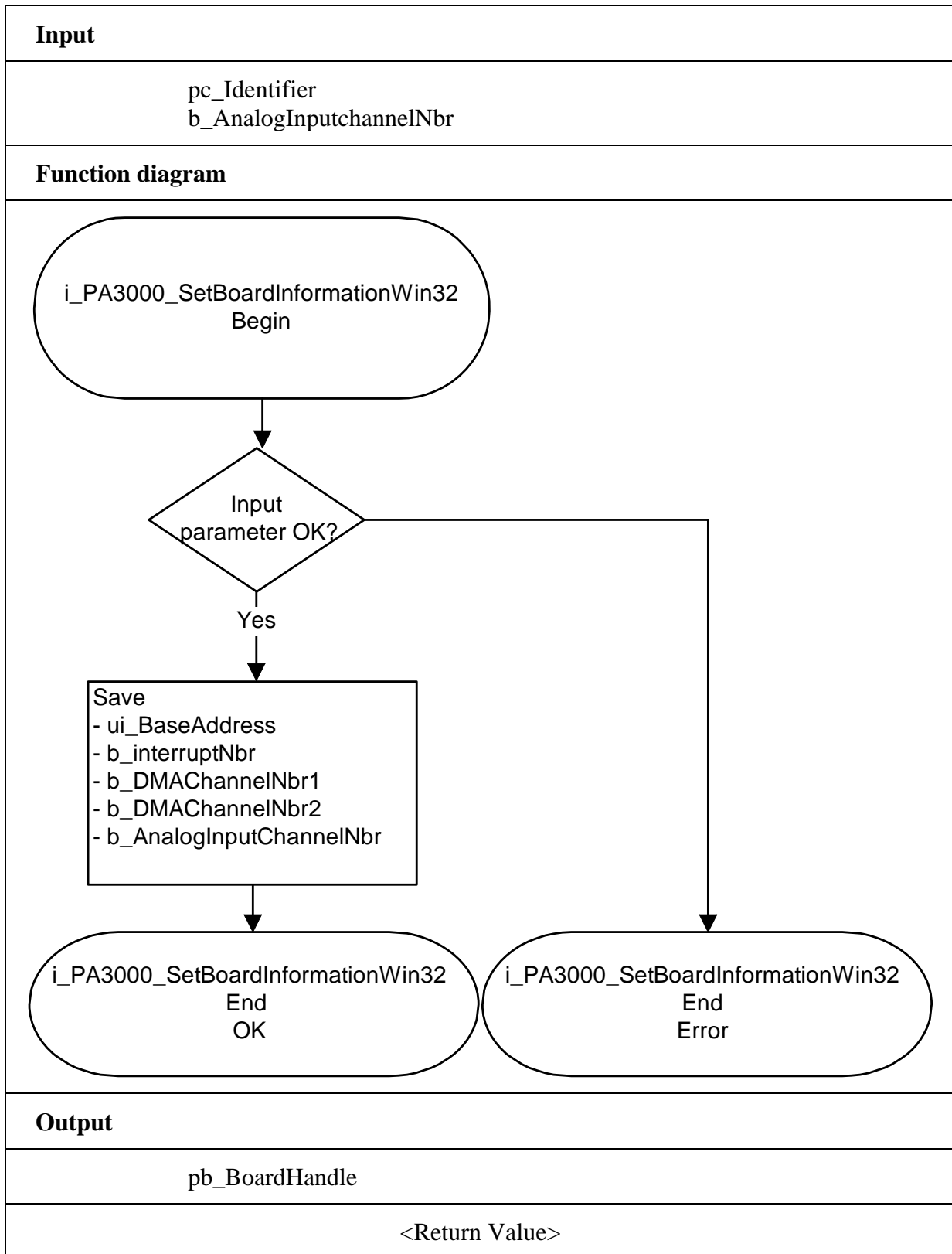
ANSI C :

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

```
i_ReturnValue = i_PA3000_SetBoardInformationWin32
                ("PA3000-00", 16, &b_BoardHandle);
```

Return value:

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



4) i_PA3000_GetHardwareInformation (...)

<Return value> = i_PA3000_GetHardwareInformation
 (BYTE b_BoardHandle,
 PUINT pui_BaseAddress,
 PBYTE pb_InterruptNbr,
 BYTE pb_DMACHannelNbr1,
 BYTE pb_DMACHannelNbr2)

Parameters:**-Input:**

BYTE b_BoardHandle Handle of board **PA3000**

- Output:

PUINT pui_BaseAddress **PA3000** base address
 PBYTE pb_InterruptNbr **PA3000** interrupt channel.
 PBYTE pb_DMACHannelNbr1 Number of the first DMA channel
 (DMA5,6,7). If 0, no DMA is used
 PBYTE pb_DMACHannelNbr2 Number of the second DMA channel
 (DMA5,6,7). If 0, no DMA is used

Task:

Returns the base address, the interrupt and DMA number of the **PA3000**.

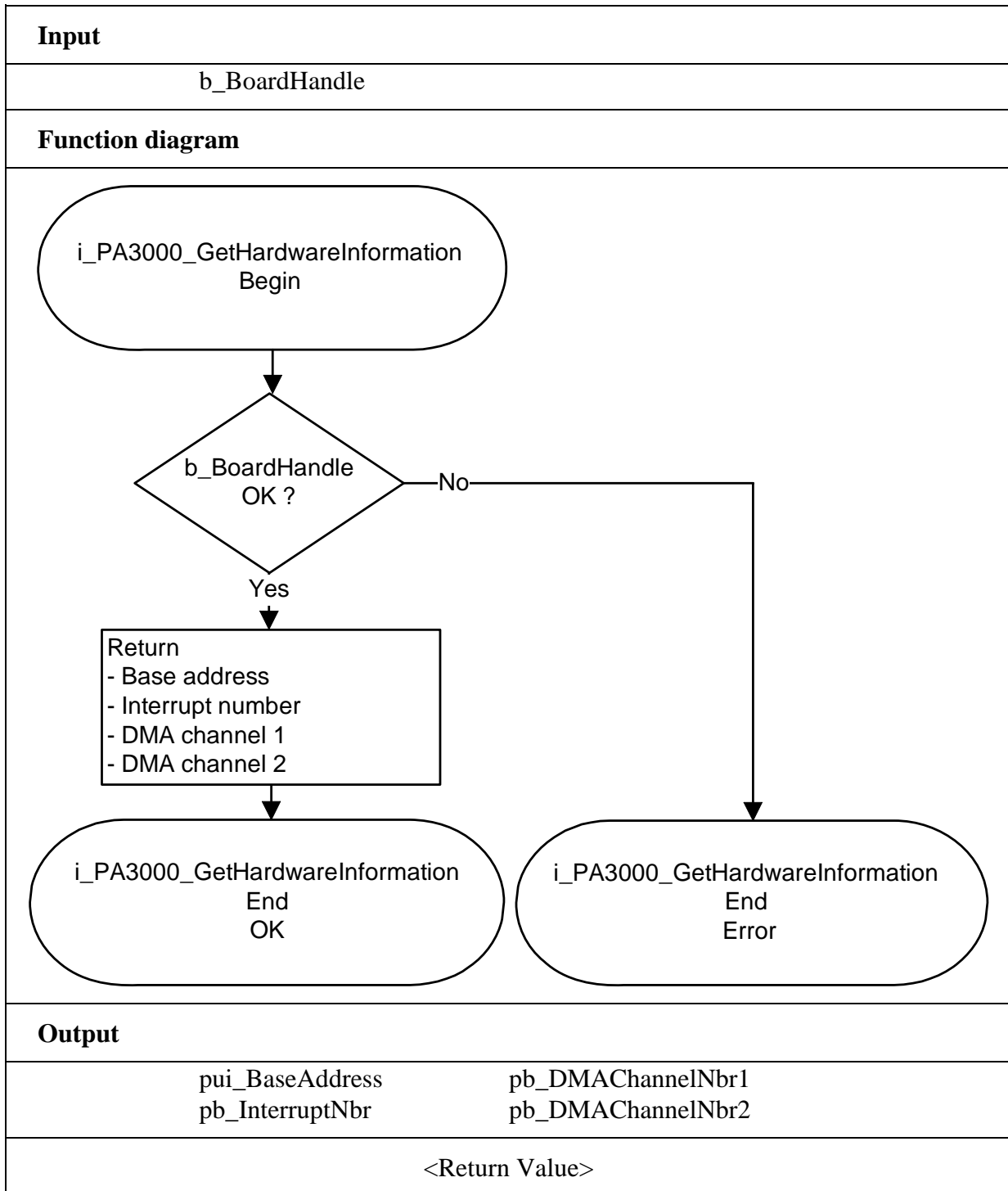
Calling convention:ANSI C:

```
int          i_ReturnValue;
unsigned int  ui_BaseAddress;
unsigned char b_InterruptNbr;
unsigned char b_DMACHannelNbr1;
unsigned char b_DMACHannelNbr2;
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_PA3000_GetHardwareInformation (b_BoardHandle,
                                                &ui_BaseAddress,
                                                &b_InterruptNbr,
                                                &b_DMACHannelNbr1,
                                                &b_DMACHannelNbr2);
```

Return value:

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



i**IMPORTANT!**

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

5) i_PA3000_CloseBoardHandle**Syntax:**

<Return value> = i_PA3000_CloseBoardHandle (BYTE b_BoardHandle)

Parameters:**- Input:**

BYTE b_BoardHandle Handle of board **PA 3000**

- Output:

No output signal has occurred.

Task:

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

Calling convention:ANSI C:

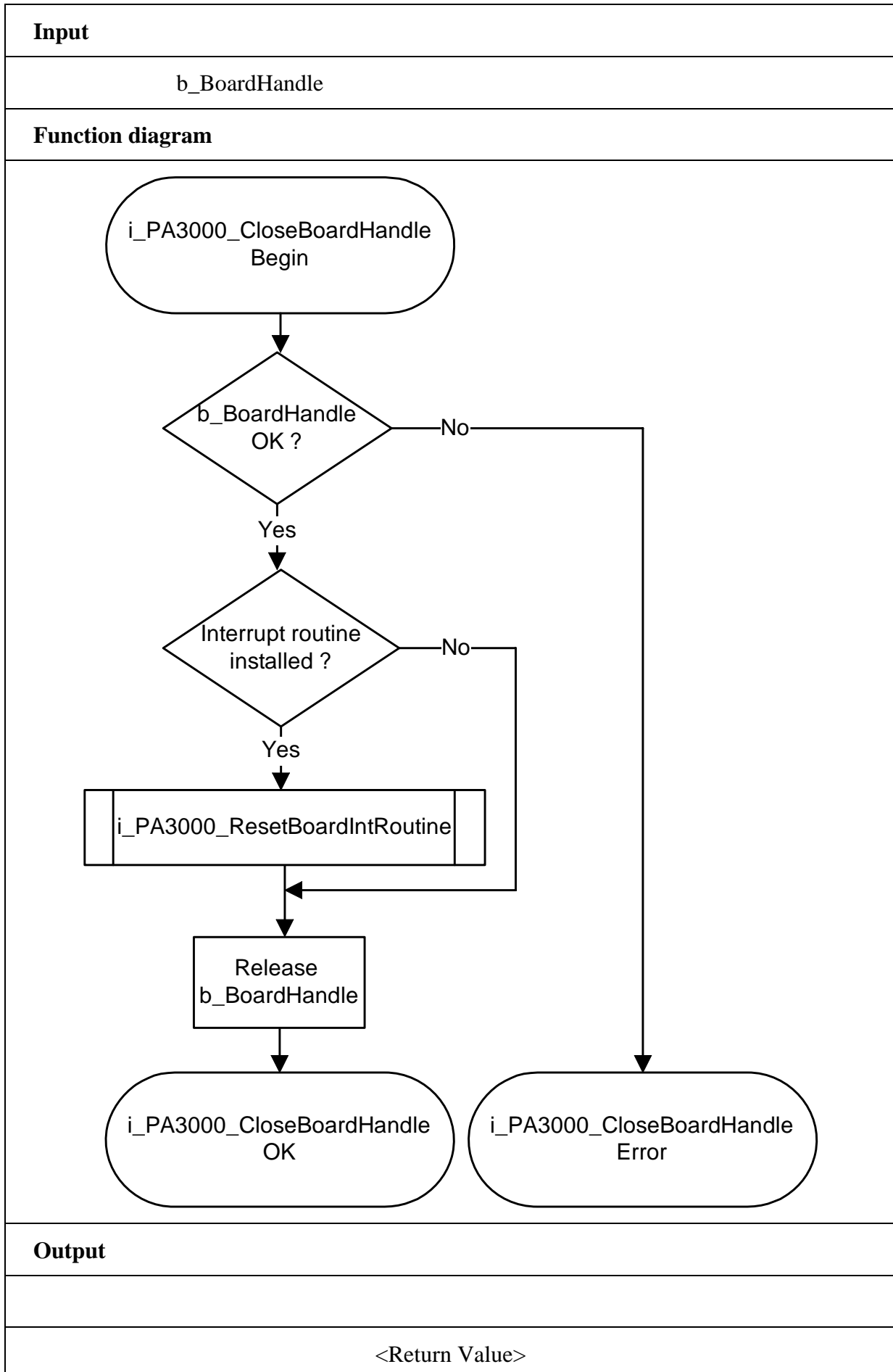
```
int          i_ReturnValue;  
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_PA3000_CloseBoardHandle (b_BoardHandle);
```

Return value:

0: No error

-1: The handle parameter of the board is wrong



3.2 Interrupt

i

IMPORTANT!

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

1) i_PA3000_SetBoardIntRoutineDos (..)

Syntax:

```
<Return value> = i_PA3000_SetBoardIntRoutineDos
                    (BYTE b_boardHandle
                    VOID   v_FunctionName
                    (BYTE  b_BoardHandle,
                     BYTE  b_InterruptMask,
                     PUINT pui_AnalogInputValue))
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board PA 3000
VOID	v_FunctionName	Name of the user interrupt routine

-Output:

No output signal has occurred

Task:

This function can be called up several times.

This function must be called up for each PA 3000 on which an interrupt is to be enabled. It installs a user interrupt on all boards on which the interrupt has been enabled.

When the function is called up for the first time (first board):

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

If you operate several boards **PA 3000** which have to react to interrupts, call up the function as often as you operate boards **PA 3000**.

From the second callup of the function (next board):

- interrupts are enabled.

The variable *v_FunctionName* is only relevant when the function is called up **for the first time**.

Interrupt

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

If several boards **PA 3000** are operated and if they have to react to interrupts, the variable *b_BoardHandle* returns the identification number (handle) of the board which has generated the interrupt.

The user interrupt routine must have the following syntax:

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

v_FunctionName Name of the user interrupt routine
b_BoardHandle Handle of the PA 3000 which has generated the interrupt
b_InterruptMask Mask of the events which have generated the interrupt
pui_AnalogInputValue The values of the analog input channels and of the DMA buffer are returned.

Table 3-1: Values returned for the analog inputs

Event	Variable name	Variable content
<i>b_InterruptMask</i> = 1	<i>pui_AnalogInputValue</i> [0]	Number of the last analog input
	<i>pui_AnalogInputValue</i> [1]	Value of the analog input
<i>b_InterruptMask</i> = 2	<i>pui_AnalogInputValue</i> [0]	Number of the last analog input
	<i>pui_AnalogInputValue</i> [1]	Value of the analog input
<i>b_InterruptMask</i> = 4	<i>pui_AnalogInputValue</i>	Values of the DMA buffer

Table 3-2: Interrupt mask

Mask	Meaning
0000 0001	End of Conversion (EOC)
0000 0010	Conversion driven by timer is completed
0000 0100	DMA conversion cycle is completed
0000 1000	Timer 2 has run down

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

Calling convention:ANSI C :

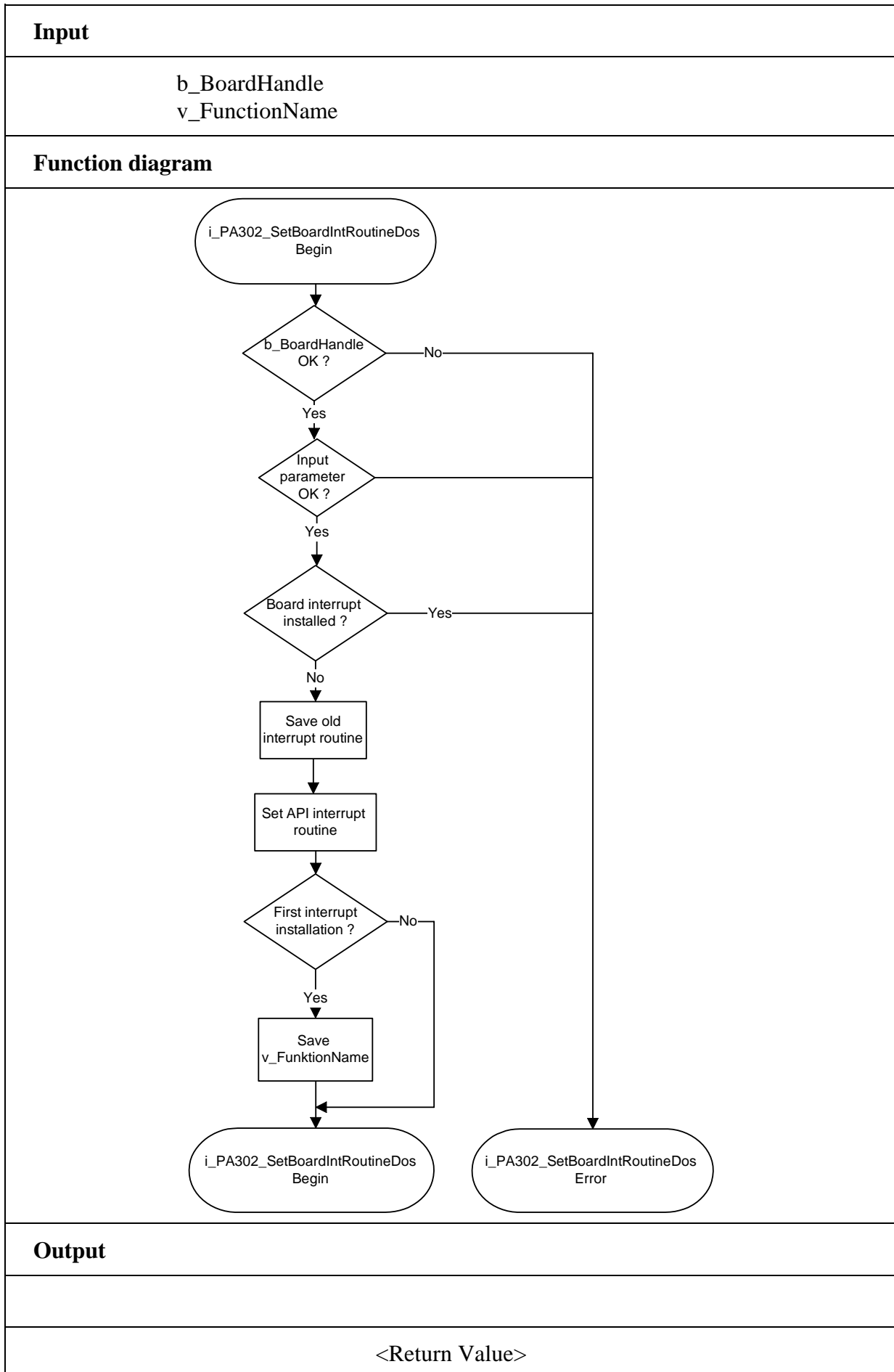
```
void    v_FunctionName    (unsigned char b_BoardHandle,  
                           unsigned char b_InterruptMask,  
                           unsigned int * ui_AnalogInputValue)  
    {  
        .  
        .  
    }
```

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

```
i_ReturnValue = i_PA3000_SetBoardIntRoutineDos (b_BoardHandle,  
                                                v_FunctionName );
```

Return value:

- 0: No error
- 1: The handle parameter of the board is wrong
- 2: No interrupt line has been initialised
- 3: Interrupt already installed



i**IMPORTANT!**

This function is only available for Visual Basic DOS.

2) i_PA3000_SetBoardIntRoutineVBDos (.)**Syntax:**

<Return value> = i_PA3000_SetBoardIntRoutineVBDos
(BYTE b_BoardHandle)

Parameters:**- Input:**

BYTE b_BoardHandle Handle of board **PA3000**

- Output:

No output signal has occurred

Task:

This function must be called up for each **PA3000** 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 boards **PA3000** which have to react to interrupts, call up the function as much as you operate boards **PA3000**.

Interrupt

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

Controlling the interrupt management

Please use instead the following functions

"ON UEVENT GOSUB xxxxxxxxx" of Visual Basic for DOS

and

"i_PA3000_TestInterrupt"

This function tests the interrupt of the **PA3000**. It is used to obtain the values of *b_BoardHandle* , *b_InterruptMask*, *pui_AnalogInputValue*.

Calling convention:

Visual Basic DOS:

```
Dim Shared i_ReturnValue           As Integer
Dim Shared i_BoardHandle           As Integer
Dim Shared i_InterruptMask         As Integer
Dim Shared l_AnalogInputValue( )   As Long
```

IntLabel:

```
i_ReturnValue = i_PA3000_TestInterrupt (i_BoardHandle, _
                                         i_InterruptMask, _
                                         l_AnalogInputValue (0) )
```

```
.
.
.
```

Return

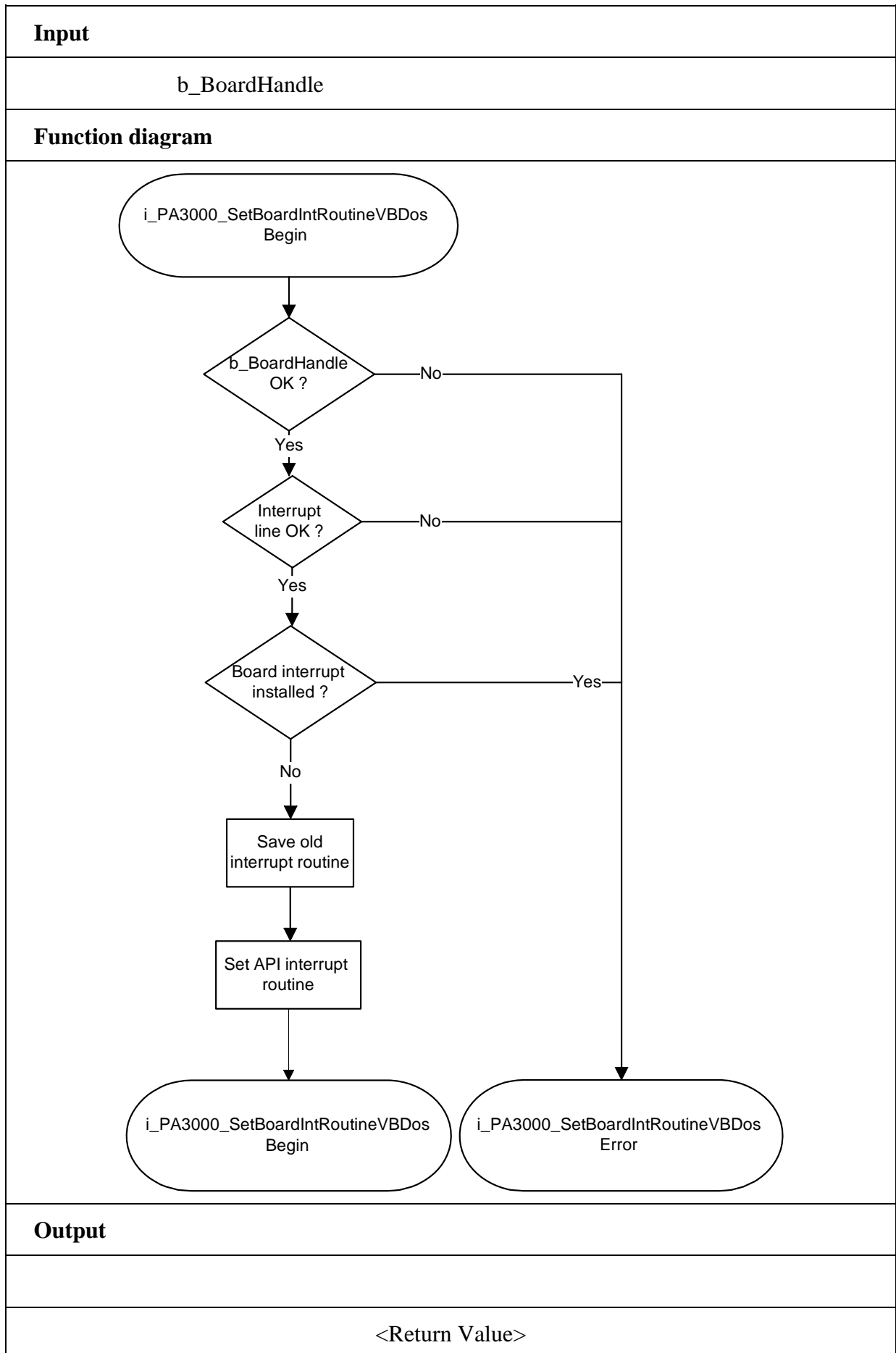
ON UEVENT GOSUB IntLabel

UEVENT ON

```
i_ReturnValue = i_PA3000_SetBoardIntRoutineVBDos (b_BoardHandle)
```

Return value:

- 0: No error
- 1: The handle parameter of the board is wrong
- 2: No interrupt line has been initialised
- 3: Interrupt already installed



3) i_PA3000_SetBoardIntRoutineWin16



IMPORTANT!

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

Syntax:

```
<Return value> = i_PA3000_SetBoardIntRoutineWin16
    (BYTE    b_BoardHandle,
     VOID    v_FunctionName (BYTE b_BoardHandle,
                             BYTE    b_InterruptMaske,
                             PUINT   pui_AnalogInputValue))
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the PA 3000
VOID	v_FunctionName	Name of the user interrupt routine.

- Output:

No output signal has occurred

Task:

This function can be called up several times.

First calling (first board):

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

If you operate several boards **PA 3000** which have to react to interrupts, call up the function as often as you operate boards **PA 3000**. 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.

The user interrupt routine must have the following syntax:

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

<i>v_FunctionName</i>	Name of the user interrupt routine
<i>b_BoardHandle</i>	Handle of the PA 3000 which has generated the interrupt
<i>b_InterruptMask</i>	Mask of the events which have generated the interrupt
<i>pui_AnalogInputValue</i>	The values of the analog input channels and of the DMA buffer are returned.

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

i

IMPORTANT!

If you use Visual Basic for Windows the following parameter has no signification. You must use the „i_PA3000_TestInterrupt“ function.

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

Calling convention:

ANSI C :

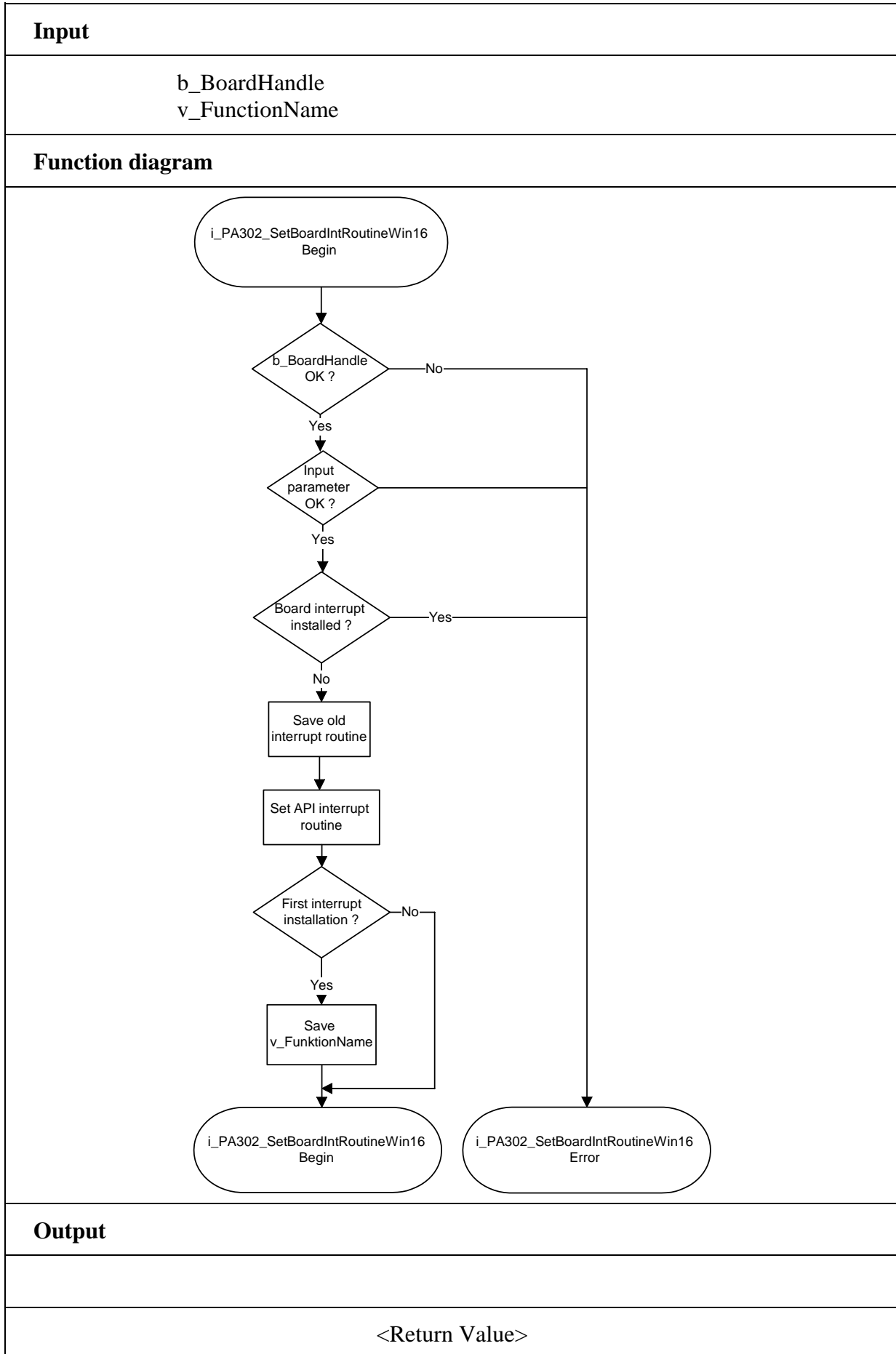
```
void v_FunctionName (unsigned char b_BoardHandle,
                    unsigned char b_InterruptMask,
                    unsigned int * ui_AnalogInputValue)
{
    .
    .
}
```

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

```
i_ReturnValue = i_PA3000_SetBoardIntRoutineWin16 (b_BoardHandle,
                                                  v_FunctionName );
```

Return value:

- 0: No error
- 1: The handle parameter of the board is wrong
- 2: No interrupt line has been initialised
- 3: Interrupt already installed



i**IMPORTANT!**

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

4) i_PA3000_SetBoardIntRoutineWin32 (..)**Syntax:**

```
<Return value> = i_PA3000_SetBoardIntRoutineWin32
    (BYTE      b_BoardHandle,
     BYTE      b_UserCallingMode,
     ULONG     ul_UserSharedMemorySize,
     VOID **   ppv_UserSharedMemory,
     VOID      v_FunctionName (BYTE  b_BoardHandle,
                               BYTE  b_InterruptMask,
                               PUINT pui_AnalogInputValue,
                               BYTE  b_UserCallingMode,
                               VOID * pv_UserSharedMemory))
```

Parameters:**- Input:**

BYTE	b_BoardHandle	Handle of board PA 3000
BYTE	b_UserCallingMode	PA3000_SYNCHRONOUS_MODE : The user routine is directly called by the driver interrupt routine. PA3000_ASYNCHRONOUS_MODE : The user routine is called by the driver interrupt thread.
VOID	v_FunctionName	Name of the user interrupt routine
ULONG	ul_UserSharedMemorySize	

Other compilers:

Determines the size in bytes of the user shared memory.
Only used if you have selected PA3000_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 PA3000_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, 4 rings (ring 0 to ring 3) are available.

- The user application operates in ring 3. This ring does not give access to hardware.
- VXD and SYS driver operate in ring 0 and give access to hardware.
- Ring 0 has no direct access to global variable from ring 3. It has to use a shared

memory.

- Ring 0 and ring 3 have a pointer that points on this shared memory. The 2 pointers are not configured under the same address.

This function must be called up for each **PA 3000** for which an interrupt is to be enabled. It installs one user interrupt function in all boards on which an interrupt is to be enabled.

First calling (first board):

- the user interrupt routine is installed
- interrupts are enabled
- user shared memory is allocated if PA3000_SYNCHRONOUS_MODE has been selected.

If you operate several boards **PA 3000** which have to react to interrupts, call up the function as often as you operate boards **PA 3000**. The variable *v_FunctionName* is only relevant **for the first calling**.

From the second call of the function (next board):

- interrupts are enabled.

Interrupt

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

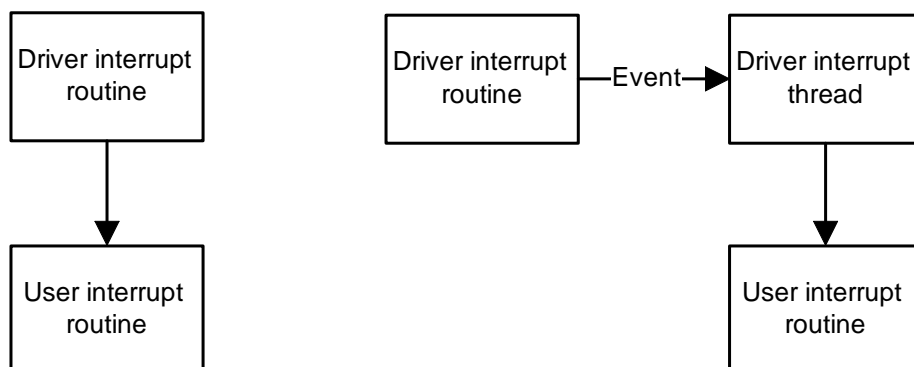
If several boards are operated and if they have to react to interrupts, the variable *b_BoardHandle* returns the identification number (handle) of the board which has generated the interrupt.

User interrupt routine can be called :

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

Synchronous mode

Asynchronous mode



SYNCHRONOUS MODE	
ADVANTAGE	The code of the user interrupt routine is directly called by the driver interrupt routine (ring 0). The time between the interrupt and the user interrupt routine is reduced.
RESTRICTIONS	The user cannot debug the user interrupt routine.
	The user routine cannot call Windows API functions.
	The user routine cannot call functions which have access to global variables. The user can still use a shared memory.
	This mode is not available for Visual Basic

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

Shared memory

If you have selected the PA3000_SYNCHRONOUS_MODE you cannot have access to global variables. But you have the possibility to create a shared memory (*ppv_UserSharedMemory*). The user shared memory can have all predefined compiler types or user define types.

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

The user interrupt routine must have the following syntax:

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

<i>v_FunctionName</i>	Name of the user interrupt routine
<i>b_BoardHandle</i>	Handle of the PA 3000 which has generated the interrupt
<i>b_InterruptMask</i>	Mask of the events which have generated the interrupt.
<i>pui_AnalogInputValue</i>	The latched values of the counter are returned.
<i>b_UserCallingMode</i>	PA3000_SYNCHRONOUS_MODE: The user tine is directly called by driver interrupt routine. PA3000_ASYNCHRONOUS_MODE: The user routine is called by driver interrupt thread
<i>pv_UserSharedMemory</i>	Pointer of the user shared memory.

i

IMPORTANT!

If you use Visual Basic 4 the following parameters have no meaning. You must used the „i_PA3000_TestInterrupt“ function.

```

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

The user can give another name for *v_FunctionName*, *b_BoardHandle*, *b_InterruptMask*, *pui_AnalogInputValue*, *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 int   *ui_AnalogInputValue,
                          unsigned char  b_UserCallingMode,
                          void *        pv_UserSharedMemory)
{
    str_UserStruct * ps_InterruptSharedMemory;

    ps_InterruptSharedMemory = (str_UserStruct *) pv_UserSharedMemory;
    .
    .
}

int      i_ReturnValue;
unsigned char  b_BoardHandle;
    
```

```
i_ReturnValue = i_PA3000_SetBoardIntRoutineWin32
                (b_BoardHandle, PA3000_SYNCHRONOUS_MODE,
                 sizeof (str_UserStruct),
                 (void **) &ps_UserSharedMemory,
                 v_FunctionName);
```

Visual Basic 5:

```
Sub    v_FunctionName    (ByVal i_BoardHandle    As Integer,
                          ByVal i_InterruptMask  As Integer,
                          l_AnalogInputValue     As Long,
                          b_UserCallingMode     As Integer,
                          l_UserSharedMemory     As Long)
```

```
End Sub
```

```
Dim i_ReturnValue As Integer
```

```
Dim i_BoardHandle As Integer
```

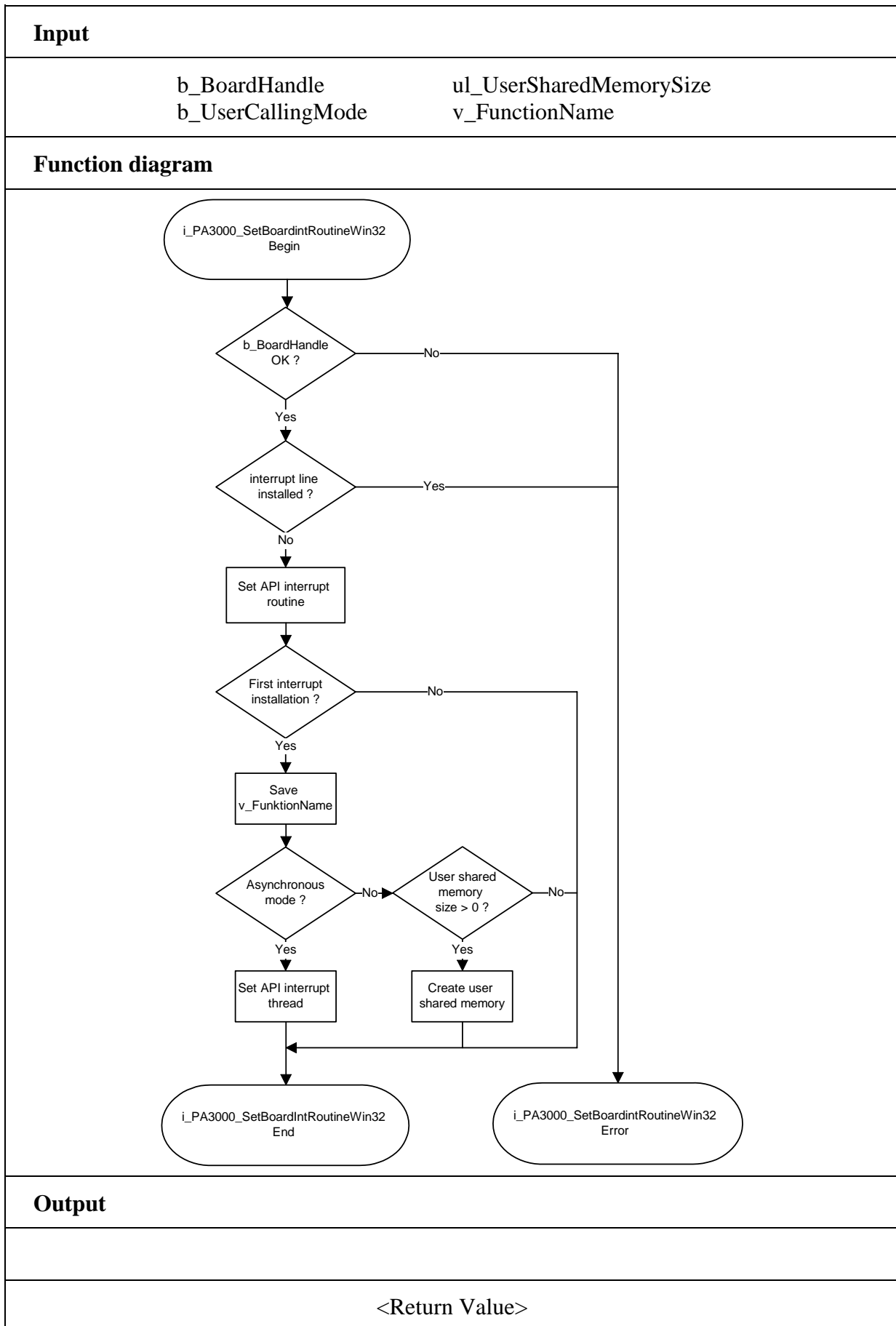
```
...
```

```
i_ReturnValue = i_PA3000_SetBoardIntRoutineWin32
                (i_BoardHandle,
                 PA3000_ASYNCHRONOUS_MODE,
                 0,
                 0,
                 AddressOf v_FunctionName)
```

```
...
```

Return value:

- 0: No error
- 1: The handle parameter of the board is wrong
- 2: No interrupt line initialised
- 3: Interrupt already installed
- 4: The calling mode selected for the user interrupt routine is wrong
- 5: No memory available for the allocation of user shared memory.



5) i_PA3000_TestInterrupt (..)

Syntax:

<Return value> = i_PA3000_TestInterrupt (PBYTE pb_BoardHandle,
 PBYTE pb_InterruptMask,
 PUINT pui_AnalogInputValue)

Parameters:

- Input:

No input signal occurs.

- Output:

PBYTE pb_BoardHandle Handle of the board **PA 3000** which has generated the interrupt

PBYTE pb_InterruptMask Mask of the events which have generated the interrupt.

PUINT pui_AnalogInputValue The values of the analog input channels and of the DMA buffer are returned.

Values returned for the analog inputs

Event	Variable name	Variable content
<i>b_InterruptMask</i> = 1	<i>pui_AnalogInputValue</i> [0]	Number of the last analog input
	<i>pui_AnalogInputValue</i> [1]	Value of the analog input
<i>b_InterruptMask</i> = 2	<i>pui_AnalogInputValue</i> [0]	Number of the last analog input
	<i>pui_AnalogInputValue</i> [1]	Value of the analog input
<i>b_InterruptMask</i> = 4	<i>pui_AnalogInputValue</i>	Values of the DMA buffer

Example 1 *pui_AnalogInputValue* [0] = 3

3	= Number of analog inputs
1	= Analog value 0
2	= Analog value 1
3	= Analog value 2

Interrupt mask

Mask	Meaning
0000 0001	End of Conversion (EOC)
0000 0010	Conversion driven by timer is completed
0000 0100	DMA conversion cycle is completed
0000 1000	Timer 2 has run down

Task:

Verifies if a board **PA3000** has generated an interrupt. If yes, the function returns the board handle and the interrupt source.



IMPORTANT!

This function is only available in Visual Basic for DOS and Windows .

Calling convention:

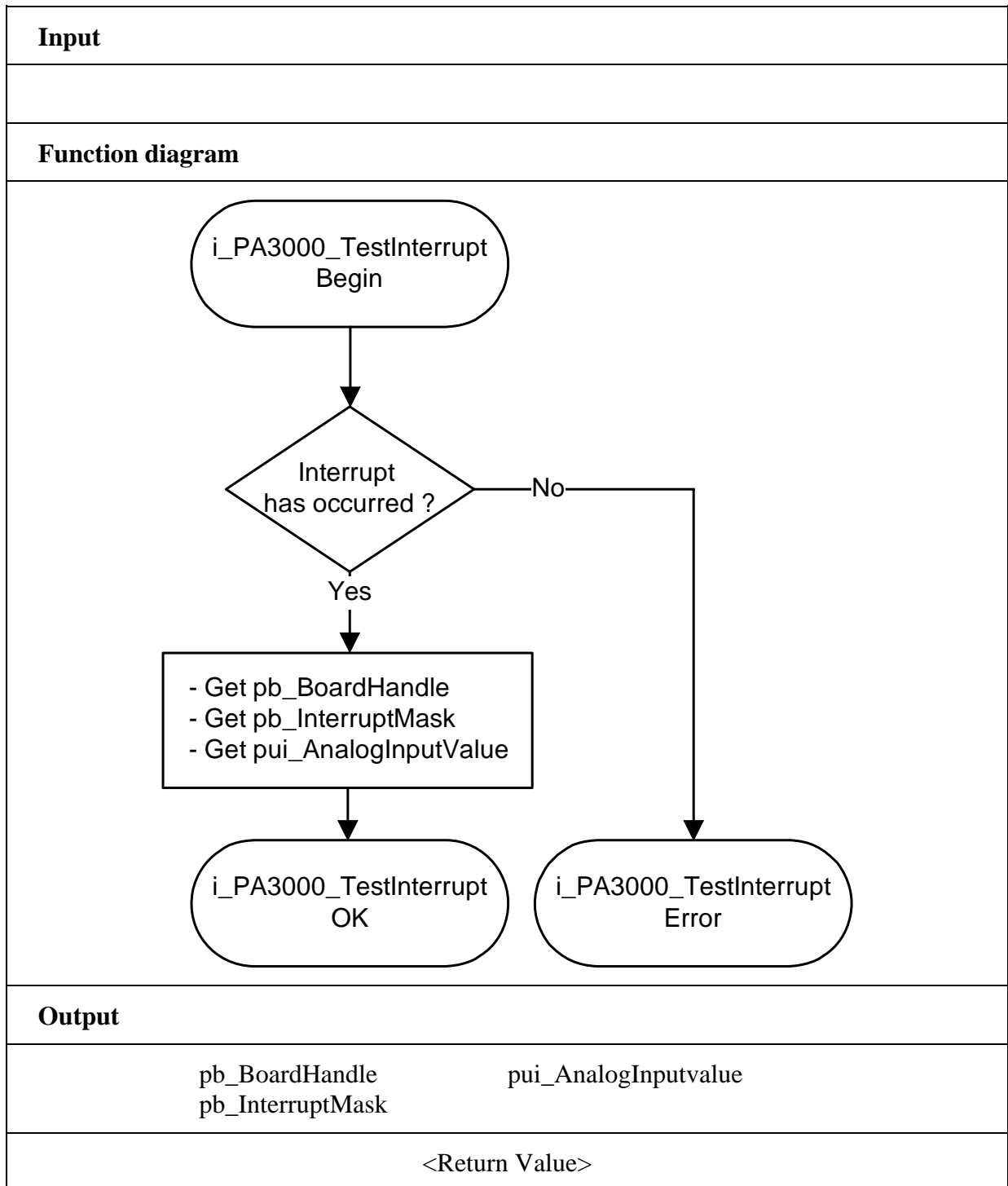
ANSI C :

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

```
i_ReturnValue = i_PA3000_TestInterrupt (&b_BoardHandle,
                                         &b_InterruptMask,
                                         ui_AnalogInputValue);
```

Return value:

- 1 No interrupt
- > 0 IRQ number



6) i_PA3000_ResetBoardIntRoutine (..)

Syntax:

<Return value> = i_PA3000_ResetBoardIntRoutine (BYTE b_BoardHandle)

Parameters:

- Input:

BYTE b_BoardHandle Handle of board **PA 3000**

- Output:

No output signal has occurred

Task:

Stops the interrupt management of board **PA 3000**.

Deinstalls the user interrupt routine if the management of interrupts of all boards **PA 3000** is stopped.

Calling convention:

ANSI C:

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

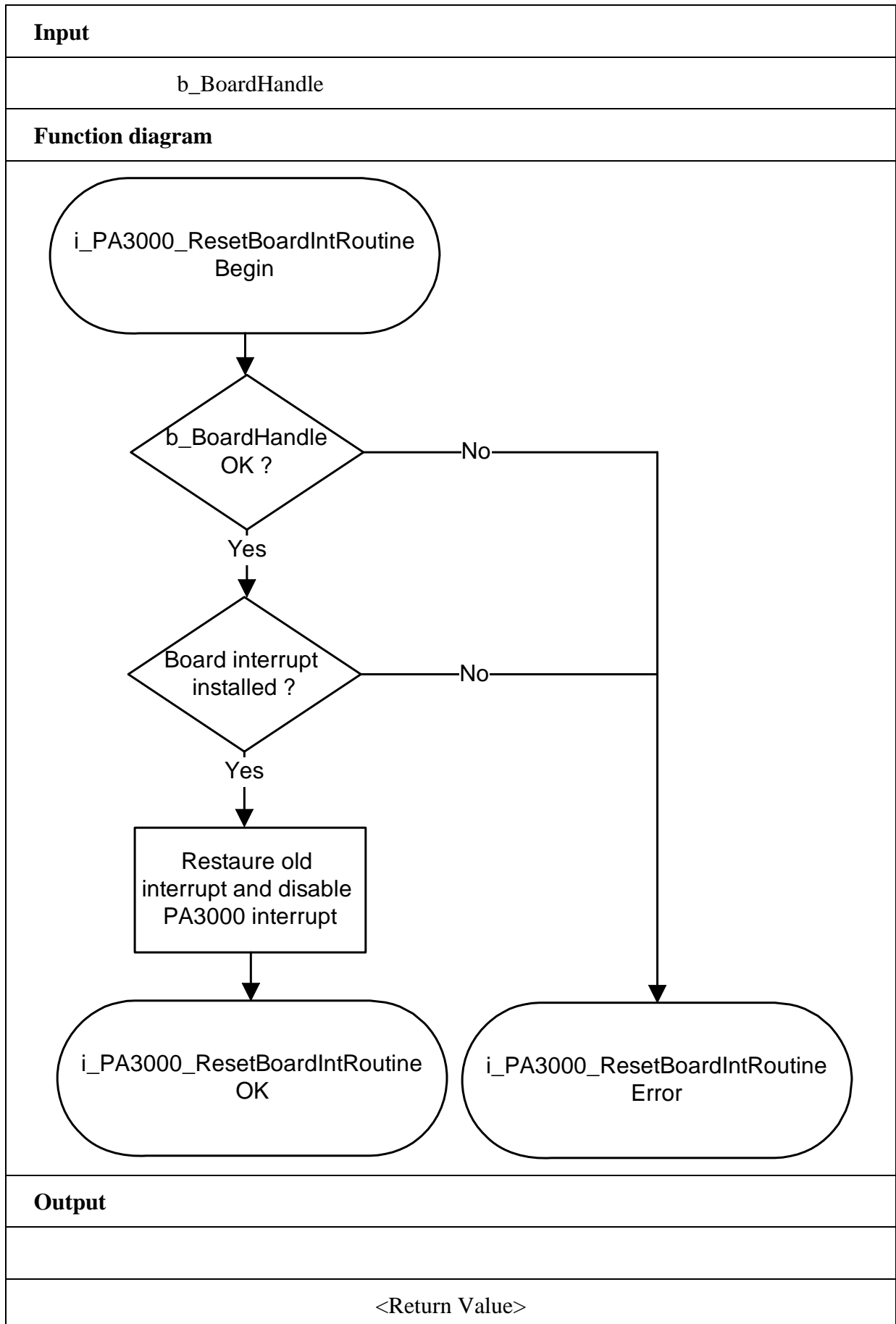
```
i_ReturnValue = i_PA3000_ResetBoardIntRoutine (b_BoardHandle);
```

Return value:

0: No error

-1: The handle parameter of the board is wrong

-2: No interurpt has been initialised with function
"i_PA3000_SetBoardIntRoutineXXX"



3.3 Direct conversion of the analog input channels

1) i_PA3000_Read1AnalogInput (...)

Syntax:

```
<Return value> = i_PA3000_Read1AnalogInput
                    (BYTE   b_BoardHandle,
                    BYTE   b_Channel,
                    BYTE   b_Gain,
                    BYTE   b_Polarity,
                    UINT   ui_ConvertTiming,
                    BYTE   b_InterruptFlag,
                    PUINT  pui_AnalogInputValue)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board PA 3000
BYTE	b_Channel	Number of the analog input channel to be read. (0 to 15) See table 3-3.
BYTE	b_Gain	Gain selection (see table 3-4)
BYTE	b_Polarity	Selection of the input voltage range of the analog input channel to convert (see table 3-5) Unipolar: 128 Bipolar: 0
UINT	ui_ConvertTiming	Selection of the conversion time from 10 μs to 45874 μs.
BYTE	b_InterruptFlag	PA3000_ENABLE: An interrupt is generated at the end of conversion. See function "i_PA3000_SetBoardIntRoutine". PA3000_DISABLE: No interrupt is generated at the end of conversion. The analog value is in the parameter <i>pui_AnalogInputValue</i> .

- Output:

PUINT	pui_AnalogInputValue	The analog value is returned (0 to 4095)
-------	----------------------	---

Task:

Reads the current value of the analog input channel *b_Channel* with a gain of *b_Gain*, an input voltage range of *b_Polarity* and a conversion time of *ui_ConvertTiming*.

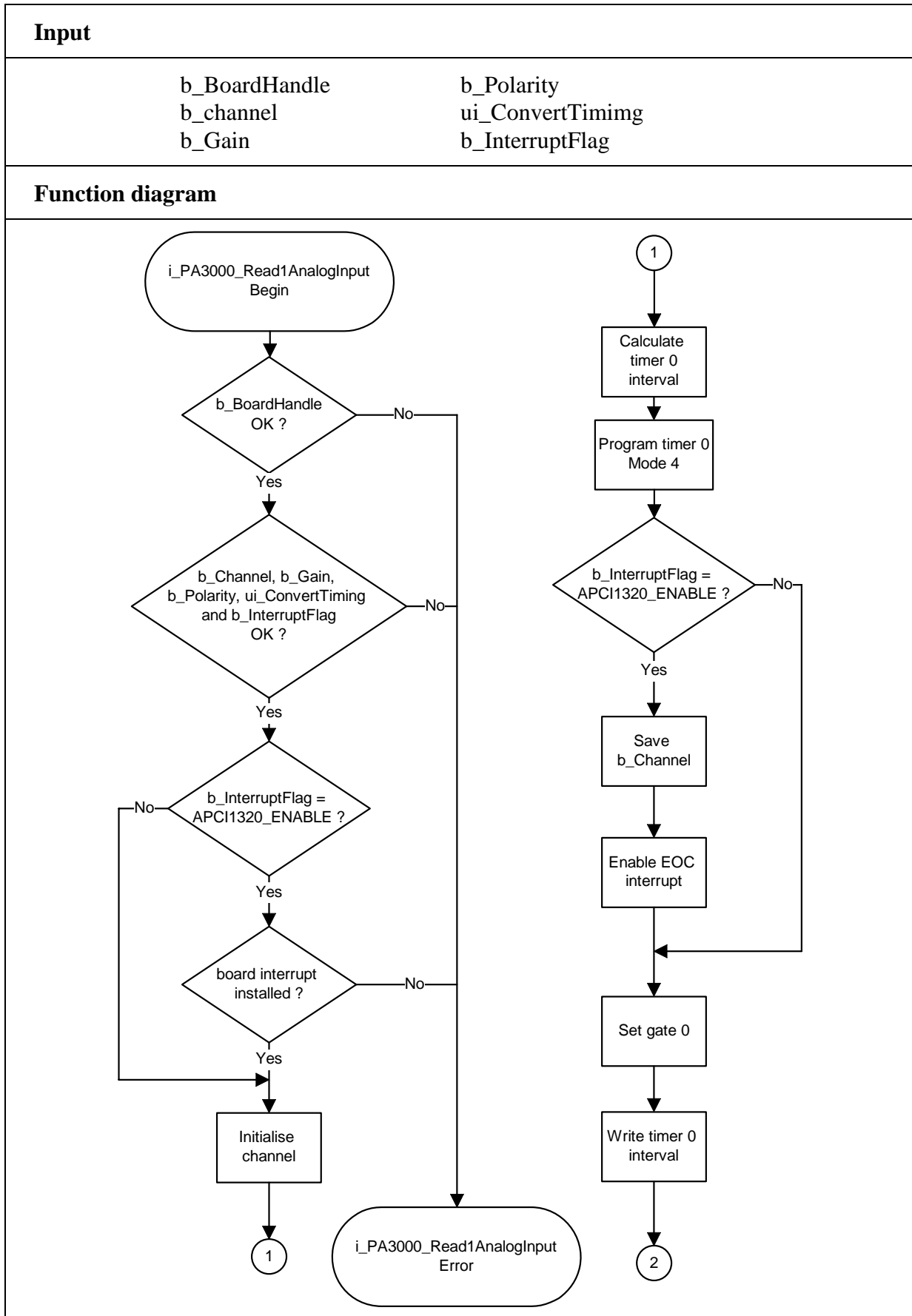
Calling convention:ANSI C :

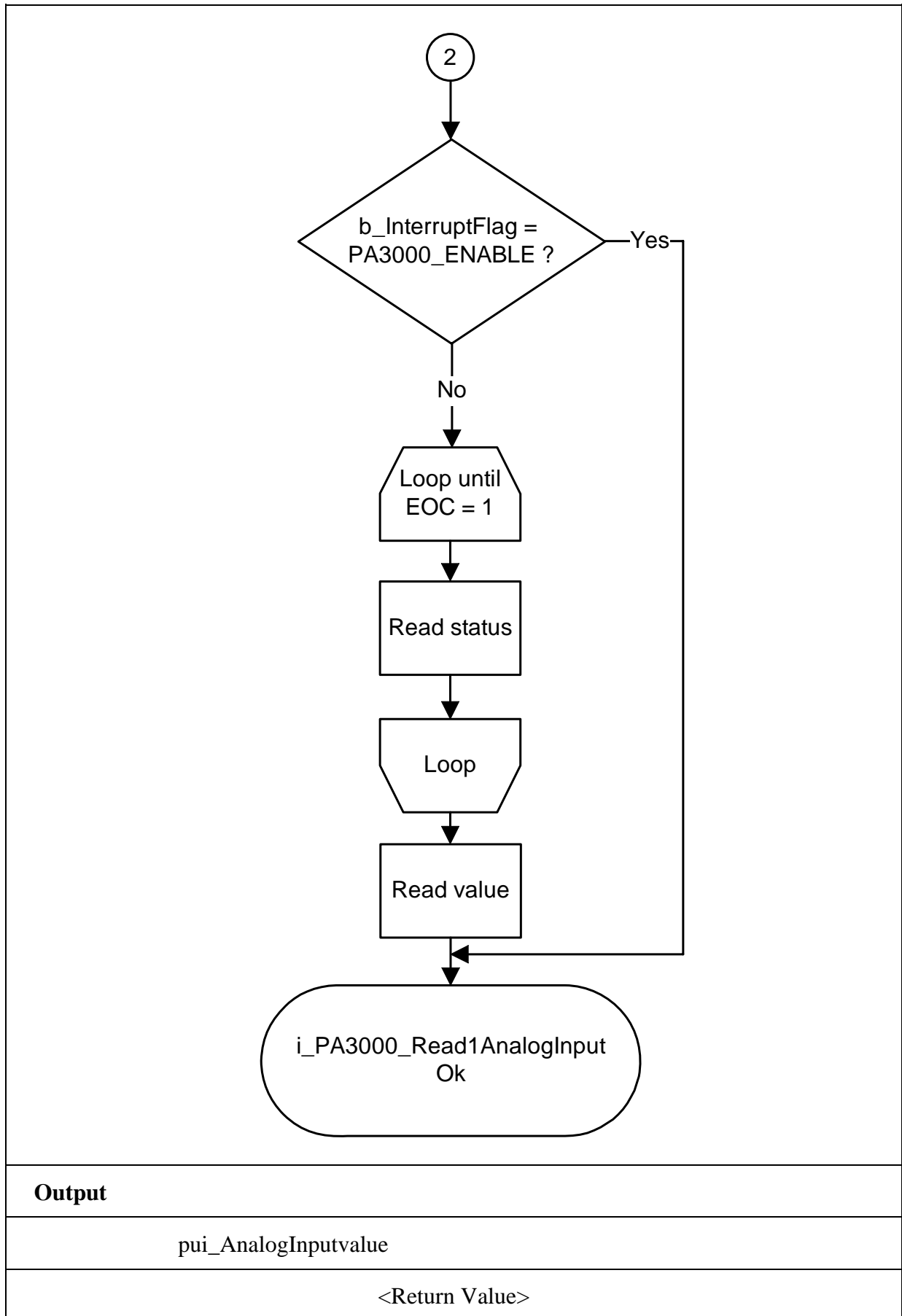
```
int          i_ReturnValue;  
unsigned char b_BoardHandle;  
unsigned int  ui_AnalogInputValue;
```

```
i_ReturnValue = i_PA3000_Read1AnalogInput (b_BoardHandle,  
PA3000_CHANNEL_0,  
PA3000_1_GAIN,  
PA3000_UNIPOLAR,  
10,  
PA3000_DISABLE,  
& ui_AnalogInputValue);
```

Return value:

- 0: No error
- 1: The handle parameter of the board is wrong
- 2: Number of the analog input channel is wrong. See table 3-3.
- 3: The gain selected is wrong. See table 3-4.
- 4: The input voltage range selected is wrong. See table 3-5.
- 5: The conversion time selected is wrong
- 6: A wrong parameter has been passed for *b_InterruptFlag* or the user interrupt routine has not been installed.
See function "i_PA3000_SetBoardIntRoutine".





2) i_PA3000_ReadAllAnalogInput (...)

Syntax:

```
<Return value> = i_PA3000_ReadAllAnalogInput
                    (BYTE    b_BoardHandle,
                    BYTE    b_Gain,
                    BYTE    b_Polarity,
                    UINT    ui_ConvertTiming
                    PUINT   pui_AnalogInputValueArray)
```

Parameters:

- Input

BYTE	b_BoardHandle	Handle of board PA 3000
BYTE	b_Gain	Gain selection. See table 3-4.
BYTE	b_Polarity	Selection of the input voltage range of the analog input channels to convert (see table 3-5.) Unipolar: 128 Bipolar: 0
UINT	ui_ConvertTiming	Selection of the conversion time from 10 µs to 45874 µs.

- Output:

PUINT	pui_AnalogInputValue	The analog values are returned
-------	----------------------	--------------------------------

Task:

Reads the current values of all analog input channels with a gain of *b_Gain*, an input voltage range of *b_Polarity*.

Calling convention:

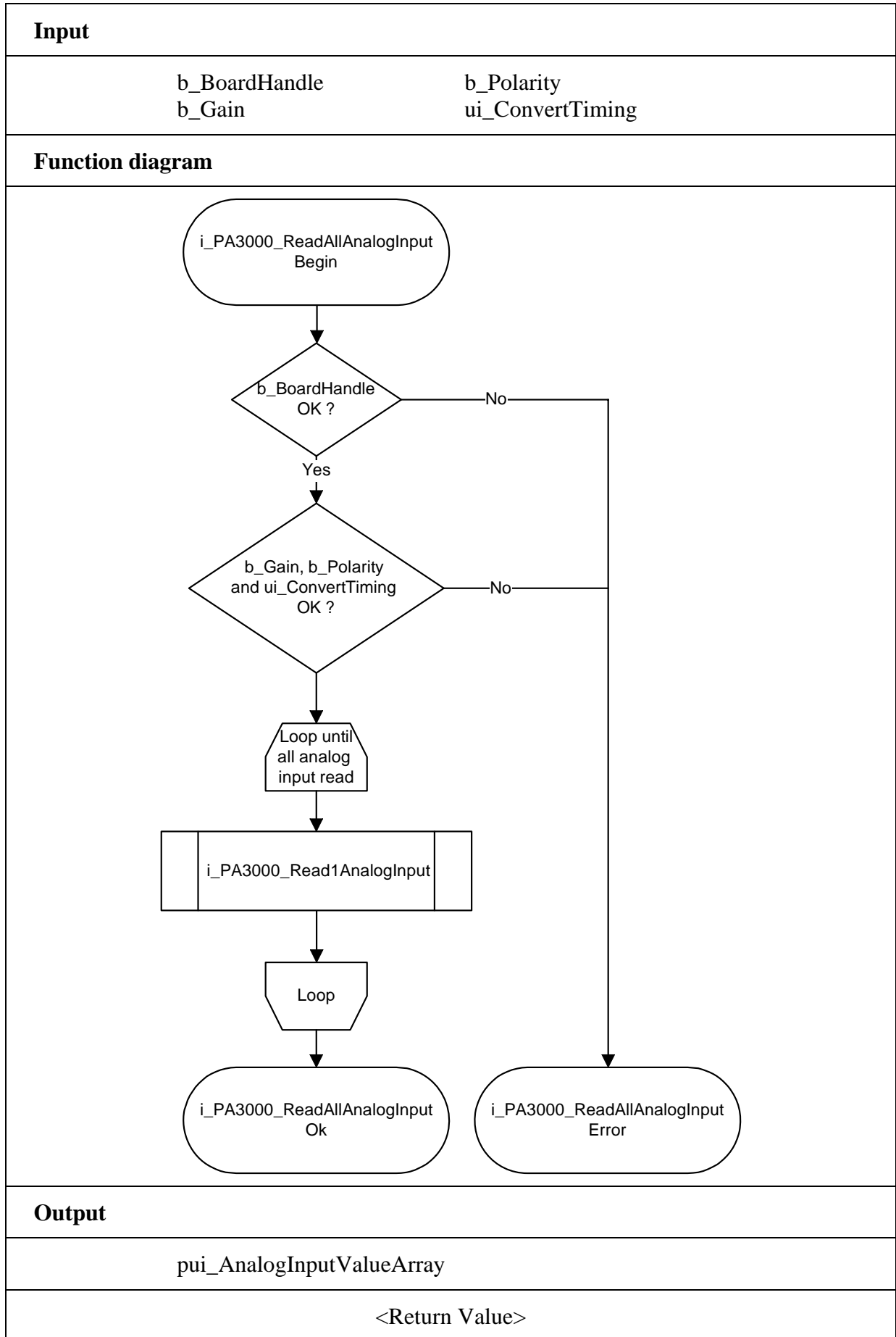
ANSI C:

```
int            i_ReturnValue;
unsigned char  b_BoardHandle;
unsigned char  b_Gain;
unsigned char  b_Polarity;
unsigned int   ui_AnalogInputValueArray [16];
```

```
i_ReturnValue = i_PA3000_ReadAllAnalogInput (b_BoardHandle,
                                             b_Gain,
                                             b_Polarity,
                                             ui_AnalogInputValue);
```

Return value:

- 0: No error
- 1: The handle parameter of the board is wrong
- 2: The gain selected is wrong. See table 3-4.
- 3 The input voltage range selected is wrong. See table 3-5.
- 4 The conversion time selected is wrong



3.4 Cyclic conversion of analog input channels

1) i_PA3000_InitAnalogInputAcquisition (...)

Syntax:

```
<Return value> = i_PA3000_InitAnalogInputAcquisition
                    (BYTE      b_BoardHandle,
                    BYTE      b_SequenzArraySize,
                    PBYTE     pb_ChannelArray,
                    PBYTE     pb_GainArray,
                    PBYTE     pb_PolarityArray,
                    BYTE      b_AcquisitionMode,
                    BYTE      b_ExternTrigger,
                    UINT       ui_AcquisitionTiming,
                    LONG      l_DelayTiming,
                    UINT       ui_NumberOfAcquisition,
                    BYTE      b_DMAUsed,
                    BYTE      b_AcquisitionCycle)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board PA 3000
BYTE	b_SequenzArraySize	Size of the scan lists (1 to 16 elements)
PBYTE	pb_ChannelArray	Scan list for the analog input channels. See table 3-3.
PBYTE	pb_GainArray	Scan list for gain See table 3-4.
PBYTE	pb_PolarityArray	Scan list for the input voltage range See table 3-5.
BYTE	b_AcquisitionMode	Two conversion cycles are possible: - PA3000_SIMPLE_MODUS: A conversion occurs every <i>ui_AcquisitionTiming</i> (time interval) See example 2. - PA3000_DELAY_MODUS: Both times are used in this mode: <i>ui_AcquisitionTiming</i> and <i>l_DelayTiming</i> . Conversions occur every <i>ui_AcquisitionTiming</i> (time interval) until all the analog input channels have been acquired (determined by <i>b_SequenzArraySize</i>) Step 1 of example 3. Afterwards there is a waiting time of <i>l_DelayTiming</i> . Step 2 of example 3. The two steps are repeated. See example 3.

BYTE	b_ExternTrigger	<ul style="list-style-type: none"> - PA3000_DISABLE: the cyclic conversion is initialised and started immediately after the call of function "i_PA3000_StartAnalog-InputAcquisition". - PA3000_ENABLE: the cyclic conversion is initialised after the call of function "i_PA3000_StartAnalog-InputAcquisition". A logic "1" on the digital input 1 starts converting. The conversion is stopped when the digital input 1 is at logic "1" again.
UINT	ui_AcquisitionTiming	<p>Time in μs between 2 conversions of successive input channels</p> <ul style="list-style-type: none"> - from 10 μs to 45874 μs, if you use the option PA3000_DMA_USED. - from 150 μs to 45874 μs, if you use the option PA3000_NOT_DMA_USED <p>See example 1 and 2.</p>
LONG	l_DelayTiming	<p>Waiting time in μs between two conversion cycles (from 70 μs to 4587400 μs). This parameter has only a signification if you use the mode PA3000_DELAY_MODUS.</p>
UINT	ui_NumberOfAcquisition	<p>If you use DMA, this parameter determines how many conversions have to happen (1 to 32767). If you do not use DMA, it determines how many acquisition cycles are to be performed.</p>
BYTE	b_DMAUsed	<p>Determines if DMA must be used or not.</p> <ul style="list-style-type: none"> - PA3000_DMA_USED: All conversion values are saved An interrupt is generated when the conversions (set by <i>ui_NumberOfAcquisition</i>) are completed. See function "i_PA3000_SetBoardIntRoutine". - PA3000_DMA_NOT_USED: An interrupt is generated after the conversion of the last channel group. The analog value is returned to the user through the interrupt routine. See function "i_PA3000_SetBoardIntRoutine".

BYTE *b_AcquisitionCycle* Determines the type of DMA conversion.

- PA3000_CONTINUOUS:
An interrupt is generated each time a DMA conversion cycle is completed. A new DMA conversion cycle is started.
- PA3000_SINGLE:
The DMA conversion cycle is only carried out once: i.e. you receive one single interrupt at the end of the first DMA conversion cycle. See example 1.

- Output:

No output signal has occurred.

Task:

This function initialises a cyclic conversion.

The priority of the analog input channels is set with the scan list.

The scan list allows to determine the input voltage range and the gain for each analog input. See example 1.

The DMA option (PA3000_DMA_USED) allows to acquire in the background analog values of a high frequency.

An interrupt is generated at the end of conversion. In your interrupt routine a "2" is passed through the parameter *b_InterruptMask*. The DMA buffer is returned through the parameter *pui_AnalogInputValue*.

See function "i_PA3000_SetBoardIntRoutine".

You have to:

- set the priority of the analog input channels through the scan list.
- enter the mode through the parameter *b_AcquisitionMode*.
- enter the time between two conversions through the parameter *ui_AcquisitionTiming*.
- enter the waiting time between two conversion cycles through the parameter *l_DelayTiming* (if you use the mode PA3000_DELAY_MODUS).
- determine if DMA must be used (parameter *b_DMAUsed*)
- enter the number of acquisitions through the parameter *ui_NumberOfAcquisition*
- enter the DMA conversion cycle through the parameter *b_AcquisitionCycle* (if DMA is used)

