



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



**Attention!**  
Product discontinuation  
due to EC RoHS directive  
More info: [www.addi-data.com](http://www.addi-data.com)

Technical description

**ADDIALOG PA 3110**

Standard software

Edition: 09.01-07/2005

<b>1</b>	<b>INTRODUCTION .....</b>	<b>1</b>
<b>2</b>	<b>DIN 66001- GRAPHICAL SYMBOLS .....</b>	<b>4</b>
<b>3</b>	<b>SOFTWARE FUNCTIONS (API) .....</b>	<b>5</b>
<b>3.1</b>	<b>Initialisation.....</b>	<b>5</b>
	1) i_PA3110_InitCompiler (..).....	5
	2) i_PA3110_SetBoardInformation (...) .....	7
	3) i_PA3110_SetBoardInformationWin32 (...) .....	9
	4) i_PA3110_SetAnalogInputResolution (...) .....	11
	5) i_PA3110_GetHardwareInformation (...).....	13
	6) i_PA3110_CloseBoardHandle .....	15
<b>3.2</b>	<b>Interrupt.....</b>	<b>17</b>
	1) i_PA3110_SetBoardIntRoutineDos (..).....	17
	2) i_PA3110_SetBoardIntRoutineVBDos (..) .....	21
	3) i_PA3110_SetBoardIntRoutineWin16.....	24
	4) i_PA3110_SetBoardIntRoutineWin32 (..) .....	27
	5) i_PA3110_TestInterrupt (..) .....	33
	6) i_PA3110_ResetBoardIntRoutine (..) .....	35
<b>3.3</b>	<b>Direct conversion of the analog input channels .....</b>	<b>37</b>
	1) i_PA3110_Read1AnalogInput (...).....	37
	2) i_PA3110_ReadMoreAnalogInput (...).....	41
<b>3.4</b>	<b>Cyclic conversion of analog input channels .....</b>	<b>45</b>
	1) i_PA3110_InitAnalogInputAcquisition (...).....	45
	2) i_PA3110_StartAnalogInputAcquisition (...).....	55
	3) i_PA3110_StopAnalogInputAcquisition (...).....	58
	4) i_PA3110_ClearAnalogInputAcquisition (...).....	61
<b>3.5</b>	<b>Analog output channels.....</b>	<b>63</b>
	1) i_PA3110_Write1AnalogValue (...).....	63
	2) i_PA3110_WriteMoreAnalogValue (...).....	65
<b>3.6</b>	<b>Timer.....</b>	<b>68</b>
	1) i_PA3110_InitTimerWatchdog (...) .....	68
	2) i_PA3110_StartTimerWatchdog (...) .....	70
	3) i_PA3110_StopTimerWatchdog (...).....	72
	4) i_PA3110_ReadTimer (...) .....	74
	5) i_PA3110_WriteTimer (...) .....	76
	6) i_PA3110_ReadWatchdogStatus (...).....	78
<b>3.7</b>	<b>Functions to be used in Kernel mode .....</b>	<b>80</b>
	1) i_PA3110_KRNL_Write1AnalogValue (...).....	80

**Tabellen**

Table 1-1: Type Declaration for Dos and Windows 3.1X .....	1
Table 1-2: Type Declaration for Windows 95/NT .....	1
Table 1-3: Define value .....	2
Table 3-1: Values returned for the analog inputs.....	18
Table 3-2: Interrupt mask .....	18
Table 3-3: Selection of the analog input channels.....	47
Table 3-4: Gain selection .....	47
Table 3-5 Selection of the input voltage range .....	48

# 1 INTRODUCTION



## IMPORTANT!

Note the following conventions in the text:

Function: "i\_PA3110\_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
<b>VOID</b>	void	void	pointer		any
<b>BYTE</b>	unsigned char	unsigned char	byte	integer	integer
<b>INT</b>	int	int	integer	integer	integer
<b>UINT</b>	unsigned int	unsigned int	word	long	long
<b>LONG</b>	long	long	longint	long	long
<b>PBYTE</b>	unsigned char *	unsigned char *	var byte	integer	integer
<b>PINT</b>	int *	int *	var integer	integer	integer
<b>PUINT</b>	unsigned int *	unsigned int *	var word	long	long
<b>PCHAR</b>	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
<b>VOID</b>	void	void	pointer		any
<b>BYTE</b>	unsigned char	unsigned char	byte	integer	integer
<b>INT</b>	int	int	integer	integer	integer
<b>UINT</b>	unsigned int	unsigned int	long	long	long
<b>LONG</b>	long	long	longint	long	long
<b>PBYTE</b>	unsigned char *	unsigned char *	var byte	integer	integer
<b>PINT</b>	int *	int *	var integer	integer	integer
<b>PUINT</b>	unsigned int *	unsigned int *	var long	long	long
<b>PCHAR</b>	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
DLL_COMPILER_VB_5	4	4
PA3110_DISABLE	0	0
PA3110_ENABLE	1	1
PA3110_CHANNEL0	0	0
PA3110_CHANNEL1	1	1
PA3110_CHANNEL2	2	2
PA3110_CHANNEL3	3	3
PA3110_CHANNEL4	4	4
PA3110_CHANNEL5	5	5
PA3110_CHANNEL6	6	6
PA3110_CHANNEL7	7	7
PA3110_CHANNEL8	8	8
PA3110_CHANNEL9	9	9
PA3110_CHANNEL10	10	A
PA3110_CHANNEL11	11	B
PA3110_CHANNEL12	12	C

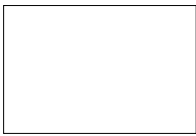
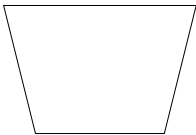
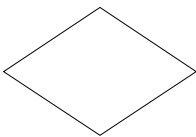
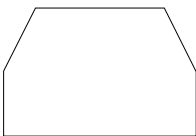

<b>Define name</b>	<b>Decimal value</b>	<b>Hexadecimal value</b>
PA3110_CHANNEL13	13	D
PA3110_CHANNEL14	14	E
PA3110_CHANNEL15	15	F
PA3110_1_GAIN	0	0
PA3110_2_GAIN	16	10
PA3110_5_GAIN	32	20
PA3110_10_GAIN	48	30
PA3110_UNIPOLAR	128	80
PA3110_BIPOLAR	0	0
PA3110_SIMPLE_MODUS	0	0
PA3110_DELAY_MODUS	1	1
PA3110_DMA_NOT_USED	0	0
PA3110_DMA_USED	1	1
PA3110_SINGLE	0	0
PA3110_CONTINUOUS	1	1

## 2 DIN 66001- GRAPHICAL SYMBOLS

This chapter describes all software functions (API) necessary for the operation of the PA 1700-2 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><b>Process, general</b> (including inputs and outputs)</p>
	<p><b>Manual operation</b> (including inputs and outputs)</p>
	<p><b>Decision</b>  <b>Selection unit</b> (e.g.: switch)</p>
	<p><b>Loop limit</b> Beginning</p>
	<p><b>Loop limit</b> End</p>

### 3 SOFTWARE FUNCTIONS (API)

#### 3.1 Initialisation

##### 1) i\_PA3110\_InitCompiler (..)

**Syntax:**

<Return value> = i\_PA3110\_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;

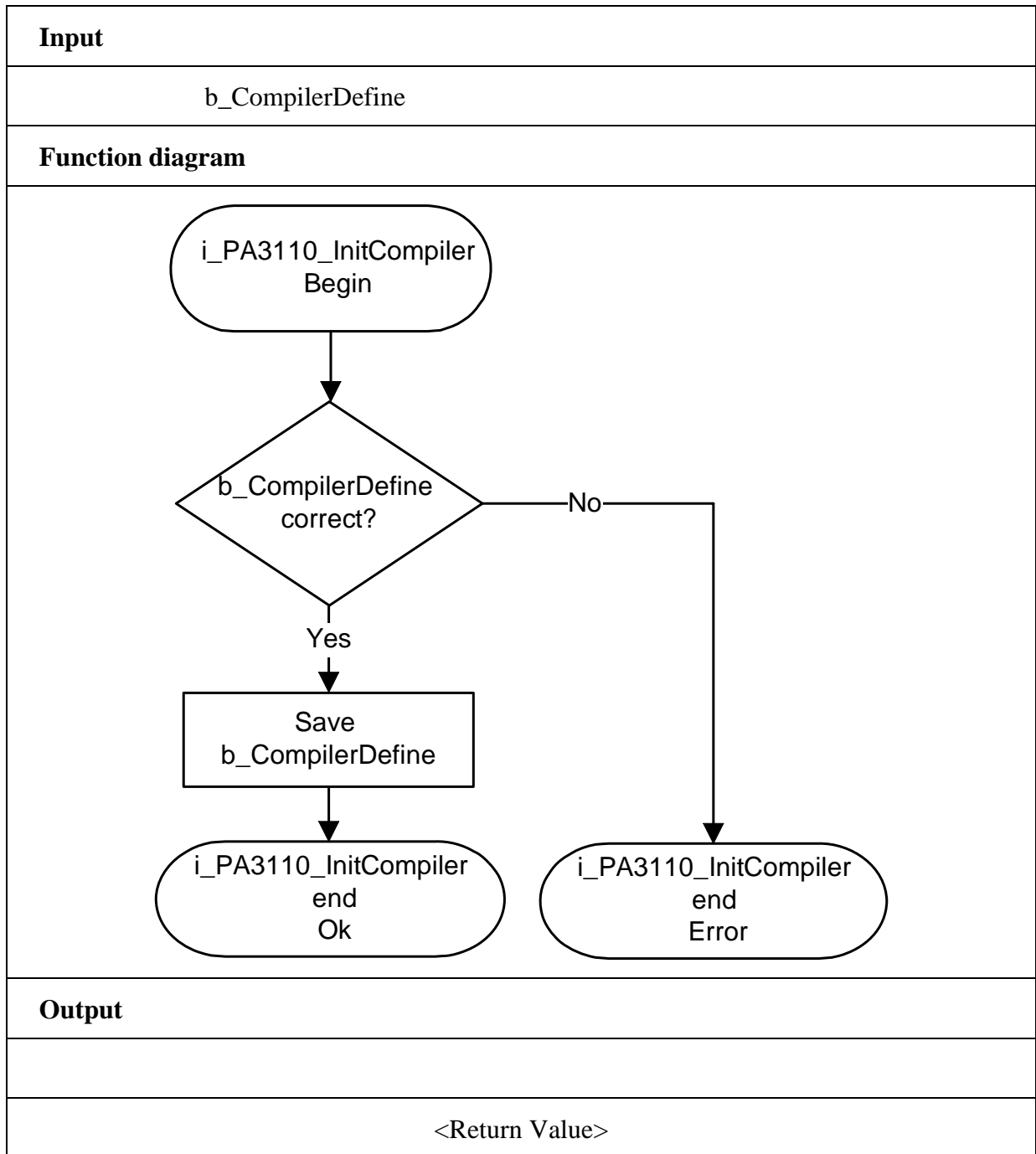
i\_ReturnValue = i\_PA3110\_InitCompiler (DLL\_COMPILER\_C);

**Return value:**

0: No error

-1: Compiler parameter is wrong





**i**

**IMPORTANT!**

**This function is only available for DOS and Windows 3.11 applications**

**2) i\_PA3110\_SetBoardInformation (...)**

**Syntax:**

```
<Return value> = i_PA3110_SetBoardInformation
                    (UINT    ui_Address,
                     BYTE    b_InterruptNbr,
                     BYTE    b_DMACHannelNbr,
                     BYTE    b_AnalogInputChannelNbr,
                     BYTE    b_AnalogOutputChannelNbr,
                     PBYTE   pb_BoardHandle)
```

**Parameters:**

**- Input:**

UINT	ui_Address	Base address of board <b>PA 3110</b>
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_DMACHannelNbr	Number of the channel (DMA5, 6 or 7). If 0, DMA is not used.
BYTE	b_AnalogInputChannelNbr	Number of the analog input channels (8 or 16)
BYTE	b_AnalogOutputChannelNbr	Number of the analog output channels (4 or 8)

**- Output:**

PBYTE	pb_BoardHandle	Handle <sup>1</sup> of board <b>PA 3110</b> to use the functions
-------	----------------	--

**Task:**

Verifies if board **PA 3110** is present. Stores the following information:

- base address,
- the interrupt number
- the number of the DMA channel
- and the number of analog input channels.
- and the number of analog output 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;
```

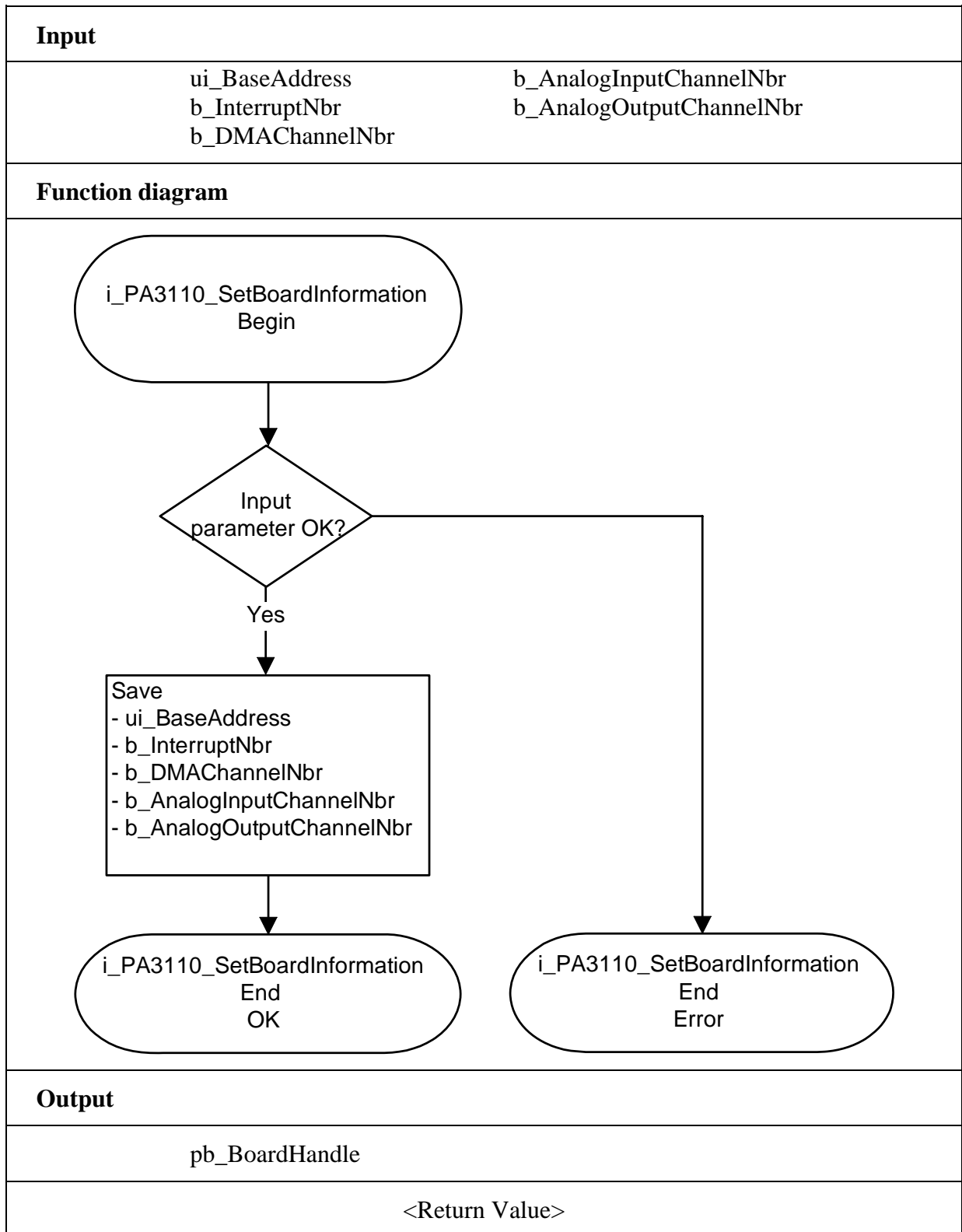
```
i_ReturnValue = i_PA3110_SetBoardInfomation (0x390, 0, 0, 16, 8,
                                             &b_BoardHandle);
```

---

<sup>1</sup> 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: Number of analog output channels is wrong
- 6: 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\_PA3110\_SetBoardInformationWin32 (...)**

**Syntax:**

```
<Return value> = i_PA3110_SetBoardInformationWin32
                                (PCHAR      pc_Identifier
                                BYTE        b_AnalogInputChannelNbr
                                BYTE        b_AnalogOutputChannelNbr
                                PBYTE      pb_BoardHandle)
```

**Parameters:**

**- Input:**

PCHAR	pc_Identifier	Identifier string for the board selection The identifier string is determined by the ADDIREG registration program.
BYTE	b_AnalogInputChannelNbr	Number of the analog input channels (8 or 16)
BYTE	b_AnalogOutputChannelNbr	Number of the analog output channels (4 or 8)

**- Output:**

PBYTE	pb_BoardHandle	Handle of the board PA 3110 to use the functions
-------	----------------	---

**Task:**

Call up all hardware information about the **PA 3110** given by the ADDIREG registration program and stores the following information:

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

Then verifies if board **PA 3110** is present.

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

Handles allow to operate several boards.

**Calling convention:**

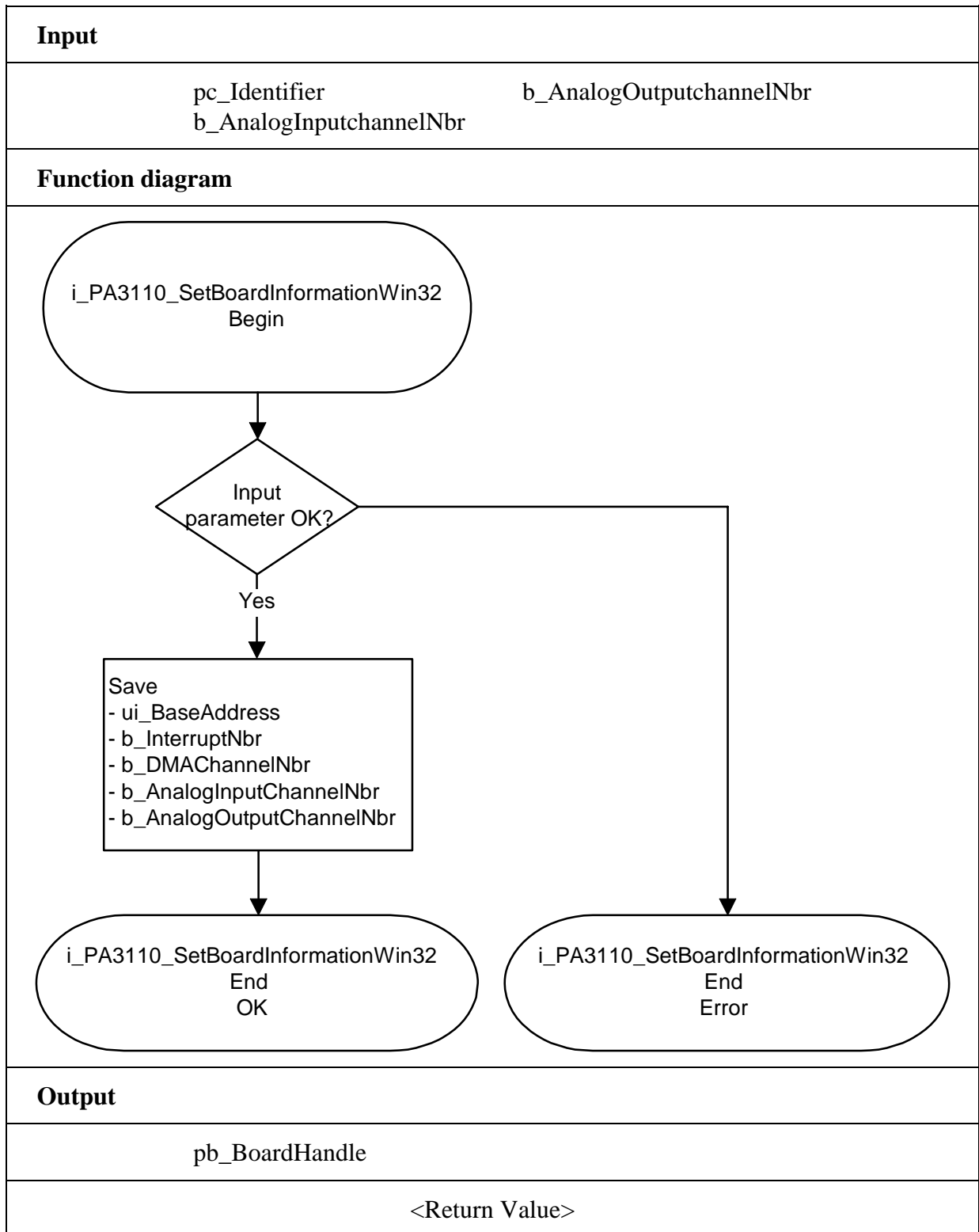
ANSI C:

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

```
i_ReturnValue = i_PA3110_SetBoardInformationWin32
                ("PA3110-00", 16, 8, &b_BoardHandle);
```

**Return value:**

- 0: No error
- 1: Board not present
- 2: Number of analog input channels is wrong
- 3: Number of analog output channels is wrong
- 4: No handle is available for the board (up to 10 handles can be used)
- 5: Error when opening the driver in Windows NT/95



#### 4) i\_PA3110\_SetAnalogInputResolution (...)

**Syntax:**

```
<Return value> = i_PA3110_SetAnalogInputResolution
                    (BYTE  b_BoardHandle,
                     BYTE  b_AnalogInputResolution)
```

**Parameters:**

**-Input:**

BYTE b\_BoardHandle                    Handle of board **PA 3110**  
 BYTE b\_AnalogInputResolution      Analog input resolution: 14 or 16 bit

**- Output:**

No output signal has occurred.

**Task:**

Sets the analog input resolution of the selected **PA 3110**.

**Calling convention:**

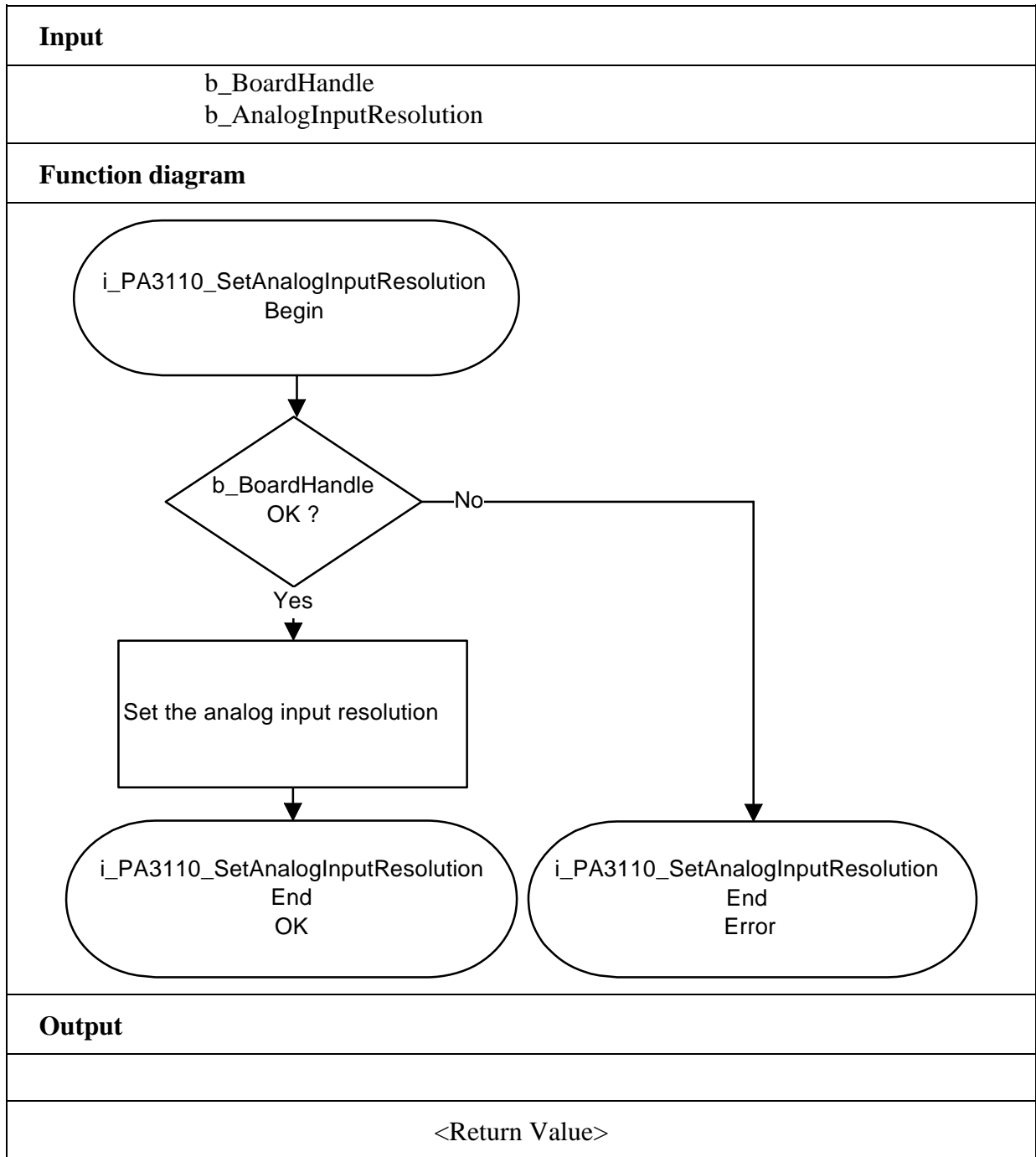
ANSI C:

```
int                    i_ReturnValue;
```

```
i_ReturnValue = i_PA3110_SetAnalogInputResolution (b_BoardHandle,
                                                    14);
```

**Return value:**

- 0: No error
- 1: The handle parameter of the board is wrong
- 2: Analog input resolution selection is wrong (14 or 16 bit)



### 5) i\_PA3110\_GetHardwareInformation (...)

**Syntax:**

```
<Return value> = i_PA3110_GetHardwareInformation
                    (BYTE   b_BoardHandle,
                     PUINT  pui_BaseAddress,
                     PBYTE  pb_InterruptNbr,
                     BYTE   pb_DMACHannelNbr)
```

**Parameters:**

**-Input:**

BYTE b\_BoardHandle Handle of board **PA 3110**

**- Output:**

PUINT pui\_BaseAddress **PA 3110** base address  
 PBYTE pb\_InterruptNbr **PA 3110** interrupt line.  
 PBYTE pb\_DMACHannelNbr Number of the DMA channel  
 (DMA5,6,7). If 0, no DMA is used

**Task:**

Returns the base address, the interrupt and DMA number of the **PA 3110**.

**Calling convention:**

ANSI C :

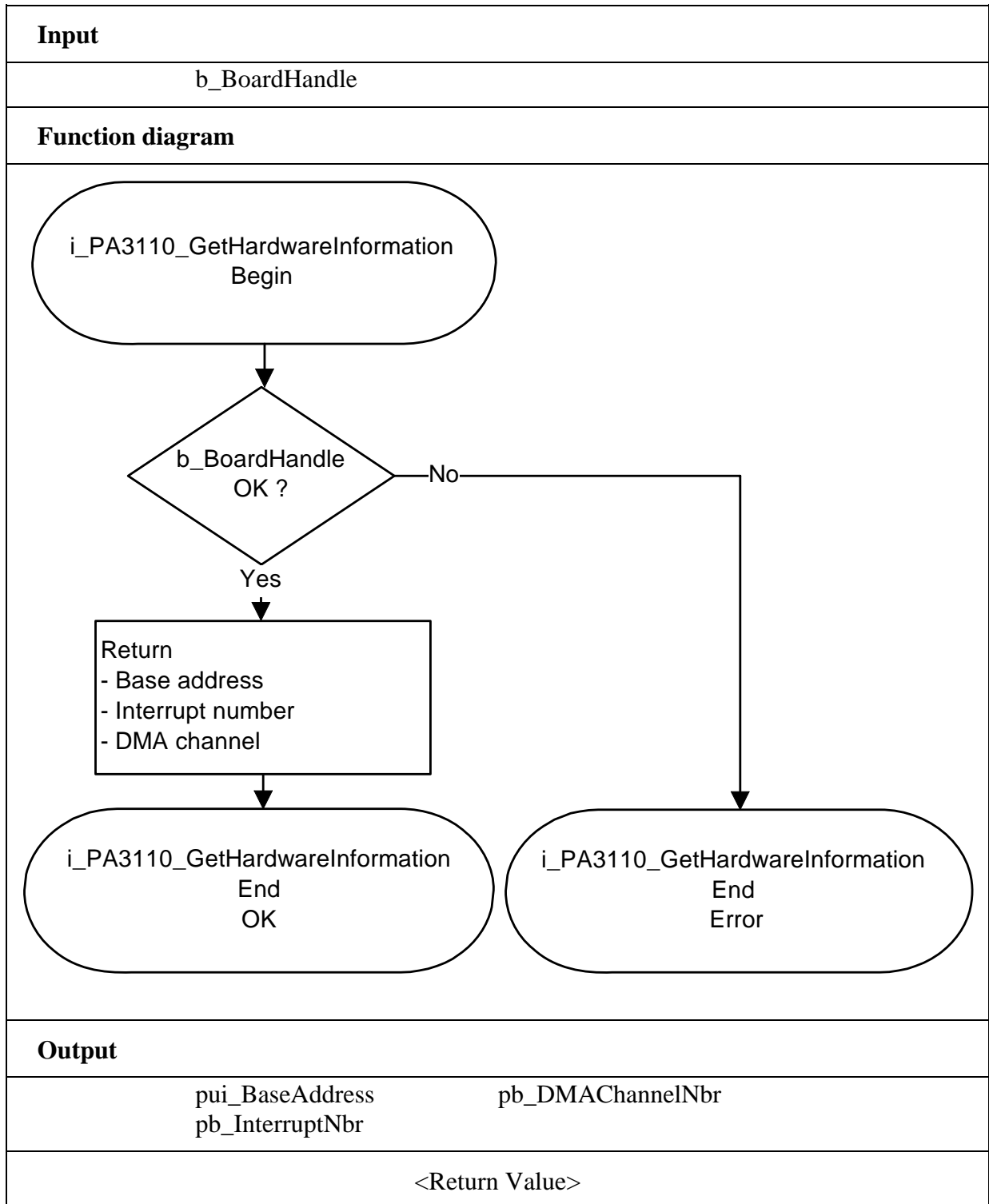
```
int          i_ReturnValue;
unsigned int ui_BaseAddress;
unsigned char b_DMACHannelNbr;
unsigned char b_InterruptNbr;
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_PA3110_GetHardwareInformation (b_BoardHandle,
                                                &ui_BaseAddress,
                                                &b_InterruptNbr,
                                                &b_DMACHannelNbr);
```

**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!

**6) i\_PA3110\_CloseBoardHandle****Syntax:**

<Return value> = i\_PA3110\_CloseBoardHandle (BYTE b\_BoardHandle)

**Parameters:****- Input:**

BYTE b\_BoardHandle Handle of board **PA 3110**

**- Output:**

No output signal has occurred.

**Task:**

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

**Calling convention:**

ANSI C:

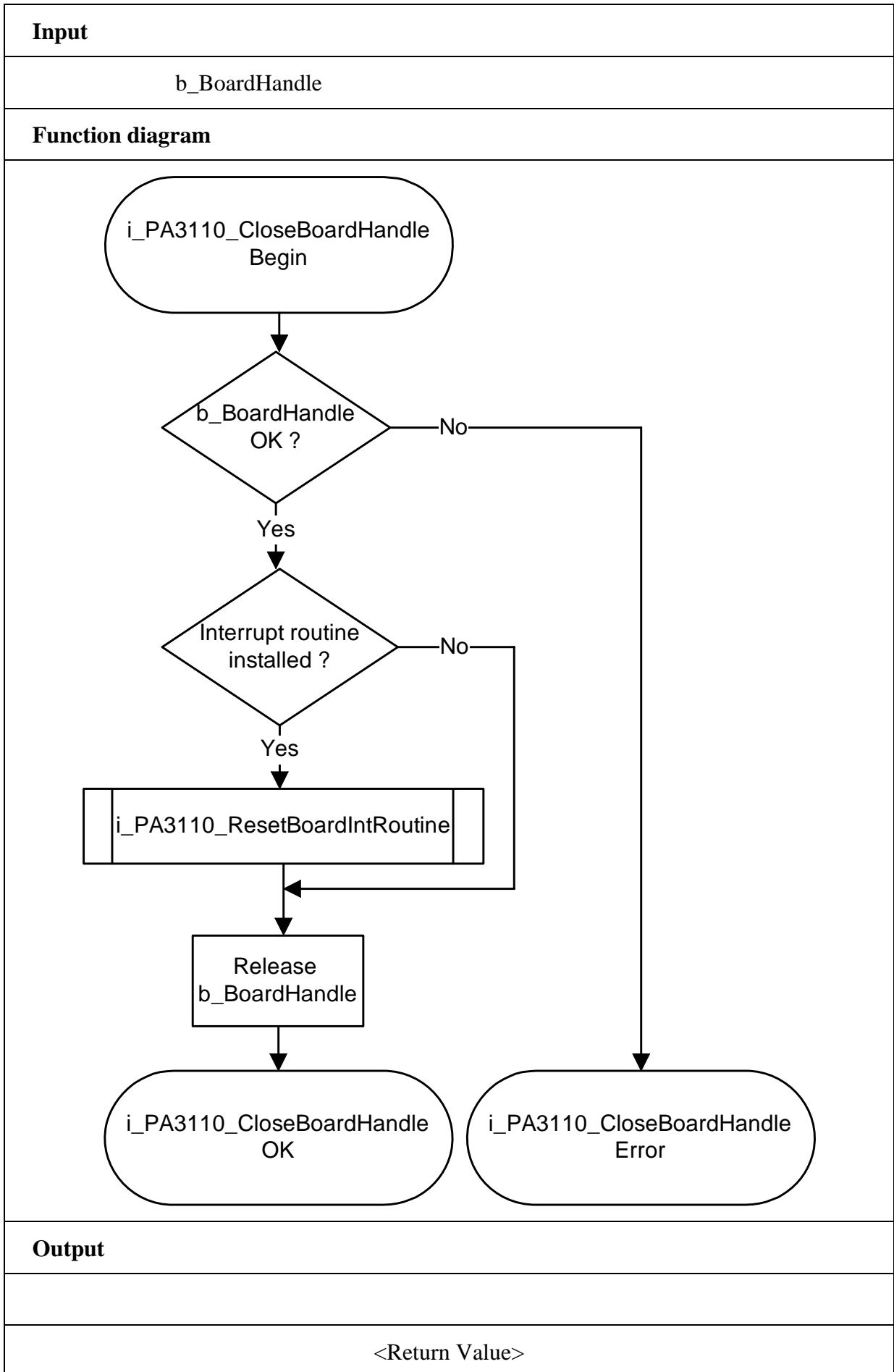
```
int          i_ReturnValue;  
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_PA3110_CloseBoardHandle (b_BoardHandle);
```

**Return value:**

0: No error

-1: The handle parameter of the board is wrong



## 3.2 Interrupt



### IMPORTANT!

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

#### 1) i\_PA3110\_SetBoardIntRoutineDos (..)

##### Syntax:

```
<Return value> = i_PA3110_SetBoardIntRoutineDos
                    (BYTE b_boardHandle
                    VOID    v_FunctionName
                    (BYTE  b_BoardHandle,
                     BYTE  b_InterruptMask,
                     PUINT pui_AnalogInputValue))
```

##### Parameters:

###### - Input:

BYTE	b_BoardHandle	Handle of board <b>PA 3110</b>
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 3110 on which an interrupt is to be enabled. It installs an 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 3110** which have to react to interrupts, call up the function as often as you operate boards **PA 3110**.

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 3110** 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 3110** 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> [0]	Number of the last analog input
	<i>pui_AnalogInputValue</i> [1]	Value of the analog input
<i>b_InterruptMask</i> = 8	<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	Conversion driven by EOS is completed
0000 1000	DMA conversion cycle is completed
0001 0000	Timer 2 has run down
0010 0000	Watchdog has run down

The user can give another name for *v\_FunctionName*, *b\_BoardHandle*, *b\_InterruptMask*, *pui\_AnalogInputValue*.

**Calling convention:**

ANSI C :

```

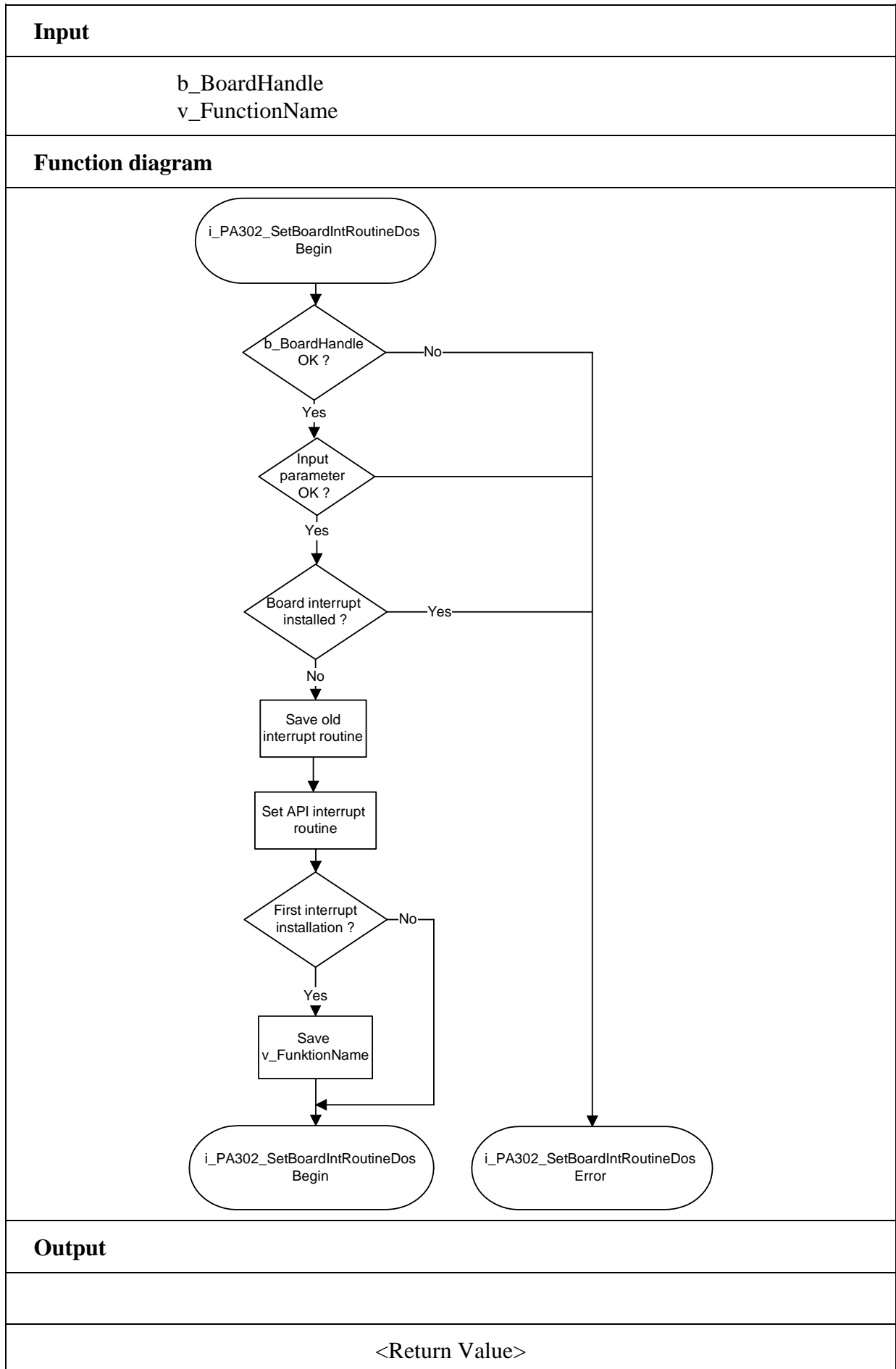
void    v_FunctionName    (unsigned char b_BoardHandle,
                           unsigned char b_InterruptMask,
                           unsigned int * ui_AnalogInputValue)
    {
        .
        .
    }

int      i_ReturnValue;
unsigned char b_BoardHandle;

i_ReturnValue = i_PA3110_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\_PA3110\_SetBoardIntRoutineVBDos (..)**

**Syntax:**

<Return value> = i\_PA3110\_SetBoardIntRoutineVBDos  
(BYTE b\_BoardHandle)

**Parameters:**

**- Input:**

BYTE b\_BoardHandle Handle of board **PA 3110**

**- Output:**

No output signal has occurred.

**Task:**

This function must be called up for each **PA 3110** 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 **PA 3110** which have to react to interrupts, call up the function as much as you operate boards **PA 3110**.

***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\_PA3110\_TestInterrupt"

This function tests the interrupt of the **PA 3110**. 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_PA3110_TestInterrupt (i_BoardHandle, _
                                         i_InterruptMask, _
                                         l_AnalogInputValue (0) )

```

.

.

.

Return

ON UEVENT GOSUB IntLabel

UEVENT ON

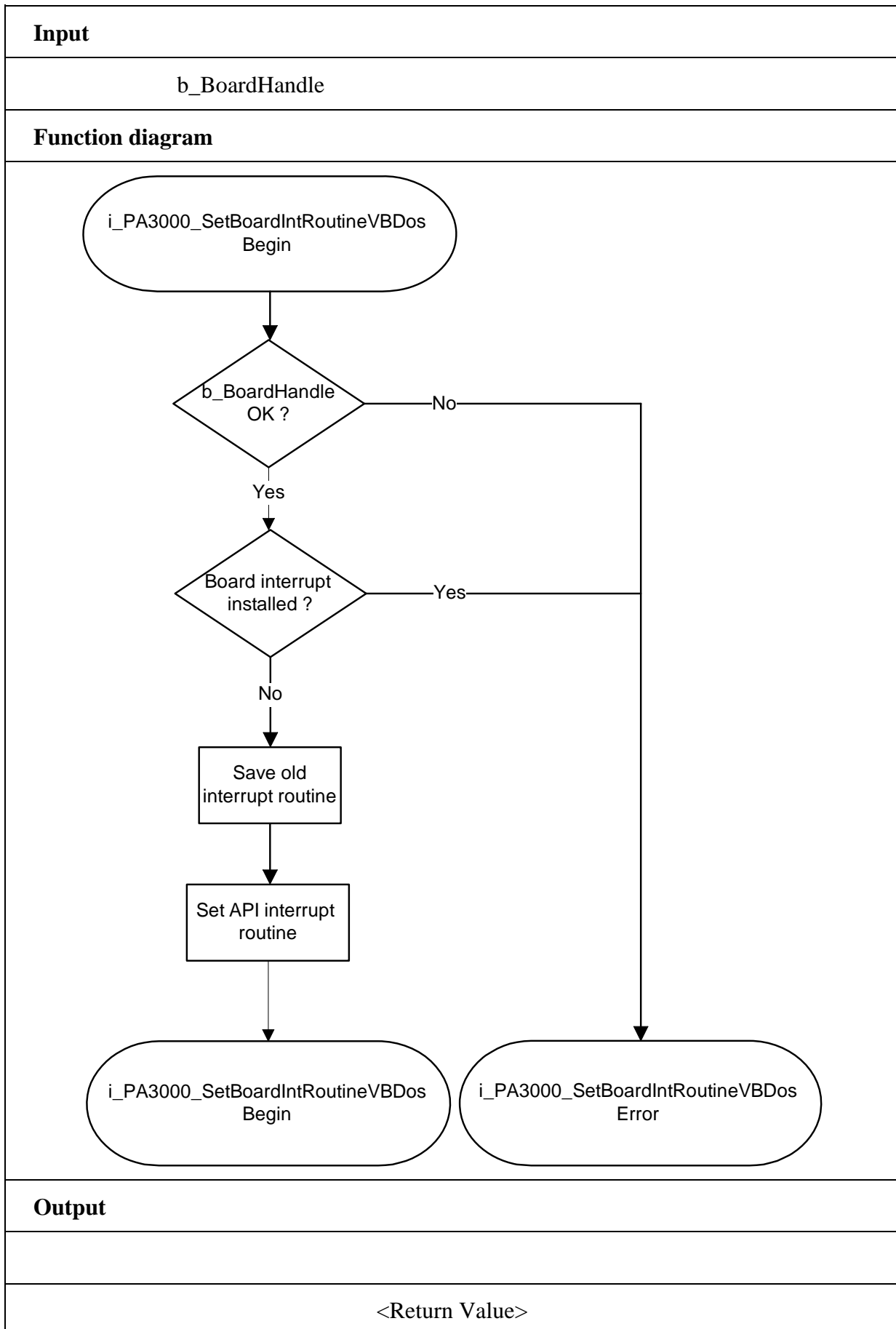
```

i_ReturnValue = i_PA3110_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\_PA3110\_SetBoardIntRoutineWin16

#### Syntax:

```
<Return value> = i_PA3110_SetBoardIntRoutineWin16
    (BYTE    b_BoardHandle,
     VOID    v_FunctionName (BYTEb_BoardHandle,
                             BYTE    b_InterruptMaske,
                             PUINT   pui_AnalogInputValue))
```

#### Parameters:

##### - Input:

BYTE b\_BoardHandle            Handle of the **PA 3110**  
 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):

- interrupts are enabled.

If you operate several boards **PA 3110** which have to react to interrupts, call up the function as often as you operate boards **PA 3110**. 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)
```

*v\_FunctionName*            Name of the user interrupt routine

*b\_BoardHandle*            Handle of the **PA 3110** 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.

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 meaning. You must use the „i\_PA3110\_TestInterrupt“ function.

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

**Calling convention:**

ANSI C :

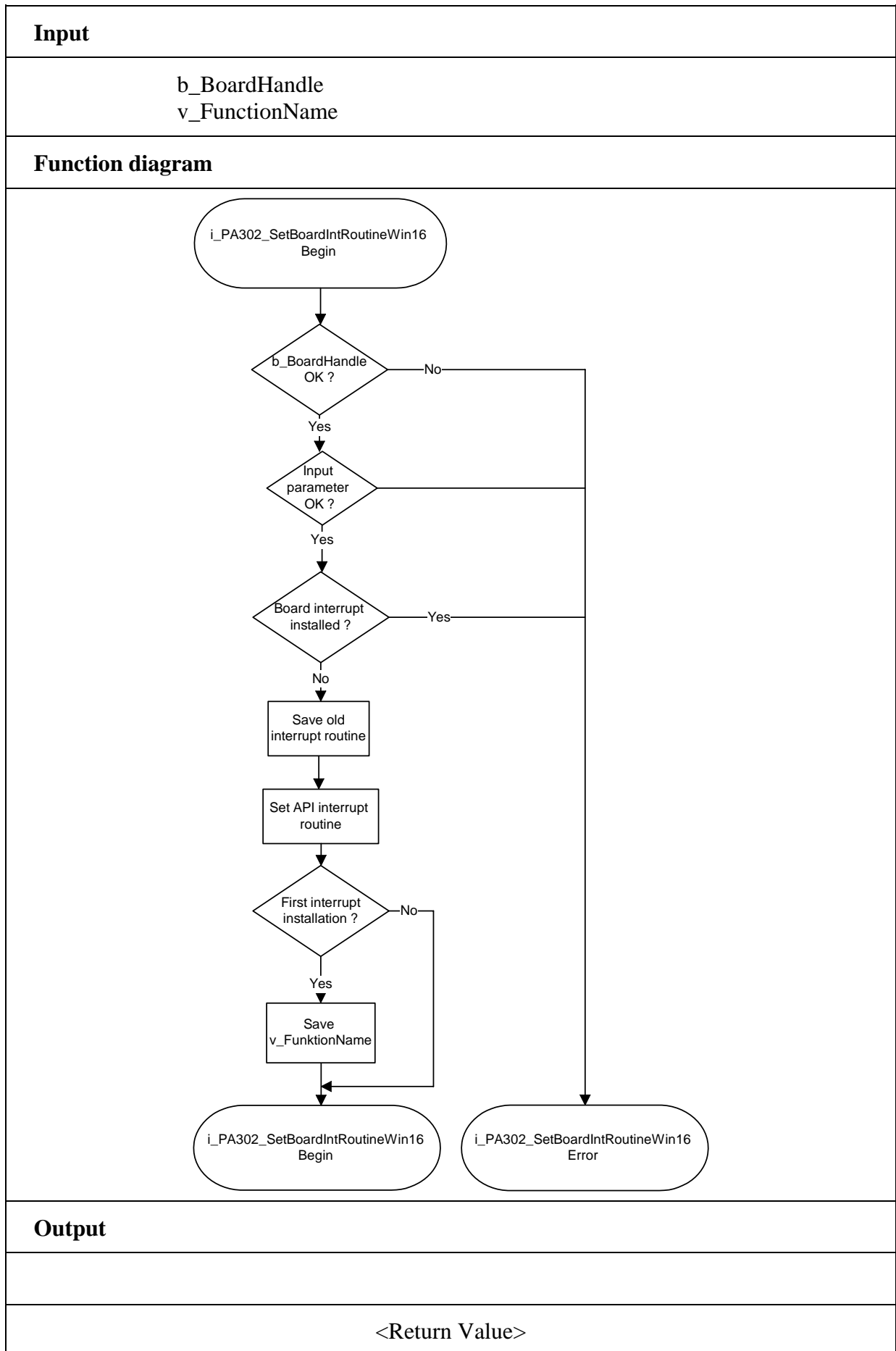
```
void v_FunctionName (unsigned char b_BoardHandle,
                   unsigned char b_InterruptMask,
                   unsigned int * ui_AnalogInputValue)
{
    .
    .
}
```

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

```
i_ReturnValue = i_PA3110_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





**IMPORTANT!**

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

**4) i\_PA3110\_SetBoardIntRoutineWin32 (..)**

**Syntax:**

```
<Return value> = i_PA3110_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 <b>PA 3110</b>
BYTE	b_UserCallingMode	PA3110_SYNCHRONOUS_MODE : The user routine is directly called by the driver interrupt routine. PA3110_ASYNCHRONOUS_MODE : The user routine is called by the driver interrupt thread.
VOID	v_FunctionName	Name of the user interrupt routine
ULONG	ul_UserSharedMemorySize	Determines the size in bytes of the user shared memory. Only used if you have selected PA3110_SYNCHRONOUS_MODE



**IMPORTANT!**

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

**- Output:**

VOID **	ppv_UserSharedMemory	User shared memory address Only used if you have selected PA3110_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 drivers operate in ring 0 and give access to hardware.
- Ring 0 has no direct access to the global variables of 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 3110** 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 PA3110\_SYNCHROUNOUS\_MODE has been selected.

If you operate several boards **PA 3110** which have to react to interrupts, call up the function as often as you operate boards **PA 3110**. The variable *v\_FunctionName* is only relevant **for the first calling**.

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

- interrupts are enabled.

***Interrupt***

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

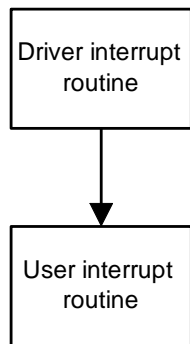
If several boards are operated and if they have to react to interrupts, the variable *b\_BoardHandle* returns the identification number (handle) of the board which has generated the interrupt.

User interrupt routine can be called:

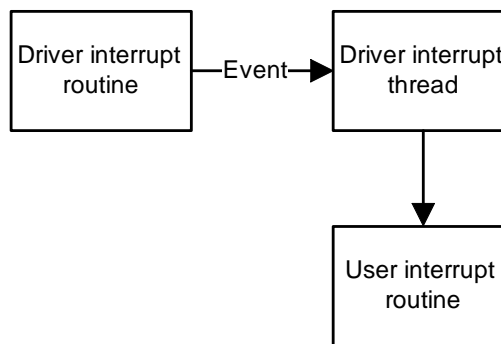
- directly by driver interrupt routine (Synchronous mode). The code of the user interrupt routine directly operates in ring 0.
- by the driver interrupt thread (Asynchronous mode). An event is generated and the interrupt thread calls up the user interrupt routine. The code of the user interrupt routine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

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

**Synchronous mode**



**Asynchronous mode**



**SYNCHRONOUS MODE**

<b>ADVANTAGE</b>	The code of the user interrupt routine is directly called by driver interrupt routine (ring 0). The time between interrupt and the user interrupt routine is reduced.
<b>RESTRICTION</b>	The user cannot debug the user interrupt routine.
	The user routine cannot call Windows API functions.
	The user routine cannot call functions which give access to global variables. The user can still use a shared memory.
	The user routine can only call PA 3110 driver functions with the following extension "i_PA3110_KRNL_XXXX"
	This mode is not available for Visual Basic

**ASYNCHRONOUS MODE**

<b>ADVANTAGE</b>	The user can debug the user interrupt routine
	The user routine can call Windows API functions.
	The user routine can call functions which give access to global variables.
	The user routine can call all PA 3110 driver functions with the following extension: "i_PA3110_XXXX"
<b>RESTRICTION</b>	The code of the user interrupt routine is called by driver interrupt thread routine (ring 3). The time between interrupt and the user interrupt routine is increased.



**Shared memory**

If you have selected the PA3110\_SYNCHRONOUS\_MODE you cannot have access to global variables. But you have the possibility to create a shared memory (*ppv\_UserSharedMemory*). The user shared memory can have all predefined compiler types or user define types.

The variable *ul\_UserSharedMemorySize* indicates the size in byte of the selected user type. A pointer of the variable *ppv\_UserSharedMemory* is given to the user interrupt routine with the variable *pv\_UserSharedMemory*. This is not possible for Visual Basic.

The user interrupt routine must have the following syntax:

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

<i>v_FunctionName</i>	Name of the user interrupt routine
<i>b_BoardHandle</i>	Handle of the PA 3110 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>	PA3110_SYNCHRONOUS_MODE : The user routine is directly called by driver interrupt routine. PA3110_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\_PA3110\_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;

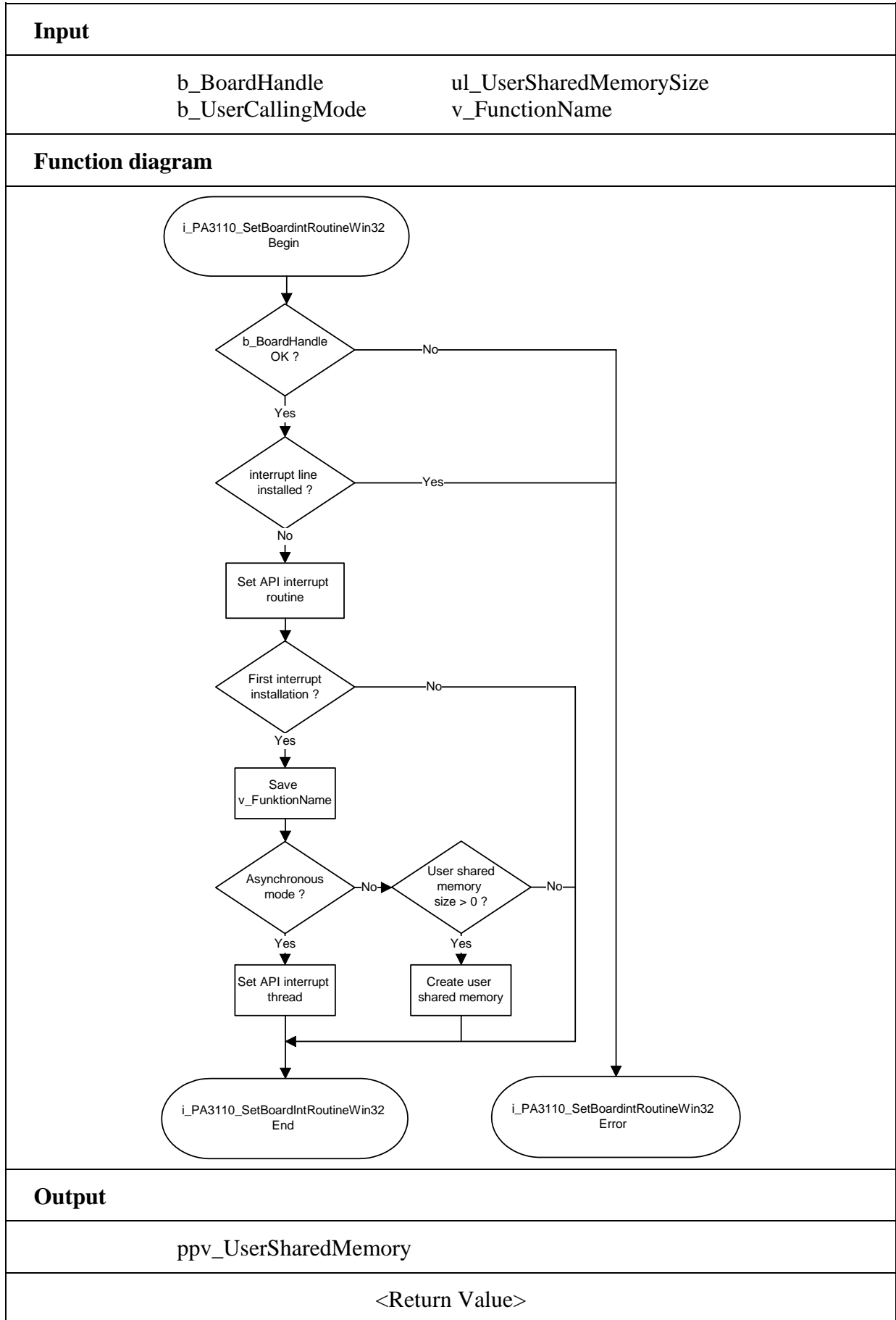
ps_InterruptSharedMemory = (str_UserStruct *) pv_UserSharedMemory;
        .
        .
    }

int      i_ReturnValue;
unsigned char  b_BoardHandle;

i_ReturnValue = i_PA3110_SetBoardIntRoutineWin32
                (b_BoardHandle, PA3110_SYNCHRONOUS_MODE,
                sizeof (str_UserStruct),
                (void **) &ps_UserSharedMemory,
                v_FunctionName);
    
```

**Return value:**

- 0: No error
- 1: The handle parameter of the board is wrong
- 2: All interrupt lines cannot have the value 0 or interrupt already installed
- 3: Parameter b\_UserCallingMode is wrong.



**i**

**IMPORTANT!**

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

**5) i\_PA3110\_TestInterrupt (..)**

**Syntax:**

<Return value> = i\_PA3110\_TestInterrupt (PBYTE pb\_BoardHandle, PBYTE pb\_InterruptMask, PUINT pui\_AnalogInputValue)

**Parameters:**

**- Input:**

No input signal occurs

**- Output:**

PBYTE pb_BoardHandle	Handle of the board <b>PA 3110</b> 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.

**Task:**

Verifies if a board **PA 3110** has generated an interrupt. If yes, the function returns the board handle and the interrupt source.

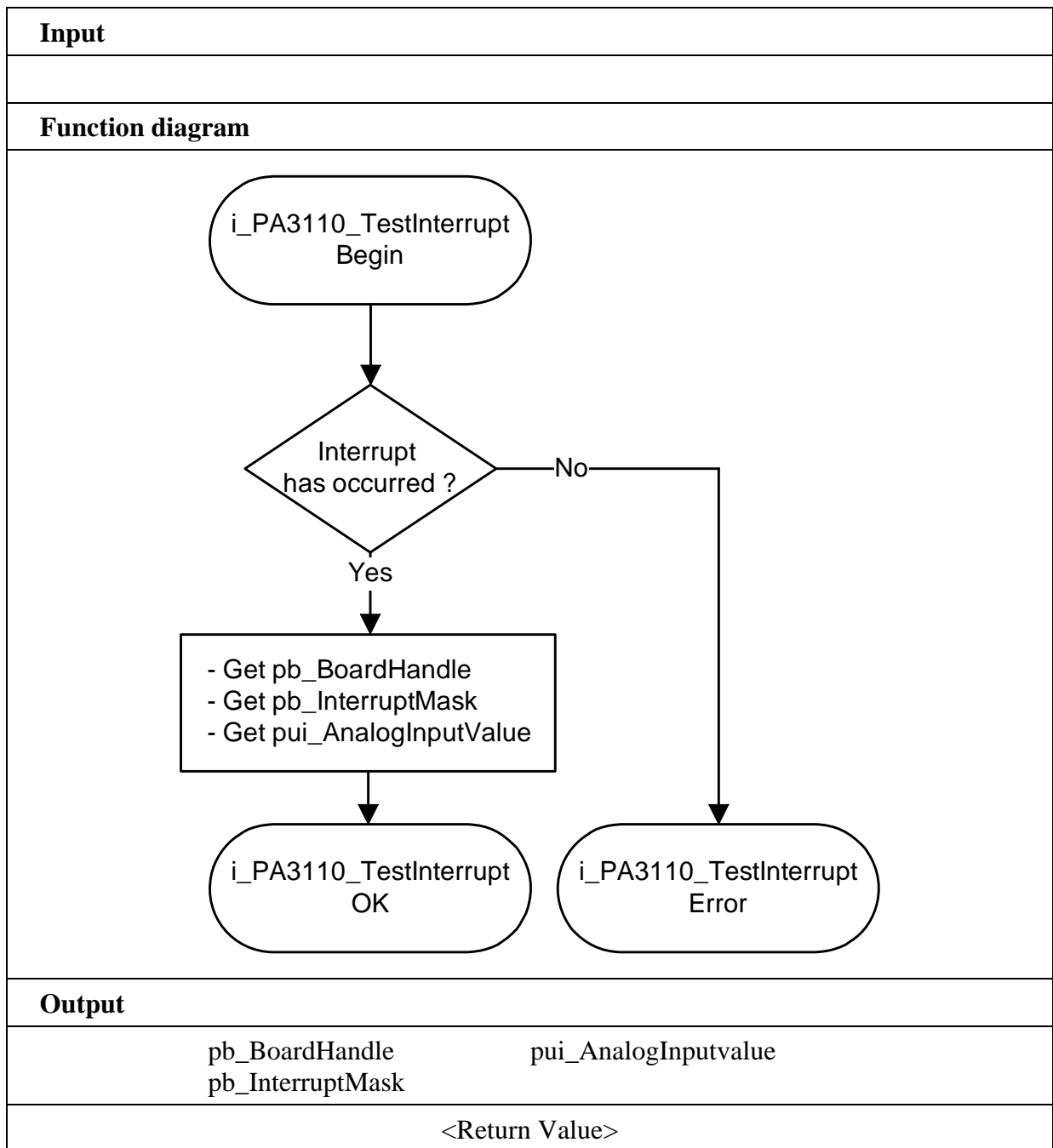
**Calling convention:**

ANSI C:

```
int i_ReturnValue;
unsigned char b_BoardHandle;
unsigned char b_InterruptMask;
unsigned int ui_AnalogInputValue [XX];
i_ReturnValue = i_PA3110_TestInterrupt (&b_BoardHandle,
&b_InterruptMask,
ui_AnalogInputValue);
```

**Return value:**

-1 No interrupt  
> 0 IRQ number



**6) i\_PA3110\_ResetBoardIntRoutine (..)****Syntax:**

<Return value> = i\_PA3110\_ResetBoardIntRoutine (BYTE b\_BoardHandle)

**Parameters:****- Input:**

BYTE b\_BoardHandle Handle of board **PA 3110**

**- Output:**

No output signal has occurred.

**Task:**

Stops the interrupt management of board **PA 3110**.

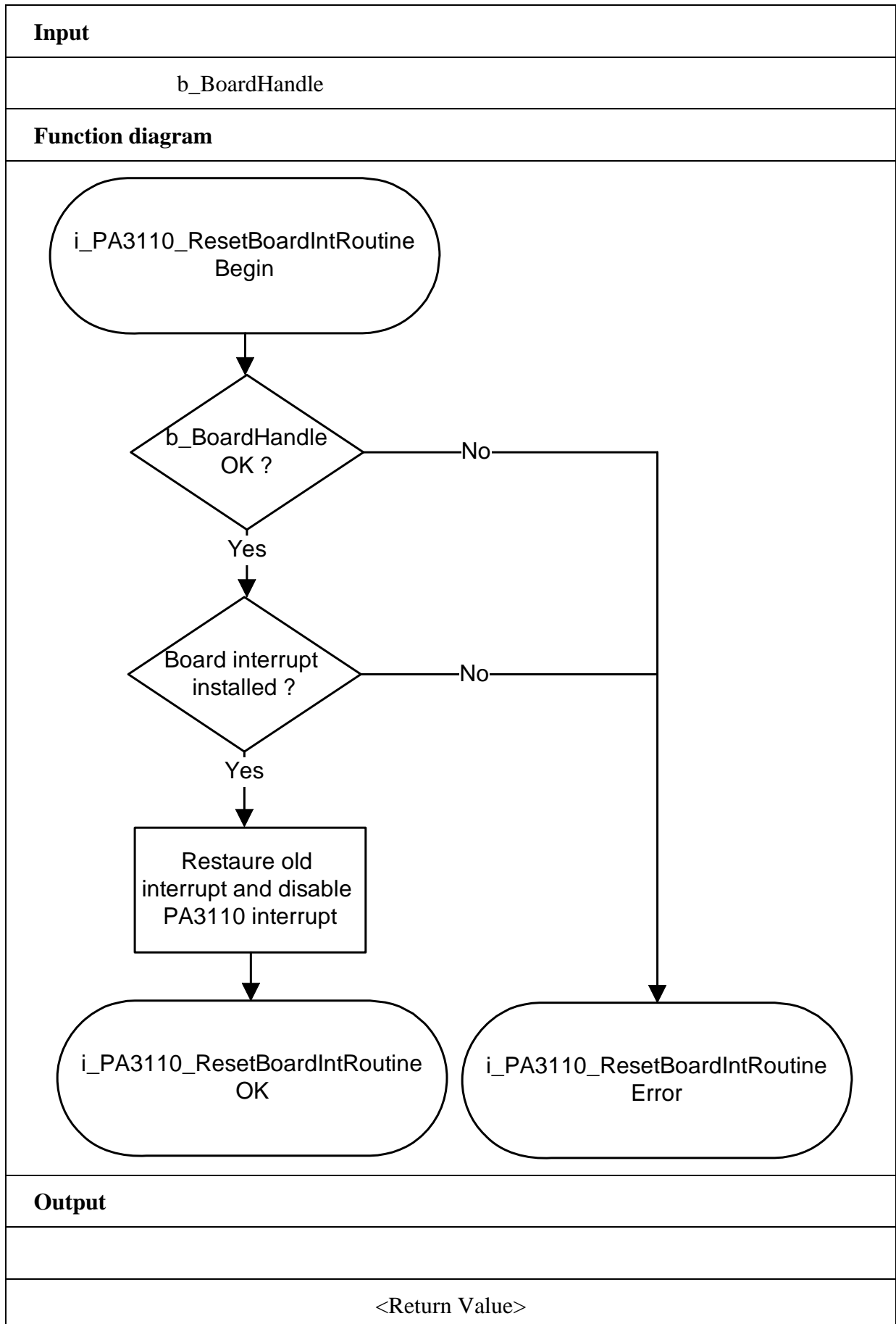
Deinstalls the user interrupt routine if the management of interrupts of all boards **PA 3110** is stopped.

**Return value:**

0: No error

-1: The handle parameter of the board is wrong

-2: No interrupt has been initialised with function  
"i\_PA3110\_SetBoardIntRoutineXXX"



### 3.3 Direct conversion of the analog input channels

#### 1) i\_PA3110\_Read1AnalogInput (...)

**Syntax:**

```
<Return value> = i_PA3110_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 <b>PA 3110</b>
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)
UINT	ui_ConvertTiming	Selection of the conversion time from 7 $\mu$ s to 45874 $\mu$ s.
BYTE	b_InterruptFlag	PA3110_ENABLE: An interrupt is generated at the end of conversion. See function "i_PA3110_SetBoardIntRoutineXXX". PA3110_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 14-bit: 0 to 16383 16-bit: 0 to 65535
-------	----------------------	--

**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*.



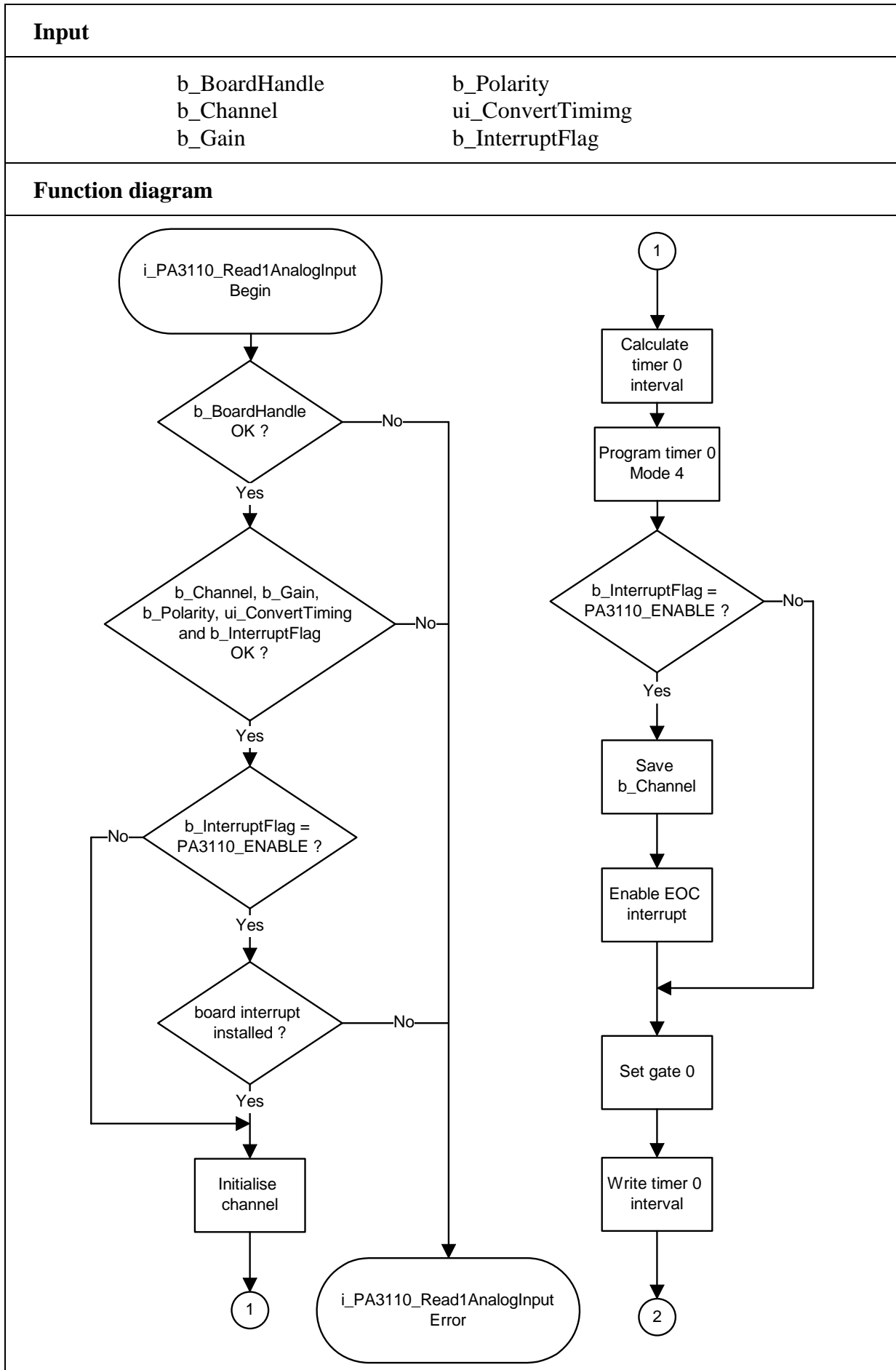
**Calling convention:**ANSI C :

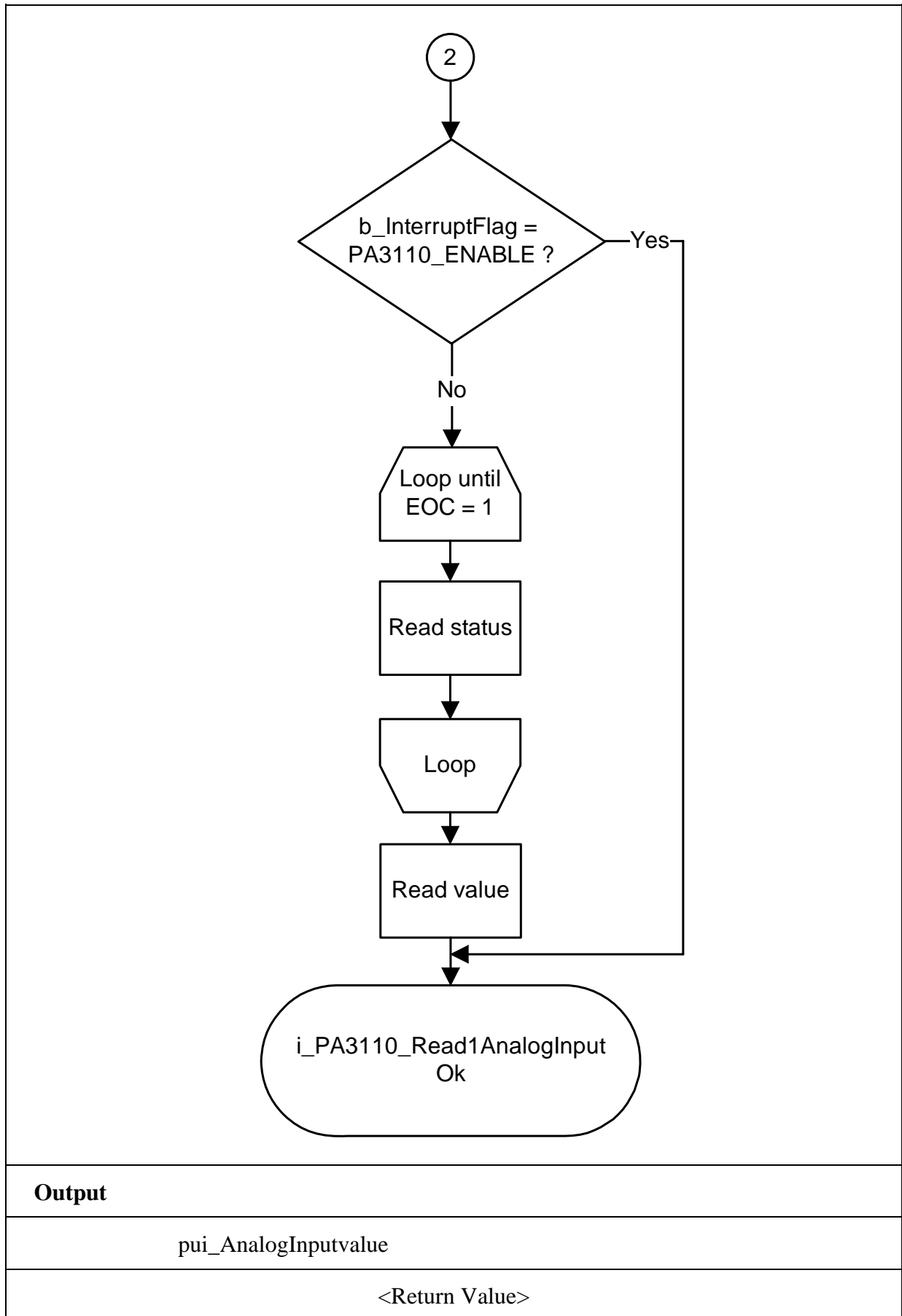
```
int          i_ReturnValue;  
unsigned char b_BoardHandle;  
unsigned int  ui_AnalogInputValue;
```

```
i_ReturnValue = i_PA3110_Read1AnalogInput (b_BoardHandle,  
PA3110_CHANNEL_0,  
PA3110_1_GAIN,  
PA3110_UNIPOLAR,  
10,  
PA3110_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\_PA3110\_SetBoardIntRoutine".





## 2) i\_PA3110\_ReadMoreAnalogInput (...)

### Syntax:

```
<Return value> = i_PA3110_ReadMoreAnalogInput
                    (BYTE   b_BoardHandle,
                     BYTE   b_SequenzArraySize,
                     PBYTE  pb_ChannelArray,
                     PBYTE  pb_GainArray,
                     PBYTE  pb_PolarityArray,
                     UINT   ui_ConvertTiming
                     BYTE   b_InterruptFlag,
                     PUINT  pui_AnalogInputValueArray)
```

### Parameters:

#### - Input:

BYTE	b_BoardHandle	Handle of board <b>PA 3110</b>
BYTE	b_SequenzArraySize	Size of the scan lists (1 up 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.
UINT	ui_ConvertTiming	Selection of the conversion time From 10 $\mu$ s up to 45874 $\mu$ s.
BYTE	b_InterruptFlag	PA3110_ENABLE : An interrupt is generated when the last conversion of the channel group is completed (EOS). See function "i_PA3110_SetBoardIntRoutine". PA3110_DISABLE : No interrupt is generated at the end of conversion. The analog values are located in parameter <i>pui_AnalogInputValueArray</i> .

#### - Output:

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

### Task:

Reads several analog input channels.

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 channel. The gain is defined with parameter *pb\_GainArray* for each analog input channel.

The input voltage range is defined with parameter *pb\_PolarityArray* for each analog input channel.

**Calling convention:**ANSI C :

```
int          i_ReturnValue;
unsigned char b_BoardHandle;
unsigned char b_ChannelArray    [16];
unsigned char b_GainArray       [16];
unsigned char b_PolarityArray   [16];
unsigned int  ui_AnalogInputValueArray [16];
```

```
i_ReturnValue = i_PA3110_ReadAllAnalogInput (b_BoardHandle,
                                             16,
                                             b_ChannelArray,
                                             b_GainArray,
                                             b_PolarityArray,
                                             PA3110_DISABLE,
                                             ui_AnalogInputValue);
```

**Return value:**

0: No error

-1: The handle parameter of the board is wrong

-2: The size of the scan list is wrong

-3: Wrong parameter detected in table "pb\_ChannelArray"

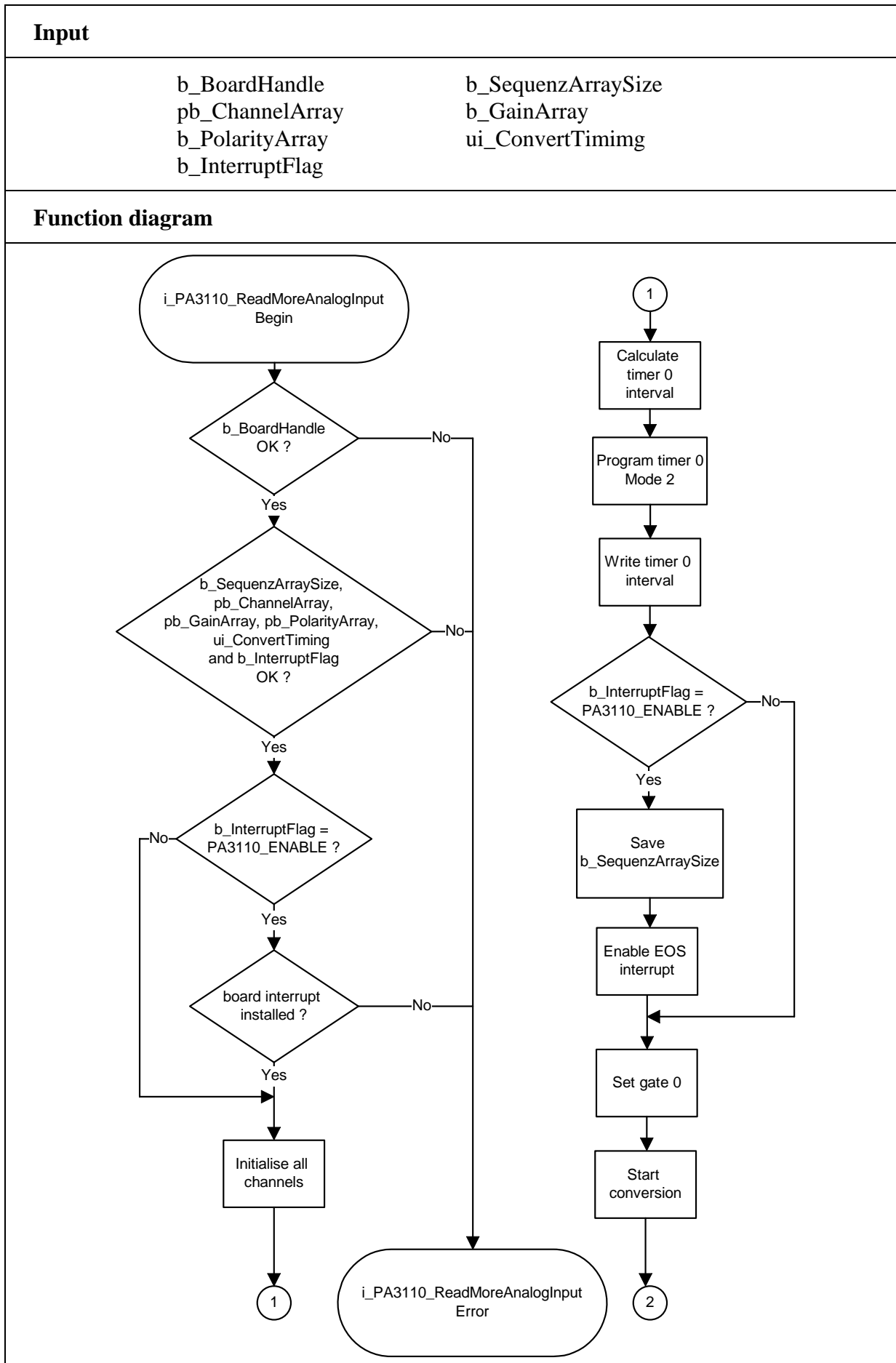
-4: Wrong parameter detected in table "pb\_GainArray"

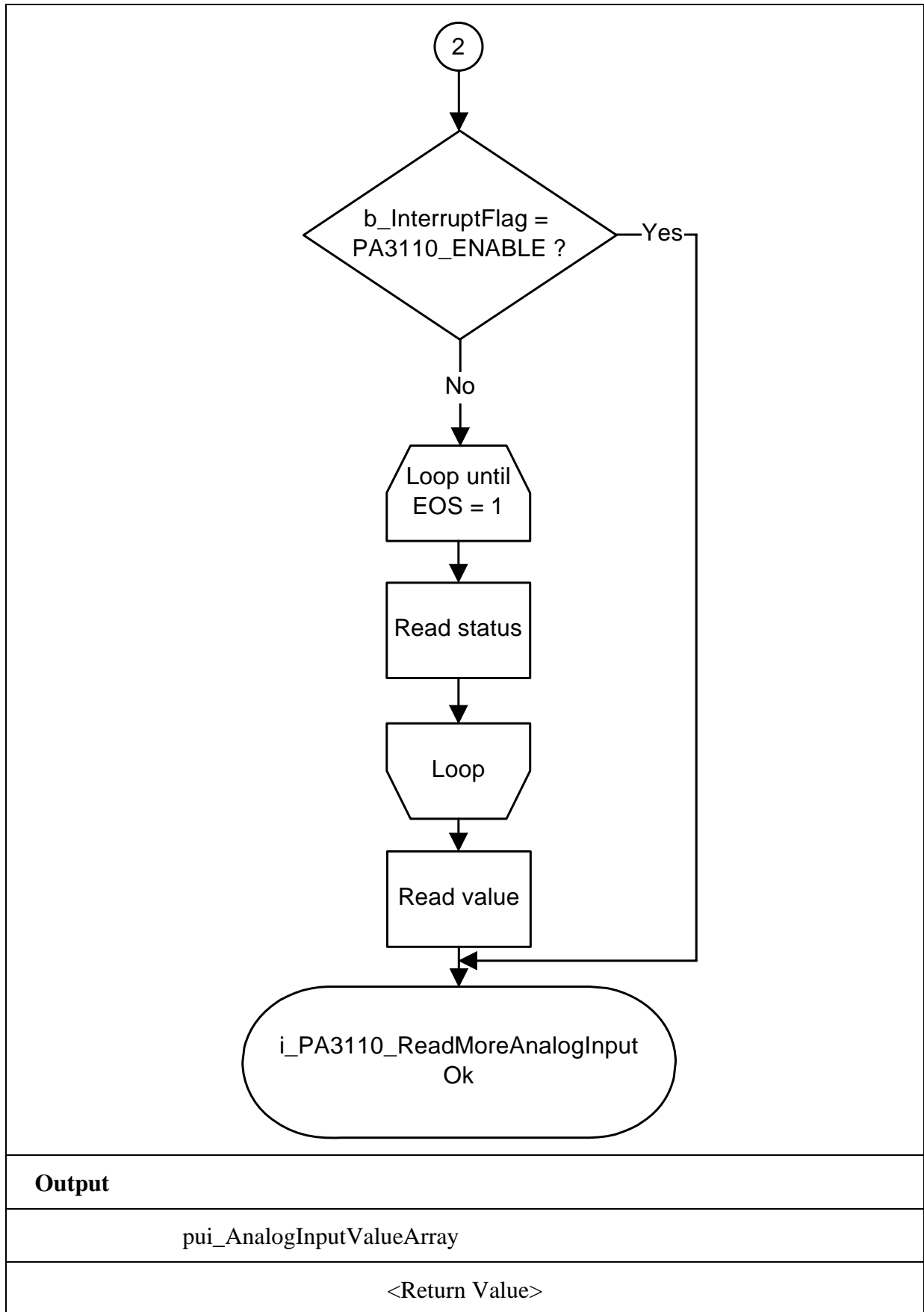
-5: Wrong parameter detected in table "pb\_PolarityArray"

-6: Selected conversion time is wrong

-7: Wrong parameter entered for *b\_InterruptFlag*, or the user interrupt routine has not been installed.

See function "i\_PA3110\_SetBoardIntRoutine".





### 3.4 Cyclic conversion of analog input channels

#### 1) i\_PA3110\_InitAnalogInputAcquisition (...)

**Syntax:**

```
<Return value> = i_PA3110_InitAnalogInputAcquisition
                    (BYTE      b_BoardHandle,
                    BYTE      b_SequenzArraySize,
                    PBYTE     pb_ChannelArray,
                    PBYTE     pb_GainArray,
                    PBYTE     pb_PolarityArray,
                    BYTE      b_AcquisitionMode,
                    UINT      ui_AcquisitionTiming,
                    LONG      l_DelayTiming,
                    UINT      ui_NumberOfAcquisition,
                    BYTE      b_DMAUsed,
                    BYTE      b_AcquisitionCycle)
```

**Parameters:**

**- Input:**

BYTE	b_BoardHandle	Handle of board <b>PA 3110</b>
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: - PA3110_SIMPLE_MODUS: A conversion occurs every <i>ui_AcquisitionTiming</i> (time interval) See example 2. - PA3110_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.



UINT	ui_AcquisitionTiming	Time in $\mu\text{s}$ between 2 conversions of successive input channels - from 7 $\mu\text{s}$ to 45874 $\mu\text{s}$ , if you use the option PA3110_DMA_USED. - from 1500 $\mu\text{s}$ to 45874 $\mu\text{s}$ , if you use the option PA3110_DMA_NOT_USED See example 1 and 2.
LONG	l_DelayTiming	Waiting time in $\mu\text{s}$ between two conversion cycles (from 70 $\mu\text{s}$ to 4587400 $\mu\text{s}$ ). This parameter has only a signification if you use the mode PA3110_DELAY_MODUS.
UINT	ui_NumberOfAcquisition	This parameter determines how many conversions have to happen (1 to 32767).
BYTE	b_DMAUsed	Determines if DMA must be used or not. - PA3110_DMA_USED: All the conversion values are saved and returned to the user. An interrupt is generated when the conversions (set by <i>ui_NumberOfAcquisition</i> ) are completed. See function "i_PA3110_SetBoardIntRoutineXXX". - PA3110_DMA_NOT_USED: An interrupt is generated at the end of each conversion and the analog value is returned to the user through the interrupt routine. See function "i_PA3110_SetBoardIntRoutineXXX".
BYTE	b_AcquisitionCycle	Determines the type of DMA conversion. - PA3110_CONTINUOUS: An interrupt is generated each time a DMA conversion cycle is completed. A new DMA conversion cycle is started. - PA3110_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 (PA3110\_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\_InterruptMaske*. The DMA buffer is returned through the parameter *pui\_AnalogInputValue*.

See function "i\_PA3110\_SetBoardIntRoutineXXX".

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 PA3110\_DELAY\_MODUS).
- determine if DMA must be used (parameter *b\_DMAUsed*)
- enter the number of acquisitions through the parameter *ul\_NumberOfAcquisition* (if DMA is used)
- enter the DMA conversion cycle through the parameter *b\_AcquisitionCycle* (if DMA is used)

**Table 3-3: Selection of the analog input channels**

pb_ChannelArray Parameter	Analog input channel	Define decimal value
PA3110_CHANNEL_0	0	0
PA3110_CHANNEL_1	1	1
PA3110_CHANNEL_2	2	2
PA3110_CHANNEL_3	3	3
PA3110_CHANNEL_4	4	4
PA3110_CHANNEL_5	5	5
PA3110_CHANNEL_6	6	6
PA3110_CHANNEL_7	7	7
PA3110_CHANNEL_8	8	8
PA3110_CHANNEL_9	9	9
PA3110_CHANNEL_10	10	10
PA3110_CHANNEL_11	11	11
PA3110_CHANNEL_12	12	12
PA3110_CHANNEL_13	13	13
PA3110_CHANNEL_14	14	14
PA3110_CHANNEL_15	15	15

**Table 3-4: Gain selection**

pb_GainArray Parameter	Gain	Define decimal value
PA3110_1_GAIN	1	0
PA3110_2_GAIN	2	16
PA3110_5_GAIN	5	32
PA3110_10_GAIN	10	48

**Table 3-5 Selection of the input voltage range**

pb_PolarityArray parameter	Voltage range	Define decimal value
PA3110_UNIPOLAR	0-10V (gain 1)	128
PA3110_BIPOLAR	±10V (gain 1)	0

**Examples:**

## Example 1:

```

b_RamArraySize           = 16
pb_ChannelArray [0]     = PA3110_CHANNEL_0
pb_ChannelArray [1]     = PA3110_CHANNEL_1
pb_ChannelArray [2]     = PA3110_CHANNEL_2
pb_ChannelArray [3]     = PA3110_CHANNEL_3
pb_ChannelArray [4]     = PA3110_CHANNEL_2
pb_ChannelArray [5]     = PA3110_CHANNEL_1
pb_ChannelArray [6]     = PA3110_CHANNEL_0
pb_ChannelArray [7]     = PA3110_CHANNEL_4
pb_ChannelArray [8]     = PA3110_CHANNEL_5
pb_ChannelArray [9]     = PA3110_CHANNEL_6
pb_ChannelArray [10]    = PA3110_CHANNEL_7
pb_ChannelArray [11]    = PA3110_CHANNEL_8
pb_ChannelArray [12]    = PA3110_CHANNEL_9
pb_ChannelArray [13]    = PA3110_CHANNEL_10
pb_ChannelArray [14]    = PA3110_CHANNEL_11
pb_ChannelArray [15]    = PA3110_CHANNEL_12

```

```

pb_GainArray [0]       = PA3110_1_GAIN
pb_GainArray [1]       = PA3110_1_GAIN
pb_GainArray [2]       = PA3110_1_GAIN
pb_GainArray [3]       = PA3110_1_GAIN
pb_GainArray [4]       = PA3110_5_GAIN
pb_GainArray [5]       = PA3110_5_GAIN
pb_GainArray [6]       = PA3110_5_GAIN
pb_GainArray [7]       = PA3110_10_GAIN
pb_GainArray [8]       = PA3110_10_GAIN
pb_GainArray [9]       = PA3110_10_GAIN
pb_GainArray [10]      = PA3110_1_GAIN
pb_GainArray [11]      = PA3110_1_GAIN
pb_GainArray [12]      = PA3110_2_GAIN
pb_GainArray [13]      = PA3110_2_GAIN
pb_GainArray [14]      = PA3110_2_GAIN
pb_GainArray [15]      = PA3110_1_GAIN

pb_PolarityArray [0]   = PA3110_UNIPOLAR
pb_PolarityArray [1]   = PA3110_UNIPOLAR
pb_PolarityArray [2]   = PA3110_UNIPOLAR
pb_PolarityArray [3]   = PA3110_UNIPOLAR
pb_PolarityArray [4]   = PA3110_BIPOLAR
pb_PolarityArray [5]   = PA3110_BIPOLAR
pb_PolarityArray [6]   = PA3110_BIPOLAR
pb_PolarityArray [7]   = PA3110_UNIPOLAR
pb_PolarityArray [8]   = PA3110_UNIPOLAR
pb_PolarityArray [9]   = PA3110_BIPOLAR
pb_PolarityArray [10]  = PA3110_UNIPOLAR
pb_PolarityArray [11]  = PA3110_UNIPOLAR
pb_PolarityArray [12]  = PA3110_UNIPOLAR
pb_PolarityArray [13]  = PA3110_UNIPOLAR
pb_PolarityArray [14]  = PA3110_UNIPOLAR
pb_PolarityArray [15]  = PA3110_UNIPOLAR

```

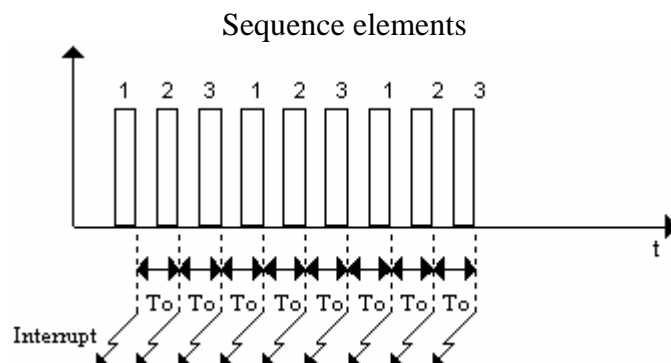
In this example the priority is set as follows:

Element	Analog input channel	Input voltage range
1	0	0-10 V
2	1	0-10 V
3	2	0-10 V
4	3	0-10 V
5	2	$\pm 2$ V
6	1	$\pm 2$ V
7	0	$\pm 2$ V
8	4	0-1 V
9	5	0-1 V
10	6	$\pm 1$ V
11	7	0-10 V
12	8	0-10 V
13	9	0-5 V
14	10	0-5 V
15	11	0-5 V
16	12	0-10 V

Example 2:

```
i_PA3110_InitAnalogInputAcquisition
    (b_BoardHandle,
     3,
     pb_ChannelArray,
     pb_GainArray,
     pb_PolarityArray,
     PA3110_SIMPLE_MODUS,
     T0,
     0,
     3,
     PA3110_DMA_NOT_USED,
     PA3110_SINGLE)
```

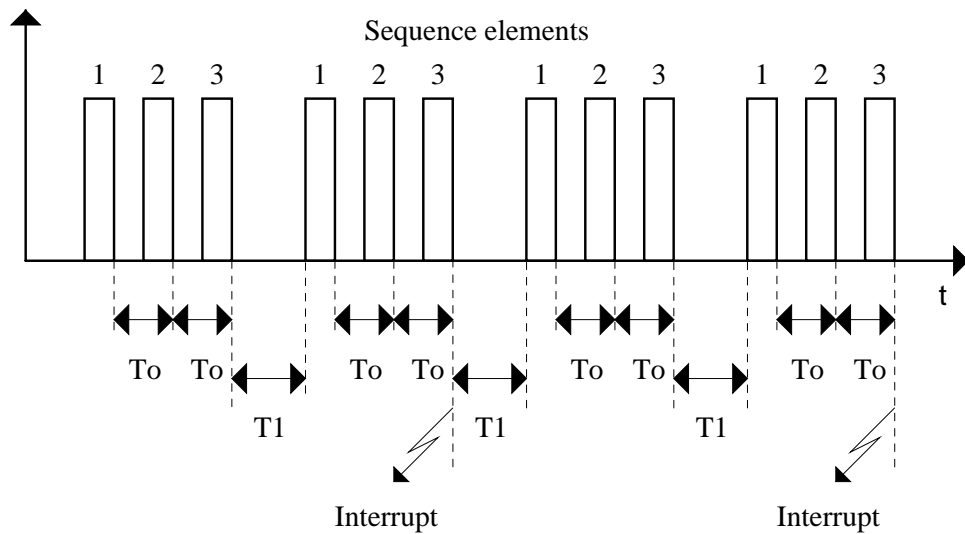
- b\_AcquisitionMode = PA3110\_SIMPLE\_MODUS
- b\_ExternTrigger = PA3110\_DISABLE
- ui\_AcquisitionTiming = T0
- b\_DMAUsed = PA3110\_DMA\_NOT\_USED
- ui\_NumberOfAcquisition = 3



Example 3: Cyclic conversion with DMA without external trigger

```
i_PA3110_InitAnalogInputAcquisition
    (b_BoardHandle,
     3,
     pb_ChannelArray,
     pb_GainArray,
     pb_PolarityArray,
     PA3110_DELAY_MODUS,
     T0,
     T1,
     6,
     PA3110_DMA_USED,
     PA3110_CONTINUOUS)
```

- b\_AcquisitionMode = PA3110\_DELAY\_MODUS
- ui\_NumberOfAcquisition = 6
- ui\_AcquisitionTiming = T0
- l\_DelayTiming = T1
- b\_DMAUsed = PA3110\_DMA\_USED
- b\_AcquisitionCycle = PA3110\_CONTINUOUS
- b\_ExternTrigger = PA3110\_DISABLE



**Calling convention:**ANSI C :

```

int          i_ReturnValue;
unsigned char b_BoardHandle;
unsigned int  ui_AnalogInputValue;
unsigned char b_ChannelArray    [16];
unsigned char b_GainArray       [16];
unsigned char b_PolarityArray   [16];

```

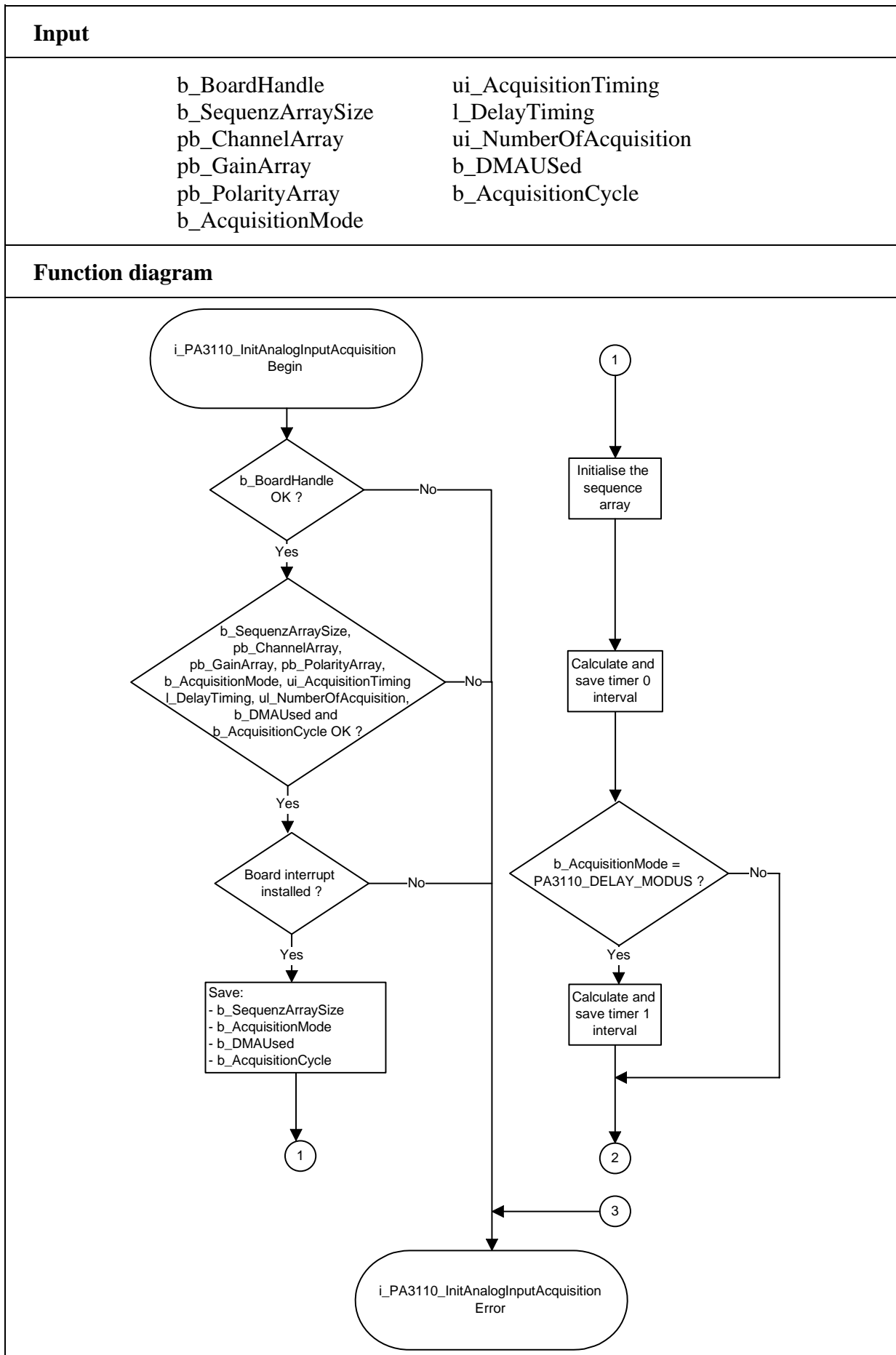
```

i_ReturnValue = i_PA3110_InitAnalogInputAcquisition
                (b_BoardHandle,
                 16,
                 b_ChannelArray,
                 b_GainArray,
                 b_PolarityArray,
                 PA3110_DELAY_MODUS,
                 100,
                 3110,
                 1000,
                 DMA_NOT_USED,
                 PA3110_SINGLE);

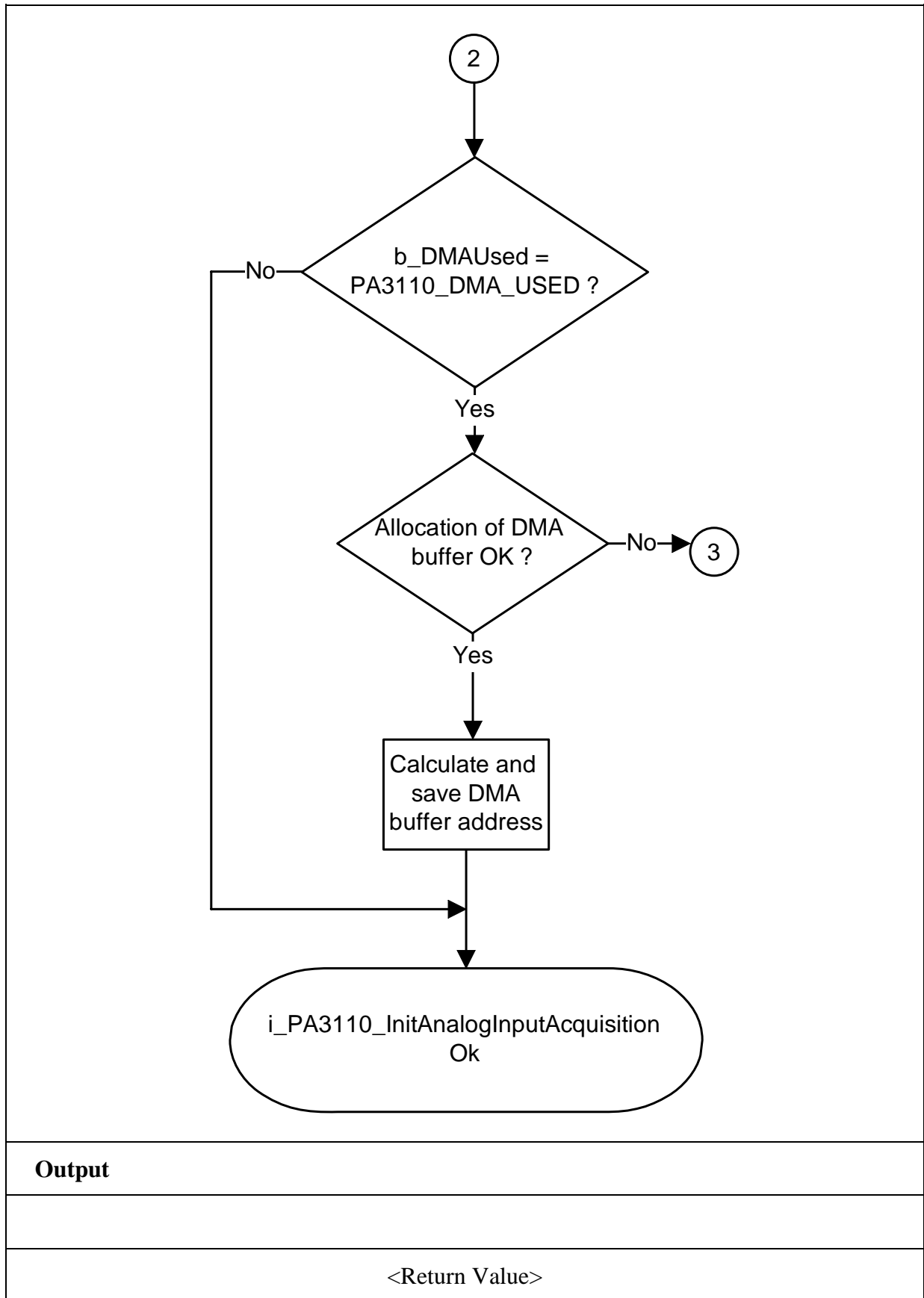
```

**Return value:**

- 0: No error
- 1: The handle parameter of the board is wrong
- 2: The user interrupt routine is not installed.  
See function "i\_PA3110\_SetBoardIntRoutine"
- 3: The size of the sequence table is wrong
- 4: Wrong parameter in table "pb\_ChannelArray"
- 5: Wrong parameter in table "pb\_GainArray"
- 6: Wrong parameter in table "pb\_PolarityArray"
- 7: The number of analog conversions is wrong (1 to 32767)
- 8: The waiting time between 2 conversion cycles is too high
- 9: The selected time for *ui\_AcquisitionTiming* or *l\_DelayTiming* is wrong
- 10: The parameter *b\_DMAUsed* is wrong
- 11: The parametered running time of the DMA conversion cycle is wrong  
(PA3110\_CONTINUOUS or PA3110\_SINGLE)
- 12: The parametered conversion cycle is wrong  
(PA3110\_SIMPLE\_MODUS or PA3110\_DELAY\_MODUS)
- 13: DMA channel not installed  
See function "i\_PA3110\_SetBoardInformation"
- 14: Not enough memory available.







## 2) i\_PA3110\_StartAnalogInputAcquisition (...)

**Syntax:**

<Return value> = i\_PA3110\_StartAnalogInputAcquisition  
(BYTE b\_BoardHandle)

**Parameters:**

**- Input:**

BYTE b\_BoardHandle Handle of the PA 3110

**- Output:**

No output signal has occurred.

**Task:**

Starts the cyclic conversion. It has been previously initialised with function "i\_PA3110\_InitAnalogInputAcquisition".

**Calling convention:**

ANSI C :

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

```
i_ReturnValue = i_PA3110_StartAnalogInputAcquisition (b_BoardHandle);
```

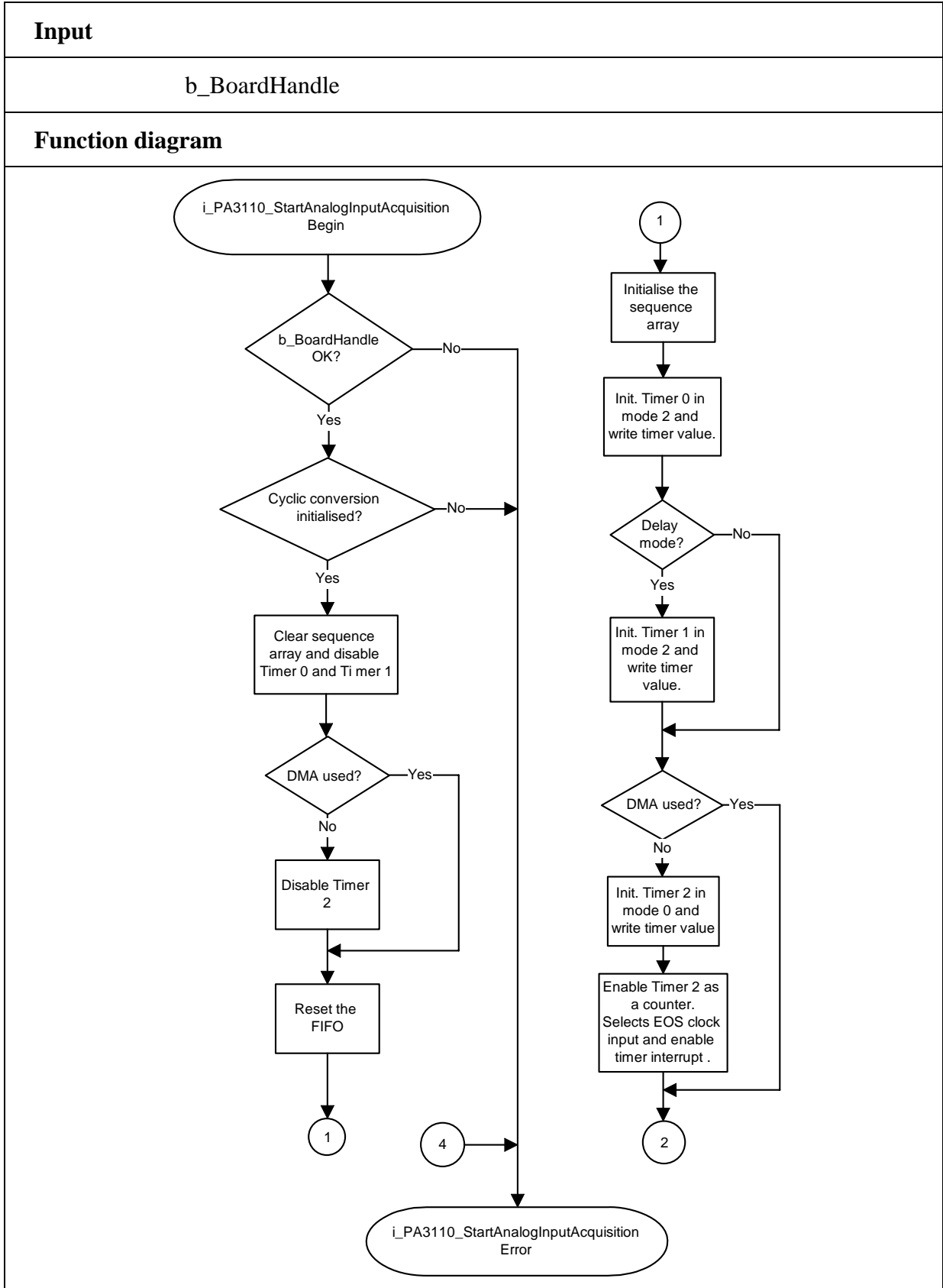
**Return value:**

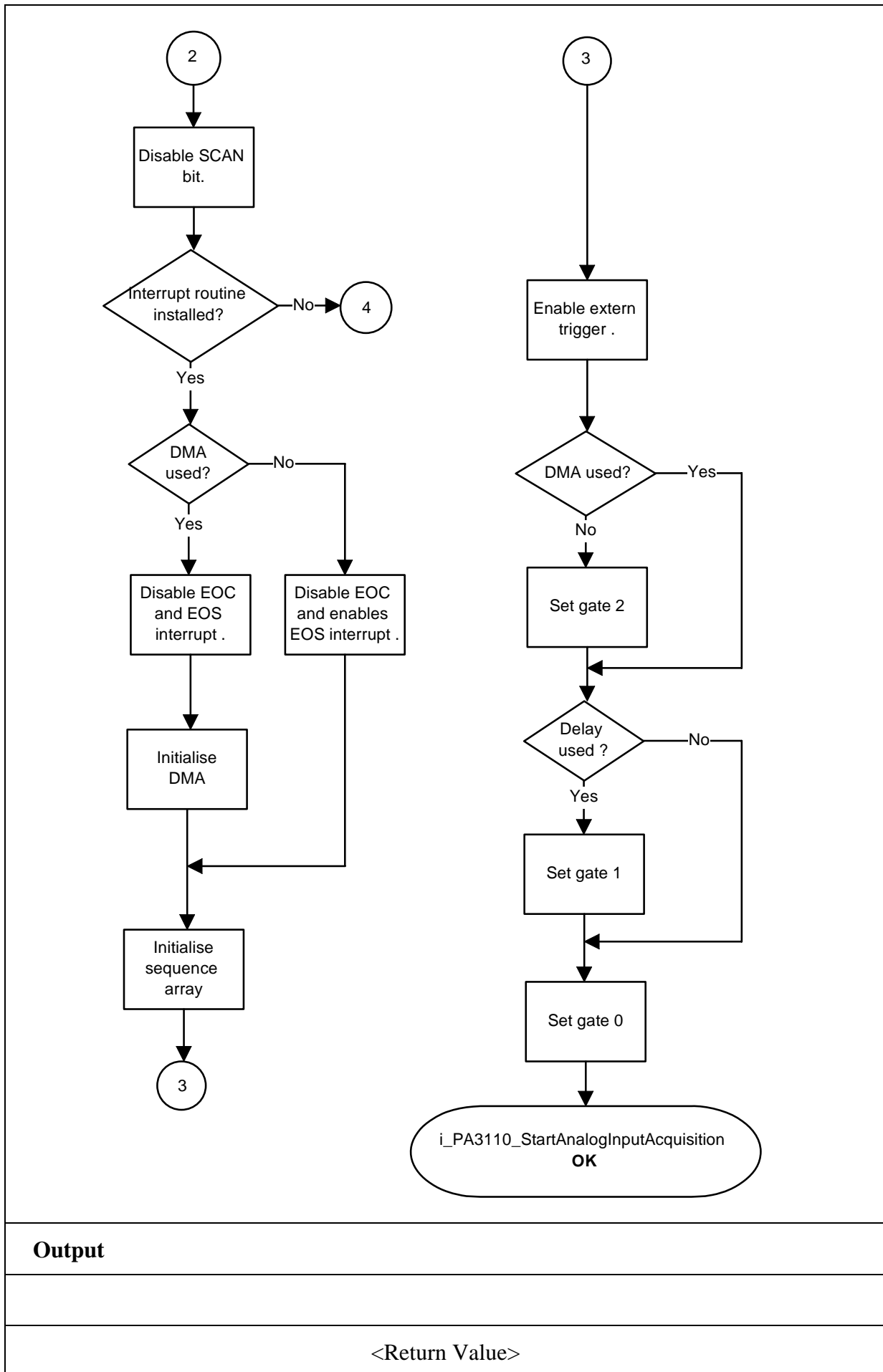
0: No error

-1: The handle parameter of the board is wrong

-2: The cyclic conversion has not been initialised.

"i\_PA3110\_InitAnalogInputAcquisition"





**Output**

<Return Value>

### 3) i\_PA3110\_StopAnalogInputAcquisition (...)

**Syntax:**

<Return value> = i\_PA3110\_StopAnalogInputAcquisition  
(BYTE b\_BoardHandle)

**Parameters:****- Input:**

BYTE b\_BoardHandle Handle of board **PA 3110**

**- Output:**

No output signal has occurred.

**Task:**

Stops the cyclic conversion. It has been previously started with function "i\_PA3110\_StartAnalogInputAcquisition".

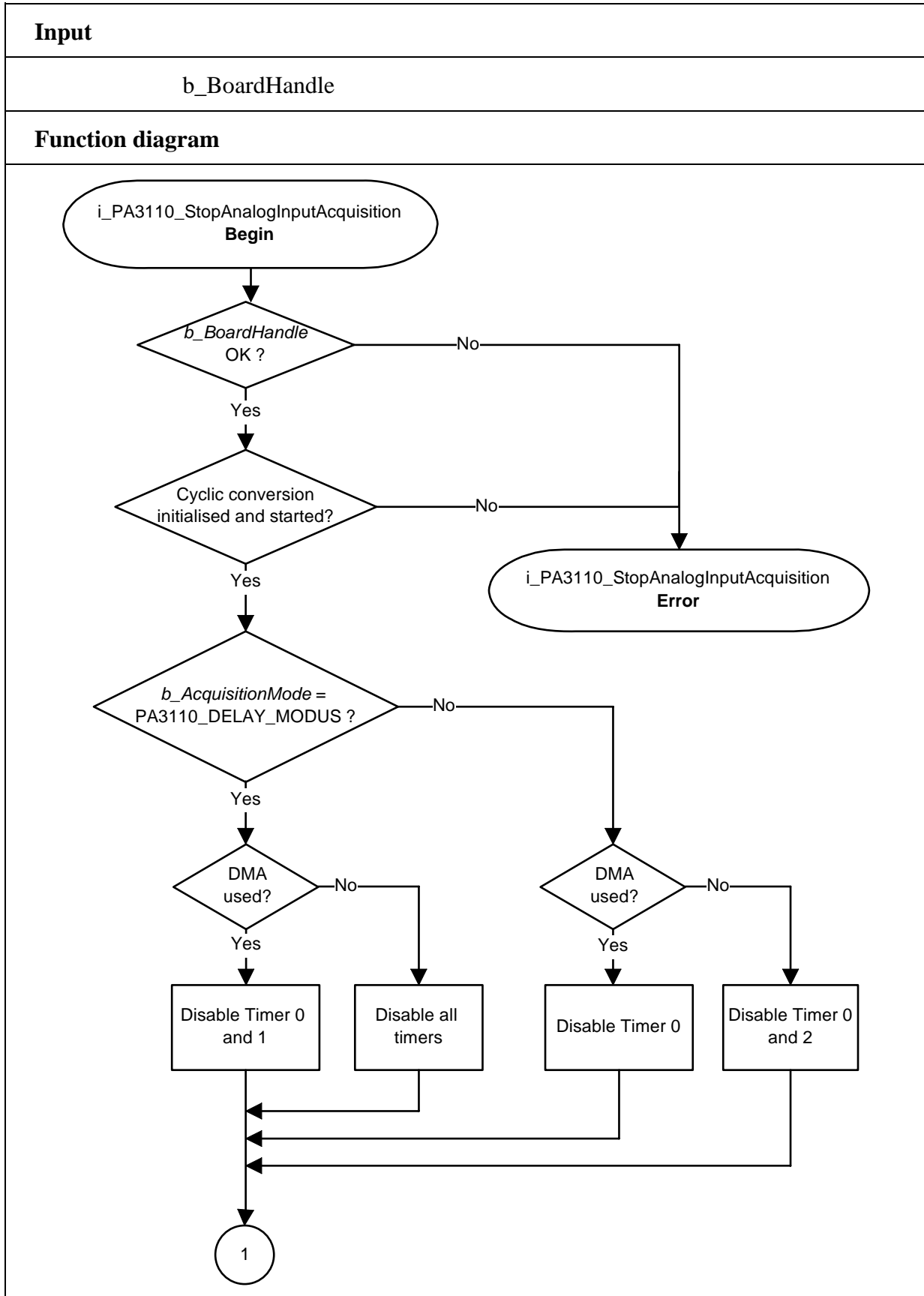
**Calling convention:**ANSI C:

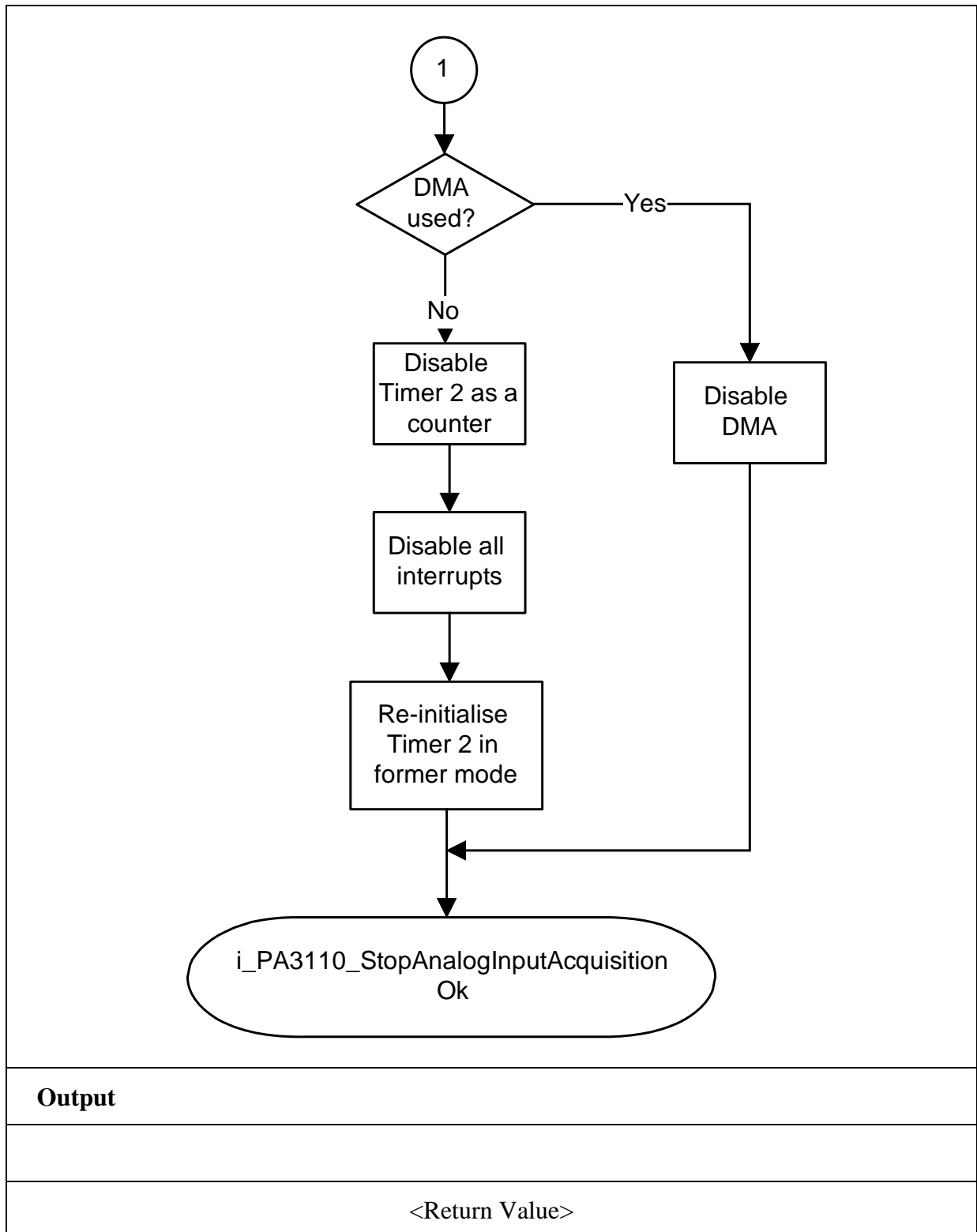
```
int          i_ReturnValue;  
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_PA3110_StopAnalogInputAcquisition (b_BoardHandle);
```

**Return value:**

- 0: No error
- 1: The handle parameter of the board is wrong
- 2: The cyclic conversion has not been started.  
"i\_PA3110\_StartAnalogInputAcquisition"





**Output**

<Return Value>

#### 4) i\_PA3110\_ClearAnalogInputAcquisition (...)

**Syntax:**

<Return value> = i\_PA3110\_ClearAnalogInputAcquisition  
 (BYTE b\_BoardHandle)

**Parameters:**

**- Input:**

BYTE b\_BoardHandle Handle of board PA 3110

**- Output:**

No output signal has occurred.

**Task:**

Deinstalls the DMA buffer.

**Calling convention:**

ANSI C:

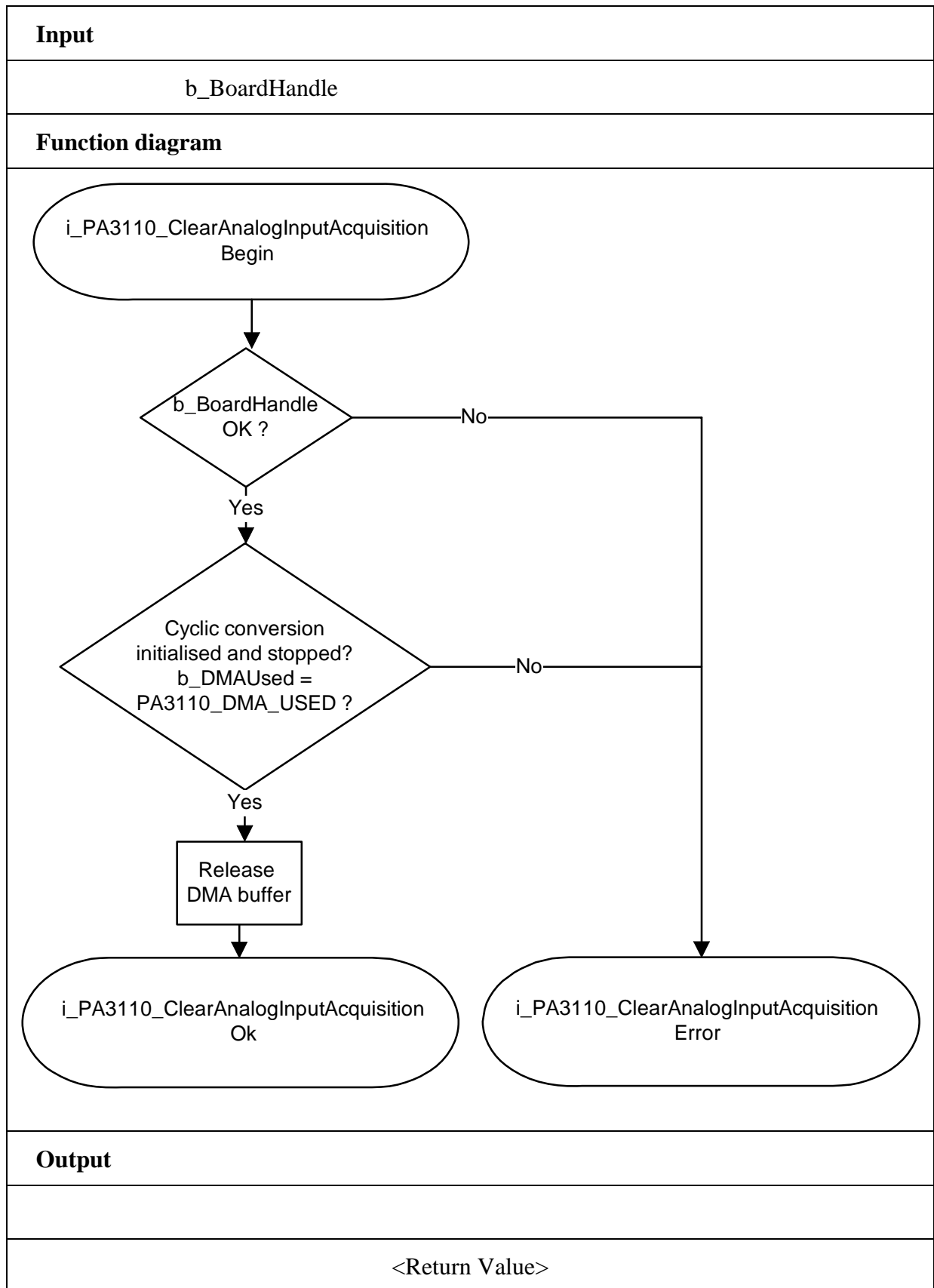
```
int i_ReturnValue;
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_PA3110_StartAnalogInputAcquisition (b_BoardHandle);
```

**Return value:**

- 0: No error.
- 1: The handle parameter of the board is wrong.
- 2: The cyclic conversion has not been initialised.  
 "i\_PA3110\_InitAnalogInputAcquisition"





## 3.5 Analog output channels

### 1) i\_PA3110\_Write1AnalogValue (...)

**Syntax:**

```
<Return value> = i_PA3110_Write1AnalogValue
                    (BYTE b_BoardHandle,
                     BYTE   b_ChannelNbr,
                     UINT   ui_ValueToWrite)
```

**Parameters:**

**- Input:**

BYTE	b_BoardHandle	Handle of board <b>PA 3110</b>
BYTE	b_ChannelNbr	Number of the analog output channel
UINT	ui_ValueToWrite	Analog output value to write (0 to 4095)

**- Output:**

No output signal has occurred.

**Task:**

Writes an analog value (*ui\_ValueToWrite*) on the analog output channel *b\_ChannelNbr*.

**Calling convention:**

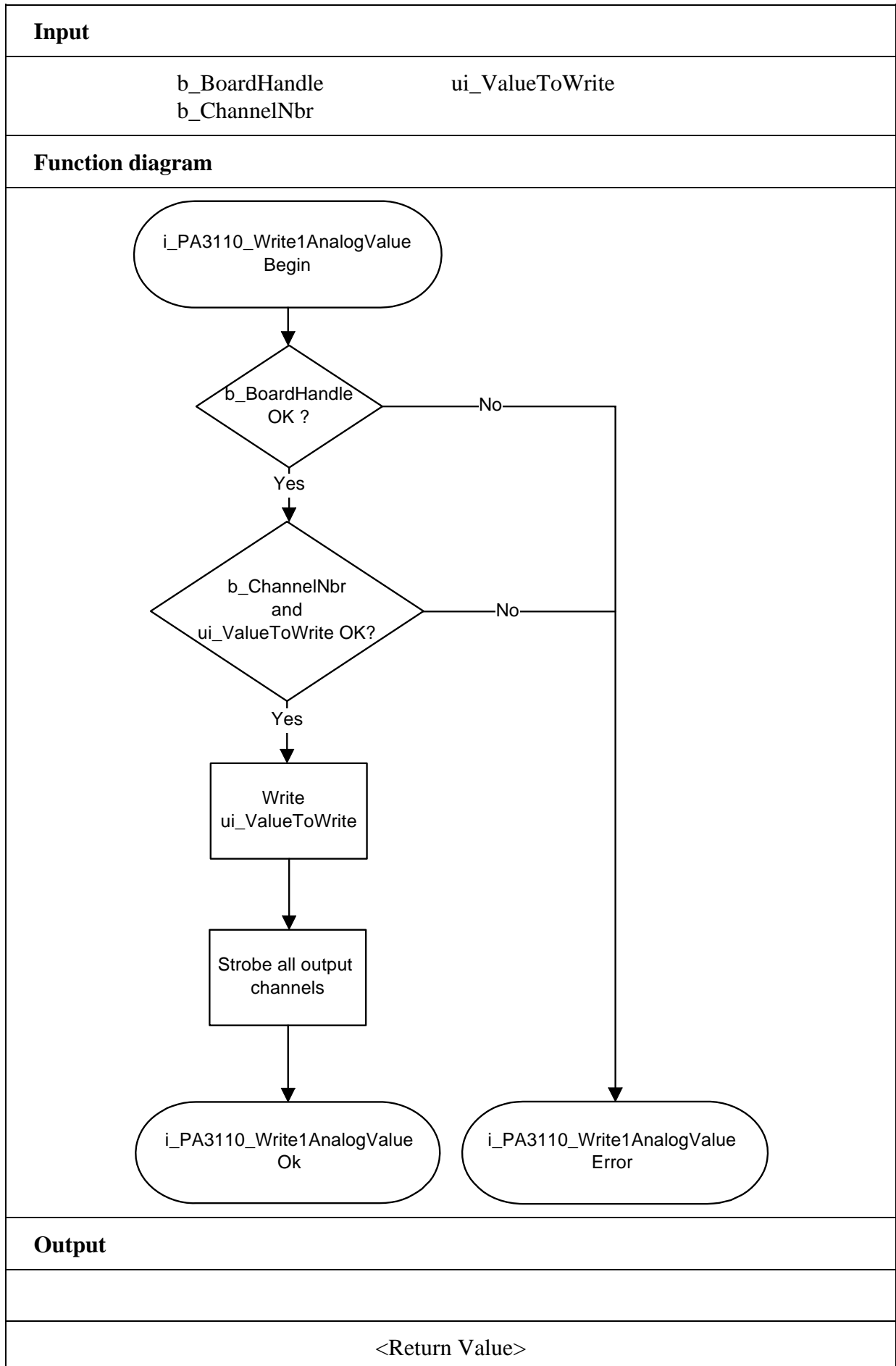
ANSI C:

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

```
i_ReturnValue = i_PA3110_Write1AnalogValue (b_BoardHandle,
                                             1,
                                             4095);
```

**Return value:**

0: No error  
 -1: The handle parameter of the board is wrong  
 -2: Number of the analog output channel is wrong.  
 -3: Output value too high



## 2) i\_PA3110\_WriteMoreAnalogValue (...)

### Syntax:

```
<Return value> = i_PA3110_WriteMoreAnalogValue
                    (BYTE b_BoardHandle,
                     BYTE   b_FirstChannelNbr,
                     BYTE   b_NbrOfChannel,
                     PUINT  pui_ValueArray)
```

### Parameters:

#### - Input:

BYTE	b_BoardHandle	Handle of board <b>PA 3110</b>
BYTE	b_FirstChannelNbr	Number of the first analog output channel (0 to 7)
BYTE	b_NbrOfChannel	Number of analog output channels you wish to write on (1 to 8)
PUINT	pui_ValueArray	Table of analog output values on which you want to write

#### - Output:

No output signal has occurred.

### Task:

Writes several analog values on several analog output channels.

The variable *b\_FirstChannelNbr* defines the first analog output channel.

The variable *b\_NbrOfChannel* defines the number of analog output channels .

### Example:

#### Parameter

```
b_FirstChannelNbr = 2
b_NbrOfChannel   = 3

pui_ValueArray [0] = 0
pui_ValueArray [1] = 2047
pui_ValueArray [2] = 4095
```

The value 0 (0V) is written in the buffer of analog output 2  
 The value 2047 (5V) is written in the buffer of analog output 3  
 The value 4095 (10V) is written in the buffer of analog output 4.

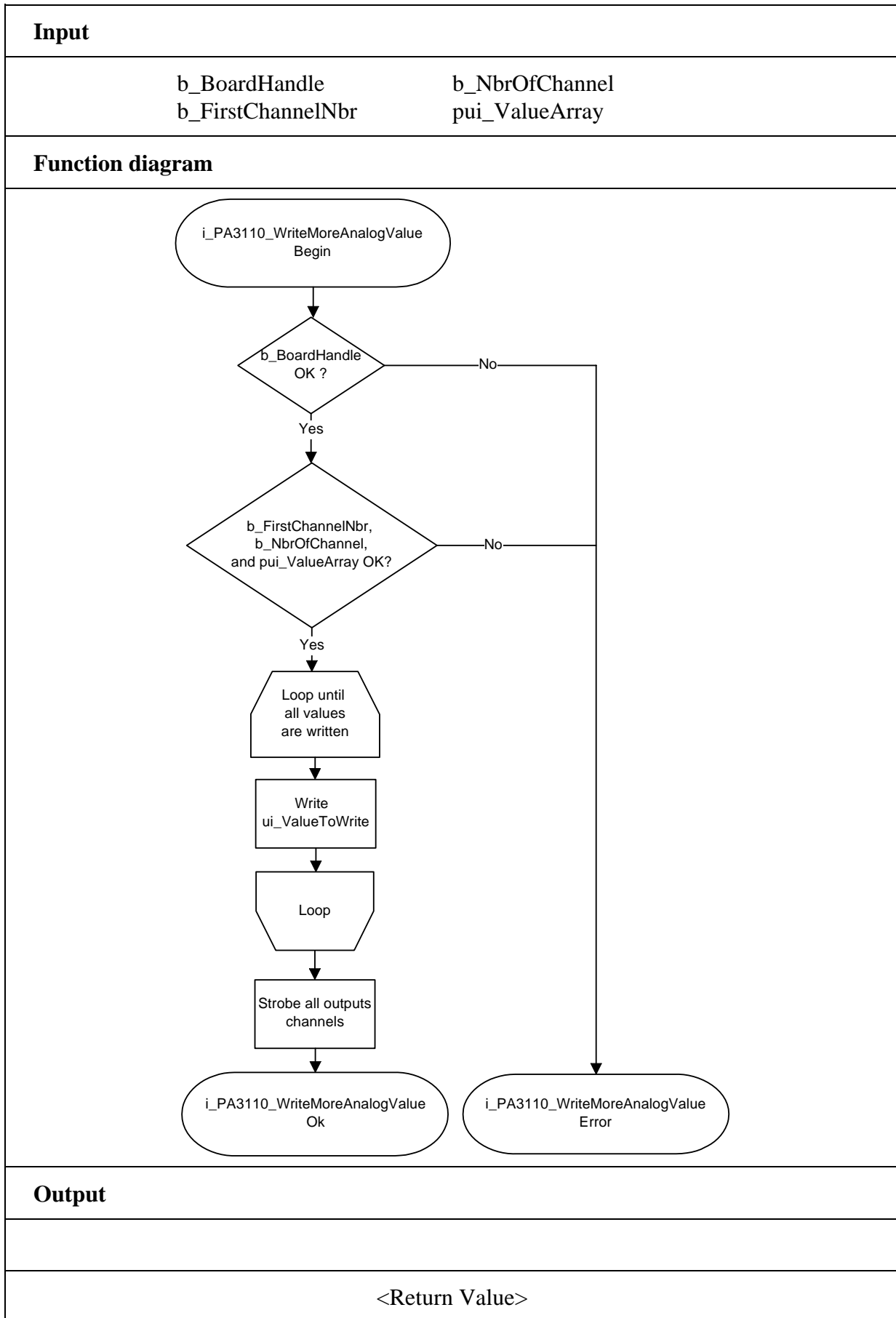
**Calling convention:**ANSI C :

```
int          i_ReturnValue;
unsigned char b_BoardHandle;
unsigned char b_PolarityArray [8];
unsigned int  ui_ValueArray [8];
```

```
i_ReturnValue = i_PA3110_WriteMoreAnalogValue
                                                    (b_BoardHandle,
                                                    0,
                                                    8,
                                                    b_PolarityArray,
                                                    ui_ValueArray);
```

**Return value:**

- 0: No error
- 1: The handle parameter of the board is wrong
- 2: Number of the analog output channel is wrong
- 3: The number of analog output channels you wish to write on is wrong  
See function "i\_PA3110\_SetBoardInformation"
- 4: One or several output value are too high.



## 3.6 Timer

### 1) i\_PA3110\_InitTimerWatchdog (...)

#### Syntax:

```
<Return value> = i_PA3110_InitTimerWatchdog
                                     (BYTE b_BoardHandle,
                                     BYTE b_TimerMode,
                                     LONG l_DelayValue,
                                     BYTE b_InterruptFlag)
```

#### Parameters:

##### - Input:

BYTE	b_BoardHandle	Handle of board <b>PA 3110</b>
BYTE	b_TimerMode	Defines the mode of the timer - PA3110_TIMER: The timer/watchdog is used as an edge generator. - PA3110_WATCHDOG: The timer/watchdog is used as a watchdog for the analog output channels.
LONG	l_DelayValue	Time interval (watchdog time) of the timer from 70 $\mu$ s up to 4587400 $\mu$ s
BYTE	b_InterruptFlag	PA3110_ENABLE: <u>Timer</u> : an interrupt is generated at the end of each time interval <u>Watchdog</u> : an interrupt is generated when the watchdog has run down PA3110_DISABLE: No interrupt is generated.

##### - Output:

No output signal has occurred

#### Task:

Initialises the timer as an edge generator or as a watchdog for the analog outputs.

#### Calling convention:

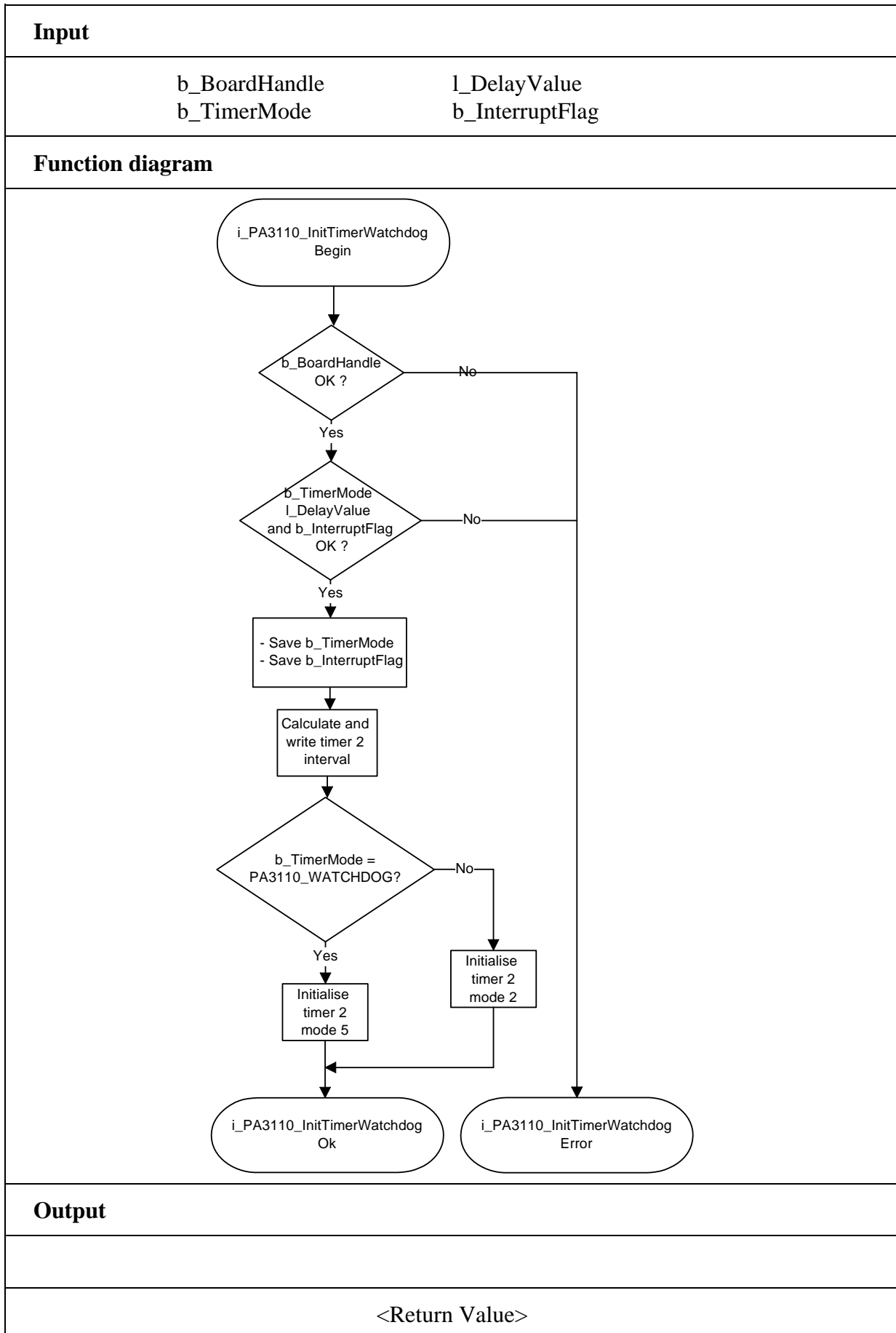
##### ANSI C :

```
int          i_ReturnValue;
unsigned char b_BoardHandle;

i_ReturnValue = i_PA3110_InitTimerWatchdog (b_BoardHandle,
                                             PA3110_UNIPOLAR,
                                             1000,
                                             PA3110_DISABLE);
```

#### Return value:

- 0: No error.
- 1: The handle parameter of the board is wrong.
- 2: The mode parametered for the timer is wrong.
- 3: The user interrupt routine has not been installed  
See function "i\_PA3110\_SetBoardIntRoutineXXX"
- 4: The interrupt parameter is wrong
- 5: Time selection is wrong





## 2) i\_PA3110\_StartTimerWatchdog (...)

**Syntax:**

<Return value> = i\_PA3110\_StartTimerWatchdog (BYTE b\_BoardHandle)

**Parameters:****- Input:**

BYTE b\_BoardHandle                      Handle of board **PA 3110**

**- Output:**

No output signal has occurred.

**Task:**

Starts the timer/watchdog.

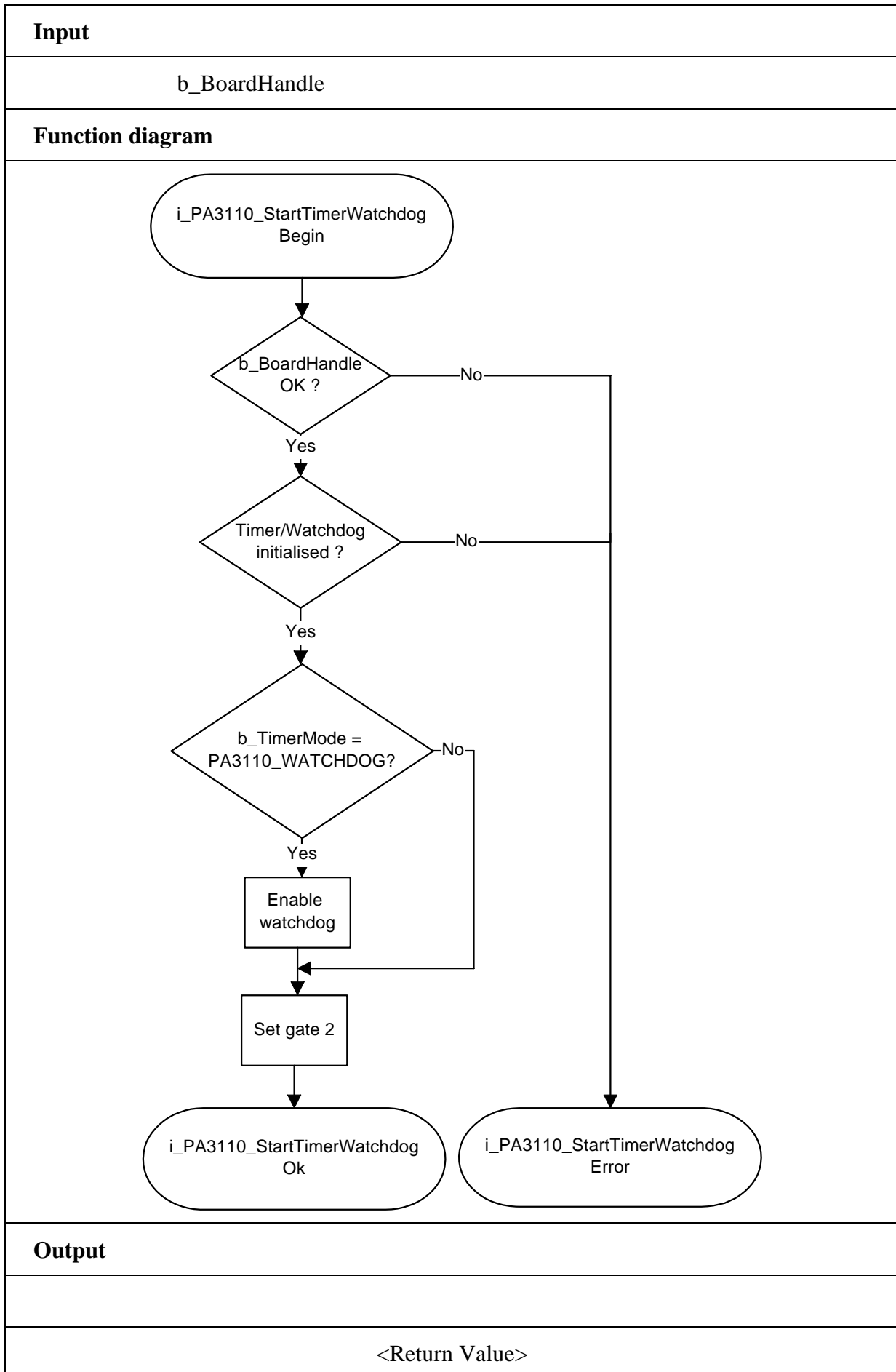
**Calling convention:**ANSI C:

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

```
i_ReturnValue = i_PA3110_StartTimerWatchdog (b_BoardHandle);
```

**Return Value:**

- 0: No error.
- 1: The handle parameter of the board is wrong.
- 2: Timer/watchdog has not been initialised.



### 3) i\_PA3110\_StopTimerWatchdog (...)

**Syntax:**

<Return value> = i\_PA3110\_StopTimerWatchdog  
(BYTE b\_BoardHandle)

**Parameters:****- Input:**

BYTE b\_BoardHandle Handle of the board **PA 3110**

**- Output:**

No output signal has occurred.

**Task:**

Stops the timer/watchdog.

**Calling convention:**ANSI C:

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

```
i_ReturnValue = i_PA3110_StopTimerWatchdog (b_BoardHandle);
```

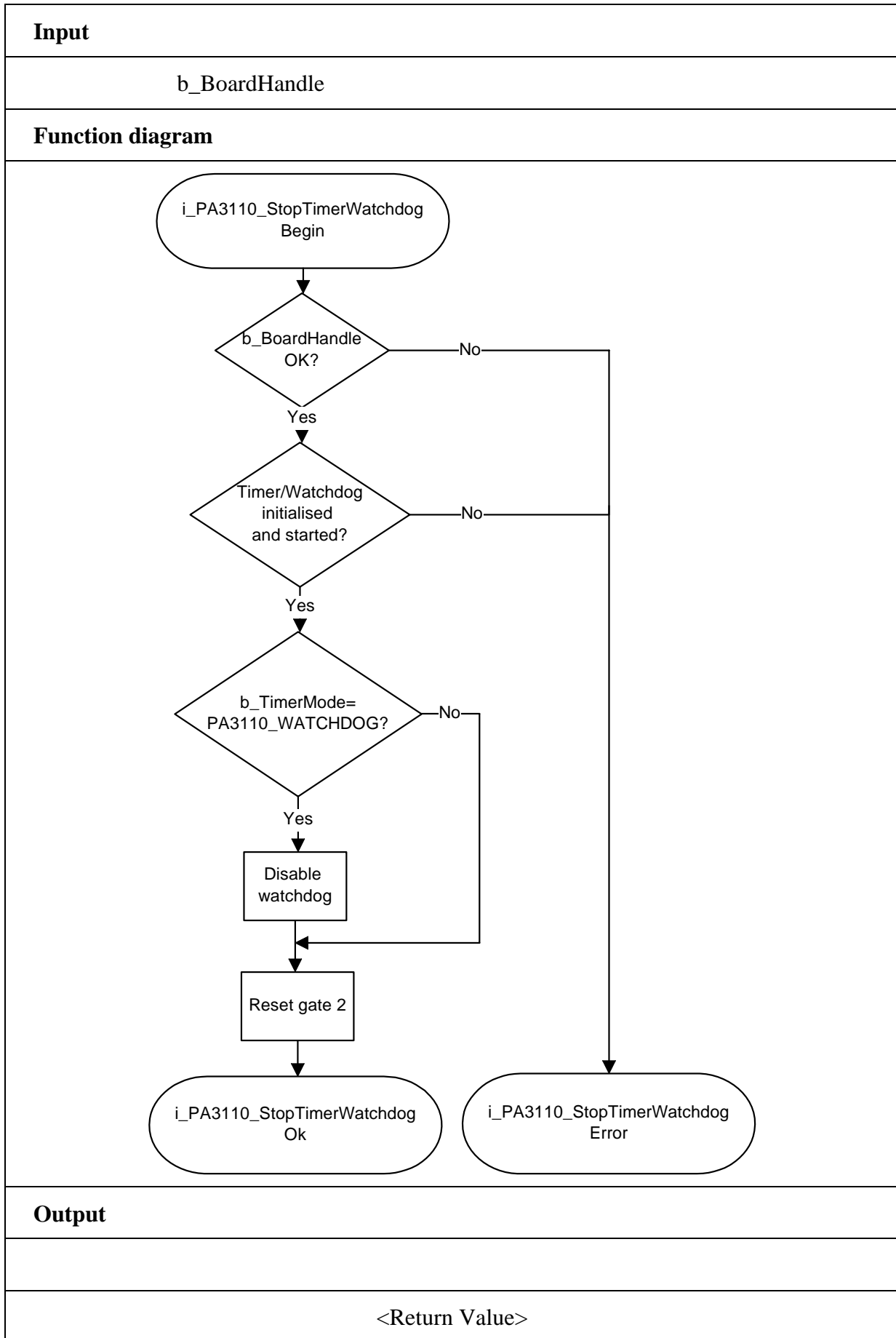
**Return Value:**

0: No error.

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

-2: Timer/watchdog has not been initialised.

-3: Timer/watchdog has not been started.



#### 4) i\_PA3110\_ReadTimer (...)

**Syntax:**

<Return value> = i\_PA3110\_ReadTimer  
(BYTE b\_BoardHandle  
LONG pl\_ReadValue)

**Parameters:****- Input:**

BYTE b\_BoardHandle Handle of the board **PA 3110**

**- Output:**

PLONG pl\_ReadValue Current timer value  
(from 0 to FFFFFFF Hex)

**Task:**

Reads the current value of the timer.

**Calling convention:**ANSI C:

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

```
i_ReturnValue = i_PA3110_ReadTimer (b_BoardHandle,  
                                     &l_ReadValue);
```

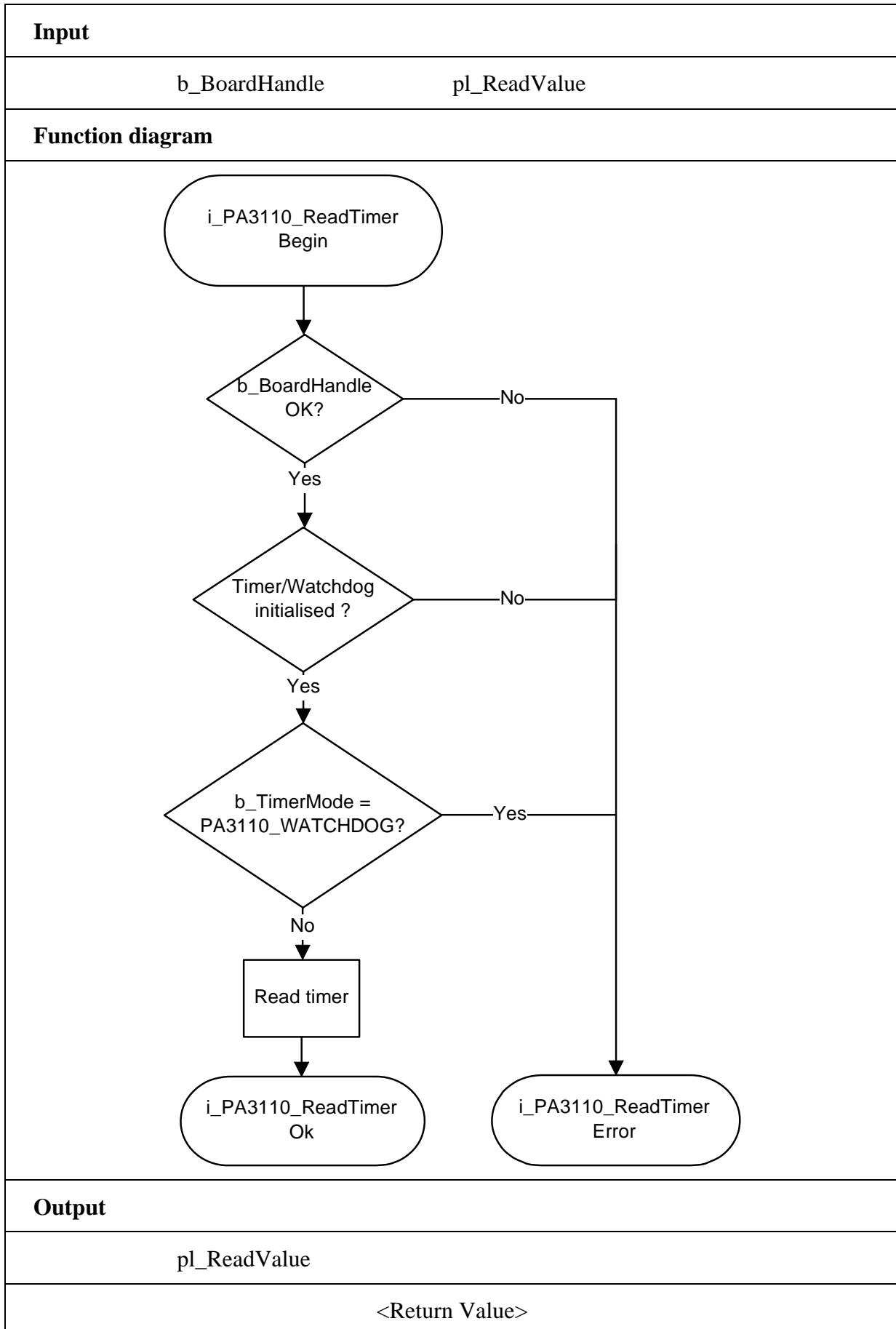
**Return value:**

0: No error.

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

-2: Timer/watchdog has not been initialised.

-3: Timer/watchdog has been initialised as a watchdog.



### 5) i\_PA3110\_WriteTimer (...)

**Syntax:**

<Return value> = i\_PA3110\_WriteTimer (BYTE b\_BoardHandle  
LONG l\_WriteValue)

**Parameters:****- Input:**

BYTE	b_BoardHandle	Handle of the board <b>PA 3110</b>
LONG	l_WriteValue	New timer value (from 0 to FFFFFFF Hex)

**- Output:**

No output signal has occurred.

**Task:**

Writes a new value in the timer.

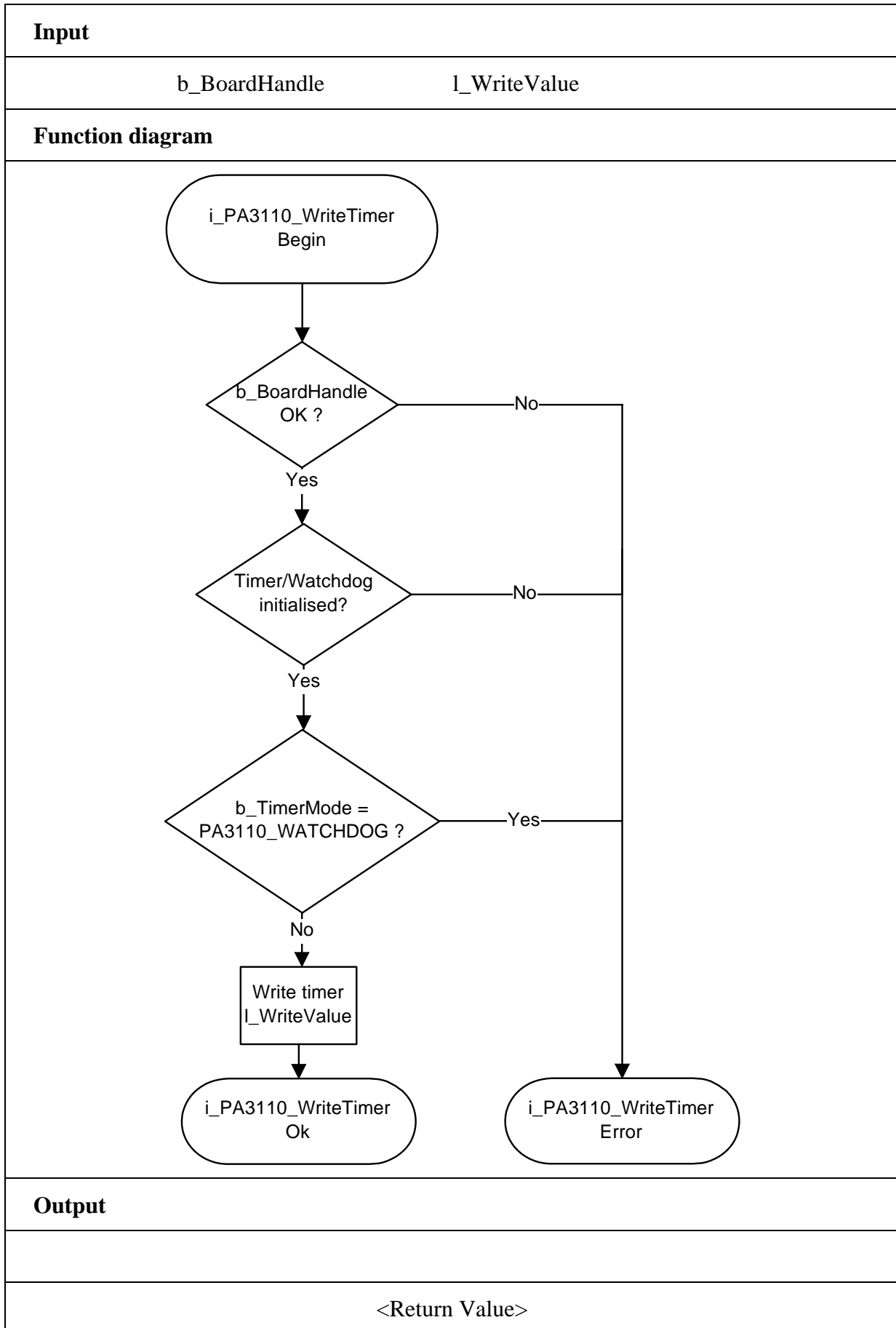
**Calling convention:**ANSI C:

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

```
i_ReturnValue = i_PA3110_WriteTimer (b_BoardHandle,  
1000);
```

**Return value:**

- 0: No error.
- 1: The handle parameter of the board is wrong.
- 2: Timer/watchdog has not been initialised.
- 3: Timer/watchdog has been initialised as a watchdog.





**6) i\_PA3110\_ReadWatchdogStatus (...)****Syntax:**

<Return Value> = i\_PA3110\_ReadWatchdogStatus  
(BYTE b\_BoardHandle  
PBYTE pb\_WatchdogStatus)

**Parameters:****- Input:**

BYTE b\_BoardHandle Handle of board **PA 3110**

**- Output:**

PBYTE pb\_WatchdogStatus 0: Watchdog has not run down  
1: Watchdog has run down

**Task:**

Writes a new value in the timer.

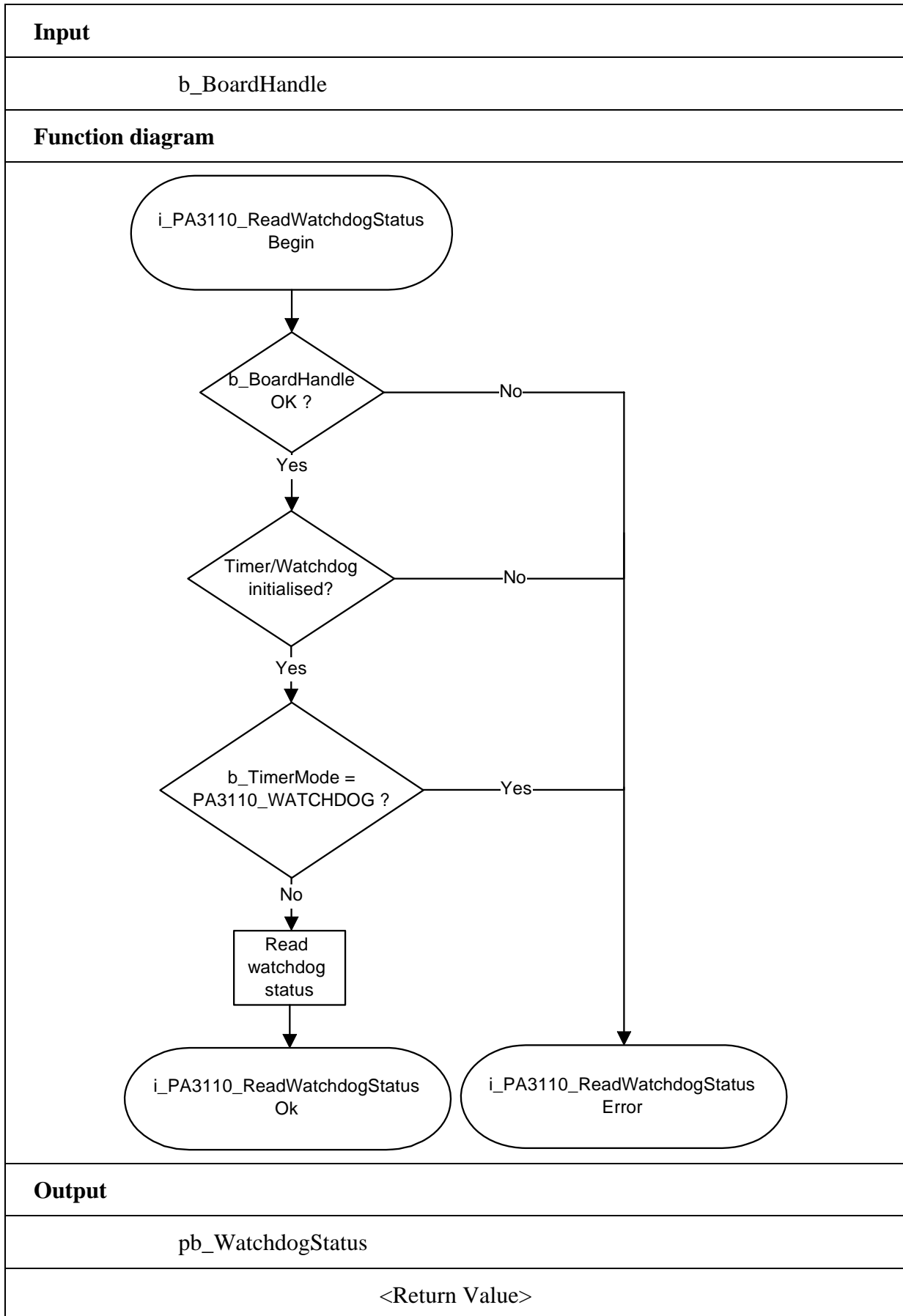
**Calling convention:**ANSI C:

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

```
i_ReturnValue = i_PA3110_ReadWatchdogStatus (b_BoardHandle,  
                                              &b_WatchdogStatus);
```

**Return Value:**

- 0: No error.
- 1: The handle parameter of the board is wrong.
- 2: Timer/watchdog has not been initialised.
- 3: Timer/watchdog has been initialised as a timer.



## 3.7 Functions to be used in Kernel mode

### 1) `i_PA3110_KRNL_Write1AnalogValue (...)`

**Syntax:**

```
<Return value> = i_PA3110_KRNL_Write1AnalogValue
                    (UINT      ui_Address,
                     BYTE      b_ChannelNbr,
                     UINT      ui_ValueToWrite)
```

**Parameters:**

**- Input:**

UINT	ui_Address	Address of the board <b>PA 3110</b>
BYTE	b_ChannelNbr	Number of the analog output channel (0 to 7)
UINT	ui_ValueToWrite	Analog output value to write (0 to 4095)

**- Output:**

No output signal has occurred.

**Task:**

Writes an analog value (*ui\_ValueToWrite*) on the analog output channel *b\_ChannelNbr*.

**Calling convention:**

ANSI C:

```
int          i_ReturnValue;
```

```
i_ReturnValue = i_PA3110_KRNL_Write1AnalogValue
                (0x390,
                 1,
                 4095);
```

**Return value:**

0: No error  
 -1: Channel number is wrong  
 -2: Output value too high

