ADDI-DATA®

CE

Technical description

ADDICOUNT PA 1700-2

Standard software

Edition: 05.01-07/2005

Tables

1 INTRODUCTION

IMPORTANT!
Note the following conventions in the text:

Function: "i_PA1700_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

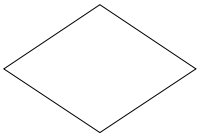
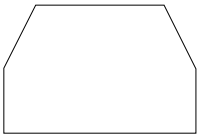
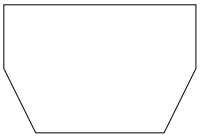
Define name	Decimal value	Hexadecimal value
DLL_COMPILER_C	0	0
DLL_COMPILER_VB	1	1
DLL_COMPILER_PASCAL	2	2
DLL_LABVIEW	3	3
DLL_COMPILER_VB_5	4	4
PA1700_DISABLE	0	0
PA1700_ENABLE	1	1

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.

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

3 SOFTWARE FUNCTIONS (API)

3.1 Board initialisation

1) i_PA1700_InitCompiler (..)

Syntax:
<Return value> = i_PA1700_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.

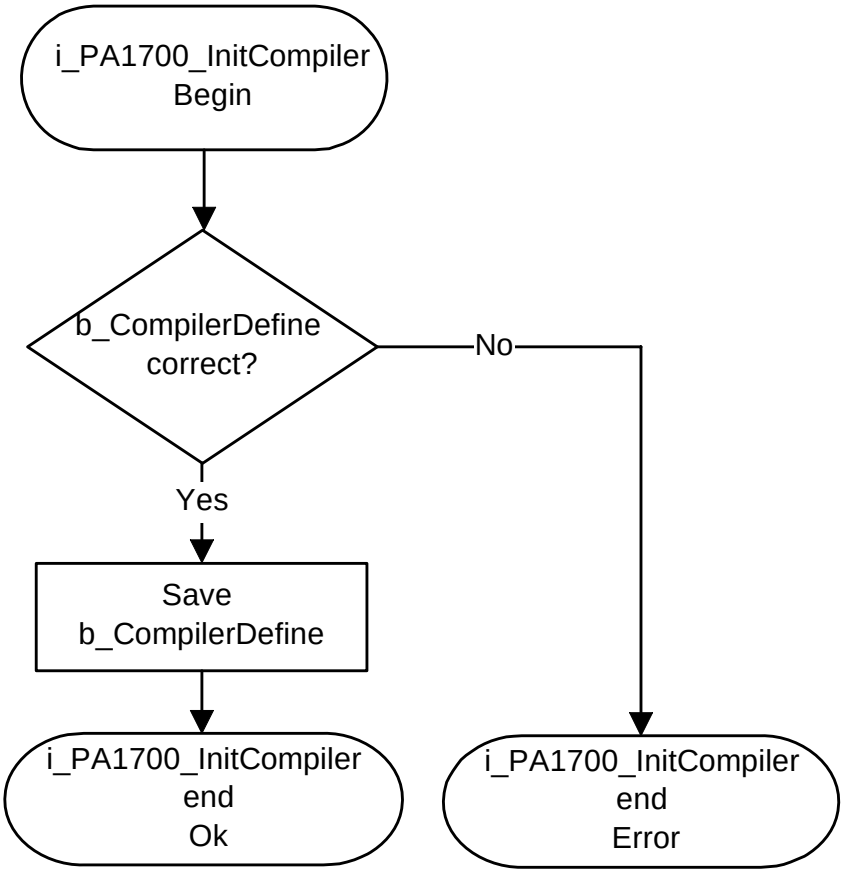
WICHTIG!
This function is only available with a Windows environment.

Calling convention:

ANSI C :

```
int i_ReturnValue;

i_ReturnValue = i_PA1700_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_PA1700_SetBoardInformation (...)

Syntax:

```
<Return value> =  i_PA1700_SetBoardInformation
                                    (UINT     ui_Address,
                                     BYTE     b_StrobeInterruptNbr,
                                     BYTE     b_TimerInterruptNbr,
                                     PBYTE    pb_BoardHandle)
```

Parameters:

- Input:

UINT	ui_Address	Base address of board **PA 1700**
BYTE	b_StrobeInterruptNbr	External strobe interrupt line of the board (IRQ3, 5, 10, 11, 12, 14, 15) If 0, no interrupt line is used
BYTE	b_TimerInterruptNbr	Timer interrupt line of the board (IRQ3, 5, 10, 11, 12, 14, 15) If 0, no interrupt line is used

- Output:

PBYTE	pb_BoardHandle	Handle[1] of board **PA 1700** to use the functions

Task:
Verifies if board **PA 1700** is present. Stores the following information:
- base address,
- the external strobe interrupt number
- the timer 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_PA1700_SetBoardInformation  (0x390, 0, 0,
                                               &b_BoardHandle);
```

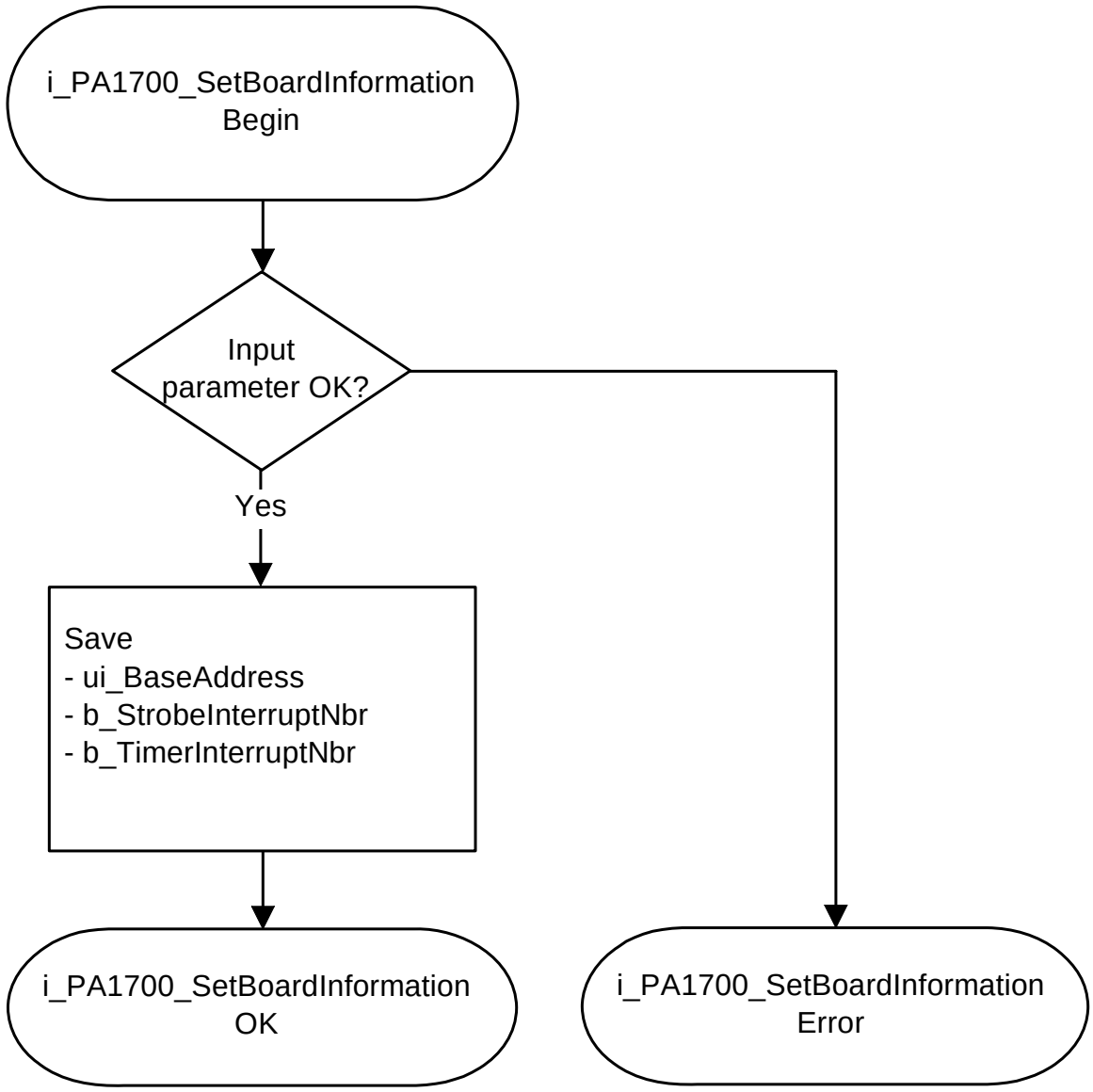
Return value:
0: No error
-1: Board not present
-2: External strobe IRQ number is wrong
-3: Timer IRQ number is wrong
-4: No handle is available for the board (up to 10 handles can be used)

[1] Identification number of the board

Input

ui_BaseAddress b_TimerInterruptNbr
b_StrobeInterruptNbr

Function diagram



Output

pb_BoardHandle

<Return Value>

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

3) i_PA1700_SetBoardInformationWin32 (...)

Syntax:
<Return value> = i_PA1700_SetBoardInformationWin32
(PCHAR pc_Identifier
PBYTE pb_BoardHandle)

Parameters:
- Input:

PCHAR	pc_Identifier	Identifier string for the **PA 1700** selection The identifier string is determined by the ADDIREG registration program.

- Output:

PBYTE	pb_BoardHandle	Handle of the board PA 1700 to use the functions

Task:
Call up all hardware information about the **PA 1700** given by the ADDIREG registration program and stores the following information:
- the base address,
- the external strobe interrupt number,
- the timer interrupt number,

Then verifies if the board **PA 1700** 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 :

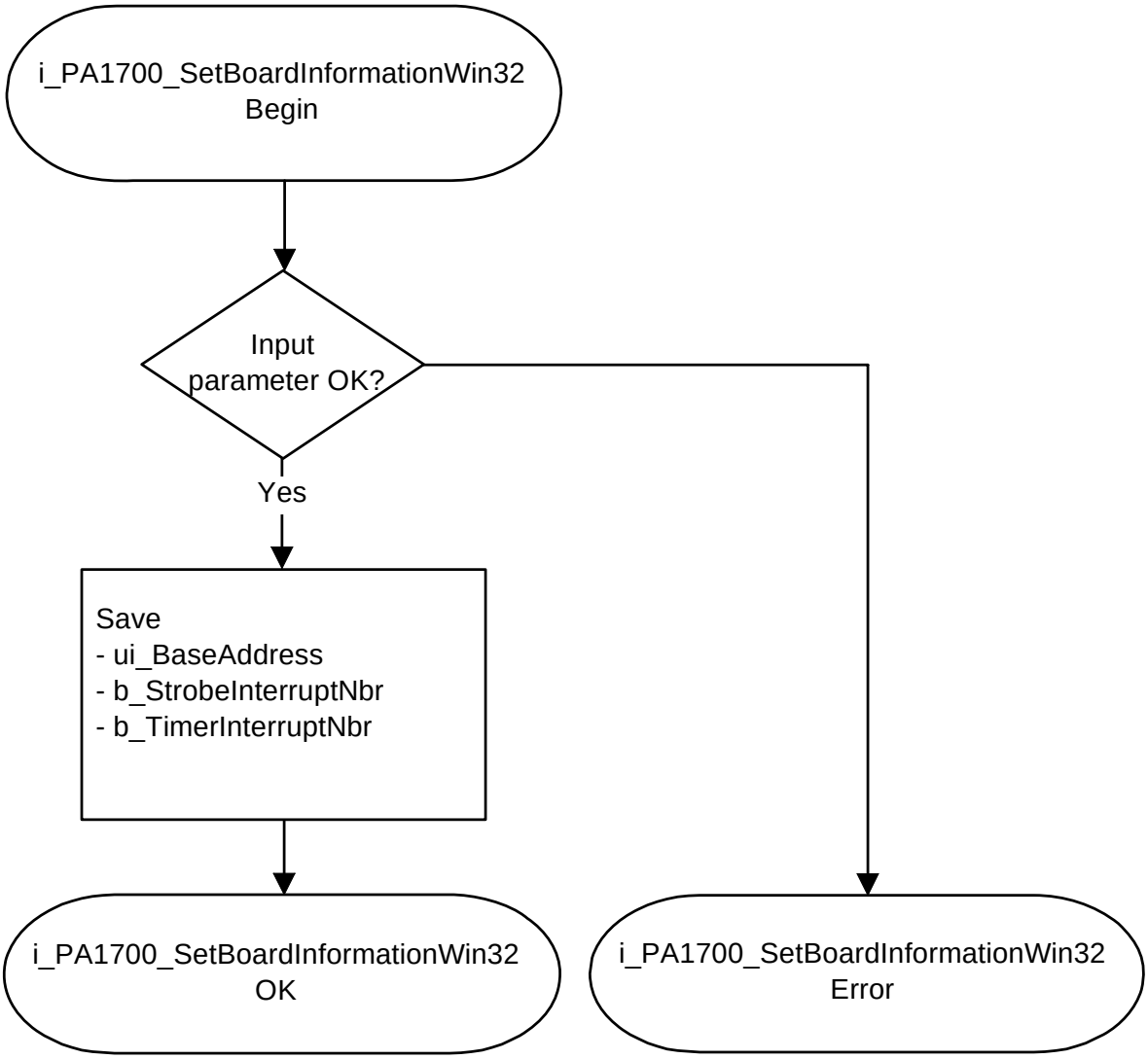
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_SetBoardInformationWin32
                                    ("PA1700-00", &b_BoardHandle);
```

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

Input

pc_Identifier

Function diagram



Output

pb_BoardHandle

<Return Value>

4) i_PA1700_GetHardwareInformation (...)

```
<Return value> =  i_PA1700_GetHardwareInformation
                                        (BYTEb_BoardHandle,
                                         PUINT      pui_BaseAddress,
                                         PBYTE      pb_StrobeInterruptNbr,
                                         PBYTE      pb_TimerInterruptNbr)
```

Parameters:

-Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**

- Output:

PUINT	pui_BaseAddress	**PA 1700** base address
PBYTE	pb_StrobeInterruptNbr	External strobe interrupt channel.
PBYTE	pb_TimerInterruptNbr	Timer interrupt channel.

Task:

Returns the base address and the interrupt number of the **PA 1700**.

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned int    ui_BaseAddress;
unsigned char   b_StrobeInterruptNbr;
unsigned char   b_TimerInterruptNbr;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_GetHardwareInformation
                                        (b_BoardHandle,
                                         &ui_BaseAddress,
                                         &b_StrobeInterruptNbr,
                                         & b_TimerInterruptNbr);
```

Return value:

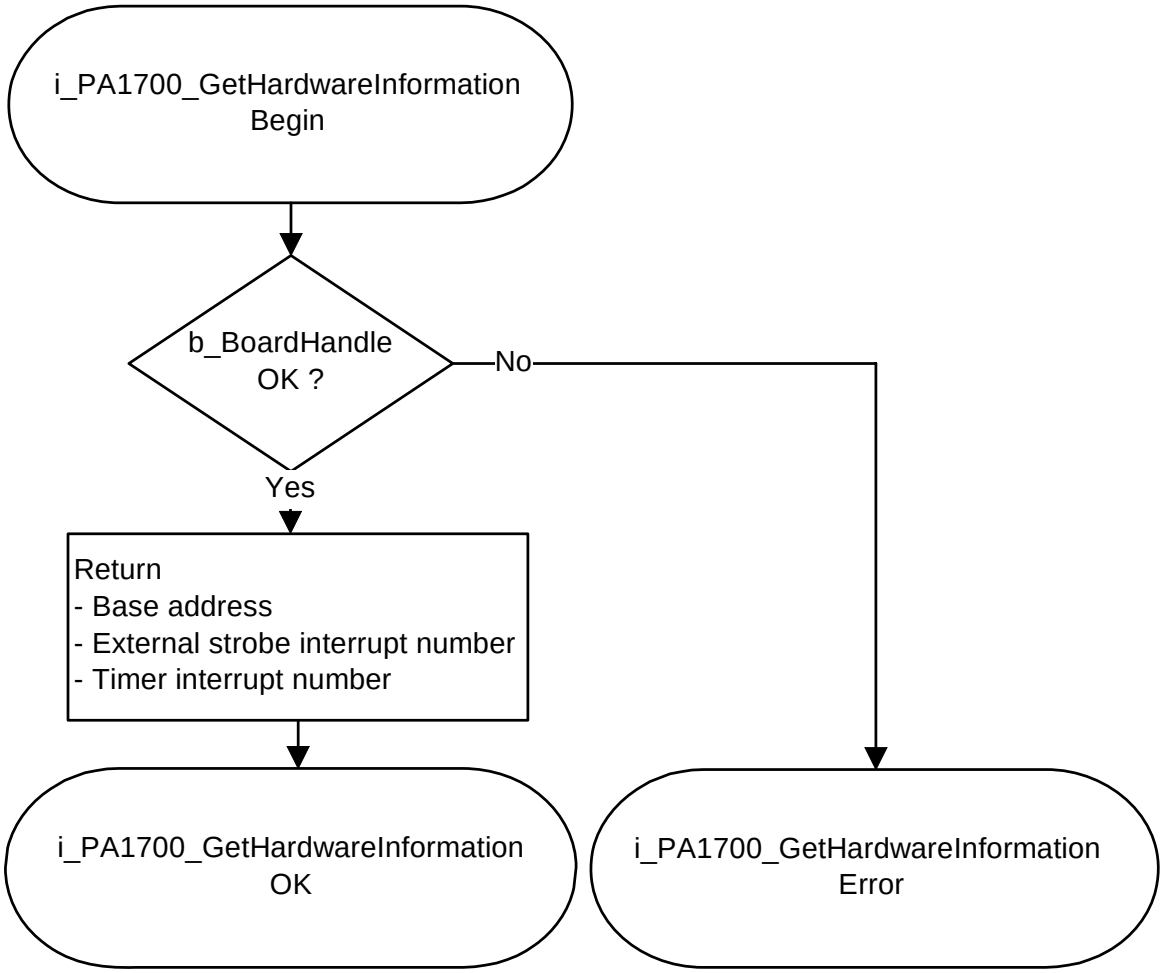
0: No error

-1: The handle parameter of the board is wrong

Input

b_BoardHandle

Function diagram



Output

pui_BaseAddress pb_TimerInterruptNbr
pb_StrobeInterruptNbr

<Return Value>

5) i_PA1700_CloseBoardHandle

Syntax:
<Return value> = i_PA1700_CloseBoardHandle (BYTE b_BoardHandle)

Parameters:
- Input
BYTE b_BoardHandle Handle of board **PA 1700**
- Output:
No output signal has occurred.

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

Calling convention:
ANSI C :

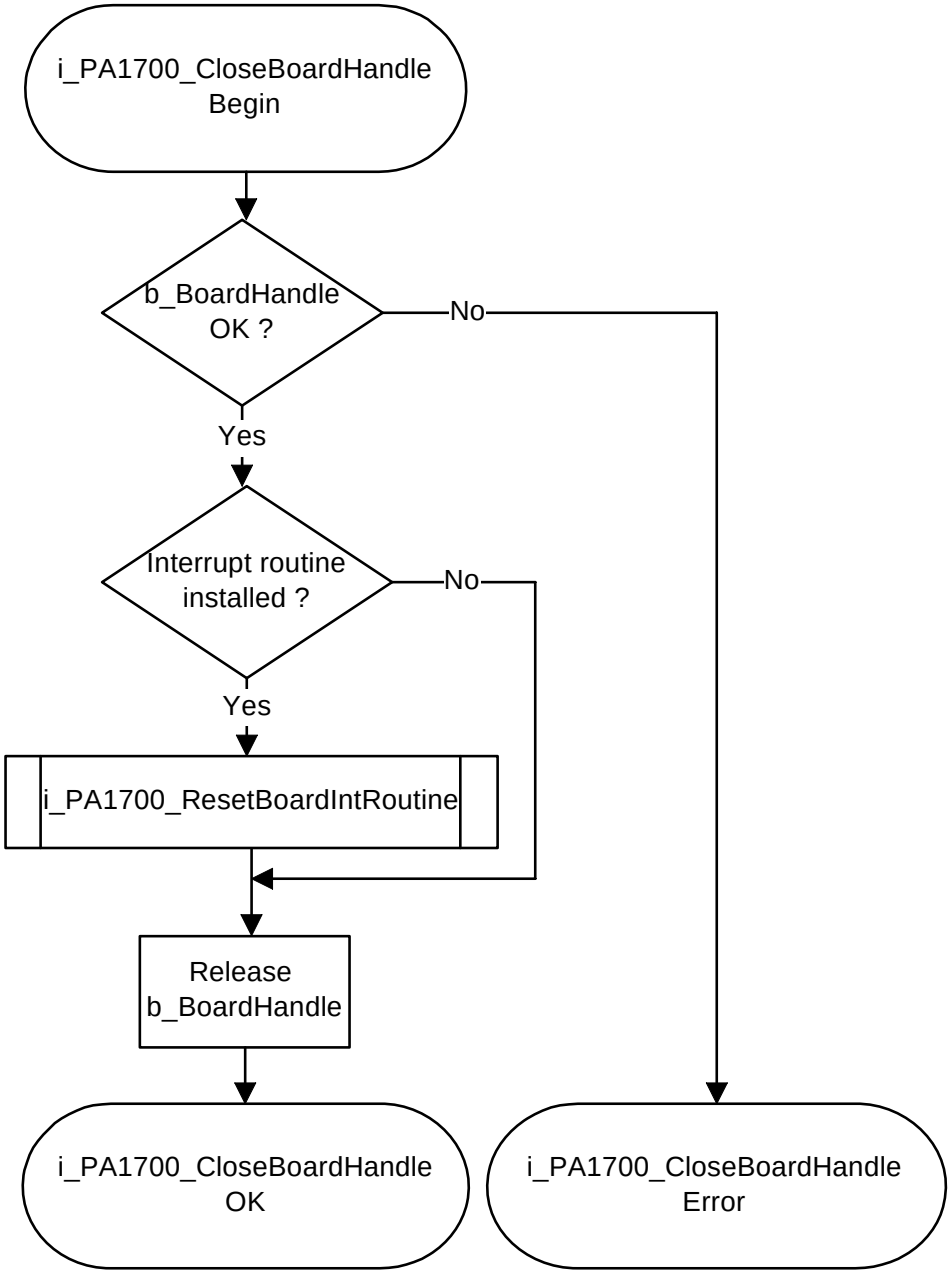
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_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 DOS applications

1) i_PA1700_SetBoardIntRoutineDos (..)

Syntax:
<Return value> = i_PA1700_SetBoardIntRoutineDos
(BYTEb_boardHandle
VOID v_FunctionName
(BYTE b_BoardHandle,
BYTE b_InterruptMask,
ULONG ul_CounterLatchValue))

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
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 1700** on which an interrupt is to be enabled. It installs a user interrupt on all boards on which the interrupt has been enabled.

When the function is called up for the first time (first board):
- the user interrupt routine is installed
- interrupts are enabled.

If you operate several boards **PA 1700** which have to react to interrupts, call up the function as often as you operate boards **PA 1700**.

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 1700** 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,
                           ULONG    ul_CounterLatchValue)
```

v_FunctionName	Name of the user interrupt routine
b_BoardHandle	Handle of the **PA 1700** which has generated the interrupt
b_InterruptMask	Mask of the events which have generated the interrupt
ul_CounterLatchValue	The value of latched counter

Table 3-1: Interrupt mask

b_InterruptMask	Meaning
0000 000**1**	Hardware latch on the first register of hardware counter 0 (32-bit)
0000 00**1**0	Hardware latch on the second register of hardware counter 0 (32-bit)
0000 0**1**00	Hardware latch on the first register of hardware counter 1 (32-bit)
0000 **1**000	Hardware latch on the second register of hardware counter 1 (32-bit)
000**1** 0000	Hardware latch on the first register of hardware counter 2 (32-bit)
00**1**0 0000	Hardware latch on the second register of hardware counter 2 (32-bit)
0**1**00 0000	Timer interrupt
1000 000**1**	Overflow interrupt has occurred on hardware counter 0
1000 0**1**00	Overflow interrupt has occurred on hardware counter 1
100**1** 0000	Overflow interrupt has occurred on hardware counter 2

b_InterruptMask	Source	*ul_CounterLatchValue* value
b_InterruptMask = 1	Hardware strobe on the first latch register of hardware counter 0	Latched value of the first latch register (32-bit)
b_InterruptMask = 2	Hardware strobe on the second latch register of hardware counter 0	Latched value of the second latch register (32-bit)
b_InterruptMask = 4	Hardware strobe on the first latch register of hardware counter 1	Latched value of the first latch register (32-bit)
b_InterruptMask = 8	Hardware strobe on the second latch register of hardware counter 1	Latched value of the second latch register (32-bit)
b_InterruptMask = 16	Hardware strobe on the first latch register of hardware counter 2	Latched value of the first latch register (32-bit)
b_InterruptMask = 32	Hardware strobe on the second latch register of hardware counter 2	Latched value of the second latch register (32-bit)

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

Calling convention:

ANSI C :

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

int               i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_SetBoardIntRoutineDos
                                        (b_BoardHandle,
                                         v_FunctionName );
```

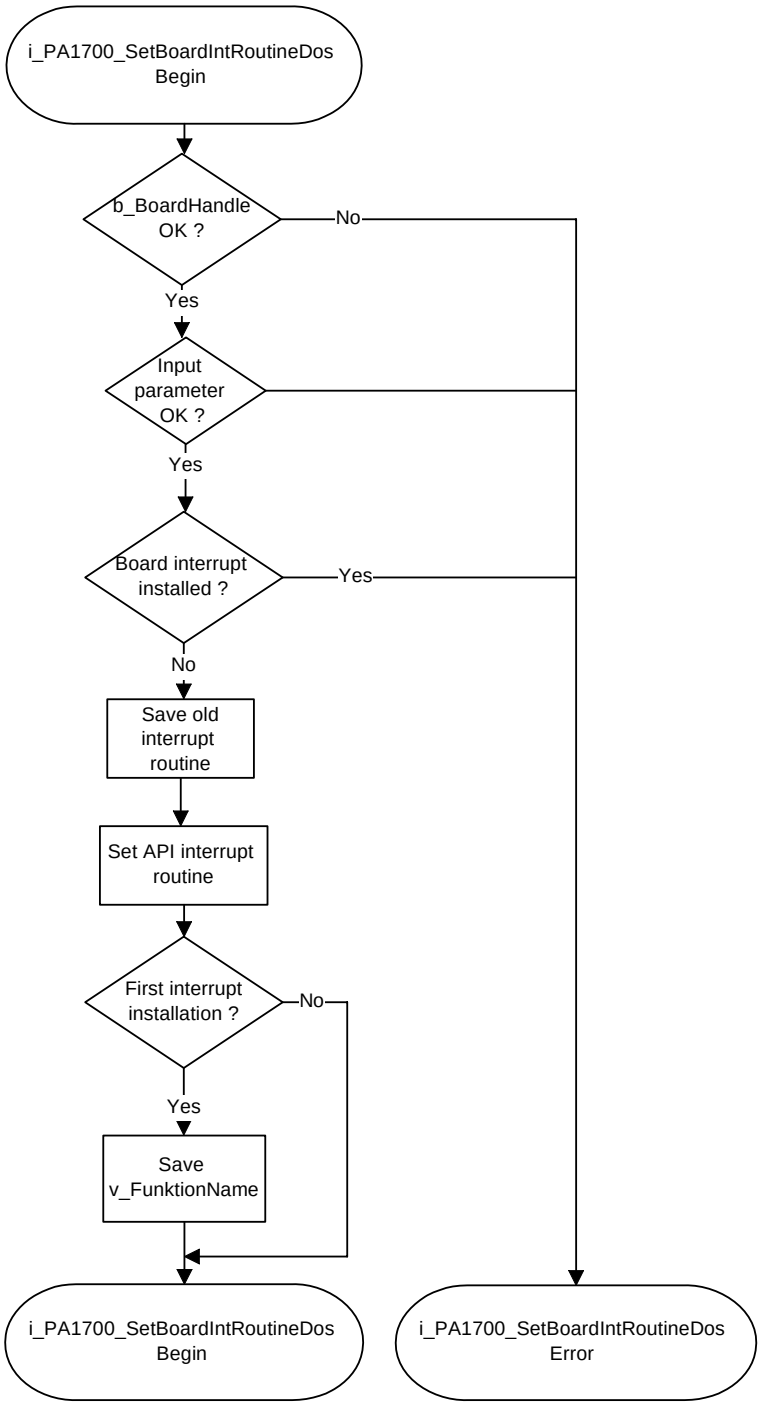
Return value:

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

Input

b_BoardHandle v_FunctionName

Function diagram



Output

<Return Value>

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

2) i_PA1700_SetBoardIntRoutineVBDos (..)

Syntax:
<Return value> = i_PA1700_SetBoardIntRoutineVBDos
(BYTE b_BoardHandle)

Parameters:
- Input:
BYTE b_BoardHandle Handle of board **PA 1700**
- Output:
No output signal has occurred.

Task:
This function must be called up for each **PA 1700** 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 1700** which have to react to interrupts, call up the function as much as you operate boards **PA 1700**.

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_PA1700_TestInterrupt"

This function tests the interrupt of the **PA 1700**. It is used to obtain the values of *b_BoardHandle , b_InterruptMask, ul_CounterLatchValue*.

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_CounterLatchValue  As Long

IntLabel:

i_ReturnValue = i_PA1700_TestInterrupt    (i_BoardHandle, _
                                           i_InterruptMask,  _
                                           l_CounterLatchValue)
.
.
.

Return

ON UEVENT GOSUB IntLabel
UEVENT ON
i_ReturnValue = i_PA1700_SetBoardIntRoutineVBDos    (b_BoardHandle)
```

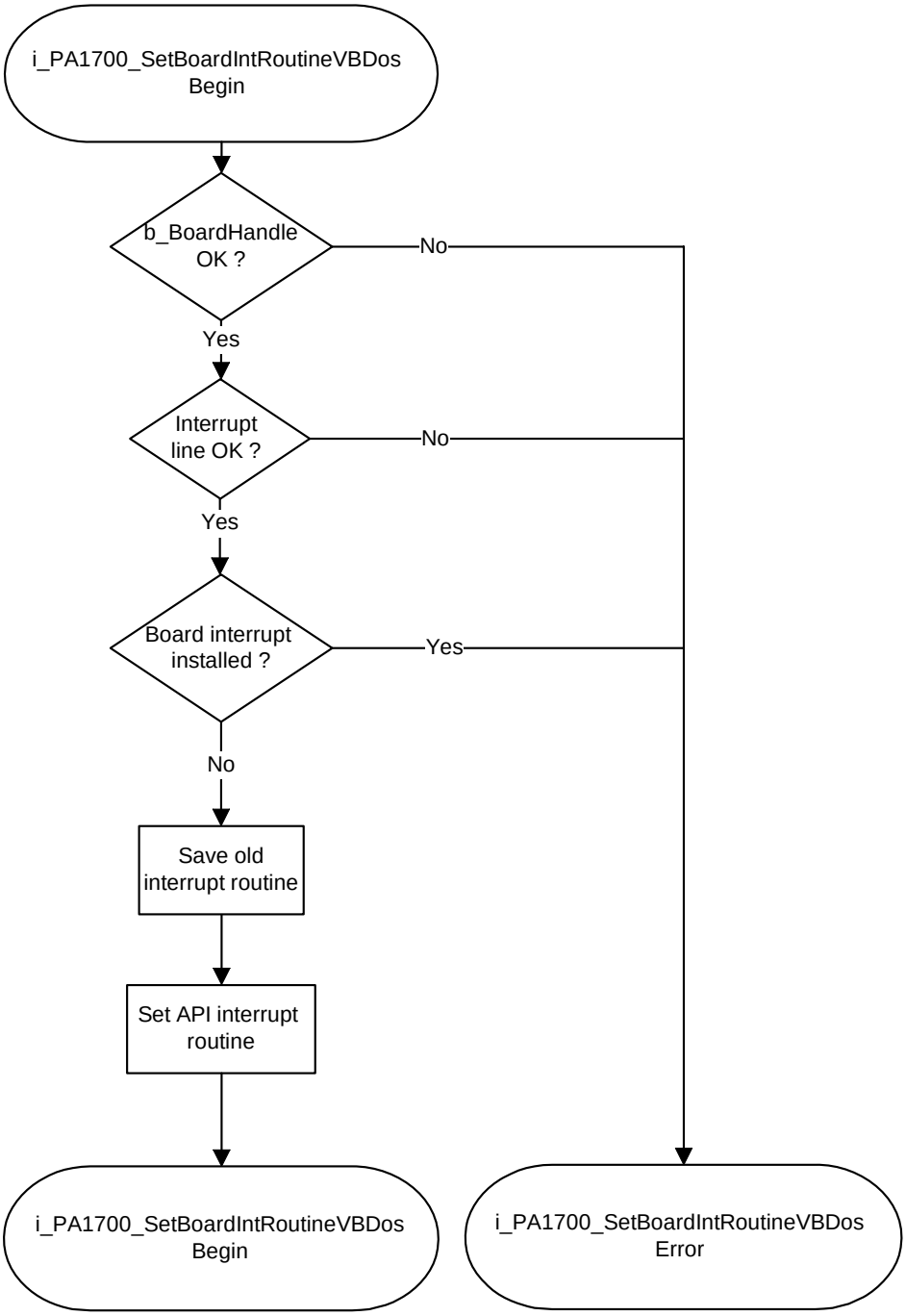
Return value:

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

Input

b_BoardHandle

Function diagram



Output

<Return Value>

3) i_PA1700_SetBoardIntRoutineWin16

Syntax:

```
<Return value> = i_PA1700_SetBoardIntRoutineWin16
                 (BYTE     b_BoardHandle,
                  VOID     v_FunctionName  (BYTEb_BoardHandle,
                                            BYTE        b_InterruptMask,
                                            ULONG       ul_CounterLatchValue))
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the **PA 1700**
VOID	v_FunctionName	Name of the user interrupt routine.

- Output:

No output signal has occurred.

Task:

This function can be called up several times.

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

If you operate several boards **PA 1700** which have to react to interrupts, call up the function as often as you operate boards **PA 1700**. 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,
                           ULONG    ul_CounterLatchValue)
```

v_FunctionName	Name of the user interrupt routine
b_BoardHandle	Handle of the **PA 1700** which has generated the interrupt
b_InterruptMask	Mask of the events which have generated the interrupt
ul_CounterLatchValue	The value of latched counter

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

IMPORTANT!
If you use Visual Basic for Windows the following parameters have no meaning. Please use the "PA1700_TestInterrupt" function.

```
VOID   v_FunctionName                (BYTE      b_BoardHandle,
                                      BYTE      b_InterruptMask,
                                      ULONG     ul_CounterLatchValue)
```

Calling convention:
ANSI C :

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

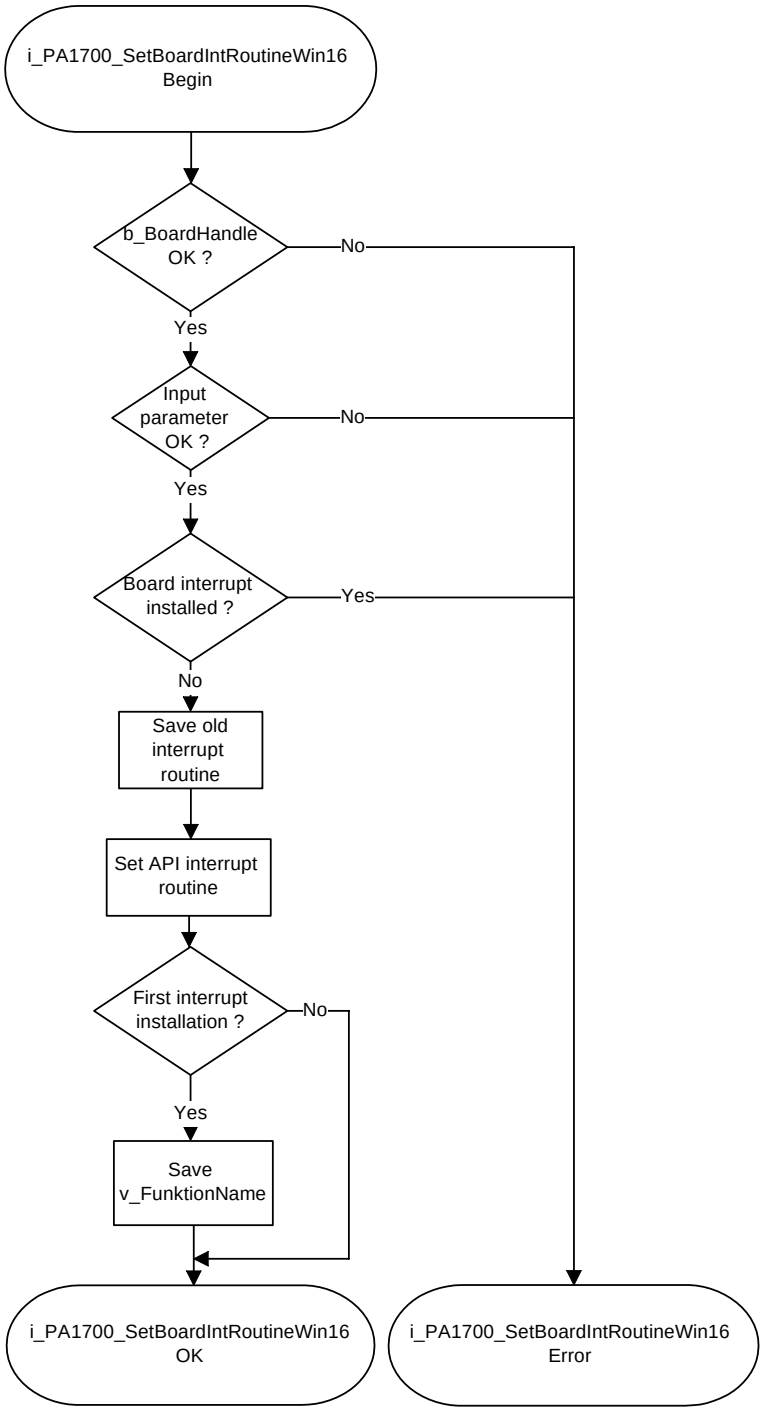
```
int                  i_ReturnValue;
unsigned char        b_BoardHandle;

i_ReturnValue = i_PA1700_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

Input

b_BoardHandle v_FunctionName

Function diagram



Output

<Return Value>

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

4) i_PA1700_SetBoardIntRoutineWin32 (..)

Syntax:

```
<Return value> = i_PA1700_SetBoardIntRoutineWin32
                 (BYTE     b_BoardHandle,
                  BYTE     b_UserCallingMode,
                  ULONG    ul_UserSharedMemorySize,
                  VOID **  ppv_UserSharedMemory,
                  VOID     v_FunctionName  (BYTE     b_BoardHandle,
                                            BYTE     b_InterruptMask,
                                            ULONG    ul_CounterLatchValue,
                                            BYTE     b_UserCallingMode,
                                            VOID *   ppv_UserSharedMemory))
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_UserCallingMode	PA1700_SYNCHRONOUS_MODE : The user routine is directly called by the driver interrupt routine. PA1700_ASYNCHRONOUS_MODE : The user routine is called by driver the interrupt thread.
VOID	v_FunctionName	Name of the user interrupt routine
ULONG	ul_UserSharedMemorySize	Determines the size in byte of the user shared memory. Only used if you have selected PA1700_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 PA1700_SYNCHRONOUS_MODE

Task:

This function can be called up several times.

This function must be called up for each PA 1700 on which an interrupt is to be enabled. It installs a user interrupt on all boards on which the interrupt has been enabled.

When the function is called up for the first time (first board):
- the user interrupt routine is installed
- interrupts are enabled.

If you operate several boards **PA 1700** which have to react to interrupts, call up the function as often as you operate boards **PA 1700**.

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.

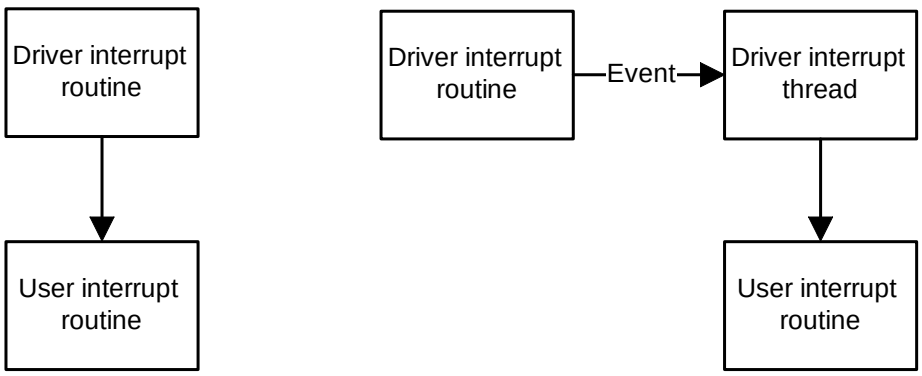
If several boards **PA 1700** are operated and if they have to react to interrupts, the variable *b_BoardHandle* returns the identification number (handle) of the board which has generated the interrupt.

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

- The user application operates in ring 3. This ring does not give access to hardware.
- VXD and SYS driver operate in ring 0 and give access to hardware.
- Ring 0 has no direct access to 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.

User interrupt routine can be called :
- directly by the driver interrupt routine (synchronous mode). The code of the user interrupt routine directly operates in ring 0.
- by the driver interrupt thread (asynchronous mode). An event is generated and the interrupt thread calls up the user interrupt routine. The code of the user interrupt routine operates in ring 3.
  The driver interrupt thread have the highest priority (31) in the system.

Synchronous mode **Asynchronous mode**



	SYNCHRONOUS MODE
ADVANTAGE	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.
RESTRICTION	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 1700 driver functions with the following extension "i_PA1700_KRNL_XXXX"
	This mode is not available for Visual Basic

	ASYNCHRONOUS MODE
ADVANTAGE	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 1700 driver functions with the following extension: "i_PA1700_XXXX"
RESTRICTION	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 PA1700_SYNCHRONOUS_MODE you cannot have access to the 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-defined 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,
                       ULONG  ul_CounterLatchValue,
                       BYTE   b_UserCallingMode,
                       VOID * pv_UserSharedMemory)
```

v_FunctionName	Name of the user interrupt routine
b_BoardHandle	Handle of the **PA 1700** which has generated the interrupt
b_InterruptMask	Mask of the events which have generated the interrupt
ul_CounterLatchValue	The value of latched counter
b_UserCallingMode	PA1700_SYNCHRONOUS_MODE: The user routine is directly called by the driver interrupt routine. PA1700_ASYNCHRONOUS_MODE: The user routine is called by the driver interrupt thread
pv_UserSharedMemory	Pointer of the user shared memory.

IMPORTANT!
If you use Visual Basic 4 the following parameters have no meaning. You must use the "i_PA1700_TestInterrupt" function.

```
BYTE     b_UserCallingMode,
ULONG    ul_UserSharedMemorySize,
VOID **  ppv_UserSharedMemory,

VOID     v_FunctionName  (BYTE   b_BoardHandle,
                          BYTE   b_InterruptMask
                          ULONG  ul_CounterLatchValue,
                          BYTE   b_UserCallingMode,
                          VOID * pv_UserSharedMemory)
```

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

Calling convention:
ANSI C:

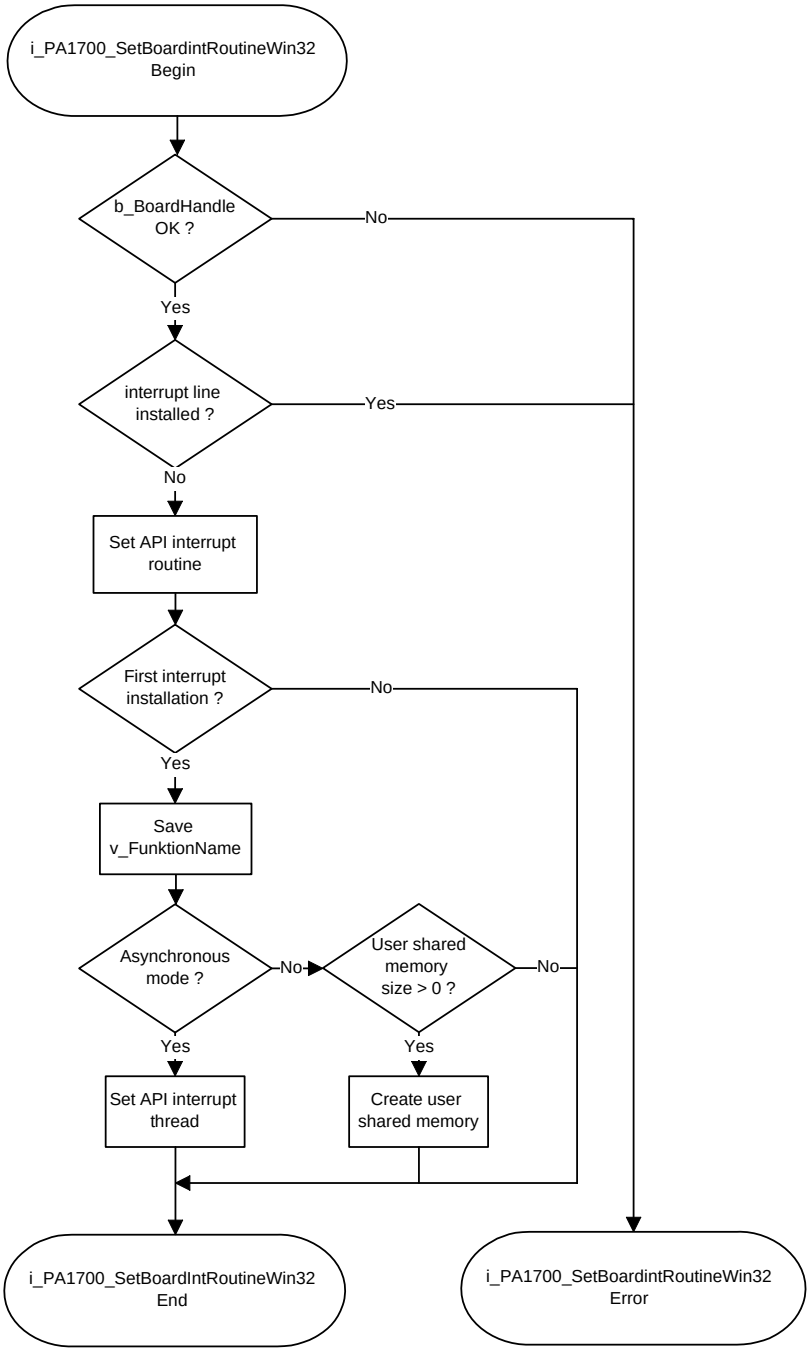
```
typedef struct
        {
        .
        .
        .
        }str_UserStruct;

    str_UserStruct * ps_UserSharedMemory;

void        v_FunctionName              (unsigned char   b_BoardHandle,
                                         unsigned char   b_InterruptMask,
                                         unsigned long  ul_CounterLatchValue,
                                         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_ PA1700_SetBoardIntRoutineWin32
                                        (b_BoardHandle,
                                         PA1700_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: Parameter b_UserCallingMode is wrong.

Input

b_BoardHandle	ul_UserSharedMemorySize
b_UserCallingMode	v_FunctionName

Function diagram



Output

ppv_UserSharedMemory

<Return Value>

5) i_PA1700_TestInterrupt (..)

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

Syntax:
<Return value> = i_PA1700_TestInterrupt (PBYTE pb_BoardHandle,
PBYTE pb_InterruptMask,
PULONG pul_CounterLatchValue)

Parameters:
- Input:
No input signal has to occur.
- Output:

PBYTE pb_BoardHandle	Handle of the board **PA 1700** which has generated the interrupt
PBYTE pb_InterruptMask	Mask of the events which have generated the interrupt.
PULONG pul_CounterLatchValue	The value of latched counter

Interrupt mask

pb_InterruptMask	Meaning
0000 000**1**	Hardware latch on the first register of hardware counter 0 (32-bit)
0000 00**1**0	Hardware latch on the second register of hardware counter 0 (32-bit)
0000 0**1**00	Hardware latch on the first register of hardware counter 1 (32-bit)
0000 **1**000	Hardware latch on the second register of hardware counter 1 (32-bit)
000**1** 0000	Hardware latch on the first register of hardware counter 2 (32-bit)
00**1**0 0000	Hardware latch on the second register of hardware counter 2 (32-bit)
0**1**00 0000	Timer interrupt

b_InterruptMask	**Source**	***ul_CounterLatchValue* value**
b_InterruptMask = 1	Hardware strobe on the first latch register of hardware counter 0	Latched value of the first latch register (32-bit)
b_InterruptMask = 2	Hardware strobe on the second latch register of hardware counter 0	Latched value of the second latch register (32-bit)
b_InterruptMask = 4	Hardware strobe on the first latch register of hardware counter 1	Latched value of the first latch register (32-bit)
b_InterruptMask = 8	Hardware strobe on the second latch register of hardware counter 1	Latched value of the second latch register (32-bit)
b_InterruptMask = 16	Hardware strobe on the first latch register of hardware counter 2	Latched value of the first latch register (32-bit)
b_InterruptMask = 32	Hardware strobe on the second latch register of hardware counter 2	Latched value of the second latch register (32-bit)

Task:
Verifies if a board **PA1700** has generated an interrupt. If yes, the function returns the board handle and the interrupt source.

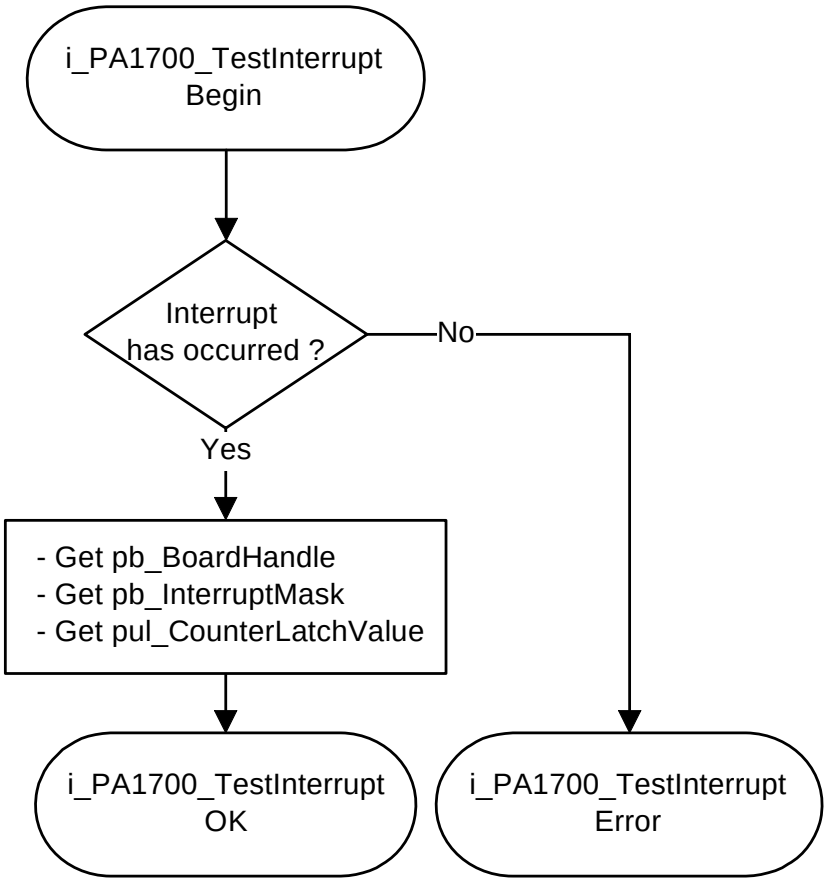
Calling convention:
ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned char   b_InterruptMask;
unsigned long   ul_CounterLatchValue;

i_ReturnValue = i_PA1700_TestInterrupt      (&b_BoardHandle,
                                             &b_InterruptMask,
                                             &ul_CounterLatchValue);
```

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

Input

Function diagram



Output

pb_BoardHandle pul_CounterLatchValue
pb_InterruptMask

<Return Value>

6) i_PA1700_ResetBoardIntRoutine (..)

Syntax:
<Return value> = i_PA1700_ResetBoardIntRoutine (BYTE b_BoardHandle)

Parameters:
- Input:
BYTE b_BoardHandle Handle of board **PA 1700**
- Output:
No output signal has occurred.

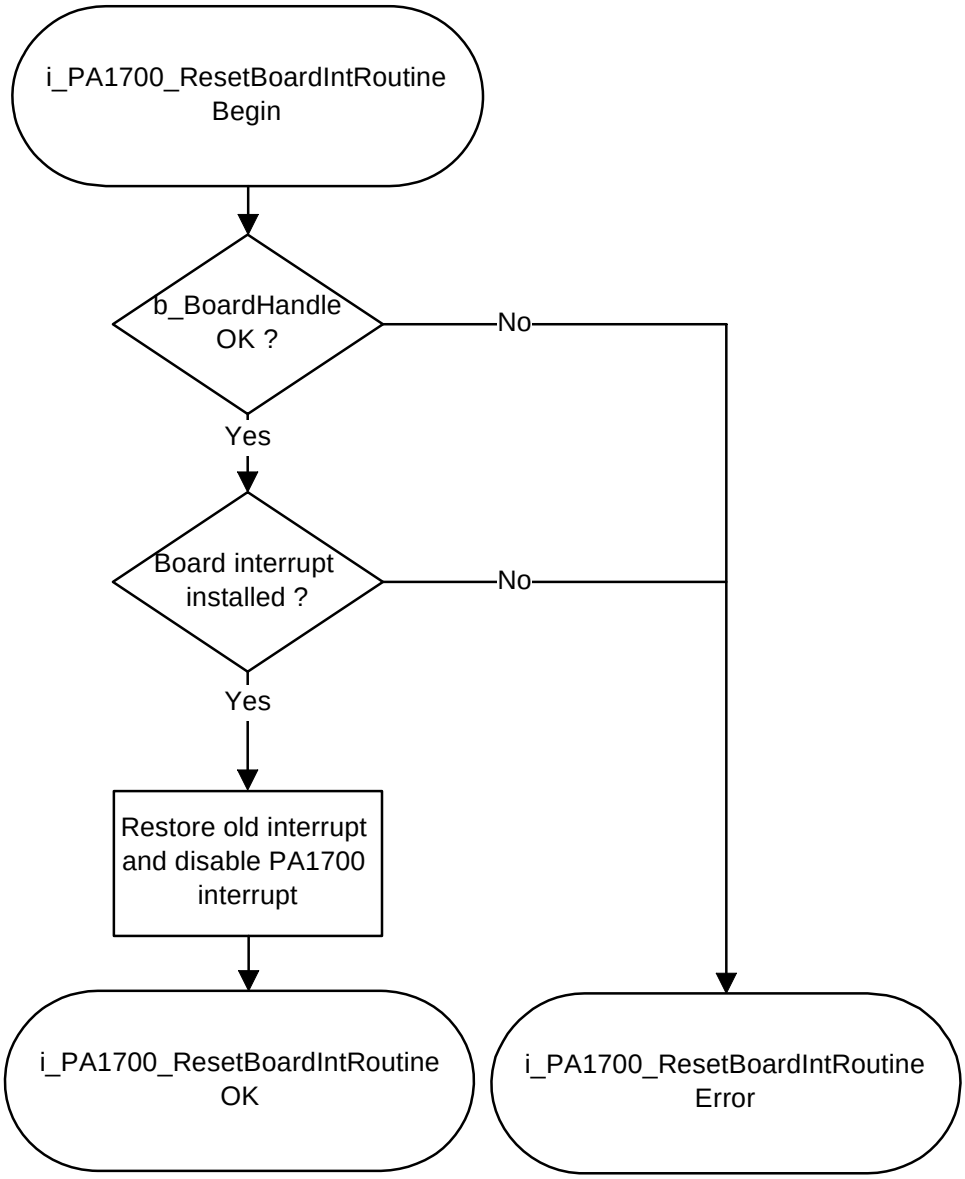
Task:
Stops the interrupt management of board **PA 1700.**
Deinstalls the user interrupt routine if the management of interrupts of all boards **PA 1700** is stopped.

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: No interurpt has been initialised with function "i_PA1700_SetBoardIntRoutineXXX"

Input

b_BoardHandle

Function diagram



Output

<Return Value>

3.3 Counter

3.3.1 Counter initialisation

1) i_PA1700_InitCounter(...)

Syntax:
<Return value> = i_PA1700_InitCounter

```
                    (BYTE   b_BoardHandle,
                     BYTE   b_HardCounterNbr,
                     BYTE   b_CounterRange,
                     BYTE   b_FirstCounterModus,
                     BYTE   b_FirstCounterOption,
                     BYTE   b_SecondCounterModus,
                     BYTE   b_SecondCounterOption)
```

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of the **PA 1700** board
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)
BYTE	b_CounterRange	Selection of the counter range. See table 3-2.
BYTE	b_FirstCounterModus	First counter operating mode. See table 3-3.
BYTE	b_FirstCounterOption	First counter option. See table 3-4, 3-5.
BYTE	b_SecondCounterModus	Second counter operating mode. See table 3-3.
BYTE	b_SecondCounterOption	Second counter option. See table 3-4, 3-5.

- Output:
No output signal has occurred.

Task:
Configures the counter operating mode of the selected hardware counter (*b_HardCounterNbr*).
You must call up this function before calling up any other function which gives access to the counter.

IMPORTANT!
If you have configured the hardware counter for two 16-bit counters, a mixed mode with a counter in quadruple/double/single mode and the other counter in direct mode is not possible!

Table 3-2: Counter range

Parameter	Passed value	Description
b_HardCounterNbr	PA1700_16BIT_COUNTER	The hardware counter is configured for two 16-bit counters. - *b_FirstCounterModus* and *b_FirstCounterOption* configure the first 16-bit counter. - *b_SecondCounterModus* and *b_SecondCounterOption* configure the second 16-bit counter.
b_HardCounterNbr	PA1700_32BIT_COUNTER	The hardware counter is configured for one 32-bit counter. - *b_FirstCounterModus* and *b_FirstCounterOption* configure the 32-bit counter. - *b_SecondCounterModus* and *b_SecondCounterOption* are not used.

Table 3-3: Counter operating mode

Parameter	Passed value	Description
b_FirstCounterModus or *b_SecondCounterModus*	PA1700_QUADRUPLE_MODE	In the quadruple mode, the edge analysis circuit generates a counting pulse of each edge of 2 phase-shifted signals.
B_FirstCounterModus or *b_SecondCounterModus*	PA1700_DOUBLE_MODE	Functions in the same way as the quadruple mode, except that only 2 of the 4 edges are analysed per period
b_FirstCounterModus or *b_SecondCounterModus*	PA1700_SIMPLE_MODE	Functions in the same way as the quadruple mode, except that only 1 of the 4 edges is analysed per period.
B_FirstCounterModus or *b_SecondCounterModus*	PA1700_DIRECT_MODE	In the direct mode both edge analysis circuits are inactive. The inputs A, B in the 32-bit mode or A, B and C, D in the 16-bit mode represent, each, one clock pulse gate circuit. Thereby frequency and pulse duration measurements can be performed.

IMPORTANT!
This option is only available if you have selected the quadruple, double or simple mode.

Table 3-4: Counter operating option for quadruple/double/simple mode

Parameter	Passed value	Description
b_FirstCounterOption or *b_SecondCounterOption*	PA1700_HYSTERESIS_ON	In both edge analysis circuits one hysteresis circuit is available. It suppresses each time the first counting pulse after a change of rotation.
b_FirstCounterOption or *b_SecondCounterOption*	PA1700_HYSTERESIS_OFF	The first counting pulse is not suppressed after a change of rotation.

IMPORTANT!
This option is only avaible if you have selected the direct mode.

Table 3-5: Counter operating option for direct mode

Parameter	Passed value	Description
B_FirstCounterOption or *b_SecondCounterOption*	PA1700_INCREMENT	The counter increments after each counter pulse
B_FirstCounterOption or *b_SecondCounterOption*	PA1700_DECREMENT	The counter decrements after each counter pulse

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
i_ReturnValue = i_PA1700_InitCounter
                                    (b_BoardHandle,
                                     0,
                                     PA1700_16BIT_COUNTER,
                                     PA1700_QUADRUPLE_MODE,
                                     PA1700_HYSTERESIS_ON,
                                     PA1700_QUADRUPLE_MODE,
                                     PA1700_HYSTERESIS_ON);
```

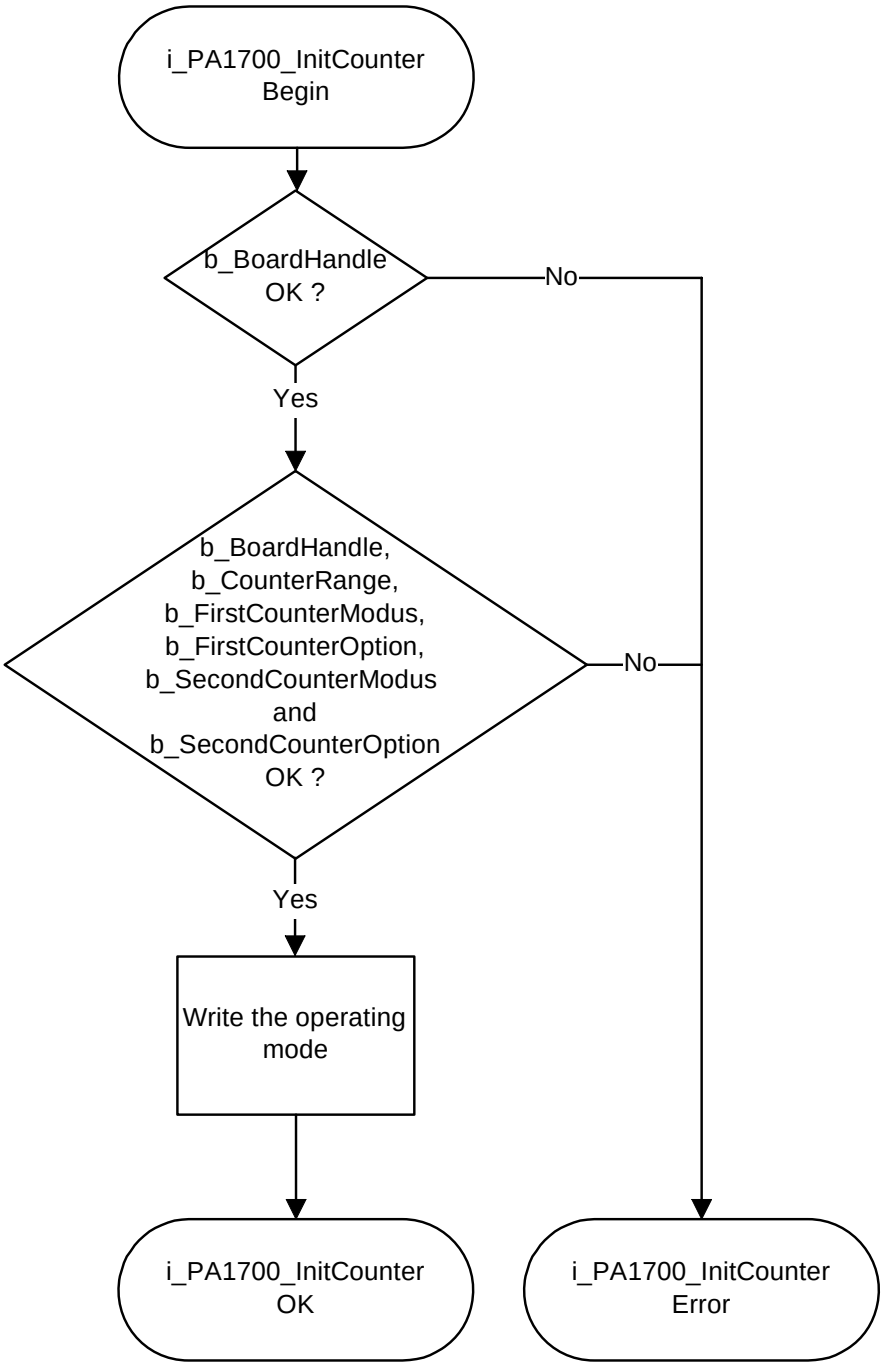
Return value:

0: No error
-1: The handle parameter of the board is wrong
-2 The selected hardware counter number is wrong
-3: The selected counter range is wrong. See table 3-2
-4: The selected operating mode of the first counter is wrong. See table 3-3
-5: The selected operating option of the first counter is wrong. See table 3-4
-6: The selected operating mode of the second counter is wrong. See table 3-3
-7: The selected operating option of the second counter is wrong. See table 3-4

Input

b_BoardHandle	b_FirstCounterOption
b_CounterRange	b_SecondCounterModus
b_FirstCounterModus	b_SecondCounterOption

Function



Output

<Return Value>

2) i_PA1700_CounterAutoTest (...)

Syntax:
<Return value> = i_PA1700_CounterAutoTest
(BYTE b_BoardHandle,
PBYTE pb_TestStatus)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of **PA 1700**

- Output:

PBYTE	pb_TestStatus	Automatic test of the counter. See table 3-6

Task:
This mode tests the component. All 8-bit counter chains are internally operated as downward counters. Independently from the external signals, all 8-bit counter chains are parallely decremented by each negative clock pulse edge of CLKX.

Table 3-6: Counter automatic test

***pb_TestStatus* mask**	**Error description**
0000	No error detected
000**1**	Error detected on counter 0
00**1**0	Error detected on counter 1
0**1**00	Error detected on counter 2

Calling convention:
ANSI C :

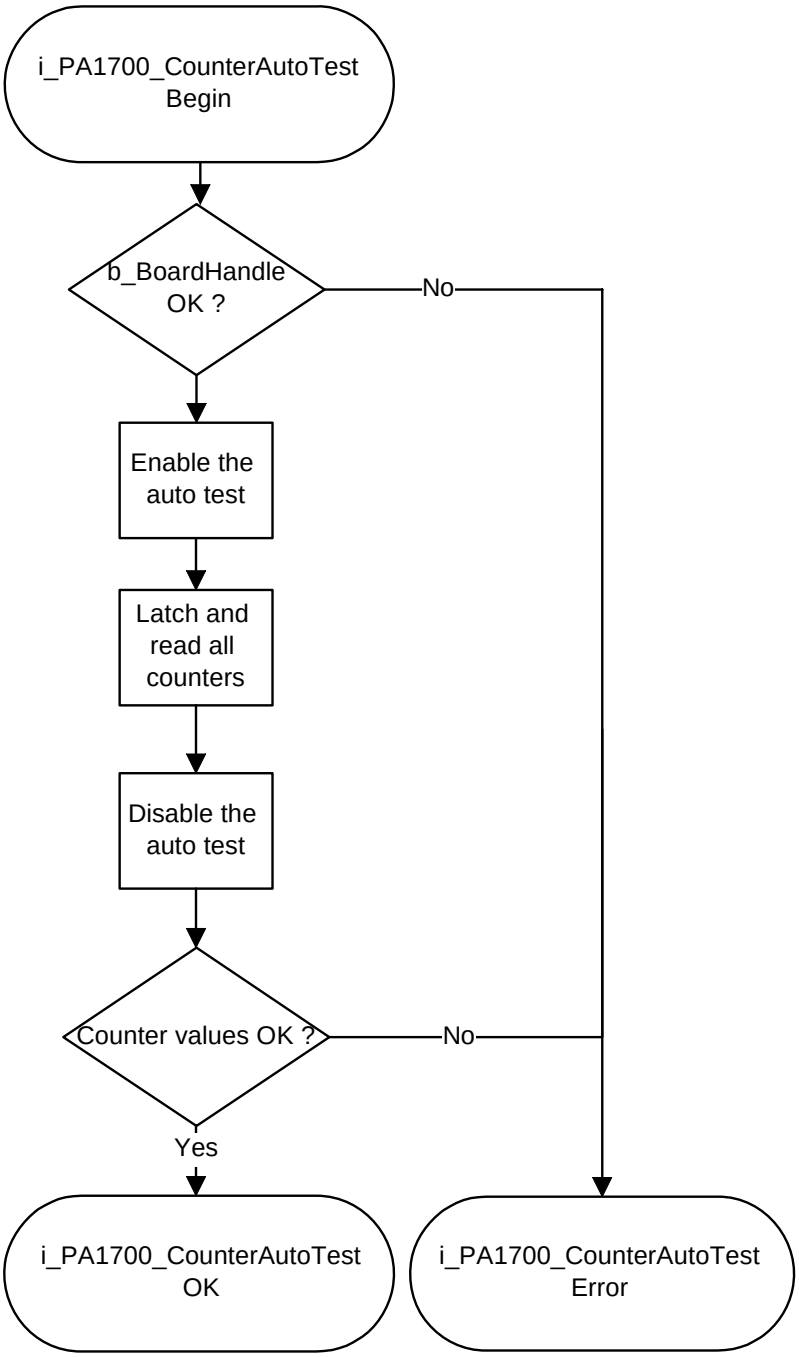
```
int             i_ReturnValue;
unsigned char   b_TestStatus;

i_ReturnValue = i_PA1700_CounterAutoTest    (b_BoardHandle,
                                             &b_TestStatus);
```

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

Input

b_BoardHandle

Function



Output

pb_TestStatus

<Return Value>

3) i_PA1700_ClearCounterValue (...)

Syntax:
<Return value> = i_PA1700_ClearCounterValue
(BYTE b_BoardHandle,
BYTE b_HardCounterNbr)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)

- Output:
No output signal has occurred.

Task:
Clears the counter value of the selected hardware counter (*b_HardCounterNbr*).

Calling convention:

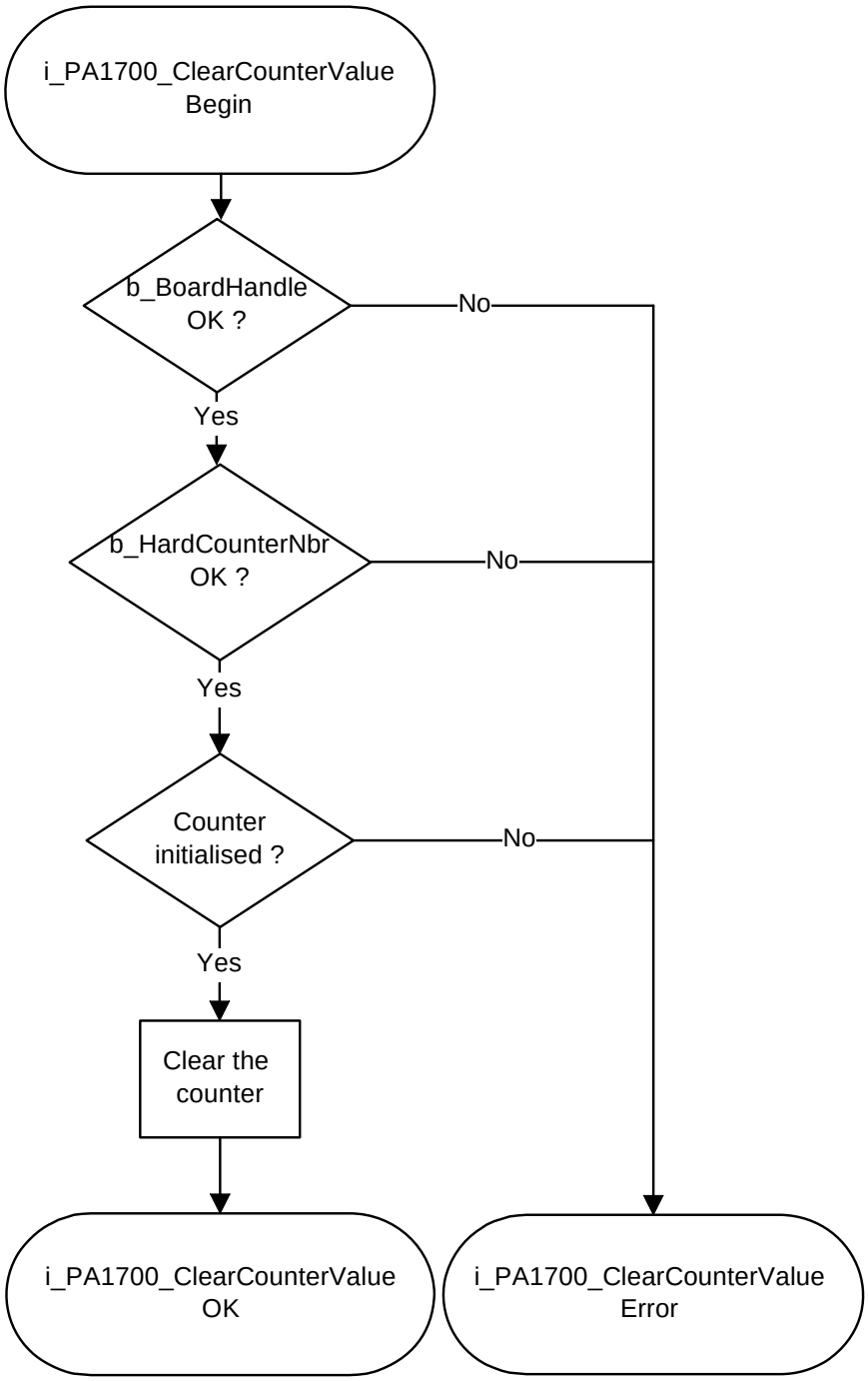
```
ANSI C :
int             i_ReturnValue;
unsigned char   b_BoardHandle;
i_ReturnValue = i_PA1700_ClearCounterValue      (b_BoardHandle,
                                                 0);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"

Input

b_BoardHandle

Function



Output

<Return Value>

4) i_PA1700_ClearAllCounterValue (...)

Syntax:
<Return value> = i_PA1700_ClearAllCounterValue
(BYTE b_BoardHandle)

Parameters:
- Input:
BYTE b_BoardHandle Handle of board **PA 1700**
- Output:
No output signal has occurred.

Task:
Clears all counter values.

Calling convention:
ANSI C :

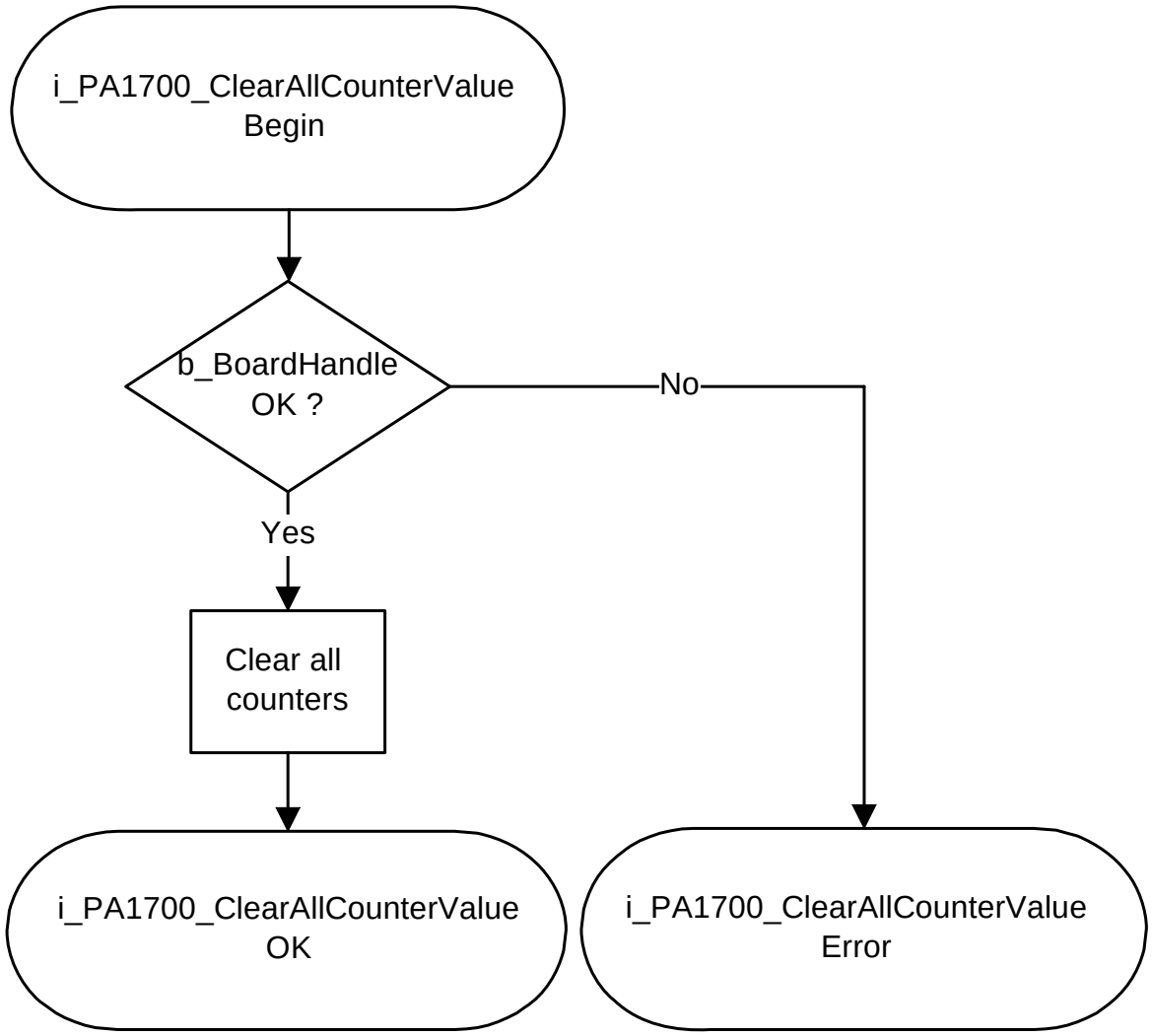
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_ClearAllCounterValue
                                        (b_BoardHandle);
```

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

Input

b_BoardHandle

Function



Output

<Return Value>

3.3.2 Reading the counter

1) i_PA1700_LatchCounter (...)

Syntax:

```
<Return value> = i_PA1700_LatchCounter
                        (BYTE    b_BoardHandle,
                         BYTE    b_HardCounterNbr,
                         BYTE    b_LatchReg)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)
BYTE	b_LatchReg	Selected latch register 0: for the first latch register 1: for the second latch register

- Output:

No output signal has occurred.

Task:

Latches the current value of the selected hardware counter (*b_HardCounterNbr*) into the selected latch register (*b_LatchReg*).

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
i_ReturnValue = i_PA1700_LatchCounter       (b_BoardHandle,
                                             0,
                                             0);
```

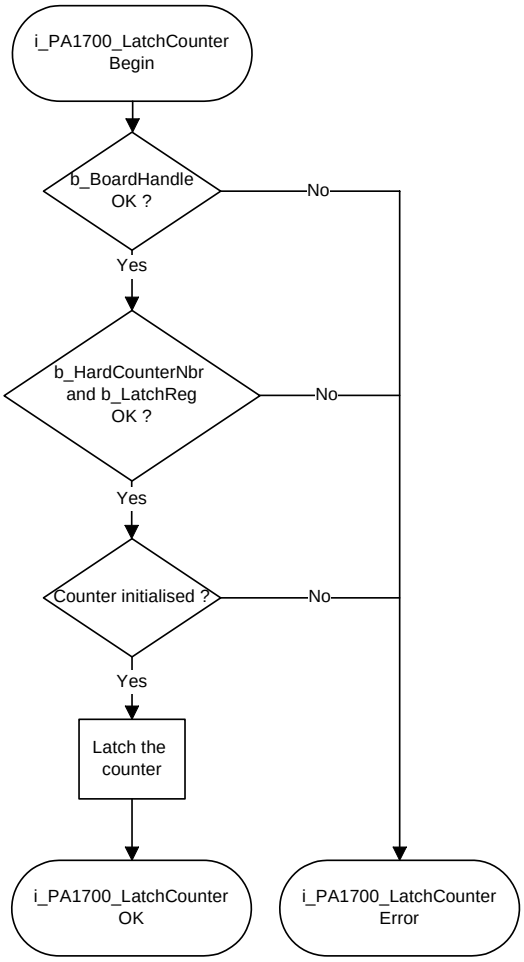
Return value:

0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"
-4: The selected latch register parameter is wrong

Input

b_BoardHandle b_LatchReg
b_HardCounterNbr

Function



Output

pb_TestStatus

<Return Value>

2) i_PA1700_ReadLatchRegisterStatus (...)

Syntax:

```
<Return value> = i_PA1700_ReadLatchRegisterStatus
                          (BYTE    b_BoardHandle,
                           BYTE    b_HardCounterNbr,
                           BYTE    b_LatchReg,
                           PBYTE   pb_LatchStatus)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)
BYTE	b_LatchReg	Selected latch register 0: for the first latch register 1: for the second latch register

- Output:

PBYTE	pb_LatchStatus	Latch register status. 0: No latch has occurred 1: A software latch has occurred 2: A hardware latch has occurred 3: A software and hardware latch have occurred

Task:

Reads the latch register status of the selected hardware counter (*b_HardCounterNbr*) and selected latch register (*b_LatchReg*).

Calling convention:

ANSI C :

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

i_ReturnValue = i_PA1700_LatchRegisterStatus  (b_BoardHandle, 0, 0,
                                                & b_LatchStatus);
```

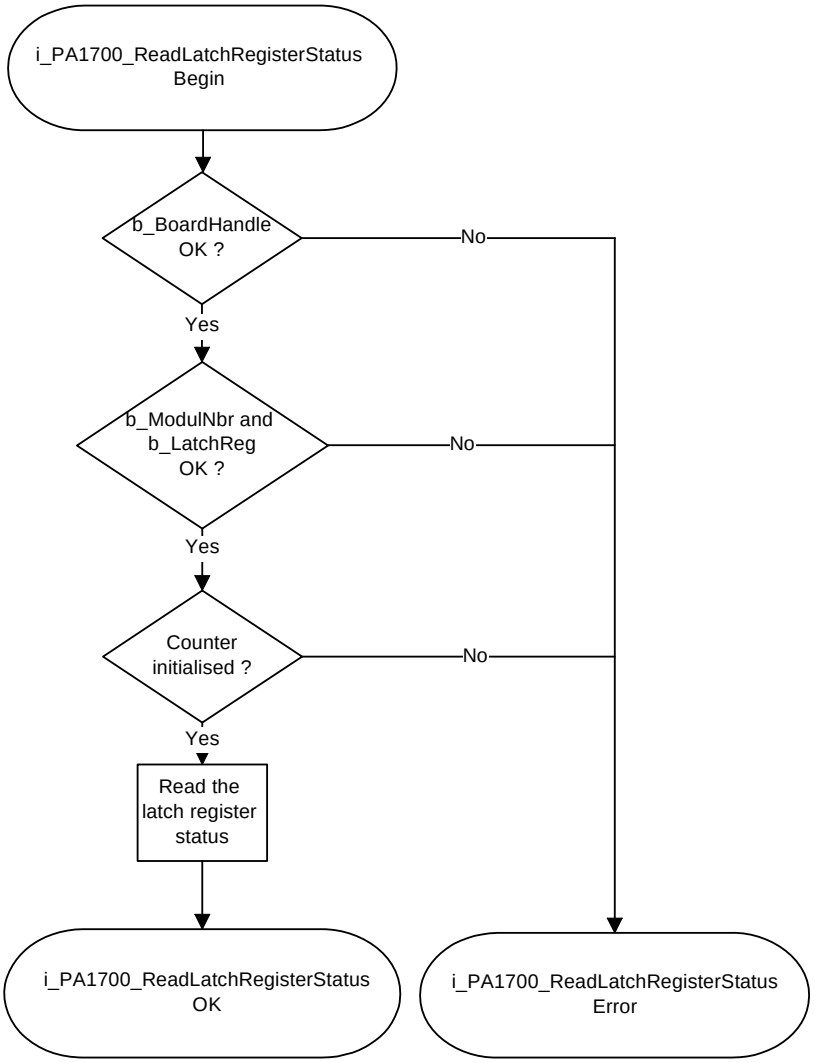
Return value:

 0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"
-4: The selected latch register parameter is wrong

Input

b_BoardHandle b_LatchReg
b_HardCounterNbr

Function



Output

pb_LatchStatus

<Return Value>

3) i_PA1700_ReadLatchRegisterValue (...)

Syntax:

```
<Return value> = i_PA1700_ReadLatchRegisterValue
                                    (BYTEb_BoardHandle,
                                     BYTE      b_HardCounterNbr,
                                     BYTE      b_LatchReg,
                                     PULONG    pul_LatchValue)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)
BYTE	b_LatchReg	Selected latch register 0: for the first latch register 1: for the second latch register

- Output:

PULONG	pul_LatchValue	Latch register value

Task:

Reads the latch register value of the selected hardware counter (*b_HardCounterNbr*) and selected latch register (*b_LatchReg*).

Calling convention:

ANSI C :

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

i_ReturnValue = i_PA1700_LatchCounter
                                        (b_BoardHandle,
                                         0,
                                         0,
                                         &ul_LatchValue);
```

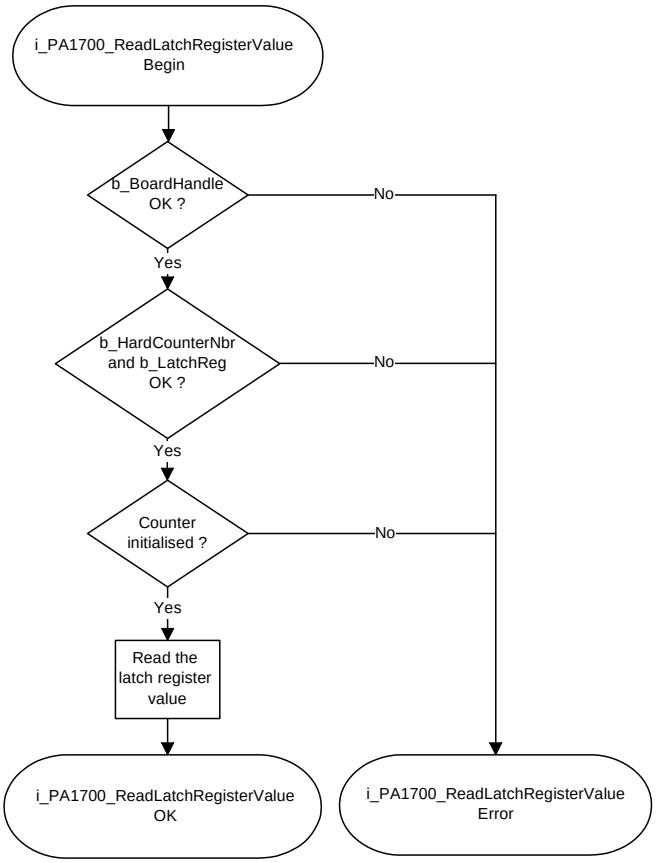
Return value:

 0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"
-4: The selected latch register parameter is wrong

Input

b_BoardHandle b_LatchReg
b_HardCounterNbr

Function



Output

pul_LatchValue

<Return Value>

4) i_PA1700_EnableLatchInterrupt (...)

Syntax:
<Return value> = i_PA1700_EnableLatchInterrupt
(BYTE b_BoardHandle,
BYTE b_LatchReg)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_LatchReg	Number of the latch register to enable the interrupt (0 or 1)

- Output:
No output signal has occurred.

Task:
Enables the latch interrupt of the selected latch register (*b_LatchReg*). Each software or hardware latch generates an interrupt.

Calling convention:
ANSI C :

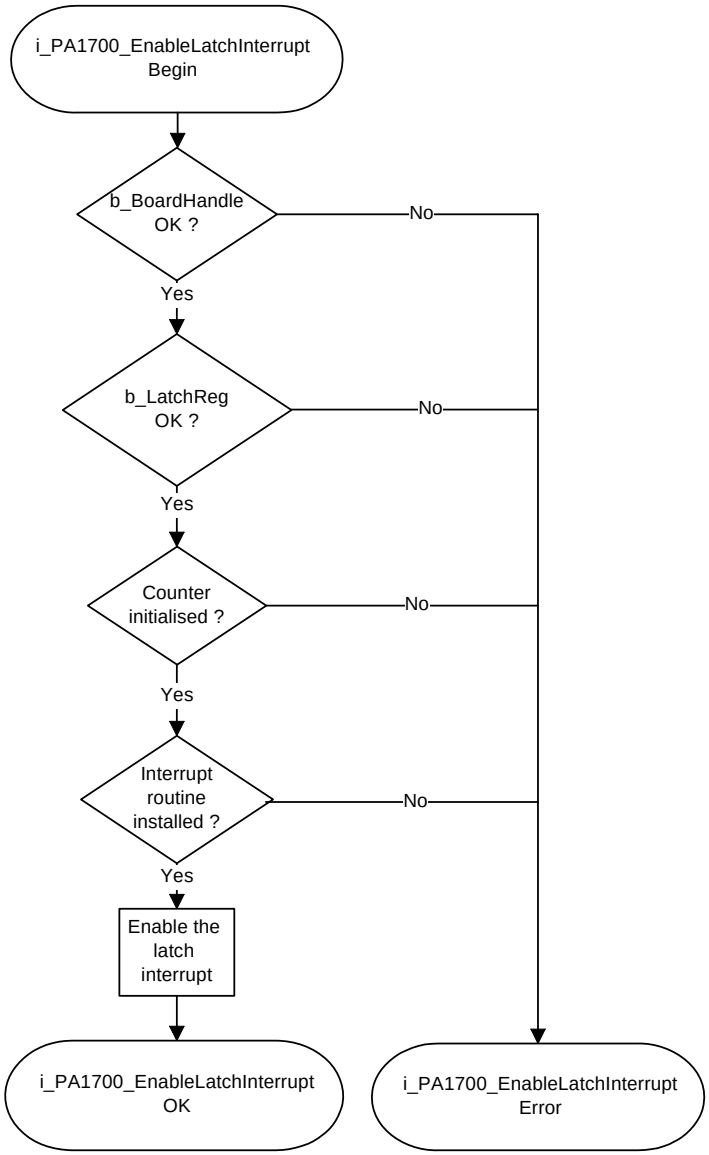
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_EnableLatchInterrupt   (b_BoardHandle, 0);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected latch register is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"
-4: Interrupt routine not installed. See function "i_PA1700_SetBoardIntRoutine"

Input

b_BoardHandle b_LatchReg

Function



Output

<Return Value>

5) i_PA1700_DisableLatchInterrupt (...)

Syntax:
<Return value> = i_PA1700_DisableLatchInterrupt
(BYTE b_BoardHandle,
BYTE b_LatchReg)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_LatchReg	Number of the latch register to disable the interrupt (0 or 1)

- Output:
No output signal has occurred.

Task:
Disables the latch interrupt of the selected latch register (*b_LatchReg*).

Calling convention:

ANSI C :

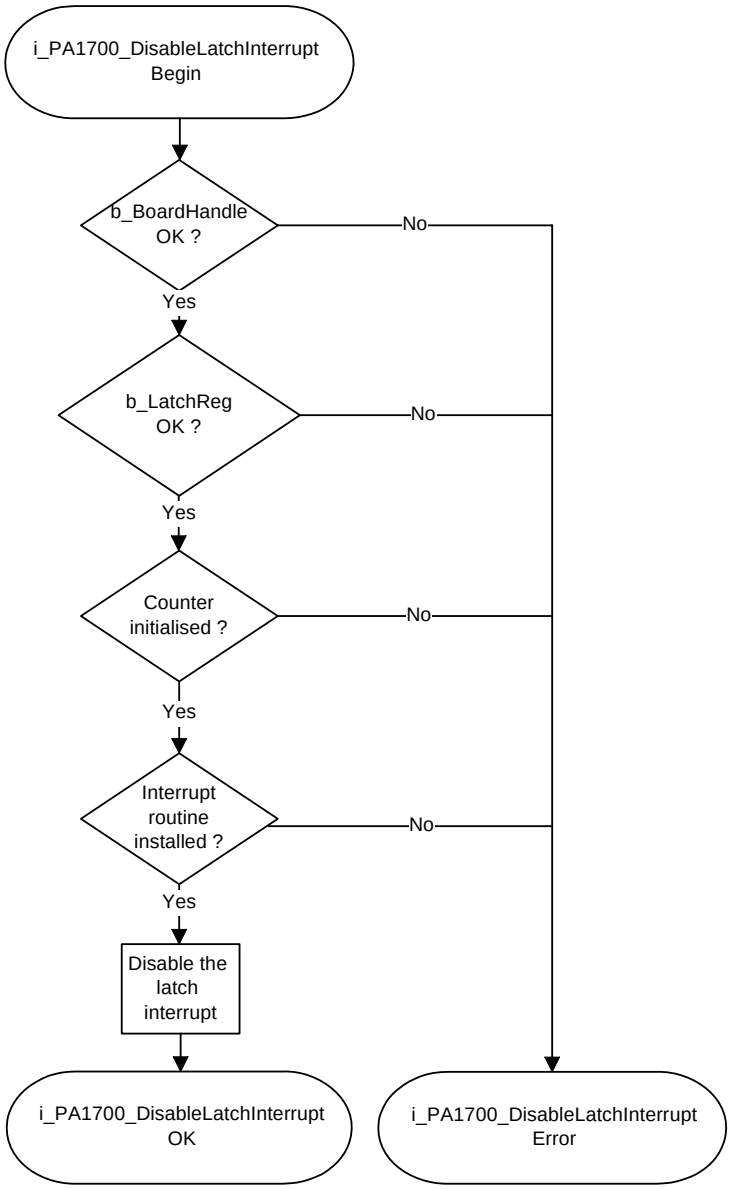
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_DisableLatchInterrupt
                                        (b_BoardHandle, 0);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected latch register is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"
-4: Interrupt routine not installed. See function "i_PA1700_SetBoardIntRoutine"

Input

b_BoardHandle b_LatchReg

Function



Output

<Return Value>

6) i_PA1700_Read16BitCounterValue (...)

Syntax:
<Return value> = i_PA1700_Read16BitCounterValue
(BYTE b_BoardHandle,
BYTE b_HardCounterNbr,
BYTE b_SelectedCounter,
PUINT pui_CounterValue)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)
BYTE	b_SelectedCounter	Selected 16-bit counter (0 or 1)

- Output:

PUINT	pui_CounterValue	16-bit counter value

Task:
Latches the 16-bit counter (*b_SelectedCounter*) of the selected hardware counter (*b_HardCounterNbr*) into the first latch register and returns the latched value.

Calling convention:
ANSI C :

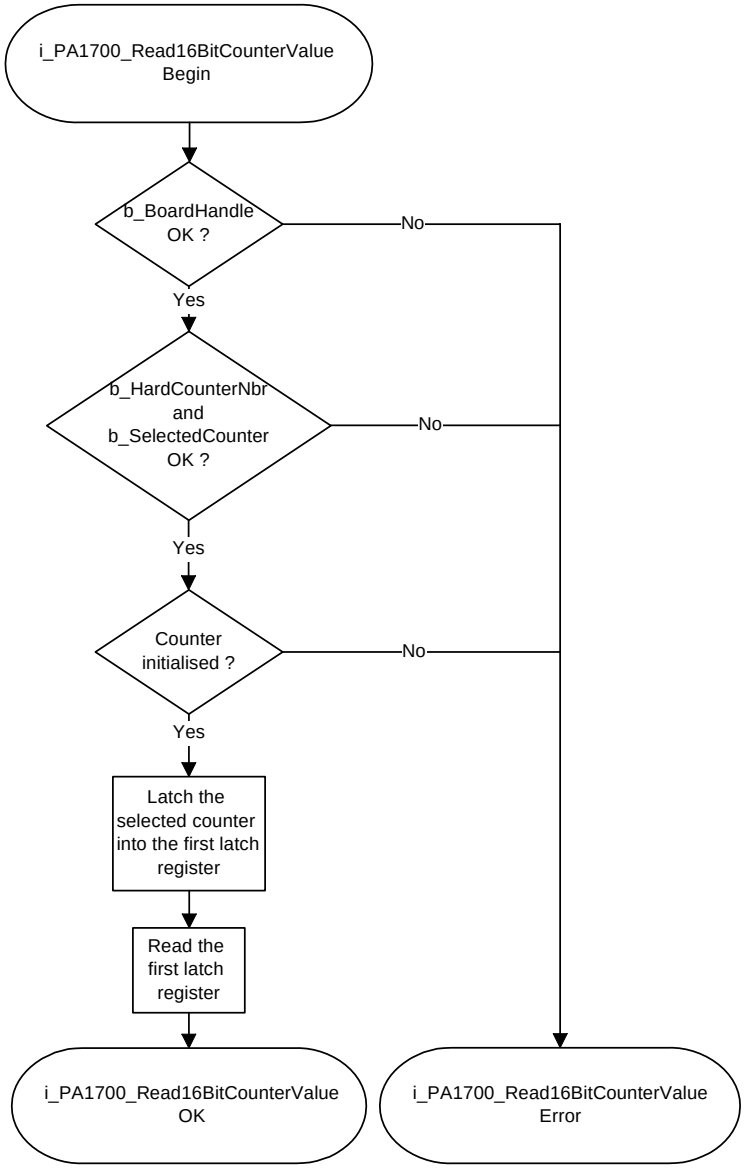
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned int    ui_CounterValue;

i_ReturnValue = i_PA1700_Read16BitCounterValue
                                        (b_BoardHandle,
                                         0,
                                         0,
                                         &ui_CounterValue);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"
-4: The selected 16-bit counter is wrong

Input

b_BoardHandle b_SelectedCounter
b_HardCounterNbr

Function



Output

pui_CounterValue

<Return Value>

7) i_PA1700_Read32BitCounterValue (...)

Syntax:
<Return value> = i_PA1700_Read32BitCounterValue
(BYTE b_BoardHandle,
BYTE b_HardCounterNbr,
PULONG pul_CounterValue)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)

- Output:

PULONG	pul_CounterValue	32-bit counter value

Task:
Latches the 32-bit counter of the selected hardware counter (*b_HardCounterNbr*) into the first latch register and returns the latched value.

Calling convention:

ANSI C :

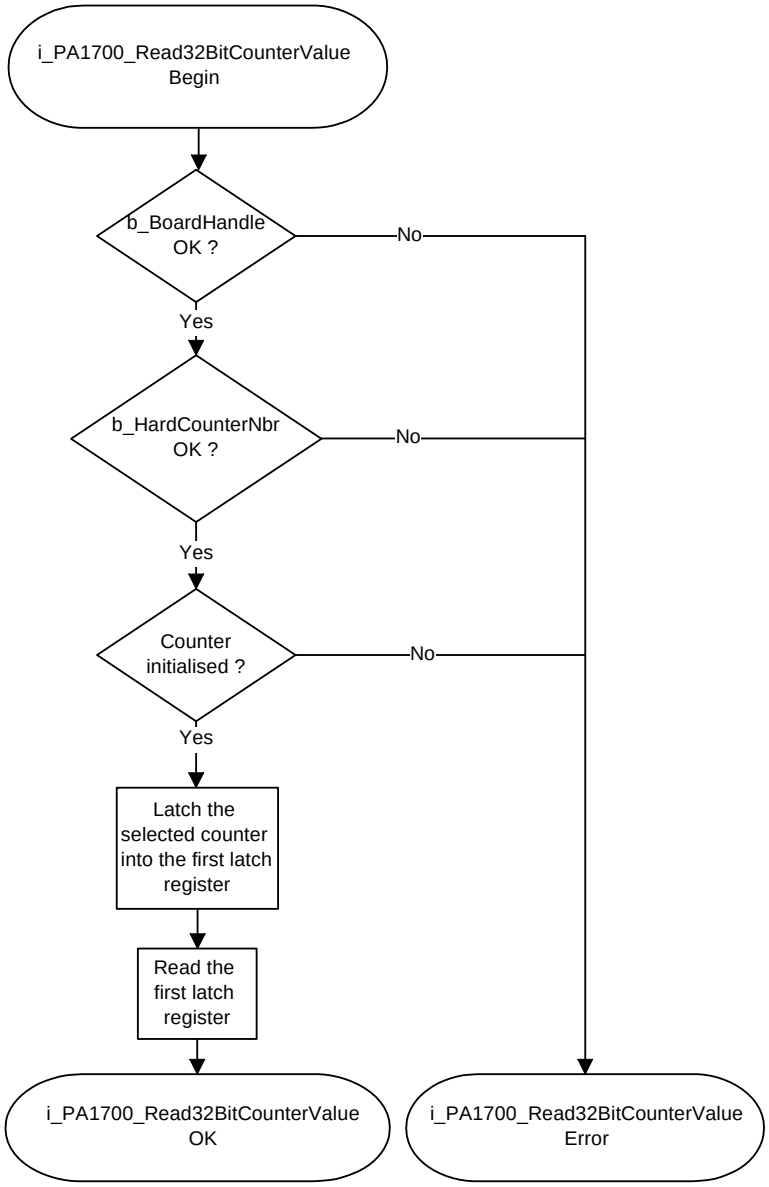
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned long   ul_CounterValue;

i_ReturnValue = i_PA1700_Read32BitCounterValue
                                        (b_BoardHandle,
                                         0,
                                         &ul_CounterValue);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"

Input

b_BoardHandle b_SelectedCounter
b_HardCounterNbr ui_WriteValue

Function



Output

<Return Value>

3.3.3 Writing in the counter

1) i_PA1700_Write16BitCounterValue (...)

Syntax:
< Return value > = i_PA1700_Write16BitCounterValue
(BYTE b_BoardHandle
BYTE b_HardCounterNbr,
BYTE b_SelectedCounter,
UINT ui_WriteValue)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)
BYTE	b_SelectedCounter	Selected 16-bit counter (0 or 1)
UINT	ui_WriteValue	16-bit value to be written

- Output:
No output signal has occurred.

Task:
Writes a 16-bit value (*ui_WriteValue*) into the selected 16-bit counter (*b_SelectedCounter*) of the selected hardware counter (*b_HardCounterNbr*).

Calling convention:

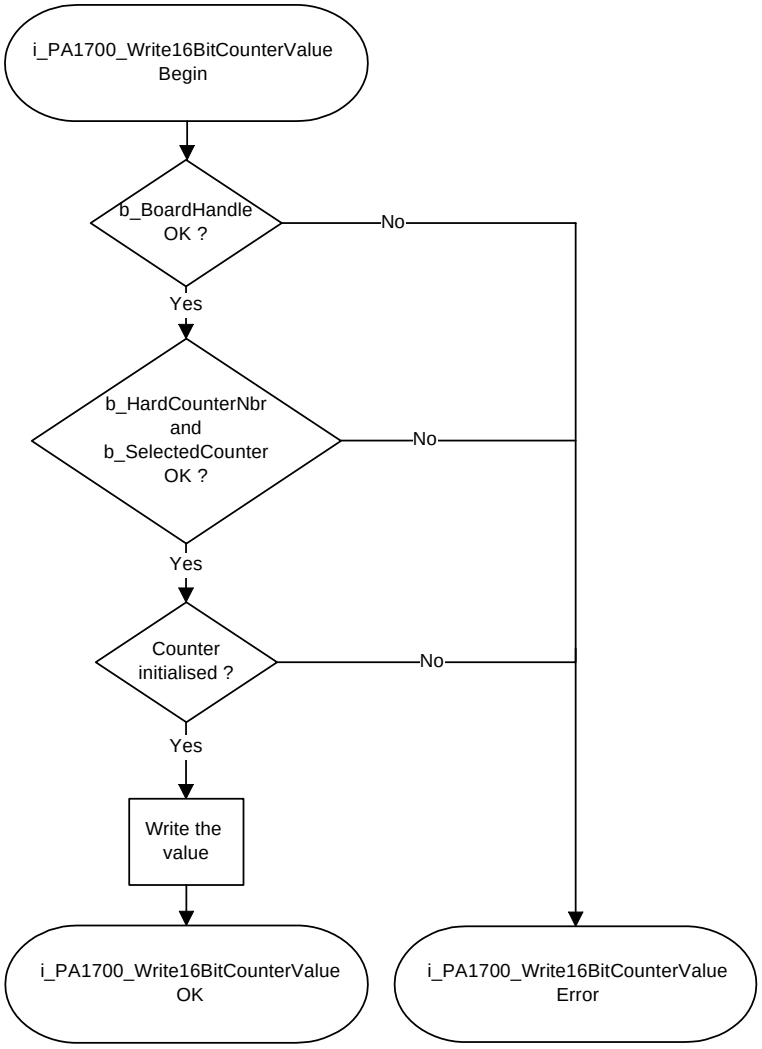
```
ANSI C :
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_Write16BitCounterValue
                                        (b_BoardHandle,
                                         0,
                                         0,
                                         2000);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"
-4: The selected 16-bit counter parameter is wrong

Input

b_BoardHandle	b_SelectedCounter
b_HardCounterNbr	pui_CounterValue

Function



Output

<Return Value>

2) i_PA1700_Write32BitCounterValue (...)

Syntax:
< Return value > = i_PA1700_Write32BitCounterValue
(BYTE b_BoardHandle
BYTE b_HardCounterNbr,
ULONG ul_WriteValue)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)
ULONG	ul_WriteValue	32-bit write value

- Output:
No output has occurred

Task:
Writes a 32-bit value (*ul_WriteValue*) into the 32-bit counter of the selected hardware counter (*b_HardCounterNbr*).

Calling convention:

ANSI C :

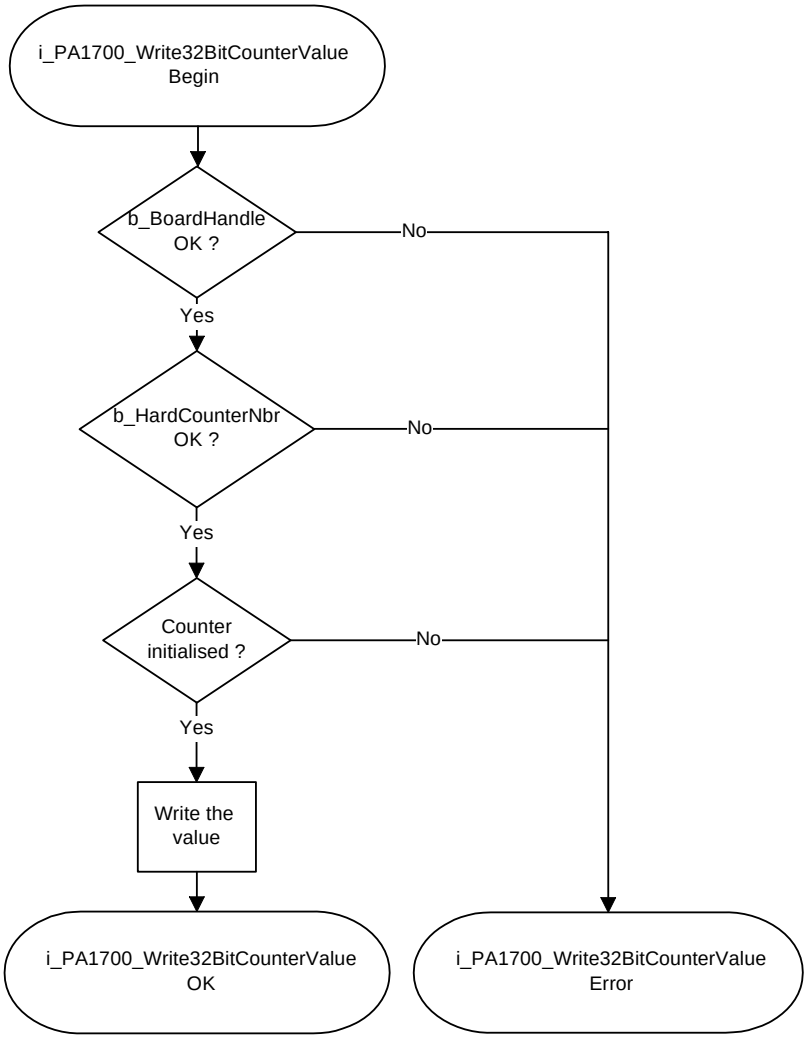
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_Write32BitCounterValue
                                        (b_BoardHandle,
                                         0,
                                         200000);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"

Input

b_BoardHandle ul_WriteValue
b_HardCounterNbr

Function



Output

<Return Value>

3.3.4 Index

1) i_PA1700_InitIndex (...)

Syntax:
<Return value> = i_PA1700_InitIndex
(BYTE b_BoardHandle,
BYTE b_HardCounterNbr,
BYTE b_AutoMode)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)
BYTE	b_AutoMode	Enables or disables the automatic index reset. PA1700_ENABLE: Enables the automatic mode PA1700_DISABLE: Disables the automatic mode

- Output:
No output signal has occurred.

Task:
Initialises the index corresponding to the selected hardware counter (*b_HardCounterNbr*).
If an INDEX flag occurs, you have the possibility to clear the 32-bit counter or to latch the current 32-bit value into the first latch register.
If you have enabled the automatic mode, each INDEX action is automatically cleared. If not, you must read the index status ("i_PA1700_ReadIndexStatus") after each INDEX action.

Calling convention:
ANSI C :

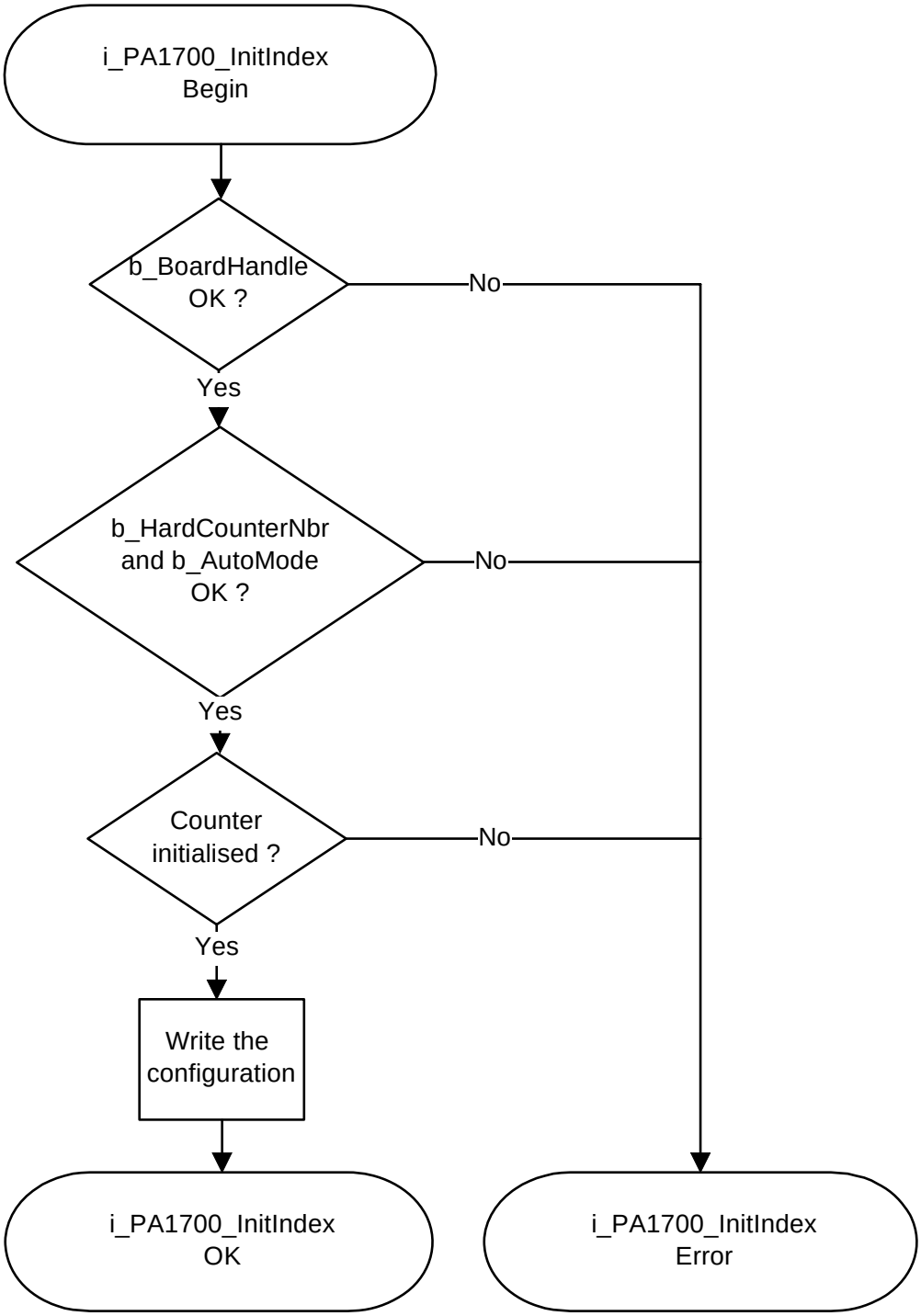
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_InitIndex
                                    (b_BoardHandle,
                                     0,
                                     PA1700_ENABLE);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"
-4: The auto mode parameter is wrong

Input

b_BoardHandle

Function



Output

<Return Value>

2) i_PA1700_EnableIndex (...)

Syntax:
<Return value> = i_PA1700_EnableIndex
(BYTE b_BoardHandle,
BYTE b_HardCounterNbr)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)

- Output:
No output signal has occurred.

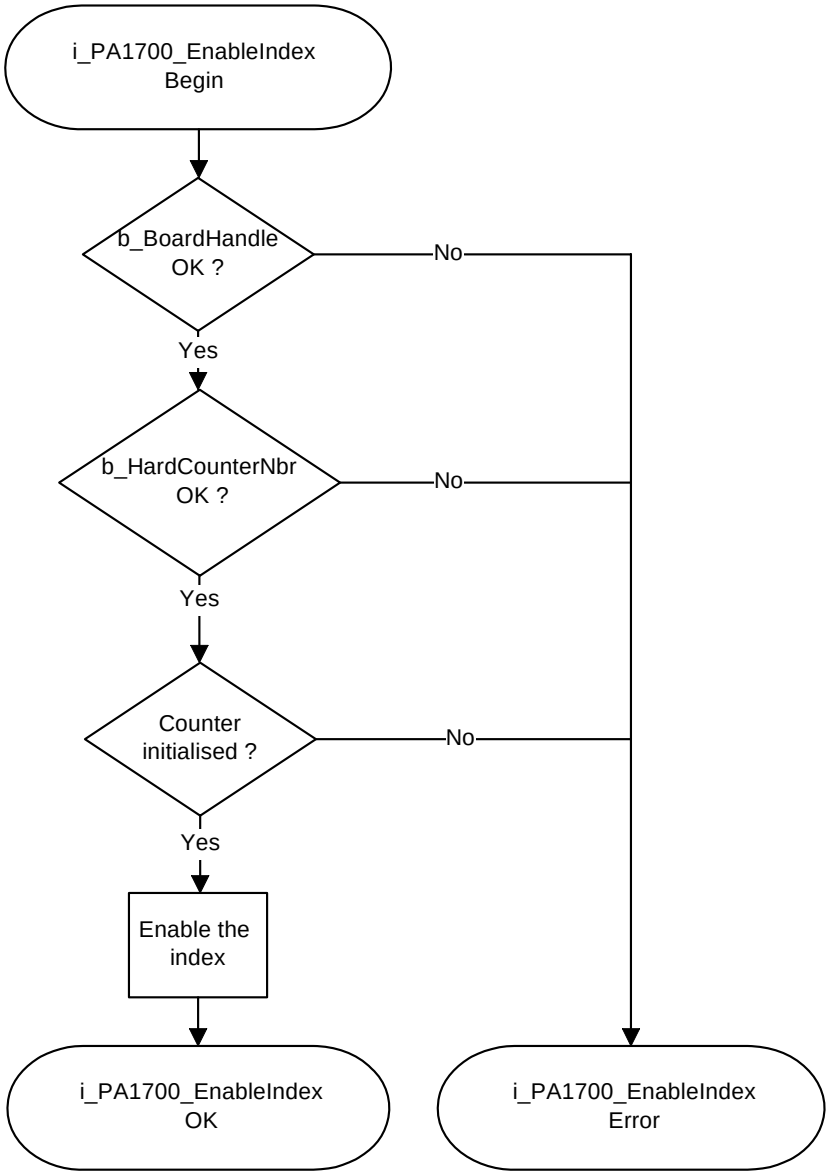
Task:
Enables the INDEX actions

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
i_ReturnValue = i_PA1700_EnableIndex    (b_BoardHandle, 0);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"
-4: Index not initialised. See function "i_PA1700_InitIndex"

Input
b_BoardHandle b_HardCounterNbr
Function

Output
<Return Value>

3) i_PA1700_DisableIndex (...)

Syntax:
<Return value> = i_PA1700_DisableIndex
(BYTE b_BoardHandle,
BYTE b_HardCounterNbr)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)

- Output:
No output signal has occurred.

Task:
Disables the INDEX actions.

Calling convention:
ANSI C :

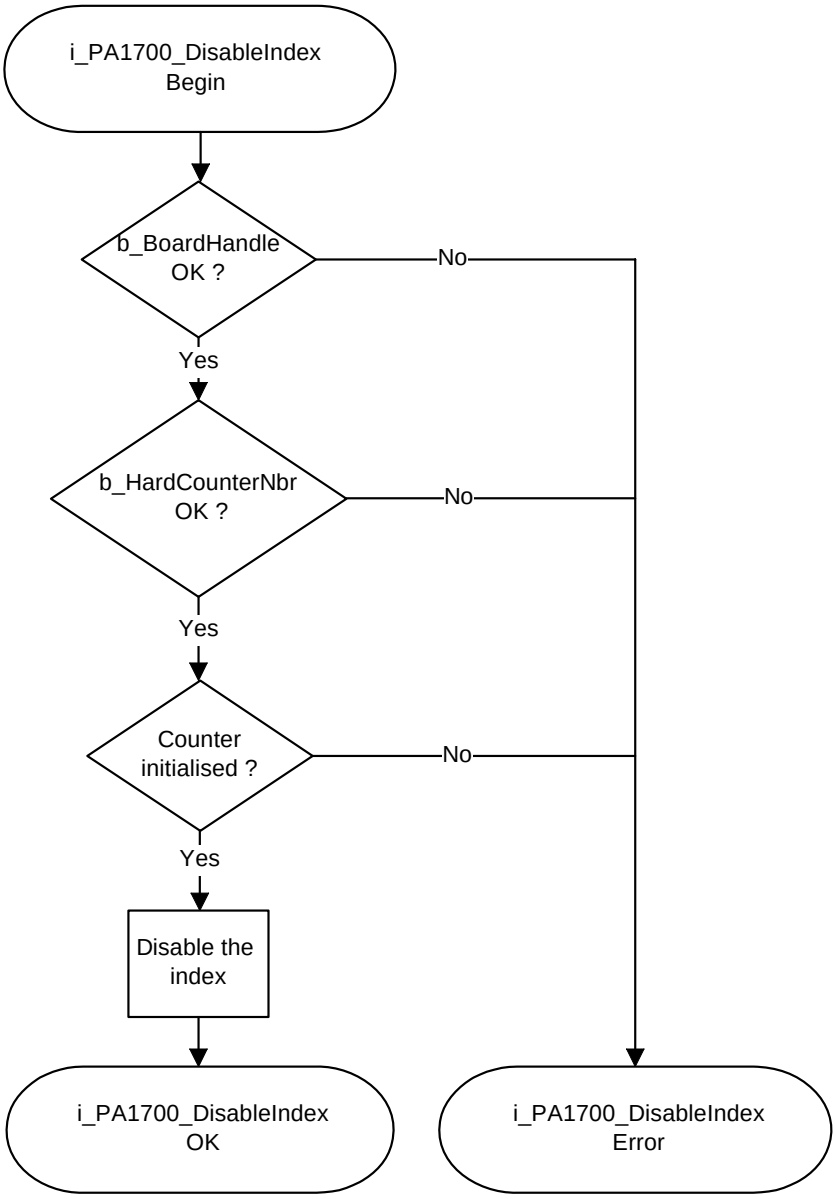
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_DisableIndex   (b_BoardHandle, 0);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"
-4: Index not initialised. See function "i_PA1700_InitIndex"

Input

b_BoardHandle b_HardCounterNbr

Function



Output

<Return Value>

4) i_PA1700_GetIndexStatus (...)

Syntax:
<Return value> = i_PA1700_GetIndexStatus
(BYTEb_BoardHandle,
BYTE b_HardCounterNbr,
PBYTE pb_IndexStatus)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)

- Output:

PBYTE	pb_IndexStatus	0: No INDEX has occurred 1: An INDEX has occurred

Task:
Returns the index status.

Calling convention:
ANSI C :

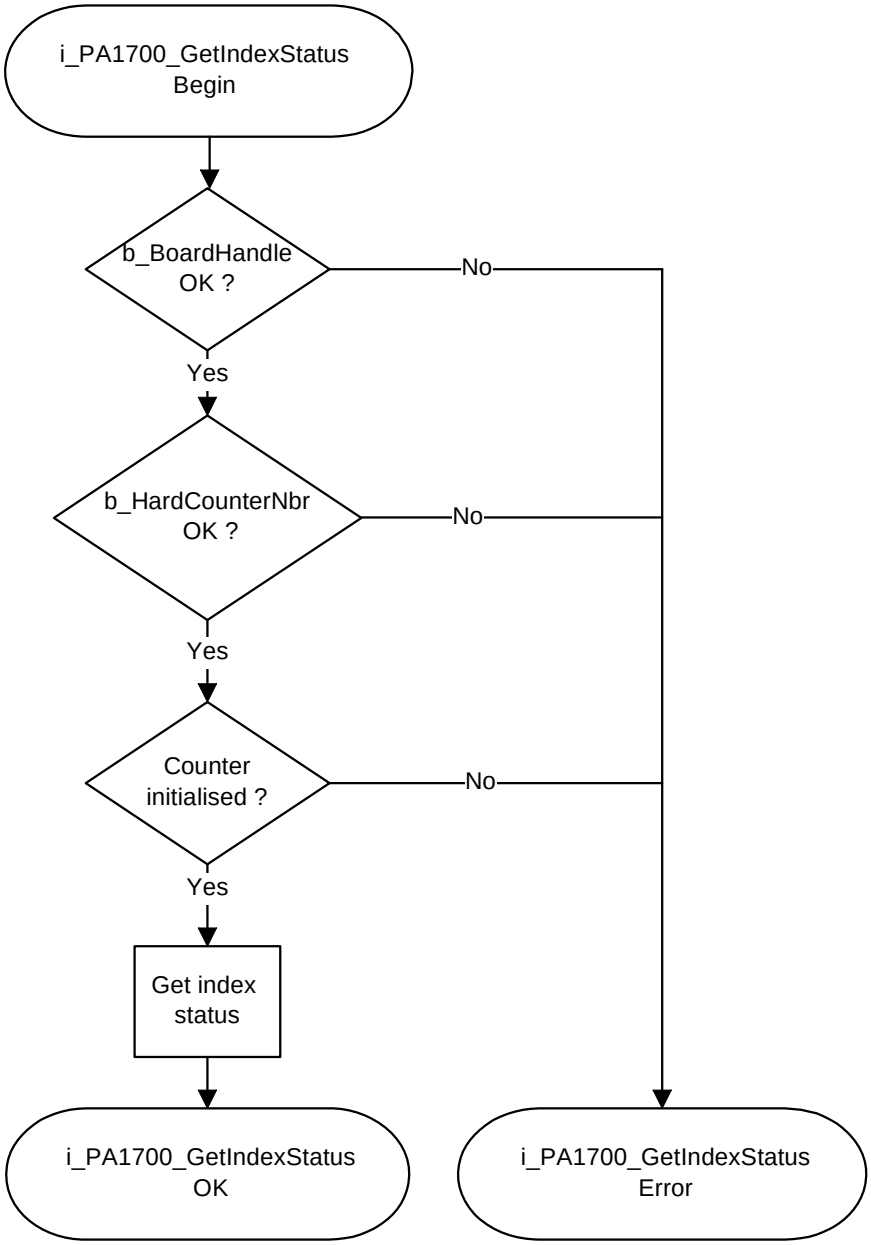
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned char   b_IndexStatus;

i_ReturnValue = i_PA1700_ GetIndexStatus
                                    (b_BoardHandle,
                                     0,
                                     &b_IndexStatus);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"
-4: Index not initialised. See function "i_PA1700_InitIndex"

Input

b_BoardHandle b_HardCounterNbr

Function



Output

pb_IndexStatus

<Return Value>

3.3.5 Reference

1) i_PA1700_GetReferenceStatus (...)

Syntax:
<Return value> = i_PA1700_GetReferenceStatus
(BYTE b_BoardHandle,
BYTE b_HardCounterNbr,
PBYTE pb_ReferenceStatus)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)

- Output:

PBYTE	pb_ReferenceStatus	0: No REFERENCE has occurred 1: A REFERENCE has occurred

Task:
Returns the reference status.

Calling convention:

ANSI C :

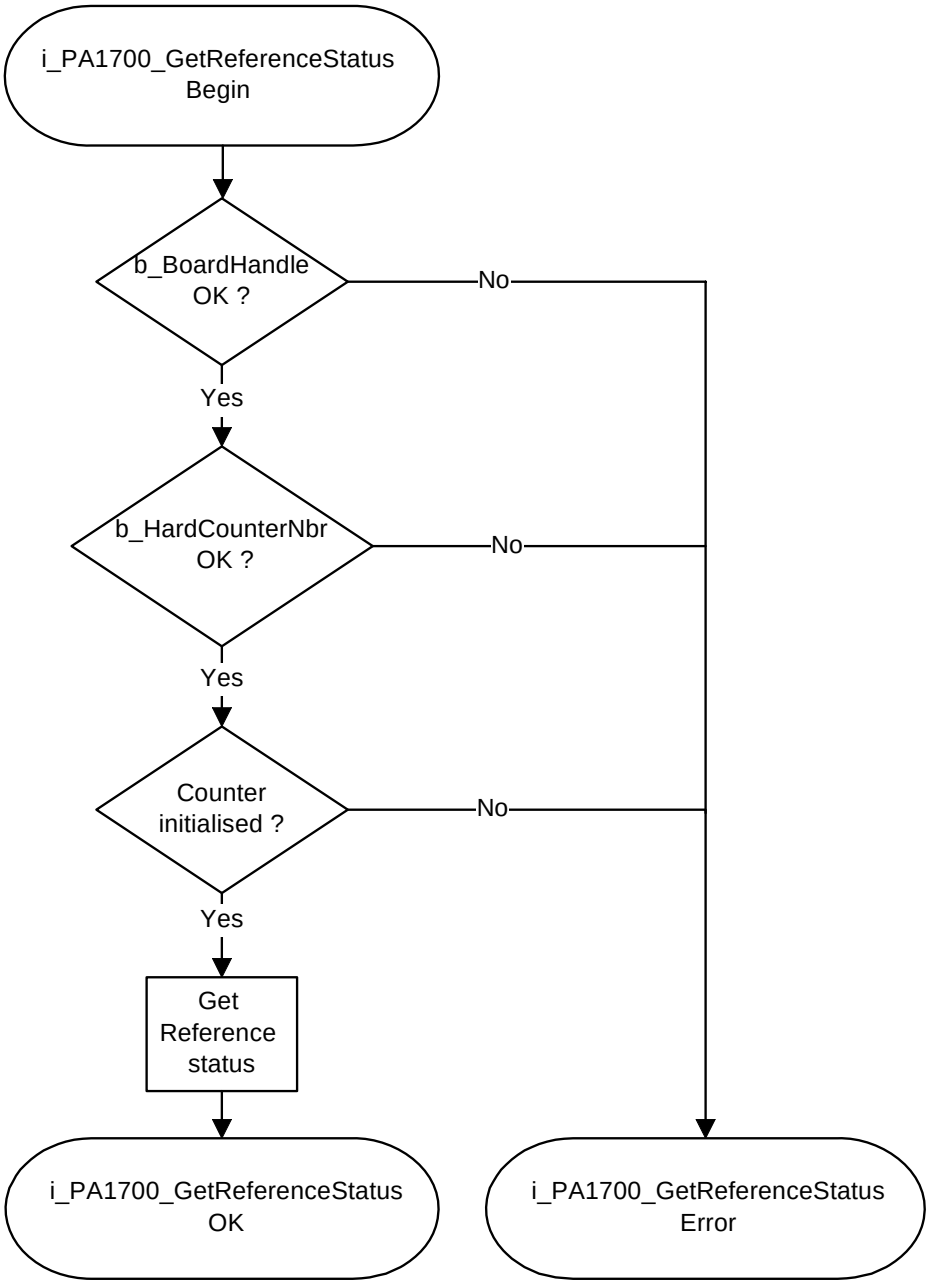
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned char   b_ReferenceStatus;

i_ReturnValue = i_PA1700_ GetReferenceStatus (b_BoardHandle,
                                              0,
                                              &b_ReferenceStatus);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"

Input

b_BoardHandle b_HardCounterNbr

Function



Output

pb_ReferenceStatus

<Return Value>

3.3.6 UAS, CB, U/D#

1) i_PA1700_GetUASStatus (...)

Syntax:
<Return value> = i_PA1700_GetUASStatus
(BYTEb_BoardHandle,
BYTE b_HardCounterNbr,
PBYTE pb_UASStatus)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)

- Output:

PBYTE	pb_UASStatus	0: UAS is low "0" 1: UAS is high "1"

Task:
Returns the UAS status

Calling convention:

ANSI C :

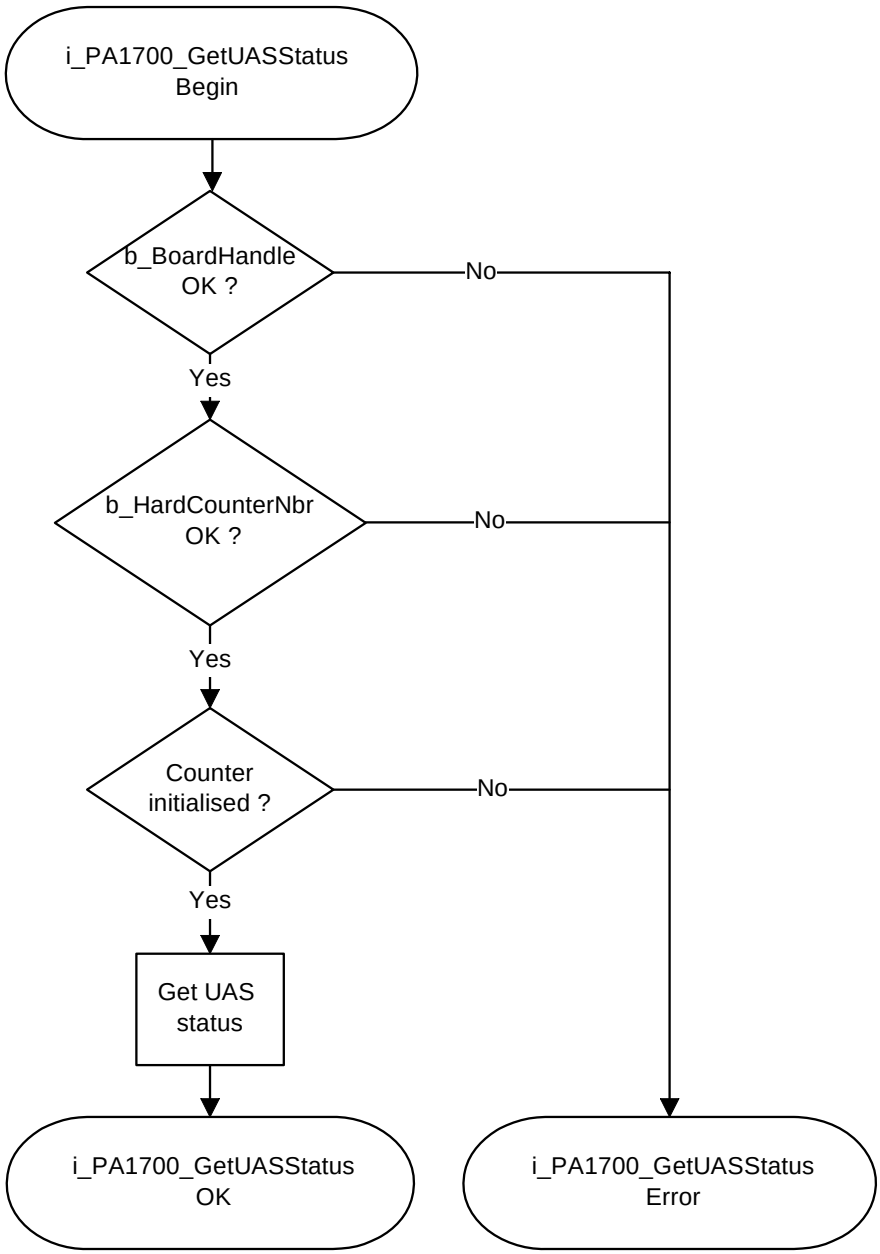
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned char   b_UASStatus;

i_ReturnValue = i_PA1700_ GetUASStatus
                                (b_BoardHandle,
                                 0,
                                 &b_UASStatus);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"

Input

b_BoardHandle b_HardCounterNbr

Function



Output

pb_UASStatus

<Return Value>

2) i_PA1700_GetCBStatus (...)

Syntax:
<Return value> = i_PA1700_GetCBStatus
(BYTEb_BoardHandle,
BYTE b_HardCounterNbr,
PBYTE pb_CBStatus)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)

- Output:

PBYTE	pb_CBStatus	0: No counter overflow 1: Counter overflow

Task:
Returns the counter overflow status.

Calling convention:

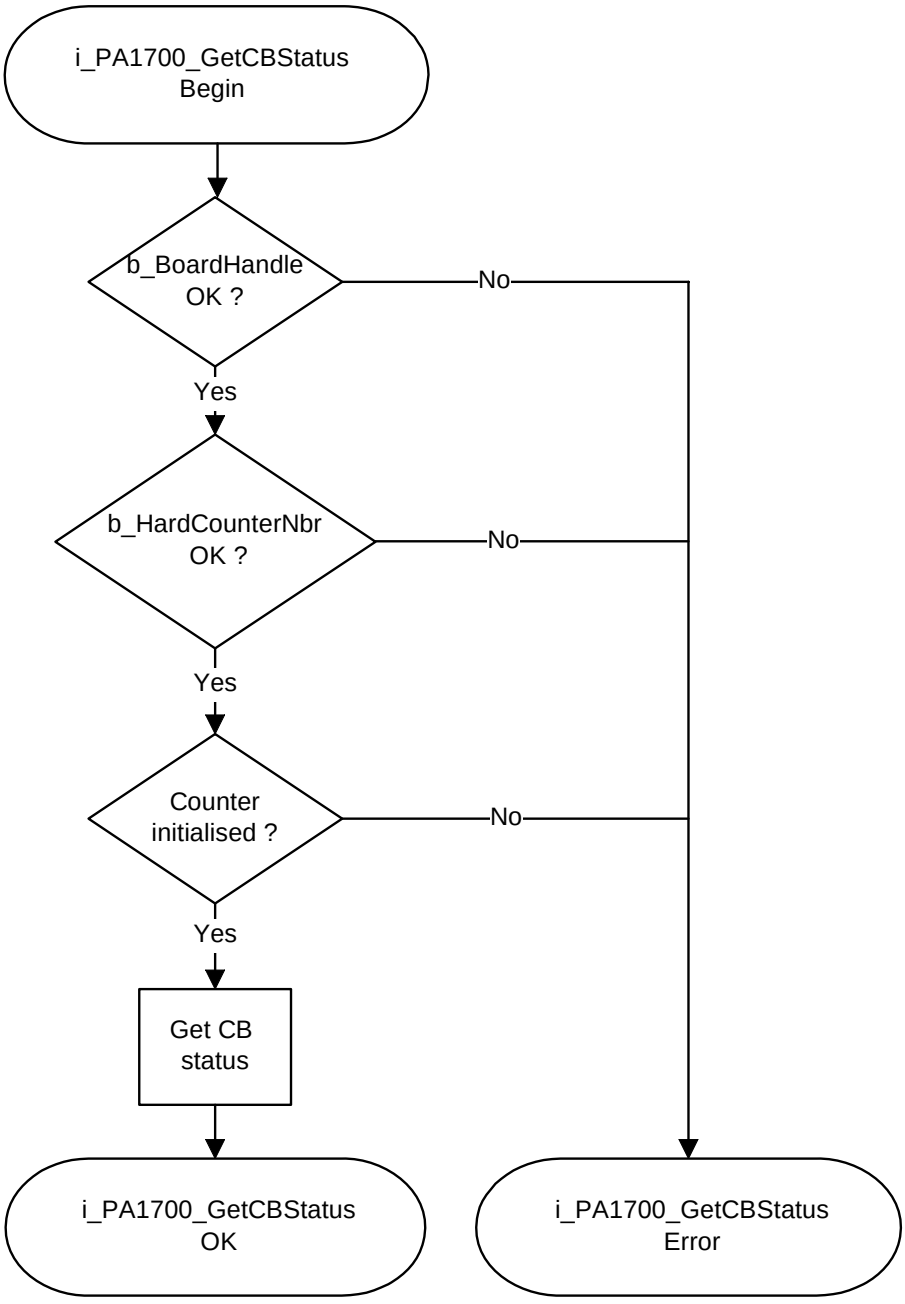
```
ANSI C :
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned char   b_CBStatus;
i_ReturnValue = i_PA1700_ GetCBStatus   (b_BoardHandle,
                                         0,
                                         &b_CBStatus);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"

Input

b_BoardHandle b_HardCounterNbr

Function



Output

pb_CBStatus

<Return Value>

3) i_PA1700_EnableCBInterrupt (...)

Syntax:
<Return value> = i_PA1700_EnableCBInterrupt (BYTE b_BoardHandle)

Parameter:
- Input:
BYTE b_BoardHandle Handle of board **PA1700**
- Output:
No output signal has occurred.

Task:
Enables the counter-overflow interrupt for each 32-bit counter

Calling convention:
ANSI C :

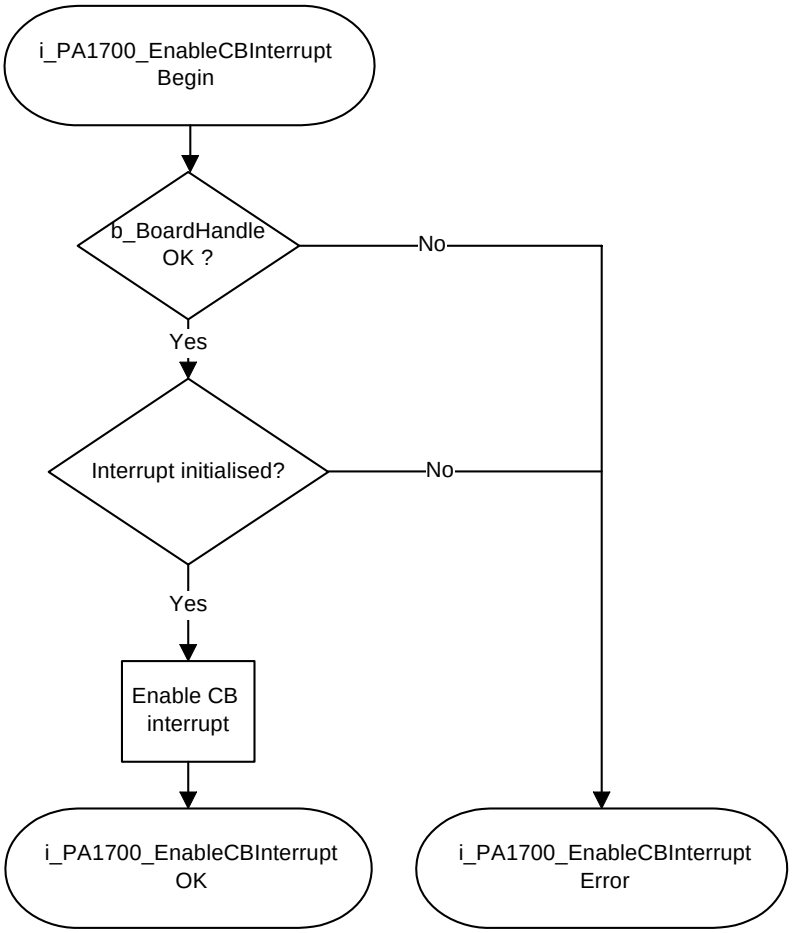
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_EnableCBInterrupt (b_BoardHandle);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: Interrupt function not initialised. See function "i_PA1700_SetBoardIntRoutineX"

Input

b_BoardHandle

Function



Output

<Return Value>

4) i_PA1700_DisableCBInterrupt (...)

Syntax:
<Return value> = i_PA1700_DisableCBInterrupt (BYTE b_BoardHandle)

Parameter:
- Input:
BYTE b_BoardHandle Handle of board **PA1700**
- Output:
No output signal has occurred.

Task:
Disables the counter-overflow interrupt for each 32-bit counter.

Calling convention:
ANSI C :

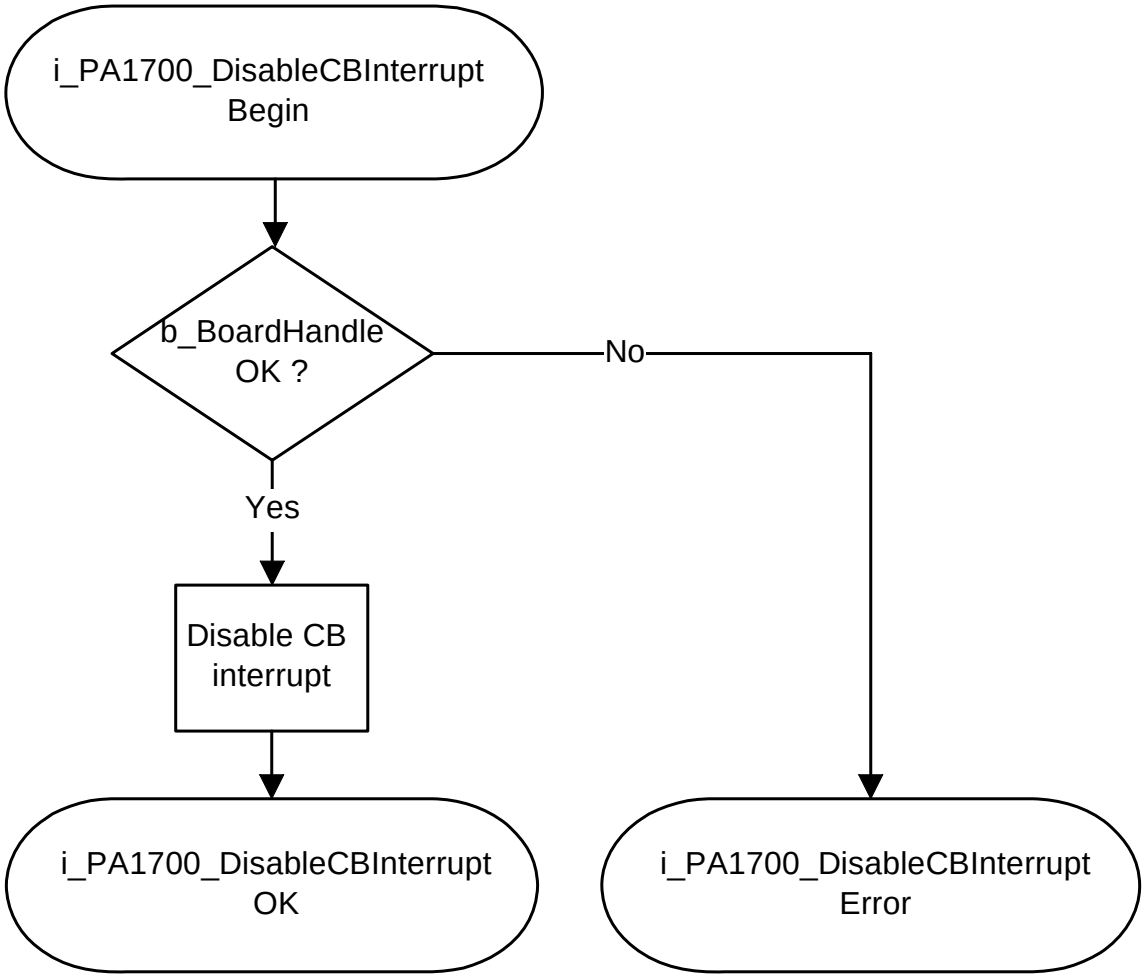
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_DisableCBInterrupt (b_BoardHandle);
```

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

Input

b_BoardHandle

Function



Output

<Return Value>

5) i_PA1700_GetUDStatus (...)

Syntax:
<Return value> = i_PA1700_GetUDStatus
(BYTEb_BoardHandle,
BYTE b_HardCounterNbr,
PBYTE pb_UDStatus)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)

- Output:

PBYTE	pb_UDStatus	0: Counter runs downwards in the selected mode 1: Counter runs upwards in the selected mode

Task:
Returns the counter status.

Calling convention:
ANSI C :

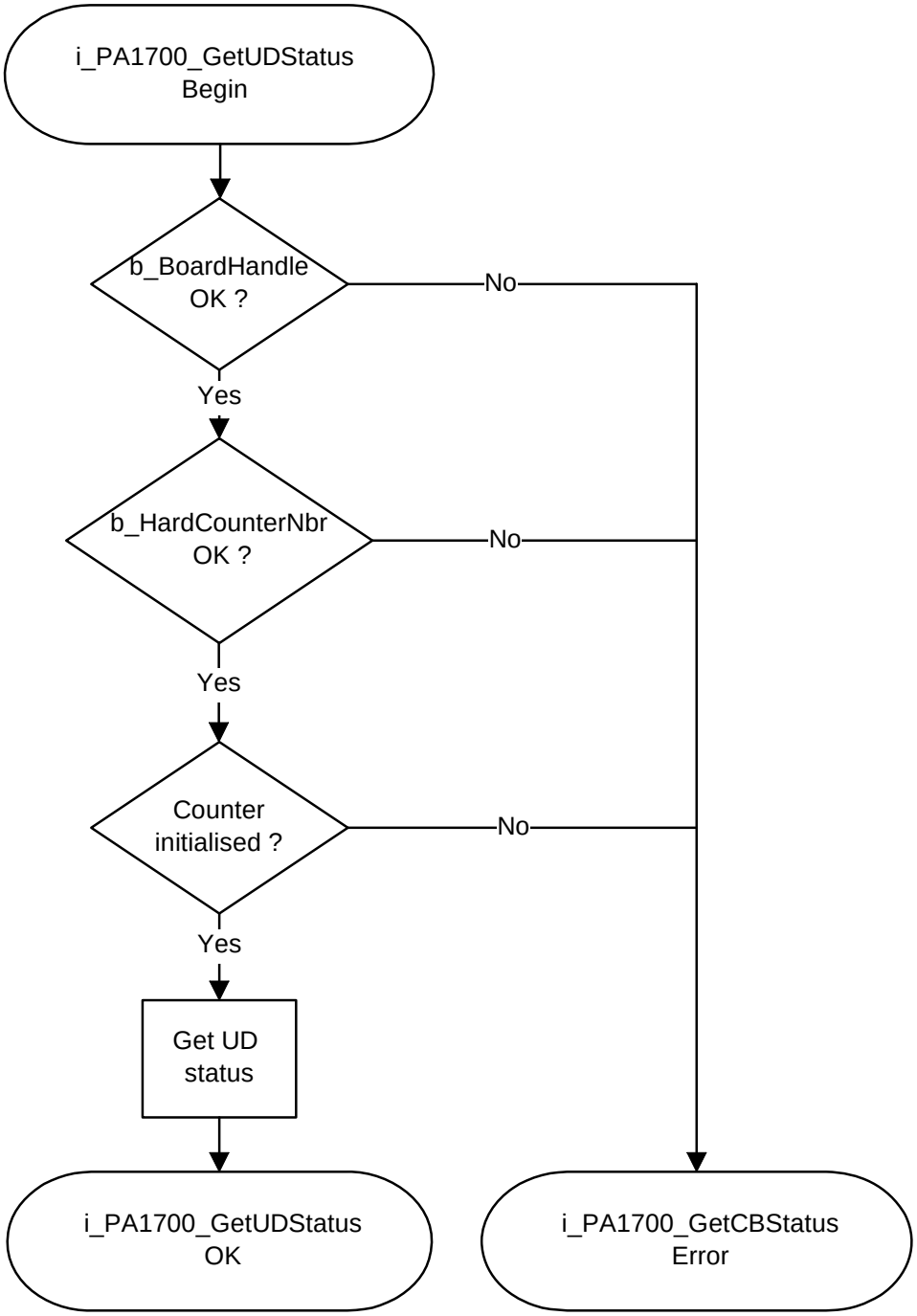
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned char   b_UDStatus;

i_ReturnValue = i_PA1700_ GetCBStatus   (b_BoardHandle,
                                         0,
                                         &b_UDStatus);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: The selected hardware counter number is wrong
-3: Counter not initialised. See function "i_PA1700_InitCounter"

Input

b_BoardHandle b_HardCounterNbr

Function



Output

pb_UDStatus

<Return Value>

3.4 Timer

3.4.1 Timer initialisation

1) i_PA1700_InitTimer (...)

Syntax:

```
<Return value> = i_PA1700_InitTimer   (BYTEb_BoardHandle,
                                       BYTE      b_TimerNbr,
                                       BYTE      b_TimerMode,
                                       UINT      ui_ReloadValue)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_TimerNbr	Number of the timer to be configured (0 to 2)
BYTE	b_TimerMode	Timer mode selection (0 to 5) 0: Interrupt on terminal count 1: Hardware triggerable action 2: Rate generator 3: Square wave mode 4: Software triggered strobe 5: Hardware triggered strobe See table 3-7.
UINT	ui_ReloadValue	Start counting value or divider factor See table 3-7.

- Output:

No output signal has occurred

Task:

Configures the timer (*b_TimerNbr*) operating mode (*b_TimerMode*). You must call up this function before you call up any other function which gives access to the timer.

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_InitTimer  (b_BoardHandle,
                                     0,
                                     2,
                                     0xFF00);
```

Return value:

0: No error
-1: The handle parameter of the board is wrong
-2: Timer selection wrong
-3: Timer mode selection is wrong

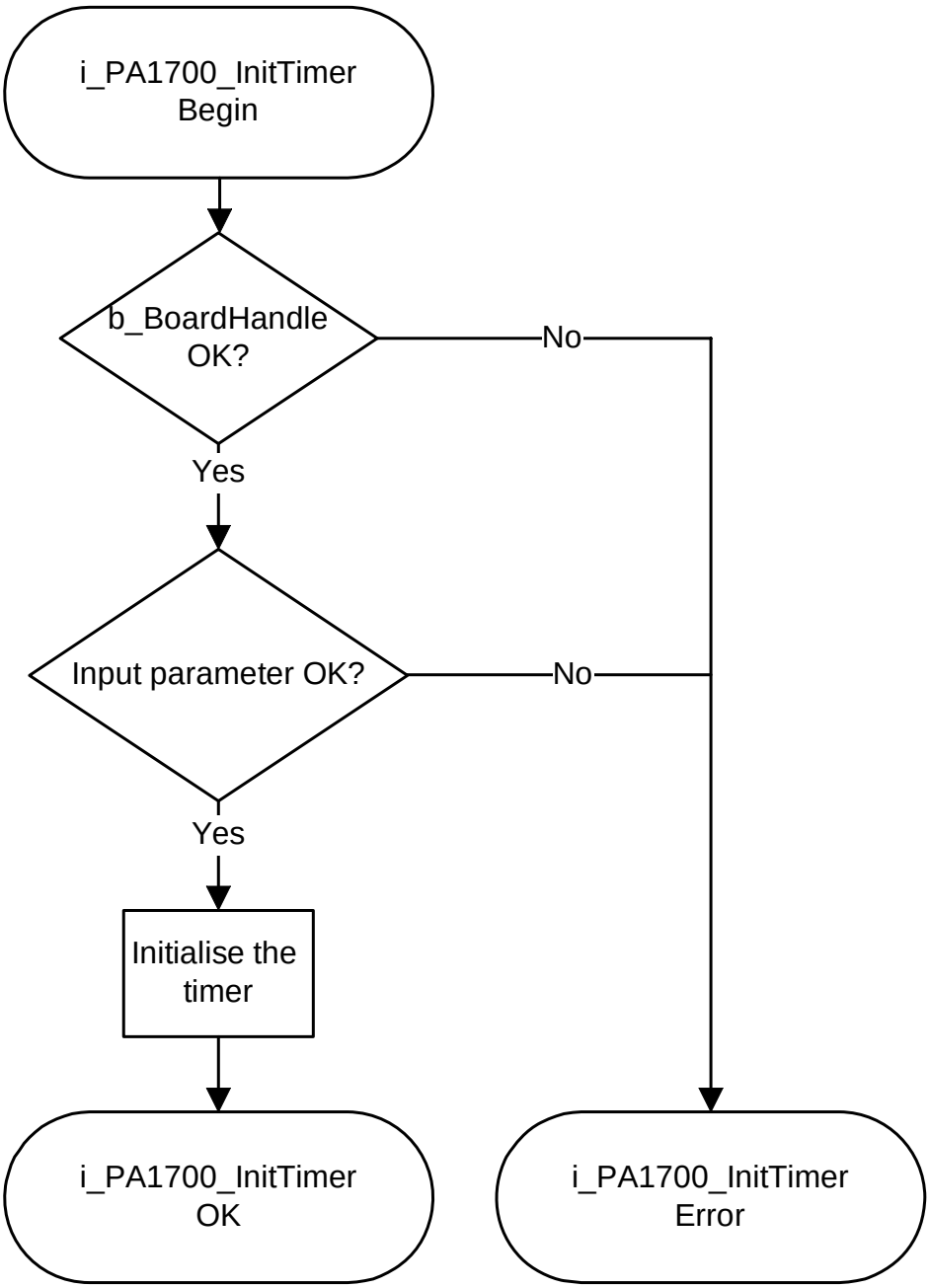
Table 3-7: Timer mode

Selected mode	Mode description	*u_ReloadValue* description	Hardware gate input action
0	Mode 0 is typically used for event counting. After the initialisation, OUT is initially low, and will remain low until the counter reaches zero. OUT then goes high and remains high until a new count is written. See "i_PA1700_WriteTimerValue" function.	Start counting value	Hardware gate
1	Mode 1 is similar to mode 0 except for the gate input action. The gate input is not used to enable or disable the timer (like in Mode 0), but to trigger it.	Start counting value	Hardware trigger
2	This mode functions like a divide-by-*ul_ReloadValue* counter. It is typically used to generate a real time clock interrupt. OUT will initially be high after the initialisation. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the counter reloads the initial count (*ul_ReloadValue*) and the process is repeated. This action can generate an interrupt. See function "i_PA1700_SetBoardIntRoutineX" and "i_PA1700_EnabledTimer"	Division factor	Hardware gate
3	Mode 3 is typically used for baud rate generation. This mode is similar to mode 2 except for the duty cycle of OUT. OUT will initially be high after the initialisation. When half the initial count (*ul_ReloadValue*) has expired, OUT goes low for the remainder of the count. The mode is periodic; the sequence above is repeated indefinitely.	Division factor	Hardware gate
4	OUT will be initially high after the initialisation. When the initial count expires, OUT will go low for one CLK pulse and then go high again. The counting sequences is triggered by writing a new value. See "i_PA1700_WriteTimerValue" function. If a new count is written during counting, it will be loaded on the next CLK pulse.	Start counting value	Hardware gate
5	Mode 5 is similar to mode 4 except for the gate input action.The gate input is not used to enable or disable the timer, but to trigger it.	Start counting value	Hardware trigger

Input

b_BoardHandle	b_TimerMode
b_TimerNbr	ui_ReloadValue

Function



Output

<Return Value>

2) i_PA1700_EnableTimer (...)

Syntax:

```
<Return value> = i_PA1700_EnableTimer   (BYTE   b_BoardHandle,
                                         BYTE   b_TimerNbr,
                                         BYTE   b_InterruptEnable)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_TimerNbr	Number of timer to be enabled (0 to 2)
BYTE	b_InterruptEnable	Enables or disables the timer interrupt. PA1700_ENABLE: Enables the timer interrupt PA1700_DISABLE: Disables the timer interrupt Only available for timer 2

- Output:

No output signal has occurred.

Task:

Enables the timer (*b_TimerNbr*). You must call up the "i_PA1700_InitTimer" function before calling up this function.
If you enable the timer interrupt, the timer generates an interrupt when the timer value have reached zero. See function "i_PA1700_SetBoardIntRoutineXX".

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_EnableTimer   (b_BoardHandle,
                                        0,
                                        PA1700_DISABLE);
```

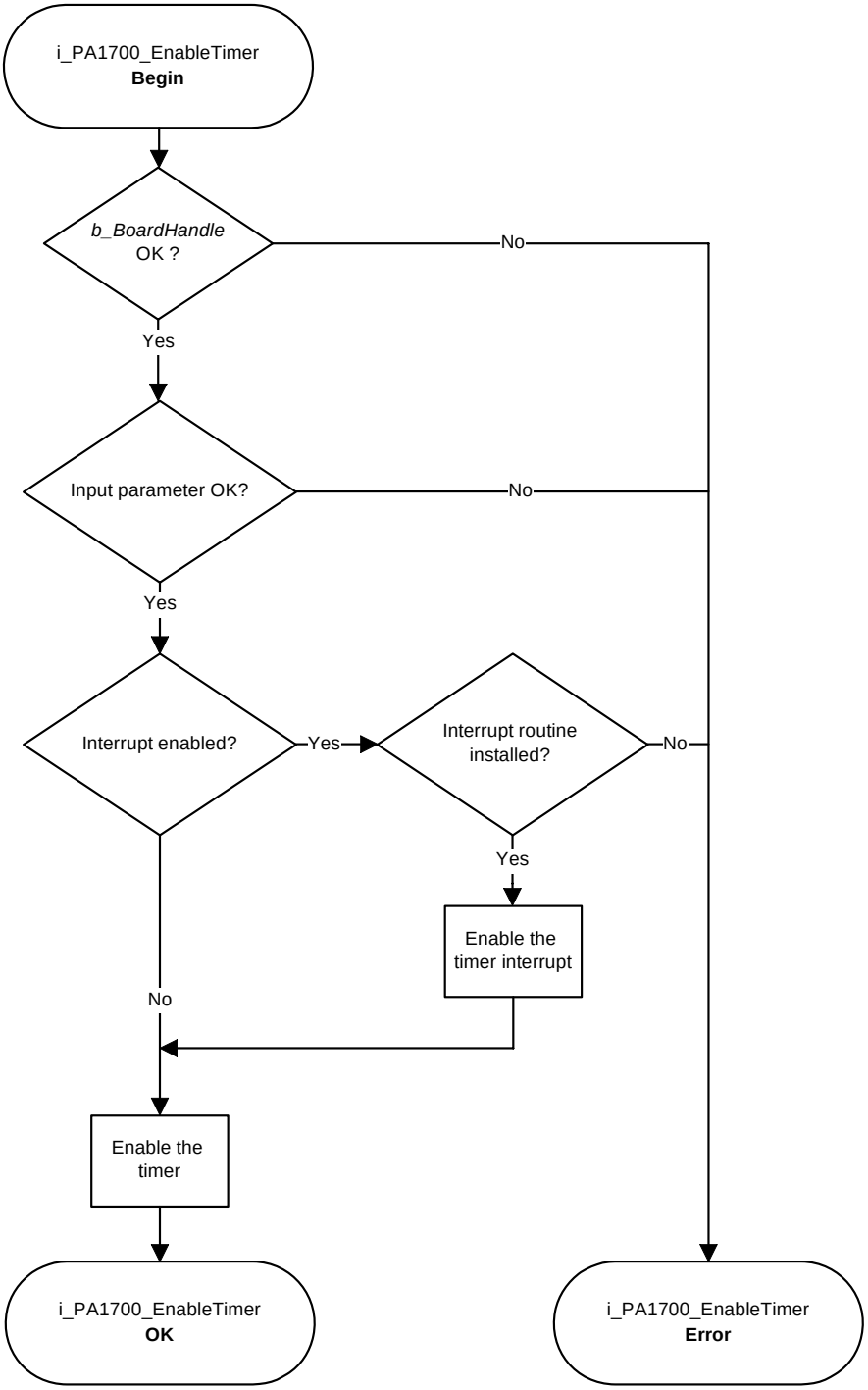
Return value:

0: No error
-1: The handle parameter of the board is wrong
-2: Timer selection wrong
-3: Timer not initialised. See function "i_PA1700_InitTimer"
-4: Interrupt parameter is wrong
-5: Interrupt function not initialised.
See function "i_PA1700_SetBoardIntRoutineXX"

Input

b_BoardHandle b_InterruptEnable
b_TimerNbr

Function



Output

<Return Value>

3) i_PA1700_DisableTimer (...)

Syntax:
<Return value> = i_PA1700_DisableTimer (BYTE b_BoardHandle,
BYTE b_TimerNbr)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_TimerNbr	Number of the timer to be disabled (0 to 2)

- Output:
No output signal has occurred

Task:
Disables the timer (*b_TimerNbr*).

Calling convention:
ANSI C :

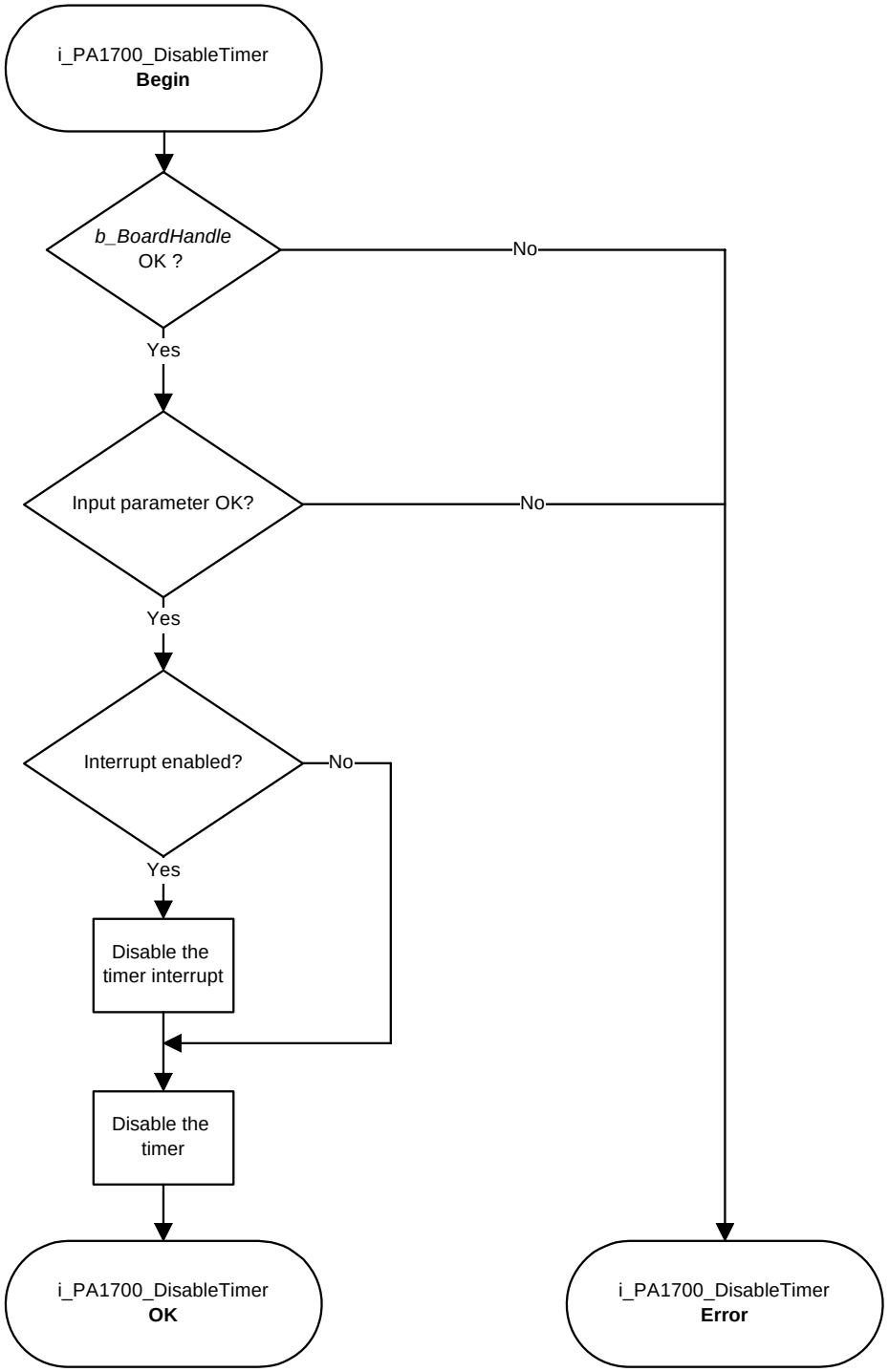
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_DisableTimer   (b_BoardHandle,  0);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: Timer selection wrong
-3: Timer not initialised. See function "i_PA1700_InitTimer"

Input

b_BoardHandle b_TimerNbr

Function



Output

<Return Value>

3.4.2 Reading the timer

1) i_PA1700_ReadTimerValue (...)

Syntax:
<Return value> = i_PA1700_ReadTimerValue
(BYTE b_BoardHandle,
BYTE b_TimerNbr,
PUINT pui_TimerValue)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_TimerNbr	Timer number to be read (0 to 2)

- Output:

PUINT	pui_TimerValue	Timer value

Task:
Returns the timer value of the selected timer (*b_TimerNbr*).

Calling convention:
ANSI C :

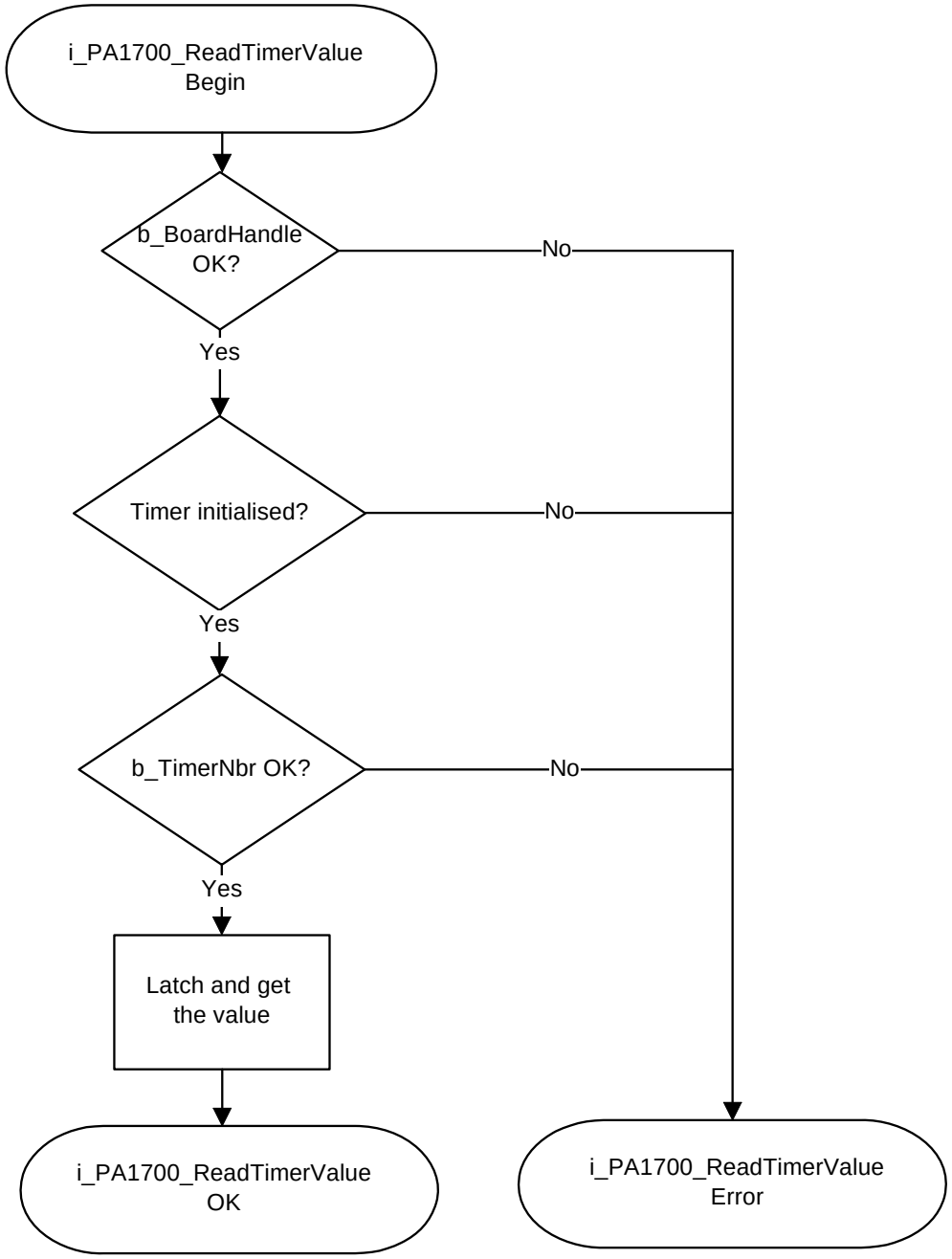
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned int    ui_TimerValue;

i_ReturnValue = i_PA1700_ReadTimerValue  (b_BoardHandle,
                                          0,
                                         & ui_TimerValue);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: Timer selection wrong
-3: Timer not initialised. See function "i_PA1700_InitTimer"

Input

b_BoardHandle b_TimerNbr

Function



Output

pui_TimerValue

<Return Value>

2) i_PA1700_ReadAllTimerValue (...)

Syntax:
<Return value> = i_PA1700_ReadAllTimerValue
(BYTE b_BoardHandle,
PUINT pui_TimerValueArray)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**

- Output:

PUINT	pui_TimerValueArray	Timer value array. Element 0 contains the timer 0 value. Element 1 contains the timer 1 value. Element 2 contains the timer 2 value.

Task:
Returns all timer values.

Calling convention:
ANSI C :

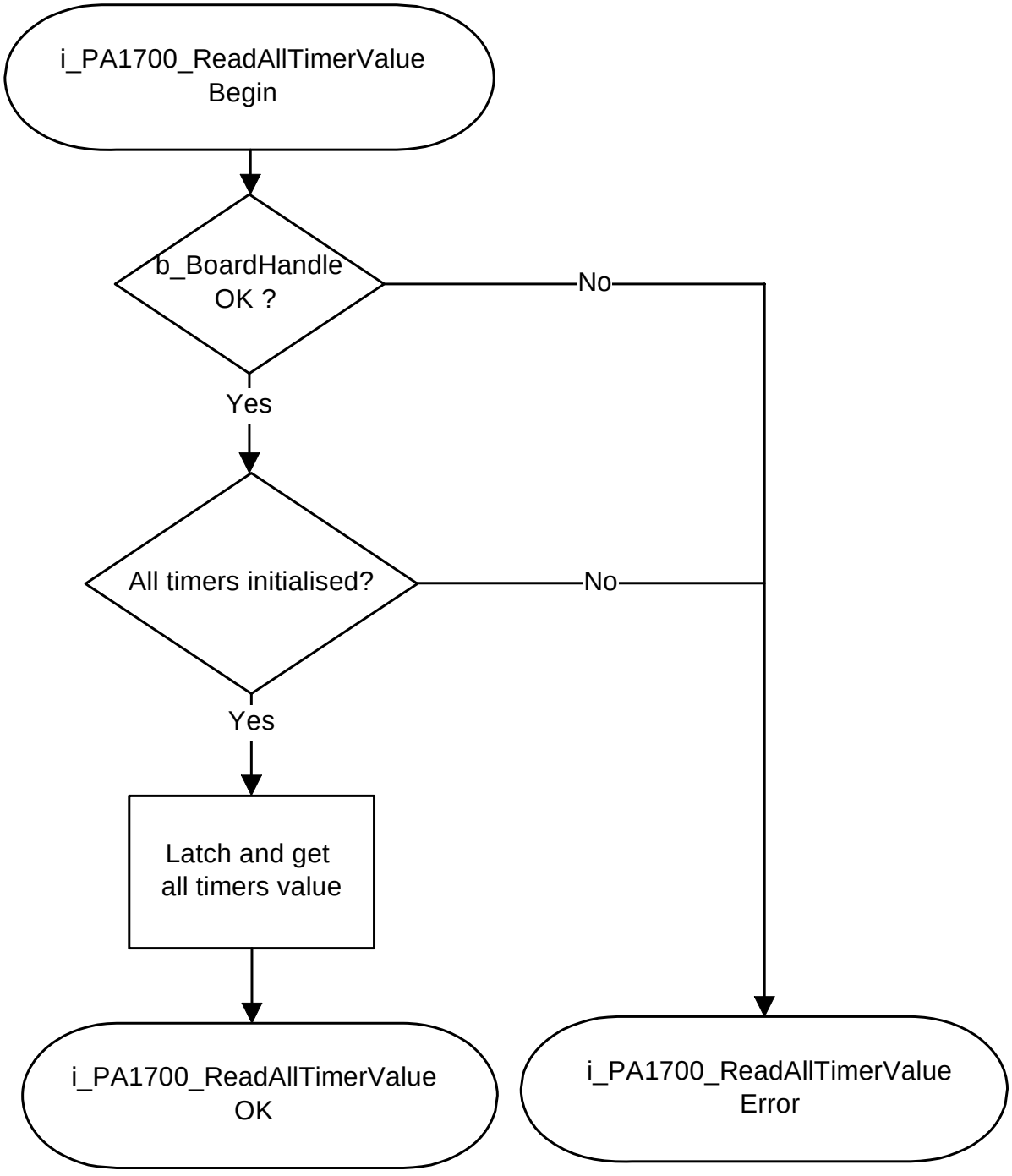
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;
unsigned int    ui_TimerValueArray [3];

i_ReturnValue = i_PA1700_ReadAllTimerValue      (b_BoardHandle,
                                                ui_TimerValueArray);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: Timer selection wrong
-3: Timer 0 not initialised. See function "i_PA1700_InitTimer"
-4: Timer 1 not initialised. See function "i_PA1700_InitTimer"
-5: Timer 2 not initialised. See function "i_PA1700_InitTimer"

Input

b_BoardHandle

Function



Output

pui_TimerValueArray

<Return Value>

3.4.3 Writing in the timer

1) i_PA1700_WriteTimerValue (...)

Syntax:
<Return value> = i_PA1700_WriteTimerValue
(BYTE b_BoardHandle
BYTE b_TimerNbr,
UINT ui_WriteValue)

Parameters:
- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_TimerNbr	Number of the timer to be tested (0 to 2)
UINT	ui_WriteValue	Value to be written

- Output:
No output signal has occurred

Task:
Writes the value (ul_WriteValue) into the selected timer (*b_TimerNbr*). The action depends on the timer mode. See table 3-7

Calling convention:
ANSI C :

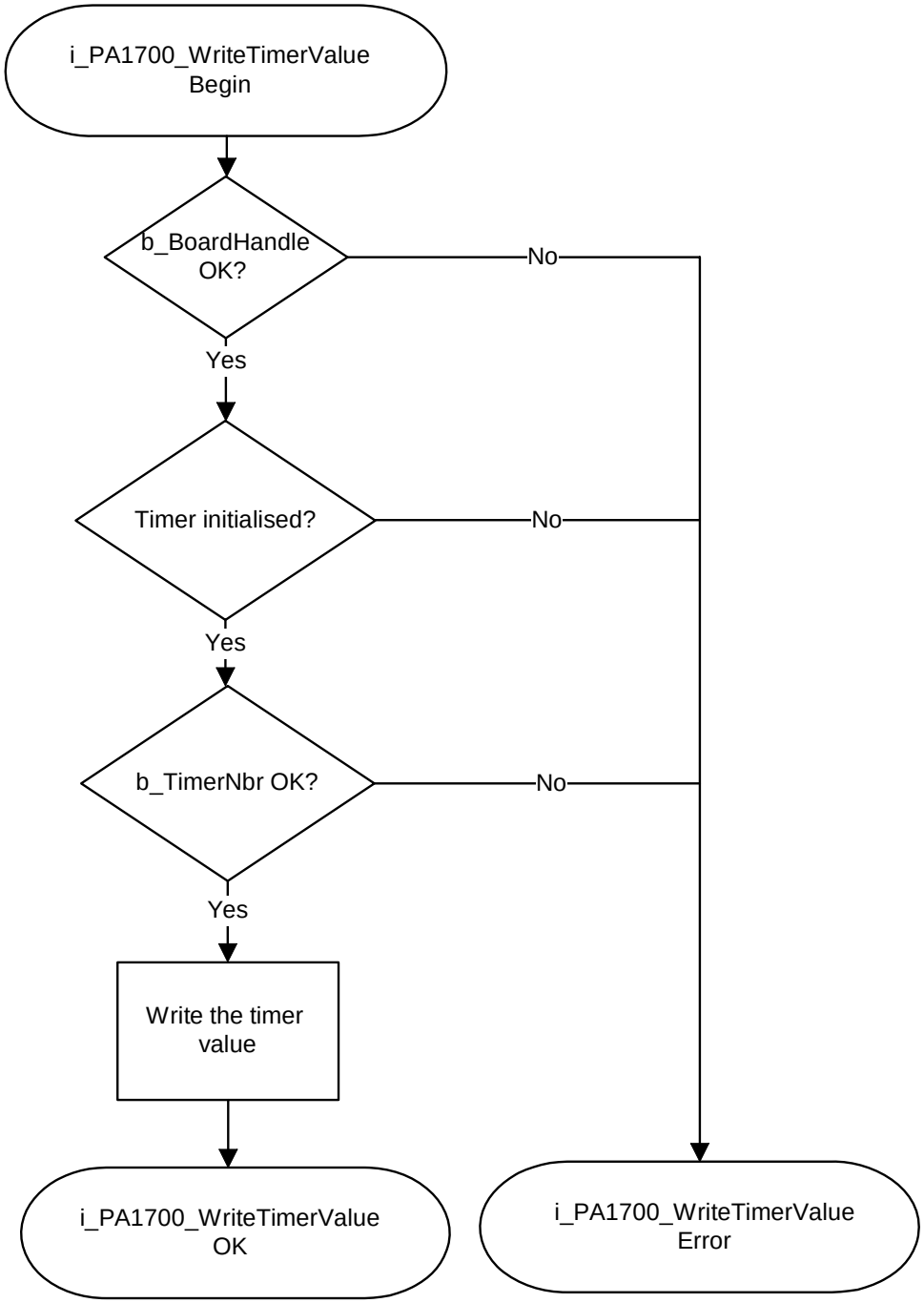
```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_WriteTimerValue
                                        (b_BoardHandle,
                                         0,
                                         0xFF00);
```

Return value:
0: No error
-1: The handle parameter of the board is wrong
-2: Timer selection wrong
-3: Timer not initialised. See function "i_PA1700_InitTimer"

Input

b_BoardHandle ui_WriteValue
b_TimerNbr

Function



Output

<Return Value>

3.5 PIO

3.5.1 PIO initialisation

1) i_PA1700_InitPIO (...)

Syntax:

```
<Return value> = i_PA1700_InitPIO    (BYTE   b_BoardHandle,
                                      BYTE   b_PortAMode,
                                      BYTE   b_PortBMode,
                                      BYTE   b_PortCLowMode,
                                      BYTE   b_PortCHighMode)
```

Paramters:

- Input:

BYTE	b_BoardHandle	Handle of board **PA 1700**
BYTE	b_PortAMode	Mode selection for port A 0: Used for output 1: Used for input
BYTE	b_PortBMode	Mode selection for port B 0: Used for output 1: Used for input
BYTE	b_PortCLowMode	Low mode selection for port C 0: Used for output 1: Used for input
BYTE	b_PortCHighMode	High mode selection for port C 0: Used for output 1: Used for input

- Output:

No output signal has occurred.

Task:

Initialises the PIO (82C55A).

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_InitPIO    (b_BoardHandle,
                                     0,
                                     0,
                                     0,
                                     0);
```

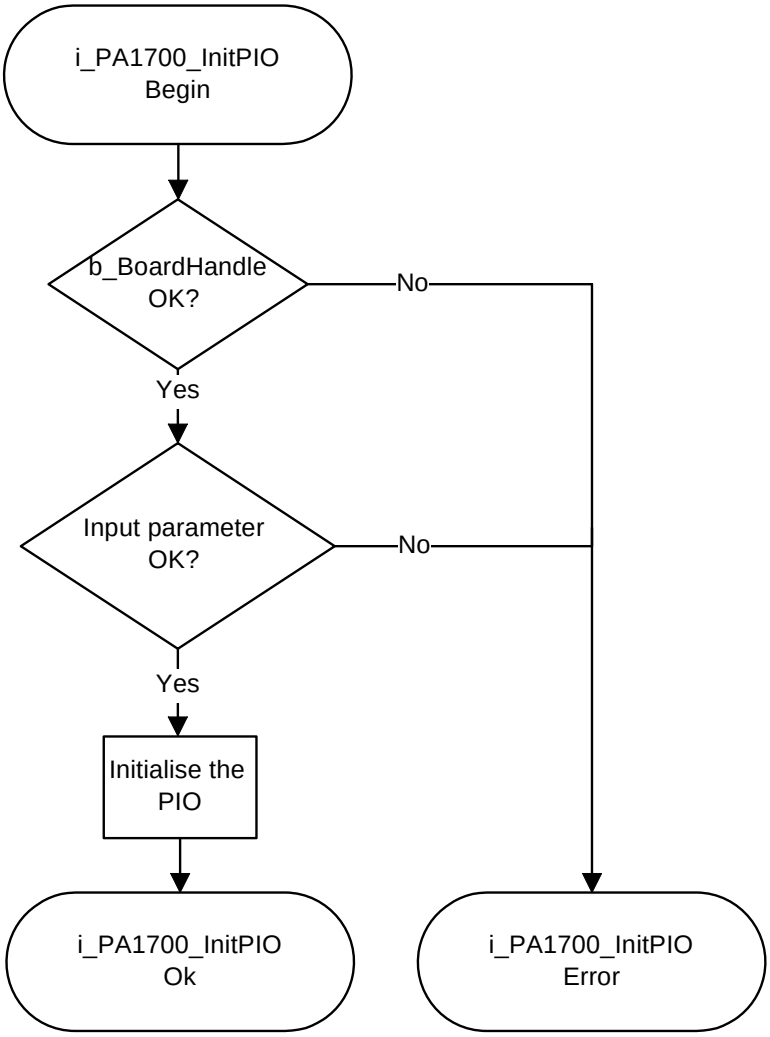
Return value:

 0: No error
-1: The handle parameter of the board is wrong
-2: Port A mode selection is wrong
-3: Port B mode selection is wrong
-4: Port C low mode selection is wrong
-5: Port C high mode selection is wrong

Input

b_BoardHandle
b_PortAMode
b_PortBMode
b_PortCLowMode
b_PortCHighMode

Function



Output

<Return Value>

2) i_PA1700_ReadPIO (...)

Syntax:

```
<Return value> = i_PA1700_ReadPIO    (BYTE    b_BoardHandle,
                                      BYTE    b_SelectedPort,
                                      PBYTE   pb_PortValue)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the board **PA 1700**
BYTE	b_SelectedPort	Port selection to read 0: Port A 1: Port B 2: Port C 3: Status register

- Output:

PBYTE	pb_PortValue	Port value

Task:

Reads the selected port value.

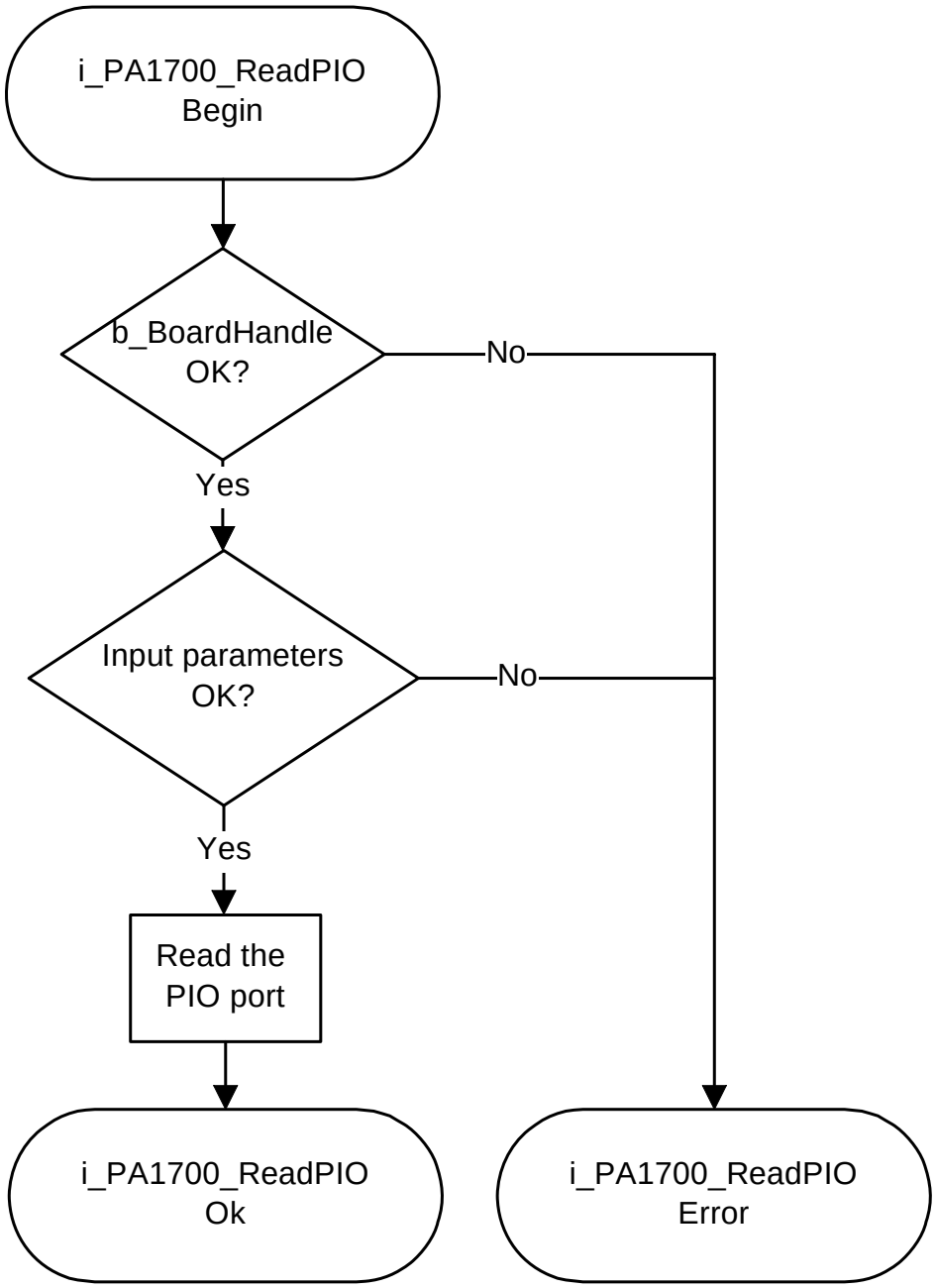
Calling convention:

ANSI C :

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

i_ReturnValue = i_PA1700_ReadPort   (b_BoardHandle,
                                     0,
                                     &b_PortValue);
```

Return value:

0: No error.
-1: The handle parameter of the board is wrong.
-2: Port selection error.

Input
b_BoardHandle b_SelectedPort
Function

Output
pb_PortValue
<Return Value>

3) i_PA1700_WritePIO (...)

Syntax:

```
<Return value> = i_PA1700_WritePIO    (BYTE    b_BoardHandle,
                                       BYTE    b_SelectedPort,
                                       BYTE    b_WriteValue)
```

Parameters:

- Input:

BYTE	b_BoardHandle	Handle of the board **PA 1700**
BYTE	b_SelectedPort	Selection of the board to be read 0: Port A 1: Port B 2: Port C 3: Commando register
BYTE	b_WriteValue	Value to be written

- Output:

No output signal has occurred

Task:

Writes the value in the selected port

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned char   b_BoardHandle;

i_ReturnValue = i_PA1700_WritePort  (b_BoardHandle,
                                     0,
                                     0x55);
```

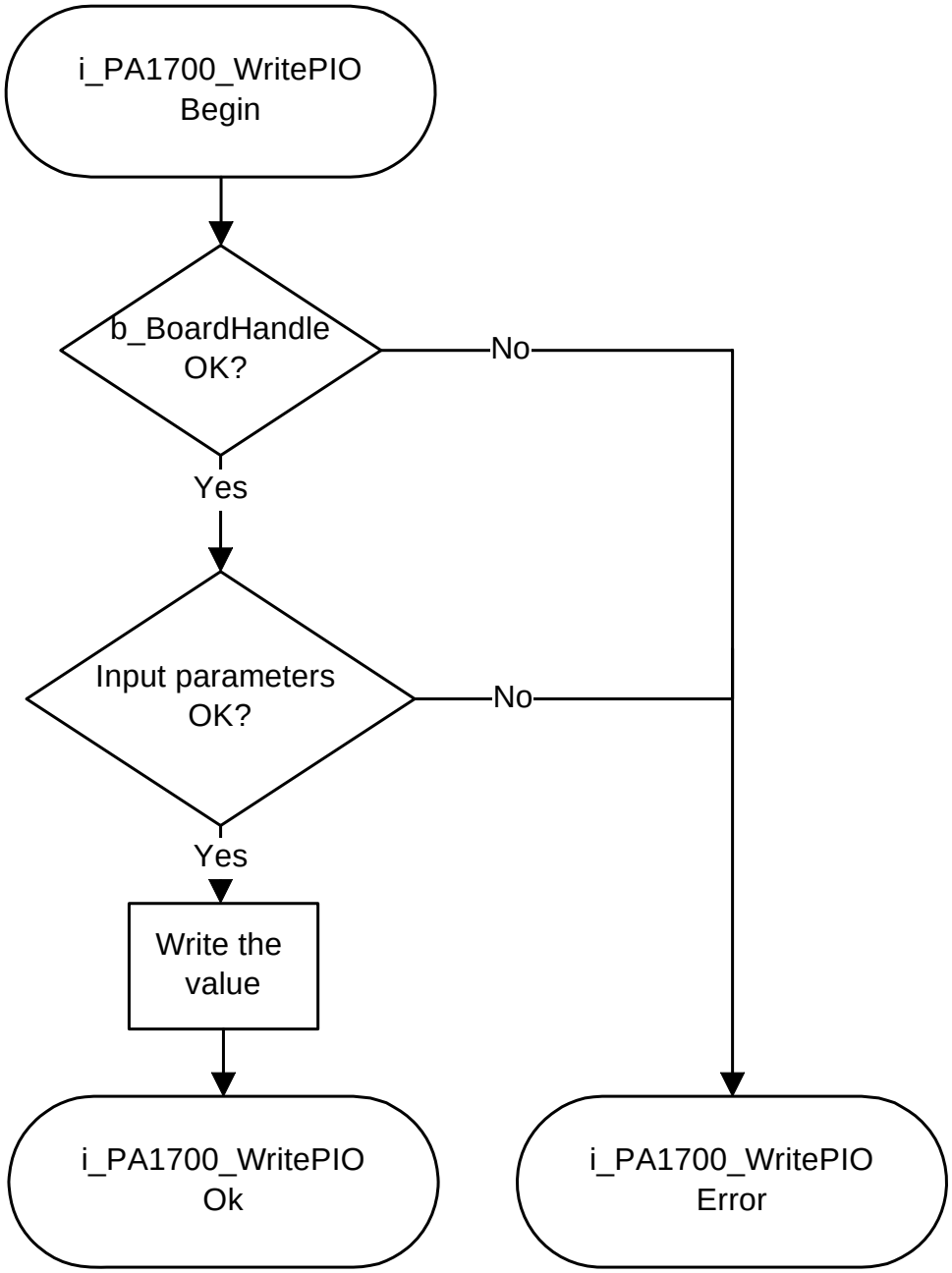
Return value:

0: No error.
-1: The handle parameter of the board is wrong.
-2: Port selection error.

Input

b_BoardHandle b_WriteValue
b_SelectedPort

Function



Output

<Return Value>

3.6 Functions to be used in Kernel mode

IMPORTANT!
These functions are only available for the Windows NT and Windows 95 user interrupt routine in the synchronous mode.
See function "i_PA1700_SetBoardIntRoutineWin32"

3.6.1 Counter

1) i_PA1700_KRNL_ClearCounterValue (...)

Syntax:
<Return value> = i_PA1700_KRNL_ClearCounterValue
(UINT ui_BaseAddress,
BYTE b_HardCounterNbr)

Parameters:
- Input:

UINT	ui_BaseAddress	**PA1700** base address. See "i_PA1700_GetHardwareInformation"
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)

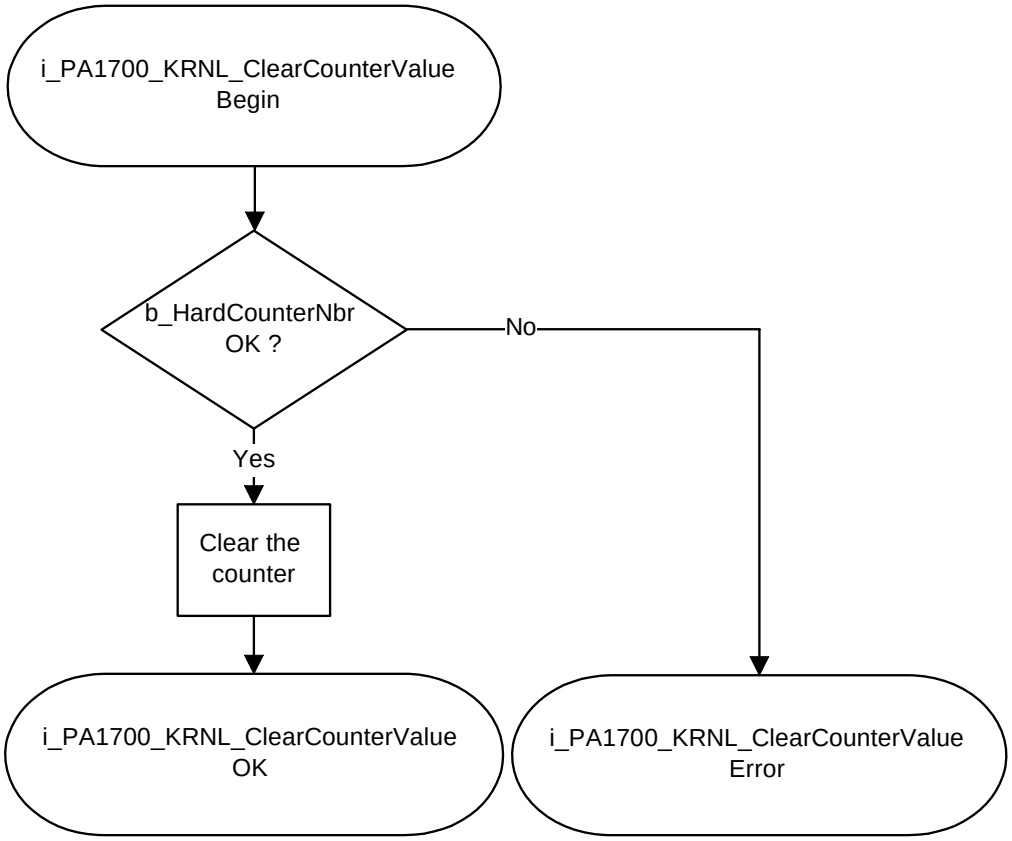
- Output:
No output signal has occurred.

Task:
Clears the counter value of the selected hardware counter (*b_HardCounterNbr*).

Calling convention:

```
ANSI C :
int             i_ReturnValue;
unsigned int    ui_BaseAddress;

i_ReturnValue = i_PA1700_KRNL_ClearCounterValue (ui_BaseAddress,
                                                 0);
```

Return value:
0: No error
-1: The selected hardware counter number is wrong

Input
ui_ BaseAddress b_HardCounterNbr
Function

Output
<Return Value>

2) i_PA1700_KRNL_Read16BitCounterValue (...)

Syntax:
<Return value> = i_PA1700_KRNL_Read16BitCounterValue
(UINT ui_BaseAddress,
BYTE b_HardCounterNbr,
BYTE b_SelectedCounter,
PUINT pui_CounterValue)

Parameters:

- Input:		
UINT	ui_BaseAddress	Base address of the **PA1700**. See "i_PA1700_GetHardwareInformation"
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)
BYTE	b_SelectedCounter	Selected 16-bit counter (0 or 1)
- Output:		
PUINT	pui_CounterValue	16-bit counter value

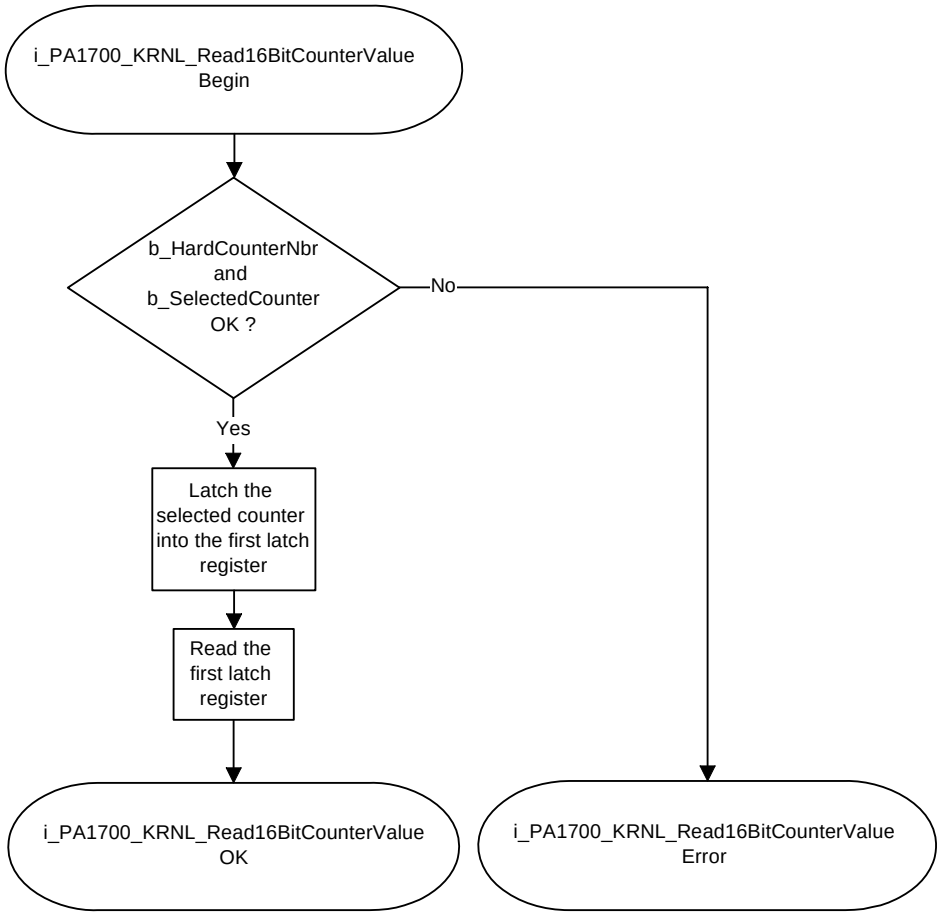
Task:
Latches the 16-bit counter (*b_SelectedCounter*) of the selected hardware counter (*b_HardCounterNbr*) into the first latch register and returns the latched value.

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned int    ui_BaseAddress;
unsigned int    ui_CounterValue;

i_ReturnValue = i_PA1700_KRNL_Read16BitCounterValue
                                        (ui_BaseAddress,
                                         0,
                                         0,
                                         &ui_CounterValue);
```

Return value:
0: No error
-1: The selected hardware counter number is wrong
-2: The selected 16-bit counter is wrong

Input
ui_BaseAddress b_SelectedCounter b_HardCounterNbr
Function

Output
pui_CounterValue
<Return Value>

3) i_PA1700_KRNL_Read32BitCounterValue (...)

Syntax:

```
<Return value> = i_PA1700_KRNL_Read32BitCounterValue
                                 (UINT    ui_BaseAddress,
                                  BYTE    b_HardCounterNbr,
                                  PULONG  pul_CounterValue)
```

Parameters:

- Input:

UINT	ui_BaseAddress	Base address of the **PA 1700**. See "i_PA1700_GetHardwareInformation"
BYTE	b_HardCounterNbr	Module number to be configured (0 to 2)

- Output:

PULONG	pul_CounterValue	32-bit counter value

Task:

Latches the 32-bit counter of the selected hardware counter (*b_HardCounterNbr*) into the first latch register and returns the latched value.

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned int    ui_BaseAddress;
unsigned long   ul_CounterValue;

i_ReturnValue = i_PA1700_KRNL_Read32BitCounterValue
                                                (ui_BaseAddress,
                                                 0,
                                                 &ul_CounterValue);
```

Return value:

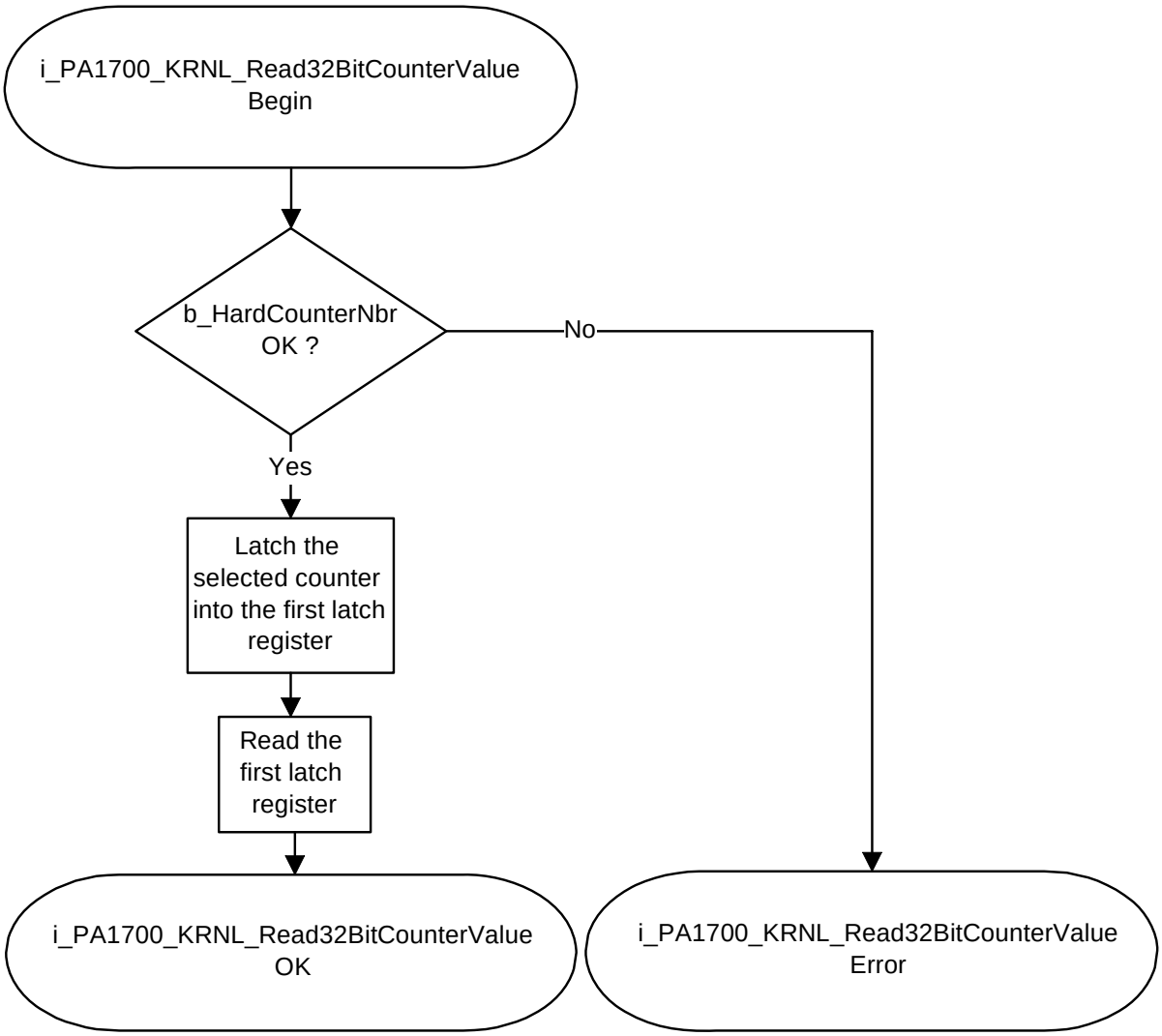
0: No error

-1: The selected hardware counter number is wrong

Input

ui_BaseAddress b_HardCounterNbr

Function



Output

pul_CounterValue

<Return Value>

4) i_PA1700_KRNL_Write16BitCounterValue (...)

Syntax:

```
< Return value > = i_PA1700_KRNL_Write16BitCounterValue
                                        (UINT    ui_BaseAddress,
                                         BYTE    b_HardCounterNbr,
                                         BYTE    b_SelectedCounter,
                                         UINT    ui_WriteValue)
```

Parameters:

- Input:

UINT	ui_BaseAddress	Base address of the **PA 1700**. See "i_PA1700_GetHardwareInformation"
BYTE	b_HardCounterNbr	Module number to be configured (0 to 2)
BYTE	b_SelectedCounter	Selected 16-bit counter (0 or 1)
UINT	ui_WriteValue	16-bit write value

- Output:

No output signal has occurred.

Task:

Writes a 16-bit value (*ui_WriteValue*) into the 16-bit counter (*b_SelectedCounter*) of the selected hardware counter (*b_HardCounterNbr*).

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned int    ui_BaseAddress;

i_ReturnValue = i_PA1700_KRNL_Write16BitCounterValue
                                                (ui_BaseAddress,
                                                 0,
                                                 0,
                                                 2000);
```

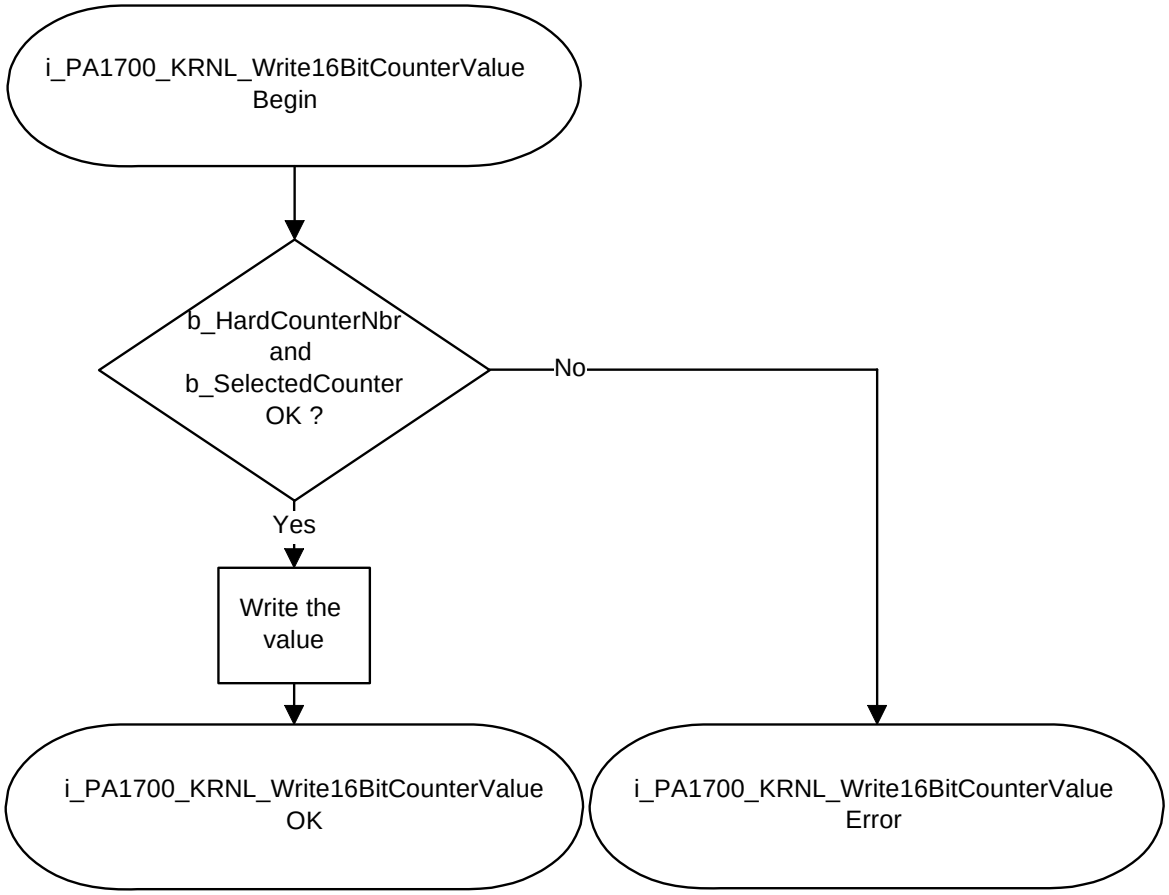
Return value:

0: No error
-1: The selected hardware counter is wrong
-2: The selected 16-bit counter is wrong

Input

ui_BaseAddress	b_SelectedCounter
b_HardCounterNbr	ui_WriteValue

Function



Output

<Return Value>

5) i_PA1700_KRNL_Write32BitCounterValue (...)

Syntax:
< Return value > = i_PA1700_KRNL_Write32BitCounterValue
(UINT ui_BaseAddress,
BYTE b_HardCounterNbr,
ULONG ul_WriteValue)

Parameters:
- Input:

UINT	ui_BaseAddress	Base address of the **PA 1700**. See "i_PA1700_GetHardwareInformation"
BYTE	b_HardCounterNbr	Number of the hardware counter to be configured (0 to 2)
ULONG	ul_WriteValue	32-bit value to be written

- Output:
No output signal has occurred.

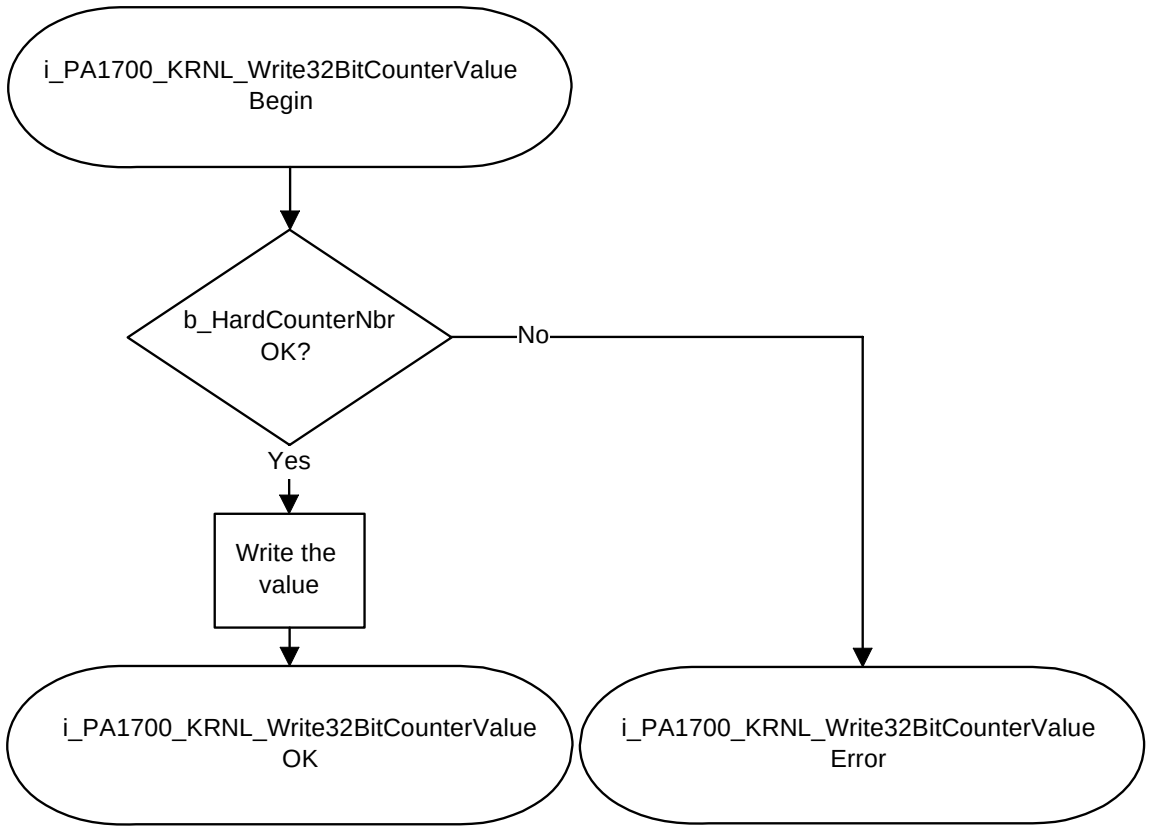
Task:
Writes a 32-bit value (*ul_WriteValue*) into the 32-bit counter of the selected hardware counter (*b_HardCounterNbr*).

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned int    ui_BaseAddress;

i_ReturnValue = i_PA1700_KRNL_Write32BitCounterValue
                                        (ui_BaseAddress,
                                         0,
                                         200000);
```

Return value:
0: No error
-2: The selected hardware counter number is wrong

Input
ui_BaseAddress ul_WriteValue b_HardCounterNbr
Function

Output
<Return Value>

3.6.2 Counter

1) i_PA1700_KRNL_ReadTimerValue (...)

Syntax:
<Return value> = i_PA1700_KRNL_ReadTimerValue
(UINT ui_BaseAddress,
BYTE b_TimerNbr,
PUINT pui_TimerValue)

Parameters:
- Input:

UINT	ui_BaseAddress	Base address of the **PA 1700** board
BYTE	b_TimerNbr	Timer number to be read (0 to 2)

- Output:

PUINT	pui_TimerValue	Timer value

Task:
Returns the timer value of the selected timer (*b_TimerNbr*)

Calling convention:
ANSI C :

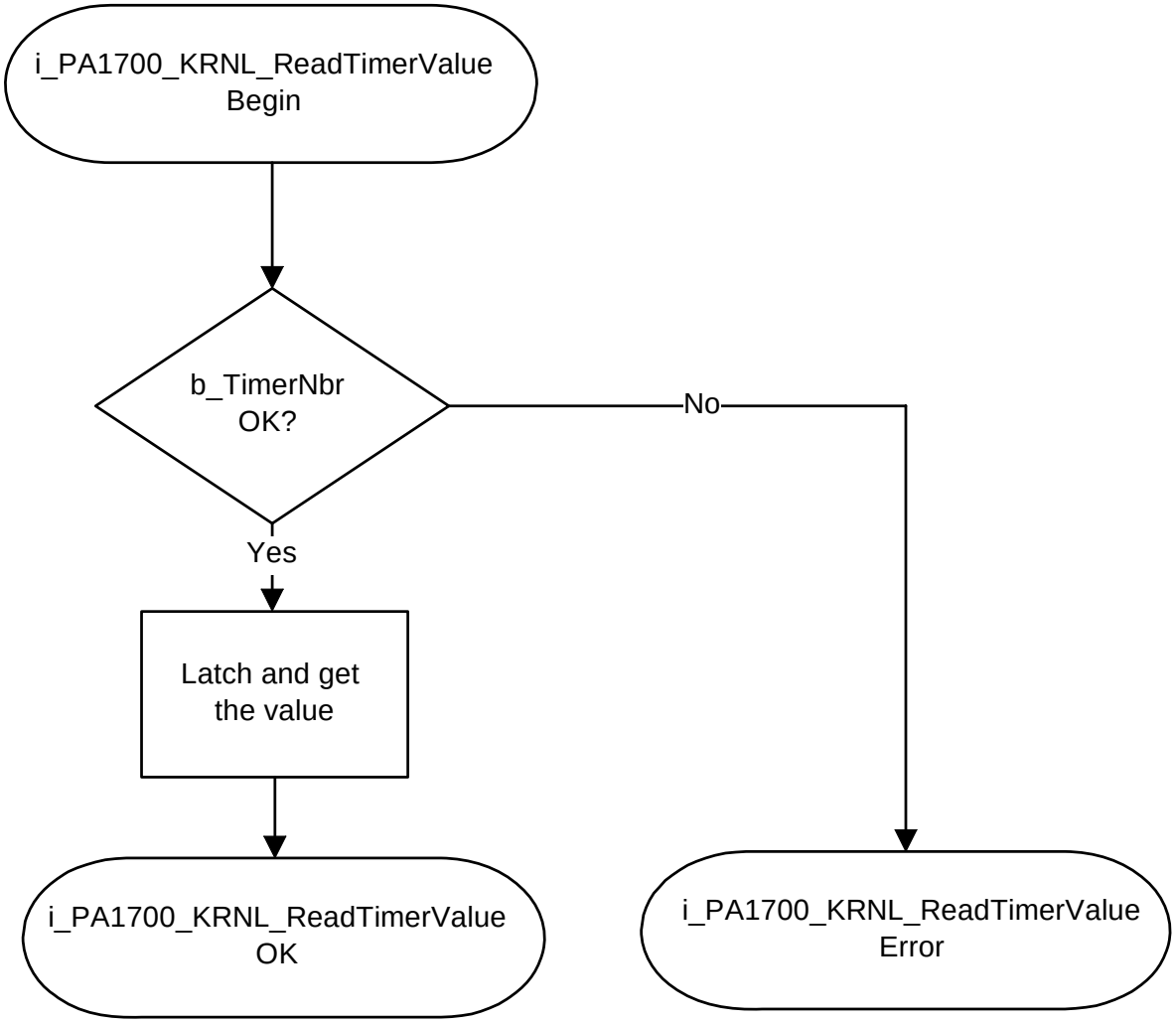
```
int             i_ReturnValue;
unsigned int    ui_BaseAddress;
unsigned int    ui_TimerValue;

i_ReturnValue = i_PA1700_KRNL_ReadTimerValue    (ui_BaseAddress,
                                                 0,
                                                 &ui_TimerValue);
```

Return value:
0: No error
-1: Timer selection wrong

Input

ui_BaseAddress pui_TimerValue
b_TimerNbr

Function



Output

pui_TimerValue

<Return Value>

2) i_PA1700_KRNL_ReadAllTimerValue (...)

Syntax:
<Return value> = i_PA1700_KRNL_ReadAllTimerValue
(UINT ui_BaseAddress,
PUINT pui_TimerValueArray)

Parameters:
- Input:

UINT	ui_BaseAddress	Base address of the **PA 1700** board

- Output:

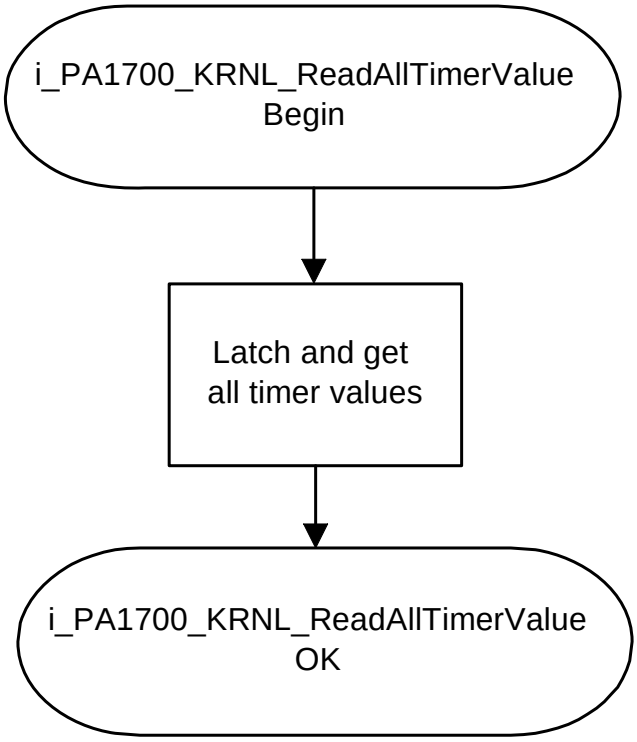
PUINT	pui_TimerValueArray	Timer value array. Element 0 contains the timer 0 value. Element 1 contains the timer 1 value. Element 2 contains the timer 2 value.

Task:
Returns all timer values.

Calling convention:
ANSI C :

```
int             i_ReturnValue;
unsigned int    ui_BaseAddress;
unsigned int    ui_TimerValueArray [3];

i_ReturnValue = i_PA1700_KRNL_ReadAllTimerValue
                                        (ui_BaseAddress,
                                         ui_TimerValueArray);
```

Return value:
0: No error

Input
ui_BaseAddress pui_TimerValueArray
Function

Output
pui_TimerValueArray
<Return Value>

3) i_PA1700_KRNL_WriteTimerValue (...)

Syntax:

```
<Return value> = i_PA1700_KRNL_WriteTimerValue
                                    (UINT   ui_BaseAddress,
                                     BYTE     b_TimerNbr,
                                     UINT   ui_WriteValue)
```

Parameters:

- Input:

UINT	ui_BaseAddress	Base address of the **PA 1700** board
BYTE	b_TimerNbr	Number of the timer to be tested (0 to 2)
UINT	ui_WriteValue	Value to be written

- Output:

No output signal has occurred.

Task:

Writes the value (ui_WriteValue) into the selected timer (*b_TimerNbr*). The action depends on the timer mode. See table 3-7

Calling convention:

ANSI C :

```
int             i_ReturnValue;
unsigned int    ui_BaseAddress;

i_ReturnValue = i_PA1700_WriteTimerValue
                                    (ui_BaseAddress,
                                     0,
                                     0xFF00);
```

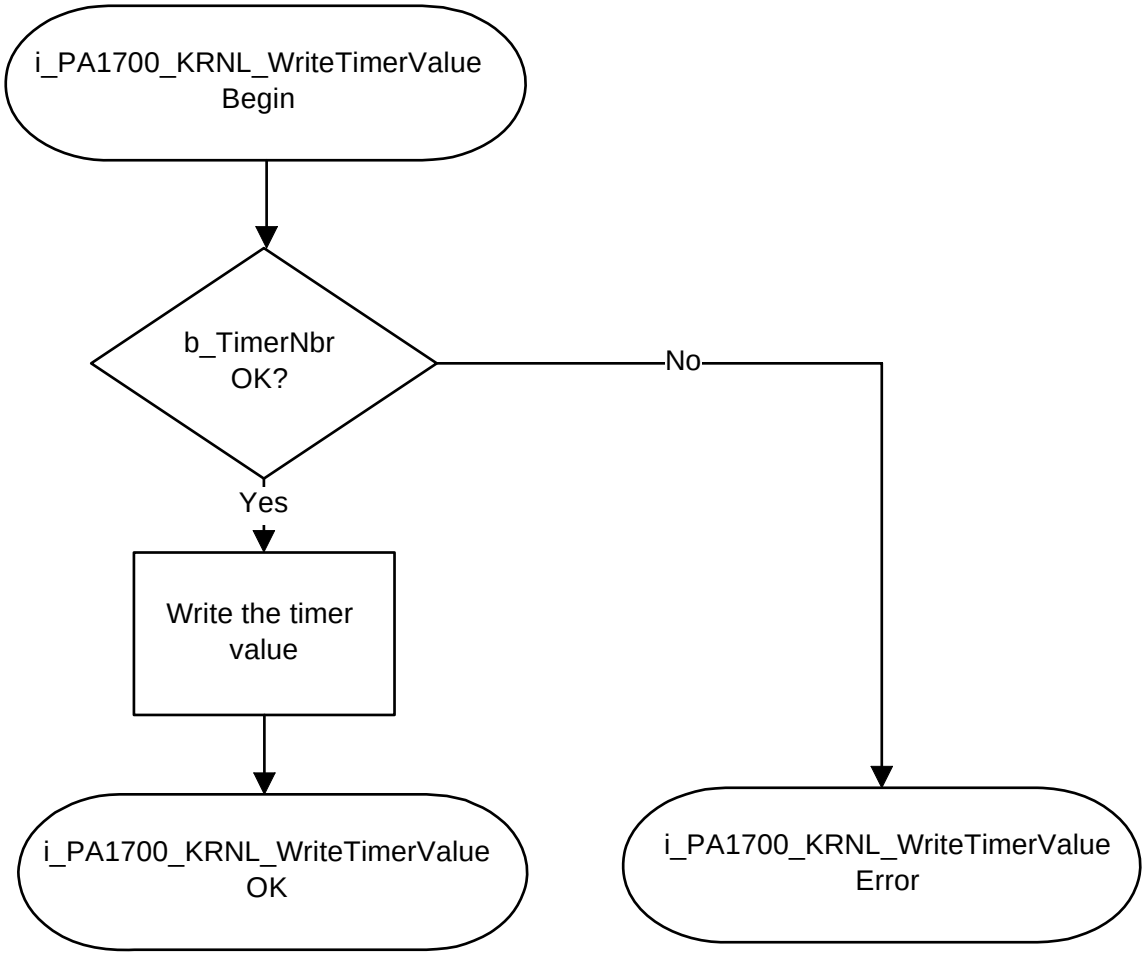
Return value:

0: No error
-2: Timer selection wrong

Input

ui_BaseAddress ui_WriteValue
b_TimerNbr

Function



Output

<Return Value>

3.6.3 PIO

1) i_PA1700_KRNL_ReadPIO (...)

Syntax:
<Return value> = i_PA1700_KRNL_ReadPIO
(UINT ui_Address,
BYTE b_SelectedPort,
PBYTE pb_PortValue)

Parameters:
- Input:

UINT	ui_Address	Address of the board **PA 1700**
BYTE	b_SelectedPort	Selection of the port to be read 0: Port A 1: Port B 2: Port C 3: Status register

- Output:

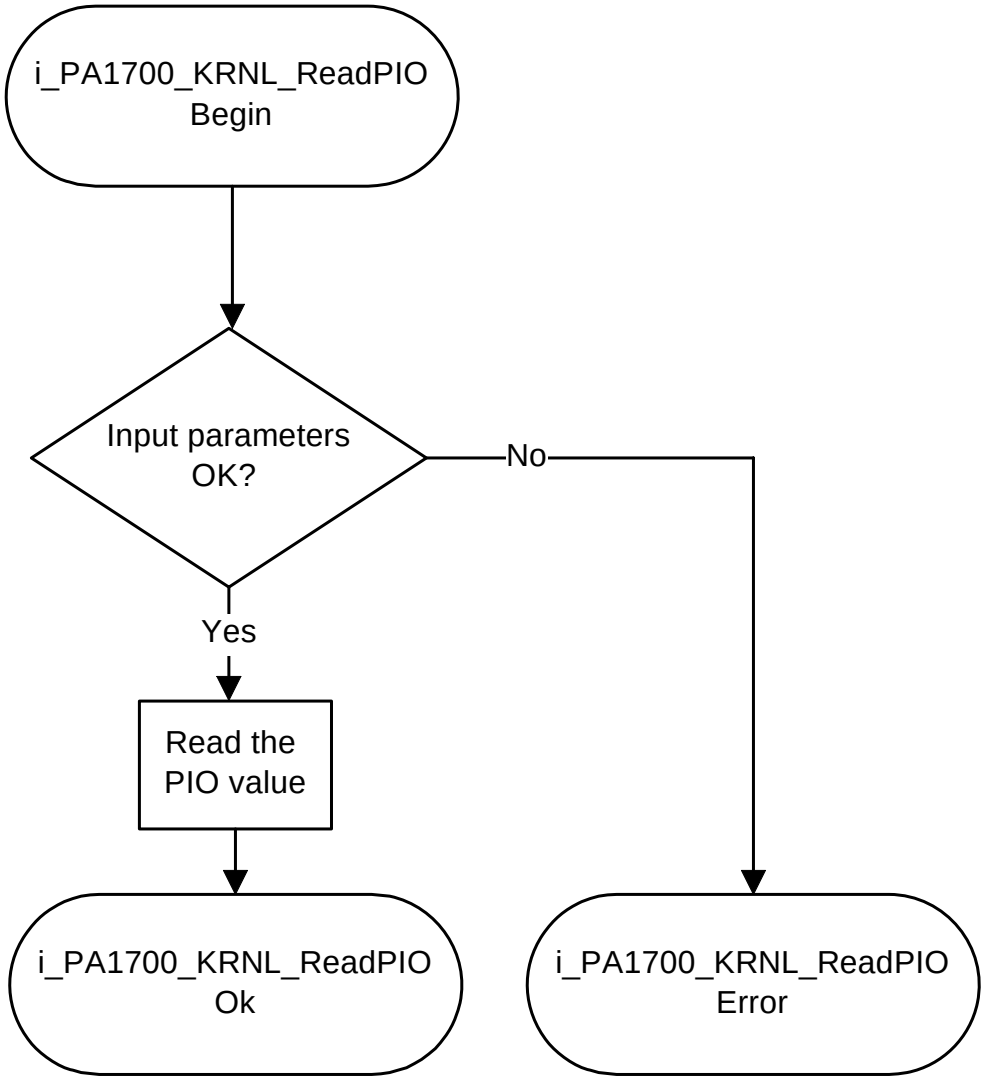
PBYTE	pb_PortValue	Port value

Task:
Reads the selected port value.

Calling convention:
ANSI C :

```
int             i_ReturnValue;
unsigned char   b_PortValue;

i_ReturnValue = i_PA1700_KRNL_ReadPort(0x390,
                                       0,
                                       &b_PortValue);
```

Return value:
0: No error.
-1: Port selection error.

Input
ui_Address pb_PortValue b_SelectedPort
Function

Output
pb_PortValue
<Return Value>

2) i_PA1700_KRNL_WritePIO (...)

Syntax:
<Return value> = i_PA1700_KRNL_WritePIO
(UINT ui_Address,
BYTE b_SelectedPort,
BYTE b_WriteValue)

Parameters:
- Input:

UINT	ui_Address	Address of the board **PA 1700**
BYTE	b_SelectedPort	Selection of the port to be read 0: Port A 1: Port B 2: Port C 3: Commando register
BYTE	b_WriteValue	Value to be written

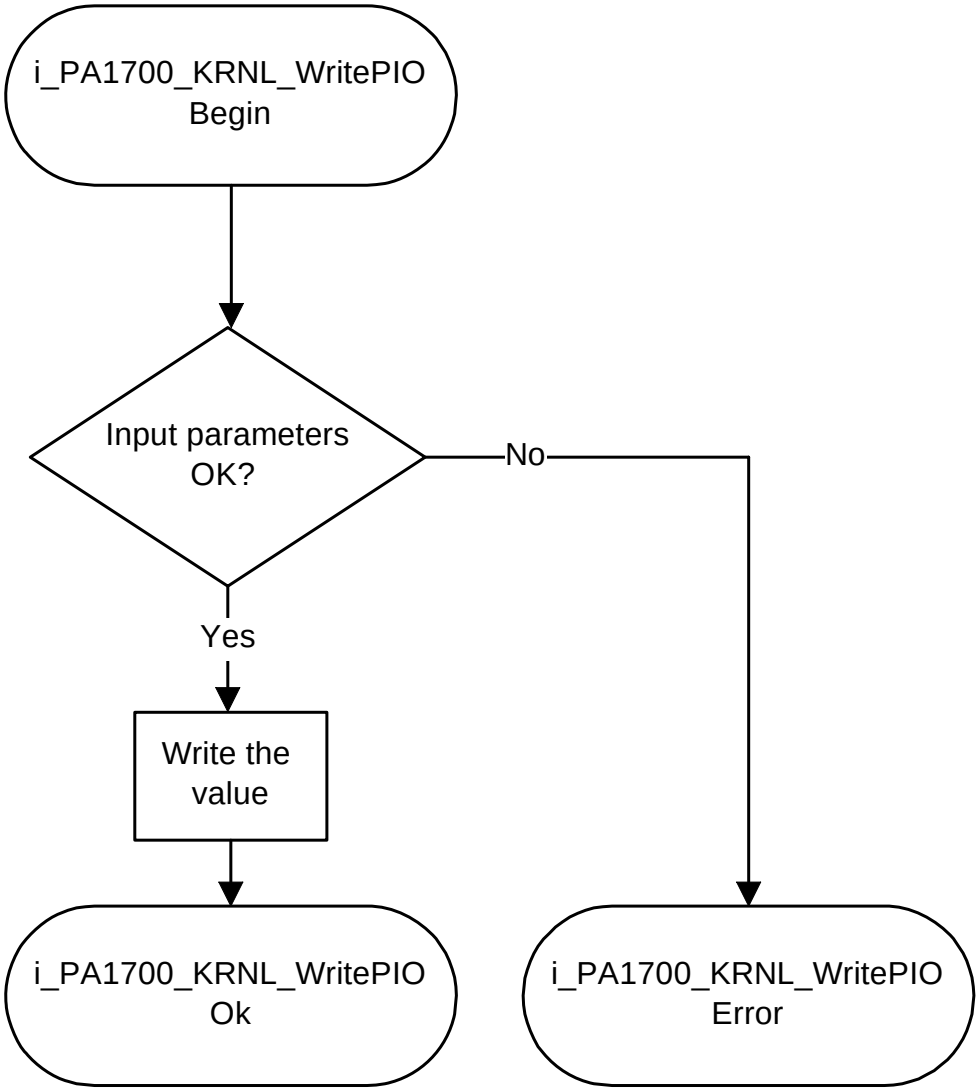
- Output:
No output signal has occurred

Task:
Writes the value of the selected port

Calling convention:
ANSI C :

```
int       i_ReturnValue;

i_ReturnValue = i_PA1700_KRNL_WritePort          (0x390,
                                                  0,
                                                  0x55);
```

Return value:
0: No error.
-1: Port selection error.

Input
ui_Address b_SelectedPort b_WriteValue
Function

Output
<Return Value>