



DIN EN ISO 9001:2000
certified



ADDI-DATA GmbH
Airpark Business Center
Airport Boulevard B210
77836 Rheinmünster
Germany



Technical support:
+49 7229 1847- 0

Function description

APCI-/CPCI-1710

Multifunction counter board
- Incremental encoder, Pulse encoder -

Edition: 05.02 – 05/2008

Product information

This manual contains the technical installation and important instructions for correct commissioning and usage, as well as production information according to the current status before printing. The content of this manual and the technical product data may be changed without prior notice. ADDI-DATA GmbH reserves the right to make changes to the technical data and the materials included herein.

Warranty and liability

The user is not permitted to make changes to the product beyond the intended use, or to interfere with the product in any other way.

ADDI-DATA shall not be liable for obvious printing and phrasing errors. In addition, ADDI DATA, if legally permissible, shall not be liable for personal injury or damage to materials caused by improper installation and/or commissioning of the board by the user or improper use, for example, if the board is operated despite faulty safety and protection devices, or if notes in the operating instructions regarding transport, storage, installation, commissioning, operation, thresholds, etc. are not taken into consideration. Liability is further excluded if the operator changes the board or the source code files without authorisation and/or if the operator is guilty of not monitoring the permanent operational capability of working parts and this has led to damage.

Copyright

This manual, which is intended for the operator and its staff only, is protected by copyright. Duplication of the information contained in the operating instructions and of any other product information, or disclosure of this information for use by third parties, is not permitted, unless this right has been granted by the product licence issued. Non-compliance with this could lead to civil and criminal proceedings.

ADDI-DATA software product licence

Please read this licence carefully before using the standard software. The customer is only granted the right to use this software if he/she agrees with the conditions of this licence.

The software must only be used to set up the ADDI-DATA boards.

Reproduction of the software is forbidden (except for back-up and for exchange of faulty data carriers). Disassembly, decompilation, decryption and reverse engineering of the software are forbidden. This licence and the software may be transferred to a third party if this party has acquired a board by purchase, has agreed to all the conditions in this licence contract and the original owner does not keep any copies of the software.

Trademarks

- ADDI-DATA is a registered trademark of ADDI-DATA GmbH.
- Turbo Pascal, Delphi, Borland C, Borland C++ are registered trademarks of Borland Insight Company.
- Microsoft C, Visual C++, Windows XP, 98, Windows 2000, Windows 95, Windows NT, EmbeddedNT and MS DOS are registered trademarks of Microsoft Corporation.
- LabVIEW, LabWindows/CVI, DasyLab, Diadem are registered trademarks of National Instruments Corp.
- CompactPCI is a registered trademark of PCI Industrial Computer Manufacturers Group.
- VxWorks is a registered trademark of Wind River Systems Inc.

WARNING

The following risks result from improper implementation and from use of the board contrary to the regulations:



- ◆ Personal injury
- ◆ Damage to the board, PC and peripherals
- ◆ Pollution of the environment

◆ **Protect yourself, the others and the environment!**

◆ **Read carefully the safety precautions (yellow leaflet).**

If this leaflet is not with the documentation, please contact us and ask for it.

◆ **Observe the instructions of the manual.**

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

◆ **Used symbols:**



IMPORTANT!

designates hints and other useful information.



WARNING!

It designates a possibly dangerous situation.

If the instructions are ignored the board, PC and/or peripheral may be destroyed.

1	DEFINITION OF APPLICATION	8
1.1	Intended use	8
1.2	Usage restrictions.....	8
1.3	Technical description	8
1.4	Function description	9
1.5	Used abbreviations	9
2	INCREMENTAL ENCODER.....	10
2.1	Function description	10
2.1.1	Block diagram of the counter	10
2.1.2	Typical applications	11
2.2	Used signals	12
2.3	Pin assignment of the incremental encoder.....	13
2.4	Connection example: Encoder ROD420	14
2.5	I/O mapping	15
2.6	Description of the I/O function	16
2.6.1	MODE-Register 1 (Base + 20).....	17
	L = Low, H = HighQuadruple mode	19
	Quadruple mode	20
	Double mode	21
	Simple mode	22
	Hysteresis	23
	Direct mode	23
2.6.2	Mode register 2 (Base+20)	24
2.6.3	Strobe-Register (Base + 0)	25
	Latch logic	26
2.6.4	Interrupt-Register (Base + 0).....	27
2.6.5	Test-Register (Base + 16).....	27
	Clear-Logic	29
	Reset-Logik.....	29
	Loading the counter chains (BASE + 4 → BASE + 12).....	30
2.6.6	INDEX-Register (Base + 12)	30
2.6.7	OVERFLOW-Register (Base + 16).....	30
2.6.8	STATUS-Register (Base + 24)	30
2.6.9	Filter register (BASE +60)	31
2.6.10	Version register	33
2.7	Limit values.....	33
3	STANDARDSOFTWARE.....	34
3.1	Define values.....	34
3.2	Interruptmask	35

3.3	Counter initialisation.....	37
	1) i_APCI1710_InitCounter(...)	37
	2) i_APCI1710_CounterAutoTest (...)	42
	3) i_APCI1710_ClearCounterValue (...)	44
	4) i_APCI1710_ClearAllCounterValue (...)	46
	5) i_APCI1710_SetInputFilter (...)	48
3.4	Reading the counter	51
	1) i_APCI1710_LatchCounter (...)	51
	2) i_APCI1710_ReadLatchRegisterStatus (...)	53
	3) i_APCI1710_ReadLatchRegisterValue (...)	55
	4) i_APCI1710_EnableLatchInterrupt (...)	57
	5) i_APCI1710_DisableLatchInterrupt (...)	59
	6) i_APCI1710_Read16BitCounterValue (...)	61
	7) i_APCI1710_Read32BitCounterValue (...)	63
3.5	Writing into the counter.....	65
	1) i_APCI1710_Write16BitCounterValue (...)	65
	2) i_APCI1710_Write32BitCounterValue (...)	67
3.6	Index	69
	1) i_APCI1710_InitIndex (...)	69
	2) i_APCI1710_EnableIndex (...)	73
	3) i_APCI1710_DisableIndex (...)	75
	4) i_APCI1710_GetIndexStatus (...)	77
	5) i_APCI1710_SetIndexAndReferenceSource (...)	79
3.7	Reference.....	81
	1) i_APCI1710_InitReference (...)	81
	2) i_APCI1710_GetReferenceStatus (...)	83
3.8	UAS, CB, U/D#, external pulse (strobe).....	85
	1) i_APCI1710_GetUASStatus (...)	85
	2) i_APCI1710_GetCBStatus (...)	87
	3) i_APCI1710_GetUDStatus (...)	89
	4) i_APCI1710_GetInterruptUDLatchedStatus (...)	91
	5) i_APCI1710_InitExternalStrobe (...)	93
3.9	Compare logic.....	95
	1) i_APCI1710_InitCompareLogic (...)	95
	2) i_APCI1710_EnableCompareLogic (...)	97
	3) i_APCI1710_DisableCompareLogic (...)	99
3.10	Frequency measurement	101
	1) i_APCI1710_InitFrequencyMeasurement (...)	101
	2) i_APCI1710_EnableFrequencyMeasurement (...)	105
	3) i_APCI1710_ReadFrequencyMeasurement (...)	107
	4) i_APCI1710_DisableFrequencyMeasurement (...)	110
3.11	Digital output	112
	1) i_APCI1710_SetDigitalChlOn (...)	112

2) i_APCI1710_SetDigitalChlOff (...)	114
--------------------------------------	-----

3.12 Using functions in the kernel module.....116

1) i_APCI1710_KRNL_ClearCounterValue (...)	116
2) i_APCI1710_KRNL_Read16BitCounterValue (...)	118
3) i_APCI1710_KRNL_Read32BitCounterValue (...)	120
4) i_APCI1710_KRNL_Write16BitCounterValue (...)	122
5) i_APCI1710_KRNL_Write32BitCounterValue (...)	124
6) i_APCI1710_KRNL_GetInterruptUDLatchedStatus (...)	126
7) i_APCI1710_KRNL_ReadFrequencyMeasurement (...)	128
8) i_APCI1710_KRNL_SetDigitalChlOn (...)	130
9) i_APCI1710_KRNL_SetDigitalChlOff (...)	132

Tables

Table 1-1: Delivered manuals	9
Table 2-1: Used signals.....	12
Table 3-1: Define table	34
Table 3-2: Interruptmask of the function "incremental counter"	35
Table 3-3: Counter value: Return table	36
Table 3-4: Counter range.....	38
Table 3-5: Counter operation mode.....	38
Table 3-6: Counter option for quadruple/double/single mode	39
Table 3-7: Counter option for direct mode	39
Table 3-8: Autotest return	42
Table 3-9: Filter time	49
Table 3-10: Index action	69
Table 3-11: Selection of the index and reference logic source	79
Table 3-12: Reference level	81
Table 3-13: External pulse level	93
Table 3-14: Value of the time base.....	102
Table 3-15: Status of the counter operation.....	107

Figures

Fig. 2-1: Block diagram of the counter	11
Fig. 2-2: Pin assignment of the front connector	13
Fig. 2-3: Mode register	17
Fig. 2-4: Edge analysis circuit (quadruple mode).....	20
Fig. 2-5: Edge analysis circuit (double mode).....	21
Fig. 2-6: Edge analysis circuit (single mode)	22
Fig. 2-7: Edge analysis circuit of the hysteresis (example in quadruple mode)	23
Fig. 2-8: Pulse diagram of the gate measuring.....	23
Fig. 2-9: Pulse diagram of the latch logic	26
Fig. 2-10: Pulse diagram of the clear logic	29

1 DEFINITION OF APPLICATION

1.1 Intended use

The board **APCI-1710** must be inserted in a PC with PCI 5V/32-bit slots, which is used as electrical equipment for measurement, control and laboratory pursuant to the norm IEC 61010-1.

The board **CPCI-1710** must be inserted in a CompactPCI system with PCI 5V/32-bit slots, which is used as electrical equipment for measurement, control and laboratory pursuant to the norm IEC 61010-1

1.2 Usage restrictions

The board **APCI-/CPCI-1710** must not be used as safety related part for securing emergency stop functions

The board **APCI-/CPCI-1710** must not be used in potentially explosive atmospheres.

1.3 Technical description

This manual refers to the **APCI-1710** as well as to the **CPCI-1710** board. Make sure that you have received the following items:

- The CD 1 "Standard Software Drivers" with the ADDISET parameterizing program and the required software drivers.
- The CD 2 "Technical Manuals". This CD contains the following:

- 1) The technical description **ADDICOUNT APCI-1710 / CPCI-1710: Function-programmable counter board for the PCI bus** (containing general information on the operation of the board)
- 2) A function description for each function which you want to program on the board
- 3) The yellow leaflet "Safety precautions"

According to the used function you will find the required assignment and programming functions in the different manuals for each function:

Table 1-1: Delivered manuals

Function	PDF file (CD2 technical manuals)		Function description in SET1710	CFG file
	German	English		
Incremental counter	Inkr_zähler_d.pdf	Incr_counter_e.pdf	Incremental counter	inc_cpt.cfg
SSI	SSI_d.pdf	ssi_e.pdf	SSI	ssi.cfg
SSI monitor	SSI-Monitor_d	SSIMonitor_e.pdf	SSI_Monitor	ssi_mon.cfg
Chronos	chronos_d.pdf	chronos_e.pdf	Chronos	chronos.cfg
Counter/timer	Zähler_timer_d.pdf	Counter_timer_e.pdf	counter/timer	82x54.cfg
TOR	TOR_d.pdf	TOR_e.pdf	TOR	tor.cfg
PWM	PWM_d.pdf	PWM_e.pdf	Pulse width modulation	PWM.cfg
TTL	TTL_IO_d.pdf	TTL_IO_e.pdf	TTL I/O	tll_io.cfg
Digital I/O	dig_EA_d.pdf	dig_IO_e.pdf	Digital I/O	dig_IO.cfg
Pulse counter	Impulszähler_d.pdf	pulseCounter_e.pdf	Pulse counter	imp_cpt.cfg
ETM (Edge time measurement)	ETM_d.pdf	ETM_e.pdf	Edge time measurement	etm.cfg

Please note:

The board **CPCI-1710** is compatible with the board APCI-1710 as far as the installation of the software is concerned. The ADDIREG and SET1710 programs make no difference between PCI and CompactPCI boards. The API functions of the standard software are also identical.

1.4 Function description

Apart from a global description of the functions this manual contains:

- the pin assignment of the front connector
- a list of the used signals
- the I/O mapping
- a chapter about the API software functions of the standard software.

1.5 Used abbreviations

The signals on the 50 pin SUB-D connector refer always to one function module.

Please note the used abbreviations:

- UAS: Interference signal
- CLK: Clock
- REF: Reference point logic
- ENA: Enable

C1+ is a signal for **function module 1**.

2 INCREMENTAL ENCODER

2.1 Function description

The function "incremental encoder" is a fast counter for 90° phase-shifted signals (displacement measurement systems).

It is used especially for applications, in which high precision and reliability for tough industrial applications is required.

Properties:

- Connection of 1 or 2 incremental encoder(s), configurable as follows:
 - 1 channel with a counter depth of 32 bit, for TTL or differential incremental encoders at SUB-D front connector.
 - 2 channels with a counter depth of 16 bit for TTL or differential incremental encoders at SUB-D connector (for this configuration the index logic is not possible).
- 1 "INDEX" input for reference point logic.
- 1 "UAS" input, which can be used as interference input.
- 1 "REF" input as common digital input or for reference point logic
- 2 "EXTSTB" inputs, which allow the latching of counter values
- High counter speed
- Interrupt indication, triggered through the external strobe inputs
- Integrated test operation
- Pulse width measurement through direct inputs, programmable counter direction

Function range of the counter module:

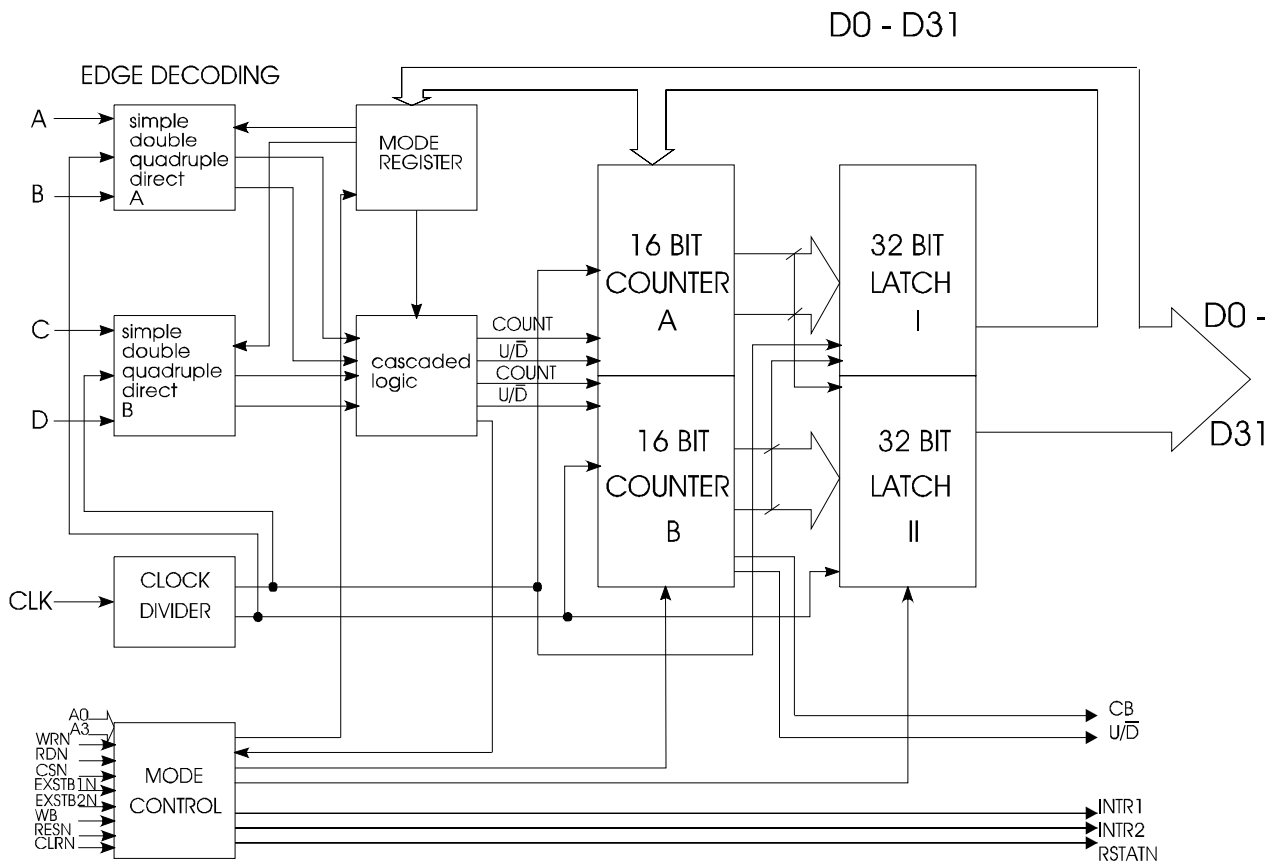
- Quadruple, double or single analyses of two-phase shift clock pulses. Further processing in 32-bit-loadable upwards or downwards counter (or a 16-bit counter)
- Direction recognition for upwards or downwards counting.
- Hysteresis switch for the absorption of the first pulse after a change in rotation; switchable
- Two 32-bit data latches, individually programmable for internal/external strobe, latch-strobe synchronized with internal clock.
- Operation mode definition via internal MODE- register, loadable / readable via data bus
- Strobe-inputs, which can be triggered either through 2 external pins or register description.

2.1.1 Block diagram of the counter

The counter module contains:

- 2 independent 16-bit counters, internally cascadable
- Simple, double, quadruple edge analysis circuits of 2 phase shifted clock pulses
- Direction discriminators
- Hysteresis circuits
- 2 x 32-bit latches
- function and control logic

Fig. 2-1: Block diagram of the counter



2.1.2 Typical applications

- Acquisition of incremental displacement measuring systems
- X, Y, Z control
- Pulse width and frequency measurement
- Incremental encoder acquisition
- Velocity measurement
- Rotation measurement
- Tolerance measurement
- Electronic mouse

2.2 Used signals

The function “incremental encoder” occupies **7 inputs (channel A to G)** and **1 output (channel H)** of the respecting function module of the **APCI-/CPCI-1710**.

On one board you can connect max. 4 x 32-bit or 8 x 16-bit incremental encoders.

Table 2-1: Used signals

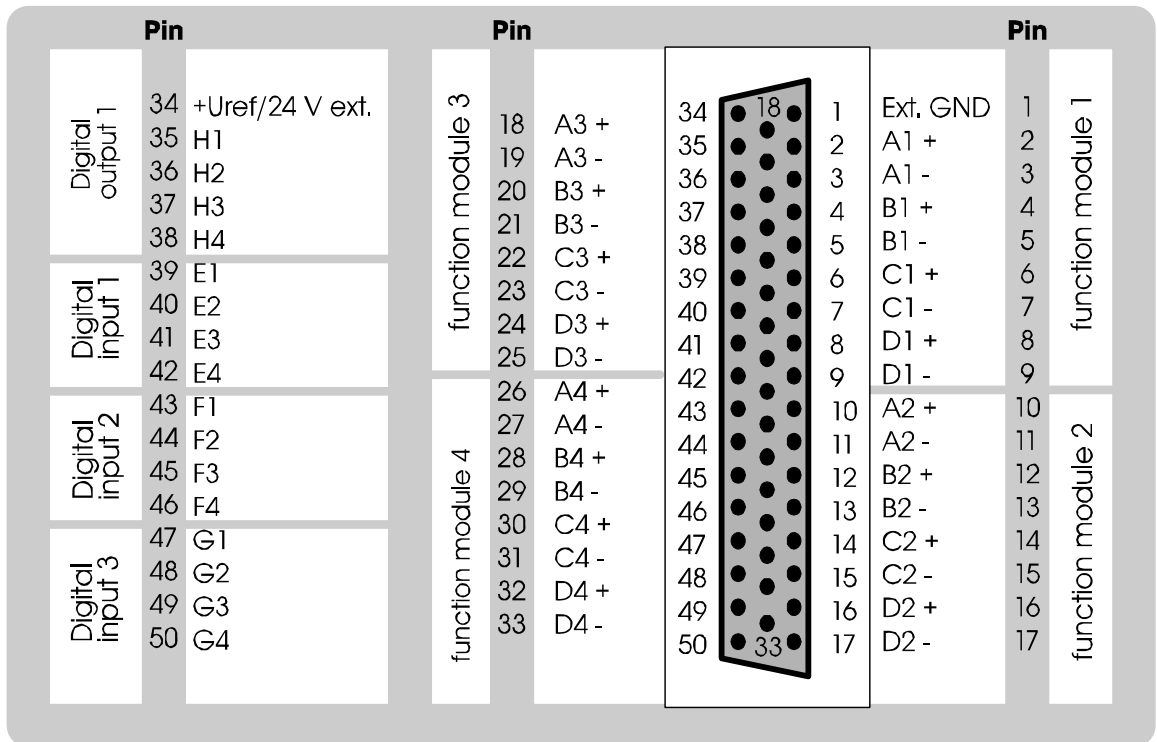
SIGNALS	AT THE CONNECTOR	POLARITY	FUNCTION
A	Ax +/-	Diff./TTL, optional 24V	Track A of the 1 st incremental encoder
B	Bx +/-	Diff./TTL, optional 24V	Track B of the 1 st incremental encoder
INDEX	Cx +/-	Diff./TTL, optional 24V	INDEX track of the incremental encoder when in 32-bit mode.
C			Track A of the 2 nd incremental encoder in 16-bit mode.
AUS	Dx +/-	Diff./TTL, optional 24V	Interference signal in 32-bit mode.
D			Track B of the 2 nd incremental encoder in 16-bit mode
REF	Ex	24V, optional 5V	Normal digital input, can be read over the register; can effect the reference point logic.
ExtStrb_a	Fx	24V, optional 5V Aktiv High	Digital input that causes the latching of the counter contents into the first latchregister. Can also generate an interrupt
ExtStrb_b	Gx	24V, optional 5V Aktiv High	Digital input that causes the latching of the counter contents into the the second latchregister; Can also generate an interrupt.
Output	Hx		Common digital output

x: Number of the function module

2.3 Pin assignment of the incremental encoder

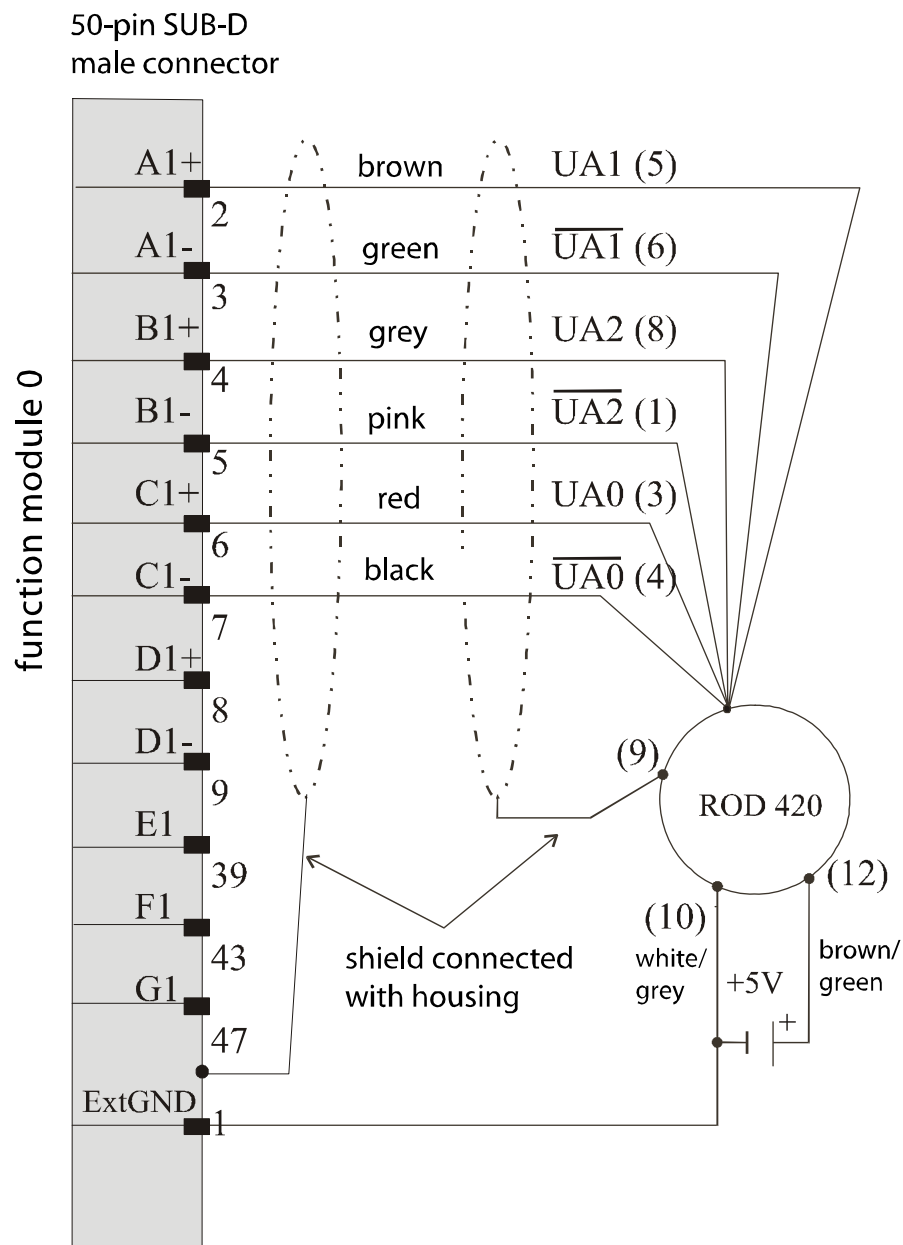
The figure below is a connection example. The function “incremental encoder” is implemented on all function modules.

Fig. 2-2: Pin assignment of the front connector



2.4 Connection example: Encoder ROD420

The encoder ROD420 (HEIDENHAIN) is connected to the function module 1 of the **APCI-/CPCI-1710** board. Differential signal for incremental information and reference signals.



2.5 I/O mapping

IORD				
	D31...D24	D23...D16	D15.....D8	D7.....D0
BYTES	HIGHBYTE	MIDHIGHBYTE	MIDLOWBYTE	LOWBYTE
BASEx + 0	y	y	Y	INT-REG
BASEx + 4	LAT1.3	LAT1.2	LAT1.1	LAT1.0
BASEx + 8	LAT2.3	LAT2.2	LAT2.1	LAT2.0
BASEx + 12	y	y	Y	INDEX-REG
BASEx + 16	y	y	Y	OVERFLOW-REG
BASEx + 20	y	y	MODREG2	MODREG1
BASEx + 24	y	y	Y	STATUS-REG
.....				
BASEx + 60	FUNKNBR2	FUNKNBR1	REVBYTE2 or FILTER CONFIG	REVBYTE1 or FILTER VALUE

-: No function; y: Data of no importance,
 x: Number of the function module.

IOWR				
	D31...D24	D23...D16	D15.....D8	D7.....D0
BYTES	HIGHBYTE	MIDHIGHBYTE	MIDLOWBYTE	LOWBYTE
BASEx + 0	-	-	-	STB-REG
BASEx + 4	COUNT.3	COUNT.2	COUNT.1	COUNT.0
BASEx + 8	y	y	COUNT.1	COUNT.0
BASEx + 12	COUNT.3	COUNT.2	Y	y
BASEx + 16	-	-	-	TEST-REG
BASEx + 20	-	-	MODREG2	MODREG1
.....	-	-	-	-
BASEx + 60	-	-	FILTER CONFIG	FILTER VALUE

-: No function; y: Data of no importance,
 x: Number of the function module.

In the 32-bit mode the counter (Ax and Bx) is loaded over the address BASE_x+4 .

In 16-bit mode:

- the first counter (Ax and Bx) is loaded over the address BASE_x+8
- the second counter (Cx and Dx) is loaded over the address BASE_x+12

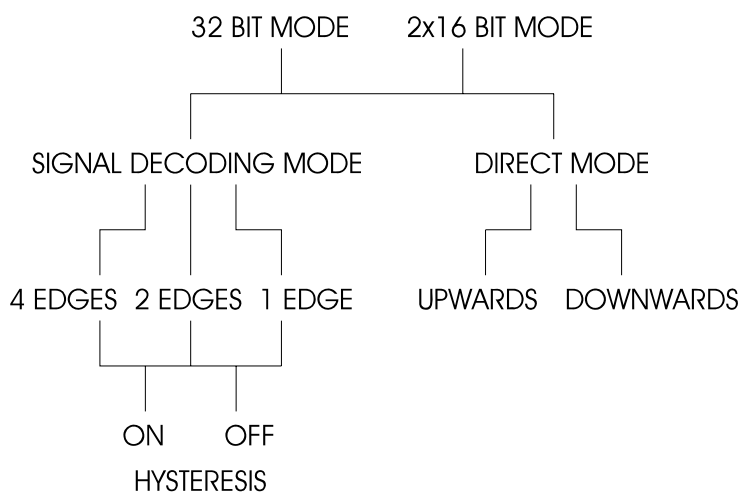
The incremental counter function occupies 7 DWORDS in the I/O range of the function module x.

2.6 Description of the I/O function

General description

The counter function is determined by **MOD-REG1 (Base + 20)**. The register is writeable and readable.

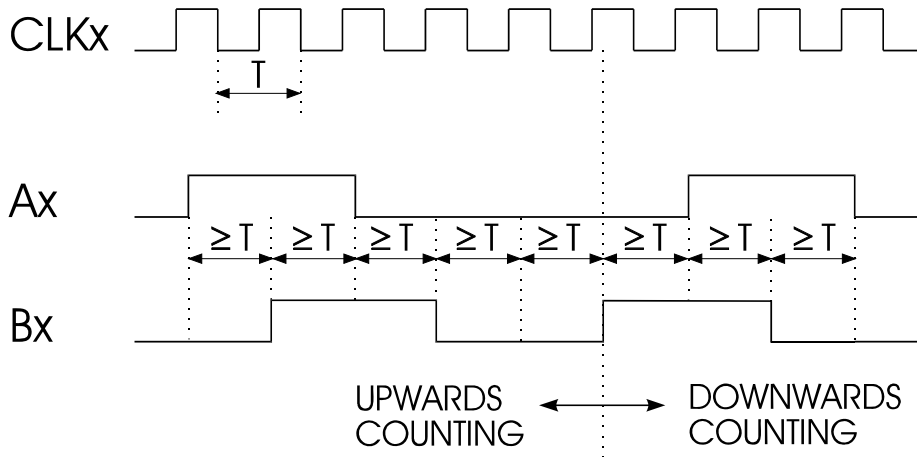
The following operating modes are possible:



Inputs for edge analysis circuit

- A, B Inputs for phase-shifted signals for 32-bit counter or 16-bit counter A in quadruple, double, simple mode.
 If signal A changes before signal B, the counter counts upwards
 If signal B changes before signal A, the counter counts downwards
- C, D Inputs for phase-shifted signals for 16-bit counter B in quadruple, double, simple mode.
 If signal C changes before signal D, the counter counts upwards
 If signal D changes before signal C, the counter counts downwards

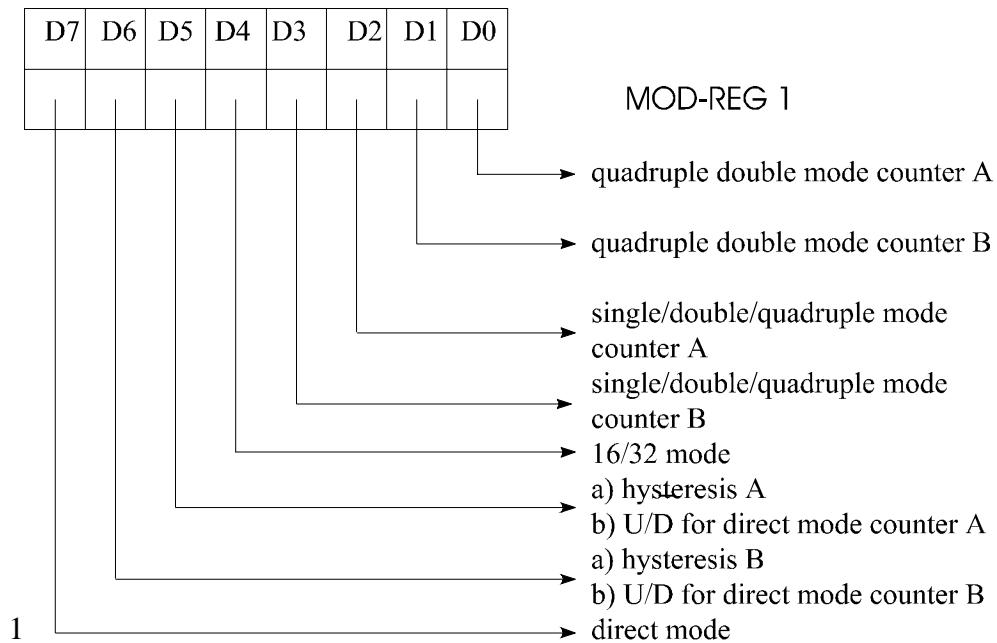
The edge analysis circuit requires 3 clock pulse periods to analyse the edges or for the direction recognition. Counting occurs with the 4th clock pulse (concerns all the operating modes related to the external negative clock pulse edge). The clock period is the same as the PCI clock period i.e. ≤ 33 MHz.



This figure shows that no defined phase shifts between the signals A, B and C, D are necessary. But minimum values which are temporarily related to the system clock pulse have to be respected (minimal edge distance).

2.6.1 MODE-Register 1 (Base + 20)

Fig. 2-3: Mode register



For the single MODE you have to program also the double MODE. Otherwise the edge analysis of the corresponding control unit (A/B) would be internally deactivated.

Independently from the analysis inputs, the counting is not processing any longer, and the last counting value is stored.1) 16/32-bit mode

The 16/32-bit mode is determined by the data bit D4.

Mode Register 1	Bit	7	6	5	4	3	2	1	0
32-bit mode		X	X	X	0	X	X	X	X
16-bit mode		X	X	X	1	X	X	X	X

In the 32-bit MODE either the quadruple/double/single edge analysis circuit A or the DIRECT MODE is used. CBX is the CARRY/BORROW output in 32 bits.

On U/D the upwards/downwards signal recognised in the edge analysis circuit A is available.

In the 16-bit MODE you can either use:

- the quadruple/double/single edge analysis circuit A for counter A
- the quadruple/double /single edge analysis circuit B for counter B
- or the DIRECT-MODE (inputs A, B for counter A, inputs C, D for counter B).

CBX is the CARRY/BORROW output in 16-bit, counter B.

On U/D the upwards/downwards signal recognised in the edge analysis circuit B is available.

Quadruple, double or simple mode is determined by the data bits D0-D3 in the mode register 1.

L to D0	Quadruple mode for edge analysis circuit A
L to D1	Quadruple mode for edge analysis circuit B
H to D0	Double mode for edge analysis circuit A
H to D1	Double mode for edge analysis circuit B
L to D2, D3	Quadruple or double mode for edge analysis A, B dependent from D0 and D1
H to D0 and D2, D1 and D3	Simple mode for edge analysis circuit B
L to D0 and D1, H to D2 and D3	Edge analysis circuit A, B deactivated

L = Low, H = High

Quadruple mode

In the quadruple MODE, the edge analysis circuit generates a counting pulse from each edge of 2 signals which are phase-shifted in relation to each other.

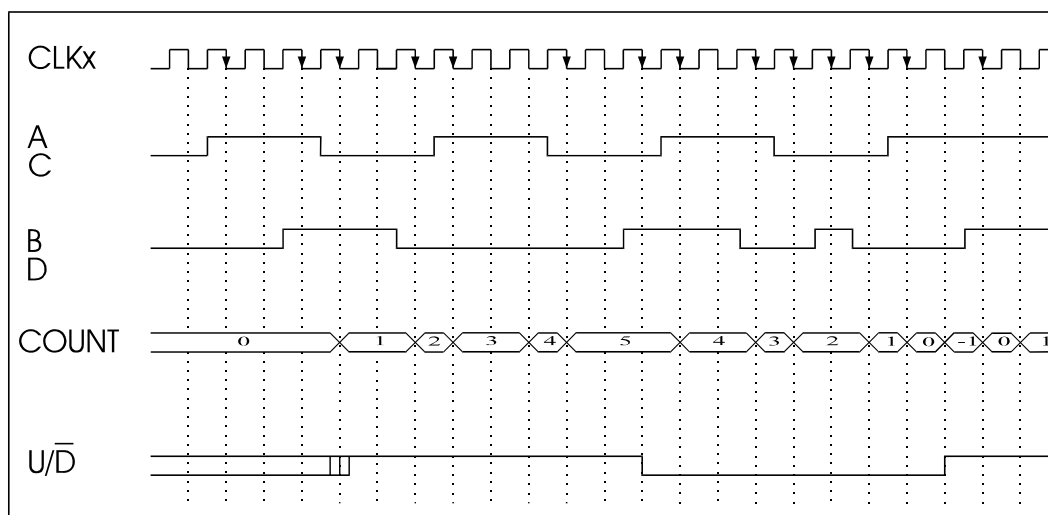
In the 32-bit MODE these signals have to be present at the inputs A and B.

In the 16-bit MODE, two edge analysis circuits are available for inputs A and B or C and D.

The signals A, B and C, D are sampled by the common clock pulse CLKX and buffered. This clock pulse must correspond to at least 4 Times the frequency of signals A, B, C, D.

The edge analysis circuit recognises the counting direction from the phase shift of signals A to B or C to D.

Fig. 2-4: Edge analysis circuit (quadruple mode)



Sampling (A, B, C, D) with CLKx ↓
 Counting with CLKx ↓

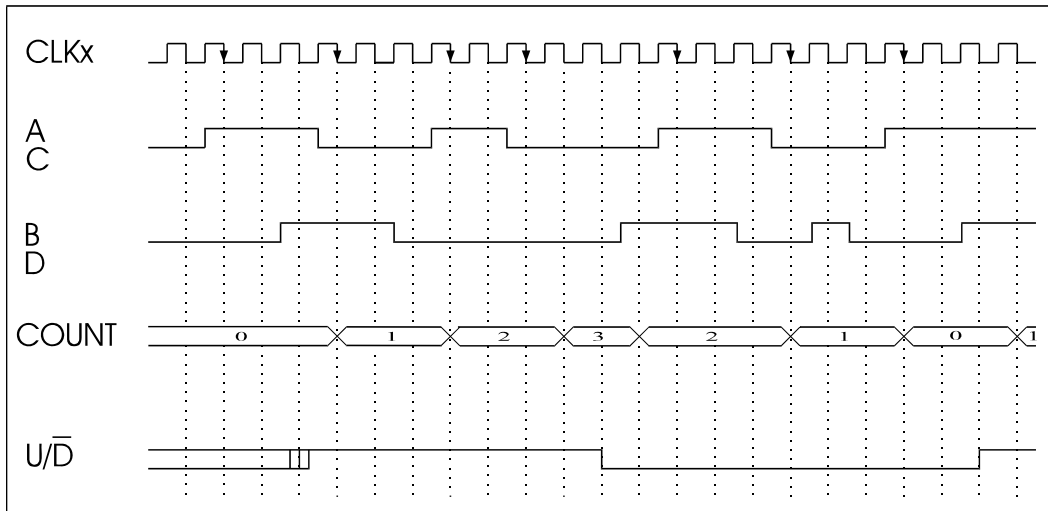
MODE Register 1	Bit	7	6	5	4	3	2	1	0
32-bit mode		0	X	0	0	X	0	X	0
16-bit mode		0	0	0	1	0	0	0	0

In this case, the hysteresis is switched off.

Double mode

Same as the quadruple MODE, except that only two of the four edges are analysed per period.

Fig. 2-5: Edge analysis circuit (double mode)



Sampling (A, B, C, D) with CLKx ↓
 Counting with CLKx ↓

MODE
Register 1
 32-bit mode
 16-bit mode

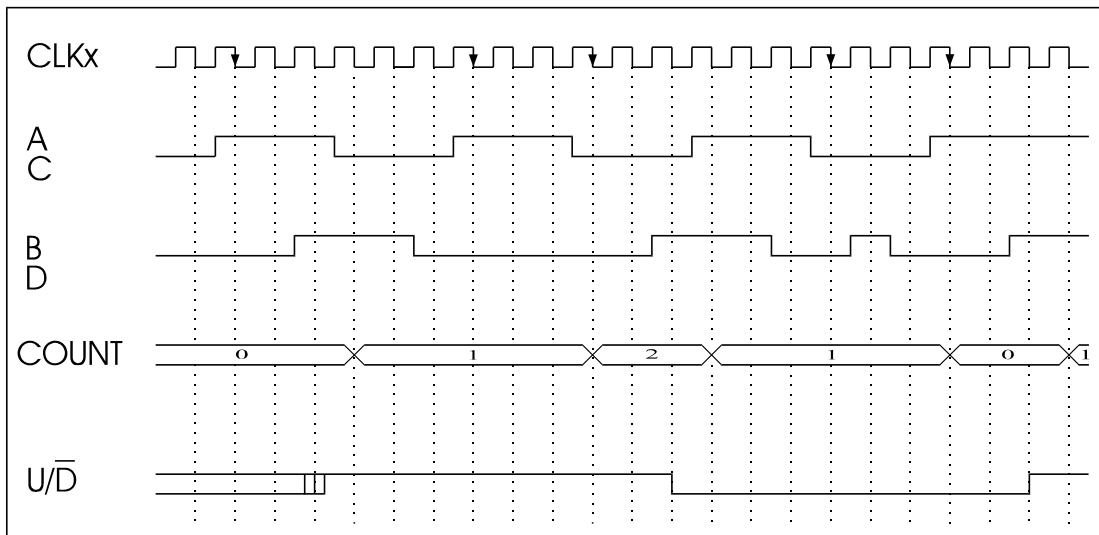
Bit	7	6	5	4	3	2	1	0
	0	X	0	0	X	0	X	1
	0	0	0	1	0	0	1	1

The hysteresis is switched off.

Simple mode

Same as the quadruple MODE, except that only one of the four edges are analysed per period.

Fig. 2-6: Edge analysis circuit (single mode)



Sampling (A, B, C, D) with CLKx ↓
 Counting with CLKx ↓

MODE

Register 1

32-bit mode

16-bit mode

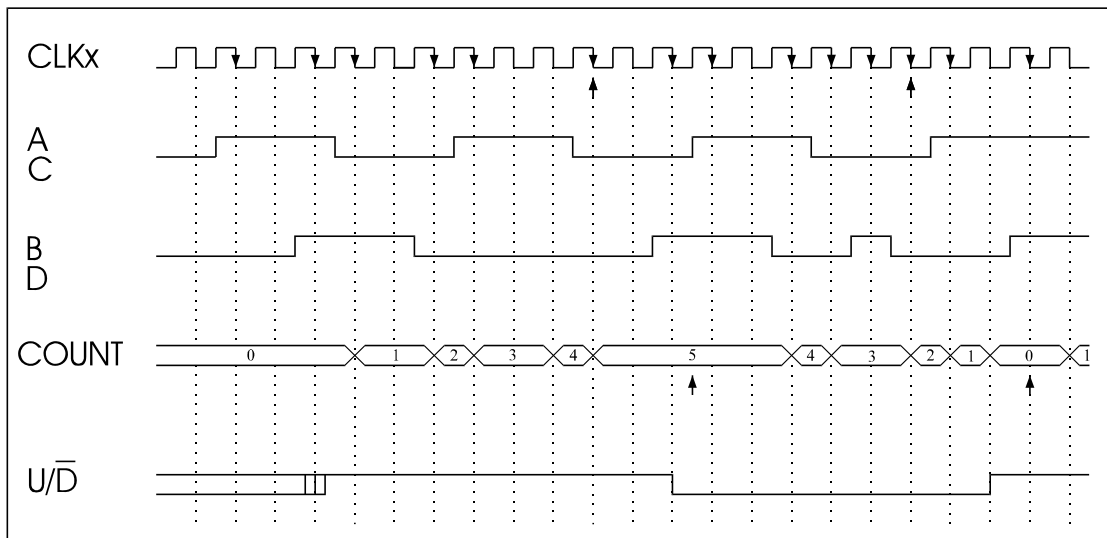
Bit	7	6	5	4	3	2	1	0
32-bit mode	0	X	0	0	X	1	X	1
16-bit mode	0	0	0	1	1	1	1	1

the hysteresis is switched off.

Hysteresis

One hysteresis circuit is available in each of both circuits. It suppresses the first counting pulse after each change of rotation.

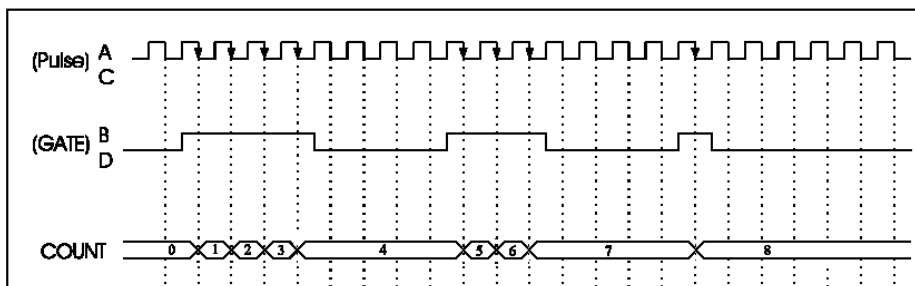
Fig. 2-7: Edge analysis circuit of the hysteresis (example in quadruple mode)



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. The 32-bit counter or both 16-bit counters can be independently set to upwards or downwards counting through bit D5 or D6 of the MODE register 1. The gate input is synchronised with the falling edge of A or C. If the gate input is always set to the high level, the counter counts the pulses by input A or C. This is the case when the differential inputs B and D are open.

Fig. 2-8: Pulse diagram of the gate measuring



Sampling (B, D) with A↓, B↓ (CLK = 0)
 CLK ≅ CLKX

MODE Register 1	Bit	7	6	5	4	3	2	1	0
32-bit mode		1	X	1	0	X	X	X	X
16-bit mode		1	0	1	1	X	X	X	X

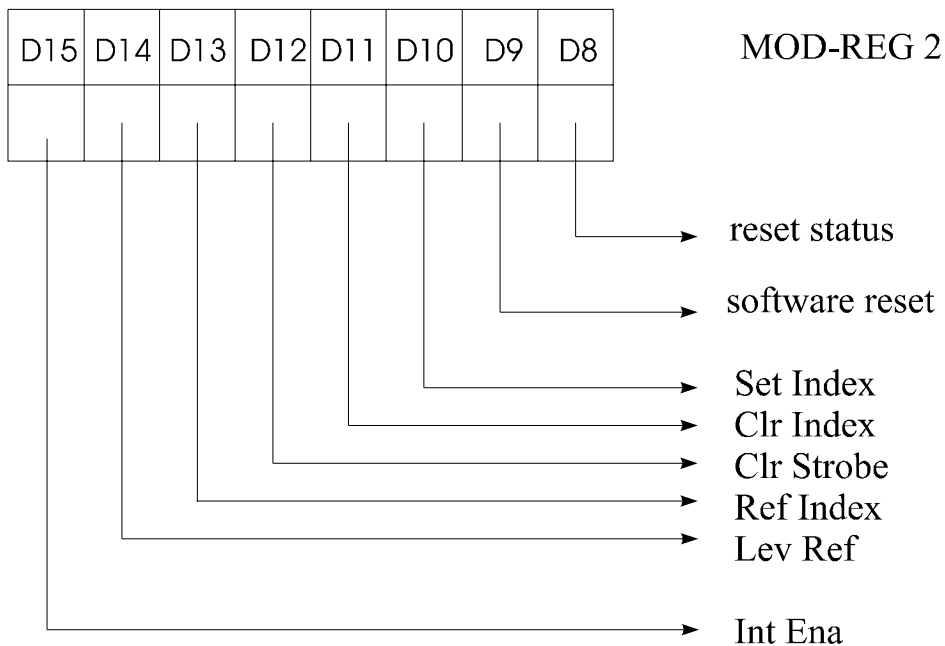
Counter A counts upwards, Counter B counts downwards (16-bit MODE).



WARNING!

A mixed mode with counter **A** in quadruple/double/single MODE and counter **B** in DIRECT MODE is **not possible!**

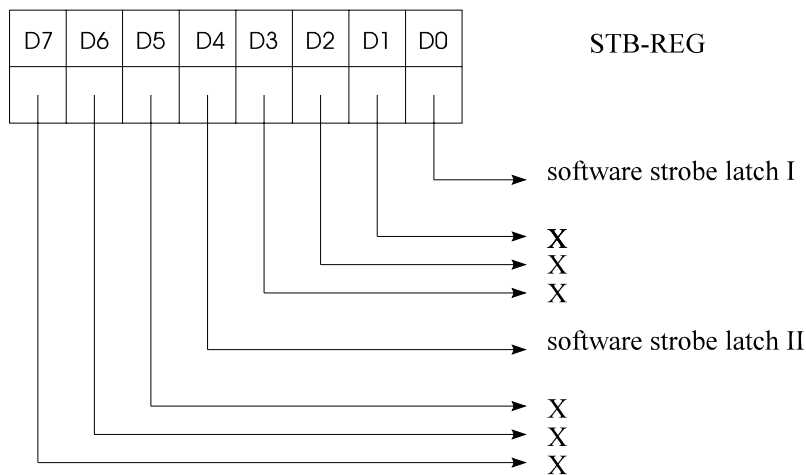
2.6.2 Mode register 2 (Base+ 20)



MODE-Register 2 allows the resolution and display of the RESET-state module (see also reset logic).

- Bit 8: **RESET-STATUS**
 0: If an external reset has occurred.
 1: Inactive (no meaning)
- Bit 9: **SOFTWARE-RESET**
 1: Start internal reset operation.
 0: Inactive (automatic resetting after reset has occurred)
- Bit 10: **SET-INDX**
 1: Erase or strobe of the counter at the reference point.
 0: Inactive (automatic resetting after reset has occurred)
- Bit 11: **CLR-INDX**
 1: SET-INDX is reset automatically after the reset has occurred
 0: Inactive: SET-INDX is not reset.
- Bit 12: **CLR-STRB**
 1: Counter value is written into latchregister1 when the reference point is reached.
 0: Inactive; Counter is deleted when reaching the reference point.
- Bit 13: **REF-INDX**
 1: REF impacts the reference point logic. (INDEX must be on "1" **AND** the respecting level on REF so that the reference point logic is recognised.
 0: Inactive: REF input has no impact on the reference point logic.
- Bit 14: **LEV-REF**
 1: REF must be on "0".
 0: inactive: REF must be on "1".
- Bit 15: **INT-ENA**
 1: Loop on strobe interrupt
 0: (Reset) interrupt is locked.

2.6.3 Strobe-Register (Base + 0)



A software strobe is generated by writing on the respecting databit of the strobe register. The strobe cycle is triggered with value 1 of the respecting databit. The value 0 is of no importance.

A hardware strobe is generated through the both Pins Fw F (ExtStb_a, low active for Latch I) and G (ExtStb_b, low active for Latch II).

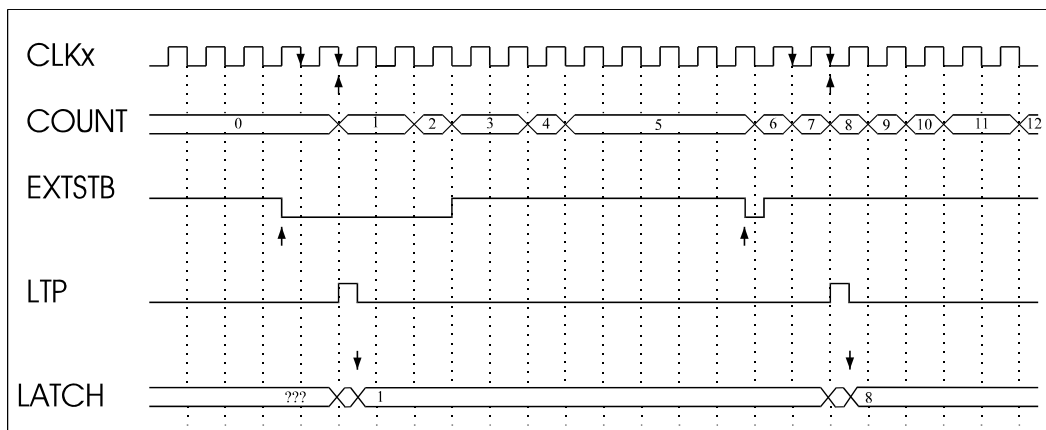
Now the strobe register is writeable.

Latch logic

It consists in two equivalent 32-bit data latch registers. These registers latch the current 32-bit counts through an own software strobe and/or external strobe input.

If internal counting pulses simultaneously occur all strobe signals are synchronised. Thus, intermediate states are not stored. Through synchronising the counting value is stored after two clock pulses.

Fig. 2-9: Pulse diagram of the latch logic

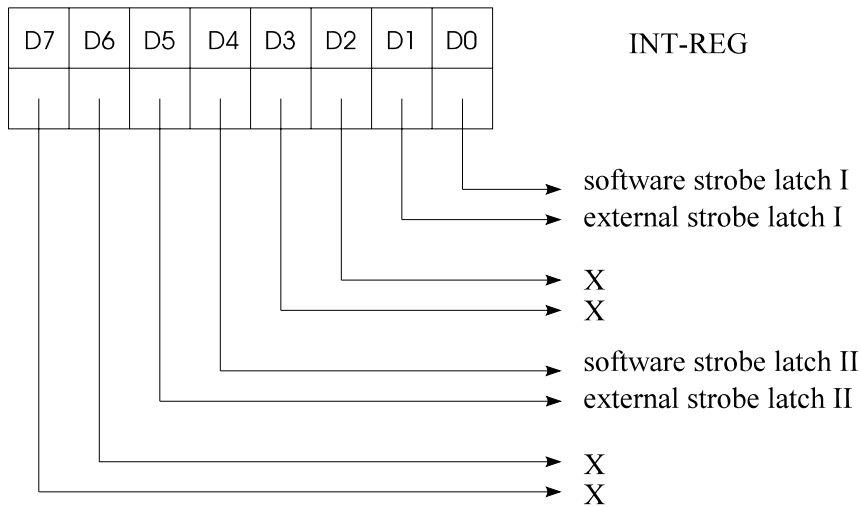


LTP: internal latch pulse
 Counting (COUNT) with CLKX↓

Selection: (See also I/O mapping 16-bit)

- LAT1.0 → BIT 0-7 LATCH I LAT2.0 → BIT 0-7 LATCH II
- LAT1.1 → BIT 8-15 LATCH I LAT2.1 → BIT 8-15 LATCH II
- LAT1.2 → BIT 16-23 LATCH I LAT2.2 → BIT 16-23 LATCH II
- LAT1.3 → BIT 24-31 LATCH I LAT2.3 → BIT 24-31 LATCH II

2.6.4 Interrupt-Register (Base + 0)



The Interrupt register contains the interrupt state and can be read only.

When a value has been latched, the value 1 indicates an active state. In this case the latch group and the internal and external latch triggering are differentiated.

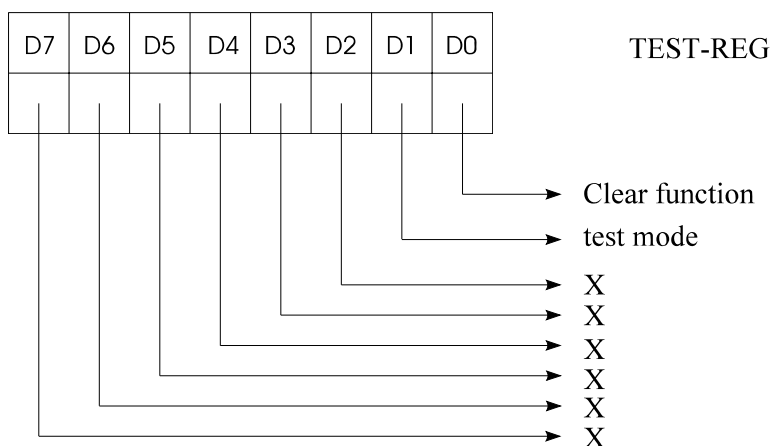
The D1 and D5 bit positions have the same meaning as interrupt request.

A display occurs only after triggering at the inputs and after the latch operation has ended. The D0 and D4 bits are active after a software strobe, in case the latch operation has ended internally.

The corresponding interrupt displays are reset by a read operation of the corresponding 32-bit latch group.

If a new strobe operation is triggered during the read operation, a new latch pulse is generated. But it is displayed neither in the interrupt register nor at the interrupt.

2.6.5 Test-Register (Base + 16)



The test register can only be written. The value 1 corresponds to the active state. The bit 0 is automatically reset.

The test mode is intended for testing the component and the connected peripheral. All 8-bit counter chains are internally operated as downwards counters. Independently from the external signals, all four 8-bit counter chains are decremented in parallel by each negative clock pulse edge of CLKX.

Once the test mode has been activated, the CLEAR function must be triggered in order to synchronise the counter chains.

After strobe operations have been triggered through the strobe register (set bit 0) or through the external pin EXSTBx1, all 4 bytes of a latch group must be identical when read at any Time.

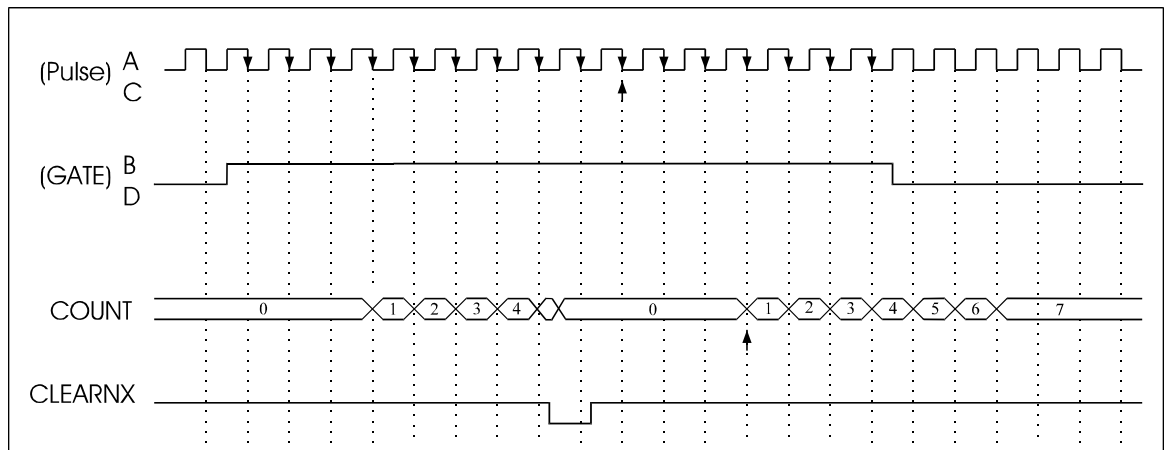
If the MODE register 1 is read during the test mode, this value is 10 HEX independently from the value determined. When the test mode is ended, the old unchanged value can be read.

The test mode is left either when a RESET operation has been triggered or when the value "0" is written in bit 1 of the test register. The clear function automatically resets the bit 0 of the test register after execution. A writing access is only possible in this bit.

Clear-Logik

At reference point logic an external clear signal at input CLRNX (= low) resets immediately all the counter chains. Removing this signal is synchronised with the complete clear operation.

Fig. 2-10: Pulse diagram of the clear logic



Sampling (A, D) with A ↓, B ↓ (CLK = 0)

After the trailing edge of the clear pulse, 3 other clock pulses are passed until the counter is enabled again.

With a software clear (see test register) the internal clear operation begins after the write operation has been ended (WRNX = High). It lasts 5 negative clock pulse edges.

This means that an internal counting operation can occur again at the 6. negative edge at the earliest.

Reset-Logik

After reset, the control logic of the circuit is set to the basic MODE.

- 32-bit counter
- Edge analysis circuit A in quadruple MODE
- Hysteresis off
- Control unit B disabled
- Bit 8/ mode register 2 on Low
- Do not affect the counter chains

The bit 8 MODE-register 2 remains as long active until this bit is set to "1" by the processor through a write operation.

The internal reset operation begins with a software reset (see MODE register 2) once the write operation is ended (WRNX = high). It is lasting 4 negative clock pulse edges (CLKX).

This means that data can be processed again by the control unit at the 5th negative clock pulse edge.

Loading the counter chains (BASE + 4 → BASE + 12)

The 16-/32-bit-counter can be loaded over the 32-bit databus.

- Through the address Base + 4 the 32-bit counter is loaded.
- Through the address Base + 8 the 16-bit counter A is loaded (bit D0-D15).
- Through the address Base + 12 the 16-bit counter B is loaded (bit D16-D31).

Selection:

- COUNT.0 → BIT 0-7 of the counter
- COUNT.1 → BIT 8-15 of the counter
- COUNT.2 → BIT 16-23 of the counter
- COUNT.3 → BIT 24-31 of the counter

In order to avoid counting errors, the counting process shall be interrupted during loading.

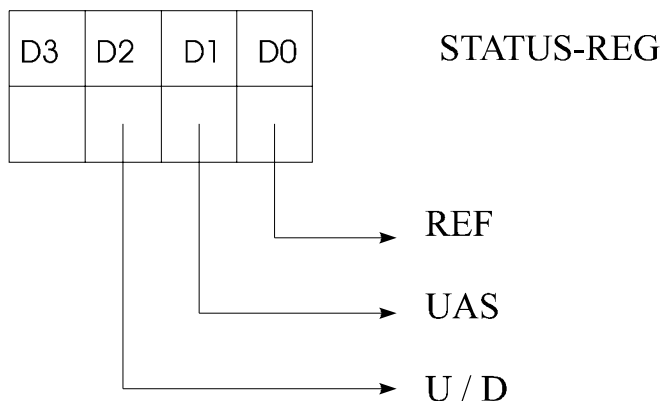
2.6.6 INDEX-Register (Base + 12)

D0 = INDX Bit = "1": an index occurred; when reading this value, the index is reset.
 "0": No index occurred.

2.6.7 OVERFLOW-Register (Base + 16)

D0 = C/B Bit = "1": An overflow or underflow is occurred. When reading this value, the bit is reset.
 "0": No overflow, the underflow is occurred.

2.6.8 STATUS-Register (Base + 24)



STATUS-REG:

D0 = REF Bit = "1": The input REF is set on logic "0"
 "0": The input REF is set on logic "1".

D1	= UAS Bit =	"1":	The input UAS is set on logic "0".
		"0":	The input UAS is set on logic "1".
D2	= U/D Bit =	"1":	The counter counts upwards (Up)
		"0":	The counter counts downwards (Down)

2.6.9 Filter register (BASE +60)

IMPORTANT!

i

The filter register can be called with the software function „i_APCI1710_InitInputFilter“. You can find a detailed description of this function in the technical description of the APCI-1710 or CPCI-1710 and is available for further function modules.

With the software function described in this manual (see 3.3), „i_APCI1710_SetInputFilter“, the filter also can be initialised. However, this software function is only available for the incremental counter and stems from former firmware versions.

By a simple reading access to Register 60, the version register, described in chapter **Fehler! Verweisquelle konnte nicht gefunden werden.** is read. By a writing access to Register 60, a digital filter can be parameterized for the inputs A, B, C, D. The set filter values refer to positive and negative pulses. All pulses that are smaller than the set time, will be filtered out. For reading back this range, Bit DQ 15 must be set. Then by a reading access to the register, the filter status can be read.

By the filter value “0000”, the filter is disabled. Bit DQ 8 indicates if a 40 MHz quartz is available on the used board. Over DQ9 the filter clock to be used can be set.

DQ3..0 : Filter values for the inputs A, B, C, D base 40 MHz

0	0000	=	Filter disabled
1	0001	=	Filter of 100 ns
2	0010	=	Filter of 150 ns
3	0011	=	Filter of 200 ns
4	0100	=	Filter of 250 ns
5	0101	=	Filter of 300 ns
6	0110	=	Filter of 350 ns
7	0111	=	Filter of 400 ns
8	1000	=	Filter of 450 ns
9	1001	=	Filter of 500 ns
10	1010	=	Filter of 550 ns
11	1011	=	Filter of 600 ns
12	1100	=	Filter of 650 ns
13	1101	=	Filter of 700 ns
14	1110	=	Filter of 750 ns
15	1111	=	Filter of 800 ns

DQ3..0 : Filter values for the inputs A, B, C, D Base 33 MHz

0	0000	=	Filter disabled
1	0001	=	Filter of 121 ns

2	0010	=	Filter of 182 ns
3	0011	=	Filter of 242 ns
4	0100	=	Filter of 303 ns
5	0101	=	Filter of 364 ns
6	0110	=	Filter of 424 ns
7	0111	=	Filter of 485 ns
8	1000	=	Filter of 545 ns
9	1001	=	Filter of 606 ns
10	1010	=	Filter of 667 ns
11	1011	=	Filter of 727 ns
12	1100	=	Filter of 788 ns
13	1101	=	Filter of 848 ns
14	1110	=	Filter of 909 ns
15	1111	=	Filter of 970 ns

DQ3..0 : Filter values for the inputs A, B, C, D base 30 MHz

0	0000	=	Filter disabled
1	0001	=	Filter of 133 ns
2	0010	=	Filter of 200 ns
3	0011	=	Filter of 267 ns
4	0100	=	Filter of 333 ns
5	0101	=	Filter of 400 ns
6	0110	=	Filter of 467 ns
7	0111	=	Filter of 533 ns
8	1000	=	Filter of 600 ns
9	1001	=	Filter of 667 ns
10	1010	=	Filter of 733 ns
11	1011	=	Filter of 800 ns
12	1100	=	Filter of 867 ns
13	1101	=	Filter of 933 ns
14	1110	=	Filter of 1000 ns
15	1111	=	Filter of 1067 ns

DQ8 : 40 MHz status (can be read only)

- 1: 40 MHz available
- 0: 40 MHz not available

DQ9 : Filter clock

- 1: 40 MHz
- 0: 33 MHz (or 30 MHz on former mainboards)

DQ15 : Enable to read the filter status

- 1: next RD access to Register 60 reads the filter status
- 0: next RD access to Register 60 reads the Version Register

2.6.10 Version register

On the base address Base + 60 the recognition and revision of the function is realised in the ASCII format.

Example: BASE + 60 "S" "C" "2" "3"
 Meaning: Incremental counter, revision 2.3

2.7 Limit values

Counter depth: 32-bit

Max. counting frequency: 5 MHz

Min. phase shift: 40° at 5 MHz

Access time onto the counter: 200 ns

3 STANDARDSOFTWARE

3.1 Define values

Table 3-1: Define table

Define Name	Decimal value	Hexadecimal value
APCI1710_DISABLE	0	0
APCI1710_ENABLE	1	1
APCI1710_16BIT_COUNTER	16	10
APCI1710_32BIT_COUNTER	0	0
APCI1710_QUADRUPLE_MODE	0	0
APCI1710_DOUBLE_MODE	3	3
APCI1710_SIMPLE_MODE	15	F
APCI1710_DIRECT_MODE	128	80
APCI1710_HYSTERESIS_ON	96	60
APCI1710_HYSTERESIS_OFF	0	0
APCI1710_INCREMENT	96	60
APCI1710_DECREMENT	0	0
APCI1710_HIGH_EDGE_CLEAR_COUNTER	0	0
APCI1710_HIGH_EDGE_LATCH_COUNTER	1	1
APCI1710_LOW_EDGE_CLEAR_COUNTER	2	2
APCI1710_LOW_EDGE_LATCH_COUNTER	3	3
APCI1710_HIGH_EDGE_LATCH_AND_CLEAR_COUNTER	4	4
APCI1710_LOW_EDGE_LATCH_AND_CLEAR_COUNTER	5	5
APCI1710_LOW	0	0
APCI1710_HIGH	1	1
APCI1710_SOURCE_0	0	0
APCI1710_SOURCE_1	1	1
APCI1710_30MHZ	30	1E
APCI1710_33MHZ	33	21
APCI1710_40MHZ	40	28

3.2 Interruptmask

Each incremental counter can generate an interrupt. In order to get this interrupt, you shall enable the interrupt with the function "i_APCI1710_SetBoardIntRoutineX".

Table 3-2: Interruptmask of the function "incremental counter"

b_ModuleMask	ul_InterruptMask	Meaning
0000 0001	0000 0000 0000 0000 0001	Hardware latch of the 1. register, module 0 (32-bit)
0000 0001	0000 0000 0000 0000 0010	Hardware latch of the 2. register, module 0 (32-bit)
0000 0001	0000 0000 0000 0000 0100	Index interrupt from module 0 (32-bit)
0000 0001	0000 0000 0000 0000 1000	Compare interrupt from module 0 (32-bit)
0000 0001	0001 0000 0000 0000 0000	Interrupt at the end of frequency measurement, module 0 (32-bit)
0000 0010	0000 0000 0000 0000 0001	Hardware latch of the 1. register, module 1 (32-bit)
0000 0010	0000 0000 0000 0000 0010	Hardware latch of the 2. register, module 1 (32-bit)
0000 0010	0000 0000 0000 0000 0100	Index interrupt from module 1 (32-bit)
0000 0010	0000 0000 0000 0000 1000	Compare interrupt from module 1 (32-bit)
0000 0010	0001 0000 0000 0000 0000	Interrupt at the end of frequency measurement, module 1 (32-bit)
0000 0100	0000 0000 0000 0000 0001	Hardware latch of the 1. register, module 2 (32-bit)
0000 0100	0000 0000 0000 0000 0010	Hardware latch of the 2. register, module 2 (32-bit)
0000 0100	0000 0000 0000 0000 0100	Index interrupt from module 2 (32-bit)
0000 0100	0000 0000 0000 0000 1000	Compare interrupt from module 2 (32-bit)
0000 0100	0001 0000 0000 0000 0000	Interrupt at the end of frequency measurement, module 2 (32-bit)
0000 1000	0000 0000 0000 0000 0001	Hardware latch of the 1. register, module 3 (32-bit)
0000 1000	0000 0000 0000 0000 0010	Hardware latch of the 2. register, module 3 (32-bit)
0000 1000	0000 0000 0000 0000 0100	Index interrupt from module 3 (32-bit)
0000 1000	0000 0000 0000 0000 1000	Compare interrupt from module 3 (32-bit)
0000 1000	0001 0000 0000 0000 0000	Interrupt at the end of frequency measurement, module 3 (32-bit)

Table 3-3: Counter value: Return table

b_ModuleMask	ul_InterruptMask	Source	ul_CounterLatchValue
b_ModuleMask = 1	ul_InterruptMask = 1	Hardware strobe on 1. latch register of module 0	Latched value on the first latch register (32-bit)
b_ModuleMask = 1	ul_InterruptMask = 2	Hardware strobe on 2. latch register of module 0	Latched value of the second latch register (32-bit)
b_ModuleMask = 2	ul_InterruptMask = 1	Hardware strobe on 1. latch register of module 1	Latched value of the first latch register (32-bit)
b_ModuleMask = 2	ul_InterruptMask = 2	Hardware strobe on 2. latch register of module 1	Latched value of the second latch register (32-bit)
b_ModuleMask = 4	ul_InterruptMask = 1	Hardware strobe on 1. latch register of module 2	Latched value of the first latch register (32-bit)
b_ModuleMask = 4	ul_InterruptMask = 2	Hardware strobe on 2. latch register of module 2	Latched value of the second latch register (32-bit)
b_ModuleMask = 8	ul_InterruptMask = 1	Hardware strobe on 1. latch register of module 3	Latched value of the first latch register (32-bit)
b_ModuleMask = 8	ul_InterruptMask = 2	Hardware strobe on 2. latch register of module 3	Latched value of the second latch register (32-bit)

3.3 Counter initialisation

1) i_APCI1710_InitCounter(...)

Syntax:

```
<Return Wert> = i_APCI1710_InitCounter
                    (BYTE      b_BoardHandle,
                    BYTE      b_ModulNbr,
                    BYTE      b_CounterRange,
                    BYTE      b_FirstCounterMode,
                    BYTE      b_FirstCounterOption,
                    BYTE      b_SecondCounterMode,
                    BYTE      b_SecondCounterOption)
```

Parameter

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710 ¹
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_CounterRange	Selection of the counter range. See table 3-4.
BYTE	b_FirstCounterMode	Operation mode of the first counter. See table 3-5.
BYTE	b_FirstCounterOption	Option of the first counter. See tables 3-6, 3-7.
BYTE	b_SecondCounterMode	Operation mode of the second counter. See table 3-5.
BYTE	b_SecondCounterOption	Option of the second counter. See tables 3-6, 3-7.

-Output:

There is no output.

Task:

Configures the counter operation mode of the selected module (*b_ModulNbr*). This function shall be called firstly before calling other functions, that access the counter.

i

IMPORTANT!

When you have configured the module for two 16-bit counters, a **mixed operation with a counter in the quadruple/double/single mode and with the other counter in the direct mode is possible.**

¹ Used abbreviation for the APCI-1710 and CPCI-1710

Table 3-4: Counter range

Parameter	Value to be entered	Description
b_ModulNbr	APCI1710_16BIT_COUNTER	The module is configured for 2 x 16-bit-counters. - b_FirstCounterMode and b_FirstCounterOption configure the 16-bit counter. - b_SecondCounterMode and b_SecondCounterOption configure the second 16-bit counter
b_ModulNbr	APCI1710_32BIT_COUNTER	The module is configured for a 32-bit counter. - b_FirstCounterMode and b_FirstCounterOption configure the 32-bit counter. - b_SecondCounterMode and b_SecondCounterOption are not used.

Table 3-5: Counter operation mode

Parameter	Value to be entered	Description
b_FirstCounterMode oder b_SecondCounterMode	APCI1710_QUADRUPLE_MODE	In the quadruple mode, the edge analysis circuit generates a counting pulse from each edge of two signals which are phase-shifted in relation to each other.
b_FirstCounterMode oder b_SecondCounterMode	APCI1710_DOUBLE_MODE	Same function as quadruple mode, except only 2 of the 4 edges are analysed.
b_FirstCounterMode oder b_SecondCounterMode	APCI1710_SIMPLE_MODE	Same function as quadruple mode, except one of the 4 edges is analysed in each period.
b_FirstCounterMode oder b_SecondCounterMode	APCI1710_DIRECT_MODE	In the direct mode both edge analysis circuits become inactive. The inputs A and B in 32-Bit mode or A, B and C, D in 16-Bit mode present, each, one clock pulse gate circuit. Thereby frequency and pulse duration measurements can be done.

**IMPORTANT!**

This option is not available if you have selected the direct mode.

Table 3-6: Counter option for quadruple/double/single mode

Parameter	Value to be entered	Description
<i>b_FirstCounterOption</i> oder <i>b_SecondCounterOption</i>	APCI1710_HYSTERESIS_ON	On both edge analysis circuit a hysteresis switch is available. It suppresses the first counting pulse after a change of rotation.
<i>b_FirstCounterOption</i> oder <i>b_SecondCounterOption</i>	APCI1710_HYSTERESIS_OFF	The first pulse will not be suppressed after a change of rotation.

i

IMPORTANT!

This option is not available when you have selected the quadruple/double/single mode.

Table 3-7: Counter option for direct mode

Parameter	Value to be entered	Description
<i>b_FirstCounterOption</i> oder <i>b_SecondCounterOption</i>	APCI1710_INCREMENT	The counter increments after each counting pulse.
<i>b_FirstCounterOption</i> oder <i>b_SecondCounterOption</i>	APCI1710_DECREMENT	The counter decrements after each counting pulse.

Calling convention:

ANSI C:

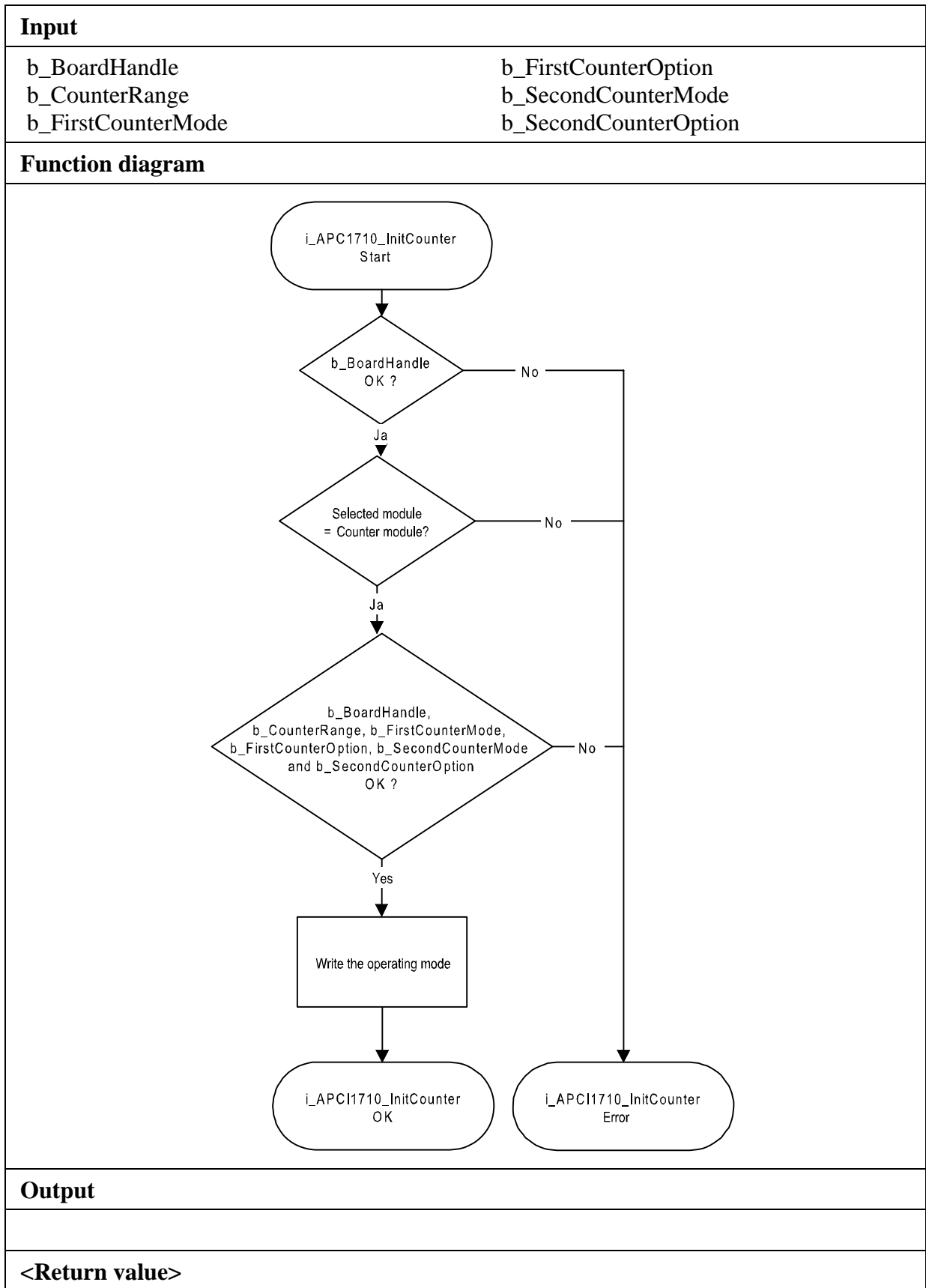
```
int          i_ReturnValue;
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_APCI1710_InitCounter
                (b_BoardHandle,
                 0,
                 APCI1710_16BIT_COUNTER,
                 APCI1710_QUADRUPLE_MODE,
                 APCI1710_HYSTERESIS_ON,
                 APCI1710_QUADRUPLE_MODE,
                 APCI1710_HYSTERESIS_ON);
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: The module is no counter module.
- 3: The selected counter range is wrong. See table 3-4

- 4: The selected operation mode of the first counter is wrong.
See table 3-5
- 5: The selected operation option for the first counter is wrong.
See table 3-6
- 6: The selected operation mode of the second counter is wrong.
See table 3-5
- 7: The selected operation mode option for the second counter is wrong.
See table 3-6



2) i_APCI1710_CounterAutoTest (...)

Syntax:

```
<Return Wert> = i_APCI1710_CounterAutoTest
                    (BYTE   b_BoardHandle,
                    PBYTE  pb_TestStatus)
```

Parameter:

-Input:

BYTE b_BoardHandle Handle of the board **xPCI-1710**

-Output:

PBYTE pb_TestStatus Autotest return. See table 3-8

Task:

A test mode is predetermined for the test of the module. All 8-bit counter chains are operated internally as downwards counters. Independently, from all external signals, all 8-bit counter chains will be decremented parallel at each negative clock edge of CLKX.

Table 3-8: Autotest return

<i>pb_TestStatus</i> Maske	Meaning
0000	No error
0001	Error on counter 0
0010	Error on counter 1
0100	Error on counter 2
1000	Error on counter 3

Calling convention:

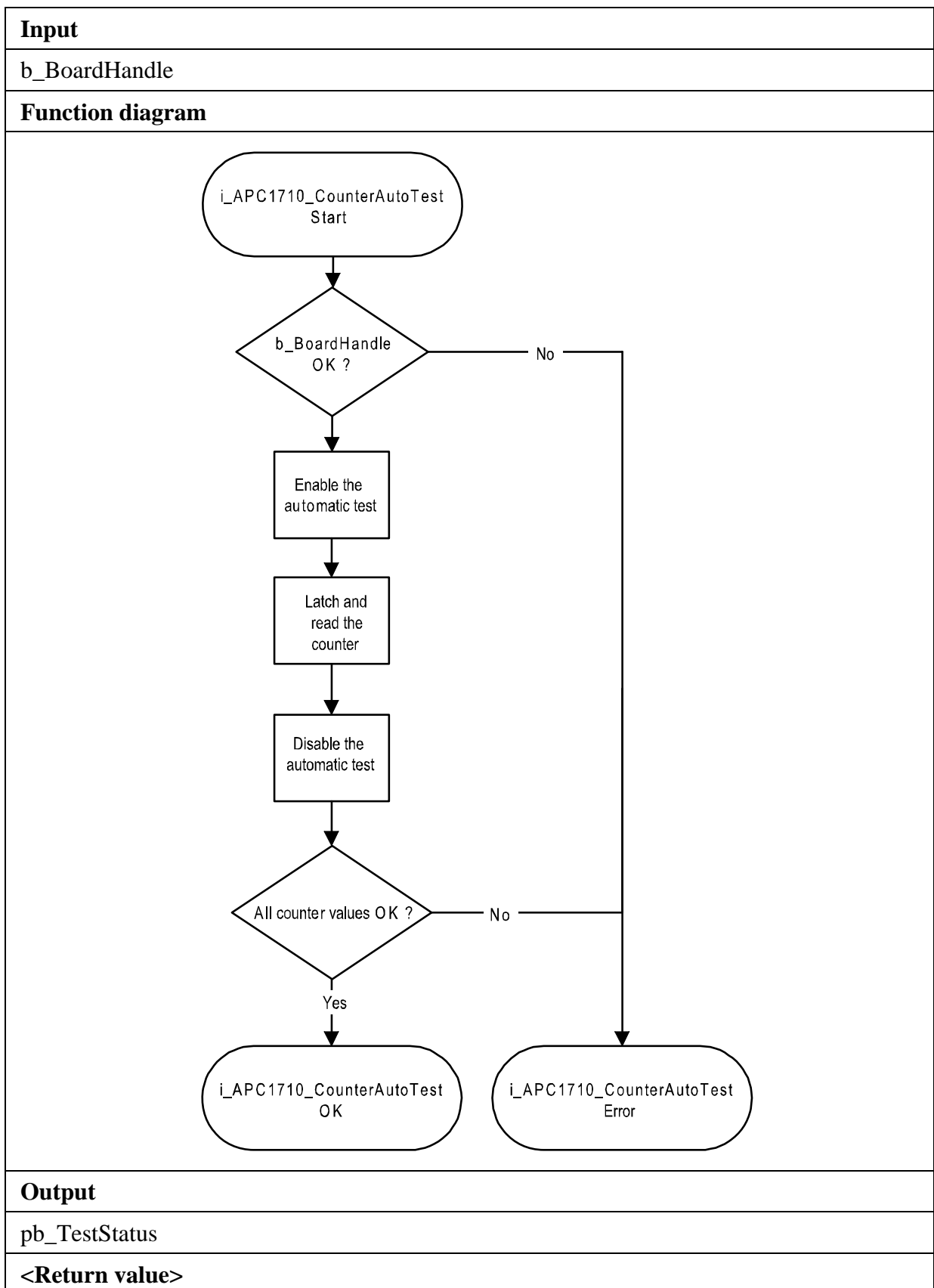
ANSI C :

```
int          i_ReturnValue;
unsigned char b_TestStatus;
```

```
i_ReturnValue = i_APCI1710_CounterAutoTest (b_BoardHandle,
                                             &b_TestStatus;
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: No counter module found



3) i_APCI1710_ClearCounterValue (...)

Syntax:

```
<Return Wert> = i_APCI1710_ClearCounterValue  
                (BYTE   b_BoardHandle,  
                BYTE   b_ModulNbr)
```

Parameter:**-Input:**

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

There is no output.

Task:

Deletes the counter value of the selected module (*b_ModulNbr*).

Calling convention:

ANSI C :

```
int          i_ReturnValue;  
unsigned char b_BoardHandle;  
i_ReturnValue = i_APCI1710_ClearCounterValue  
                (b_BoardHandle,  
                0);
```

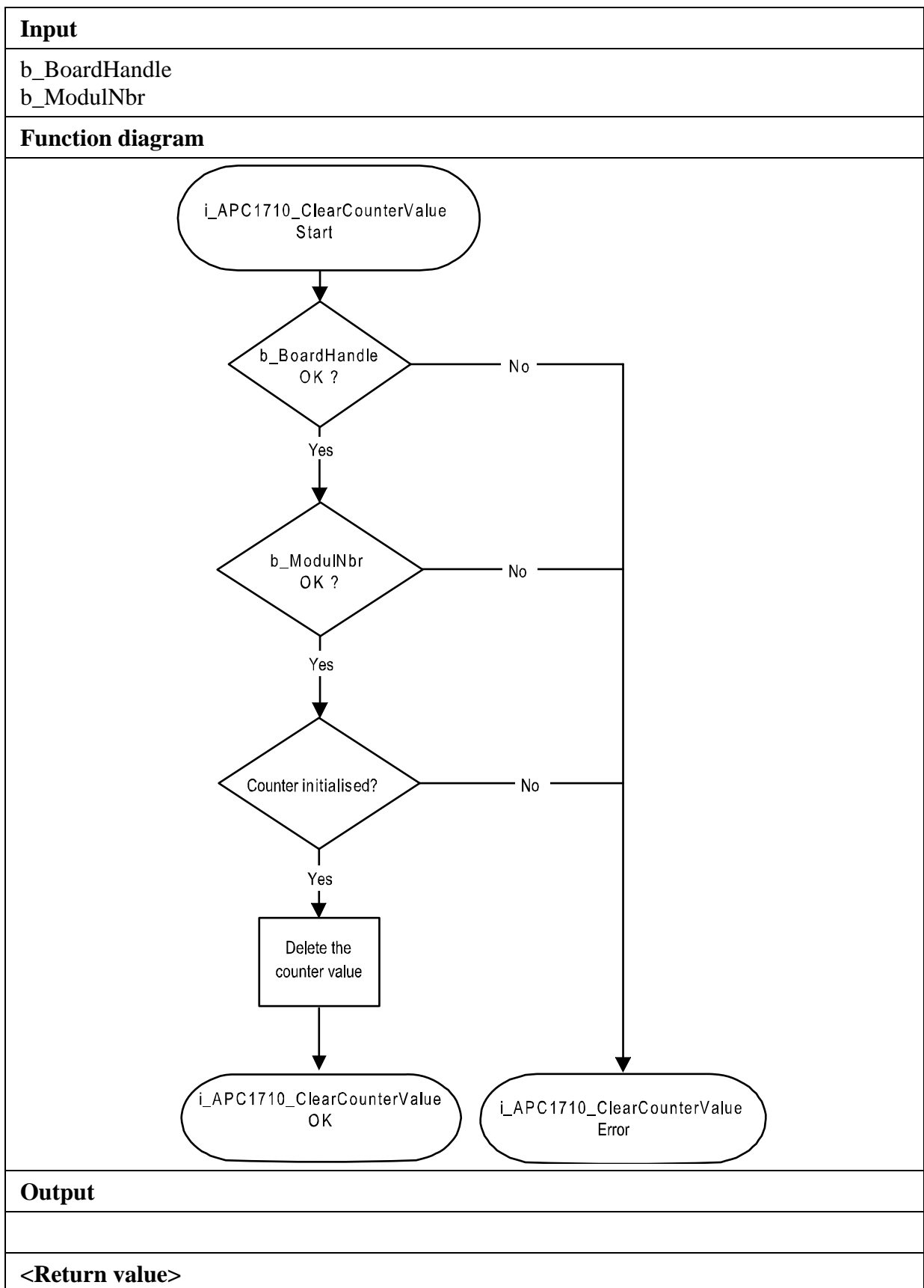
Return value:

0: No error

-1: Handle parameter of the board is wrong.

-2: The selected module number is wrong

-3: Counter not initialised. See function "i_APCI1710_InitCounter"



4) i_APCI1710_ClearAllCounterValue (...)

Syntax:

<Return Wert> = i_APCI1710_ClearAllCounterValue
(BYTE b_BoardHandle)

Parameter:**-Input:**

BYTE b_BoardHandle Handle of the board **xPCI-1710**

-Output:

There is no output.

Task:

Deletes all counter values.

Calling convention:

ANSI C :

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

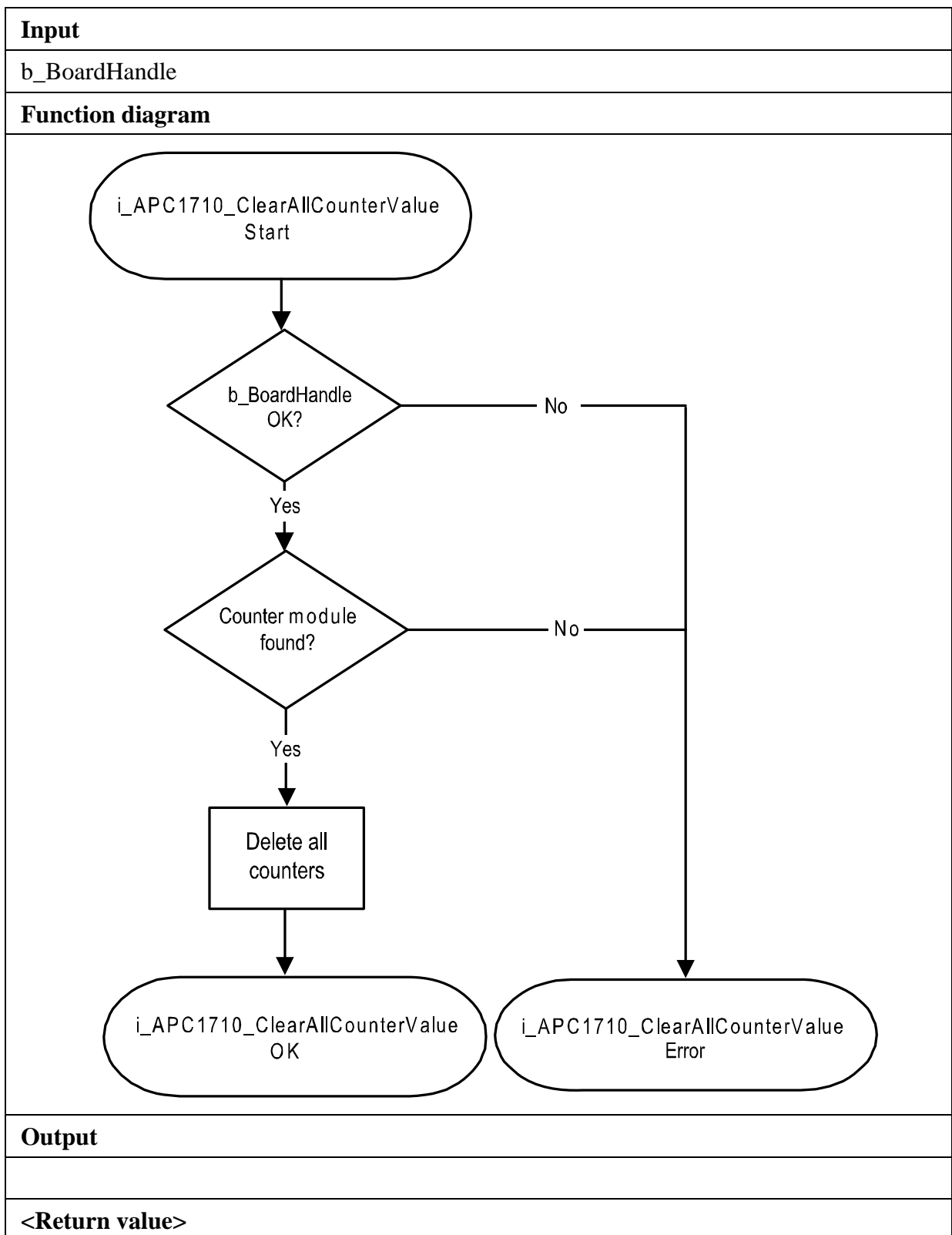
```
i_ReturnValue = i_APCI1710_ClearAllCounterValue   (b_BoardHandle);
```

Return value:

0: No error

-1: Handle parameter of the board is wrong.

-2: No counter module found.



5) i_APCI1710_SetInputFilter (...)

Syntax:

```
<Return Wert> = i_APCI1710_SetInputFilter
                    (BYTE  b_BoardHandle,
                     BYTE  b_ModulNbr,
                     BYTE  b_PCInputClock,
                     BYTE  b_Filter)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board x PCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
	BYTE b_PCInputClock	Selection of the PCI bus clock
		- APCI1710_30MHZ: The PC has a PCI bus clock of 30 MHz
		- APCI1710_33MHZ: The PC has a PCI bus clock of 33 MHz
		- APCI1710_40MHZ: The PC has a PCI bus clock of 40 MHz
BYTE	b_Filter	Selection of the filter for the diff. inputs A,B,C and D. See table 3-9

-Output:

There is no output.

Task:

Disables or enables the software filter in the selected module (b_ModulNbr) for the inputs A,B,C and D.

b_Filter indicates the filter time.

Table 3-9: Filter time

<i>b_PCIInputClock</i>	APCI1710_30MHZ	APCI1710_33MHZ	APCI1710_40MHZ
<i>b_Filter</i>			
0	Filter not used	Filter not used	Filter not used
1	133 ns	121 ns	100 ns
2	200 ns	182 ns	150 ns
3	267 ns	242 ns	200 ns
4	333 ns	303 ns	250 ns
5	400 ns	364 ns	300 ns
6	467 ns	424 ns	350 ns
7	533 ns	485 ns	400 ns
8	600 ns	545 ns	450 ns
9	667 ns	606 ns	500 ns
10	733 ns	667 ns	550 ns
11	800 ns	727 ns	600 ns
12	867 ns	788 ns	650 ns
13	933 ns	848 ns	700 ns
14	1000 ns	909 ns	750 ns
15	1067 ns	970 ns	800 ns

Calling convention:

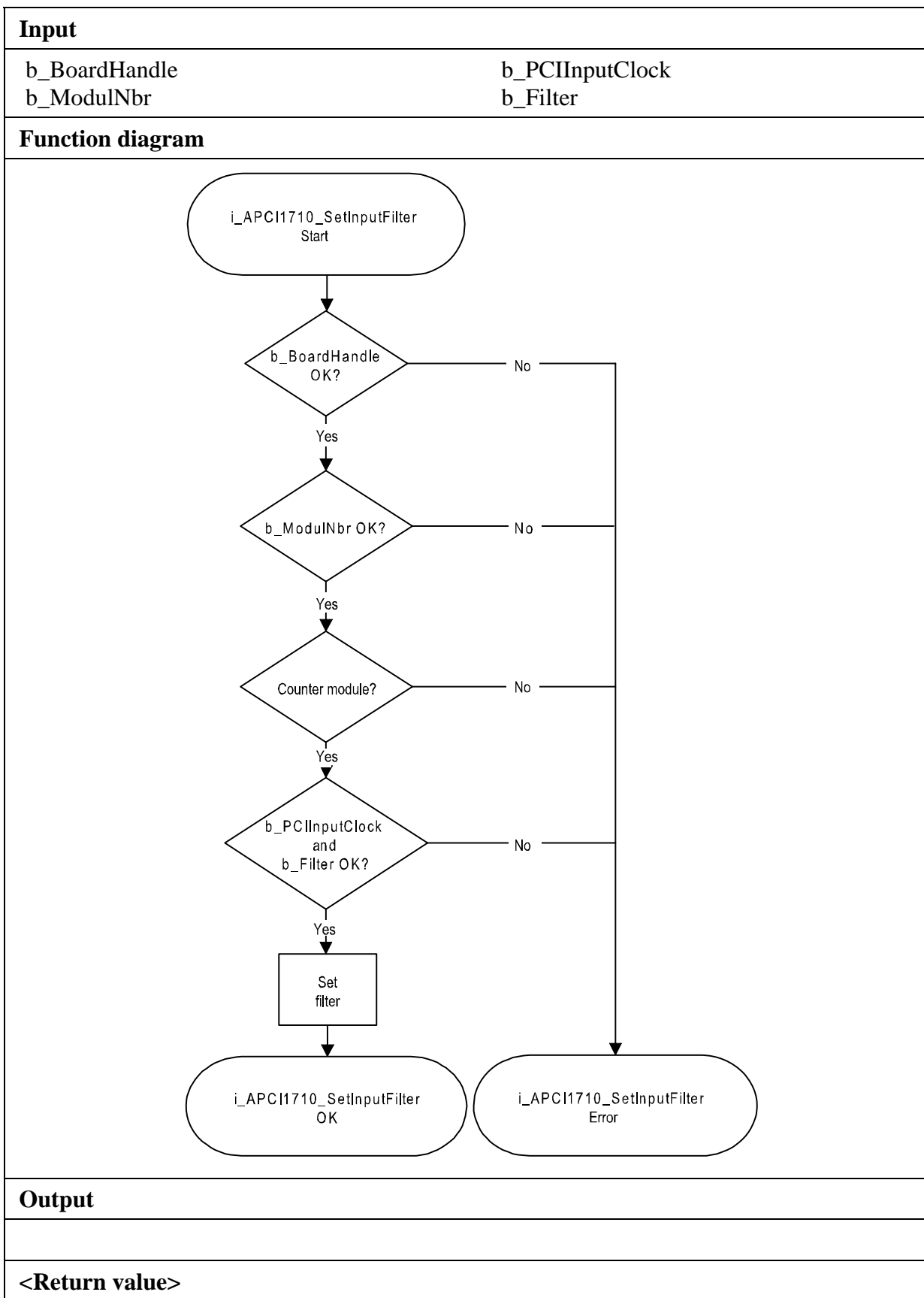
ANSI C :

```

int                i_ReturnValue;
unsigned char      b_BoardHandle;
i_ReturnValue = i_APCI1710_SetInputFilter
                (b_BoardHandle
                0,
                APCI1710_40MHz,
                9);
    
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: The selected module number is wrong.
- 3: The selected module is no counter module.
- 4: The selected PCI input clock is wrong.
- 5: The selected filter time is wrong.
- 6: On the board no 40 MHz quartz is implemented.



3.4 Reading the counter

1) i_APCI1710_LatchCounter (...)

Syntax:

```
<Return Wert> = i_APCI1710_LatchCounter
                    (BYTE b_BoardHandle,
                    BYTE b_ModulNbr,
                    BYTE b_LatchReg)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_LatchReg	Selection of the latch register 0: for the first latch register 1: for the second latch register

-Output:

There is no output.

Task:

Latching of the current value of the selected module (*b_ModulNbr*) into the selected latch register (*b_LatchReg*).

Calling convention:

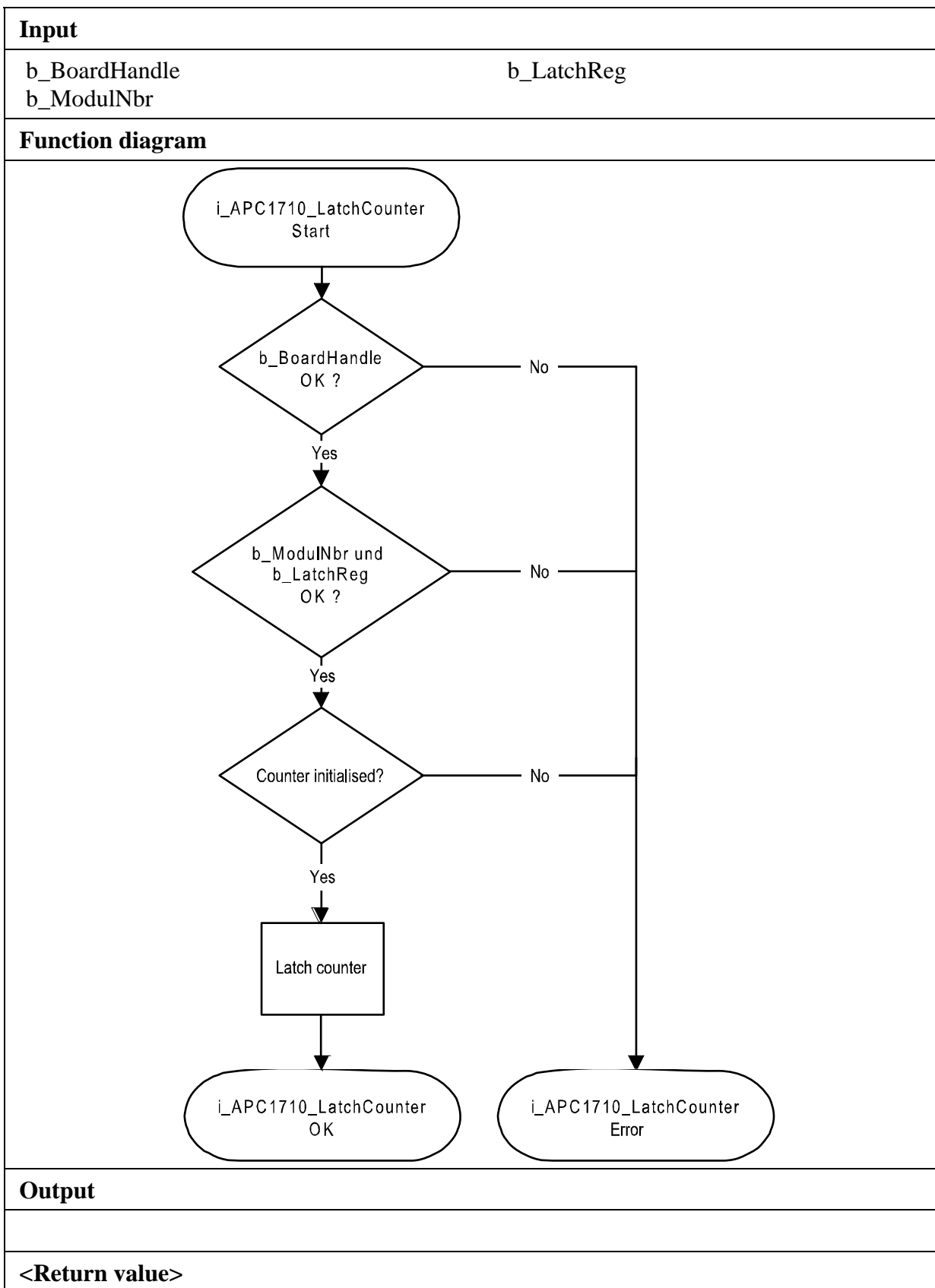
ANSI C :

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

```
i_ReturnValue = i_APCI1710_LatchCounter
                    (b_BoardHandle,
                    0,
                    0);
```

Return Wert:

0: No error
-1: Handle parameter of the board is wrong.
-2: The selected module number is wrong.
-3: Counter not initialised. See function "i_APCI1710_InitCounter"
-4: The selected latch register is wrong.



2) i_APCI1710_ReadLatchRegisterStatus (...)

Syntax:

```
<Return Wert> = i_APCI1710_ReadLatchRegisterStatus
                    (BYTE b_BoardHandle,
                     BYTE b_ModulNbr,
                     BYTE b_LatchReg,
                     PBYTE pb_LatchStatus)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_LatchReg	Selection of the latch register 0: for the first latch register 1: for the second latch register

-Output:

PBYTE	pb_LatchStatus	Latch register status. 0: No latch 1: A software latch was generated. 2: A hardware latch was generated 3: A software latch and a hardware latch were generated.
-------	----------------	--

Task:

Reads the status of the selected latch register (*b_LatchReg*) in the selected module (*b_ModulNbr*).

Calling convention:

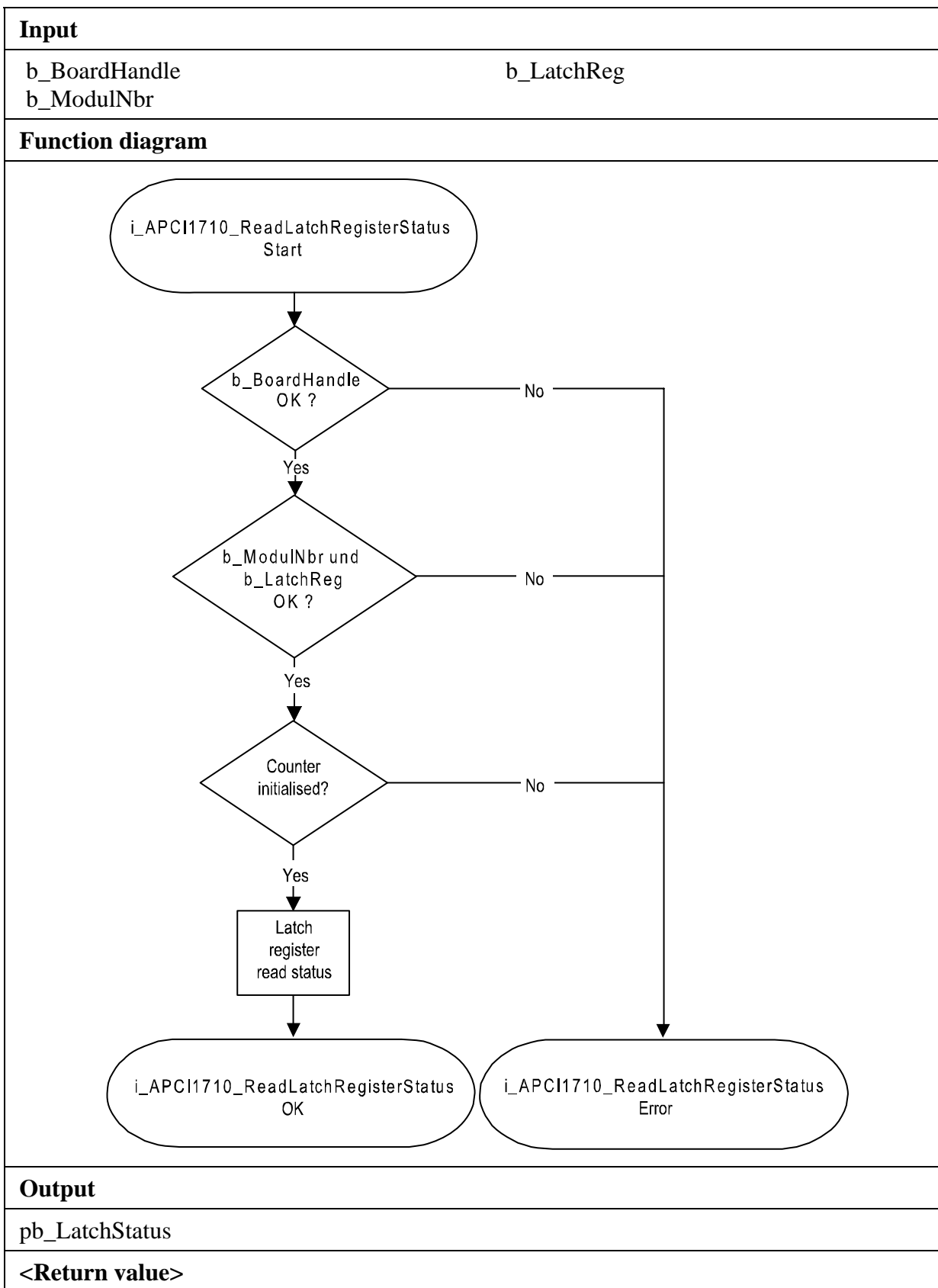
ANSI C:

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

```
i_ReturnValue = i_APCI1710_LatchRegisterStatus
                (b_BoardHandle, 0, 0,
                 b_LatchStatus);
```

Return value:

0: No error
-1: Handle parameter of the board is wrong.
-2: The selected module number is wrong.
-3: Counter not initialised. See function "i_APCI1710_InitCounter"
-4: The selected latch register is wrong.



3) i_APCI1710_ReadLatchRegisterValue (...)

Syntax:

```
<Return Wert> = i_APCI1710_ReadLatchRegisterValue
                (BYTE      b_BoardHandle,
                 BYTE      b_ModulNbr,
                 BYTE      b_LatchReg,
                 PULONG    pul_LatchValue)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_LatchReg	Selection of the latch register 0: for the first latch register 1: for the second latch register

-Output:

PULONG	pul_LatchValue	Latch register value
--------	----------------	----------------------

Task:

Reads the value in the selected latch register (*b_LatchReg*) in the selected module (*b_ModulNbr*)

Calling convention:

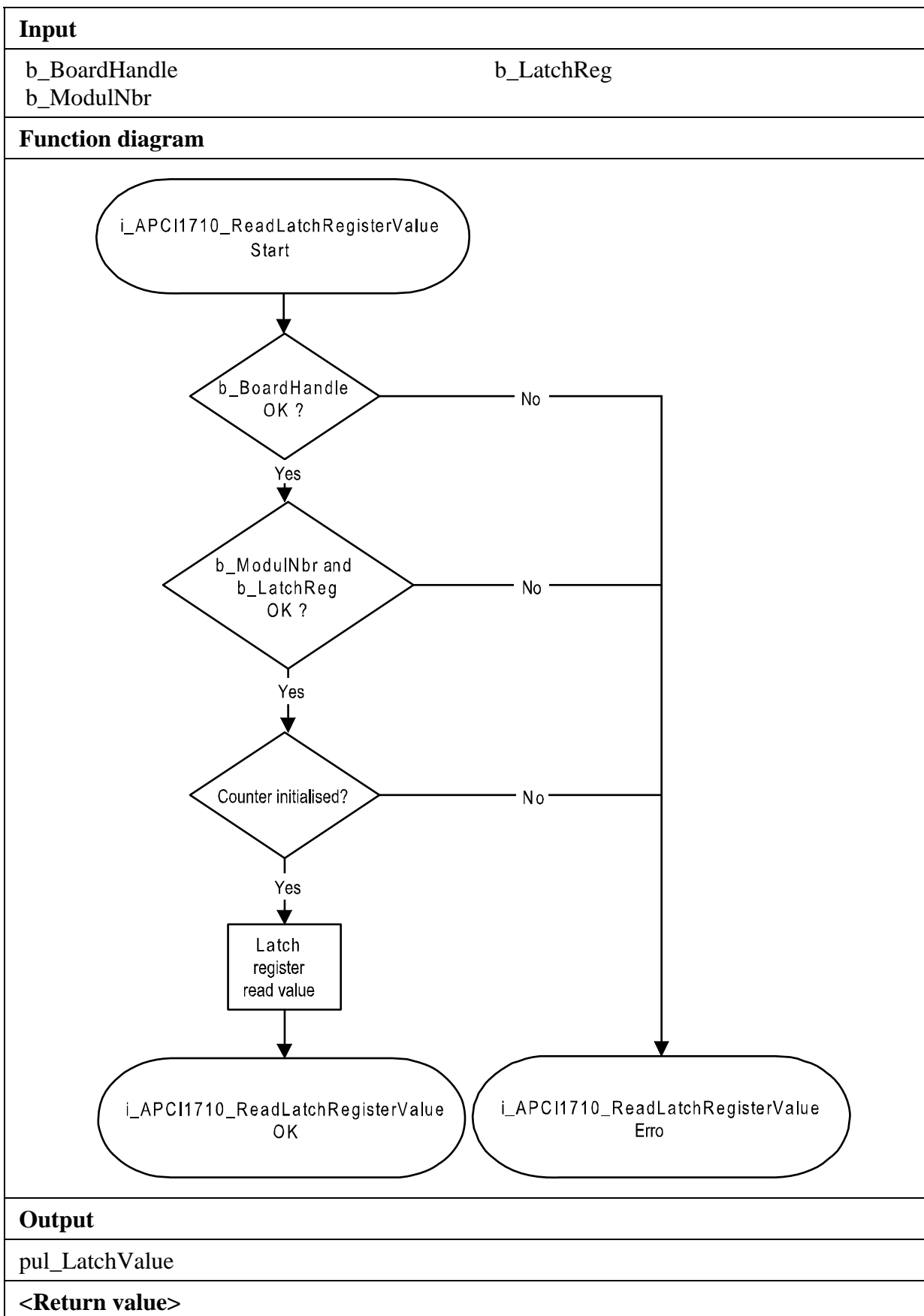
ANSI C :

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

```
i_ReturnValue = i_APCI1710_ReadLatchRegisterValue
                (b_BoardHandle,
                 0,
                 0,
                 &ul_LatchValue);
```

Return value:

0: No error
-1: Handle parameter of the board is wrong.
-2: The selected module number is wrong.
-3: Counter not initialised. See function "i_APCI1710_InitCounter"
-4: The selected latch register is wrong.



4) i_APCI1710_EnableLatchInterrupt (...)

Syntax:

```
<Return Wert> = i_APCI1710_EnableLatchInterrupt
                                     (BYTE   b_BoardHandle,
                                     BYTE   b_ModulNbr)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

There is no output.

Task:

Enables the latch interrupt of the selected module (*b_ModulNbr*). Each hardware latch generates an interrupt.

Calling convention:

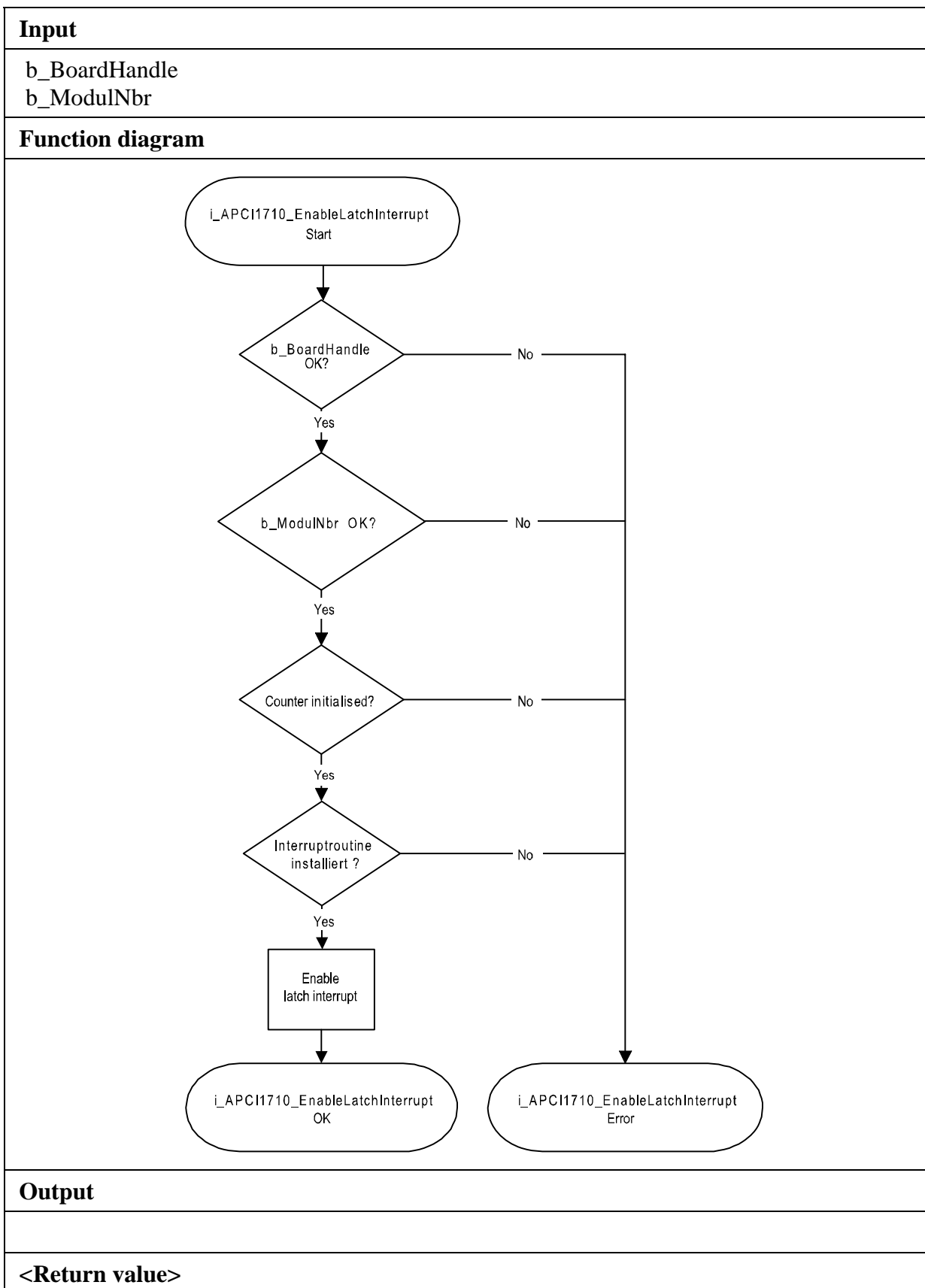
ANSI C:

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

```
i_ReturnValue = i_APCI1710_EnableLatchInterrupt
                (b_BoardHandle, 0);
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: The selected module number is wrong.
- 3: Counter not initialised. See function "i_APCI1710_InitCounter"
- 4: Interrupt routine not installed. See function "i_APCI1710_SetBoardIntRoutine"



5) i_APCI1710_DisableLatchInterrupt (...)

Syntax:

```
<Return Wert> = i_APCI1710_DisableLatchInterrupt
                    (BYTE b_BoardHandle,
                     BYTE b_ModulNbr)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

There is no output.

Task:

Disables the latch interrupt of the selected module (*b_ModulNbr*).

Calling convention:

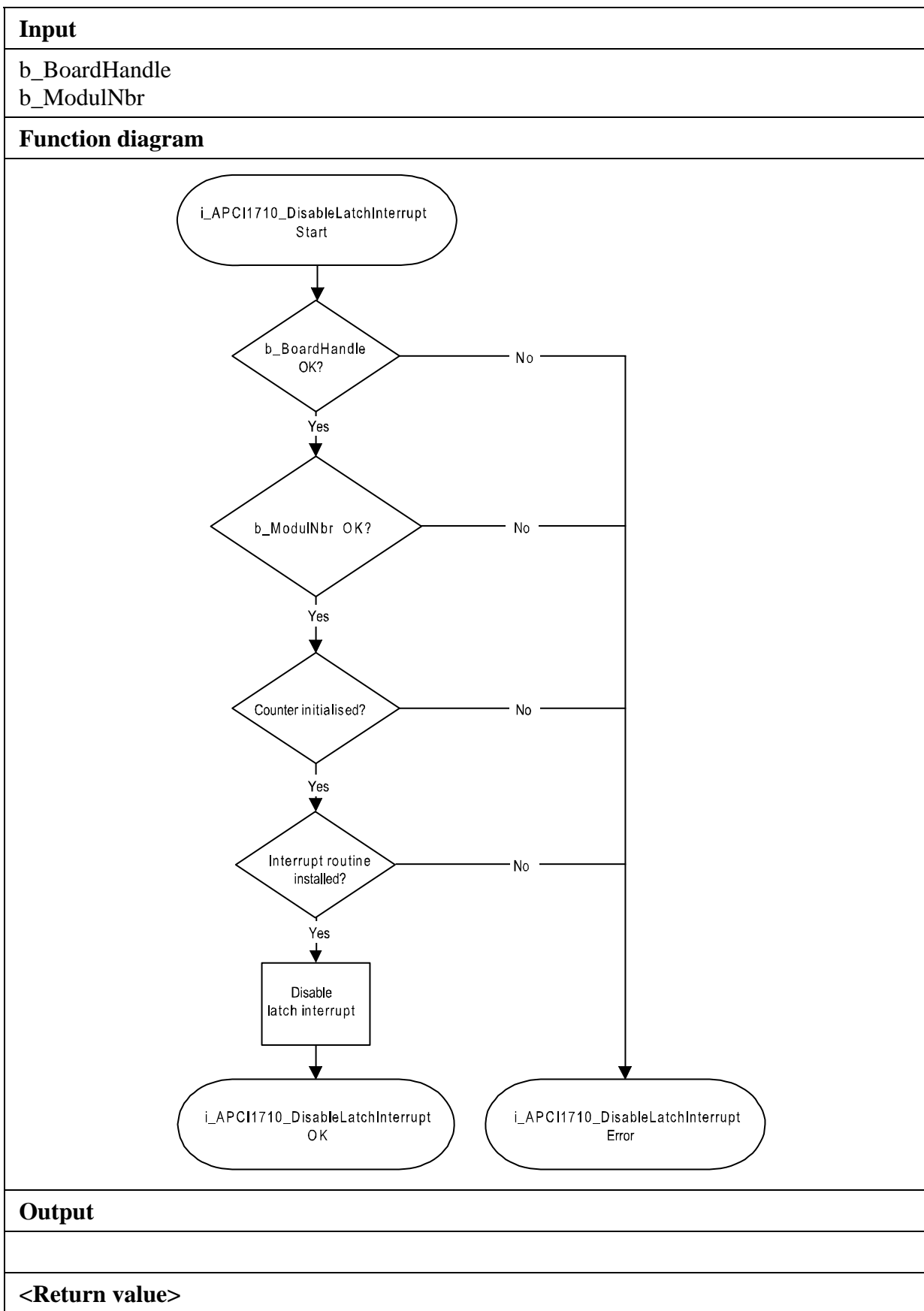
ANSI C:

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

```
i_ReturnValue = i_APCI1710_DisableLatchInterrupt
                (b_BoardHandle, 0);
```

Return value:

0: No error
-1: Handle parameter of the board is wrong.
-2: The selected module number is wrong.
-3: Counter not initialised. See function "i_APCI1710_InitCounter"
-4: Interrupt routine not installed. See function "i_APCI1710_SetBoardIntRoutine"



6) i_APCI1710_Read16BitCounterValue (...)

Syntax:

```
<Return Wert> = i_APCI1710_Read16BitCounterValue
                    (BYTE  b_BoardHandle,
                     BYTE  b_ModulNbr,
                     BYTE  b_SelectedCounter,
                     PUINT  pui_CounterValue)
```

Parameter:**-Input:**

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_SelectedCounter	Selection of the 16-bit counter (0 or 1)

-Output:

PUINT	pui_CounterValue	16-bit counter value
-------	------------------	----------------------

Task:

Latching of the 16-bit counter (*b_SelectedCounter*) from the selected module (*b_ModulNbr*) in the first latch register and return of the latched value.

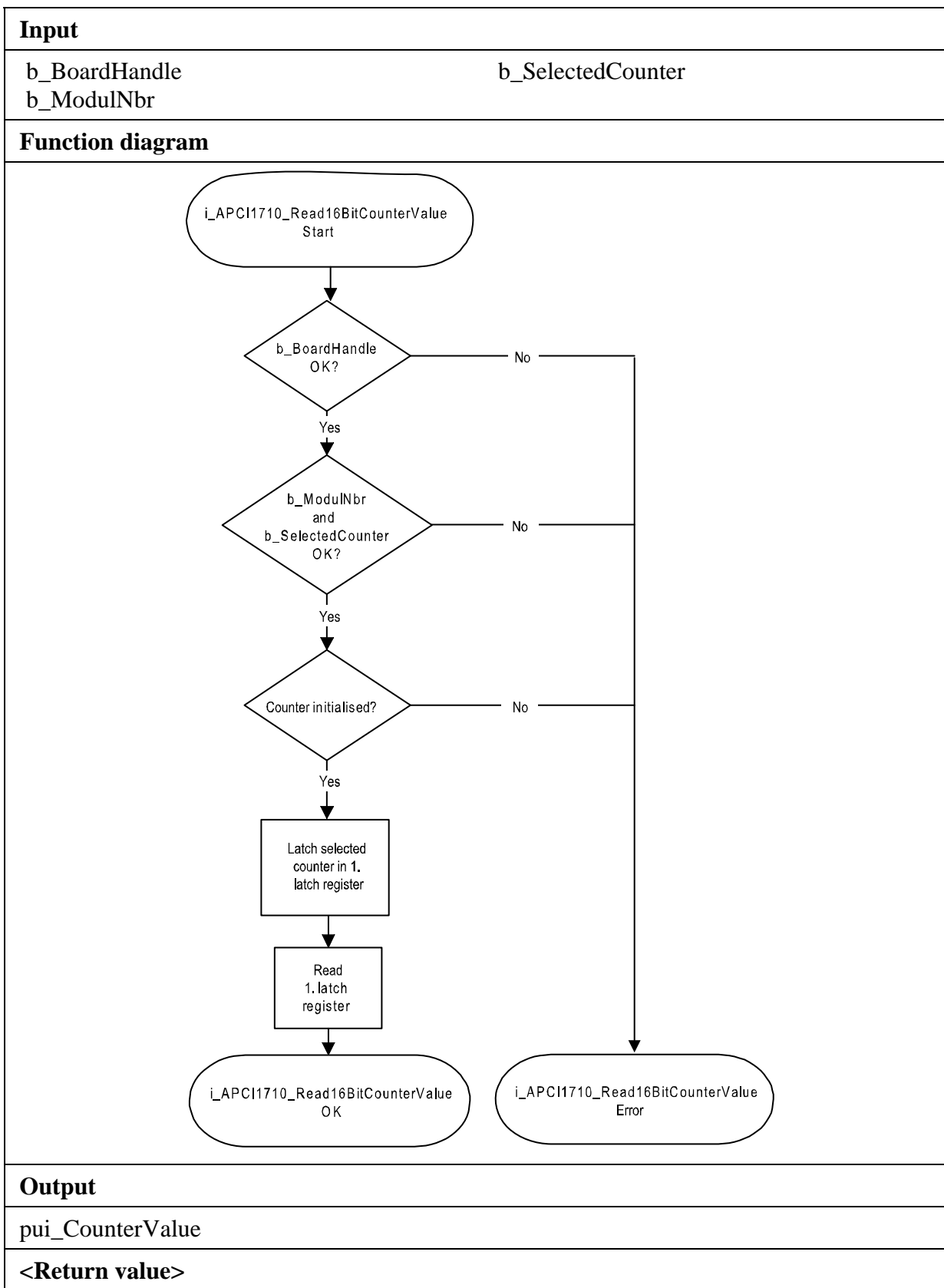
Calling convention:ANSI C:

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

```
i_ReturnValue = i_APCI1710_Read16BitCounterValue
                    (b_BoardHandle,
                     0,
                     0,
                     &ui_CounterValue);
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong
- 2: The selected module number is wrong.
- 3: Counter is not initialised. See function "i_APCI1710_InitCounter".
- 4: The selected 16-bit counter is wrong.



7) i_APCI1710_Read32BitCounterValue (...)

Syntax:

```
<Return Wert> = i_APCI1710_Read32BitCounterValue
                                     (BYTE      b_BoardHandle,
                                     BYTE      b_ModulNbr
                                     PULONG    pul_CounterValue)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

PULONG	pul_CounterValue	32-bit counter value
--------	------------------	----------------------

Task:

Latching of the 32-bit counter of the selected module (*b_ModulNbr*) in the first latch register and return of the latched value.

Calling convention:

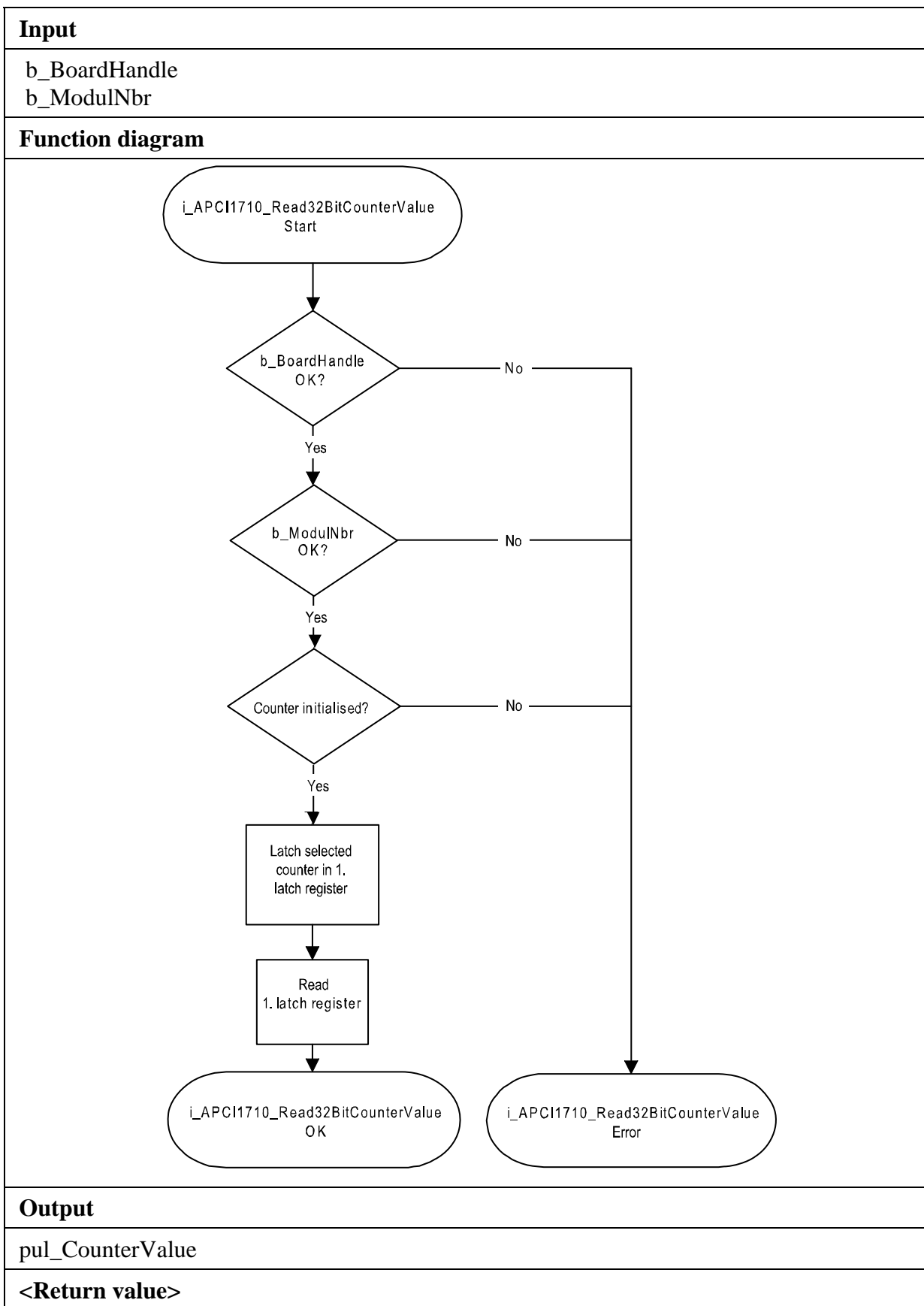
ANSI C:

int	i_ReturnValue;
unsigned char	b_BoardHandle;
unsigned long	ul_CounterValue;

```
i_ReturnValue = i_APCI1710_Read32BitCounterValue
                (b_BoardHandle,
                0,
                &ul_CounterValue);
```

Return value:

0: No error
-1: Handle parameter of the board is wrong
-2: The selected module number is wrong.
-3: Counter not initialised. See function "i_APCI1710_InitCounter"



3.5 Writing into the counter

1) i_APCI1710_Write16BitCounterValue (...)

Syntax:

```
<Return Wert> = i_APCI1710_Write16BitCounterValue
    (BYTE      b_BoardHandle
     BYTE      b_ModulNbr,
     BYTE      b_SelectedCounter,
     UINT      ui_WriteValue)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_SelectedCounter	Selection of the 16-bit counter (0 or 1)
UINT	ui_WriteValue	16-bit write value

-Output:

There is no output.

Task:

Writes the 16-bit value (*ui_WriteValue*) into the 16-bit counter (*b_SelectedCounter*) of the selected module (*b_ModulNbr*).

Calling convention:

ANSI C:

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

```
i_ReturnValue = i_APCI1710_Write16BitCounterValue (b_BoardHandle,
                                                    0,
                                                    0,
                                                    2000);
```

Return value:

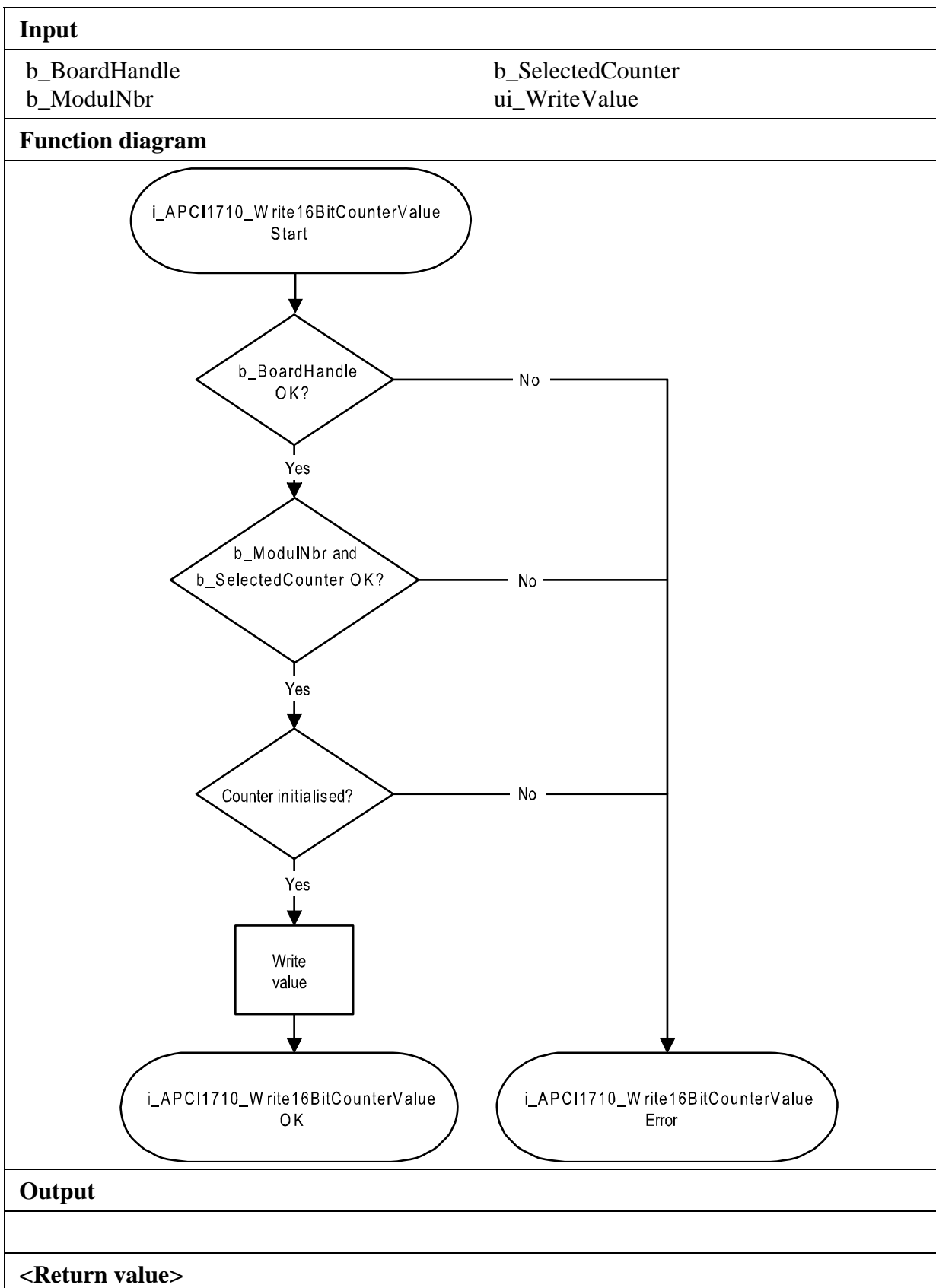
0: No error

-1: Handle parameter of the board is wrong.

-2: The selected module number is wrong

-3: Counter not initialised. See function "i_APCI1710_InitCounter"

-4: The 16-bit counter parameter is wrong.



2) i_APCI1710_Write32BitCounterValue (...)

Syntax:

```
<Return Wert> = i_APCI1710_Write32BitCounterValue
                    (BYTE   b_BoardHandle
                    BYTE   b_ModulNbr,
                    ULONG  ul_WriteValue)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
ULONG	ul_WriteValue	32-bit write value

-Output:

There is no output.

Task:

Writes the 32-bit value (*ul_WriteValue*) into the counter of the selected module (*b_ModulNbr*).

Calling convention:

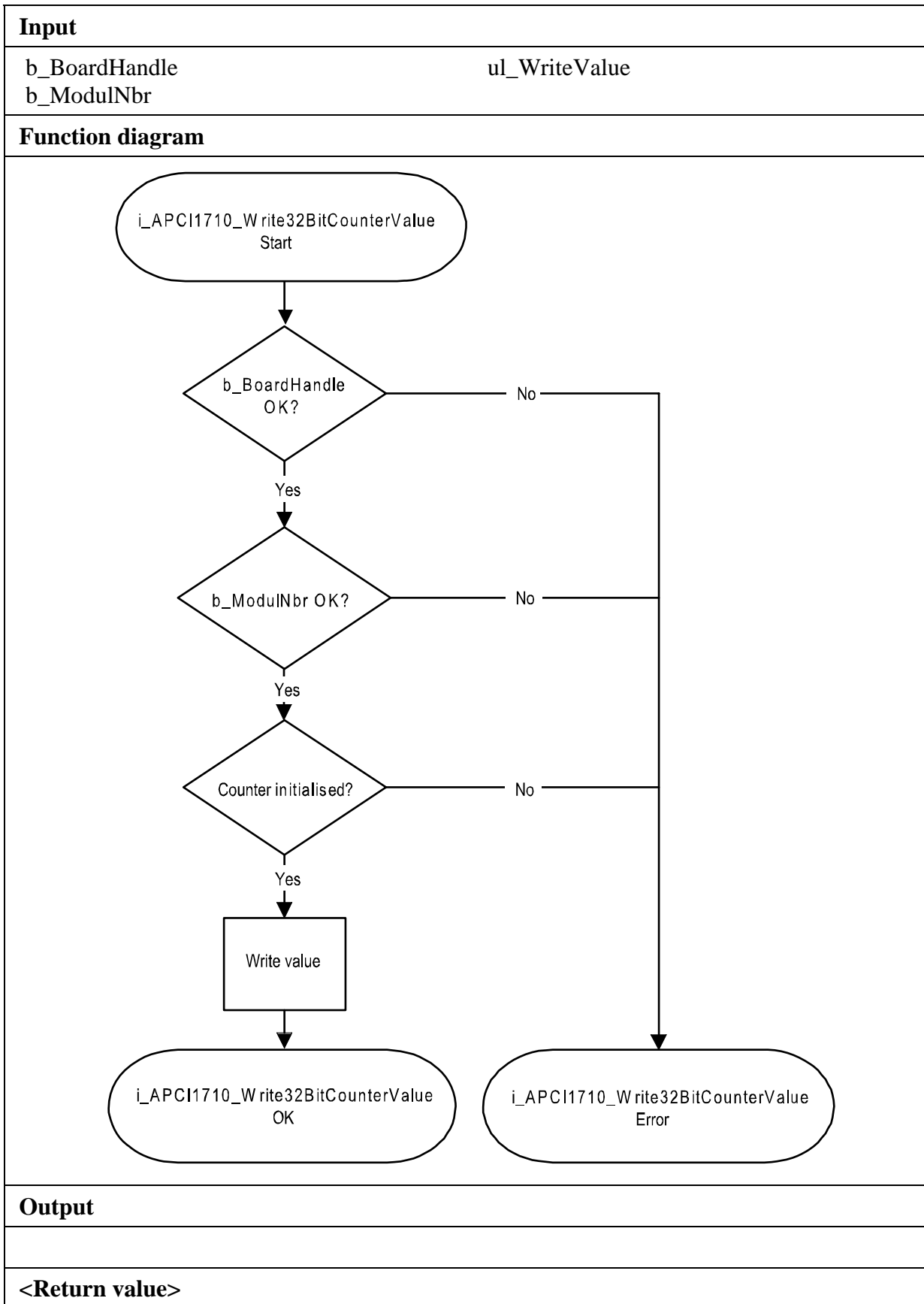
ANSI C:

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

```
i_ReturnValue = i_APCI1710_Write32BitCounterValue (b_BoardHandle,
                                                    0,
                                                    200000);
```

Return value:

0: No error
-1: Handle parameter of the board is wrong.
-2: The selected module number is wrong.
-3: Counter not initialised. See function "i_APCI1710_InitCounter"



3.6 Index

1) i_APCI1710_InitIndex (...)

Syntax:

<Return Wert> = i_APCI1710_InitIndex
 (BYTE b_BoardHandle,
 BYTE b_ModulNbr,
 BYTE b_ReferenceAction,
 BYTE b_IndexOperation,
 BYTE b_AutoMode,
 BYTE b_InterruptEnable)

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_ReferenceAction	Determines if the reference for the index-transfer shall be reset or not - APCI1710_ENABLE: Reference shall be set for the index transfer - APCI1710_DISABLE: Reference is of no importance
BYTE	b_IndexOperation	Index operation mode. See table 3-10
BYTE	b_AutoMode	Enables or disables the automatic index reset - APCI1710_ENABLE: Enables the automatic mod - APCI1710_DISABLE: Disables the automatic mode
BYTE	b_InterruptEnable	Enables or disables the interrupt APCI1710_ENABLE: Interrupt enabled APCI1710_DISABLE: Interrupt disabled

-Output:

There is no output

Task:

Initialises the index, that corresponds with the selected module (*b_ModulNbr*). If there is an INDEX marker (Flag), you can delete the 32-bit counter or latch the current 32-bit value into the first latch register. The *b_IndexOperation* parameter allows selecting the INDEX action.

If you have enabled the automatic mode, each INDEX action is deleted automatically. Otherwise you shall read the index status ("i_APCI1710_ReadIndexStatus") after each INDEX action.

Table 3-10: Index action

b_IndexOperation	Description
APCI1710_HIGH_EDGE_LATCH_COUNTER	After an index signal (High level), the counter value (32-bit) is latched into the first latch register
APCI1710_LOW_EDGE_LATCH_COUNTER	After an index signal (Low level), the counter value (32-bit) is latched into the first latch register
APCI1710_HIGH_EDGE_CLEAR_COUNTER	Nach einem Index-Signal (High Pegel), wird der Zählerwert gelöscht (32-Bit). After an index signal (High level), the counter value is deleted (32-bit).
APCI1710_LOW_EDGE_CLEAR_COUNTER	Nach einem Index-Signal (Low Pegel), wird der Zählerwert gelöscht (32-Bit). After an index signal (Low level), the counter value is deleted (32-bit).
APCI1710_HIGH_EDGE_LATCH_AND_CLEAR_COUNTER	After an index signal (High level), the counter value (32-bit) is latched into the first latch register and then deleted (32-bit).
APCI1710_LOW_EDGE_LATCH_AND_CLEAR_COUNTER	After an index signal (Low level), the counter value (32-bit) is latched into the first latch register and then deleted (32-bit).

Calling convention:ANSI C:

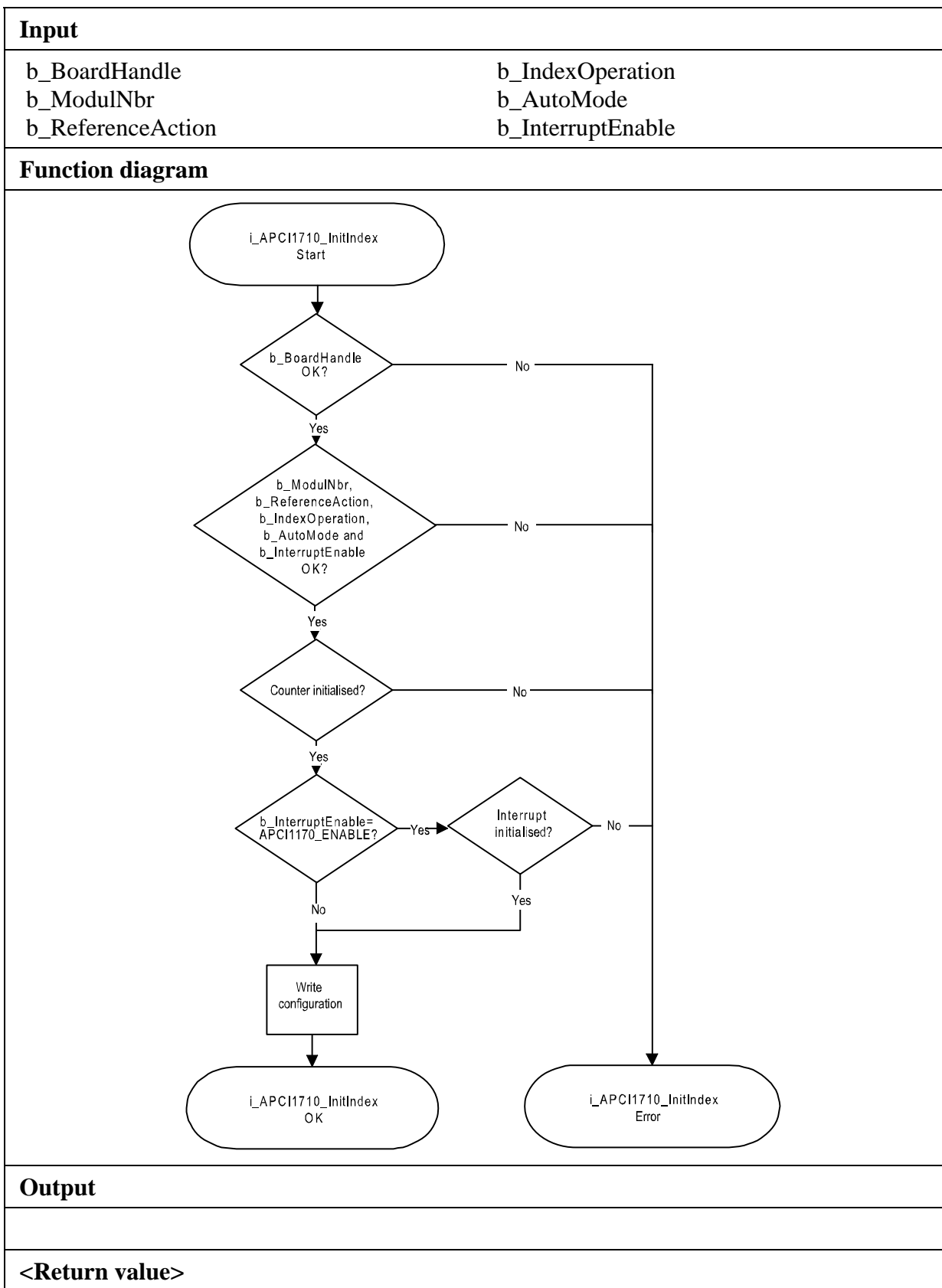
```
int          i_ReturnValue;
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_APCI1710_InitIndex
                (b_BoardHandle,
                 0,
                 APCI1710_DISABLE,
                 APCI1710_HIGH_EDGE_LATCH_COUNTER,
                 APCI1710_ENABLE,
                 APCI1710_DISABLE);
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: The selected modul number is wrong.
- 3: Counter not nitialised. See function "i_APCI1710_InitCounter"
- 4: The selected reference action is wrong.
- 5: The index operation mode is wrong.

- 6: The automatic mode parameter is wrong.
- 7: The interrupt parameter is wrong.
- 8: Interrupt not initialised.
See function "i_APCI1710_SetBoardIntRoutineXX"



2) i_APCI1710_EnableIndex (...)**Syntax:**

```
<Return Wert> = i_APCI1710_EnableIndex
                    (BYTE b_BoardHandle,
                     BYTE   b_ModulNbr)
```

Parameter:**-Input:**

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

There is no output.

Task:

Enables the INDEX actions.

Calling convention:

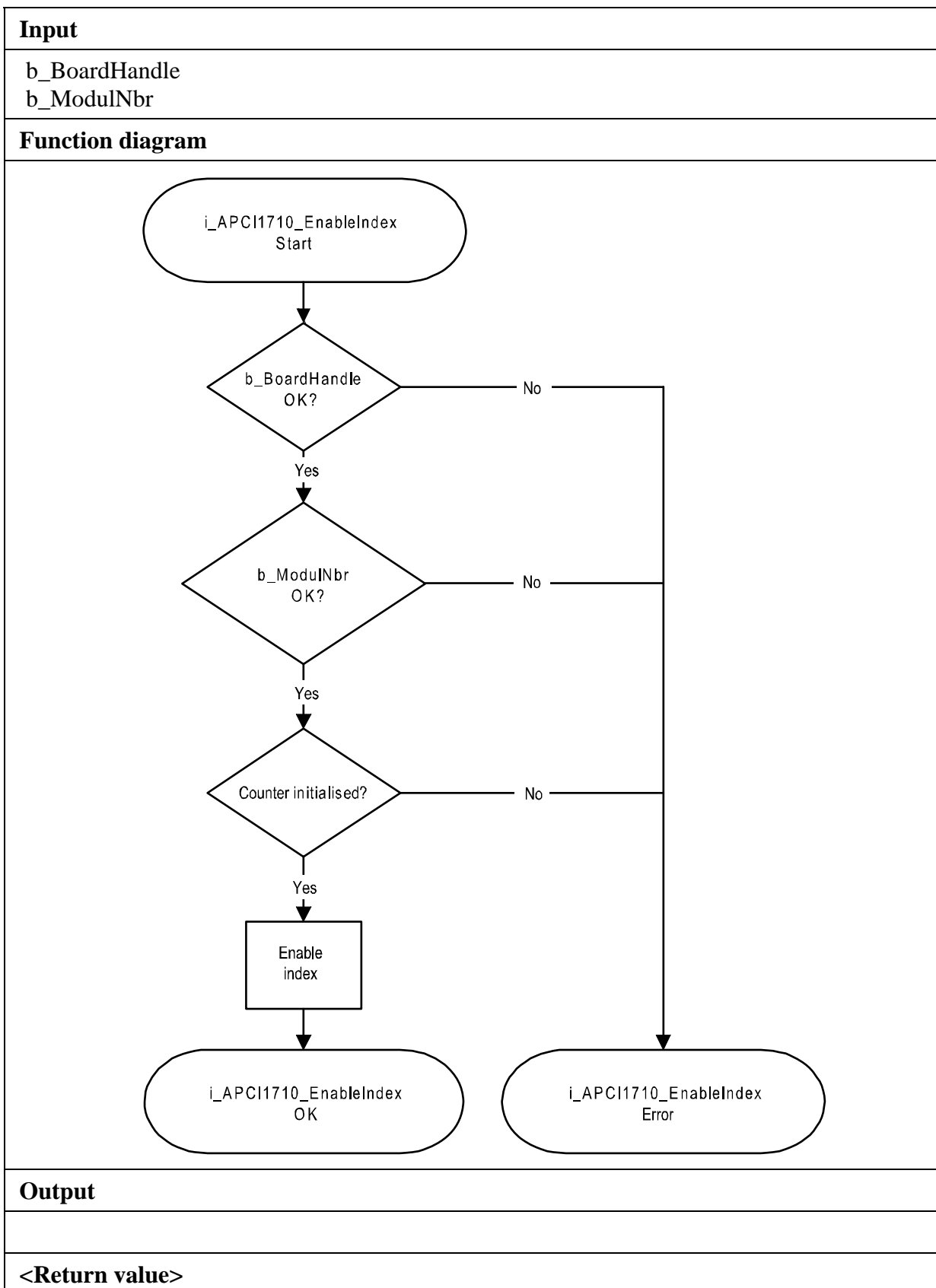
ANSI C:

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

```
i_ReturnValue = i_APCI1710_EnableIndex (b_BoardHandle, 0);
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: Selected module number is wrong.
- 3: Counter not initialised. See function "i_APCI1710_InitCounter"
- 4: Index not initialised. See function "i_APCI1710_InitIndex"



3) i_APCI1710_DisableIndex (...)**Syntax:**

```
<Return Wert> = i_APCI1710_DisableIndex
                    (BYTE    b_BoardHandle,
                     BYTE    b_ModulNbr)
```

Parameter:**-Input:**

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

There is no output.

Task:

Disables the INDEX actions.

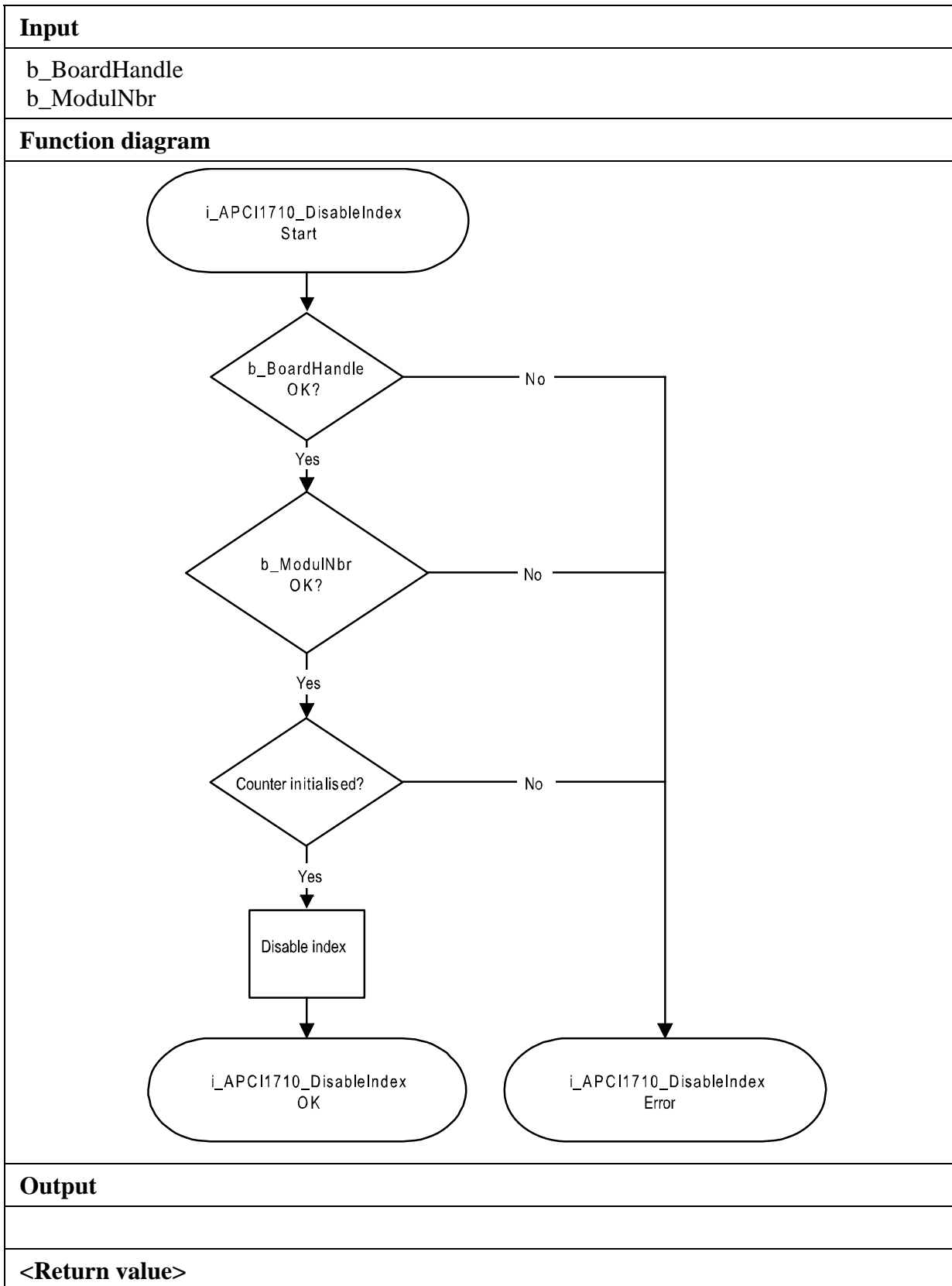
Calling convention:ANSI C:

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

```
i_ReturnValue = i_APCI1710_DisableIndex    (b_BoardHandle, 0);
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: Selected module number is wrong.
- 3: Counter not initialised. See function "i_APCI1710_InitCounter"
- 4: Index not initialised. See function "i_APCI1710_InitIndex"



4) i_APCI1710_GetIndexStatus (...)

Syntax:

```
<Return Wert> = i_APCI1710_GetIndexStatus
                    (BYTE    b_BoardHandle,
                     BYTE    b_ModulNbr,
                     PBYTE   pb_IndexStatus)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

PBYTE	pb_IndexStatus	0: No index 1: An index is determined
-------	----------------	--

Task:

Returns the index status.

Calling convention:

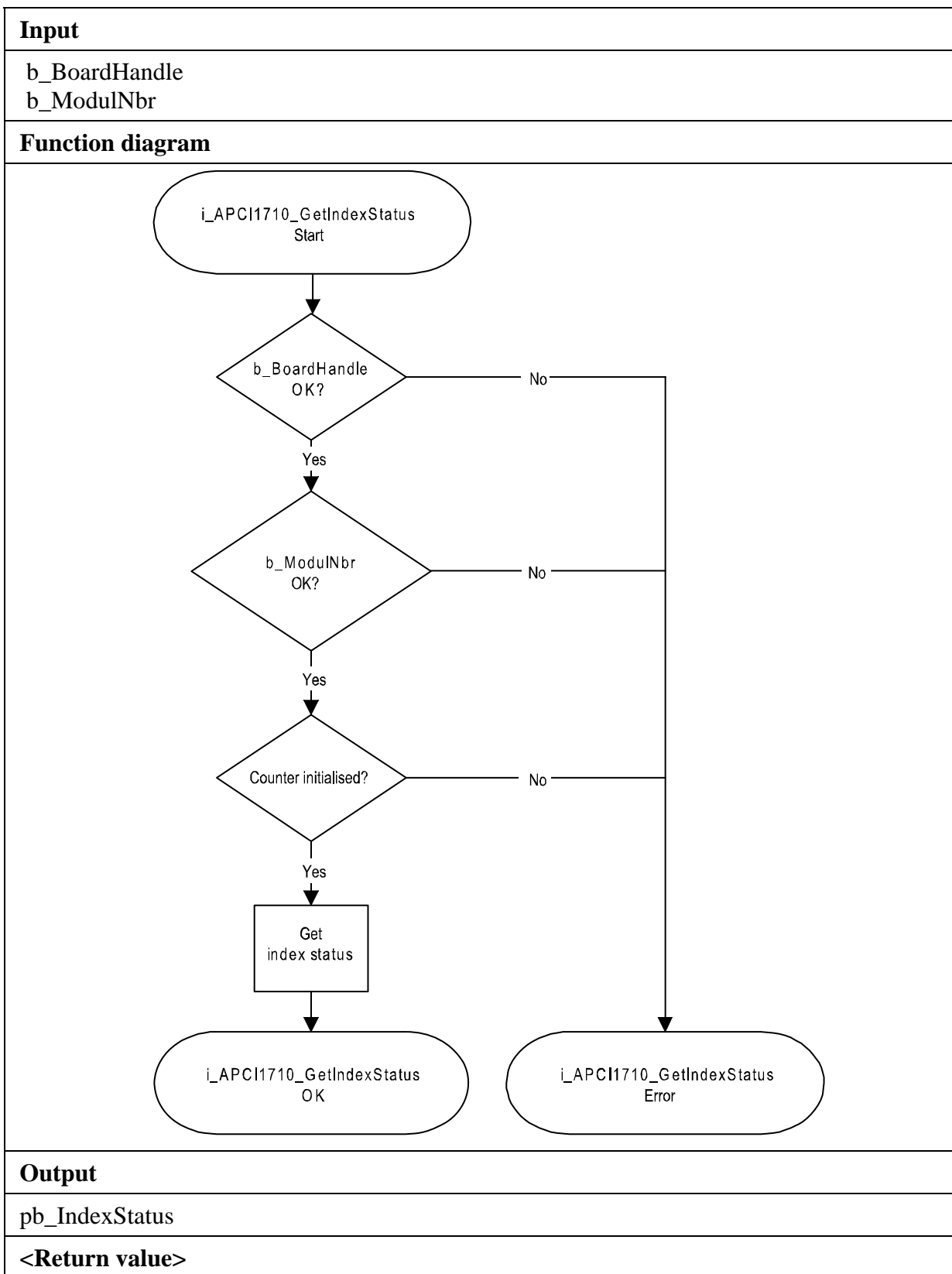
ANSI C:

int	i_ReturnValue;
unsigned char	b_BoardHandle;
unsigned char	b_IndexStatus;

```
i_ReturnValue = i_APCI1710_GetIndexStatus
                (b_BoardHandle,
                 0,
                 &b_IndexStatus);
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: The selected module number is wrong.
- 3: Counter not initialised. See function "i_APCI1710_InitCounter"
- 4: Index not initialised. See function "i_APCI1710_InitIndex"



5) **i_APCI1710_SetIndexAndReferenceSource (...)**

Syntax:

```
<Return Wert> = i_APCI1710_SetIndexAndReferenceSource
                    (BYTE    b_BoardHandle,
                    BYTE    b_ModulNbr,
                    BYTE    b_SourceSelection)
```

Parameter:

-Input:

```
BYTE    b_BoardHandle    Handle of the board xPCI-1710
BYTE    b_ModulNbr       Number of the module to be configured (0 to
                          3)
BYTE    b_SourceSelection Source of the index and of the reference logic
                          See table 3-11
```

-Output:

There is no output

Task:

Determines the hardware source for the index and the reference logic. As standard the index logic is connected to the differential input C and the reference logic to the 24 V of the input E.

Table 3-11: Selection of the index and reference logic source

b_ReferenceLevel	Connection
APCI1710_SOURCE_0	The index logic is connected to the diff. Input C and the reference logic to the 24 V of input E. Standard configuration.
APCI1710_SOURCE_1	Reference logic is connected to diff. input C and the index logic to the 24 V of input E.

Calling convention:

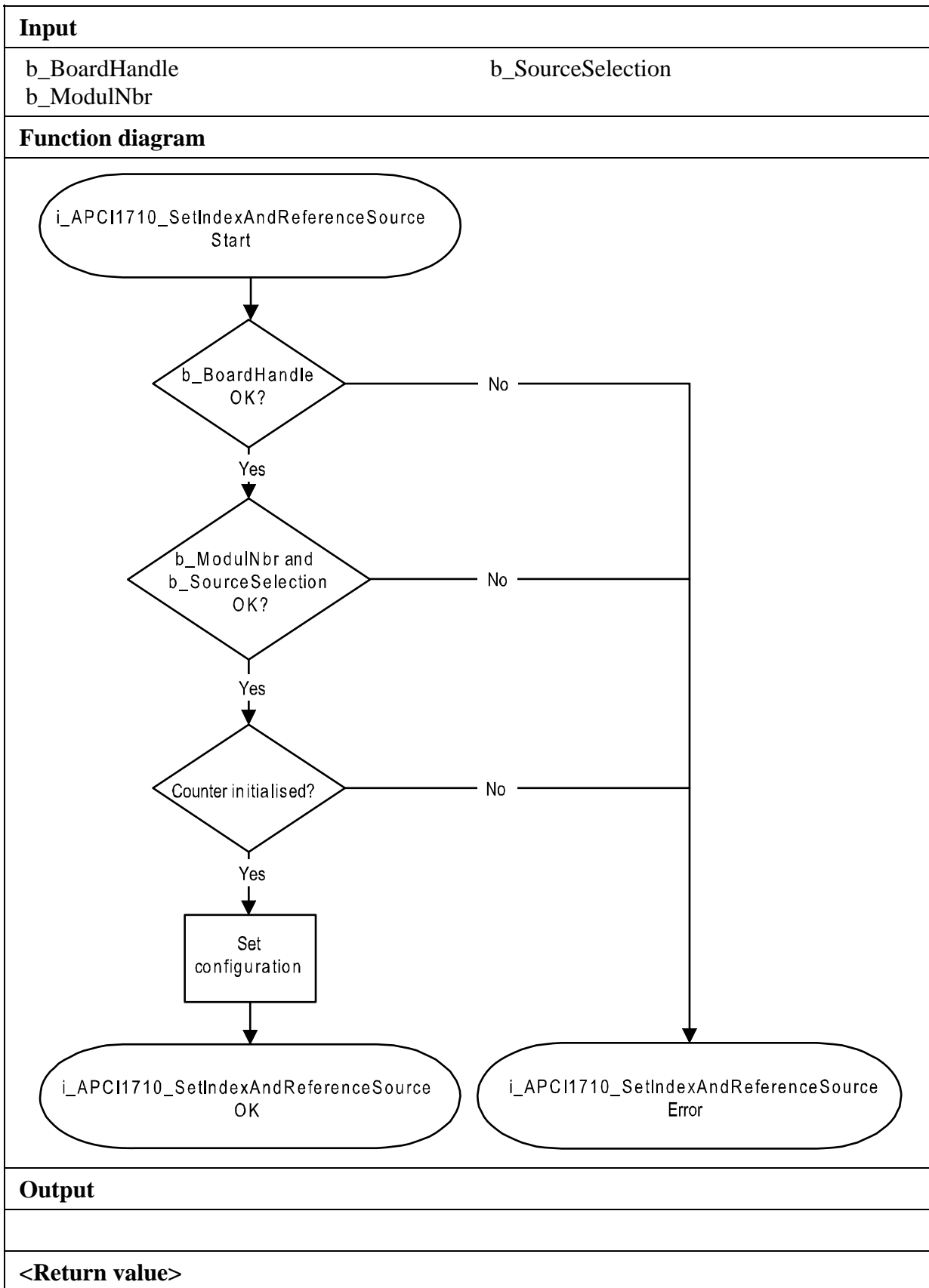
ANSI C :

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

```
i_ReturnValue = i_APCI1710_SetIndexAndReferenceSource
                (b_BoardHandle,
                0,
                APCI1710_SOURCE_0);
```

Return Wert:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: Selected module number is wrong.
- 3: The module is no counter module.
- 4: The source selection is wrong.



3.7 Reference

1) i_APCI1710_InitReference (...)

Syntax:

```
<Return Wert> = i_APCI1710_InitReference
                    (BYTE  b_BoardHandle,
                    BYTE  b_ModulNbr,
                    BYTE  b_ReferenceLevel)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_ReferenceLevel	Reference level. See table 3-12

-Output:

There is no output.

Task:

Initialises the reference that corresponds with the selected module (b_ModulNbr).

Table 3-12: Reference level

b_ReferenceLevel	Description
APCI1710_LOW	If "0", Reference initialised
APCI1710_HIGH	If "1", Reference initialised

Calling convention:

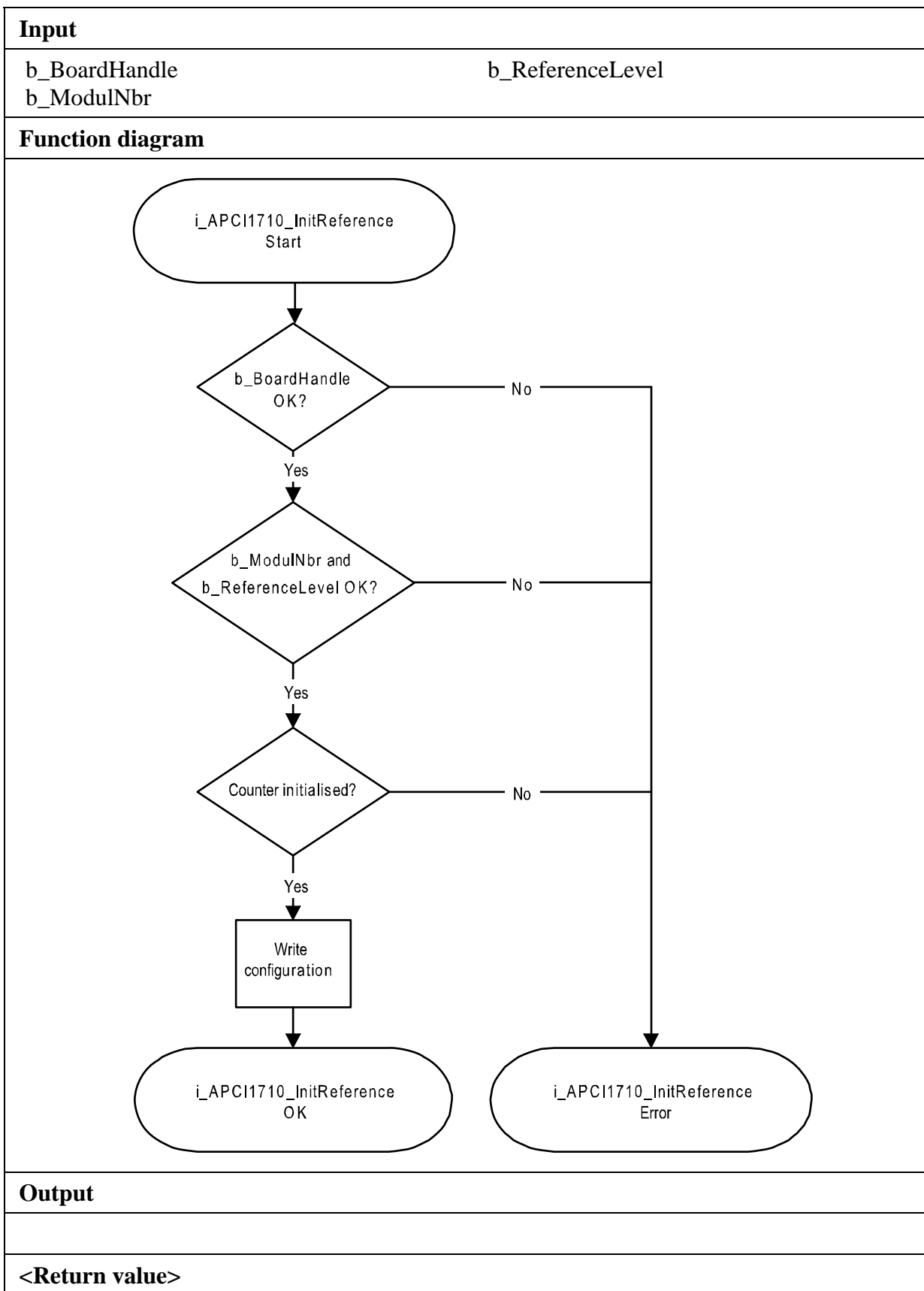
ANSI C:

```
int          i_ReturnValue;
unsigned char b_BoardHandle;

i_ReturnValue = i_APCI1710_InitReference
                (b_BoardHandle,
                0,
                APCI1710_HIGH);
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: Selected module nubmer is wrong.
- 3: Counter not initialised. See function "i_APCI1710_InitCounter"
- 4: Reference level is wrong.



2) i_APCI1710_GetReferenceStatus (...)

Syntax:

```
<Return Wert> = i_APCI1710_GetReferenceStatus
                    (BYTE    b_BoardHandle,
                     BYTE    b_ModulNbr,
                     PBYTE   pb_ReferenceStatus)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

PBYTE	pb_ReferenceStatus	0: No reference 1: A reference was enabled
-------	--------------------	---

Task:

Returns the reference status.

Calling convention:

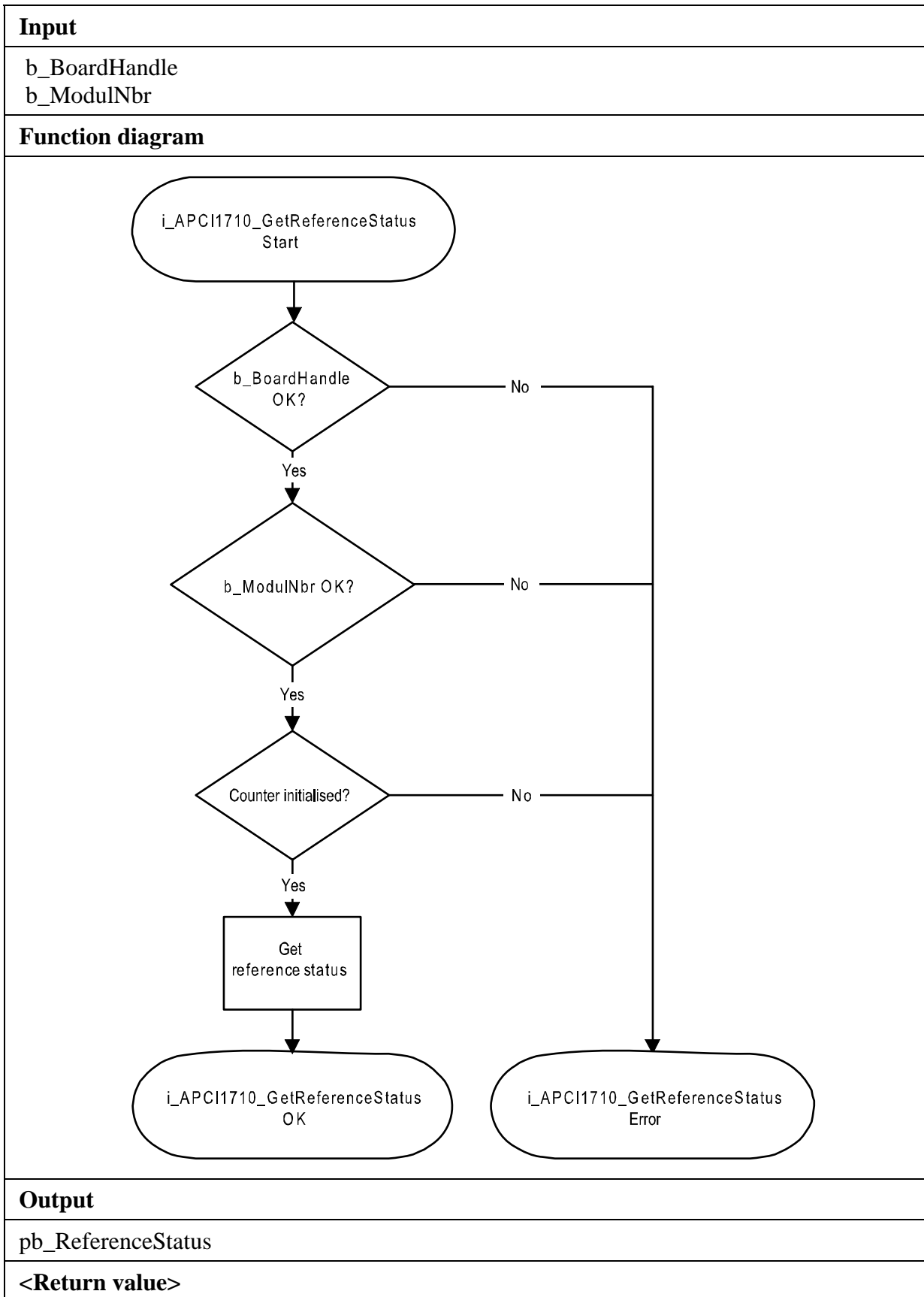
ANSI C:

int	i_ReturnValue;
unsigned char	b_BoardHandle;
unsigned char	b_ReferenceStatus;

```
i_ReturnValue = i_APCI1710_GetReferenceStatus
                (b_BoardHandle,
                 0,
                 &b_ReferenceStatus);
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: The selected module number is wrong.
- 3: Counter not initialised. See function "i_APCI1710_InitCounter"
- 4: Reference not initialised. See function "i_APCI1710_InitReference"



3.8 UAS, CB, U/D#, external pulse (strobe)

1) i_APCI1710_GetUASStatus (...)

Syntax:

```
<Return Wert> = i_APCI1710_GetUASStatus
                    (BYTE b_BoardHandle,
                     BYTE   b_ModulNbr,
                     PBYTE  pb_UASStatus)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

PBYTE	pb_UASStatus	0: UAS is set on Low ("0") 1: UAS is set on High ("1")
-------	--------------	---

Task:

Returns the UAS status.

Calling convention:

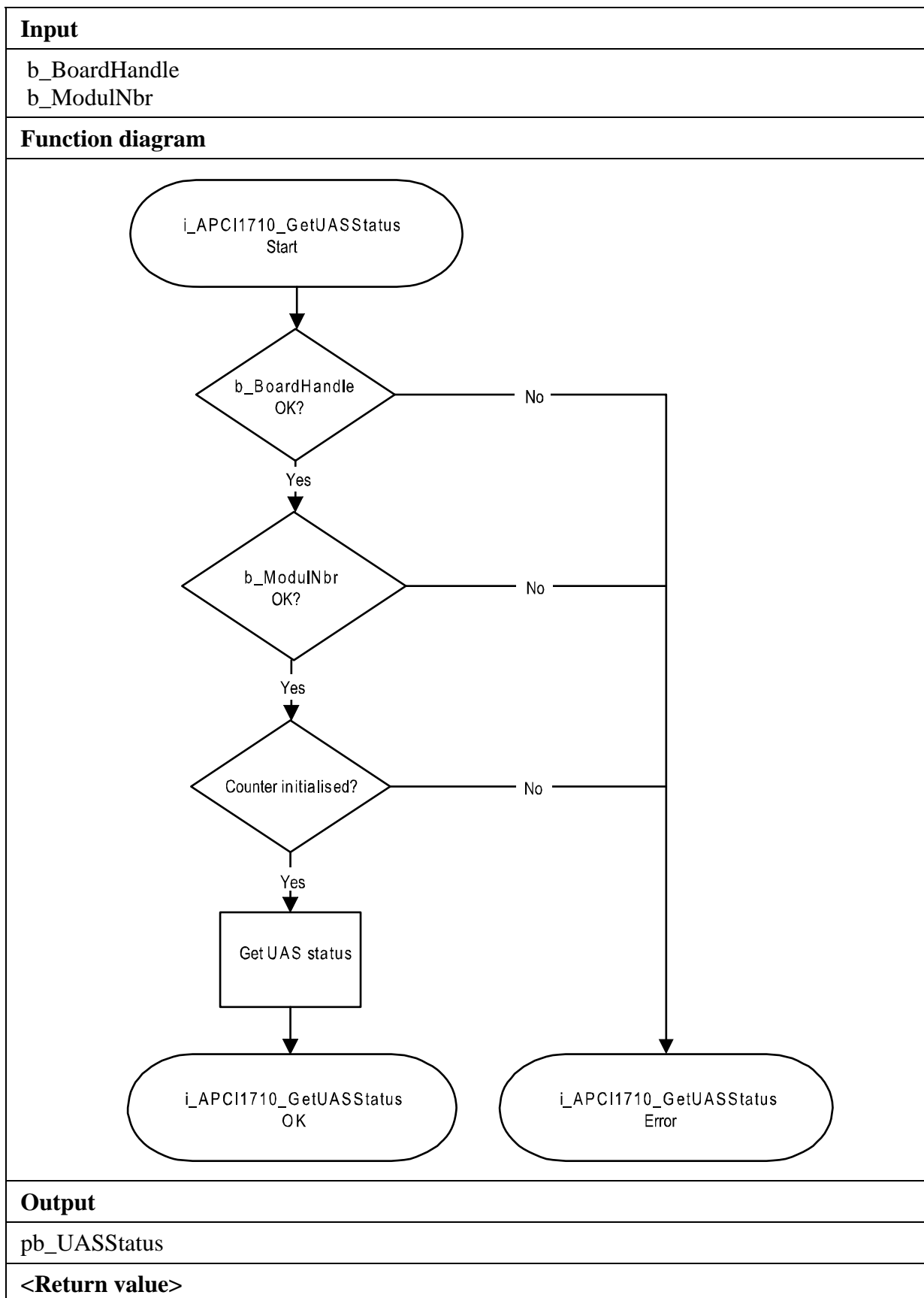
ANSI C:

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

```
i_ReturnValue = i_APCI1710_GetUASStatus
                (b_BoardHandle,
                 0,
                 &b_UASStatus);
```

Return value:

0: No error
-1: Handle parameter of the board is wrong.
-2: Selected module number is wrong.
-3: Counter not initialised. See function "i_APCI1710_InitCounter"



2) i_APCI1710_GetCBStatus (...)

Syntax:

```
<Return Wert> = i_APCI1710_GetCBStatus
                                     (BYTE      b_BoardHandle,
                                     BYTE      b_ModulNbr,
                                     PBYTE     pb_CBStatus)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

PBYTE	pb_CBStatus	0: No counter overflow 1: Counter overflow
-------	-------------	---

Task:

Returns the status of the counter overflow.

Calling convention:

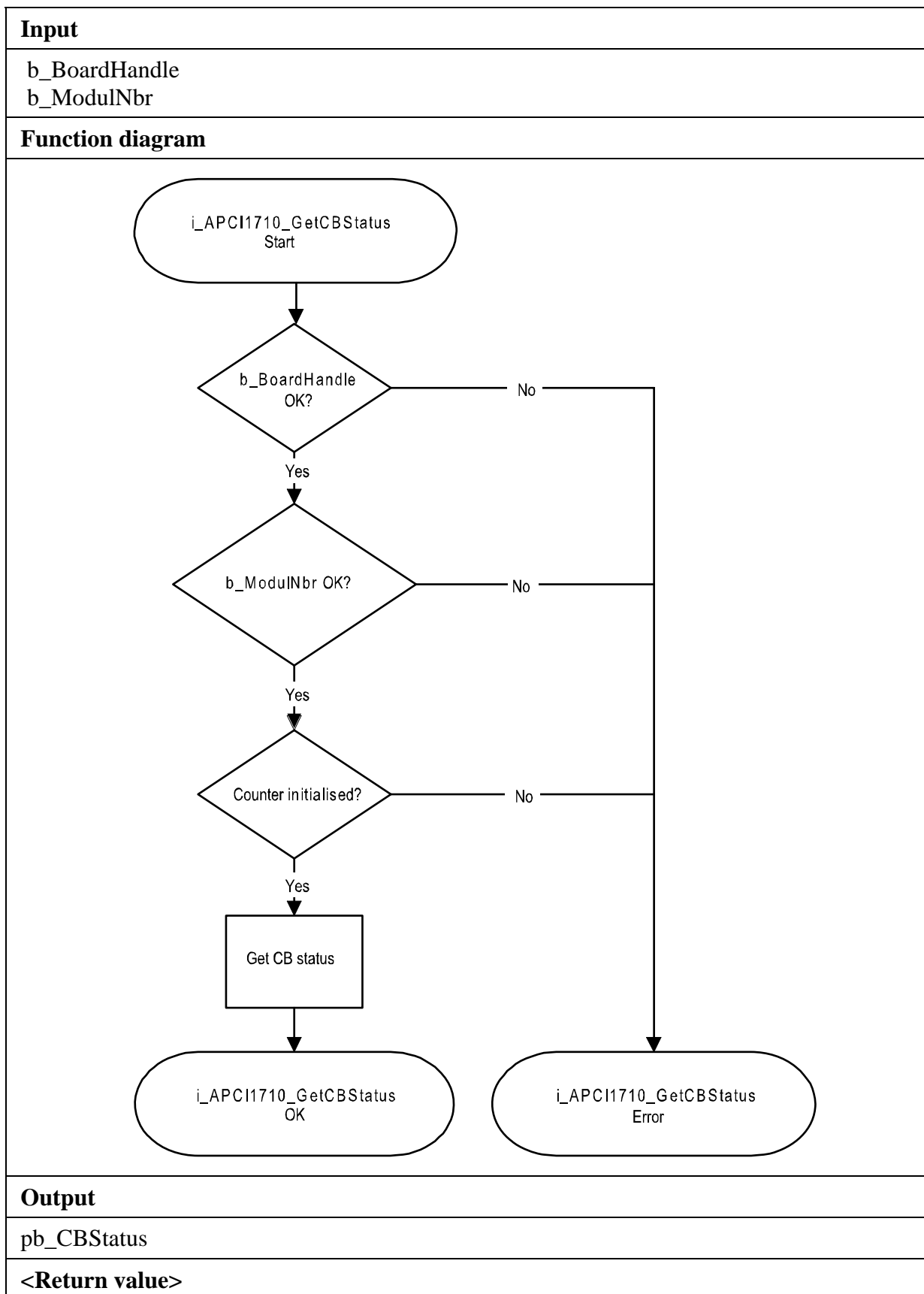
ANSI C:

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

```
i_ReturnValue = i_APCI1710_GetCBStatus
                (b_BoardHandle,
                0,
                &b_CBStatus);
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: Selected module number is wrong.
- 3: Counter not initialised. See function "i_APCI1710_InitCounter"



3) i_APCI1710_GetUDStatus (...)**Syntax:**

```
<Return Wert> = i_APCI1710_GetUDStatus
                    (BYTE    b_BoardHandle,
                     BYTE    b_ModulNbr,
                     PBYTE   pb_UDStatus)
```

Parameter:**-Input:**

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

PBYTE	pb_UDStatus	0: Counter runout in the selected mode See also table 3-5 1: counter runout in the selected mode upwards. See also table 3-5
-------	-------------	---

Task:

Returns the status of the counter runout.

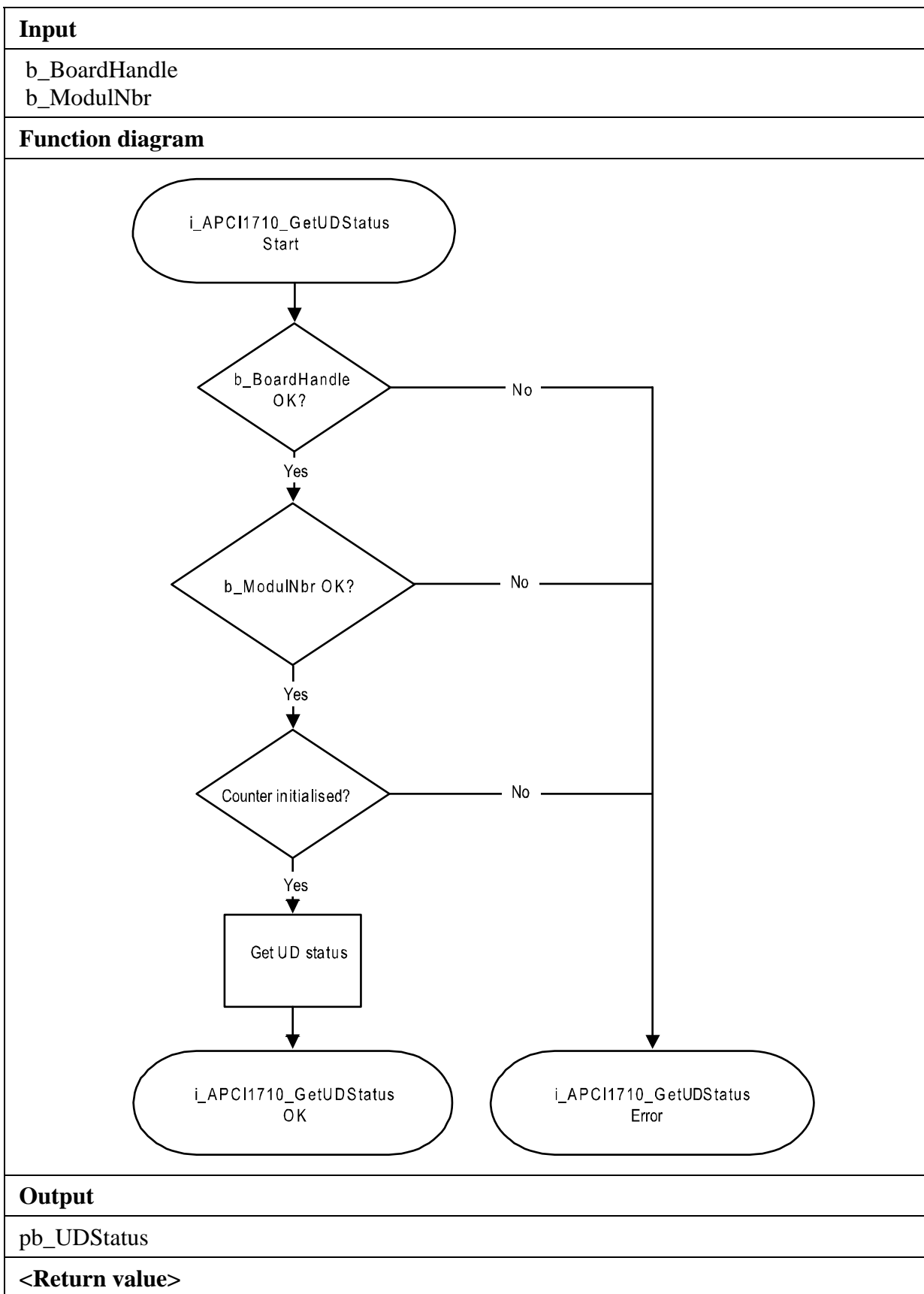
Calling convention:ANSI C:

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

```
i_ReturnValue = i_APCI1710_GetCBStatus
                (b_BoardHandle,
                 0,
                 &b_UDStatus);
```

Return value:

0: No error
-1: Handle parameter of the board is wrong.
-2: The selected module number is wrong.
-3: Counter not initialised. See function "i_APCI1710_InitCounter"



4) i_APCI1710_GetInterruptUDLatchedStatus (...)**Syntax:**

```
<Return Wert> = i_APCI1710_GetInterruptUDLatchedStatus
                    (BYTE b_BoardHandle,
                     BYTE   b_ModulNbr,
                     PBYTE  pb_UDStatus)
```

Parameter:**-Input:**

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

PBYTE	pb_UDStatus	0: Counter runout in the selected mode downwards. 1.: Counter runout in the selected mode upwards. 2: There is no index interrupt See function "i_APCI1710_InitCounter"
-------	-------------	--

Task:

Returns the latched status of the counter runout, after an index interrupt is generated.

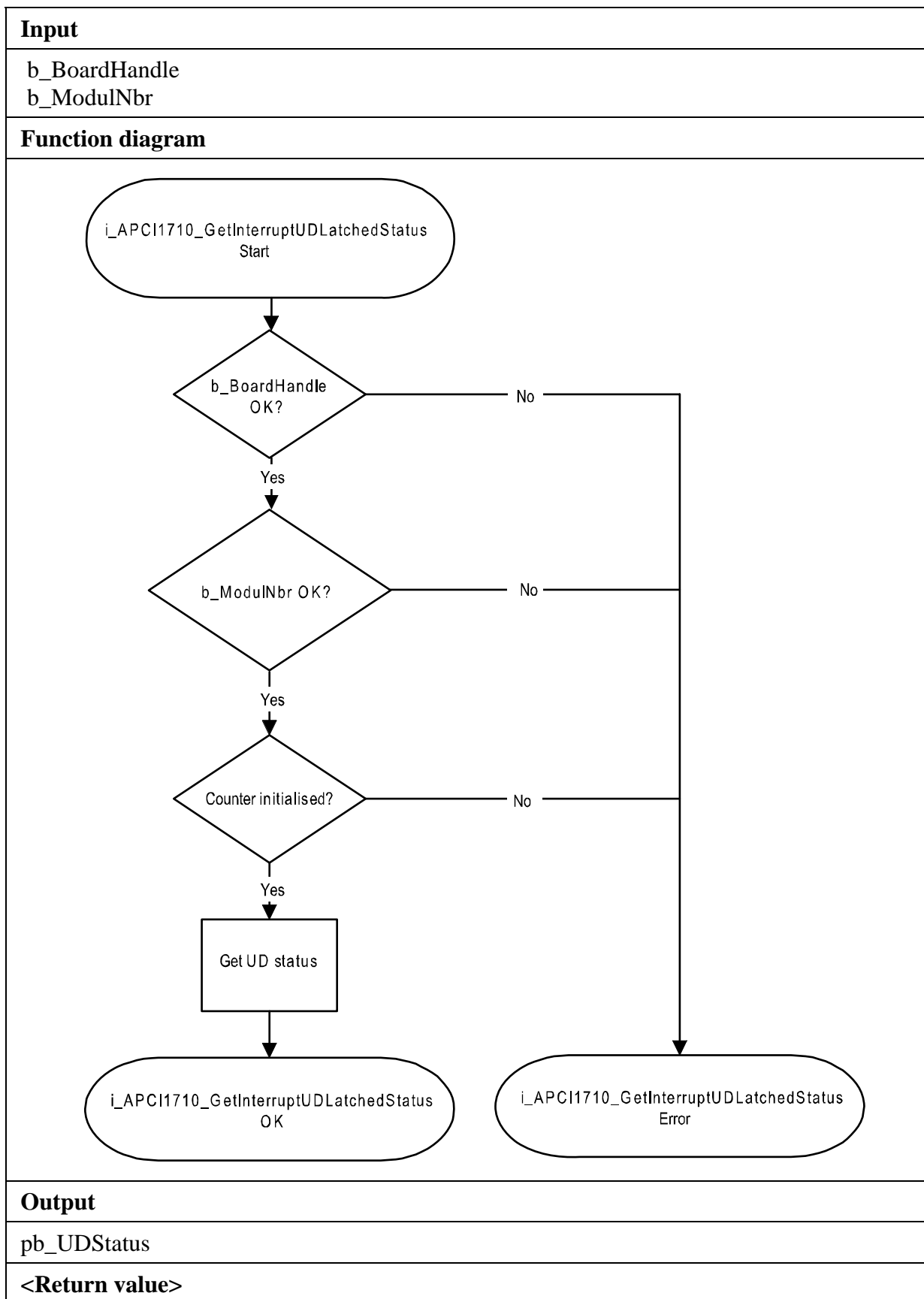
Calling convention:ANSI C:

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

```
i_ReturnValue = i_APCI1710_GetInterruptUDLatchedStatus
                (b_BoardHandle,
                 0,
                 &b_UDStatus);
```

Return value:

0: No error
-1: Handle parameter of the board is wrong.
-2: Selected module number is wrong.
-3: Counter not initialised. See function "i_APCI1710_InitCounter"
-4: Interrupt function not initialised.
See function "i_APCI1710_SetBoardIntRoutineX"



5) **i_APCI1710_InitExternalStrobe (...)**

Syntax:

```
<Return Wert> = i_APCI1710_InitExternalStrobe
                                     (BYTE      b_BoardHandle,
                                     BYTE      b_ModulNbr,
                                     BYTE      b_ExternalStrobe,
                                     BYTE      b_ExternalStrobeLevel)
```

Parameter:

-Input:

```
BYTE    b_BoardHandle    Handle of the board xPCI-1710
BYTE    b_ModulNbr       Number of the module to be configured (0 to
                           3)
BYTE    b_ExternalStrobe Selection of the external pulse (strobe)
                           0: External pulse A
                           1: External pulse B
BYTE    b_ExternalStrobeLevel Level of the external pulse.
                           See table 3-13
```

-Output:

There is no output.

Task:

Initialises the external pulse level for the selected module (*b_ModulNbr*).

Table 3-13: External pulse level

<i>b_ReferenceLevel</i>	Description
APCI1710_LOW	External latch is generated at „0“.
APCI1710_HIGH	External latch is generated at „1“ (standard configuration)

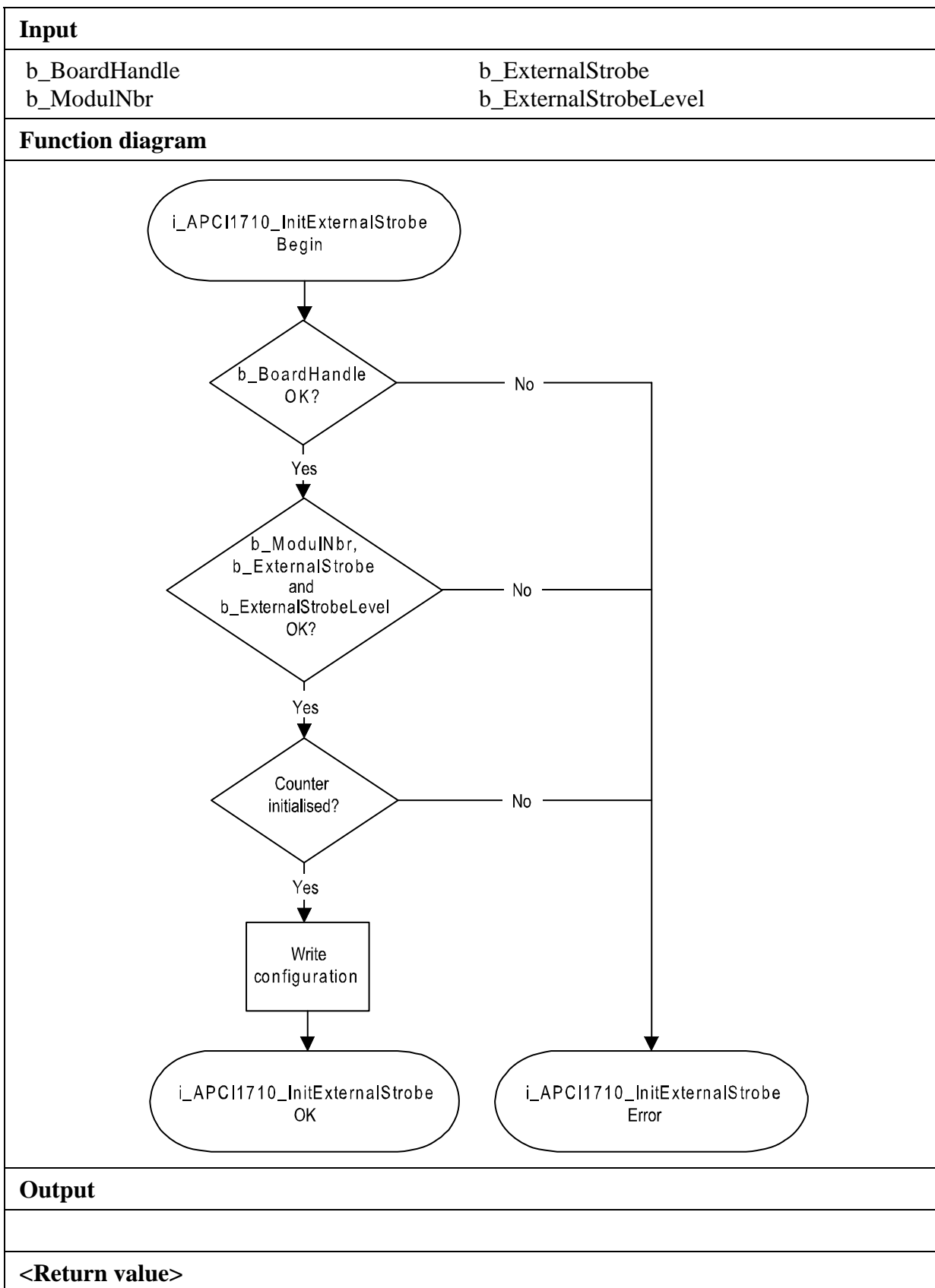
Calling convention:

ANSI C:

```
int          i_ReturnValue;
unsigned char b_BoardHandle;
i_ReturnValue = i_APCI1710_InitExternalStrobe
               (b_BoardHandle,
               0,
               0,
               APCI1710_HIGH);
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: Selected module number is wrong.
- 3: Counter not initialised. See function "i_APCI1710_InitCounter"
- 4: Selected external pulse is wrong.
- 5: External pulse level is wrong.



3.9 Compare logic

1) i_APCI1710_InitCompareLogic (...)

Syntax:

```
<Return Wert> = i_APCI1710_InitCompareLogic
                    (BYTE  b_BoardHandle,
                     BYTE  b_ModulNbr,
                     UINT  ui_CompareValue)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
UINT	ui_CompareValue	32-bit compare value

-Output:

There is no output.

Task:

Sets the 32-bit compare value. An interrupt is generated as soon as the counter has reached the compare value (*ui_CompareValue*).

Calling convention:

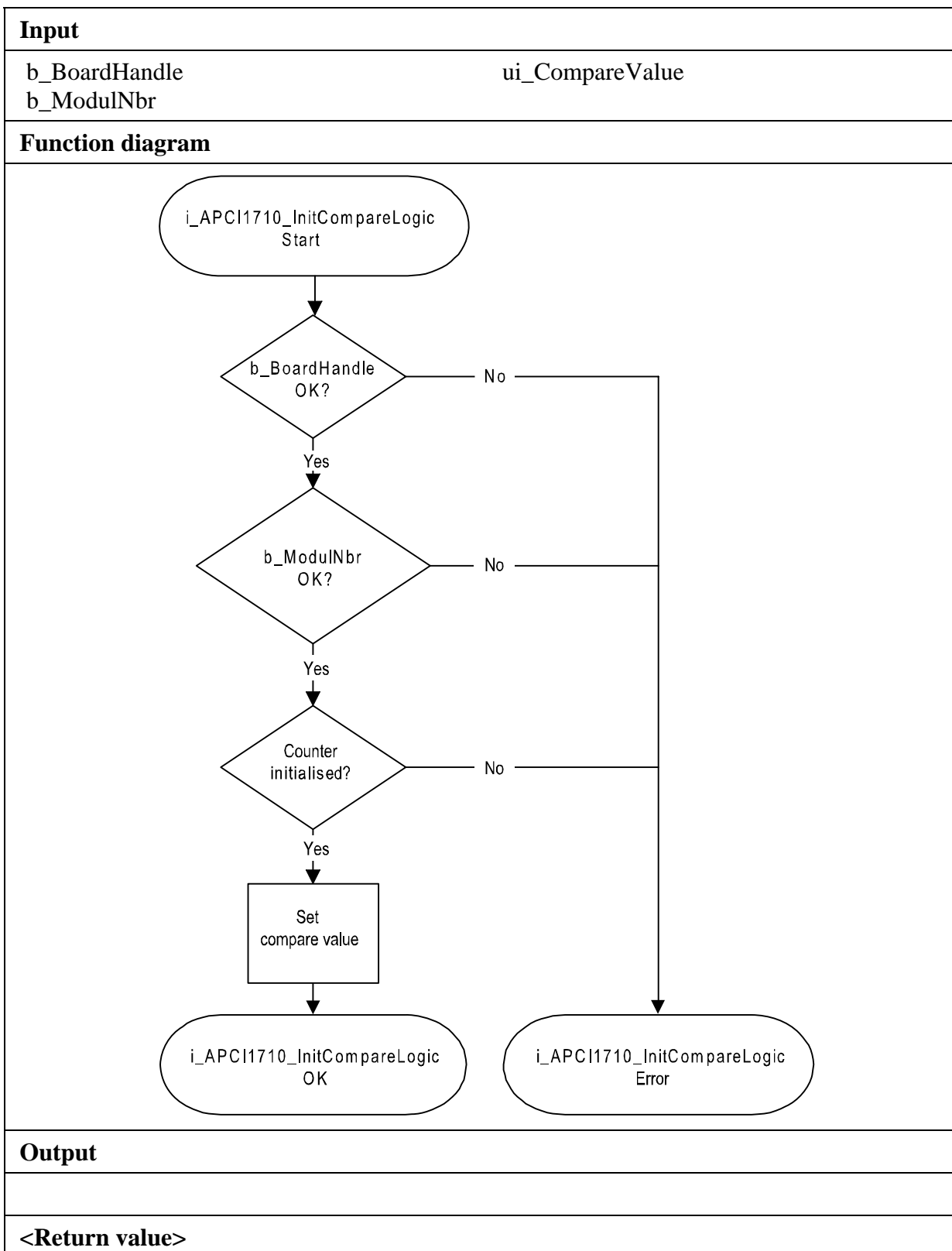
ANSI C:

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

```
i_ReturnValue = i_APCI1710_InitCompareLogic (b_BoardHandle,
                                             0,
                                             0xFF55);
```

Return value:

0: No error
 -1: Handle parameter of the board is wrong.
 -2: Selected module number is wrong.
 -3: Counter not initialised. See function "i_APCI1710_InitCounter"



2) i_APCI1710_EnableCompareLogic (...)

Syntax:

```
<Return Wert> = i_APCI1710_EnableCompareLogic
                    (BYTE b_BoardHandle,
                     BYTE b_ModulNbr)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

There is no output.

Task:

Enables the 32-bit compare logic. An interrupt is generated, as soon as the counter has reached the compare value (*ui_CompareValue*).

Calling convention:

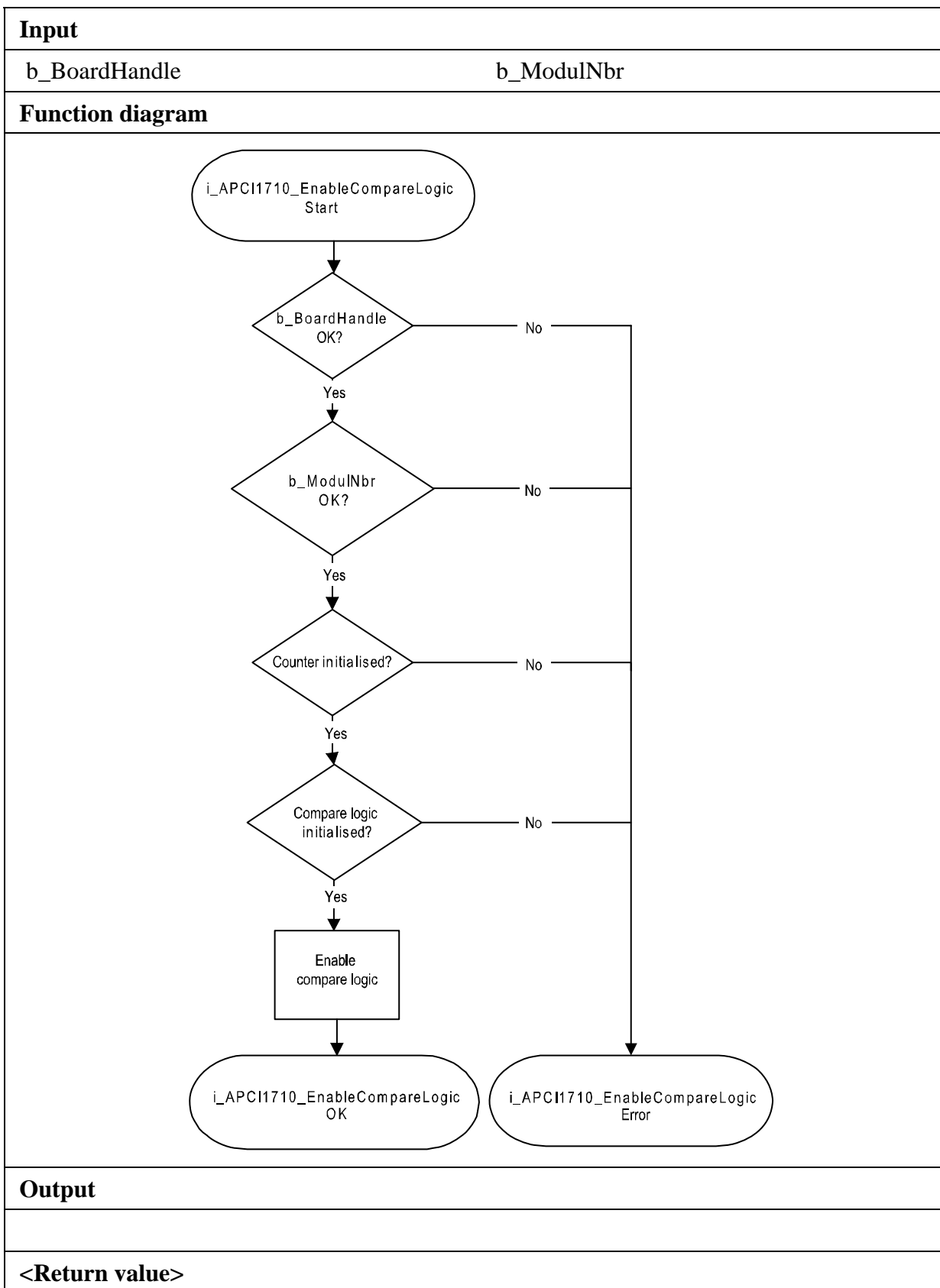
ANSI C:

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

```
i_ReturnValue = i_APCI1710_EnableCompareLogic (b_BoardHandle,
                                                0);
```

Return value:

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: Selected module number is wrong.
- 3: Counter not initialised. See function "i_APCI1710_InitCounter"
- 4: Compare logic not initialised.
See function "i_APCI1710_InitCompareLogic"
- 5: Interrupt function not initialised
See function "i_APCI1710_SetBoardIntRoutineX"



3) i_APCI1710_DisableCompareLogic (...)

Syntax:

```
<Return Wert> = i_APCI1710_DisableCompareLogic
                    (BYTE    b_BoardHandle,
                     BYTE    b_ModulNbr)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

There is no output.

Task:

Disables the 32-bit compare logic.

Calling convention:

ANSI C:

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

```
i_ReturnValue = i_APCI1710_DisableCompareLogic    (b_BoardHandle,
                                                    0);
```

Return value:

0: No error

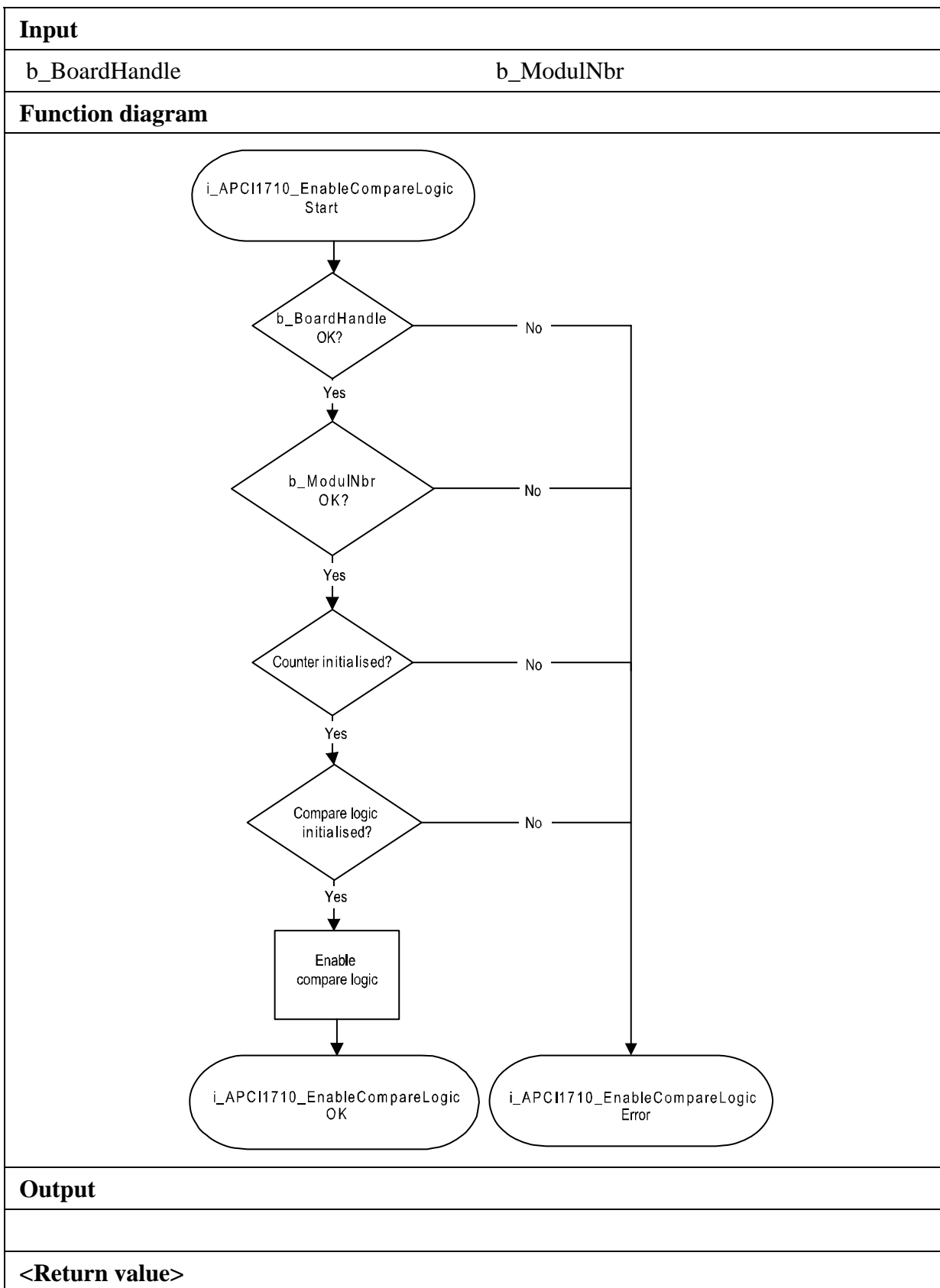
-1: Handle parameter of the board is wrong.

-2: Selected module number is wrong

-3: Counter not initialised. See function "i_APCI1710_InitCounter"

-4: Compare logic is not initialised. See function

"i_APCI1710_InitCompareLogic"



3.10 Frequency measurement

1) i_APCI1710_InitFrequencyMeasurement (...)

Syntax:

```
<Return Wert> = i_APCI1710_InitFrequencyMeasurement
                    (BYTE      b_BoardHandle,
                     BYTE      b_ModulNbr,
                     BYTE      b_PCIInputClock,
                     BYTE      b_TimeUnit,
                     ULONG     ul_TimeInterval,
                     PULONG    pul_RealTimeInterval)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_PCIInputClock	Selection of the PCI bus clock - APCI1710_30MHZ: The PC has a PCI bus clock of 30 MHz - APCI1710_33MHZ: The PC has a PCI bus clock of 33 MHz - APCI1710_40MHZ: The PC has a PCI bus clock of 40 MHz
BYTE	b_TimeUnit	Unit of the time base (0 to 2) 0: ns 1: μ s 2: ms
ULONG	ul_TimeInterval	Value of the time base. See table "Value of the time base"

-Output:

PULONG	pul_RealTimeInterval	Correct value of the time base. Returns the value, that is similar to the value entered in ul_TimingInterval
--------	----------------------	--

Table 3-14: Value of the time base

PCI bus clock	<i>b_TimingUnit</i>	<i>ul_TimingInterval</i> Min. value	<i>ul_TimingInterval</i> Max. value
30 MHz	ns (0)	266	8738133
	µs (1)	1	8738
	ms (2)	1	8
33 MHz	ns (0)	242	7943757
	µs (1)	1	7943
	ms (2)	1	7
40 MHz	ns (0)	200	6553500
	µs (1)	1	6553
	ms (2)	1	6

Task:

Sets the time for frequency measurement.

Configures the incremental counter of the selected module (*b_ModulNbr*). The parameters *ul_TimingInterval* and *ul_TimingUnit* determine the time base for the measurement. *pul_RealTimingInterval* returns the correct time value. Call this function before calling another function, which accesses the frequency measurement.

Calling convention:ANSI C:

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

```
i_ReturnValue = i_APCI1710_InitFrequencyMeasurement
                (b_BoardHandle,
                 0,
                 APCI1710_33MHZ,
                 2,
                 1,
                 &ul_RealTimingInterval);
```

Return value:

0: No error

-1: Handle parameter of the board is wrong.

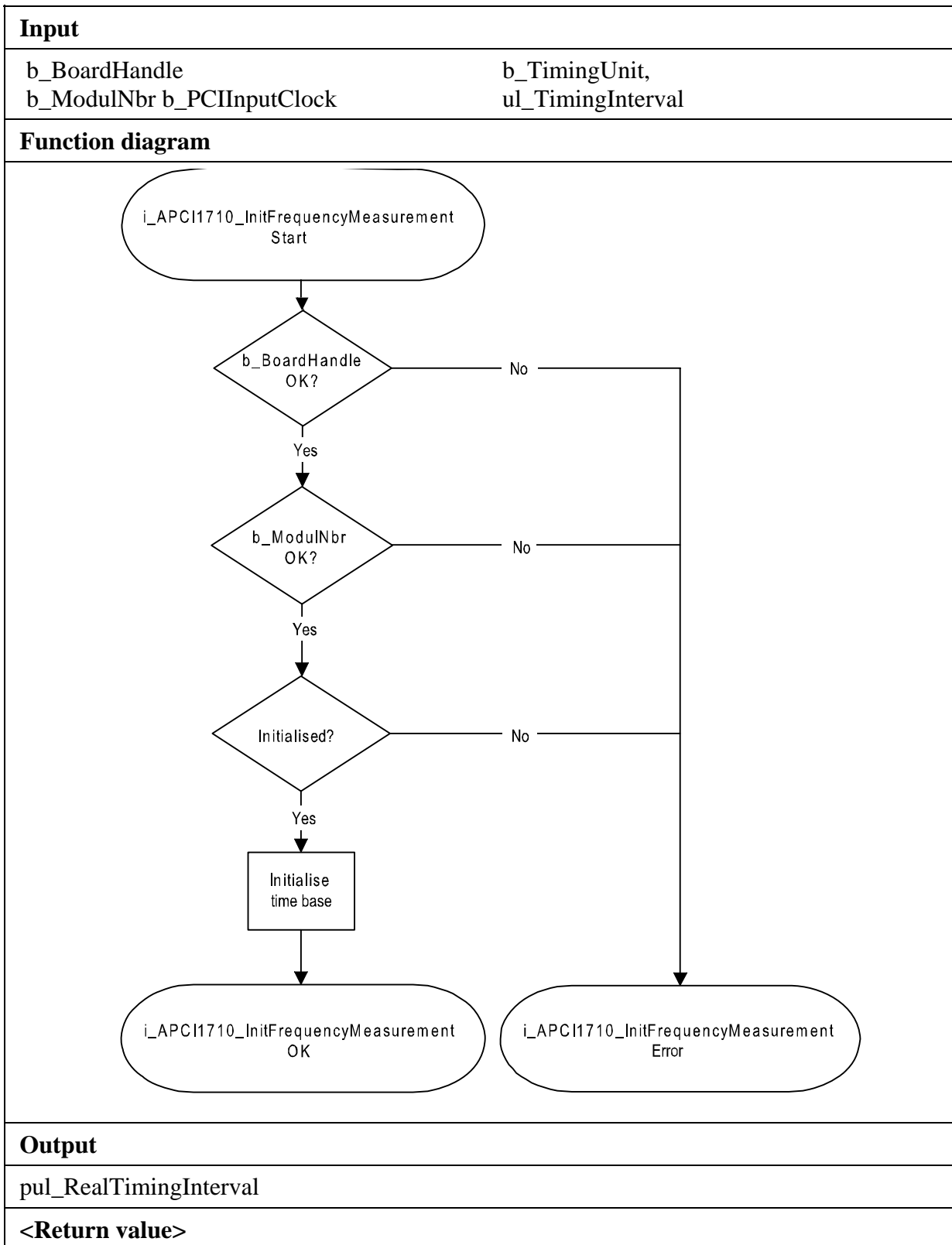
-2: Selected module number is wrong.

-3: Counter not initialised. See function "i_APCI1710_InitCounter"

-4: The selected PCI input clock is wrong.

-5: Selected time unit is wrong

- 6: Selected time base is wrong.
- 7: There is no 40 MHz quartz implemented on the board.



2) i_APCI1710_EnableFrequencyMeasurement (...)

Syntax:

```
<Return Wert> = i_APCI1710_EnableFrequencyMeasurement
                    (BYTE    b_BoardHandle,
                     BYTE    b_ModulNbr,
                     BYTE    b_InterruptEnable)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_InterruptEnable	Enables or disables the interrupt APCI1710_ENABLE: Interrupt enabled APCI1710_DISABLE: Interrupt disabled

-Output:

There is no output.

Task:

Enables the function for the frequency measurement

Calling convention:

ANSI C:

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

```
i_ReturnValue = i_APCI1710_EnableFrequencyMeasurement
                (b_BoardHandle,
                 0,
                 APCI1710_DISABLE);
```

Return value:

0: No error

-1: Handle parameter of the board is wrong.

-2: Selected module number is wrong.

-3: Counter not initialised. See function "i_APCI1710_InitCounter"

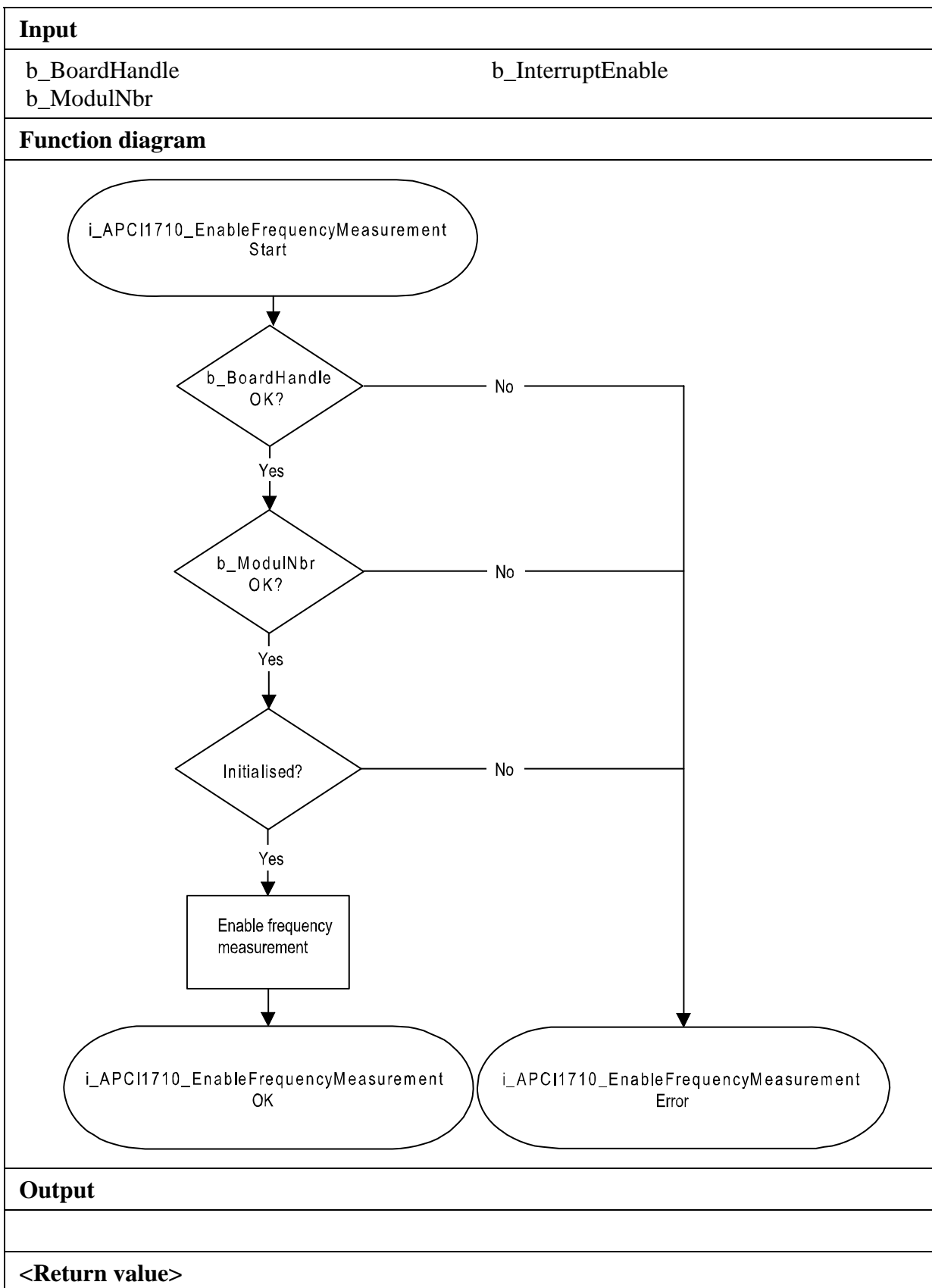
-4: The logic frequency measurement is not initialised.

See function "i_APCI1710_InitFrequencyMeasurement"

-5: Interrupt parameter is wrong.

-6: Interrupt function not initialised.

See function "i_APCI1710_SetBoardIntRoutineX"



3) i_APCI1710_ReadFrequencyMeasurement (...)

Syntax:

```
<Return Wert> = i_APCI1710_ReadFrequencyMeasurement
                    (BYTE      b_BoardHandle,
                    BYTE      b_ModulNbr,
                    PBYTE     pb_Status,
                    PBYTE     pb_UDStatus,
                    PULONG    pul_ReadValue)
```

Parameter:

-Input:

BYTE b_BoardHandle Handle of the board xPCI-1710
 BYTE b_ModulNbr Number of the module to be configured (0 to 3)

-Output:

PBYTE pb_Status Returns the status of the frequency measurement
 0: Counting process not started.
 1: Counting process started.
 2: Counting process over.
 PBYTE pb_UDStatus Status of the upwards/downwards counter.
 See table 3-15
 PULONG pul_ReadValue Returns the number of increments within the time base.

Table 3-15: Status of the counter operation

<i>pb_UDStatus</i>	2 x 16-bit counter mode		1 x 32-bit counter mode
	Second 16-bit counter	First 16-bit counter	
0	Counter operation downwards	Counter operation downwards	Counter operation downwards
1	Counter operation downwards	Counter operation upwards	X
2	Counter operation upwards	Counter operation downwards	X
3	Counter operation upwards	Counter operation upwards	Counter operation upwards

Task:

Returns the status (*pb_Status*) and the number of increments in the set time. See function "i_APCI1710_InitFrequencyMeasurement"

Calling convention:ANSI C :

```
int                i_ReturnValue;
unsigned char      b_BoardHandle;
unsigned char      b_Status;
unsigned char      b_UDStatus;
unsigned long      ul_ReadValue;
```

```
i_ReturnValue = i_APCI1710_ReadFrequencyMeasurement
                (b_BoardHandle,
                 0,
                 &b_Status,
                 &b_UDStatus,
                 &ul_ReadValue);
```

Return value:

0: No error

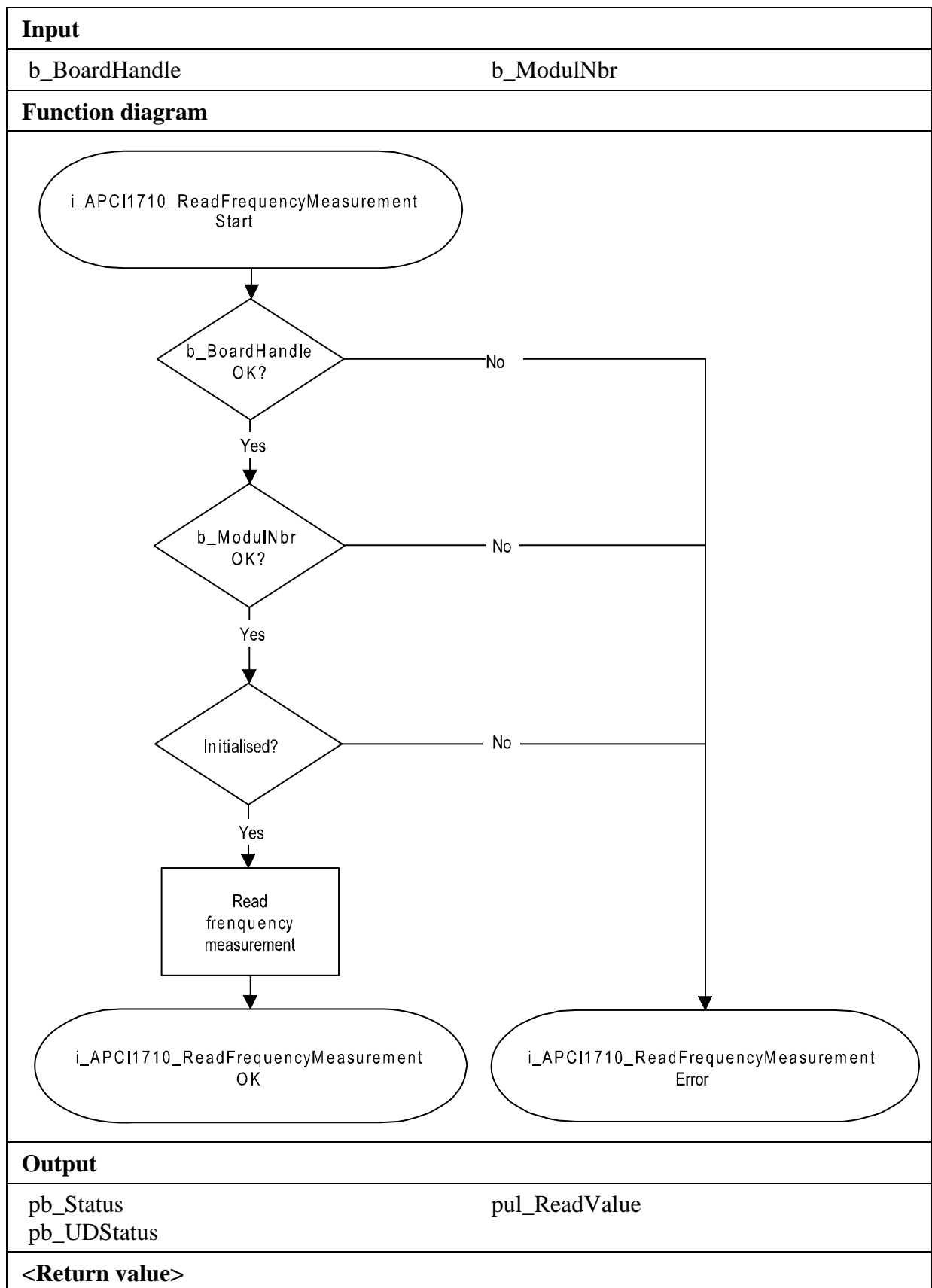
-1: Handle parameter of the board is wrong.

-2: Selected module number is wrong.

-3: Counter not initialised. See function "i_APCI1710_InitCounter"

-4: Logic of the frequency measurement is not initialised.

See function "i_APCI1710_InitFrequencyMeasurement"



4) i_APCI1710_DisableFrequencyMeasurement (...)

Syntax:

```
<Return Wert> = i_APCI1710_DisableFrequencyMeasurement  
                (BYTE    b_BoardHandle,  
                 BYTE    b_ModulNbr)
```

Parameter:**-Input:**

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

There is no output.

Task:

Disables the frequency measurement.

Calling convention:ANSI C:

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

```
i_ReturnValue = i_APCI1710_DisableFrequencyMeasurement  
                (b_BoardHandle,  
                 0);
```

Return value:

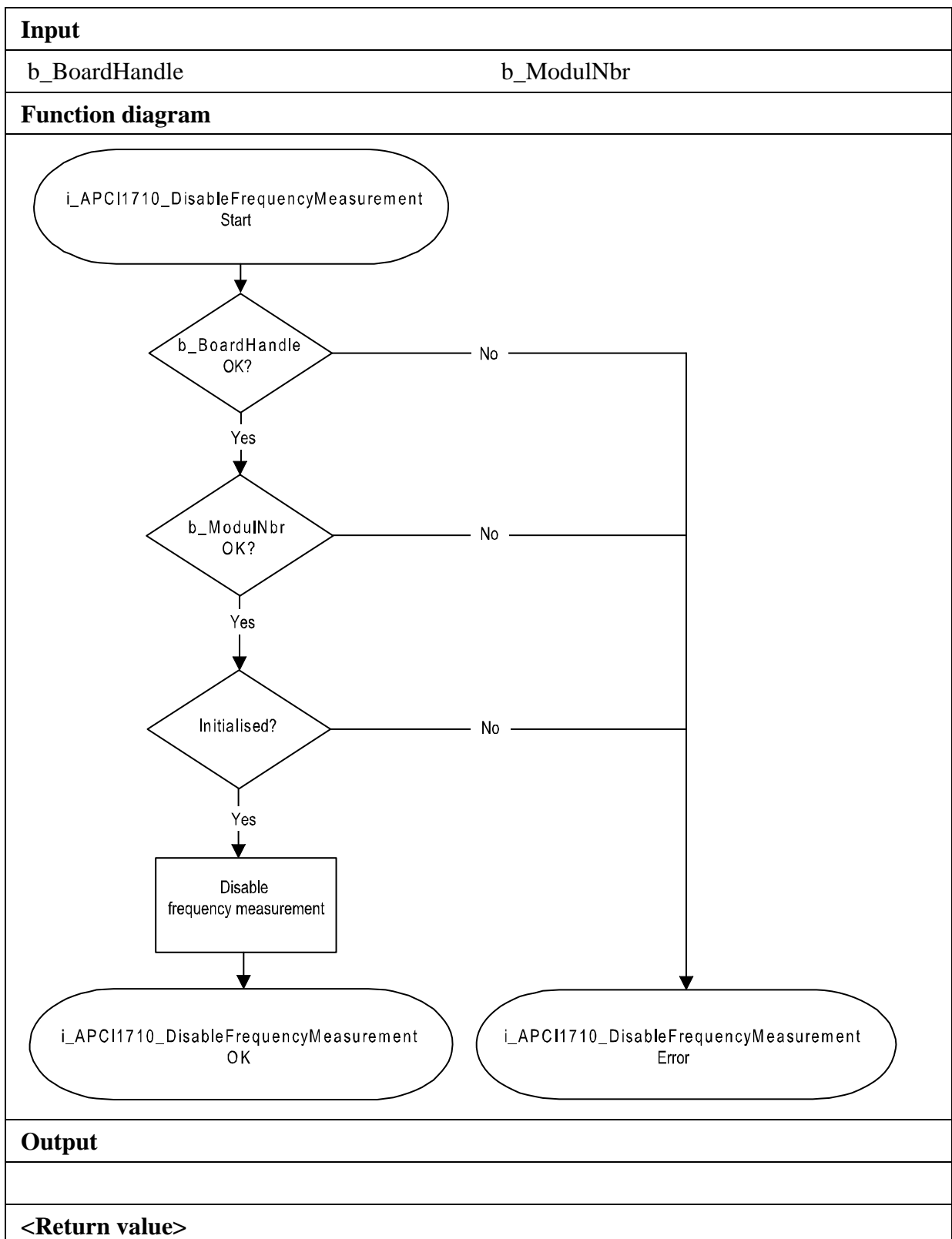
0: No error

-1: Handle parameter of the board is wrong.

-2: Selected module number is wrong.

-3: Counter not initialised. See function "i_APCI1710_InitCounter"

-4: Logic of the frequency measurement is not initialised. See function "i_APCI1710_InitFrequencyMeasurement"



3.11 Digital output

1) i_APCI1710_SetDigitalChlOn (...)

Syntax:

```
<Return Wert> = i_APCI1710_SetDigitalChlOn  
                                     (BYTE  b_BoardHandle,  
                                     BYTE  b_ModulNbr)
```

Parameter:**-Input:**

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

There is no output.

Task:

Sets the digital output H. Setting an output means setting it on „High“.

Calling convention:ANSI C:

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

```
i_ReturnValue = i_APCI1710_SetDigitalChlOn (b_BoardHandle,  
                                             0);
```

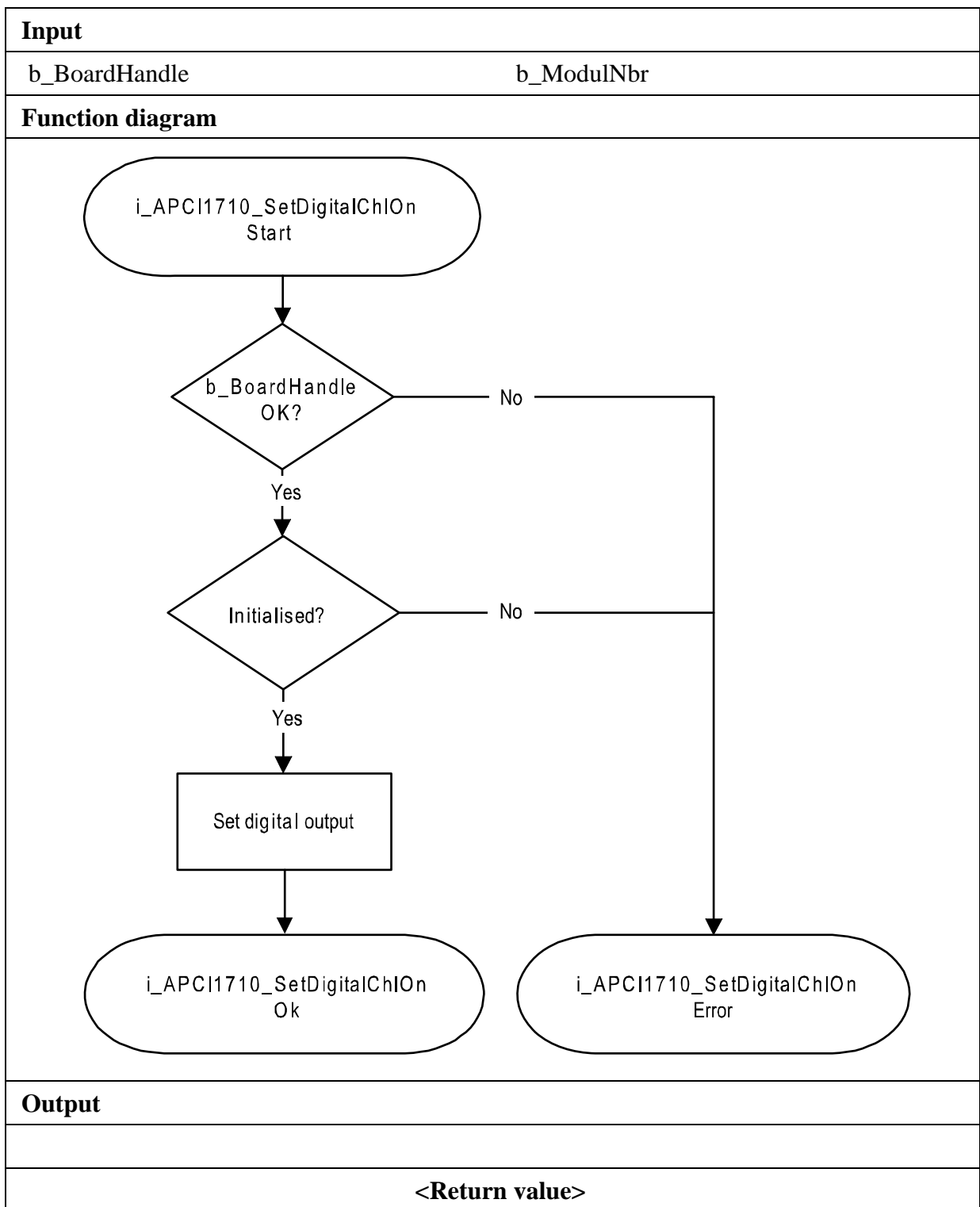
Return value:

0: No error

-1: Handle parameter of the board is wrong.

-2: Selected module number is wrong.

-3: Counter not initialised. See function "i_APCI1710_InitCounter"



2) i_APCI1710_SetDigitalChlOff (...)

Syntax:

```
<Return Wert> = i_APCI1710_SetDigitalChlOff  
                                     (BYTE   b_BoardHandle,  
                                     BYTE   b_ModulNbr)
```

Parameter:

-Input:

BYTE	b_BoardHandle	Handle of the board xPCI-1710
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

There is no output.

Task:

Resets the digital output H. Resetting an output means setting it on „Low“.

Calling convention:

ANSI C:

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

```
i_ReturnValue = i_APCI1710_SetDigitalChlOff (b_BoardHandle,  
                                             0);
```

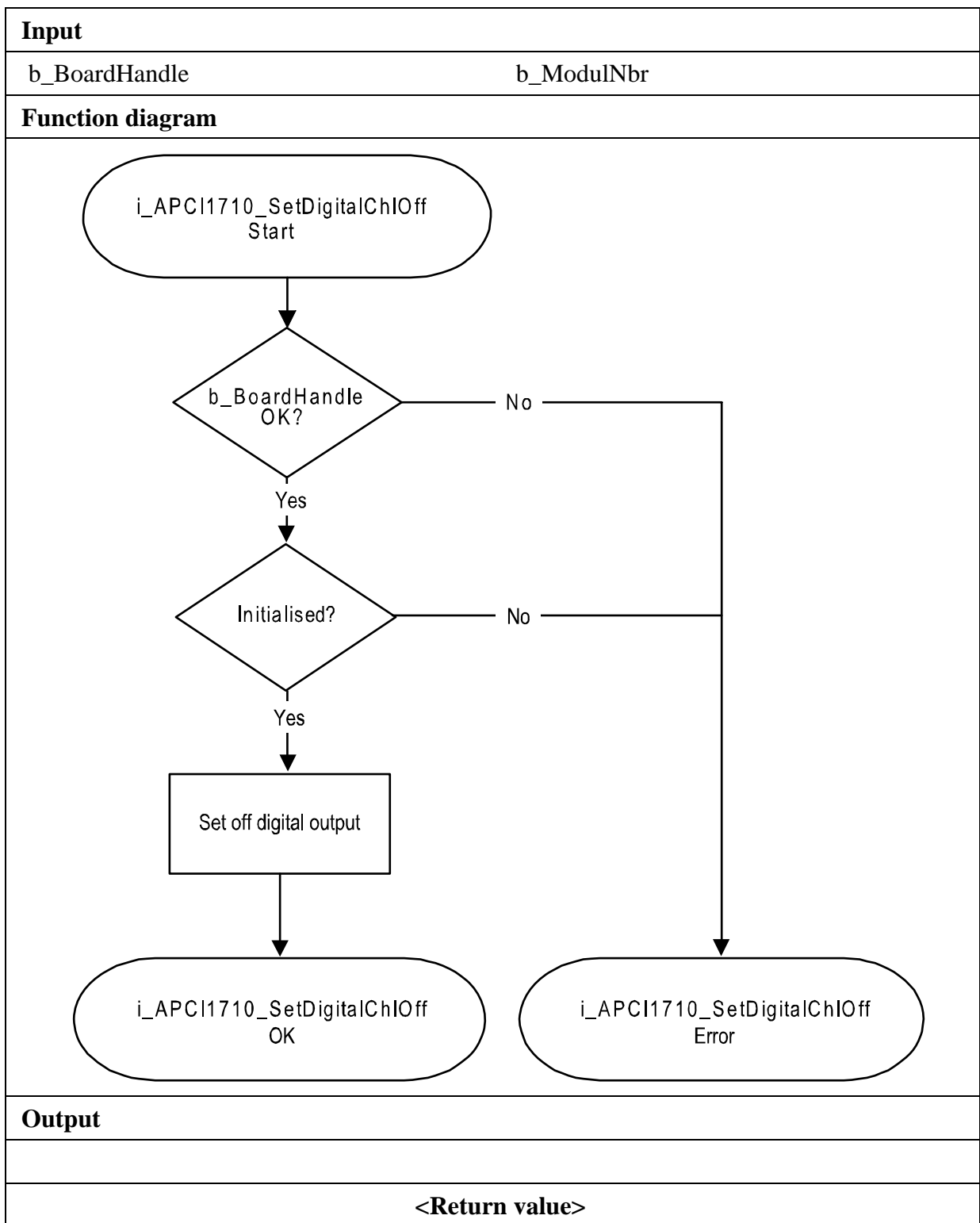
Return value:

0: No error

-1: Handle parameter of the board is wrong.

-2: Selected module number is wrong.

-3: Counter not initialised. See function "i_APCI1710_InitCounter"



3.12 Using functions in the kernel module

i

IMPORTANT!

These functions are only available for the user interrupt routine under Windows NT and Windows 95/98 in the synchronous mode. See function "i_APCI1710_SetBoardIntRoutineWin32"

1) i_APCI1710_KRNL_ClearCounterValue (...)

Syntax:

```
<Return Wert> = i_APCI1710_KRNL_ClearCounterValue
                (UINT  ui_BaseAddress,
                 BYTE  b_ModulNbr)
```

Parameter:

-Input:

UINT	ui_BaseAddress	Base address of the xPCI-1710. See "i_APCI1710_GetHardwareInformation"
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

There is no output.

Task:

Deletes the counter value of the selected module (*b_ModulNbr*).

Calling convention:

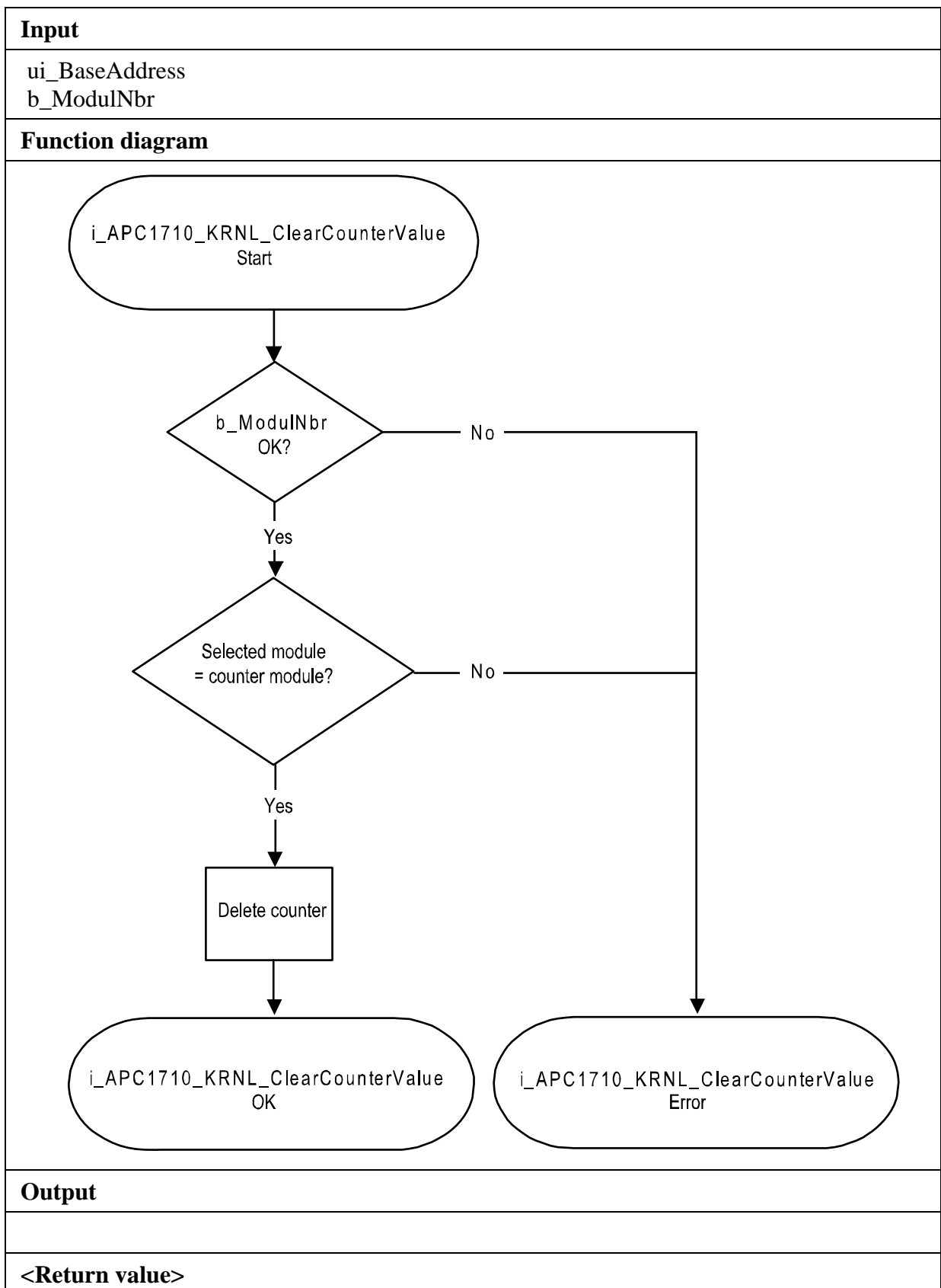
ANSI C:

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

```
i_ReturnValue = i_APCI1710_KRNL_ClearCounterValue
                (ui_BaseAddress,
                 0);
```

Return value:

0: No error
-1: Selected module number is wrong.
-2: The module is no counter module.



2) i_APCI1710_KRNL_Read16BitCounterValue (...)

Syntax:

```
<Return Wert> = i_APCI1710_KRNL_Read16BitCounterValue
                (UINT    ui_BaseAddress,
                 BYTE    b_ModulNbr,
                 BYTE    b_SelectedCounter,
                 PUINT   pui_CounterValue)
```

Parameter:

-Input:

UINT	ui_BaseAddress	Base address of the xPCI-1710. See "i_APCI1710_GetHardwareInformation"
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_SelectedCounter	Selected 16-bit counter (0 or 1)

-Output:

PUINT	pui_CounterValue	16-bit counter value
-------	------------------	----------------------

Task:

Latching of the 16-bit counter (*b_SelectedCounter*) from the selected module (*b_ModulNbr*) into the first latch register and return of the latched value.

Calling convention:

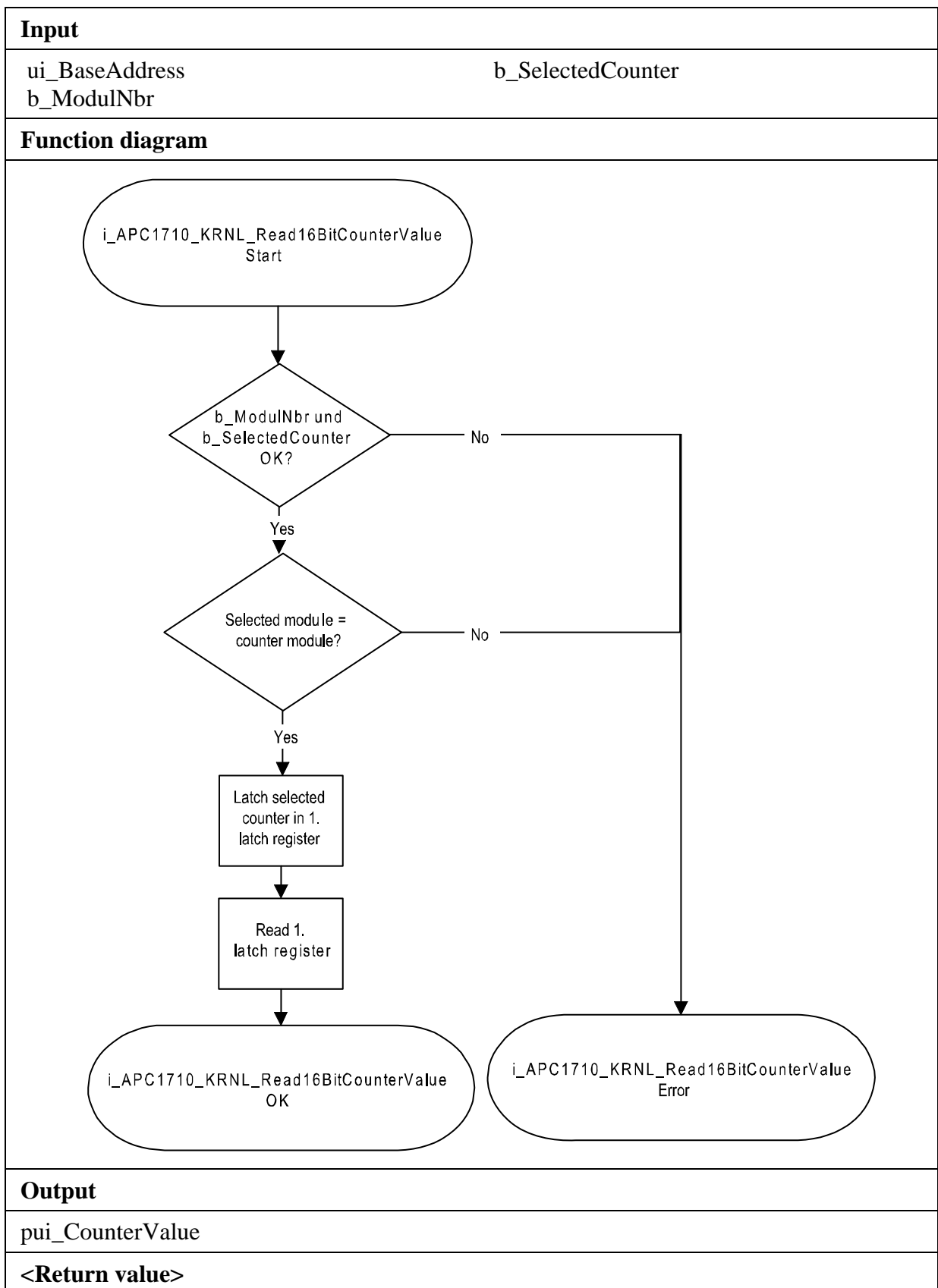
ANSI C:

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

```
i_ReturnValue = i_APCI1710_KRNL_Read16BitCounterValue
                (ui_BaseAddress,
                 0,
                 0,
                 &ui_CounterValue);
```

Return value:

0: No error
-1: Selected module number is wrong.
-2: The module is no counter module.
-3: Selected 16-bit counter is wrong.



3) i_APCI1710_KRNL_Read32BitCounterValue (...)

Syntax:

```
<Return Wert> = i_APCI1710_KRNL_Read32BitCounterValue
                    (UINT      ui_BaseAddress,
                     BYTE      b_ModulNbr,
                     PULONG    pul_CounterValue)
```

Parameter:

-Input:

UINT	ui_BaseAddress	Base address of the xPCI-1710. See "i_APCI1710_GetHardwareInformation"
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

PULONG	pul_CounterValue	32-bit counter value
--------	------------------	----------------------

Task:

Latching of the 32-bit counter (*b_SelectedCounter*) from the selected module (*b_ModulNbr*) into the first latch register and return of the latched value.

Calling convention:

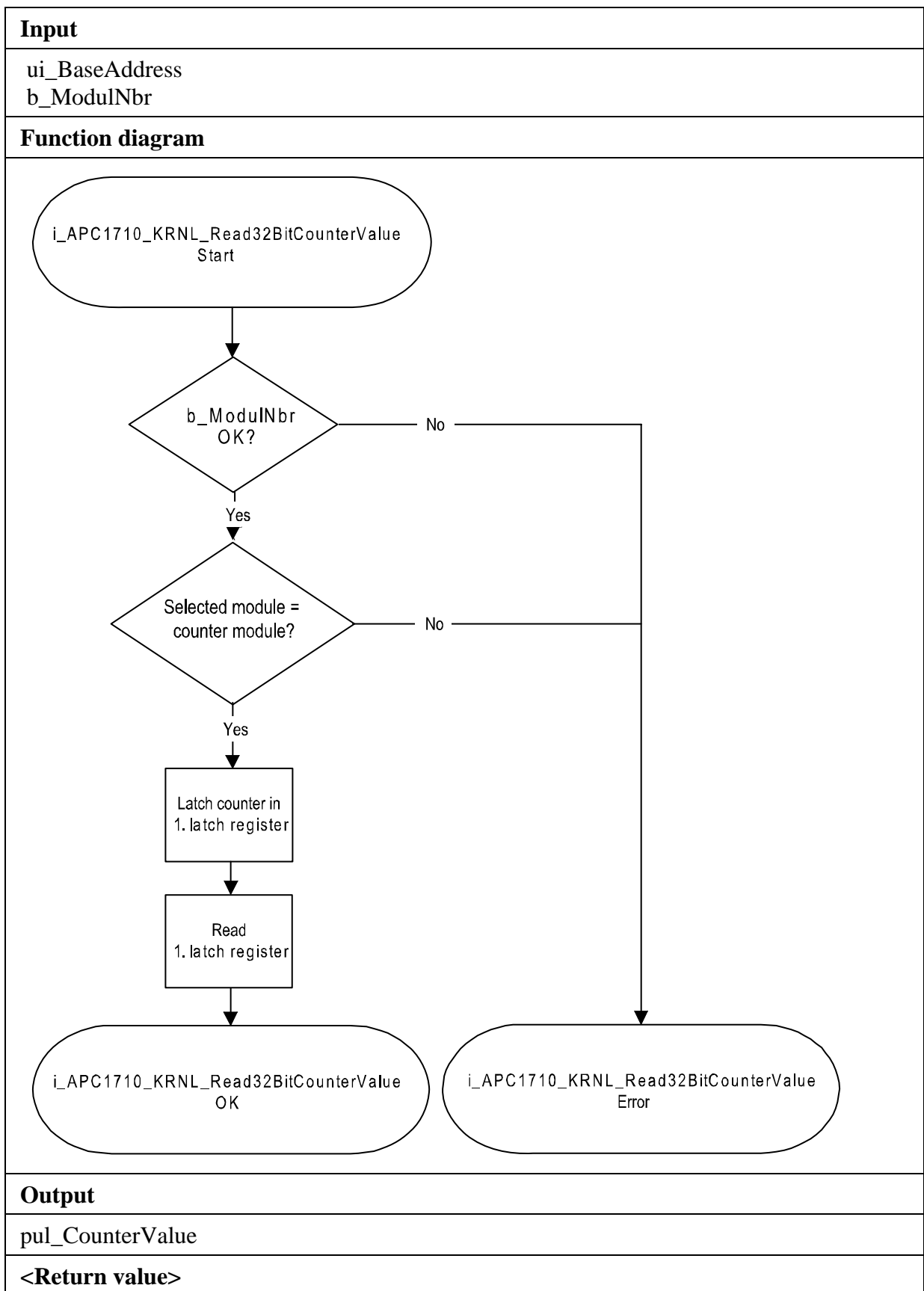
ANSI C:

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

```
i_ReturnValue = i_APCI1710_KRNL_Read32BitCounterValue
                (ui_BaseAddress,
                 0,
                 &ul_CounterValue);
```

Return value:

0: No error
-1: Selected module number is wrong.
-2: The module is no counter module.



4) `i_APCI1710_KRNL_Write16BitCounterValue (...)`

Syntax:

```
< Return Wert > = i_APCI1710_KRNL_Write16BitCounterValue
                        (UINT   ui_BaseAddress,
                        BYTE    b_ModulNbr,
                        BYTE    b_SelectedCounter,
                        UINT    ui_WriteValue)
```

Parameter:

-Input:

UINT	ui_BaseAddress	Base address of the xPCI-1710. See "i_APCI1710_GetHardwareInformation"
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_SelectedCounter	Selected 16-bit counter (0 or 1)
UINT	ui_WriteValue	16-bit write value

-Output:

There is no output.

Task:

Writes the 16-bit value (*ui_WriteValue*) into the 16-bit counter (*b_SelectedCounter*) of the selected module (*b_ModulNbr*)

Calling convention:

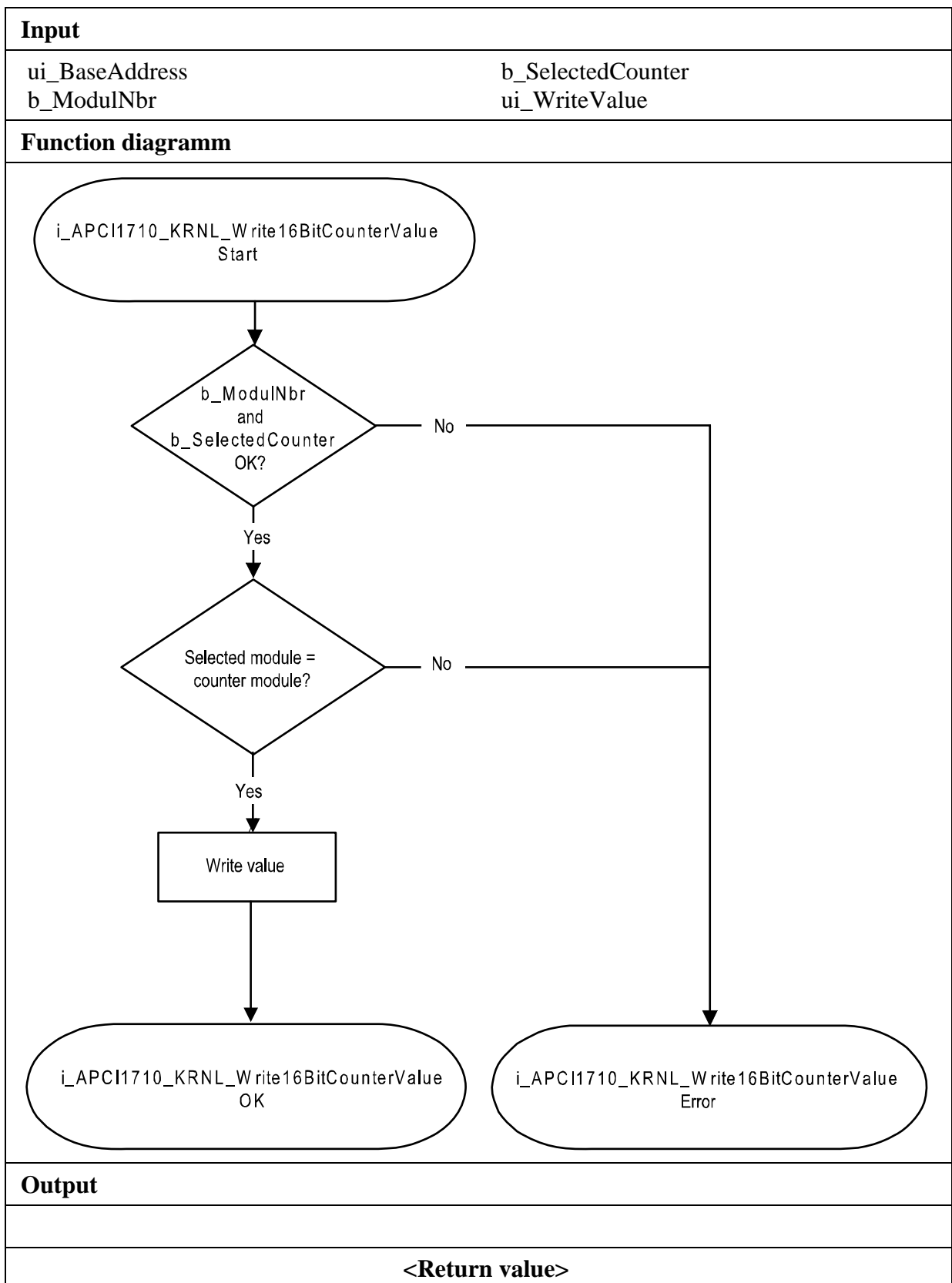
ANSI C:

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

```
i_ReturnValue = i_APCI1710_KRNL_Write16BitCounterValue
                (ui_BaseAddress,
                0,
                0,
                2000);
```

Return value:

0: No error
-1: Selected module number is wrong.
-2: The module is no counter module.
-3: Selected 16-bit counter is wrong.



5) i_APCI1710_KRNL_Write32BitCounterValue (...)

Syntax:

```
< Return Wert > = i_APCI1710_KRNL_Write16BitCounterValue  
                    (UINT   ui_BaseAddress,  
                     BYTE   b_ModulNbr,  
                     ULONG  ul_WriteValue)
```

Parameter:

-Input:

UINT	ui_BaseAddress	Base address of the xPCI-1710. See "i_APCI1710_GetHardwareInformation"
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
ULONG	ul_WriteValue	16-bit write value

-Output:

There is no output.

Task:

Writes the 32-bit value (*ui_WriteValue*) into the 32-bit counter of the selected module (*b_ModulNbr*)

Calling convention:

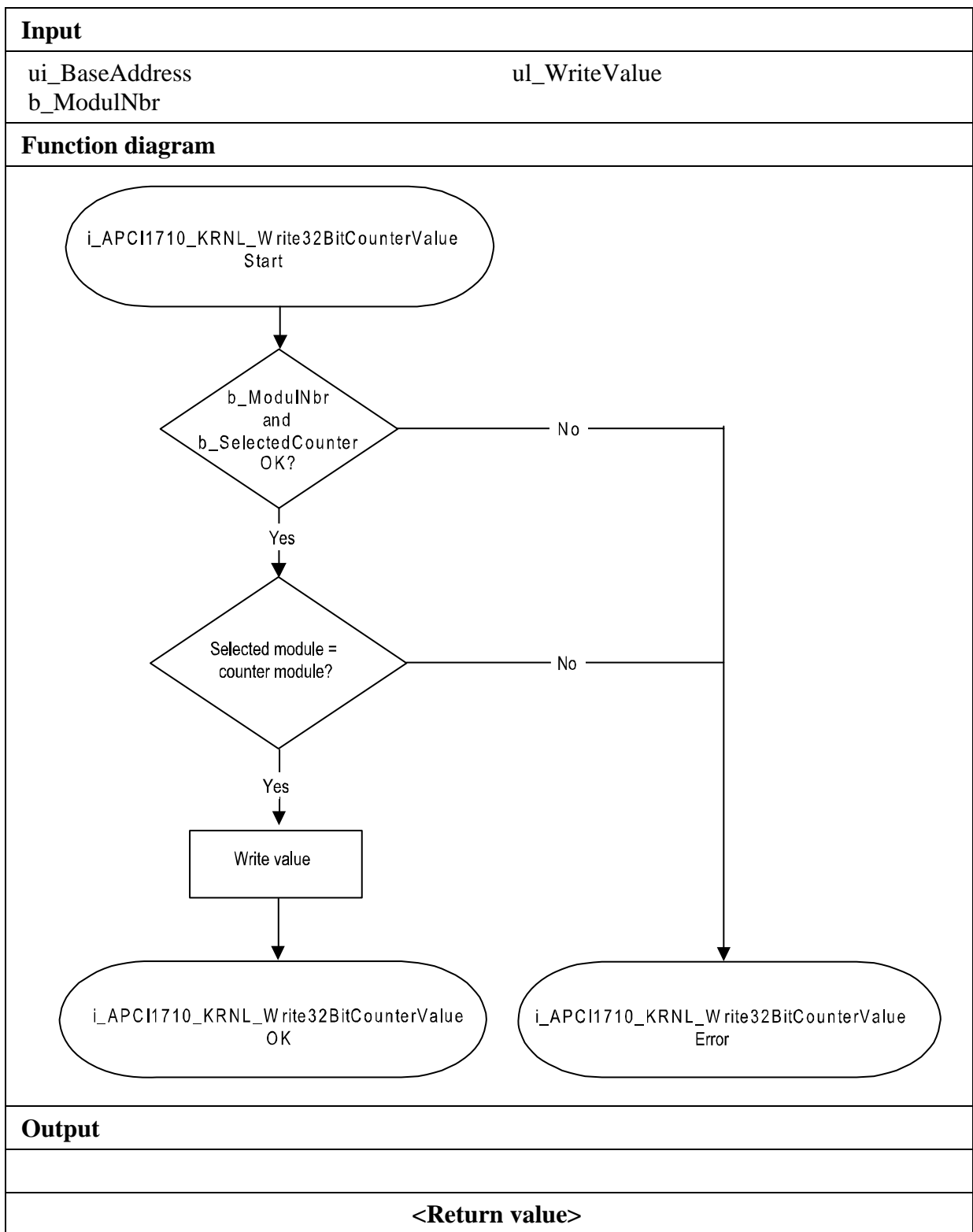
ANSI C:

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

```
i_ReturnValue = i_APCI1710_KRNL_Write32BitCounterValue  
                (ui_BaseAddress,  
                 0,  
                 200000);
```

Return value:

0: No error
-1: Selected module number is wrong.
-2: The module is no counter module.



6) i_APCI1710_KRNL_GetInterruptUDLatchedStatus (...)

Syntax:

```
<Return Wert> = i_APCI1710_KRNL_GetInterruptUDLatchedStatus
                (UINT      ui_BaseAddress,
                 BYTE      b_ModulNbr,
                 PBYTE     pb_UDStatus)
```

Parameter:

-Input:

UINT	ui_BaseAddress	Base address of the xPCI-1710. See "i_APCI1710_GetHardwareInformation"
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

PBYTE	pb_UDStatus	0: Counter operation downwards in the selected mode. 1: Counter operation upwards in the selected mode. 2: There is no index interrupt See function "i_APCI1710_InitCounter"
-------	-------------	---

Task:

Returns the latched status of the counter operation, after an index interrupt has occurred.

Calling convention:

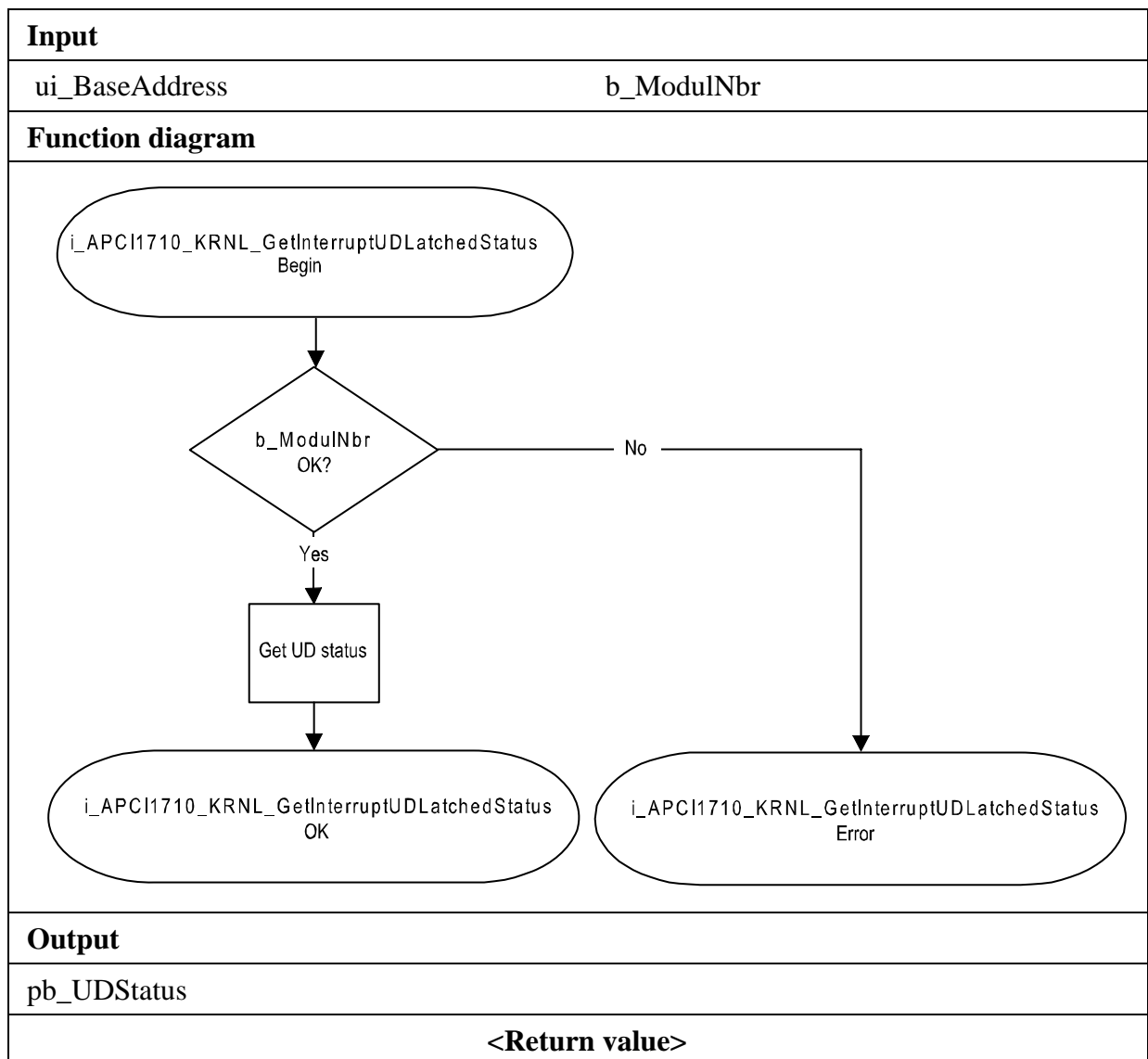
ANSI C:

```
int          i_ReturnValue;
unsigned int ui_BaseAddress;
unsigned char b_UDStatus;
```

```
i_ReturnValue = i_APCI1710_GetInterruptUDLatchedStatus
                (ui_BaseAddress,
                 0,
                 &b_UDStatus);
```

Return value:

0: No error
-1: Selected module number is wrong.
-2: The module is no counter module.



7) i_APCI1710_KRNL_ReadFrequencyMeasurement (...)

Syntax:

```
<Return Wert> = i_APCI1710_KRNL_ReadFrequencyMeasurement
                (UINT    ui_BaseAddress,
                 BYTE    b_ModulNbr,
                 PBYTE   pb_UDStatus,
                 PBYTE   pb_Status,
                 PULONG  pul_ReadValue)
```

Parameter:

- Input

UINT	ui_BaseAddress	Base address of the xPCI-1710. See "i_APCI1710_GetHardwareInformation"
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

- Output

PBYTE	pb_Status	Returns the status of the frequency measurement 0: Counting operation not started. 1: Counting operation started. 2: Counting operation stopped.
PBYTE	pb_UDStatus	Status of the upwards/downwards counter. See table3-12
PULONG	pul_ReadValue	Returns the number of increments within the time base.

Task:

Returns the status (*pb_Status*) and the number of increments in the set time. See function "i_APCI1710_InitFrequencyMeasurement"

Calling convention:

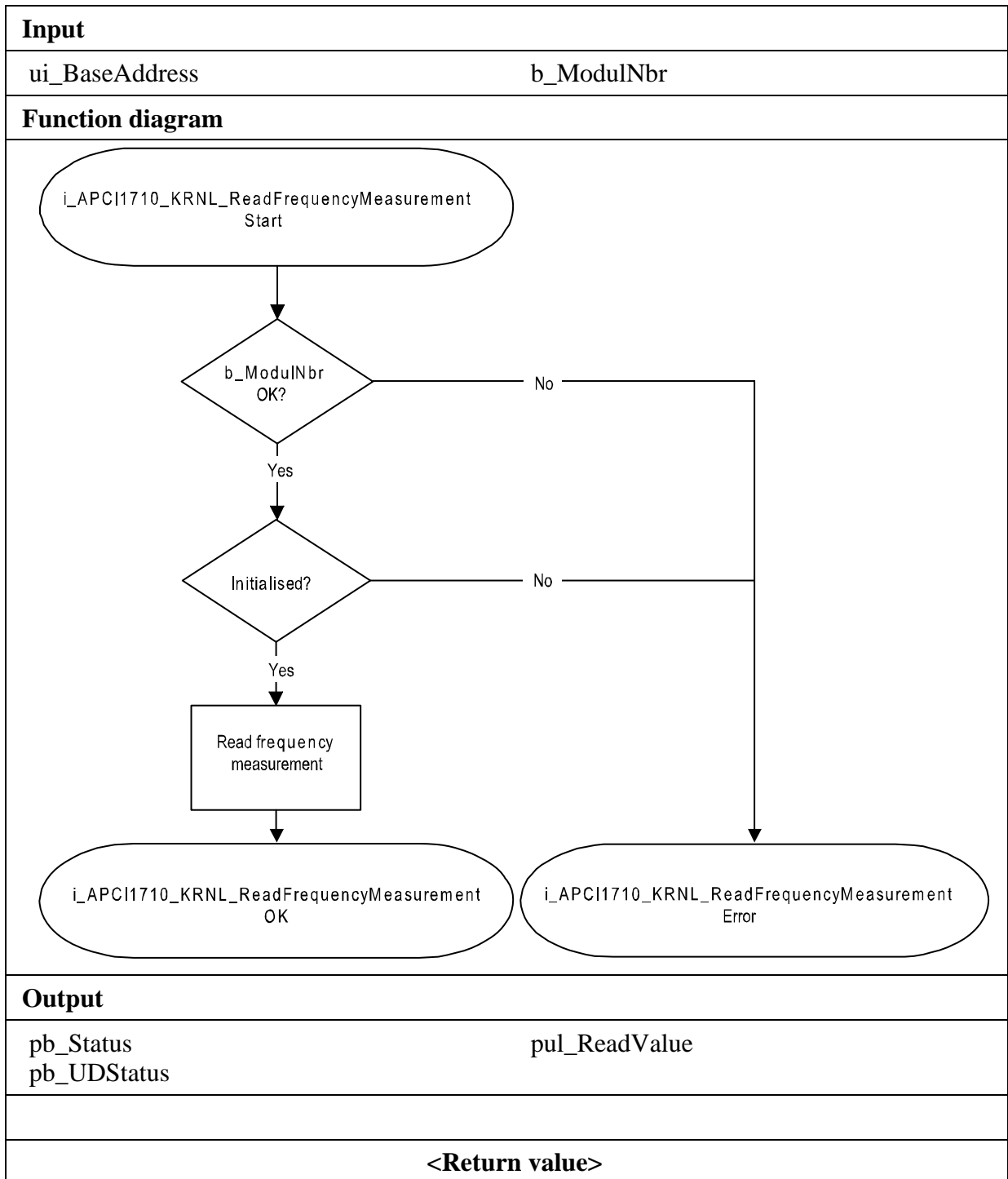
ANSI C:

```
int            i_ReturnValue;
unsigned int   ui_BaseAddress;
unsigned char  b_Status;
unsigned char  b_UDStatus;
unsigned long  ul_ReadValue;
```

```
i_ReturnValue = i_APCI1710_KRNL_ReadFrequencyMeasurement
                (ui_BaseAddress,
                 0,
                 &b_Status,
                 &b_UDStatus,
                 &ul_ReadValue);
```

Return value:

0: No error
-1: Selected module number is wrong
-2: The module is not counter module.



8) i_APCI1710_KRNL_SetDigitalChlOn (...)

Syntax:

```
<Return Wert> = i_APCI1710_KRNL_SetDigitalChlOn  
                (UINT    ui_BaseAddress,  
                 BYTE    b_ModulNbr)
```

Parameter:

-Input:

UINT	ui_BaseAddress	Base address of the xPCI-1710. See "i_APCI1710_GetHardwareInformation"
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

There is no output.

Task:

Sets the digital output on H. Setting an output, means setting an output on „High“.

Calling convention:

ANSI C:

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

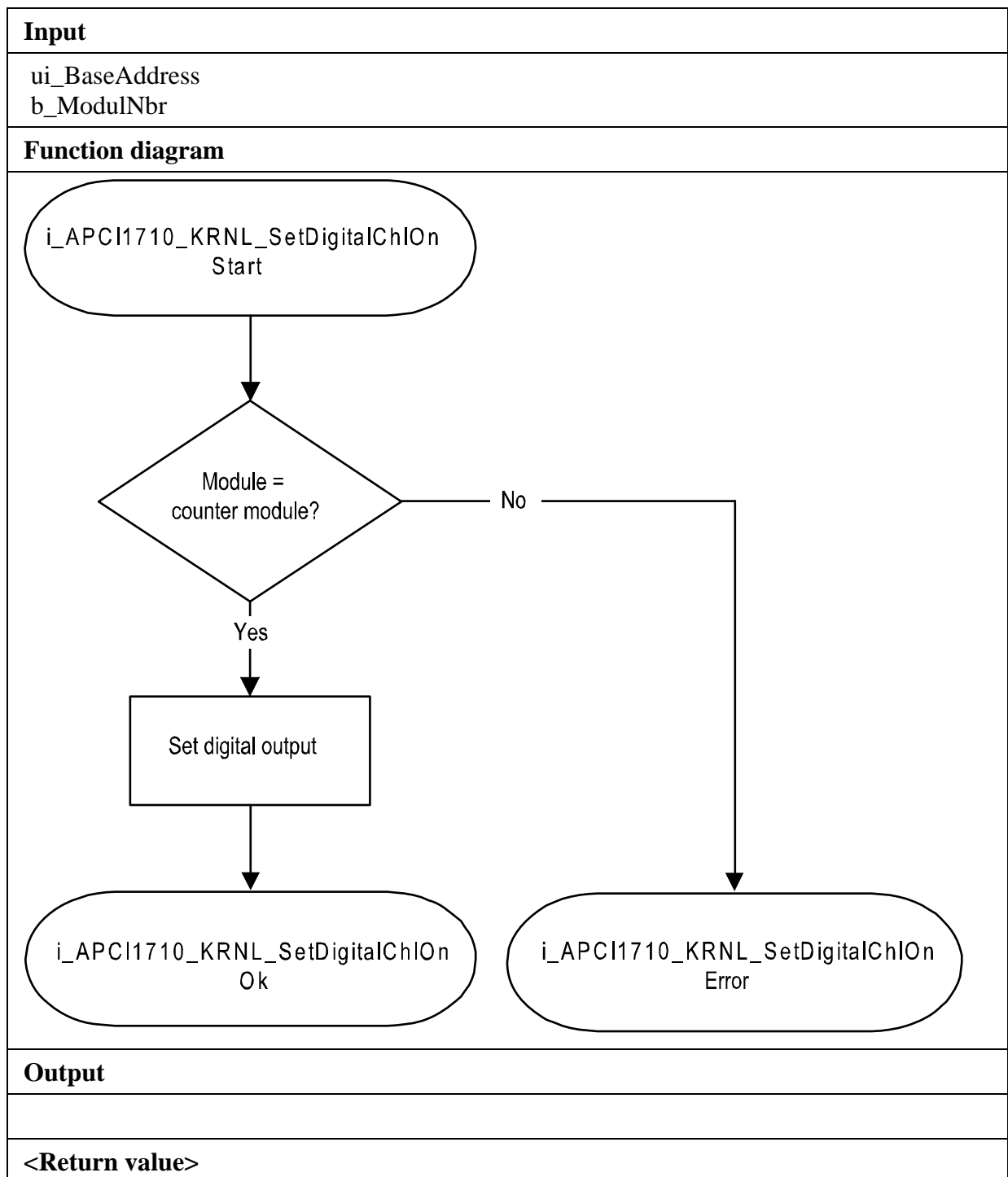
```
i_ReturnValue = i_APCI1710_KRNL_SetDigitalChlOn  
                (ui_BaseAddress,  
                 0);
```

Return value:

0: No error

-1: Selected module number is wrong.

-2: The module is no counter module.



9) i_APCI1710_KRNL_SetDigitalChlOff (...)

Syntax:

```
<Return Wert> = i_APCI1710_KRNL_SetDigitalChlOff  
                (UINT  ui_BaseAddress,  
                 BYTE  b_ModulNbr)
```

Parameter:

-Input:

UINT	ui_BaseAddress	Base address of the xPCI-1710. See "i_APCI1710_GetHardwareInformation"
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

-Output:

There is no output.

Task:

Resets the digital output H. Resetting an output means setting an output on "Low".

Calling convention:

ANSI C:

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

```
i_ReturnValue = i_APCI1710_KRNL_SetDigitalChlOff  
                (ui_BaseAddress,  
                 0);
```

Return value:

0: No error

-1: Selected module number is wrong.

-2: The module is no counter module.

