



DIN EN ISO 9001:2000  
certified



ADDI-DATA GmbH  
Dieselstraße 3  
D-77833 OTTERSWEIER



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

## Function description

**ADDICOUNT APCI-/CPCI-1710**

**Counter/Timer**

3<sup>rd</sup> edition 03/2005

## 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.

<b>1</b>	<b>DEFINITION OF APPLICATION .....</b>	<b>6</b>
1.1	Intended use .....	6
1.2	Usage restrictions.....	6
1.3	Technical description .....	6
1.4	Function description .....	7
1.5	Used abbreviations .....	7
<b>2</b>	<b>COUNTER/TIMER.....</b>	<b>8</b>
2.1	Function description .....	8
2.1.1	Block diagram of the counter/timer .....	9
2.1.2	Typical applications .....	10
2.2	Used signals .....	10
2.3	Pin assignment of the PWM.....	11
2.4	Connection example .....	12
2.5	I/O mapping .....	13
2.6	Description of the I/O functions.....	14
2.6.1	Function description .....	14
	Definition of the modes .....	14
	Mode0: Interrupt at the end of the counting process.....	14
	Mode1: Monoflop, retriggerable through hardware .....	14
	Mode2: Pulse generator .....	14
	Mode3: Square-wave generator .....	14
	Mode4: Strobe, triggered through software .....	15
	Mode5: Strobe, triggered through hardware (retriggerable).....	15
2.6.2	TIMER0-Register (Base + 0).....	15
2.6.3	TIMER1-Register (Base + 4).....	15
2.6.4	TIMER2-Register (Base + 8).....	15
2.6.5	Control-Register (Base + 12) .....	15
2.6.6	TIMER 0, 1, 2 Control-Register (Base + 16, 20, 24) .....	16
2.6.7	SET TIMER0 Register (Base + 32) .....	16
2.6.8	SET TIMER1 Register (Base + 36) .....	16
2.6.9	SET TIMER2 Register (Base + 40) .....	17
2.6.10	SET TIMER 0,1, 2 Softgate Register( Base + 44, 48, 52) .....	17
2.6.11	Versions-REGISTER (Base + 60) .....	17
2.7	Working with the „counter/timer“ function.....	17
<b>3</b>	<b>STANDARD SOFTWARE.....</b>	<b>18</b>
3.1	Introduction .....	18
3.2	Interrupt mask .....	19
3.3	Initialisation.....	20

1) i_APCI1710_InitTimer (...)	20
2) i_APCI1710_EnableTimer (...)	24
3) i_APCI1710_DisableTimer (...)	26
3.3.2 Read the timer	28
4) i_APCI1710_ReadTimerValue (...)	28
5) i_APCI1710_ReadAllTimerValue (...)	30
6) i_APCI1710_GetTimerOutputLevel (...)	32
7) i_APCI1710_GetTimerProgressStatus (...)	34
3.3.3 Writing on the timer	36
8) i_APCI1710_WriteTimerValue (...)	36
<b>3.4 Functions in the kernel mode</b>	<b>38</b>
3.4.1 Reading the timer	38
1) i_APCI1710_KRNL_ReadTimerValue (...)	38
2) i_APCI1710_KRNL_ReadAllTimerValue (...)	40
3.4.2 Writing on the timer	42
3) i_APCI1710_KRNL_WriteTimerValue (...)	42

## Figures

Fig. 2-1: Block diagram: Counter/Timer	9
Fig. 2-2: Pin assignment of the front connector	11
Fig. 2-3: Connection example	12

## Tables

Table 1-1: Delivered manuals	7
Table 2-1: Used signals	10
Table 2-2: I/O mapping of the "counter/timer" function	13
Table 3-1: Define value	18
Table 3-2: Interrupt mask of the function timer/counter	19
Table 3-3: Timer mode	21
Table 3-4: Selection of the input clock	21

# 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	ttl_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
- connection examples
- 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 COUNTER/TIMER

### 2.1 Function description

The function "counter/timer" is a counter and timer is equivalent to the Intel component 82C54.

This function stands out for its wide range of applications, its high precision and reliability in a harsh industrial environment.

**Properties:**

- The optical isolation is entirely completed by optical couplers for the input and output channels to avoid an earth circuit.
- 3 x 32-bit counters/timers (binary counting only)
- Signals up to 5 MHz can be processed
- 6 programmable modes
- Status readback and latch command
- Input and output channels can be inverted by software
- Hardware and software gates possible, rereadable
- Simple interface: No multiple assignment of the addresses
- Interrupt enabled with an individual release bit per counter/timer and interrupt status register
- Internal clock available as clock (PCI/4) or on-board 10 MHz quartz oscillator as clock, selectable per software

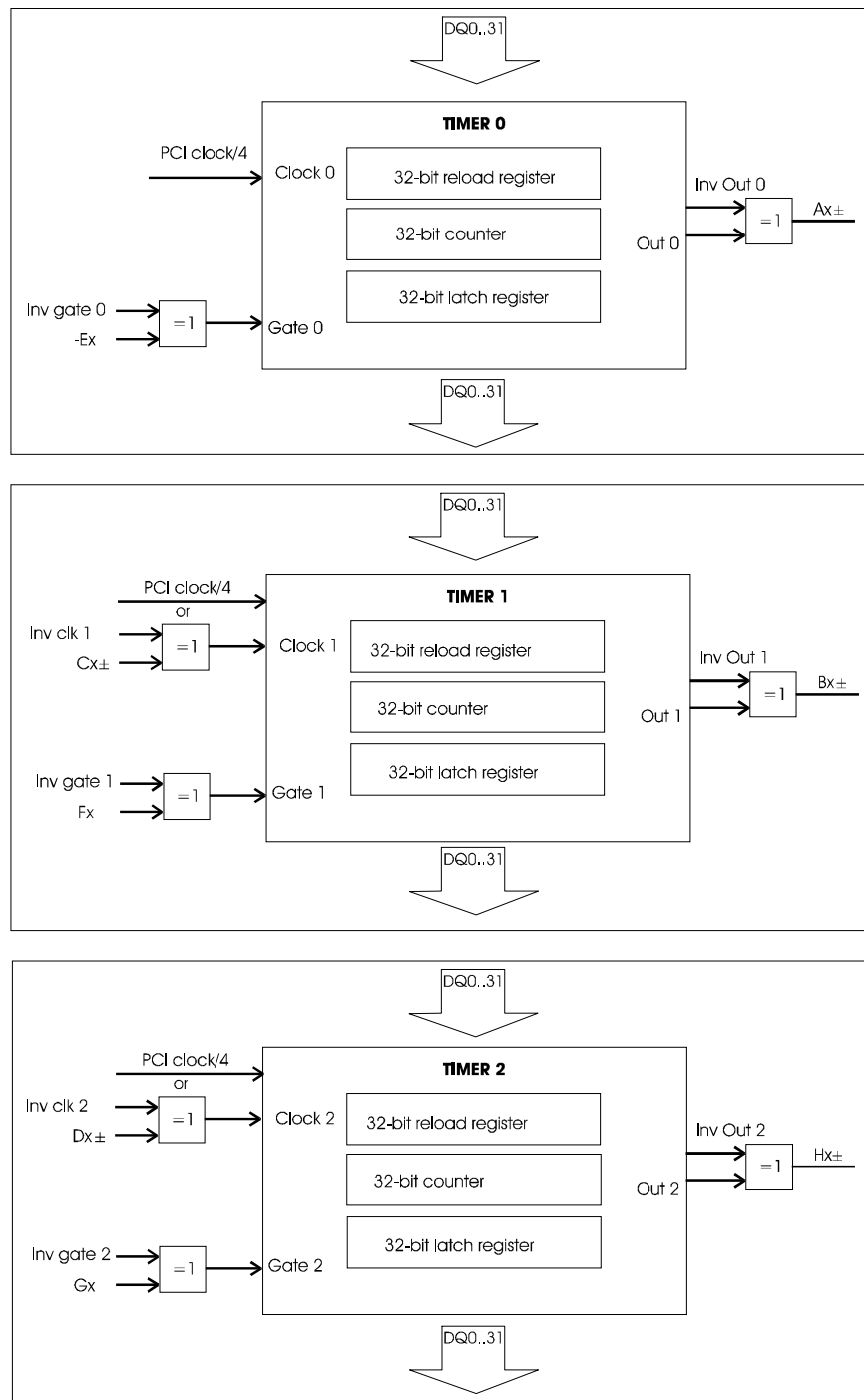


### 2.1.1 Block diagram of the counter/timer

The interface supports:

- 3 32-bit counter/timer independent from each other that are read or written over the data bus.
- A function and control logic

**Fig. 2-1: Block diagram: Counter/Timer**



## 2.1.2 Typical applications

- Digital "one-shot"
- Event counter
- Frequency generator
- Complex signal generator

## 2.2 Used signals

The function "counter/timer" occupies **5 inputs** (channels C to G) and **3 outputs** (channels A, B, H) of the respecting function module of the **APCI-/CPCI-1710**.

**Table 2-1: Used signals**

SIGNALS	AT THE CONNECTOR	POLARITY	FUNCTION
OUT 1	Ax +/-	Diff. / TTL	Output of the 1 <sup>st</sup> counter/timer
OUT 2	Bx +/-	Diff. / TTL	Output of the 2 <sup>nd</sup> counter/timer
OUT 3	Hx	24 V /Opt. 5 V	Output of the 3 <sup>rd</sup> counter/timer
GATE 1	Ex	24 V /Opt. 5V	Gate input of the 1 <sup>st</sup> counter/timer
GATE 2	Fx	24V /Opt.5 V	Gate input of the 2 <sup>nd</sup> counter/timer
GATE 3	Gx	24V /Opt.5 V	Gate input of the 3 <sup>rd</sup> counter/timer
CLK 1	-	-	Occupied by the internal clock = PCI clock divided by 4
CLK 2	Cx +/-	Diff./TTL/Opt. 24V	Clock/counter input of the 2 <sup>nd</sup> counter/timer
CLK 3	Dx +/-	Diff./TTL/Opt. 24V	Clock/counter input of the 3 <sup>rd</sup> counter/timer

**x:** Number of the function module.

## 2.3 Pin assignment of the counter/timer

**i**

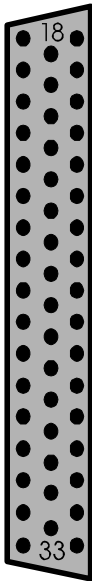
### IMPORTANT!

The function modules are defined differently in the hardware and software descriptions.

For the pin assignment (hardware) the modules are numbered from 1 to 4. For the SET1710 program or the software functions (Software) the module numbering **BEGINS** with 0.

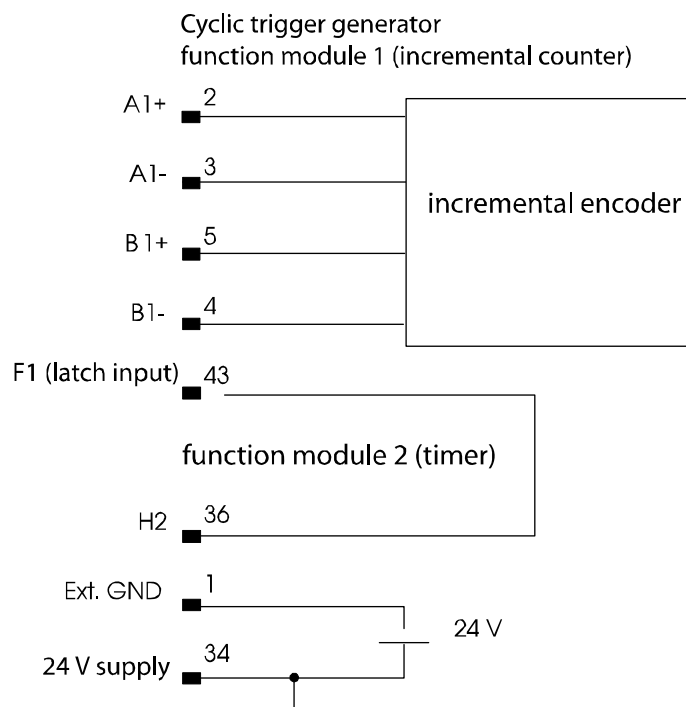
The figure below is a connection example. The function “counter/timer” is implemented on all function modules.

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

Pin		Pin		Pin		Pin				
Output 3	34	+Uref/24 V ext.	function module 3	18	A3 +		1	Ext. GND	1	function module 1
	35	H1		19	A3 -		2	A1 +	2	
	36	H2		20	B3 +		3	A1 -	3	
	37	H3		21	B3 -		4	B1 +	4	
	38	H4		22	C3 +		5	B1 -	5	
Gate 1	39	E1	function module 4	23	C3 -		6	C1 +	6	function module 2
	40	E2		24	D3 +		7	C1 -	7	
	41	E3		25	D3 -		8	D1 +	8	
	42	E4		26	A4 +		9	D1 -	9	
Gate 2	43	F1		27	A4 -		10	A2 +	10	
	44	F2		28	B4 +		11	A2 -	11	
	45	F3		29	B4 -		12	B2 +	12	
	46	F4		30	C4 +		13	B2 -	13	
Gate 3	47	G1		31	C4 -		14	C2 +	14	
	48	G2		32	D4 +		15	C2 -	15	
	49	G3		33	D4 -		16	D2 +	16	
	50	G4					17	D2 -	17	

## 2.4 Connection example

**Fig. 2-3: Connection example**



## 2.5 I/O mapping

Table 2-2: I/O mapping of the “counter/timer” function

			D31...D24	D23...D16	D15.....D8	D7.....D0
BYTES	Rd	Wr	HIGHBYTE	MIDHIGHBYTE	MIDLOWBYTE	LOWBYTE
BASE $x$ + 0	☑	☑	<b>TIMER0</b>	<b>TIMER0</b>	<b>TIMER0</b>	<b>TIMER0</b>
BASE $x$ + 4	☑	☑	<b>TIMER1</b>	<b>TIMER1</b>	<b>TIMER1</b>	<b>TIMER1</b>
BASE $x$ + 8	☑	☑	<b>TIMER2</b>	<b>TIMER2</b>	<b>TIMER2</b>	<b>TIMER2</b>
BASE $x$ + 12		☑	<b>GLOBAL CONTROL REGISTER</b>			
BASE $x$ + 16	☑	☑	0x00h	0x00h	<b>TIMER0</b> CONTROL REG	<b>TIMER0</b> CONTROL REG
BASE $x$ + 20	☑	☑	0x00h	0x00h	<b>TIMER1</b> CONTROL REG	<b>TIMER1</b> CONTROL REG
BASE $x$ + 24	☑	☑	0x00h	0x00h	<b>TIMER2</b> CONTROL REG	<b>TIMER2</b> CONTROL REG
BASE $x$ + 28			-	-	-	-
BASE $x$ + 32		☑	-	-	-	SET <b>TIMER0</b> REGISTER
BASE $x$ + 36		☑	-	-	-	SET <b>TIMER1</b> REGISTER
BASE $x$ + 40		☑	-	-	-	SET <b>TIMER2</b> REGISTER
BASE $x$ + 44		☑	-	-	-	SET <b>TIMER0</b> SOFTGATE
BASE $x$ + 48		☑	-	-	-	SET <b>TIMER1</b> SOFTGATE
BASE $x$ + 52		☑	-	-	-	SET <b>TIMER2</b> SOFTGATE
BASE $x$ + 60			FUNKNBR2	FUNKNBR1	REVBYTE2	REVBYTE1

-: No function ;  $x$ : Number of the function module.

The function “counter/timer” occupies 15 DWORDS in the I/O range of the function module  $x$ .

The accesses are always read or written in 32-bit.

## 2.6 Description of the I/O functions

### 2.6.1 Function description

The function counter/timer is similar to the component 82C54 from INTEL.

**3 x 32-bit counters/timers** are available. Each of these counters/timers can be programmed in one of **6 possible modes** (mode0 to mode5).

The counters/timers in 32-bit can be loaded and read with a new value at any time. For reading the value must first be "latched".

#### Definition of the modes

CLK PULSE: Falling edge after rising edge of the counter clock input.

TRIGGER: Rising edge of the counter GATE input.

COUNTER LOADING: counter transfer from the counting register to the counter component.

#### Mode0: Interrupt at the end of the counting process

The mode0 is considered especially for event counting. After the initialisation is the output "Low". The output keeps this position until the counter reaches 0.

Then the output is set to "High" until the new counting cycle or until writing of the new counting value.

#### Mode1: Monoflop, retriggerable through hardware

This mode is the same as mode0, except for the function of the GATE input.

The GATE input is not used in order to enable or disable the timer. It triggers the timer.

#### Mode2: Pulse generator

This mode operates as one by ul\_ReloadValue dividing counters. It is used for the generation of a real-time clock interrupt.

After the initialisation the output is set to "High". The start value is decremented on to the value „0“; the output goes „Low“ during a clock signal and "High" again. The counter reloads the start value (ul\_RelaodValue) and the cycle repeats. An interrupt can be generated after each end of a cycle.

Time calculation =  $(ul\_RelaodValue + 2) \times \text{input clock}$

#### Mode3: Square-wave generator

Mode3 generates the baudrate. It is similar to mode2, excepting the output cycle. When starting the counter, the output is set on „High“. The start value is decremented on to 0, the output is set on "Low". The counter is reloaded with the

start value and is decremented again on to 0. The output again is on “High”. The cycle is repeated automatically.

Time calculation = (ul\_RelaodValue + 2) x input clock

#### **Mode4: Strobe, triggered through software**

After the initialisation the output is to "High". When the start value is over, the output goes “Low” during a clock pulse and is set again to “High”. The counting sequence is triggered when a new value is written in.

When a new value is written in during a counting cycle, this value will be loaded at the next clock pulse.

#### **Mode5: Strobe, triggered through hardware (retriggerable)**

This mode is the same as mode4, except for the function of the GATE input.

The GATE input is not used in order to enable or disable the timber.  
It triggers the timer.

### **2.6.2 TIMER0-Register (Base + 0)**

32-bit register, in which the reload value for counter/timer 0 is written.  
When reading this address, the currently latched value of the counter/timer 0 is read.

### **2.6.3 TIMER1-Register (Base + 4)**

32-bit register, in which the reload value for counter/timer 1 is written. When reading the address, the currently latched value of the counter/timer 1 is read.

### **2.6.4 TIMER2-Register (Base + 8)**

32-bit register, in which the reload value for counter/timer 2 is written. When reading this address the currently latched value of counter/timer 2 is read.

### **2.6.5 Control-Register (Base + 12)**

32-bit register which is responsible for the 3 counters.

DQ0: = 0

DQ1: Selection of **Timer 0**, if "1" then counter stand and state are latched.

DQ2: Selection of **Timer 1**, if "1" then counter stand and state are latched.

DQ3: Selection of **Timer 2**, if "1" then counter stand and state are latched.

DQ4: Writing of a 0 → Latching of the selected timer condition.

DQ5: Writing of a 0 → Latching of the selected timer condition.

DQ6: = 1

DQ7 : = 1  
 DQ8..31: No function

With the "LATCH" function all three counters can be latched simultaneously by software. Hereto all three counters must be selected. DQ1= DQ2= DQ3 = DQ6 = DQ7 =1 , & DQ4 = 0 = Latching of the selected counters.

### 2.6.6 TIMER 0, 1, 2 Control-Register (Base + 16, 20, 24)

DQ0: Modebit 1 (only writing possible), selection of the counter mode:  
 0,1,2,3,4,5  
 DQ1: Modebit 2 (only writing possible)  
 DQ2: Modebit 3 (only writing possible)  
 DQ3: =0  
 DQ4: =0  
 DQ5: =0  
 DQ6: NULL\_COUNT (only reading possible), if "1" the timer is stopped, if "0", the timer runs out (the timer had been loaded).  
 DQ7: OUT (only reading possible), image of the output condition at the timer  
 DQ8: GATE (only reading possible), image of the gate condition on the timer.

Remaining bits on rereading on 0

### 2.6.7 SET TIMER0 Register (Base + 32)

DQ0: 1: inverts the external GATE0 signal, 0: No inversion (Reset)  
 DQ1: 1: inverts the internal CLK0 signal, 0: No inversion (Reset)  
 DQ2: 1: inverts the external OUT0 signal, 0: No inversion (Reset)  
 DQ3: 1: Interrupt release for TIMER0, 0: No interrupt (Reset)  
 DQ4DQ31..4: No function  
 SET TIMER1 Register (Base + 36)  
 DQ0: 1: inverts the external GATE1 signal, 0: No inversion (Reset)  
 DQ1: 1: inverts the external CLK1 signal, 0: No inversion (Reset)  
 DQ2: 1: inverts the external OUT1 signal, 0: No inversion (Reset)  
 DQ3: 1: Interrupt release for TIMER1, 0: No interrupt (Reset)  
 DQ4: No function  
 DQ5: 0: PCI-Bus clock used as time base.  
       1: Internal quartz /4 used as time base (10 MHz).  
 DQ6 → DQ31..4: No function

### 2.6.8 SET TIMER1 Register (Base + 36)

DQ0: 1: inverts the external GATE1 signal, 0: No inversion (Reset)  
 DQ1: 1: inverts the external CLK1 signal, 0: No inversion (Reset)  
 DQ2: 1: inverts the external OUT1 signal, 0: No inversion (Reset)  
 DQ3: 1: Interrupt release for TIMER1, 0: No interrupt (Reset)  
 DQ4: 1: external clock1 is used, 0 internal clock (pci/4)  
 DQ5: 0: PCI-Bus clock used as time base.  
       1: Internal quartz /4 used as time base (10 MHz).  
 DQ6 → DQ31..4: No function



### 2.6.9 SET TIMER2 Register (Base + 40)

DQ0: 1: inverts the external GATE2 signal, 0: No inversion (Reset)  
 DQ1: 1: inverts the external CLK2 signal, 0: No inversion (Reset)  
 DQ2: 1: inverts the external OUT2 signal, 0: No inversion (Reset)  
 DQ3: 1: Interrupt release for TIMER2, 0: No interrupt (Reset)  
 DQ4: 1: external Clock2 is used, 0 internal clock(pci/4)  
 DQ5: 0: PCI-Bus clock used as time base.  
       1: Internal quartz /4 used as time base (10 MHz).  
 DQ6 → DQ31..4: No function

### 2.6.10 SET TIMER 0,1, 2 Softgate Register( Base +44, 48, 52)

DQ0: 1 counting is allowed, depending on the external GATE signal  
       0 counting is locked (Reset).  
 DQ31..1: No function

### 2.6.11 Versions-REGISTER (Base +60)

The function and revision are recognised. (reading command, ASCII format)

**BASE + 60    "I"    "C"    "1"    "3"**

Meaning: Chronos Revision 1.3

## 2.7 Working with the „counter/timer“ function

1. Programming the counter/timer in the required mode.
2. Adapting the signals GATE or OUT to the required level.
3. Writing the reload value into the timer.
4. Counter/timer are ready for operation.
5. If you work with the interrupt, firstly the release bits for the interrupt must be set on "1".

## 3 STANDARD SOFTWARE

### 3.1 Introduction



#### IMPORTANT!

Note the following style conventions in the text:

Function:     *"i\_APCI1710\_SetBoardInformation"*

Variable     *ui\_Address*

**Table 3-1: Define value**

Define name	Decimal value	Hexadecimal value
DLL_COMPILER_C	1	1
DLL_COMPILER_VB	2	2
DLL_COMPILER_PASCAL	3	3
DLL_LABVIEW	4	4
APCI1710_PCI_BUS_CLOCK	0	0
APCI1710_FRONT_CONNECTOR_INPUT	1	1
APCI1710_30MHZ	30	1E
APCI1710_33MHZ	33	21
APCI1710_40MHZ	40	28
APCI1710_10MHZ	10	A

## 3.2 Interrupt mask

Each timer/counter can generate an interrupt. In order to get this interrupt, you shall activate the interrupt and the interrupt routine with the function "i\_APCI1710\_SetBoardIntRoutineX".

**Table 3-2: Interrupt mask of the function timer/counter**

<b>b_ModuleMask</b>	<b>ul_InterruptMask</b>	<b>Meaning</b>
0000 0001	0000 0000 0001 0000	Interrupt on Timer0 generated from module 0
0000 0001	0000 0000 0010 0000	Interrupt on Timer1 generated from module 0
0000 0001	0000 0000 0100 0000	Interrupt on Timer2 generated from module 0
0000 0010	0000 0000 0001 0000	Interrupt on Timer0 generated from module 1
0000 0010	0000 0000 0010 0000	Interrupt on Timer1 generated from module 1
0000 0010	0000 0000 0100 0000	Interrupt on Timer2 generated from module 1
0000 0100	0000 0000 0001 0000	Interrupt on Timer0 generated from module 2
0000 0100	0000 0000 0010 0000	Interrupt on Timer1 generated from module 2
0000 0100	0000 0000 0100 0000	Interrupt on Timer2 generated from module 2
0000 1000	0000 0000 0001 0000	Interrupt on Timer0 generated from module 3
0000 1000	0000 0000 0010 0000	Interrupt on Timer1 generated from module 3
0000 1000	0000 0000 0100 0000	Interrupt on Timer2 generated from module 3

### 3.3 Initialisation

#### 1) i\_APCI1710\_InitTimer (...)

##### Syntax:

```
<Return-Wert> = i_APCI1710_InitTimer
                                (BYTE    b_BoardHandle,
                                 BYTE    b_ModulNbr,
                                 BYTE    b_TimerNbr,
                                 BYTE    b_TimerMode,
                                 ULONG   ul_ReloadValue,
                                 BYTE    b_InputClockSelection,
                                 BYTE    b_InputClockLevel,
                                 BYTE    b_OutputLevel,
                                 BYTE    b_HardwareGateLevel)
```

##### Parameter:

##### -Input:

BYTE	b_BoardHandle	Handle of the board <b>APCI-/CPCI-1710</b>
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_TimerNbr	Number of the timer to be configured (0 to 2)
BYTE	b_TimerMode	Selection of the timer mode (0 to 5) 0: Interrupt at the end of the counting process. 1: Monoflop retriggerable by hardware 2: Pulse generator 3: Square wave generator 4: Strobe triggered by software 5: Strobe triggered by hardware See chapter 2.
ULONG	ul_ReloadValue	Start counter value or division factor.
BYTE	b_InputClockSelection	Selection of the input timer clock.
BYTE	b_InputClockLevel	Selection of the input clock level. 0: active at "Low" 1: active at "High" (input inverted)
BYTE	b_OutputLevel	Selection of the output clock level. 0: active at "Low" 1: active at "High" (output inverted)
BYTE	b_HardwareGateLevel	Selection of the hardware gate level 0: Active at "Low" (input inverted) 1: Active at "High" If the external gate is not used, set the parameter on "0".

##### -Output:

No output.

Table 3-3: Timer mode

Mode	Meaning	<i>u_ReloadValue</i>	Hardware gate
0	Interrupt at the end of the counting process	Start value	Hardware gate
1	Monoflop, retriggerable through hardware	Start value	Hardware trigger
2	Pulse generator	Division factor	Hardware gate
3	Square wave generator	Division factor	Hardware gate
4	Strobe, triggered through software	Start value	Hardware gate
5	Strobe, triggered through hardware (retriggerable)	Start value	Hardware trigger

Table 3-4: Selection of the input clock

<b>b_InputClockSelection</b>	<b>Description</b>
APCI1710_PCI_BUS_CLOCK	The PCI bus clock divided by 4 is used for the timer input clock. The PCI bus clock can be 30 MHz or 33 MHz. For the timer 0 only this selection and APCI1710_10MHZ are possible.
APCI1710_10MHZ	The 40 MHz clock divided by 4 is used for the timer input clock. For timer 0 only this selection and APCI1710_PCI_BUS_CLOCK are possible.
APCI1710_FRONT_CONNECTOR_INPUT	On the front connector an input clock can be supplied for timer 1 and 2. This clock source can overwrite the output clock of timer 0 or all other clock sources.

**Tak:**

Configuration of the timer (*b\_TimerNbr*) and of the operating mode (*b\_TimerMode*) of the selected module (*b\_ModulNbr*). This function must be called before calling another function, which accesses the timer.

**Calling convention:**

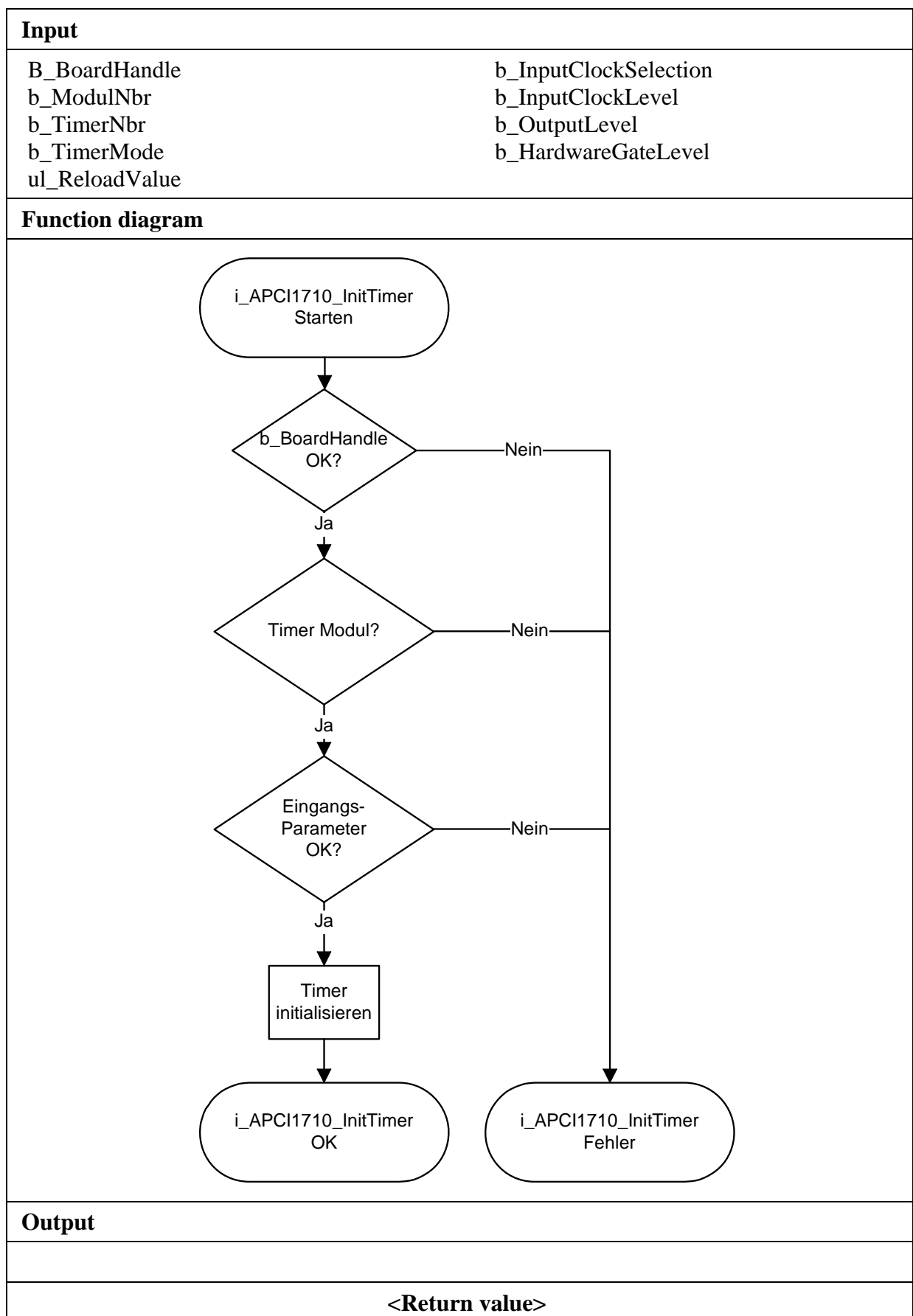
ANSI C :

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

```
i_ReturnValue = i_APCI1710_InitTimer  
                (b_BoardHandle,  
                 0,  
                 0,  
                 2,  
                 0xFF00,  
                 APCI1710_PCI_BUS_CLOCK,  
                 1,  
                 1,  
                 1);
```

**Return value:**

- 0: No error
- 1: Handle parameter of the board is wrong.
- 2: The selected module number is wrong.
- 3: The selected timer is wrong.
- 4: The selected module is no "Timer" module.
- 5: The selected operation mode is wrong.
- 6: The selected timer input clock is wrong.
- 7: The selected input clock level is wrong.
- 8: The selected output clock level is wrong.
- 9: The selected hardware gate level is wrong.



## 2) i\_APCI1710\_EnableTimer (...)

### Syntax:

```
<Return-Wert> = i_APCI1710_EnableTimer
                                (BYTE b_BoardHandle,
                                 BYTE b_ModulNbr,
                                 BYTE b_TimerNbr,
                                 BYTE b_InterruptEnable)
```

### Parameter:

#### -Input:

BYTE	b_BoardHandle	Handle of the <b>APCI-/CPCI-1710</b>
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_TimerNbr	Number of the timer to be released (0 to 2)
BYTE	b_InterruptEnable	Enables or disables the timer interrupt. APCI1710_ENABLE: Interrupt enabled APCI1710_DISABLE: Interrupt disabled

#### -Output:

There is no output.

### Task:

Enables the timer (*b\_TimerNbr*) of the selected module (*b\_ModulNbr*). Firstly call the function "i\_APCI1710\_InitTimer" before calling this function.

When the timer interrupt is released, the timer generates an interrupt, after the counting value has reached "0".

See function "i\_APCI1710\_SetBoardIntRoutineX".

### Calling convention:

ANSI C :

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

```
i_ReturnValue = i_APCI1710_EnableTimer
                (b_BoardHandle,
                 0,
                 0,
                 APCI1710_DISABLE);
```

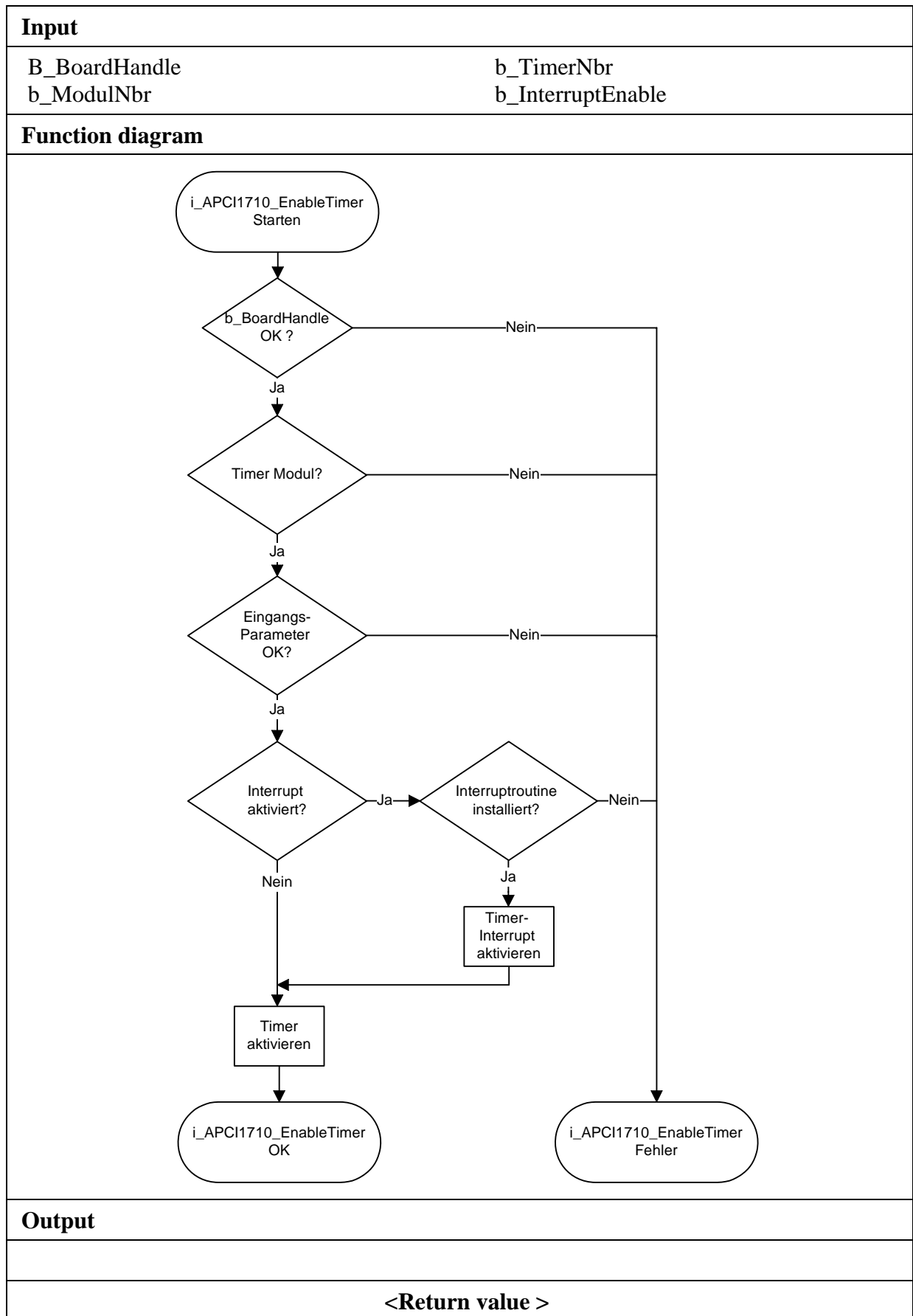
### Return value:

0: No error

- 1: Handle parameter of the board is wrong.
- 2: The selected module number is wrong.
- 3: The selected timer is wrong.
- 4: The selected module is no "Timer" module
- 5: Timer not initialised. See function "i\_APCI1710\_InitTimer"
- 6: Interrupt parameter is wrong.
- 7: Interrupt function not initialised.

See function "i\_APCI1710\_SetBoardIntRoutineX"





### 3) i\_APCI1710\_DisableTimer (...)

**Syntax:**

<Return-Wert> = i\_APCI1710\_DisableTimer  
(BYTE b\_BoardHandle,  
BYTE b\_ModulNbr,  
BYTE b\_TimerNbr)

**Parameter:****-Input:**

BYTE	b_BoardHandle	Handle of the <b>APCI-/CPCI-1710</b>
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_TimerNbr	Number of the timer to be locked (0 to 2)

**-Output:**

There is no output.

**Task:**

Disables the timer (*b\_TimerNbr*) of the selected module (*b\_ModulNbr*).

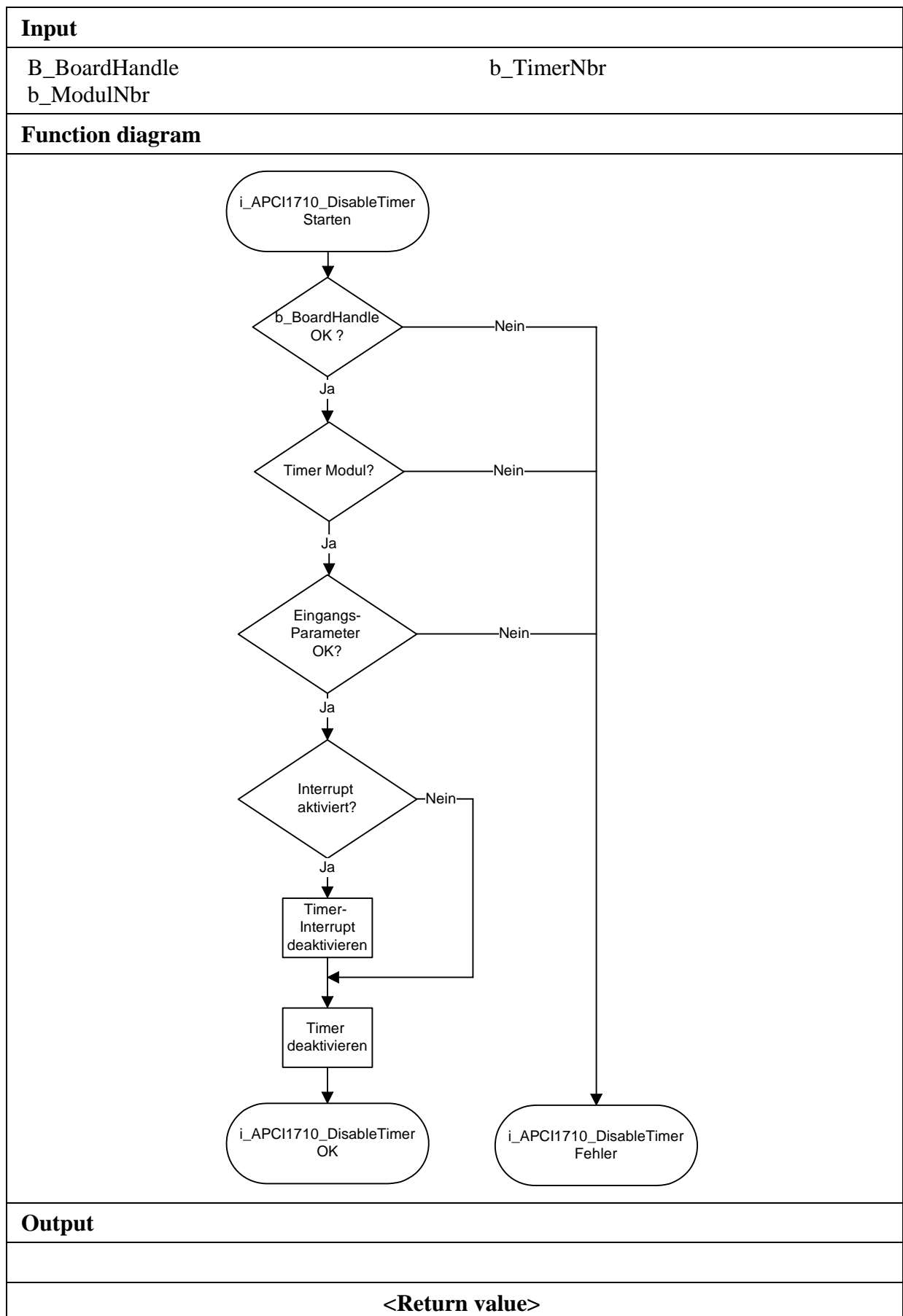
**Calling convention:**ANSI C:

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

```
i_ReturnValue = i_APCI1710_DisableTimer  
                (b_BoardHandle,  
                0,  
                0);
```

**Return value:**

0: No error  
-1: Handle parameter of the board is wrong.  
-2: The selected module number is wrong.  
-3: The selected timer is wrong.  
-4: The selected module is no "Timer" module.  
-5: Timer not initialised. See function "i\_APCI1710\_InitTimer"



### 3.3.2 Read the timer

#### 4) i\_APCI1710\_ReadTimerValue (...)

**Syntax:**

```
<Return-Wert> = i_APCI1710_ReadTimerValue
                                (BYTE      b_BoardHandle,
                                BYTE      b_ModulNbr,
                                BYTE      b_TimerNbr,
                                PULONG    pul_TimerValue)
```

**Parameter:**
**-Input:**

BYTE	b_BoardHandle	Handle of the <b>APCI-/CPCI-1710</b>
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_TimerNbr	Number of the timer to be read (0 to 2)

**-Output:**

PULONG	pul_TimerValue	Timer value
--------	----------------	-------------

**Task:**

Returns the value for the timer (*b\_TimerNbr*) of the selected module (*b\_ModulNbr*).

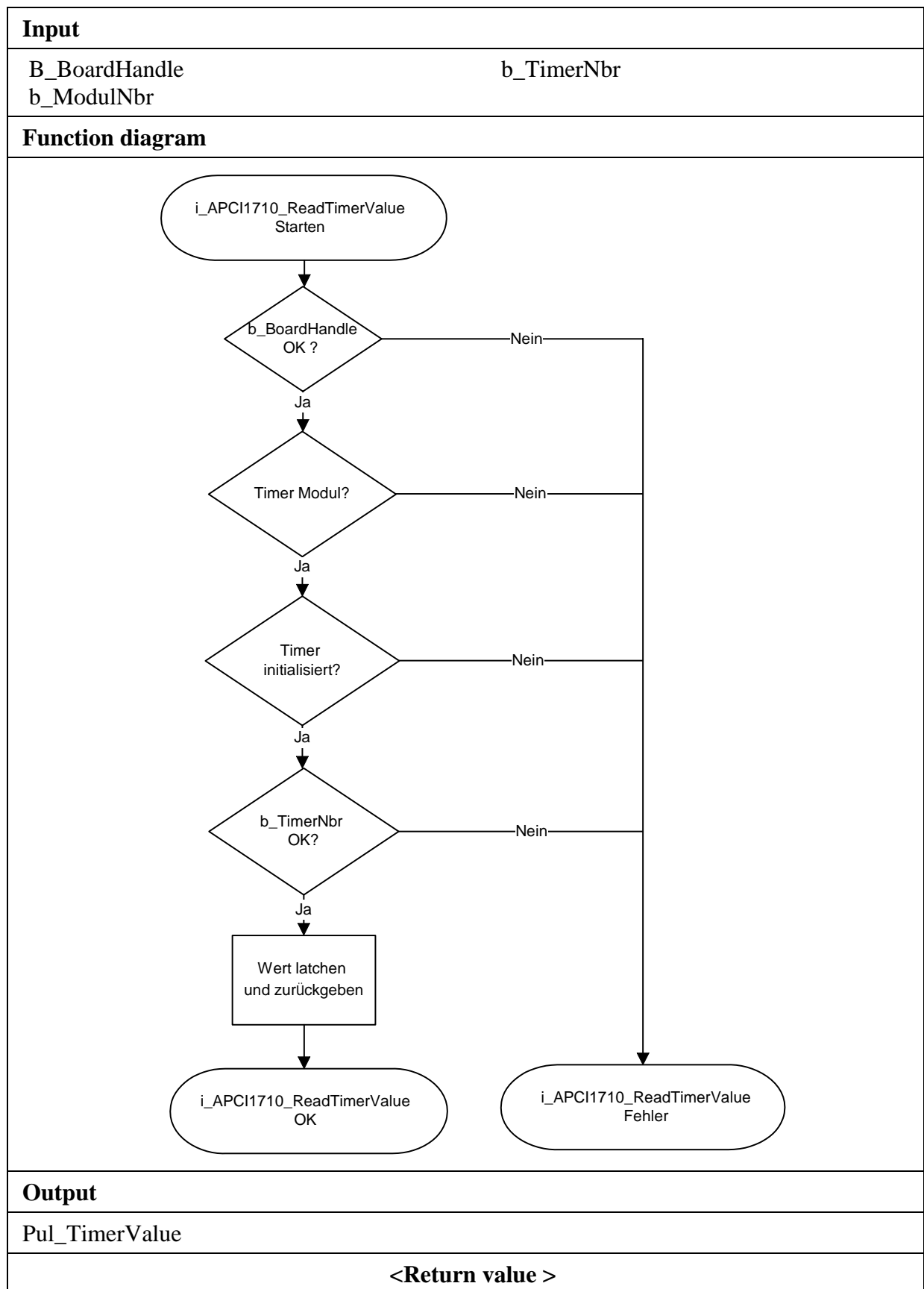
**Calling convention:**
ANSI C:

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

```
i_ReturnValue = i_APCI1710_ReadTimerValue
                (b_BoardHandle,
                 0,
                 0,
                 & ul_TimerValue);
```

**Return value:**

0: No error  
 -1: Handle parameter of the board is wrong.  
 -2: The selected module number is wrong.  
 -3: The selected timer is wrong.  
 -4: The selected module is no "Timer" module.  
 -5: Timer not initialised. See function "i\_APCI1710\_InitTimer"



## 5) i\_APCI1710\_ReadAllTimerValue (...)

### Syntax:

```
<Return-Wert> = i_APCI1710_ReadAllTimerValue
                                (BYTE      b_BoardHandle,
                                BYTE      b_ModulNbr,
                                PULONG    pul_TimerValueArray)
```

### Parameter:

#### -Input:

BYTE	b_BoardHandle	Handle of the <b>APCI-/CPCI-1710</b>
BYTE	b_ModulNbr	Number of the module to be configured

#### -Output:

PULONG	pul_TimerValueArray	Value sequence of the timer. Element 0 contains the value of timer 0 Element 1 contains the value of timer 1 Element 2 contains the value of timer 2
--------	---------------------	---

### Task:

Returns all timer values of the selected module (*b\_ModulNbr*).

### Calling convention:

ANSI C :

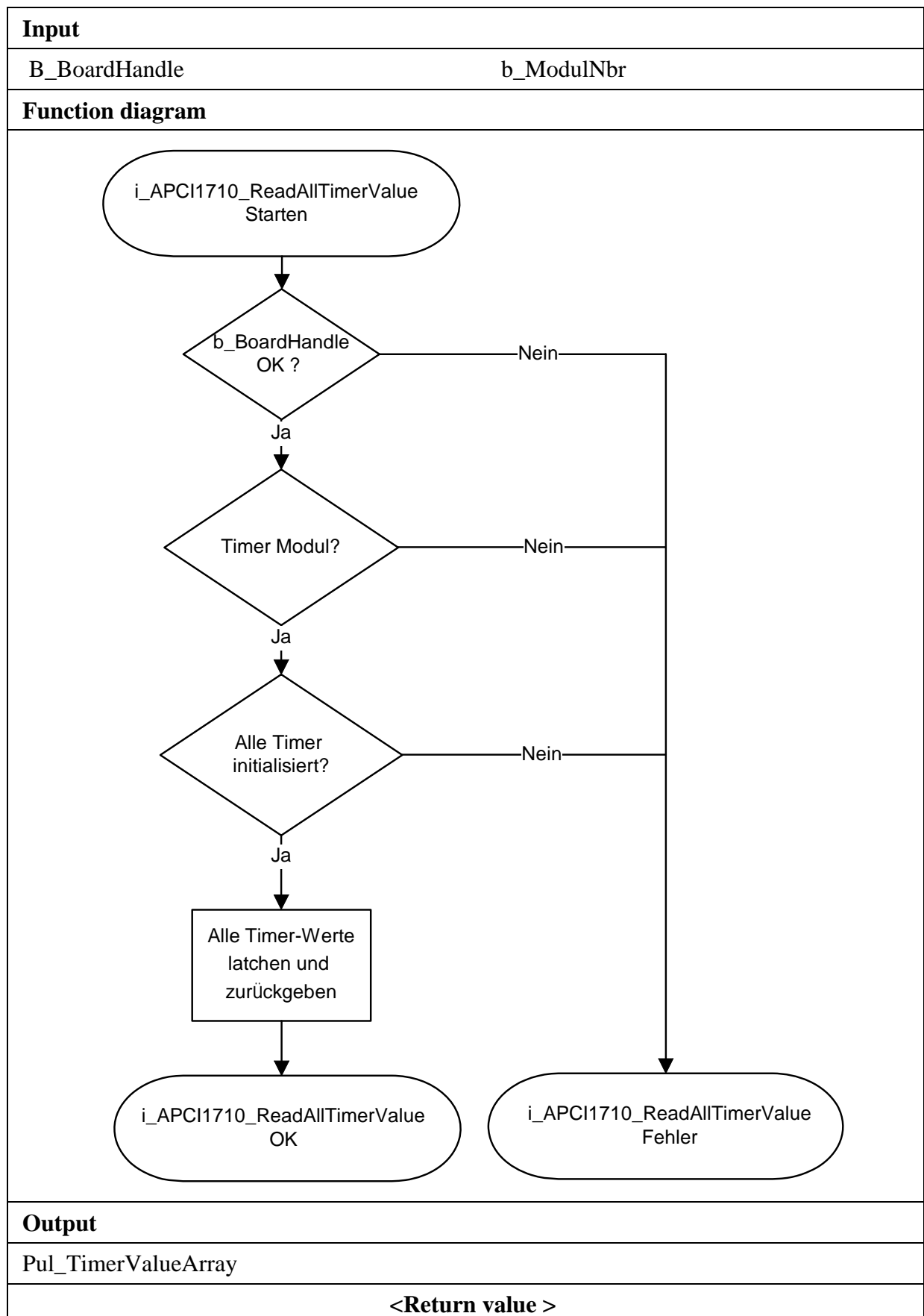
```
int          i_ReturnValue;
unsigned char b_BoardHandle;
unsigned long ul_TimerValueArray [3];
```

```
i_ReturnValue = i_APCI1710_ReadAllTimerValue
                (b_BoardHandle,
                 0,
                 ul_TimerValueArray);
```

### Return value:

0: No error

- 1: Handle parameter of the board is wrong.
- 2: The selected module number is wrong.
- 3: The selected timer is wrong.
- 4: The selected module is no "Timer" module.
- 5: Timer 0 not initialised. See function "i\_APCI1710\_InitTimer"
- 6: Timer 1 not initialised. See function "i\_APCI1710\_InitTimer"
- 7: Timer 2 not initialised. See function "i\_APCI1710\_InitTimer"



## 6) i\_APCI1710\_GetTimerOutputLevel (...)

### Syntax:

```
<Return-Wert> = i_APCI1710_GetTimerOutputLevel
                                (BYTE      b_BoardHandle,
                                BYTE      b_ModulNbr,
                                BYTE      b_TimerNbr,
                                PBYTE pb_OutputLevel)
```

### Parameter:

#### -Input:

BYTE	b_BoardHandle	Handle of the <b>APCI-/CPCI-1710</b>
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_TimerNbr	Number of the timer to be tested (0 to 2)

#### -Output:

PBYTE	pb_OutputLevel	Level of the output signal "0": The output is set to "Low" "1": The output is set to "High"
-------	----------------	---

### Task:

Returns the level of the output signal (*pb\_OutputLevel*) for the timer (*b\_TimerNbr*) of the selected module (*b\_ModulNbr*).

### Calling convention:

ANSI C :

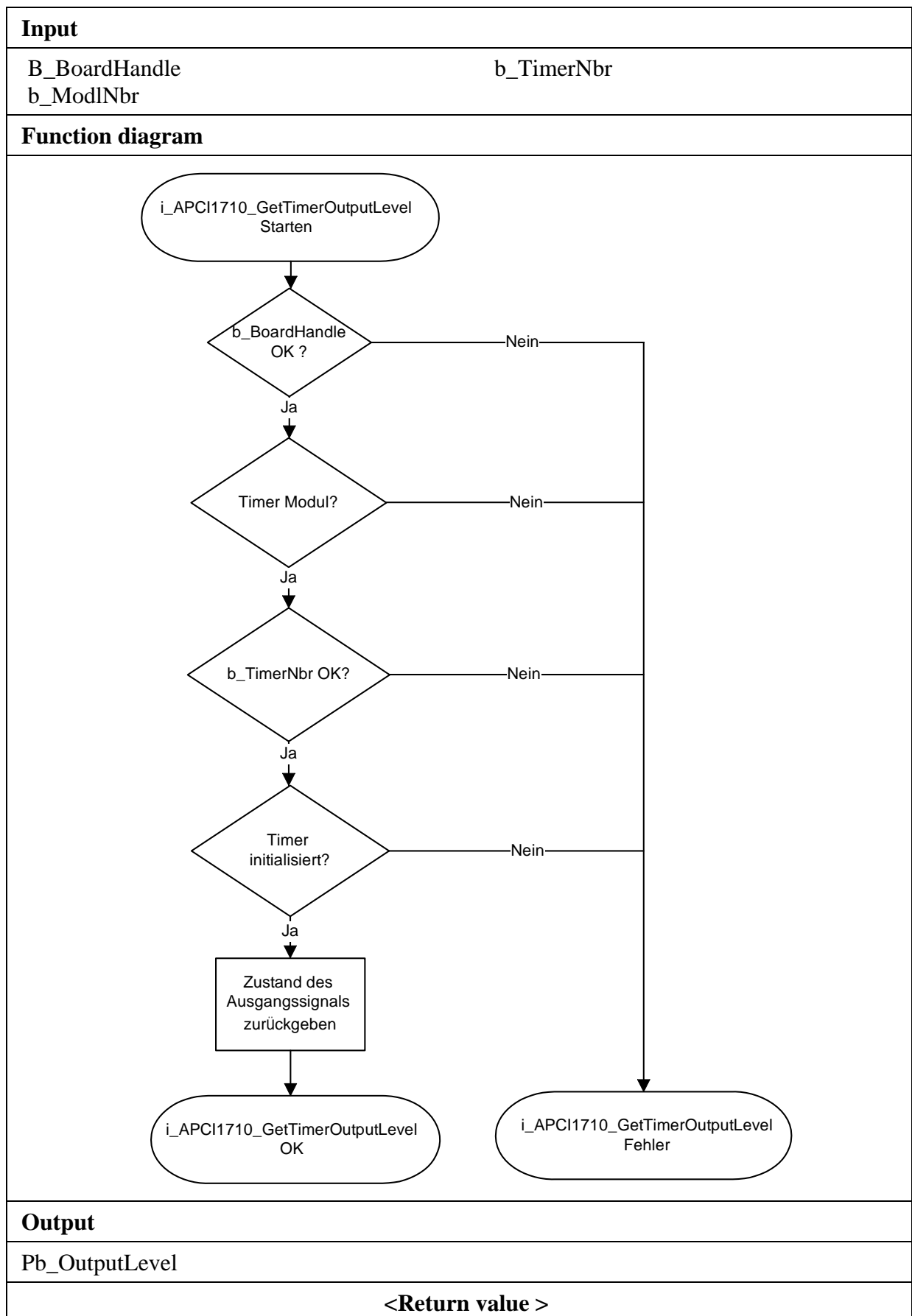
```
int          i_ReturnValue;
unsigned char b_BoardHandle;
unsigned char b_OutputLevel;
```

```
i_ReturnValue = i_APCI1710_GetTimerOutputLevel
                (b_BoardHandle,
                 0,
                 0
                 &b_OutputLevel);
```

### Return value:

0: No error  
 -1: Handle parameter of the board is wrong.  
 -2: The selected module number is wrong.  
 -3: The selected timer is wrong.  
 -4: The selected module is no "Timer" module.  
 -5: Timer not initialised. See function "i\_APCI1710\_InitTimer"





## 7) i\_APCI1710\_GetTimerProgressStatus (...)

### Syntax:

```
<Return-Wert> = i_APCI1710_GetTimerProgressStatus
                                (BYTE      b_BoardHandle,
                                BYTE      b_ModulNbr,
                                BYTE      b_TimerNbr,
                                PBYTE pb_TimerStatus)
```

### Parameter:

#### -Input:

BYTE	b_BoardHandle	Handle of the <b>APCI-/CPCI-1710</b>
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_TimerNbr	Number of the timer to be tested (0 to 2)

#### -Output:

PBYTE	pb_TimerStatus	Timer status "0": Timer is stopped "1": Timer is running
-------	----------------	--

### Task:

Returns the status (*pb\_TimerStatus*) of the timer (*b\_TimerNbr*) of the selected module (*b\_ModulNbr*).

### Calling convention:

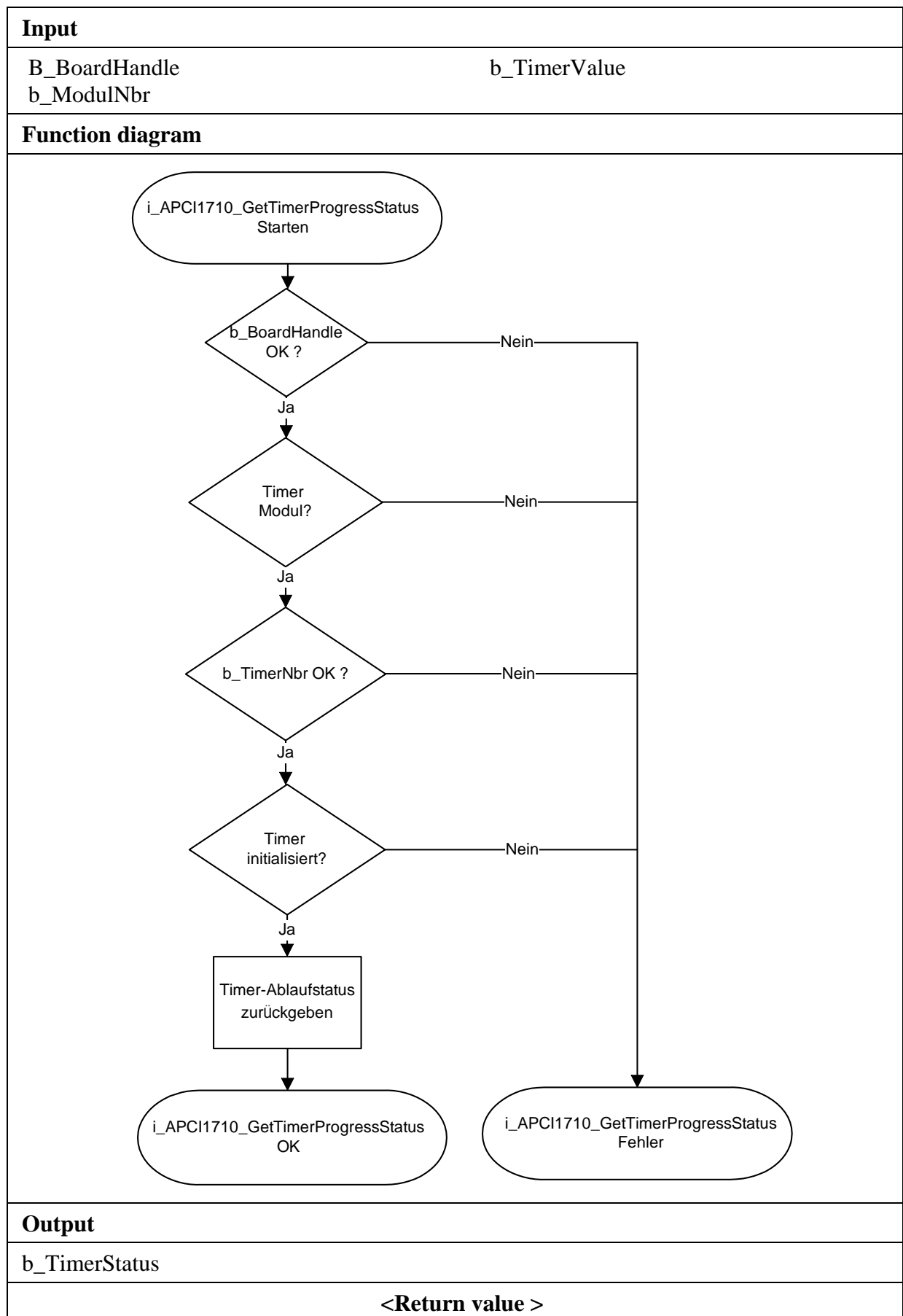
ANSI C :

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

```
i_ReturnValue = i_APCI1710_GetTimerProgress
                (b_BoardHandle,
                 0,
                 0
                 &b_TimerStatus);
```

### Return value:

0: No error  
 -1: Handle parameter of the board is wrong.  
 -2: The selected module number is wrong.  
 -3: The selected timer is wrong.  
 -4: The selected module is no "Timer" module.  
 -5: Timer not initialised. See function "i\_APCI1710\_InitTimer"



### 3.3.3 Writing on the timer

#### 8) i\_APCI1710\_WriteTimerValue (...)

##### Syntax:

```
<Return-Wert> = i_APCI1710_WriteTimerValue
                                (BYTE      b_BoardHandle
                                BYTE      b_ModulNbr,
                                BYTE      b_TimerNbr,
                                ULONG     ul_WriteValue)
```

##### Parameter:

###### -Input:

BYTE	b_BoardHandle	Handle of the <b>APCI-/CPCI-1710</b>
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_TimerNbr	Number of the timer to be tested (0 to 2)
ULONG	ul_WriteValue	Value to write

###### -Output:

There is no output.

##### Task:

Writes the value (ul\_WriteValue) into the timer (*b\_TimerNbr*) of the selected module (*b\_ModulNbr*). This function depends on the operation mode used.

##### Calling convention:

ANSI C :

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

```
i_ReturnValue = i_APCI1710_WriteTimerValue
                                (b_BoardHandle,
                                0,
                                0,
                                0xFF00FF00);
```

##### Return value:

0: No error

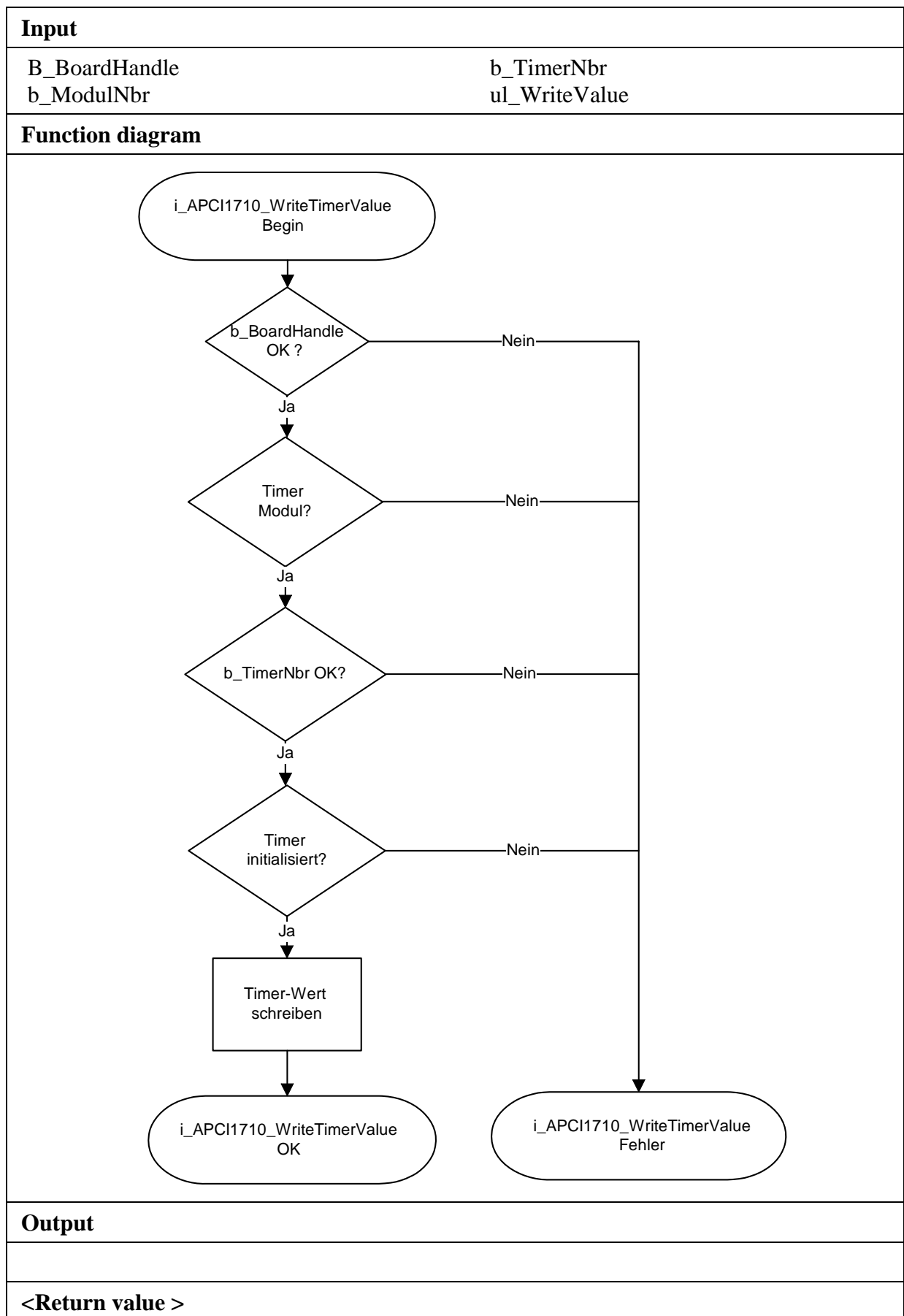
-1: Handle parameter of the board is wrong.

-2: The selected module number is wrong.

-3: The selected timer is wrong.

-4: The selected module is no "Timer" module.

-5: Timer not initialised. See function "i\_APCI1710\_InitTimer"



### 3.4 Functions in the kernel mode



#### IMPORTANT!

These functions are only available for the user interrupt routine under Windows NT and Windows 95 in the synchronous mode. See function "i\_APCI1710\_SetBoardIntRoutineWin32"

#### 3.4.1 Reading the timer

##### 1) i\_APCI1710\_KRNL\_ReadTimerValue (...)

##### Syntax:

```
<Return-Wert> = i_APCI1710_KRNL_ReadTimerValue
                                (UINT          ui_BaseAddress,
                                BYTE          b_ModulNbr,
                                BYTE          b_TimerNbr,
                                PULONG       pul_TimerValue)
```

##### Parameter:

##### -Input:

UINT	ui_BaseAddress	Base address of the <b>APCI-/CPCI-1710</b>
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_TimerNbr	Number of the timer to be read (0 to 2)

##### -Output:

PULONG	pul_TimerValue	Timer value
--------	----------------	-------------

##### Task:

Returns the value of the timer (*b\_TimerNbr*) of the selected module (*b\_ModulNbr*).

##### Calling convention:

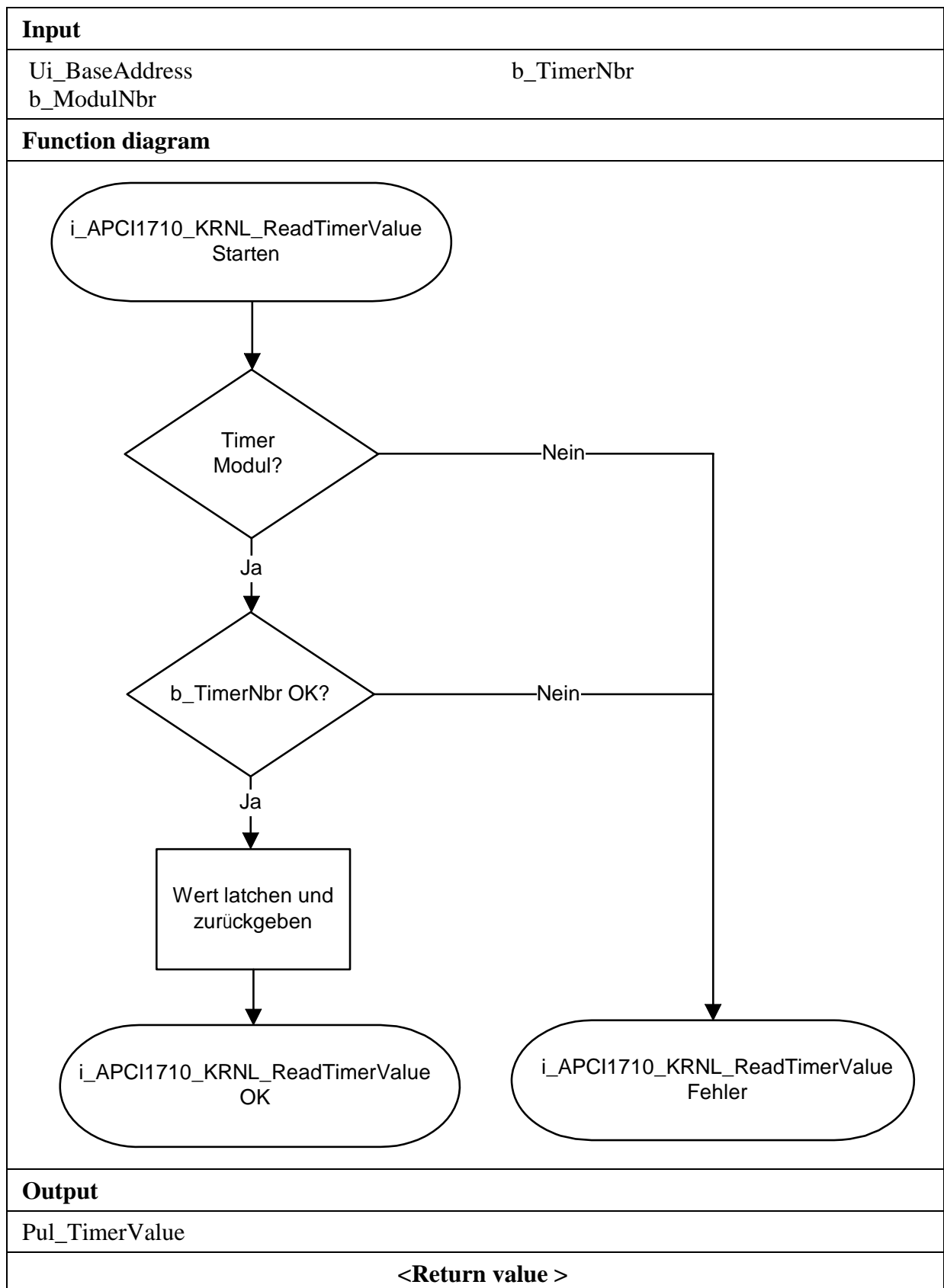
##### ANSI C :

```
int          i_ReturnValue;
unsigned int  ui_BaseAddress;
unsigned long ul_TimerValue;
```

```
i_ReturnValue = i_APCI1710_KRNL_ReadTimerValue
                (ui_BaseAddress,
                0,
                0,
                &ul_TimerValue);
```

##### Return value:

0: No error  
 -1: The selected module number is wrong.  
 -2: The selected timer is wrong.  
 -3: The selected module is no "Timer" module.



## 2) i\_APCI1710\_KRNL\_ReadAllTimerValue (...)

### Syntax:

```
<Return-Wert> = i_APCI1710_KRNL_ReadAllTimerValue
                                (UINT          ui_BaseAddress,
                                BYTE           b_ModulNbr,
                                PULONG        pul_TimerValueArray)
```

### Parameter:

#### -Input:

UINT	ui_BaseAddress	Base address of the <b>APCI-/CPCI-1710</b>
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

#### -Output:

PULONG	pul_TimerValueArray	Value sequence of the timer. Element 0 contains the value of timer 0 Element 1 contains the value of timer 1 Element 2 contains the value of timer 2
--------	---------------------	---

### Task:

Returns all timer values of the selected module (*b\_ModulNbr*).

### Calling convention:

ANSI C :

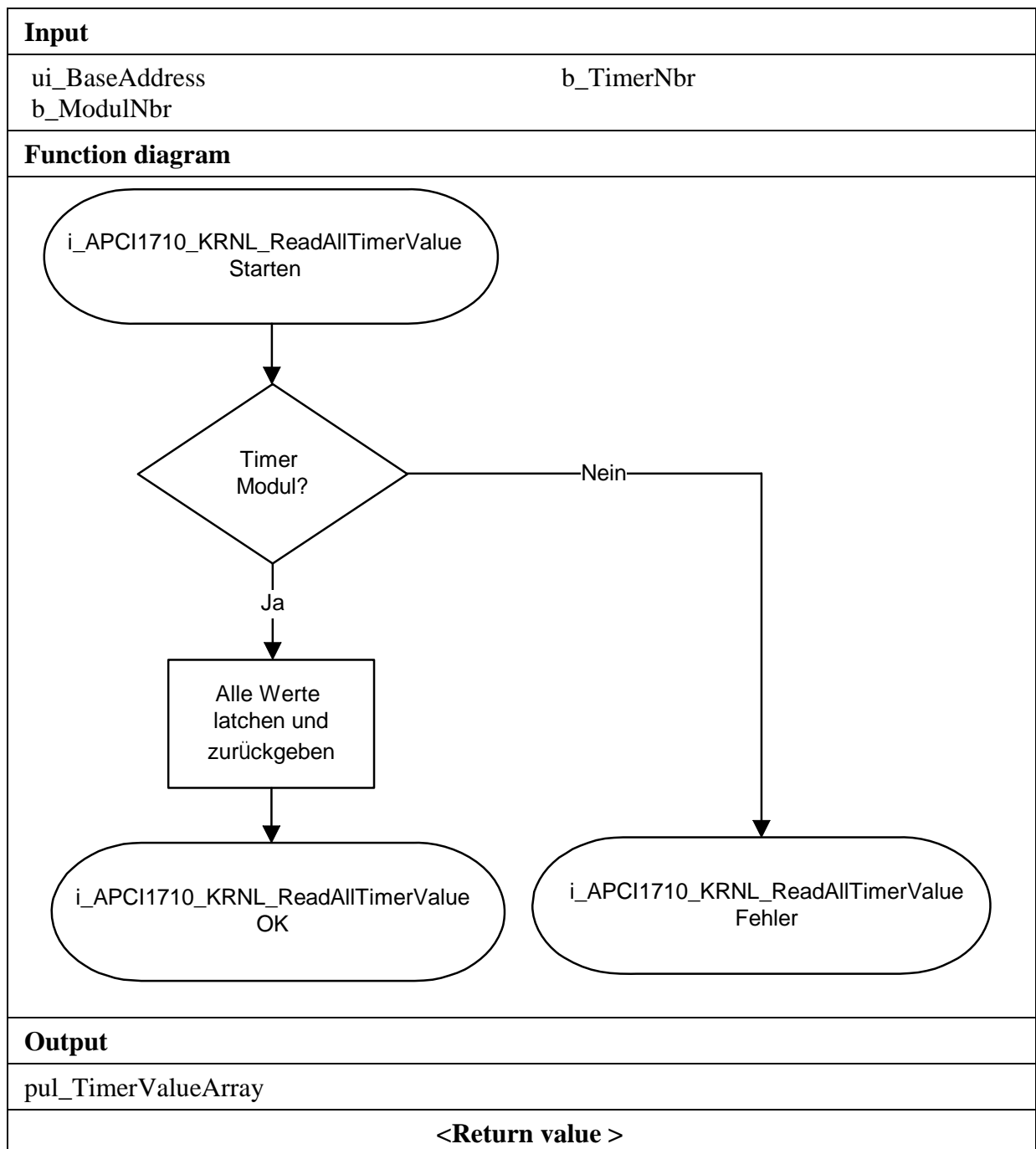
```
int          i_ReturnValue;
unsigned int  ui_BaseAddress;
unsigned long ul_TimerValueArray [3];
```

```
i_ReturnValue = i_APCI1710_KRNL_ReadAllTimerValue
                (ui_BaseAddress,
                 0,
                 ul_TimerValueArray);
```

### Return value:

0: No error  
-1: The selected module number is wrong  
-2: The selected module is no "Timer" module.





### 3.4.2 Writing on the timer

#### 3) i\_APCI1710\_KRNL\_WriteTimerValue (...)

**Syntax:**

```
<Return-Wert> = i_APCI1710_KRNL_WriteTimerValue
                                (UINT      ui_BaseAddress,
                                BYTE       b_ModulNbr,
                                BYTE       b_TimerNbr,
                                ULONG      ul_WriteValue)
```

**Parameter:**
**-Input:**

UINT	ui_BaseAddress	Base address of the <b>APCI-/CPCI-1710</b>
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)
BYTE	b_TimerNbr	Number of the timer to be tested (0 to 2)
ULONG	ul_WriteValue	Value to write

**-Output:**

There is no output.

**Task:**

Writes the value (ul\_WriteValue) into the timer (*b\_TimerNbr*) of the selected module (*b\_ModulNbr*). This function depends on the operation mode.

**Calling convention:**

ANSI C :

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

```
i_ReturnValue = i_APCI1710_WriteTimerValue
                (ui_BaseAddress,
                0,
                0,
                0xFF00FF00);
```

**Return value:**

0: No error  
 -1: The selected module number is wrong.  
 -2: The selected timer is wrong.  
 -3: The selected module is no „Timer“ module.

