ADDI-DATA®

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

Technical description

ADDINUM PA 1000

Standard software/ API functions

Edition: 06.01-07/2005

1 INTRODUCTION 1

2 DIN 66001- GRAPHICAL SYMBOLS 4

3 SOFTWARE FUNCTIONS (API) 5

3.1 Initialisation 5
- 1) i_PA1000_InitCompiler (..) 5
- 2) i_PA1000_SetBoardInformation (..) 7
- 3) i_PA1000_SetBoardInformationWin32 (...) 9
- 4) i_PA1000_GetHardwareInformation (...) 11
- 5) i_PA1000_CloseBoardHandle 13

3.2 Interrupt 15
- 1) i_PA1000_SetBoardIntRoutineDos (..) 15
- 2) i_PA1000_SetBoardIntRoutineVBDos (..) 18
- 3) i_PA1000_SetBoardIntRoutineWin16 21
- 4) i_PA1000_SetBoardIntRoutineWin32 (..) 24
- 5) i_PA1000_TestInterrupt (..) 30
- 6) i_PA1000_ResetBoardIntRoutine (..) 32

3.3 Digital input channels 34
- 1) i_PA1000_Read1DigitalInput (...) 34
- 2) i_PA1000_Read8DigitalInput (...) 36
- 3) i_PA1000_Read16DigitalInput (...) 38
- 4) i_PA1000_Read32DigitalInput (...) 40

3.4 Digital input channel - events 42
- 1) i_PA1000_SetInputEventMask (...) 42
- 2) i_PA1000_StartInputEvent (...) 46
- 3) i_PA1000_StopInputEvent (...) 48

3.5 Timer/counter 50
- 1) i_PA1000_InitTimerCounter1 (...) 50
- 2) i_PA1000_InitTimerCounter2 (...) 54
- 3) i_PA1000_InitTimerCounter3 (...) 58
- 4) i_PA1000_StartTimerCounter1(...) 62
- 5) i_PA1000_StartTimerCounter2 (...) 64
- 6) i_PA1000_StartTimerCounter3 (...) 66
- 7) i_PA1000_StopTimerCounter1 (...) 68
- 8) i_PA1000_StopTimerCounter2 (...) 70
- 9) i_PA1000_StopTimerCounter3 (...) 72
- 10) i_PA1000_TriggerTimerCounter1 (...) 74
- 11) i_PA1000_TriggerTimerCounter2 (...) 76
- 12) i_PA1000_TriggerTimerCounter3 (...) 78
- 13) i_PA1000_ReadTimerCounter1 (...) 80
- 14) i_PA1000_ReadTimerCounter2 (...) 82
- 15) i_PA1000_ReadTimerCounter3 (...) 84

3.6 Compatibility functions for the former driver version (Windows NT/95) 86
- 1) i_PA1000_SetBoardAddress (...) 86
- 2) i_PA1000_SetBoardIntRoutine (..) 88

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 9-3: Interrupt mask 16

1 INTRODUCTION

IMPORTANT!
Note the following conventions in the text:

Function: "i_PA1000_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

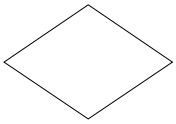
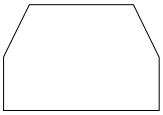
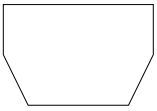
Define name	Decimal value	Hexadecimal value
DLL_COMPILER_C	0	0
DLL_COMPILER_VB	1	1
DLL_COMPILER_PASCAL	2	2
DLL_LABVIEW	3	3
DLL_COMPILER_VB_5	4	4
PA1000_DISABLE	0	0
PA1000_ENABLE	1	1
PA1000_AND	2	2
PA1000_OR	4	4
PA1000_OR_PRIORITY	6	6
PA1000_8BIT	8	8
PA1000_16BIT	16	10
PA1000_TIMER	0	0
PA1000_COUNTER	32	20
PA1000_SINGLE	0	0
PA1000_CONTINUOUS	128	80
PA1000_SOFTWARE_TRIGGER	4	4
PA1000_HARDWARE_TRIGGER	16	10
PA1000_SOFTWARE_GATE	0	0
PA1000_HARDWARE_GATE	8	8

2 DIN 66001- GRAPHICAL SYMBOLS

This chapter describes all software functions (API) necessary for the operation of the **PA 1000** board.

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

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

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

3 SOFTWARE FUNCTIONS (API)

3.1 Initialisation

1) i_PA1000_InitCompiler (..)

Syntax:
<Return value> = i_PA1000_InitCompiler (BYTE b_CompilerDefine)

Parameters:
- Input:

BYTE	b_CompilerDefine	The user has to choose the language under Windows in which he/she wants to program - DLL_COMPILER_C: The user programs in C. - DLL_COMPILER_VB: The user programs in Visual Basic for Windows. - DLL_COMPILER_VB_5: The user programs in Visual Basic 5.0. - DLL_COMPILER_PASCAL: The user programs in Pascal or Delphi. - DLL_LABVIEW : The user programs in Labview.

- Output:
No output signal has occurred.

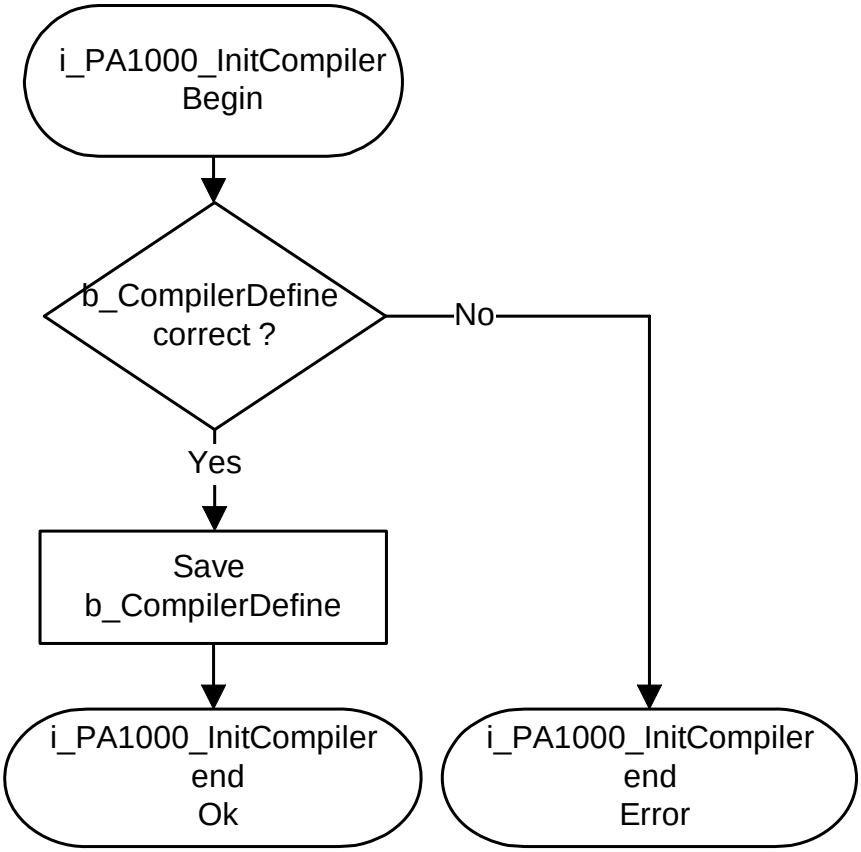
Task:
If you want to use the DLL functions, parameter in the language in which you want to program. This function must be the first to be called up.

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

Calling convention:
ANSI C :

```
int i_ReturnValue;

i_ReturnValue = i_PA1000_InitCompiler (DLL_COMPILER_C);
```

Return value:
0: No error
-1: Compiler parameter is wrong

Input
b_CompilerDefine
Function diagram

Output
<Return value>

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

2) i_PA1000_SetBoardInformation (..)

Syntax:

```
<Return value> =  i_PA1000_SetBoardInformation     (UINT    ui_BaseAddress,
                                                    BYTE     b_AccessMode,
                                                    BYTE     b_InterruptNbr,
                                                    PBYTE pb_BoardHandle)
```

Parameters:

- Input:

UINT	ui_BaseAddress	Base address of **PA1000** board
BYTE	b_AccessMode	Hardware access mode PA1000_8BIT: 8-bit access mode PA1000_16BIT: 16-bit access mode
BYTE	b_InterruptNbr	**PA 1000** interrupt number (3, 5, 10, 11, 12, 14 or 15). If 0, no interrupt is used

- Output:

PBYTE	pb_BoardHandle	Handle[1] of board **PA1000** to use the functions

Task:

Verifies if the board **PA1000** is present. Stores the following information:
- the base address,
- the hardware access mode,
- the interrupt number.

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

Calling convention:

ANSI C :

```
int              i_ReturnValue;
unsigned char b_BoardHandle;

i_ReturnValue = i_PA1000_SetBoardInformation     (0x300,
                                                  PA1000_16BIT,
                                                  0,
                                                  &b_BoardHandle);
```

Return value:

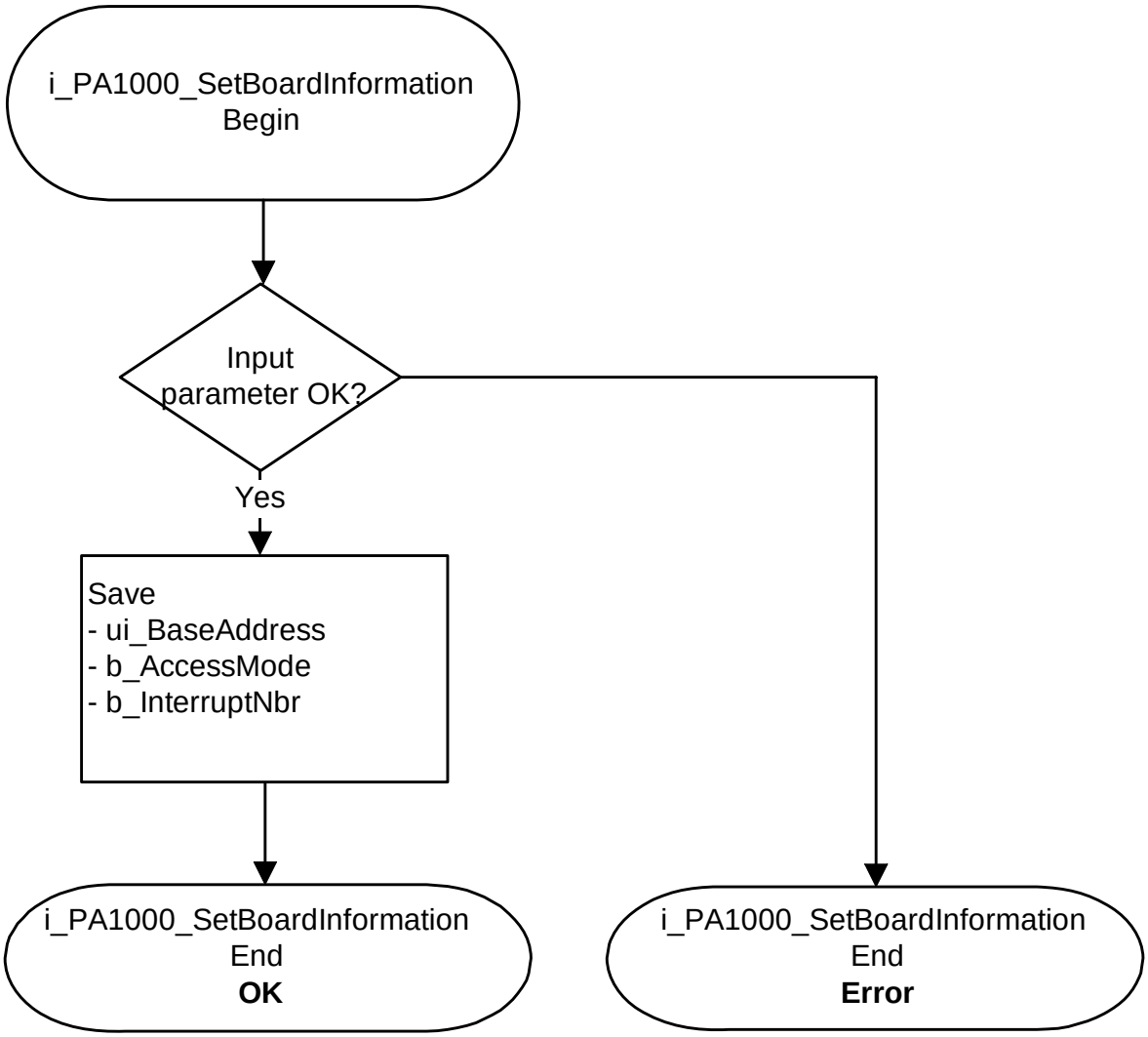
0: No error
-1: Board not present
-2: Access mode parameter is wrong
-3: Interrupt number is wrong or already used by another **PA1000**
-4: No handle is available for the board (up to 10 handles can be used)

[1] Identification number of the board

Input

ui_BaseAddress b_AccessMode
b_InterruptNbr

Function diagram



Output

pb_BoardHandle

<Return value>

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

3) i_PA1000_SetBoardInformationWin32 (...)

Syntax:
<Return value> = i_PA1000_SetBoardInformationWin32
(PCHAR pc_Identifier
BYTE b_AccessMode,
PBYTE pb_BoardHandle)

Parameters:
- Input:

PCHAR	pc_Identifier	Identifier string for the selection of **PA 1000.** The identifier string is determined by the ADDIREG registration program.
BYTE	b_AccessMode	Hardware access mode PA1000_8BIT: 8-bit access mode PA1000_16BIT: 16-bit access mode

- Output:

PBYTE	pb_BoardHandle	Handle of the board **PA 1000** to use the functions

Task:
Calls up all hardware information about the **PA 1000** given by the ADDIREG registration program and stores the following information:
- the base address,
- the interrupt number.
Then verifies if board **PA 1000** 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 :

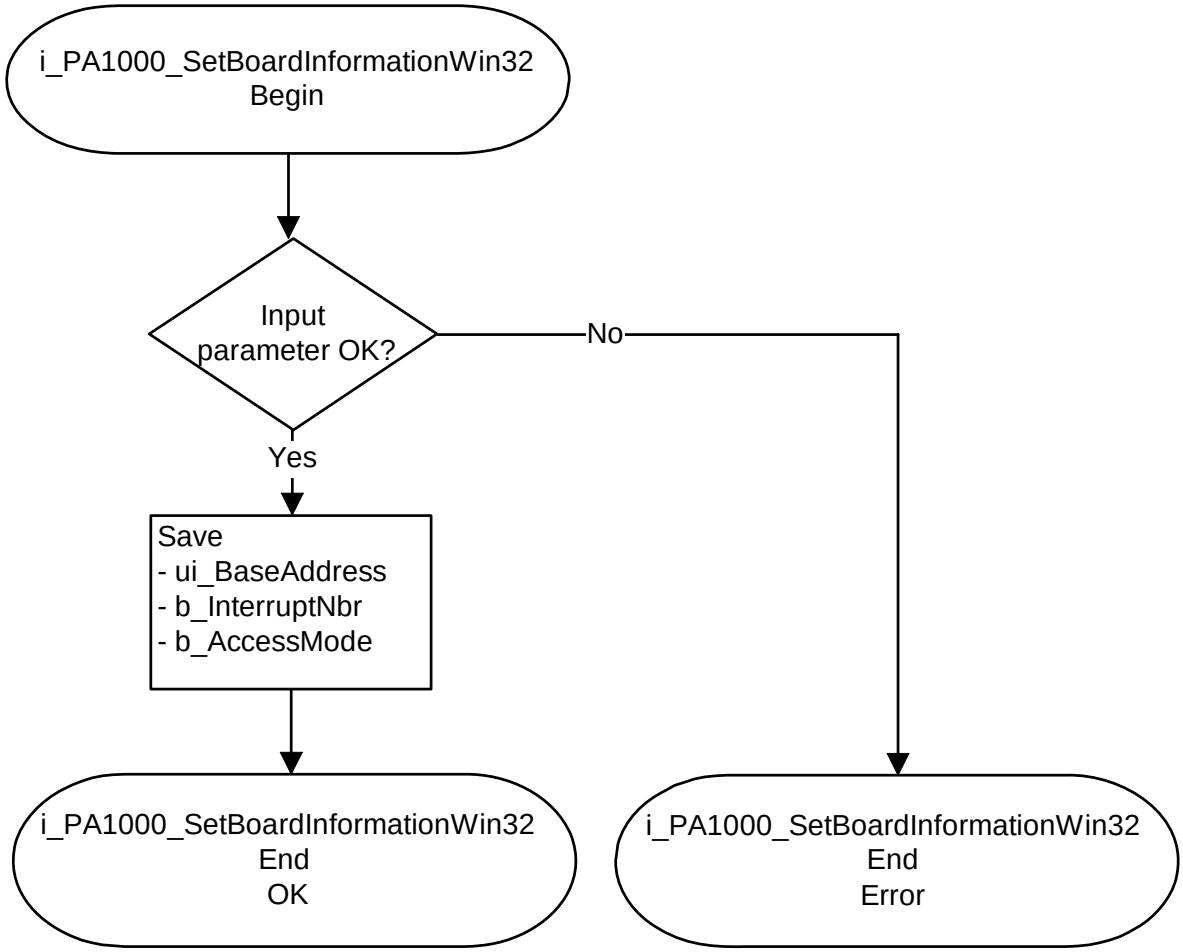
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1000_SetBoardInfomationWin32
                                    (“PA1000-00“, &b_BoardHandle);
```

Return value:
0: No error
-1: Board not present
-2: Access mode parameter is wrong
-3: No handle is available for the board (up to 10 handles can be used)
-4:Error by opening the driver under Windows NT/95

Input

pc_Identifier

Function diagram



Output

pb_BoardHandle

<Return value>

4) i_PA1000_GetHardwareInformation (...)

Syntax:

```
<Return value> =  i_PA1000_GetHardwareInformation
                                        (BYTE    b_BoardHandle,
                                         PUINT   pui_BaseAddress,
                                         PBYTE   pb_InterruptNbr)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1000**

- Output:

PUINT	pui_BaseAddress	Base address of the **PA 1000**
PBYTE	pb_InterruptNbr	Interrupt line of the **PA 1000**.

Task:

Returns the base address, the interrupt and DMA number of the **PA 1000**.

Calling convention:

ANSI C :

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

i_ReturnValue = i_PA1000_GetHardwareInformation (b_BoardHandle,
                                                 &ui_BaseAddress,
                                                 &b_InterruptNbr);
```

Return value:

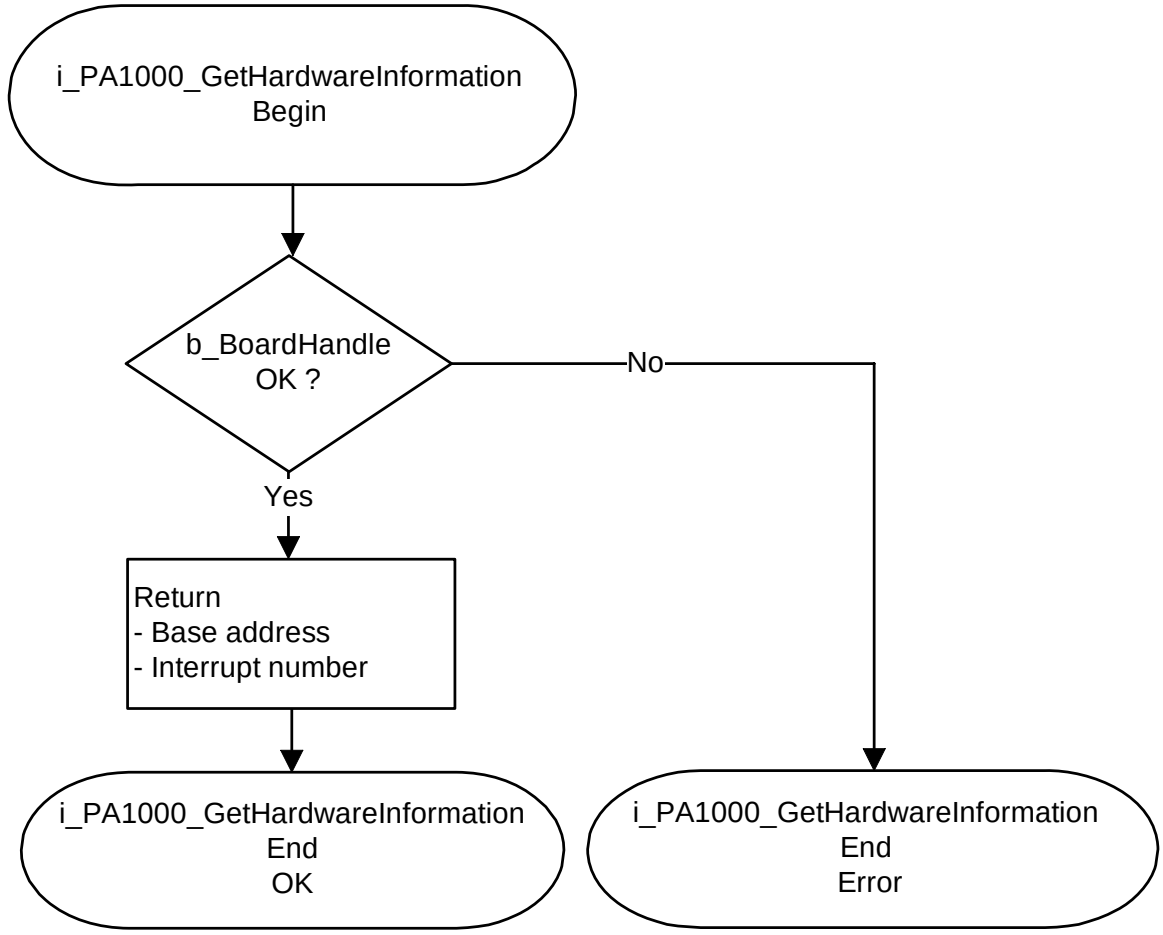
0: No error

-1: The handle parameter of the board is wrong

Input

b_BoardHandle

Function diagram



Output

pui_BaseAddress pb_InterruptNbr

<Return value>

5) i_PA1000_CloseBoardHandle

Syntax:
<Return value> = i_PA1000_CloseBoardHandle (BYTE b_BoardHandle)

Parameters:
- Input:
BYTE b_BoardHandle Handle of board **PA 1000**
- Output:
No output signal has occurred.

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

Calling convention:
ANSI C :

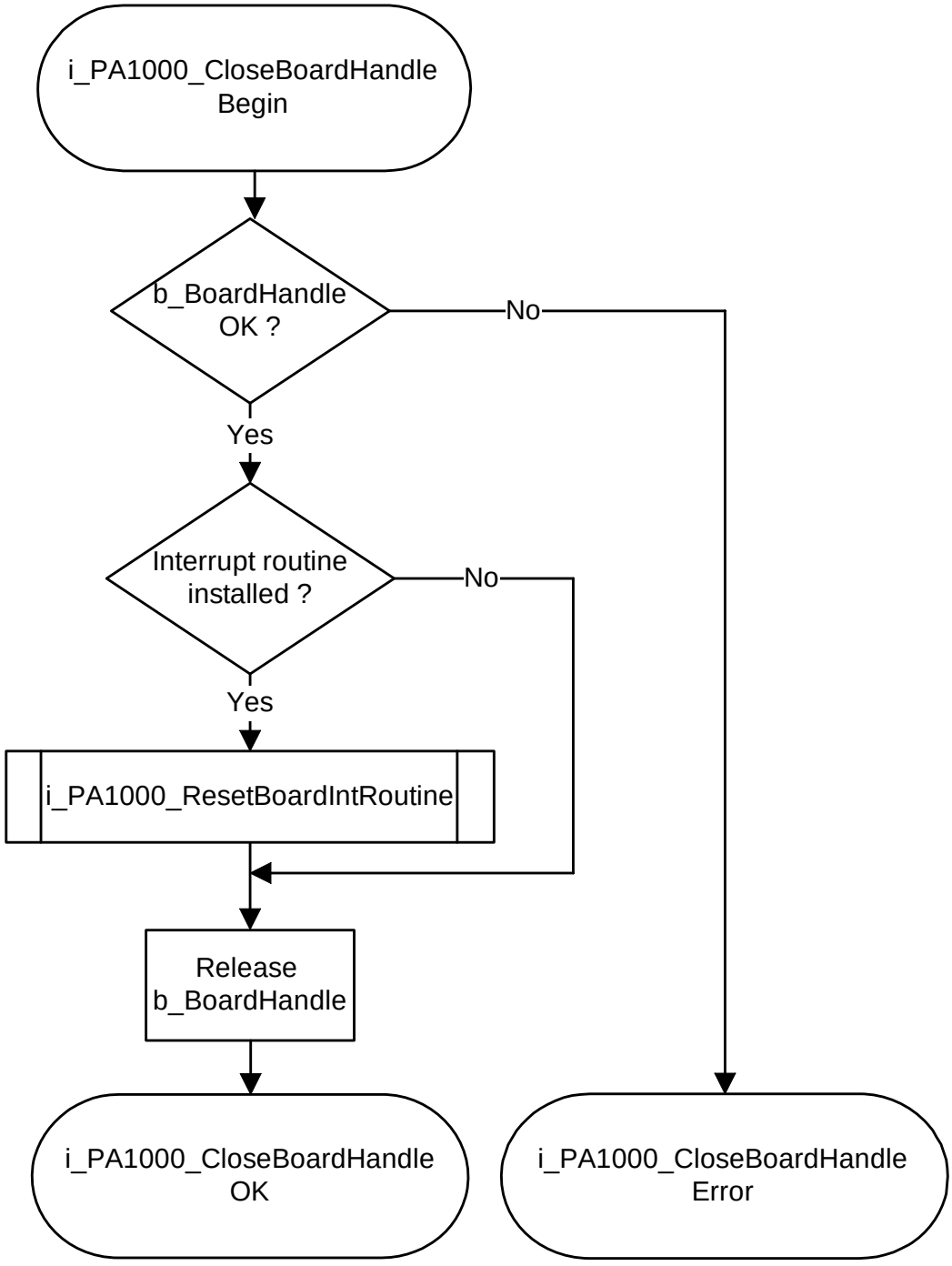
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

  i_ReturnValue = i_PA1000_CloseBoardHandle (b_BoardHandle);
```

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

Input

b_BoardHandle

Function diagram



Output

<Return value>

3.2 Interrupt

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

1) i_PA1000_SetBoardIntRoutineDos (..)

Syntax:

```
<Return value> = i_PA1000_SetBoardIntRoutineDos
                                    (BYTE    b_boardHandle
                                     VOID    v_FunctionName
                                                (BYTE    b_BoardHandle,
                                                 BYTE    b_InterruptMask,
                                                 BYTE    b_InputChannelNbr))
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1000**
VOID	v_FunctionName	Name of the user interrupt routine

- Output:

No output signal has occurred

Task:

This function must be called up for each **PA 1000** on which an interrupt action is to be enabled.

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

If you operate several boards **PA 1000** which have to react to interrupts, call up the function as often as you operate boards **PA 1000**.

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.

The user interrupt routine must have the following syntax:

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

v_FunctionName	Name of the user interrupt routine
b_BoardHandle	Handle of the **PA 1000** which has generated the interrupt
b_InterruptMask	Mask of the events which have generated the interrupt.
b_InputChannelNbr	If an interrupt is generated with a Mask 0000 000**1** and if you use the OR-PRIORITY logic, this variable gives the input number which have generated the interrupt.

Table 9-3: Interrupt mask

Mask	Meaning
0000 000**1**	Event 1 has occurred
0000 00**1**0	Event 2 has occurred
0000 0**1**00	Counter/timer 1 has run down
0000 **1**000	Counter/timer 2 has run down
000**1** 0000	Counter/timer 3 has run down

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

Calling convention:

ANSI C :

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

int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1000_SetBoardIntRoutineDos   (b_BoardHandle,
                                                  v_FunctionName );
```

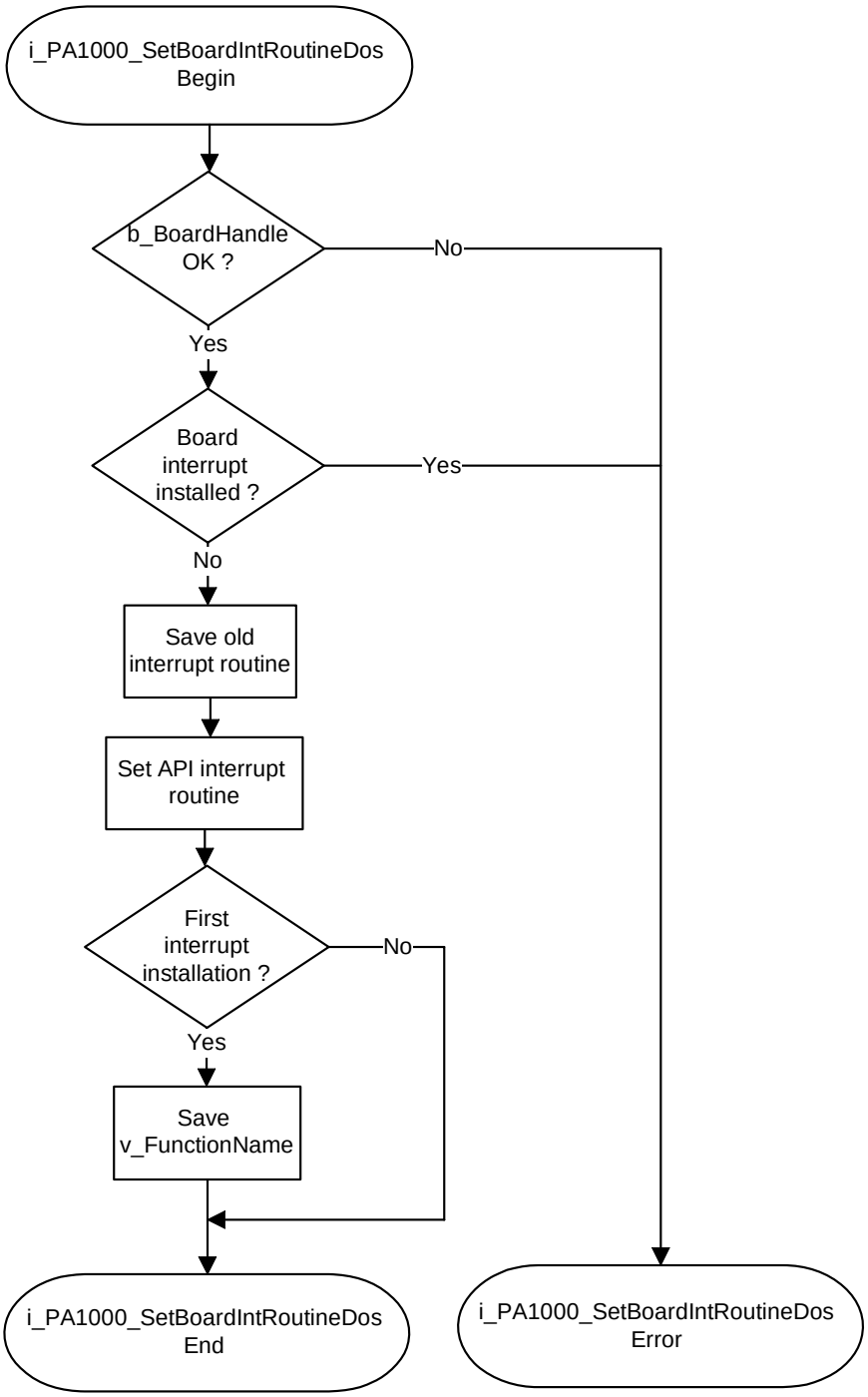
Return value:

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

Input

b_BoardHandle v_FunctionName

Function diagram



Output

<Return value>

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

2) i_PA1000_SetBoardIntRoutineVBDos (..)

Syntax:
<Return value> = i_PA1000_SetBoardIntRoutineVBDos
(BYTE b_BoardHandle)

Parameters:
- Input:
BYTE b_BoardHandle Handle of board **PA 1000**
- Output:
No output signal has occurred.

Task:
This function must be called up for each **PA 1000** 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 1000** which have to react to interrupts, call up the function as often as you operate boards **PA 1000**.

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_PA1000_TestInterrupt"

This function tests the interrupt of the **PA 1000**. It is used to obtain the values of *b_BoardHandle , b_InterruptMask, b_InputChannelNbr*.

Calling convention:

Visual Basic DOS:

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


IntLabel:


i_ReturnValue = i_PA1000_TestInterrupt    (i_BoardHandle, _
                                           i_InterruptMask,  _
                                           i_InputChannelNbr)
.
.
.

Return


ON UEVENT GOSUB IntLabel
UEVENT ON
i_ReturnValue = i_PA1000_SetBoardIntRoutineVBDos    (b_BoardHandle)
```

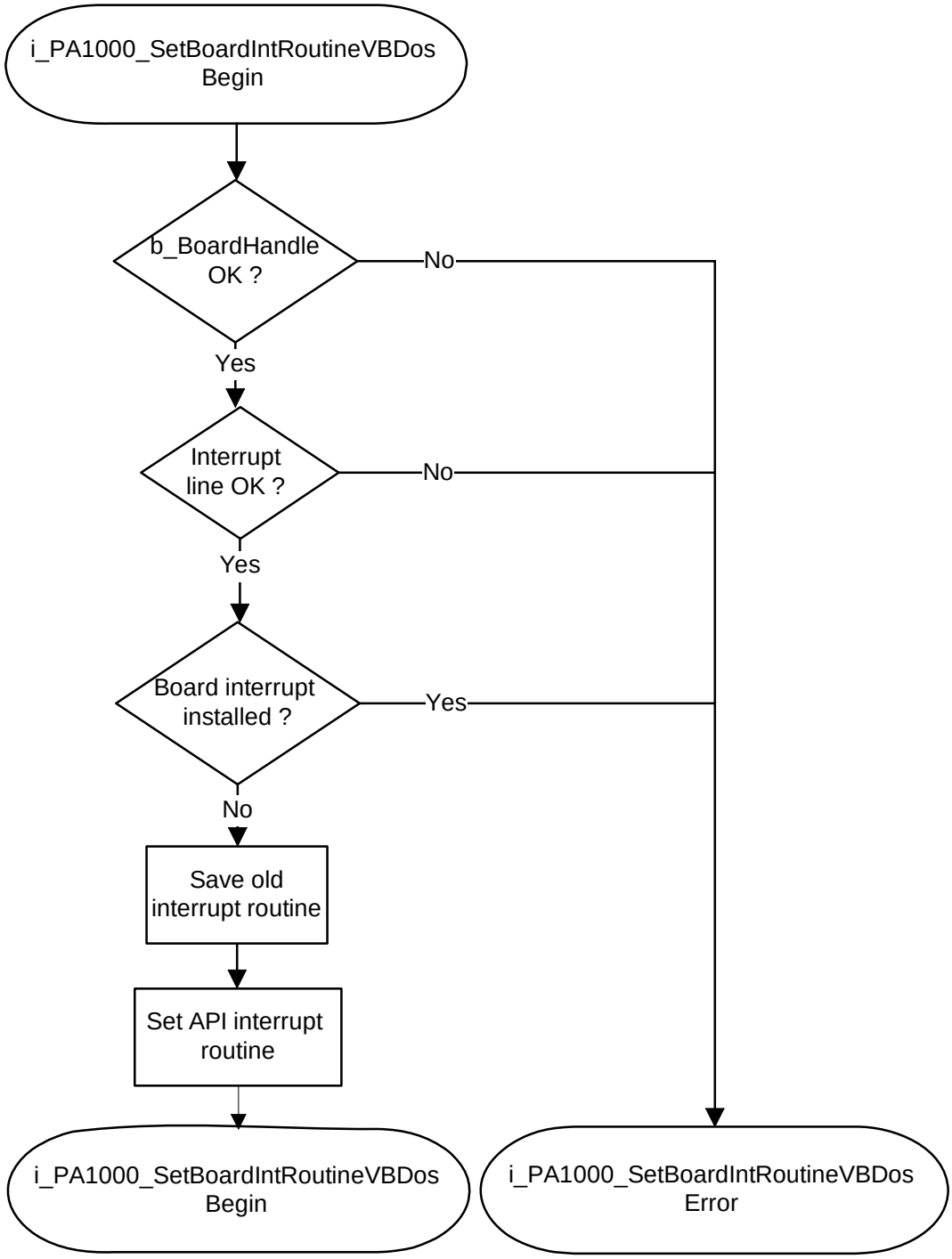
Return value:

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

Input

b_BoardHandle

Function diagram



Output

<Return value>

3) i_PA1000_SetBoardIntRoutineWin16

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

Syntax:

```
<Return value> = i_PA1000_SetBoardIntRoutineWin16
                                    (BYTE    b_boardHandle
                                     VOID    v_FunctionName
                                           (BYTE    b_BoardHandle,
                                            BYTE    b_InterruptMask,
                                            BYTE    b_InputChannelNbr))
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1000**
VOID	v_FunctionName	Name of the user interrupt routine

- Output:

No output signal has occurred

Task:

This function must be called up for each **PA 1000** on which an interrupt action is to be enabled.

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

If you operate several boards **PA 1000** which have to react to interrupts, call up the function as often as you operate boards **PA 1000**.

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.

The user interrupt routine must have the following syntax:

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

v_FunctionName	Name of the user interrupt routine
b_BoardHandle	Handle of the **PA 1000** which has generated the interrupt
b_InterruptMask	Mask of the events which have generated the interrupt.
b_InputChannelNbr	If an interrupt is generated with a Mask 0000 000**1** and if you use the OR-PRIORITY logic, this variable gives the input number which have generated the interrupt.

IMPORTANT!
If you use Visual Basic for Windows the following parameter has no meaning. You must use the „i_PA1000_TestInterrupt“ function.

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

Calling convention:

ANSI C :

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

int              i_ReturnValue;
unsigned char    b_BoardHandle;

i_ReturnValue = i_PA1000_SetBoardIntRoutineWin16 (b_BoardHandle,
                                                  v_FunctionName );
```

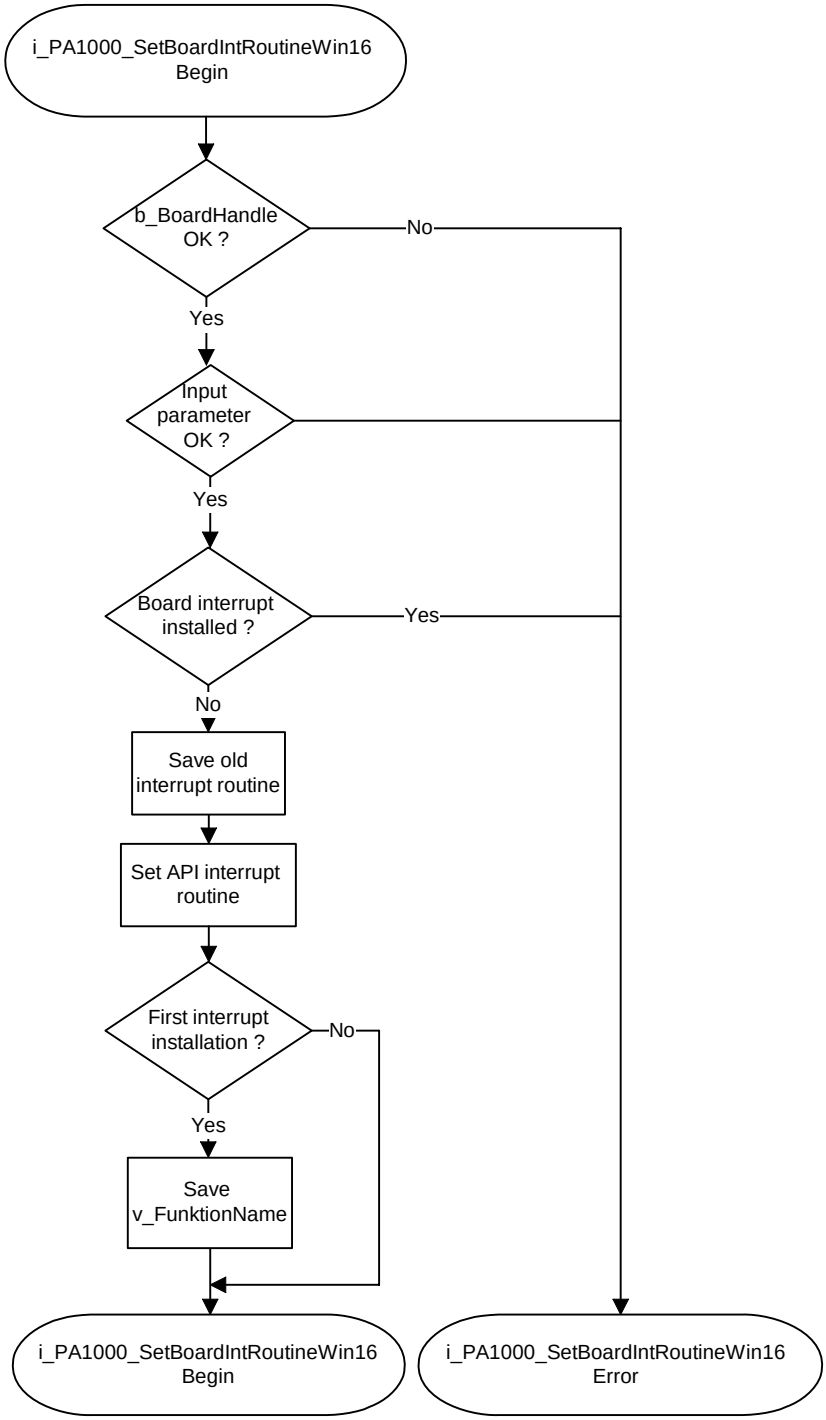
Return value:

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

Input

b_BoardHandle v_FunctionName

Function diagram



Output

<Return value>

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

4) i_PA1000_SetBoardIntRoutineWin32 (..)

Syntax:

```
<Return value> = i_PA1000_SetBoardIntRoutineWin32
                 (BYTE       b_BoardHandle,
                  BYTE       b_UserCallingMode,
                  ULONG      ul_UserSharedMemorySize,
                  VOID ** ppv_UserSharedMemory,
                  VOID       v_FunctionName    (BYTE      b_BoardHandle,
                                                BYTE      b_InterruptMask,
                                                BYTE      b_InputChannelNbr,
                                                BYTE      b_UserCallingMode,
                                                VOID *    pv_UserSharedMemory))
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1000**
BYTE	b_UserCallingMode	PA1000_SYNCHRONOUS_MODE: The user routine is directly called by the driver interrupt routine. PA1000_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 PA1000_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 PA1000_SYNCHRONOUS_MODE

Task:

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

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

- The user application operates in ring 3. This ring does not give access to hardware.
- VXD and SYS driver operate in ring 0 and give access to hardware.

- Ring 0 has no direct access to global variable from ring 3. It has to use a shared memory.
- Ring 0 and ring 3 have a pointer that points on this shared memory. The 2 pointers are not configured under the same address.

This function must be called up for each **PA 1000** 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 PA1000_SYNCHROUNOUS_MODE has been selected.

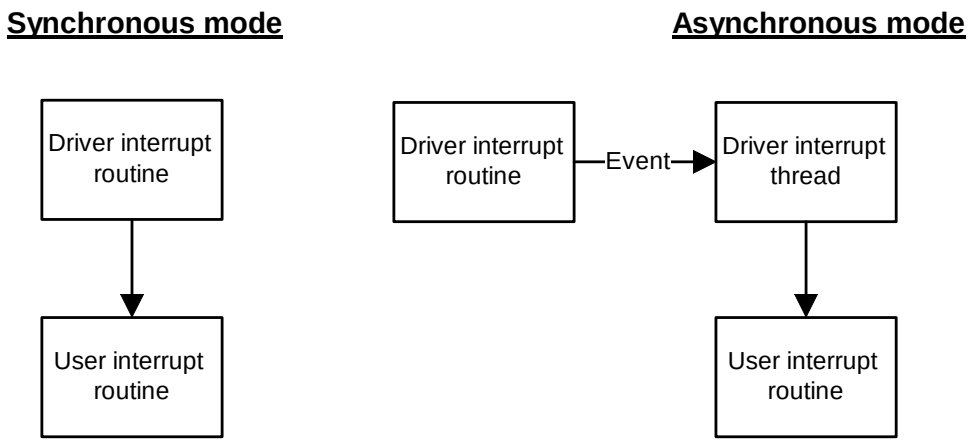
If you operate several boards **PA 1000** which have to react to interrupts, call up the function as often as you operate boards **PA 1000**. The variable *v_FunctionName* is only relevant **for the first calling**.

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
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 PA1000_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 (BYTEb_BoardHandle,
                        BYTE        b_InterruptMask,
                        BYTE        b_InputChannelNbr,
                        BYTE        b_UserCallingMode,
                        VOID *      pv_UserSharedMemory)
```

v_FunctionName	Name of the user interrupt routine
b_BoardHandle	Handle of the **PA 1000** which has generated the interrupt
b_InterruptMask	Mask of the events which have generated the interrupt.
b_InputChannelNbr	Is not used. But stays for compatibility reasons.
b_UserCallingMode	PA1000_SYNCHRONOUS_MODE: The user routine is directly called by driver interrupt routine. PA1000_ASYNCHRONOUS_MODE: The user routine is called by driver interrupt thread
pv_UserSharedMemory	Pointer of the user shared memory.

IMPORTANT!
If you use Visual Basic 4 the following parameters have no meaning. You must use the „i_PA1000_TestInterrupt“ function.

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

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

Calling convention:

ANSI C :

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

str_UserStruct * ps_UserSharedMemory;

void      v_FunctionName     (unsigned char   b_BoardHandle,
                              unsigned char   b_InterruptMask,
                              unsigned char   b_InputChannelNbr,
                              unsigned char   b_UserCallingMode,
                              void *          pv_UserSharedMemory)
          {
          str_UserStruct * ps_InterruptSharedMemory;

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

```
i_ReturnValue = i_ PA1000_SetBoardIntRoutineWin32
                                (b_BoardHandle, PA1000_SYNCHRONOUS_MODE,
                                 sizeof (str_UserStruct),
                                 (void **) &ps_UserSharedMemory,
                                 v_FunctionName);
```

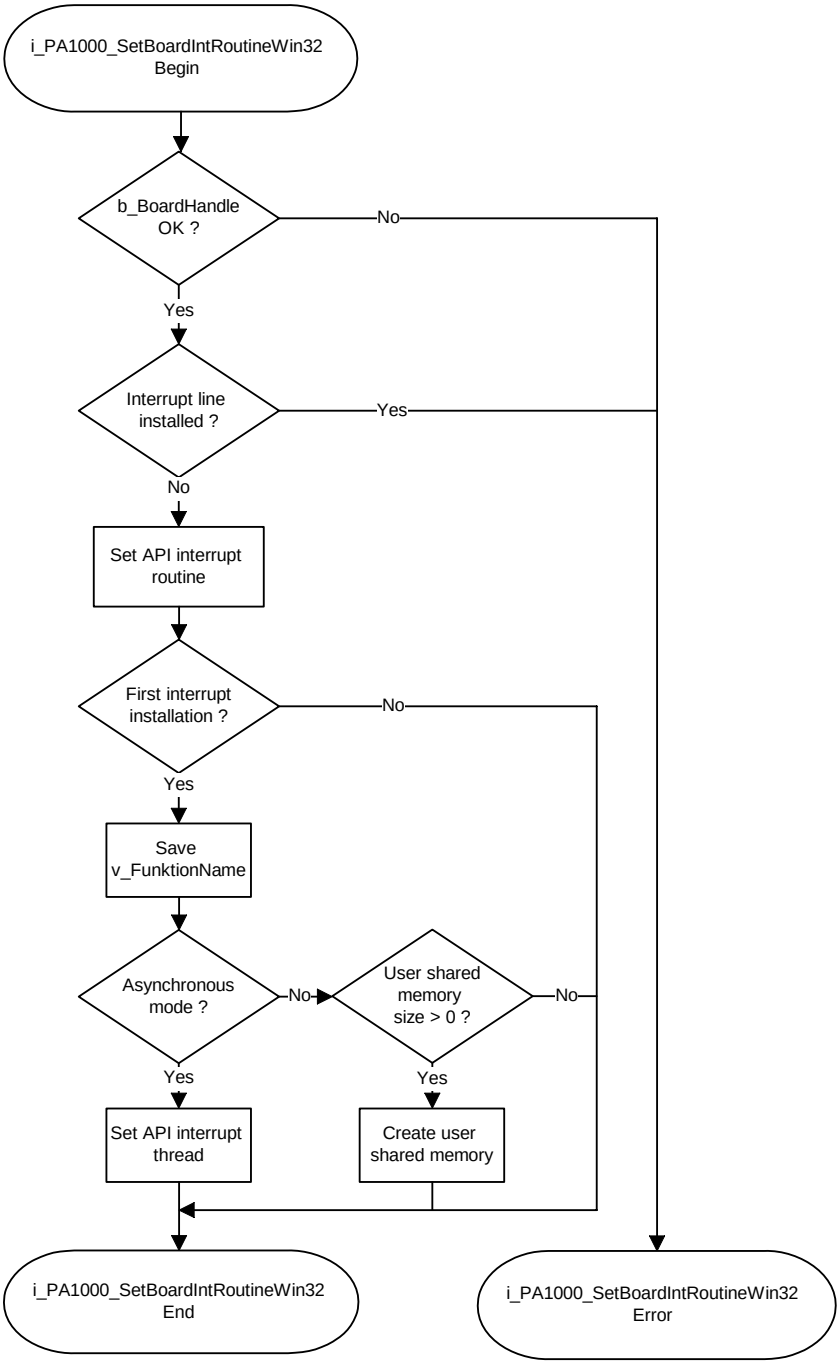
Return value:

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

Input

b_BoardHandle	ul_UserSharedMemorySize
b_UserCallingMode	v_FunctionName

Function diagram



Output

ppv_UserSharedMemory

<Return value>

5) i_PA1000_TestInterrupt (..)

Syntax:
<Return value> = i_PA1000_TestInterrupt (PBYTE pb_BoardHandle,
PBYTE pb_InterruptMask,
PBYTE pb_ChannelNbr)

Parameters:
- Input:
No input signal has occurred.
- Output:
PBYTE pb_BoardHandle Handle of the board **PA 1000** which has generated the interrupt,
PBYTE pb_InterruptMask Error mask of the event which has generated the interrupt. Several errors can simultaneously occur.

Mask	Meaning
0000 000**1**	Event 1 has occurred
0000 00**1**0	Event 2 has occurred
0000 0**1**00	Counter/timer 1 has run down
0000 **1**000	Counter/timer 2 has run down
000**1** 0000	Counter/Timer3 has run down

PBYTE pb_ChannelNbr Is not used. But stays for compatibility reasons

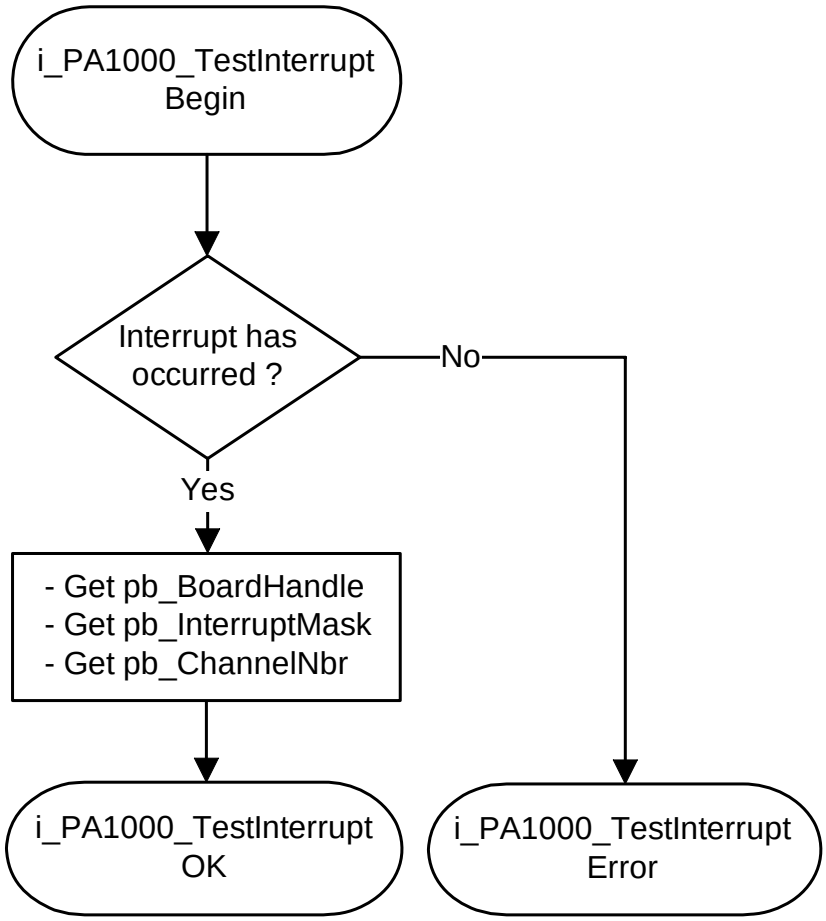
Task:
Checks if a board **PA 1000** has generated an interrupt. If yes, the function returns the board handle and the interrupt source.

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

Calling convention:
ANSI C :

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

i_Irq = i_PA1000_TestInterrupt      (&b_BoardHandle,
                                     & b_InterruptMask,
                                     & b_ChannelNbr);
```

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

Input
Function diagram

Output
pb_BoardHandle pb_ChannelNbr pb_InterruptMaske
<Return value>

6) i_PA1000_ResetBoardIntRoutine (..)

Syntax:
<Return value> = i_PA1000_ResetBoardIntRoutine
(BYTE b_BoardHandle)

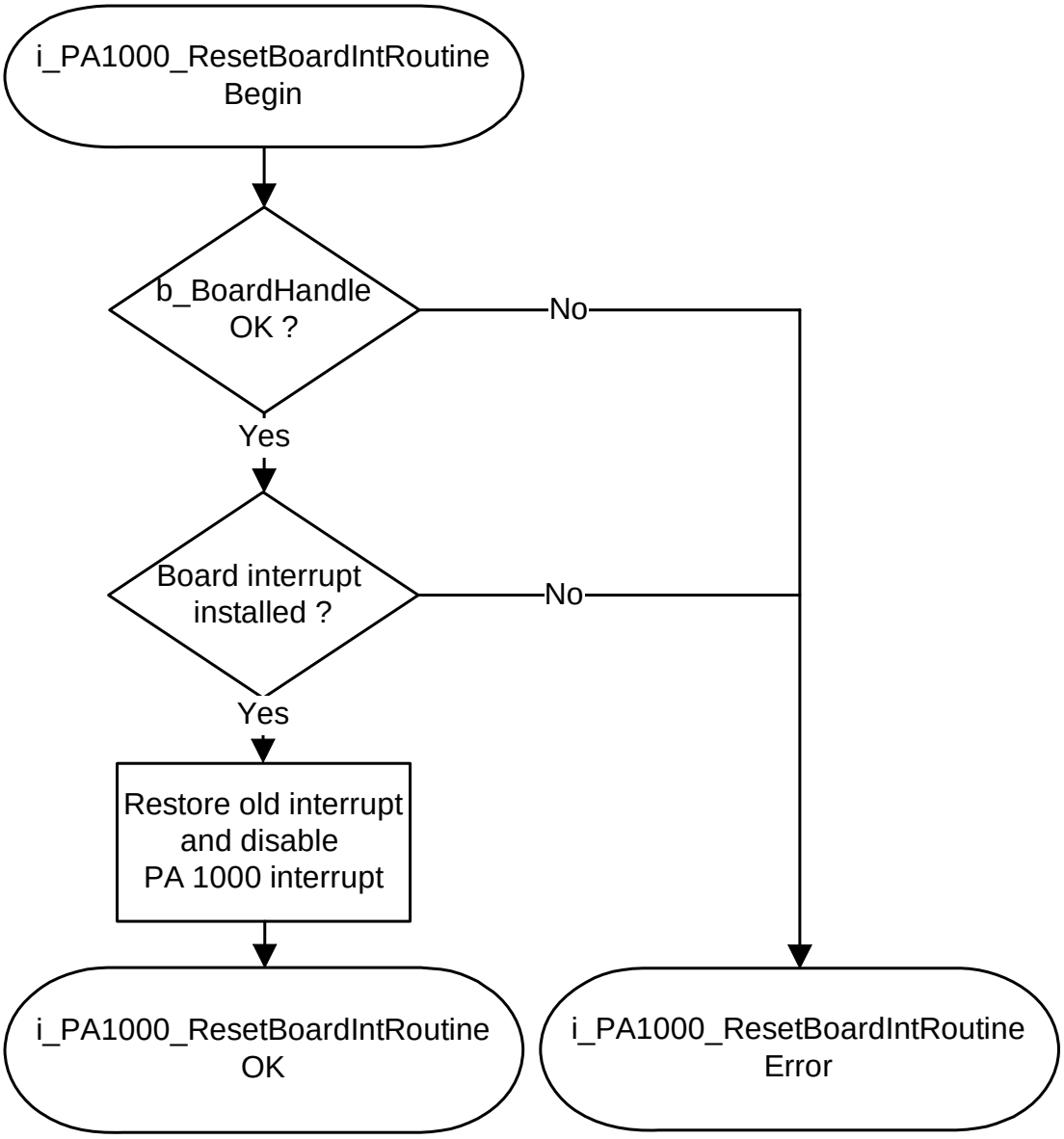
Parameters:
- Input:
BYTE b_BoardHandle Handle of board **PA 1000**
- Output:
No output signal has occurred

Task:
Stops the interrupt management of board **PA1000.**
Deinstalls the interrupt routine if the interrupt management of all **PA 1000** is stopped.

Calling convention:
ANSI C :

```
unsigned char    b_BoardHandle;

i_PA1000_ResetBoardIntRoutine       (b_BoardHandle);
```

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

Input
b_BoardHandle
Function diagram

Output
<Return value>

3.3 Digital input channels

1) i_PA1000_Read1DigitalInput (...)

Syntax :
<Return value> = i_PA1000_Read1DigitalInput (BYTE_ b_BoardHandle,
BYTE_ b_Channel,
PBYTE_ pb_ChannelValue)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1000**
BYTE	b_Channel	Number of the input channel to be read (1 to 32)

- Output:

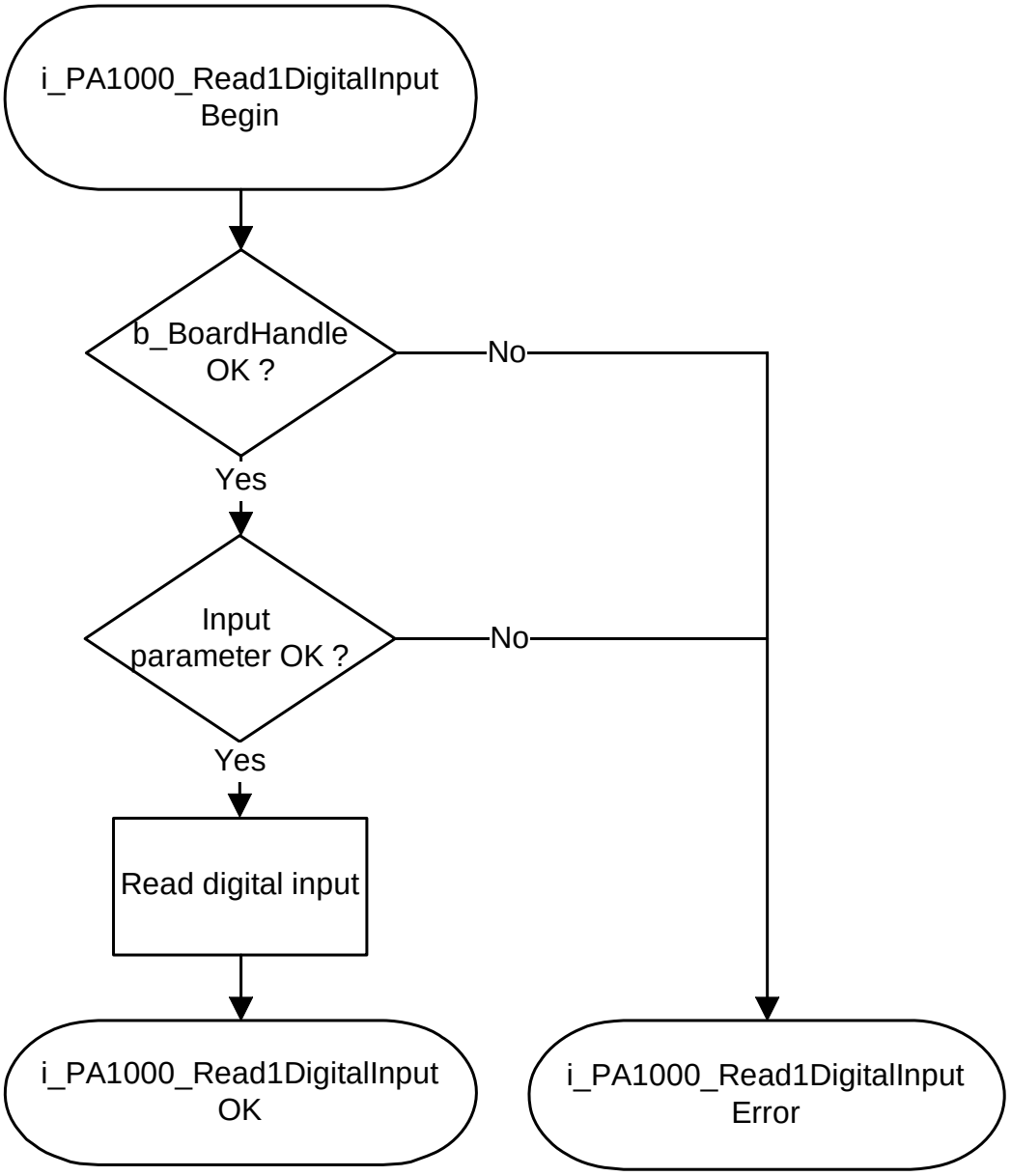
PBYTE	pb_ChannelValue	State of the digital input channel 0 -> low 1 -> high

Task:
Indicates the state of an input. The variable *b_Channel* passes the input channel to be read (1 to 32). A value is returned with the variable *pb_ChannelValue* : 0 (low) or 1 (high).

Calling convention:
ANSI C :

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

i_ReturnValue = i_PA1000_Read1DigitalInput    (b_BoardHandle,
                                               1,
                                               &b_ChannelValue);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The input number is not between 1 and 32

Input
b_BoardHandle b_Channel
Function diagram

Output
pb_ChannelValue
<Return value>

2) i_PA1000_Read8DigitalInput (...)

Syntax:

```
<Return value> =  i_PA1000_Read8DigitalInput  (BYTE   b_BoardHandle,
                                               BYTE   b_Port,
                                               PBYTE pb_PortValue)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the **PA 1000**
BYTE	b_Port	Number of the input port to be read (1 or 4)

- Output:

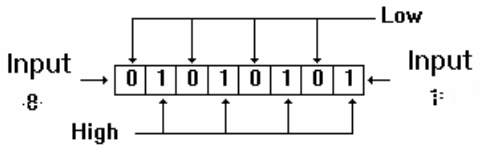
PBYTE	pb_PortValue	State of the digital input port (0 to 255)

Task:

Indicates the state of an 8-bit port. The variable *b_Port* passes the port to be read (1 or 4). A value is returned with the variable *pb_PortValue* .

Example:

b_Port = 1
pb_PortValue = 55 Hex



A voltage is present on the input channels 1, 3, 5, 7
A voltage is not present on the input channels 2, 4, 6, 8.

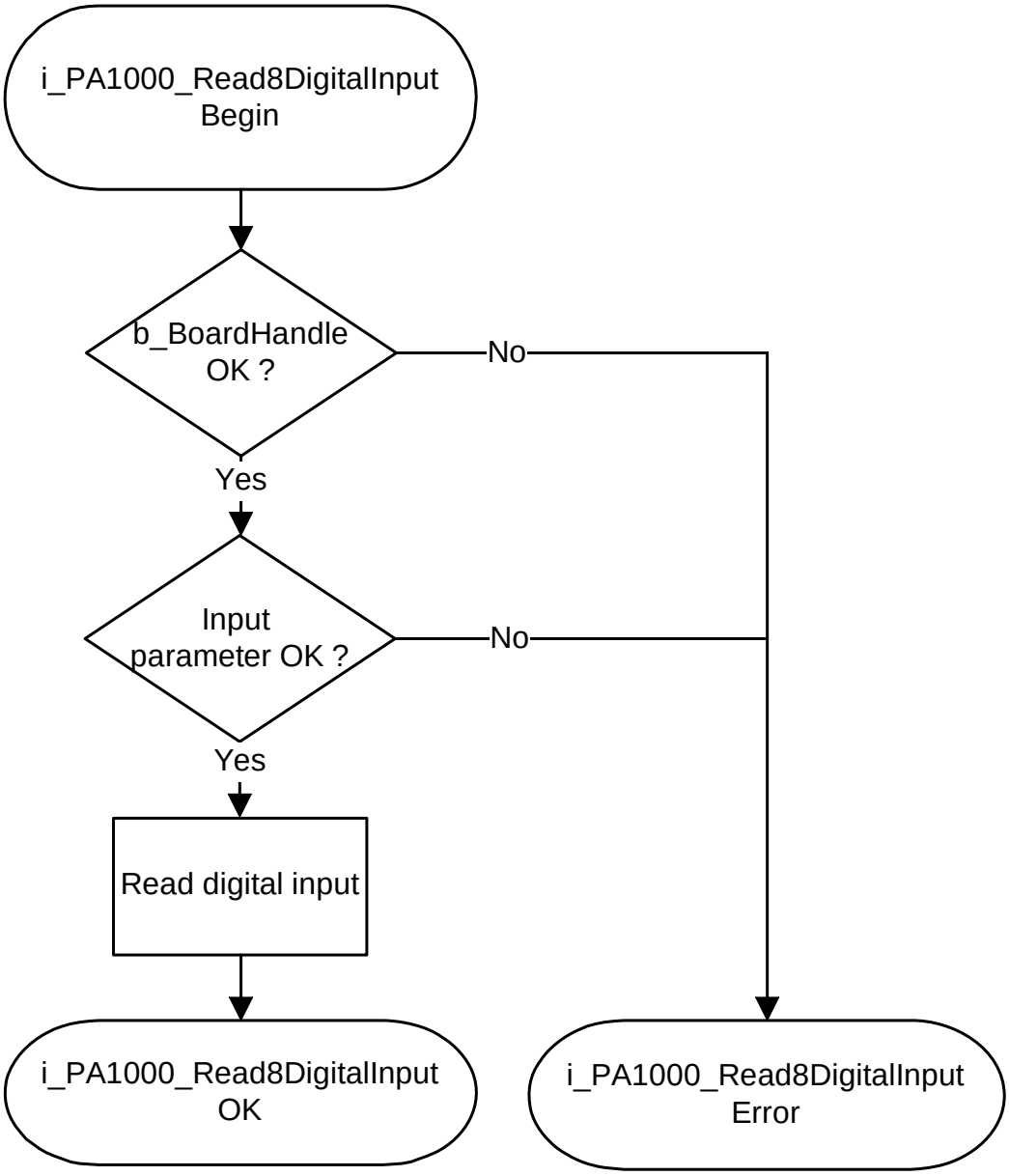
Calling convention:

ANSI C :

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

i_ReturnValue = i_PA1000_Read8DigitalInput      (b_BoardHandle,
                                                 1,
                                                 &b_PortValue);
```

Return value:

0: No error
-1: Handle parameter of the board is wrong
-2: The parametered port number is wrong (parameter 1 or 4)

Input
b_BoardHandle b_Port
Function diagram

Output
pb_PortValue
<Return value>

3) i_PA1000_Read16DigitalInput (...)

Syntax:
<Return value> = i_PA1000_Read16DigitalInput (BYTE b_BoardHandle,
BYTE b_Port
PLONG pl_InputValue)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of the **PA 1000**
BYTE	b_Port	Number of the 16-bit input port to be read (1 or 2)

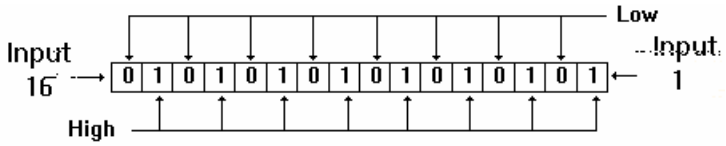
- Output:

PLONG	pl_InputValue	State of the digital input channels of both ports (0 to 65535)

Task:
Indicates the state of a 16-bit port. The variable b_Port passes the port to be read (1 or 2). A value is returned with the variable *pl_InputValue*

Example:

Parameter
b_Port = 1
pl_InputValue = 5555 Hex

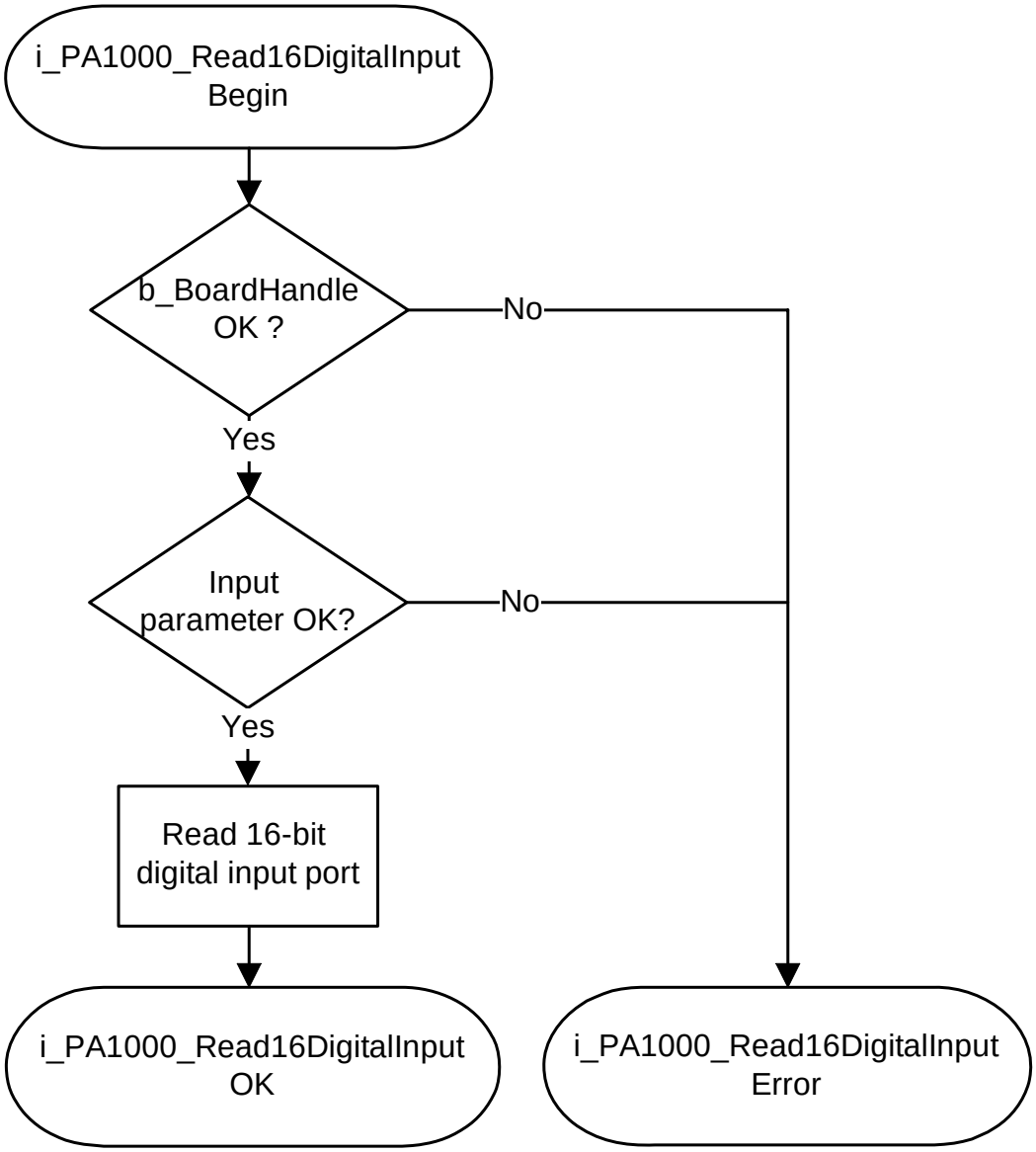


A voltage is present on the input channels 1, 3, 5, 7, 9, 11, 13, 15 .
A voltage is not present on the input channels 2, 4, 6, 8, 10, 12, 14, 16.

Calling convention:
ANSI C :

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

i_ReturnValue = i_PA1000_Read16DigitalInput
                                    (b_BoardHandle,
                                     1,
                                     & ul_PortValue);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The parametered port number is wrong (Parameter 1 or 2)

Input
b_BoardHandle b_Port
Function diagram

Output
pl_InputValue
<Return value>

4) i_PA1000_Read32DigitalInput (...)

Syntax:
<Return value> = i_PA1000_Read32DigitalInput (BYTE b_BoardHandle,
PLONG pl_InputValue)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1000**

- Output:

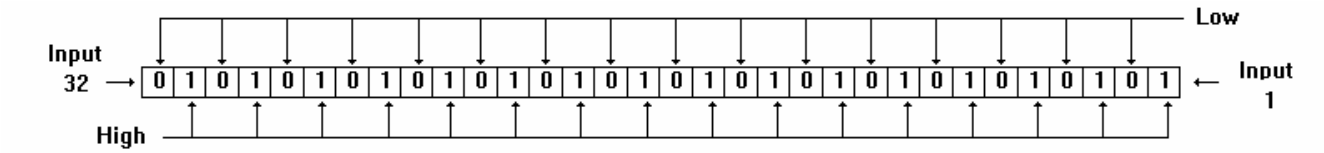
PLONG	pl_InputValue	State of the digital input channels of both ports (0 to 2^{32})

Task:
Indicates the state of the 32 inputs channels.

Example:

Parameter

pl_InputValue = 555555555 Hex

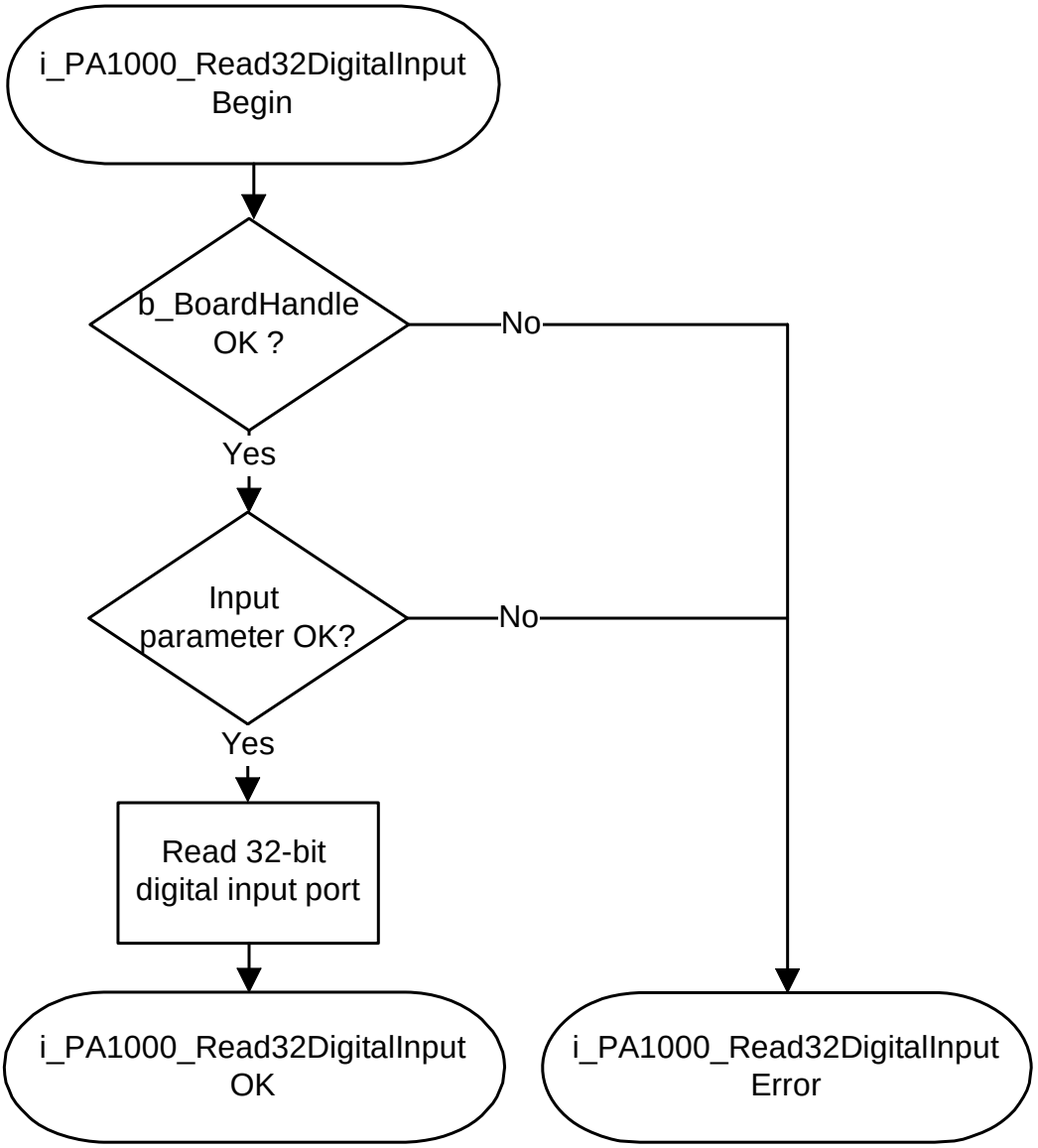


A voltage is present on the input channels 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29 and 31.
A voltage is not present on the input channels 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30 and 32.

Calling convention:
ANSI C :

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

i_ReturnValue = i_PA1000_Read32DigitalInput (b_BoardHandle,
                                             &ul_PortValue);
```

Return value:
0: No error
-1: Handle parameter of the board is wrong

Input
b_BoardHandle
Function diagram

Output
pl_InputValue
<Return value>

3.4 Digital input channel - events

1) i_PA1000_SetInputEventMask (...)

Syntax:

<Return value> = i_PA1000_SetInputEventMask
(BYTE b_BoardHandle,
BYTE b_PortNbr,
BYTE b_Logik,
PCHAR pc_EventMask)

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1000**
BYTE	b_PortNbr	Number of the input port you wish to mask (1 or 2).
BYTE	b_Logik	Event logic Three possibilities for the two ports: PA1000_AND: This logic connects the inputs with an AND logic. PA1000_OR: This logic connects the inputs with an OR logic.
PCHAR	pc_EventMask	This 8-digit character string defines the event mask of 8 inputs. Each digit indicates the input state. The state is identified by one of the following characters: "X": The input channel is not used for event "0": The input channel must be on "0" "1": The input channel must be on "1" "2": The input channel reacts to a falling edge "3": The input channel reacts to a rising edge "4": The input channel reacts to both edges Port 1: from the left to the right, the first digit of the character string is input 8 and the last digit is input 1. Port 2: from the left to the right, the first digit of the character string is input 16 and the last digit is input 9.

- Output:

No output signal has occurred

IMPORTANT!
If you use the PA1000_AND logic, you can use only one edge event.

Task:

An event can be generated for each port.

The first event is related to the first 8 inputs (port 1). The second event is related to the next 6 inputs (port 2).
An interrupt is generated when one or both events have happened.

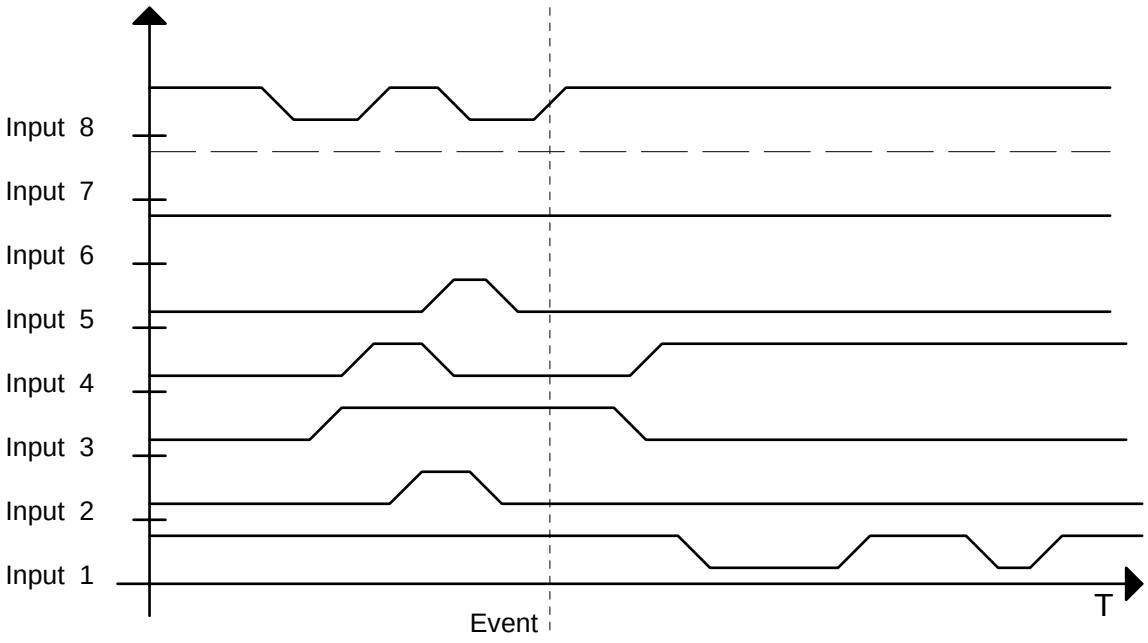
An event is a change of state (eg.: low→ high, high → low) on one or several input channels if an AND/OR/OR_PRIORITY logic has been defined.

Examples:

Example 1:

Parameter
b_PortNbr = 1
b_Logik = PA1000_AND
pc_EventMask = "3X100101"

An event is generated:
- when the inputs 2, 4 and 5 are on "0".
- when the inputs 1, 3 and 6 are on "1"
- and when a rising edge has been detected at input 8.



Calling convention:

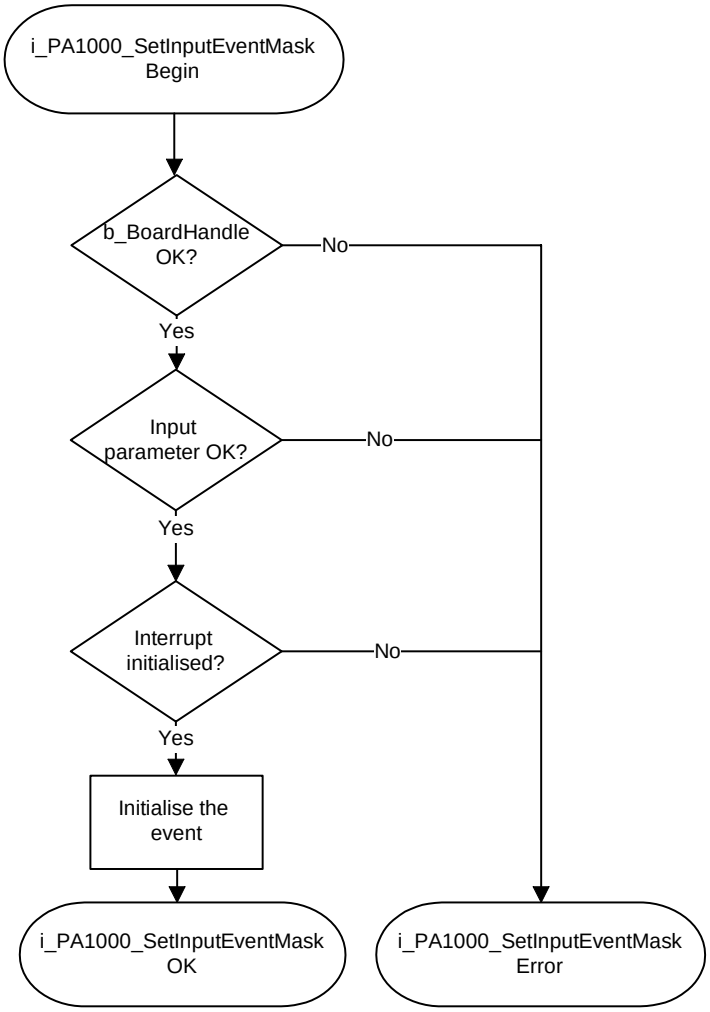
ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue =  i_PA1000_SetInputEventMask     (b_BoardHandle,
                                                 1,
                                                 PA1000_OR,
                                                 "11111111");
```

Return value:

0: No error
-1: The handle parameter of the board is wrong
-2: The parametered port number is wrong (parameter 1 or 2)
-3: Error with the logic parameter. *b_Logik* has not the expected value
-4: Error with the event mask parameter. *pc_EventMask* has not the expected value
-5: Interrupt routine not installed
-6: More than 1 edge event has been declared for an AND logic
-7: The OR PRIORITY logic does not support any edge event

Input	
b_BoardHandle	b_Logik
b_PortNbr	pc_EventMask
Function diagram	



Output

<Return value>